Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques

MATCH
Human Capital and Mobility CHRX-CT94-0452

Draft for the Summer School
Jaca (Spain), 3-11 September, 1998

MATCH (Modelling and Analysis of Time Constrained and Hierarchical Systems) is a European Union Human Capital and Mobility initiative. Partners of this initiative that have contributed to this book are from the Universities of Genova (G. Chiola), Paris (B. Baynat, Y. Dallery, C. Dutheillet, S. Haddad), Torino (G. Balbo, L. Capra, S. Donatelli, G. Franceschinis, M. Ribaudo, M. Sereno), Vienna (A. Ferscha), and Zaragoza (J. Campos, J. M. Colom, M. Silva, E. Teruel). Contributions have been also written by B. Haverkort (Univ. Aachen), H. Hermanns and U. Herzog (Univ. Erlangen), J. Hillston (Univ. Edinburgh), and P. Moreaux (Univ. Reims).

Preface

Discrete Event Dynamic Systems (DEDS) are based on a view of dynamic systems where discrete states and events play a fundamental role. DEDS are becoming of increasing interest as the importance of automation and control grows in modern technology. Application domains like flexible manufacturing systems, transport systems, distributed systems, and telecommunication networks extensively use concepts and techniques from DEDS. Design optimization, scheduling (performance control), monitoring, and implementation (possibly fault-tolerant) are among the critical issues that require the use of modelling and of qualitative and quantitative analysis techniques. Formal methods for the specification, verification, and performance evaluation of DEDS are thus becoming increasingly important as the size and complexity of systems grow.

MATCH (Modelling and Analysis of Time Constrained and Hierarchical Systems) is a Human Capital and Mobility initiative, sponsored by the European Union. Its main goal is to foster the progress of qualitative and quantitative analysis techniques useful for the evaluation of parallel and distributed systems. A concrete result of this collaborative effort is the production of two books in which a comprehensive discussion of a Petri Net-based framework to model and analyze the correctness and the performance of complex distributed systems is presented.

This book collects the Performance Evaluation contributions of the project. We consider the impact that the interplay of concurrency and cooperation among agents has at the global system level, taking into account such phenomena as queueing, service, routing, and synchronization. We start from the hypothesis that a precise description of the semantics of these phenomena is mandatory for their analysis to be trustworthy and accurate. Petri Nets are proposed in this book as the basis for the introduction of time extensions that allow an accurate description of complex applications, and that provide the possibility of exploiting the structure of the models for developing a better understanding of the behaviour of dynamic systems and for devising effective analysis techniques.

From the abstraction of the details of many real situations, probability distributions for the duration of activities are used and the Stochastic Petri Net formalism is defined. Other formalisms, like Queueing Networks and Stochastic Process Algebras, are also addressed in the book from the point of view of their relations with Stochastic Petri Nets.

The book is divided into seven parts that focus on the most relevant aspects of the definition of the formalism. The first part addresses the most important aspects of DEDS discussing how they can be captured and represented by the formalisms that are most commonly used for the verification and performance evaluation of complex systems.

The second part discusses the fundamental properties of Petri Nets and develops the basic results for the qualitative analysis of these models as well as the operational definitions of evaluation indices.

Part three discusses the implications of using stochastic variables for the representation of the duration of the activities represented in the models. The construction of the probabilistic model associated with Stochastic Petri Nets is discussed both in simple cases (that yield continuous time Markov chains) and in more complex ones (that require the analysis of more complex stochastic processes). Special attention is devoted to particular techniques that can be used for the analysis of unbounded models.

The fourth part addresses the problem of the exponential growth of the state space of the probabilistic models associated with all types of Stochastic Petri Nets. In essence, net-driven Markov Chain generations try to keep the above phenomenon under control. The techniques illustrated in this part include: the possibility of developing more compact models by explicitly addressing their symmetries, the exploitation of compositional techniques both at the model construction and solution levels, and the use of decomposition as a vehicle for the reduction of the solution of a complex problem to that of several smaller ones.

The fifth part discusses the possibility of completely avoiding the construction of the state space of the associated probabilistic model. Thus net-driven computational approaches are considered. We do so either by limiting the goal of the analysis to the computation of performance bounds, or by identifying a subclass of models for which a product-form solution can be computed in terms of the components of the representation (i.e., places and transitions) so that computationally efficient algorithms may be utilized.

Part six discusses the analysis of systems by means of decomposition methods that trade off the accuracy of the solution with the computational cost, possibly by numerical evaluation or by simulation. Special types of models are addressed in this part for which ad-hoc approximation techniques have been developed.

Finally, part seven discusses the possibility offered by discrete event simulation for analyzing models of real systems whose complexity cannot be managed by any analytical or numerical technique. Parallel and distributed simulations are compared in this part to emphasize their potential for speeding up the evaluation of extremely large models.

The effort of the authors of the chapters of this book to present the material in a coordinated manner and with the use of uniform notation is one of the important achievements of this project. This volume is thus adequate as a reference for researchers already active in the field as well as a textbook for students that face this subject for the first time and that need to be exposed to the different aspects of the area.

Special thanks go to the authors S. Haddad, B. Haverkort, H. Hermanns, U. Herzog, J. Hillston, and P. Moreaux who, without being members of the MATCH initiative, accepted the invitation to join this effort with enthusiasm. Their contributions cover many important topics that otherwise would have been omitted.

G. Balbo, University of Torino

M. Silva, University of Zaragoza

Contents

Volume 1

Volume 2

Part I

Formalisms

Chapter 1

Queueing Networks

In this chapter, we present one of the basic formalism for modeling discrete event systems, namely *Queueing Networks*. Queueing networks have been widely used in the performance evaluation of such discrete event systems as computer systems, communication networks, manufacturing systems [2, 4, 8, 9, 5]. Queueing networks are often viewed as modeling and analysis techniques. This is because, the modeling and the analysis aspects of queueing networks have usually been dealt with together. The analysis part has resulted in the development of many different analytical methods, which are of wide interest and that have also to some extent lead the way to the development of similar methods in the context of Petri nets, as will appear later in this book. In this chapter however, it is only the modeling aspect of queueing networks that we are interested in. Thus, the rest of the chapter is devoted to the presentation of the queueing network formalism as a modeling tool for discrete event systems.

We have chosen a somewhat informal presentation of queueing network models, starting from simple models and introducing more and more complex models, step by step. The main reason is that we want the reader to first understand the basic concepts behind queueing network models before being faced with complex models. Another reason is that this presentation is consistent with the way classes of queueing networks have been introduced in the literature. As a matter of fact, in the first part, we will introduce what we refer to as *Basic Queueing Networks*, which is the basic formalism that has been widely used in the literature. As it will be seen, this formalism is however not powerful enough to model many real situations, e.g. synchronization phenomena. We will then present an extension of the classical queueing network formalism. This class of networks will be generally referred to as *Extended Queueing Networks.*

1.1 Basic Queueing Network Formalism

In this section, we introduce the basic queueing network formalism. We start by considering successively: simple queues, multiclass queues, networks of queues,

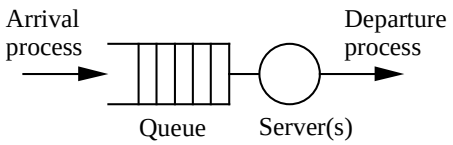


Figure 1.1: A simple queueing system

multiclass queueing networks, queueing networks with finite capacity and state dependent routing. Throughout this section, we also introduce some basic performance measures of queueing networks to emphasize what is the final goal of performance evaluation.

1.1.1 Simple Queues

A simple queueing system or *single queue* consists of a *queue*, also called *buffer*, and one or more resources called *servers*. Entities called *customers* or *jobs* generated by an *external arrival process* join the queue to receive a service at one of the servers. Upon service completion of a customer, this customer leaves the system. To completely characterize the behavior of this system, one has to define the number of servers, the service discipline, the capacity of the queue, the arrival process and the service process. Figure 1.1 shows the graphical representation of a single queue.

Number of servers. The number of servers, denoted by C, may be one (*single-server* case), a finite number greater than one (*multiple-server* case), or infinity (*infinite-server* or *delay-server* case). This last situation occurs when there is always a server ready to serve any customer arriving into the system, that is there is an unlimited number of service resources. In this case, there is no queueing in the system. It is usually assumed that all servers are identical, however more general situations, e.g. a two-server system with one fast server and one slow server, can also be considered.

Service discipline. The service discipline, or scheduling rule, defines the policy by which the servers are allocated to the various customers present in the system. The basic service discipline usually considered is *first-in-first-out* (FIFO), also known as *first-come-first-served* (FCFS). This means that the customers are processed in the order they arrive in the queue. (Note that, except for single-server systems, it does not imply that customers leave the system in their arrival order.) Other queueing discipline includes: *last-in-first-out* (LIFO), *round robin* (RR), and *processor sharing* (PS) [7].

Queue capacity. The capacity of the queue, denoted by K, is the total number of customers that can be simultaneously present in the system (either waiting or receiving service). it is important to differentiate between the following two cases: *infinite capacity* queues and *finite capacity* queues. In the case of infinite

capacity queues, any customer arriving to the system can always be accepted since there is an infinite waiting room. In the case of finite capacity queues on the other hand, one has to specify the behavior of the system when the queue is full. The classical assumption is that customers continue arriving to the system and that whenever a customer arrives and finds the queue full, it is lost. Alternatively, one could specify that the arrival process is stopped when the queue is full and resumes as soon as space becomes available.

Arrival process. The arrival process characterizes the way customers arrive to the system. In general, the arrival process can be represented by a sequence of instants $a_1, a_2, ..., a_n, ...$, where a_n is the time at which the n-th customer arrives in the system. Alternatively, the arrival process can be described by the sequence of *interarrival times*, $i_1, i_2, ..., i_n, ...$, where i_n is the time between the arrival of the n-th and $(n+1)$-th customers, i.e.: $i_n = a_{n+1} - a_n$. The i_n are random variables and, in the most general setting, they may have different distributions and be correlated. However, a special case of interest is the case where these random variables are *independently and identically distributed* (i.i.d. random variables). In this case, the arrival process is completely characterized by the common distribution of these random variables, which is referred to as the *interarrival time distribution*.

Service process. The service process characterizes the requirements of the customers in terms of service. In general, the service process can be represented by a sequence of times $s_1, s_2, ..., s_n, ...$, where s_n is the time required to serve the n-th customer, referred to as the *service time*. The s_n are random variables and in the most general setting, they may have different distributions and be correlated. However, a simple case of wide interest is the case where these random variables are independent and identically distributed (i.i.d. random variables), and also independent of the sequence of interarrival times. In this case, the service process is completely characterized by the common distribution of these random variables, which is referred to as the *service time distribution*.

Remark In many situations, arrivals and services can occur at any time, in which case the model is referred to as a *continuous time* model, and the interarrival and service time distributions are continuous time distributions. There are however situations for which arrivals and services can not occur at arbitrary times, but can only occur at specific instants, which are multiples of a given quantity called the *time unit*. In this case, the model is referred to as a *discrete time* model, and the interarrival and service time distributions are discrete time distributions.

1.1.2 Modeling and Performance Evaluation

The goal of performance evaluation is to provide quantitative information on the behavior of a discrete event system. For the purpose of illustration, suppose the discrete event system of interest is the ticket office at a train station. There

is a fixed number C of employees delivering tickets to the customers arriving to take the train. Suppose that the customers form a single waiting line and that, whenever an employee becomes available, he/she addresses the request of the first customer in the waiting line. Customers arrive according to some arrival process. Service times are variable because the service requirements vary from one customer to the next.

The main performance parameters of interest for such a system are:

- *average response time*: average time it takes for a customer to get his/her ticket;
- *average waiting time*: average time a customer waits before getting access to one of the employees;
- *average queue length*: average number of customers present in the ticket office;
- *throughput*: average number of customers leaving the system per unit of time;
- *utilization* of the servers: average percentage of time each employee is busy serving customers.

Beside their average values, it may be of interest to know about the variance of the above quantities, or more generally about their distributions. For instance, it may be important to know the variance of the waiting time of a customer, in addition to knowing its average value.

One possible way to determine these quantities is just to record the behavior of the ticket office during a long period of time and to derive the quantities of interest from this observation, in particular the above ones. The performance parameters then derived are referred to as *time averages*, which means that they have been obtained by averaging samples over a period of time.

An alternative way is to construct a *stochastic model* of this real system. The ticket office can actually be modeled as a simple queue with infinite capacity (assuming that the office is large enough to accommodate all customers arriving) and with C servers corresponding to the employees. To complete the characterization of the model, the arrival and service processes have to be specified. Assuming independence assumptions for both the arrival and service processes as described above (which in this case is realistic), the characterization reduces to the distributions of interarrival and service times. If these distributions are given, one can then "run" the model for a given period of time. This means to randomly generate arrivals according to the interarrival time distribution and service times according to the service time distribution. By doing that, we are constructing what is called a *sample path* of the model. Note that this is exaclty what discrete event simulation techniques do to calculate performance parameters; refer to Part VII for more on these techniques. The performance parameters derived using this approach are again referred to time averages,

because as in the case of the monitoring of the real system, they have been obtained by averaging samples during a finite period of time.

Remark It may appear strange at first glance to represent interarrival and service times as random variables. Indeed, service times are related to the requirement of each particular customer. For instance a customer just asking for some simple information will have a "short service time", while a customer buying a ticket for a foreign destination involving different trains, will have a "long service time". However, what is important when dealing with performance issues is not to worry about the specific characteristics and behavior of a particular customer, but rather to capture the global behavior of the system. For that reason, what is important is to represent the *typical behavior* of a customer and not the specific behavior of any particular one. Doing that means acting as an outside observer standing at the back of the ticket office and looking at what is going on (as opposed to standing next to the counter and trying to understand why a particular customer has spent exactly so much time "in service"). To this outside observer, the interarrival and service times will indeed appear as if they were random variables. Of course, the following issue is very important and arises in any stochastic modeling of a real system: in order for the performance analysis to be meaningful (i.e. the performances calculated based on the model are in agreement with those of the real system), the distributions of the interarrival and service times have to be accurately characterized. This aspect of the modeling activity is usually referred to as *workload* or *traffic* characterization. It is a very important issue in performance evaluation.

Remark When addressing performance evaluation issues, one may be interested either in the *short-term* or *long-term* behavior, which are referred to as *transient analysis* and *steady-state analysis*. The transient analysis answers questions such as: how many customers (in average or in distribution) will be served during the next hour. The steady-state analysis answers questions such as: during a very long period of time, what is the average percentage of time an employee is busy? The transient analysis depends on the initial state of the system whereas, in general, the steady-state analysis does not.

1.1.3 Notion of Classes of Customers and Other Features

In the above presentation of simple queues, there was not explicit distinctions among customers. In terms of the model, this means that all customers were assumed to be *stochastically equivalent*. There may however be situations for which there is a need to distinguish between different types of customers. This is the notion of *classes of customers* in queueing systems. Introducing the notion of classes of customers allows us to distinguish customers among each other. As far as simple queues are concerned, this distinction may occur at four levels:

- arrival process: customers of different classes may have different arrival processes, i.e., different distributions of interarrival times;

- service process: customers of different classes may have different service processes, i.e., different distributions of service times;

- service discipline: the allocation of the server may be based on the classes, e.g., customers of class 1 may have (preemptive or non-preemptive) priority over customers of class 2.

- queue capacity: the total storage area may be divided into individual storage areas for each class of customers, in which case each class has its own finite capacity buffer.

Of course, these different reasons for having different classes of customers may appear jointly.

In a multiclass queueing system, one can define both *overall performance parameters* and *class-dependent performance* parameters, in a similar way as for single-class queues. For instance, with respect to the average queue length, one may be interested in calculating the average number of customers in the queue (regardless of their classes) and/or the average number of customers of each class in the queue.

There may actually be other features that can be incorporated into the formalism of simple queues. Below, we describe some of them.

Batch arrivals and services. In some real systems, customers do not arrive one at a time, but in batches and/or customers are not served one at a time but in batches. This feature can be incorporated in queueing models. Consider for instance the arrival process. The usual way to characterize a batch arrival process is to characterize both the arrival instants of the batches and the size of the batches. The characterization of the arrival instants of the batches can be done in the same way as that of the arrival instants of individual customers. For instance, if successive interarrival times are assumed to be i.i.d. random variable, the characterization reduces to the specification of the interarrival time distribution. As for the size of the batches, it is usually assumed that it is a random variable. The sizes of successive batches may be correlated. However, it is generally assumed that the batch sizes are i.i.d. random variables, in which case the batch size is characterized by a discrete distribution. (Note that this includes the case where the batch size is constant.)

Push-out mechanisms. In the case of queues with several classes of customers and in which a common finite buffer is shared by the different classes of customers, the following behavior can be of interest for modeling real systems: there is a so-called *spatial priority* among customers so that if a customer of high priority finds the queue full upon arrival, it pushes out of the queue one of the customers of lower priority, if any (otherwise it is lost). This *push-out* mechanism can be specified by ranking the different classes of customers according to their priority with respect to the space allocation of the finite buffer.

1.1.4 Queueing Networks

A natural generalization of simple queues is to consider networks of queues, referred to as *queueing networks*. A queueing network consists of a set of M queues also referred to as *stations*. Each station consists of a queue and a set of servers (either single-server, multiple-server, or infinite-server). The usual assumption is that all queues have infinite capacity. The characterization of the service disciplines and the service processes at each station is the same as in the case of single queues. There are two basic types of queueing networks, namely *open queueing networks* and *closed queueing networks*.

Open queueing networks. In an open queueing network, customers arrive from outside according to a given arrival process and visit a set of stations after which they leave the system. The characterization of the arrival process is the same as in the case of single queues. The additional feature that needs to be specified is the *routing* of the customers, i.e., how customers travel through the network. More precisely, what has to be specified is: 1) which station is joined by a customer arriving to the system; and 2) upon completion of service at a station, whether the customer leaves the system or joins another station, and which one. The routing is usually characterized in a probabilistic way. That is, the station joined by a new customer is chosen according to a given distribution. Similarly, the station that a customer joins after service completion at a station is chosen according to a given distribution. These distributions can in general be correlated, e.g. the routing of a customer coming out of a station may depend on the routing of previous customers at this station and/or on its past routing through the network. However, a usual assumption is that routings are uncorrelated. In this case, the routing can be characterized by the following probabilities: $r_{0,j}$ is the probability that upon arrival to the system, a customer joins queue j, $j = 1, ..., M$; $r_{i,j}$ is the probability that after completion at station i, $i = 1, ..., M$, a customer joins queue j, $j = 0, 1, ..., M$, where 0 represents the outside world, i.e., $r_{i,0}$ is the probability of leaving the system after completion at station i. The matrix $\mathbf{R}$, whose components are the routing probabilities $r_{i,j}$, $i = 0, 1..., M$ and $j = 0, 1, ..., M$, is referred to as the *routing matrix* of the open queueing network. An example of an open queueing network consisting of $M = 4$ stations is shown in Figure 1.2. Note that a customer may visit a station more than once.

The routing in queueing networks gives rise to two mechanisms called *merging* and *splitting* of customer flows, also called *superposition* and *decomposition* of flows, respectively. Merging occurs when several stations feed a single station, say i. The input flow of station i is then the superposition of the incoming flows from the feeding stations. Splitting occurs when a single station, say j, feeds several stations. The outgoing flows to the different stations are then the decomposition of the output flow of station j. This is illustrated in Figure 1.3.

A topology of particular interest is that of *tandem* queueing networks. In a tandem queueing network, all customers visit station 1, then station 2, and so on until station M, after which they leave the system.

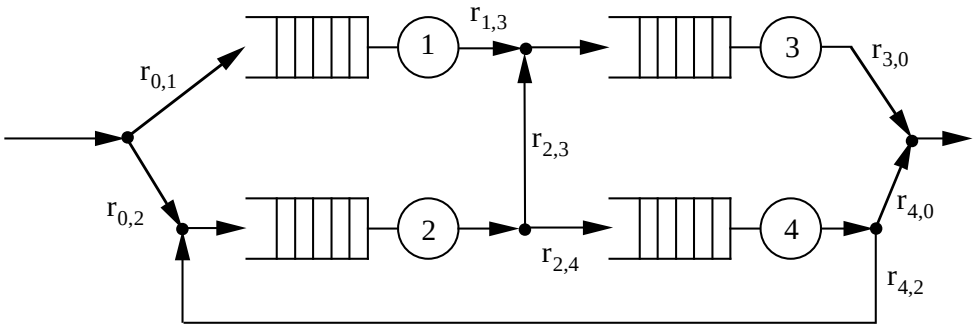


Figure 1.2: Illustration of an open queueing network

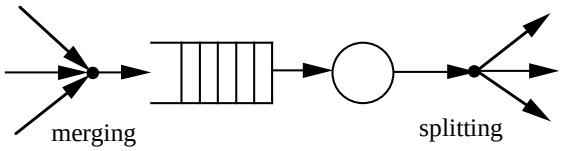


Figure 1.3: Illustration of the merging and splitting mechanisms

In a queueing network, there are two kinds of performance parameters that are of interest, namely *local and global performance parameters*. They are defined below.

Local performance parameters:

- average response time at a station: average time a customer spends at the station (per visit);
- average waiting time at a station: average time a customer waits before getting access to one of the servers (per visit);
- average queue length at a station: average number of customers present at the station;
- throughput of a station: average number of customers leaving the station per unit of time;
- utilization of the servers at a station: average percentage of time each server is busy serving customers.

Global performance parameters:

- average system time: average time a customer spends in the system;
- average system population: average number of customers present in the system;

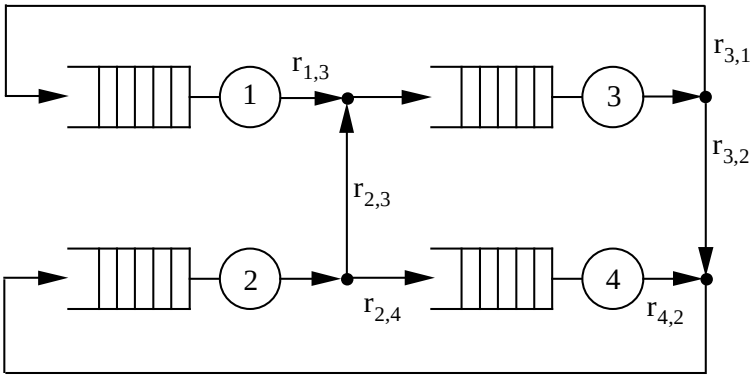


Figure 1.4: Illustration of a closed queueing network

- system throughput: average number of customers leaving the system per unit of time.

Again, as for simple queues, one may be interested not only in knowing the average of these quantities, but more generally their distributions, and in particular their variances.

Closed queueing networks. In a closed queueing network, there is a fixed number of customers, referred to as the population of the network and denoted by N, that travel through the network. Again, the additional feature that needs to be specified is the *routing* of the customers, and the above discussion applies to closed queueing networks as well. If the independence assumption is made, the routing can be characterized by the following probabilities: $r_{i,j}$ is the probability that after completion at station i, $i = 1, ..., M$, a customer joins queue j, $j = 1, ..., M$. The matrix $\mathbf{R}$, whose components are the routing probabilities $r_{i,j}$, $i, j = 1..., M$, is referred to as the *routing matrix* of the closed queueing network. An example of a closed queueing network consisting of $M = 4$ stations is shown in Figure 1.4.

For closed queueing networks, the same local performance parameters can be defined. If of interest, global performance parameters must be carefully defined since in a closed queueing network, customers never enter nor leave the system.

Multiclass queueing networks. As in the case of simple queues, there may be situations for which the notion of classes of customers is required to specify the behavior of the queueing network model. This may again be due to a need to distinguish customers with respect to the arrival process, the service processes and the service disciplines at the different queues. In addition, in the case of network of queues, the distinction may also be with respect to the routing of the customers, i.e, different classes of customers may have different probabilistic routings through the network. This leads to the definition of different types of multiclass queueing networks:

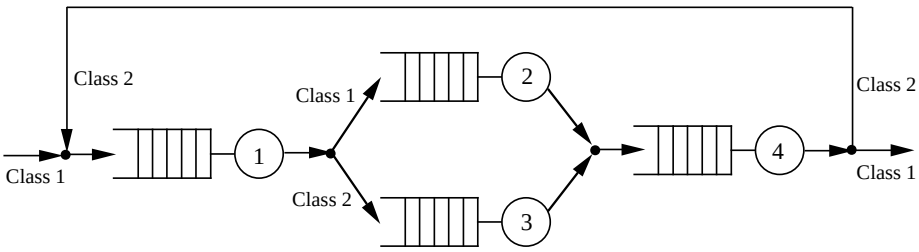


Figure 1.5: Illustration of a multiclass queueing network

- open multiclass queueing networks, in which all the classes behave as open classes, i.e., customers arrive from outside, travel through the stations, and finally leave the network;
- closed multiclass queueing networks, in which all the classes behave as closed classes, i.e., there is a constant number of customers of each class circulating in the network;
- mixed multiclass queueing networks, in which some of the classes behave as open classes, while others behave as closed classes.

Note that mixed networks is the most general class since it includes both open and closed networks as special cases. An example of a multiclass queueing network consisting of $M = 4$ stations and two classes of customers is shown in Figure 1.5. Class 1 is an open class, while class 2 is a closed class. Class-1 customers visit successively stations 1, 2 and 4, while class-2 customers visit stations 1, 3 and 4, repeatedly.

In a multiclass queueing network, one can combine the notion of overall vs class-dependent performance parameters with that of local vs global performance parameters. For instance, regarding the average queue lengths in an open multiclass queueing network, one may be interested in calculating the following measures of performance:

- average number of customers present at each station;
- average number of customers of each class present at each station;
- average number of customers present in the system;
- average number of customers of each class present in the system.

Finally in some situations, it may be of interest to allow customers to change class while traveling through the network, e.g. a customer leaving a station as a class-1 customer may switch to class-2 (with some probability) and then choose its destination according to the routing of class-2 customers. Such networks are referred to as multiclass queueing networks with *class switching* [3].

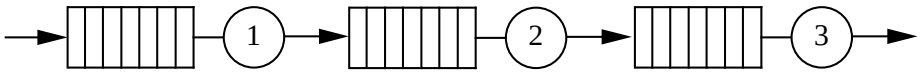


Figure 1.6: Illustration of an open queueing network with finite capacity

1.1.5 Queueing Networks with Finite Capacity and State Dependent Routing

So far, we have only considered queueing networks with infinite capacity queues. However, it is often useful to model systems having finite buffers, for which queueing networks with finite capacity have been introduced. Consider first single-class open queueing networks, in which one or more queues have finite capacity. Let K_i denote the capacity of queue i, $i = 1, ..., M$. It represents the maximal number of customers that can be simultaneously present at station i, including those in service. For such networks, one needs to further specify the behavior of the network when one or more queues are full. For simplicity, consider the case of tandem queueing networks in which each station has a single-server. An example of an open queueing network with finite capacity, consisting of $M = 3$ stations in tandem is shown in Figure 1.6.

Suppose for instance that during some period of time, queue i is full. In this situation, the behavior of the system can be specified in the following different ways:

- *Blocking-before-service* mechanism: server $i - 1$ is not allowed to start serving a new customer while queue i is full. Thus, the server is blocked and will resume its activity as soon as a space becomes available in queue i.
- *Blocking-after-service* mechanism: server $i-1$ can process a customer even though queue i is full. However, if queue i is still full upon service completion, the customer cannot be transferred and as a result, the server is blocked. As soon as a space becomes available, the customer is transferred into queue i, and the server immediately resumes its activity.
- *Block-and-recirculate* mechanism: server $i-1$ can process a customer even though queue i is full. However, if queue i is still full upon service completion, then the customer returns to queue $i-1$ to receive a new service.
- *Loss* mechanism: server $i - 1$ can process a customer even though queue i is full. However, if queue i is still full upon service completion, then the customer attempting to enter queue i is lost.

These different mechanisms allow us to model different behaviors encountered in real systems. In particular, manufacturing systems are often modeled using the blocking-after-service and also the blocking-before-service mechanims, whereas communication networks are often modeled using the block-and-recirculate mechanism or the loss mechanism. Of course, more than one mechanism may appear in a single model, that is, some servers may operate according

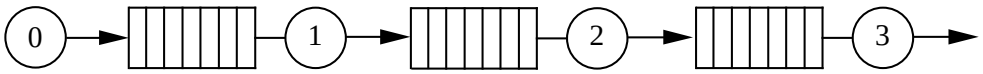


Figure 1.7: Equivalent representation of the queueing network of Figure 1.6

to a given mechanism while others operate according to another mechanism. As for the arrival process, it is usually assumed that it operates according to a loss mechanism, that is if a customer arrives to the system while queue 1 is full, it is lost. (Note that this is consistent with what we have described for single queues.)

Remark It is worth noticing that the arrival process of open queueing networks can equivalently be represented by an additional server, server 0. Server 0 operates as a source, i.e., a server with an infinite supply of customers in front of it. As a result, the server is always busy. The service time distribution of server 0 is identical to the interarrival time distribution. Such models are referred to as *saturated models*. This is in particular of interest when dealing with finite capacity queueing networks. For the purpose of illustration, the saturated model equivalent to the open tandem model of Figure 1.6 is shown in Figure 1.7.

The introduction of finite capacity queues can be generalized to open queueing networks with general topology. Again, the four different mechanisms introduced above can be considered. However, some further specifications are required. For instance, in the case of blocking after service, if both server i and server j are blocked because their destination queue, queue k, is full, one has to define which server, i or j, will transfer its customer when a space becomes available in queue k. This can be specified according to some fixed allocation rules such as first-blocked-first-unblocked or priority to a given server, or based on a state-dependent allocation rule. A similar discussion applies to the case of blocking-before-service. Refer to [10] for a detailed discussion of this issue.

Finite capacity buffers can also be introduced in closed queueing networks as well as in multiclass queueing networks. Finally, we note that in the case of multiclass queueing networks with finite capacity under the loss assumption, the push-out mechanism discussed in Section 1.1.3 can be used again.

State dependent routing mechanisms can also be introduced in queueing networks. It corresponds to a situation in which the routing of the customers depends on the actual state of the network, i.e., the number of customers currently present at each queue. One such mechanism is the so-called shortest queue routing, i.e., when a customer completes its service at station i, it joins the queue that has the smallest number of customers, among a predetermined set of queues. The most general situation can be defined in terms of a probabilistic state-dependent routing mechanism. For illustration, consider a closed single-class queueing network. Let $\mathbf{n} = (n_1, ..., n_M)$ denote the state vector of the system, where n_i is the number of customers present at station i (either waiting or receiving service). A general probabilistic state-dependent routing

can be characterized by the following set of probabilistic routings: $r_{i,j}(\mathbf{n})$, for all i, j and $\mathbf{n}$.

It is worth noting that finite capacity queueing networks under the block-and-recirculate or the loss mechanism can equivalently be viewed as queueing networks with state-dependent routing mechanisms.

Finally, batch arrivals and/or batch services can be introduced in queueing networks. Time between successive arrivals of batches and service times of batches at the different stations can be specified in a similar way as it was done for single queues. A further issue is related to the routing mechanism. Indeed, when a service completion of a batch occurs at a station, the probabilistic routing may apply to the entire batch, or to each customer individually. In the former case, the whole batch is sent to a single destination station, while in the latter, each customer of the batch chooses its own destination according to the probabilistic routing, independently of the routing of the other customers.

1.2 Extended Queueing Network Formalism

The basic queueing network formalism presented above is useful for modeling some important classes of real systems. However, it does not allow us to represent real systems involving synchronization mechanisms between different customers and assembly/disassembly (also called fork/join) mechanisms. This can be handled by using a more general formalism called the *extended queueing network formalism*. Let us introduce the main features of this formalism.

Consider first a simple *assembly* mechanism, also called *join*, illustrated in Figure 1.8. An assembly server has a set of upstream queues and one downstream queue. Its activity is to assemble (or join) a set of items, one from each of the upstream queues, into a single item that joins its output queue. Thus, as soon as there is a least one item in each of the upstream queues, the assembly operation can begin. After completion of the assembly operation, the assembled item is moved to the downstream queue (and of course one item is removed from each of the upstream queue). The assembly time can be characterized in the same way as the service time for simple servers; see Section 1.1.1.

Consider now a simple *disassembly* mechanism, also called *fork* illustrated in Figure 1.9. A disassembly server has one upstream queue and a set of downstream queues. Its activity is to disassemble (or fork) a single item into a set of items. The disassembly operation begins as soon as there is one item in the upstream queue. After completion of the disassembly operation, one item (resulting of the disassembly) is moved to each of the downstream queues (and of course the item is removed from the upstream queue). The disassembly time can be characterized in the same way as the service time for simple servers.

Of course, these two mechanisms can be combined to give rise to the so-called *assembly/disassembly* mechanism illustrated in Figure 1.10. An assembly/disassembly (A/D) server has a set of upstream queues and a set of downstream queues. The A/D operation can begin when there is at least one item in each of the upstream queues. Upon completion of the A/D operation, one item

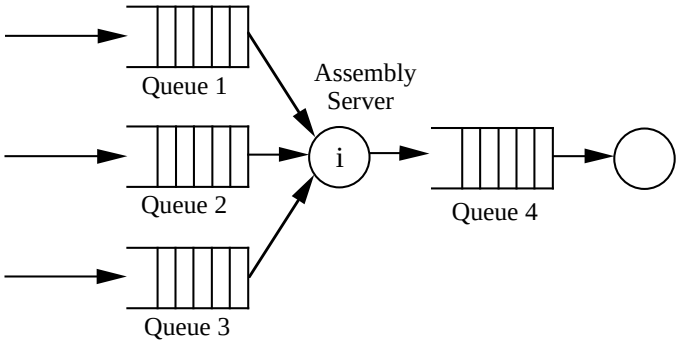


Figure 1.8: Illustration of a simple assembly mechanism

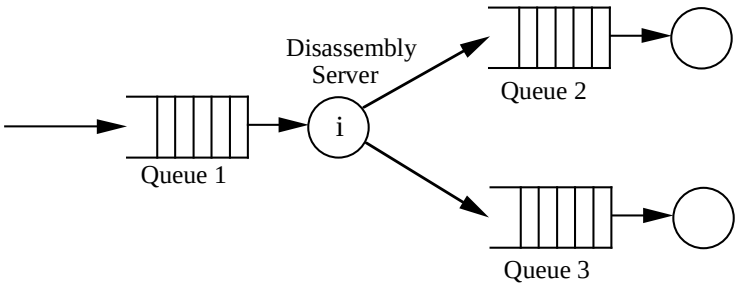


Figure 1.9: Illustration of a simple disassembly mechanism

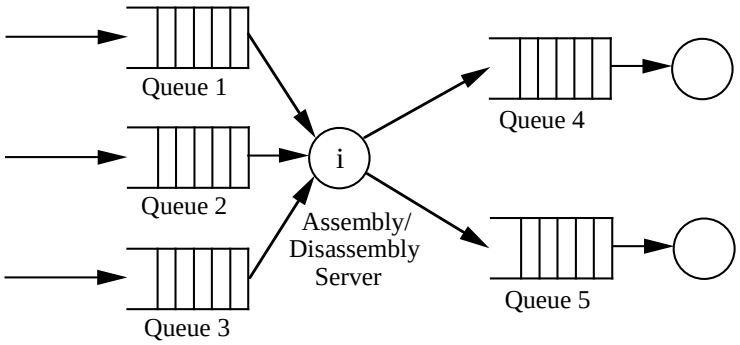


Figure 1.10: Illustration of the assembly/disassembly mechanism

is removed from each of the upstream queues and one item is added to each of the downstream queues. The assembly/disassembly time can be characterized in the same way as the service time for simple servers. As in the case of simple queues, there may be several resources performing the A/D operation. Thus, we may consider single-server, multiple-server and infinite server A/D stations.

Remark In all the models belonging to the basic queueing networks formalism described above, a server (or a set of servers in the case of multiple-server queues) is always associated with a queue, and vice-versa. This is no longer the case when dealing with the extended queueing networks formalism. Indeed, a server may now have more than one input buffer (assembly or join server) and more than one output buffer (disassembly or fork server). This implies that studying properties and deriving analytical performance evaluation methods is in general much more difficult for classes of models belonging to the extended queueing networks formalism.

One important special case of assembly/disassembly servers is the case where the A/D operation is instantaneous, i.e., the A/D time is zero. In this case, a special graphical representation is sometime used in the literature, namely the circle is replaced by a "bar". A typical situation is the pure synchronization mechanism illustrated in Figure 1.11. In this example, three flows of customers are synchronized. Customers of the first (resp. second and third) flow aim at joining queue 4 (resp. queue 5 and queue 6). However, they first need to go through the synchronization station. The customers of the first (resp. second and third) flow first join queue 1 (resp. queue 2 and queue 3). As soon as at least one customer is present in each of the upstream queues, three customers (one from each of the upstream queues) are allowed to proceed to their respective destination queues, namely queues 4, 5 and 6.

Routing mechanisms can also be incorporated in the extended queueing network formalism in a similar way as for the basic queueing network formalism. Let us illustrate this by means of the example shown in Figure 1.12. Each A/D operation at station i results in the creation of two items. One of them joins

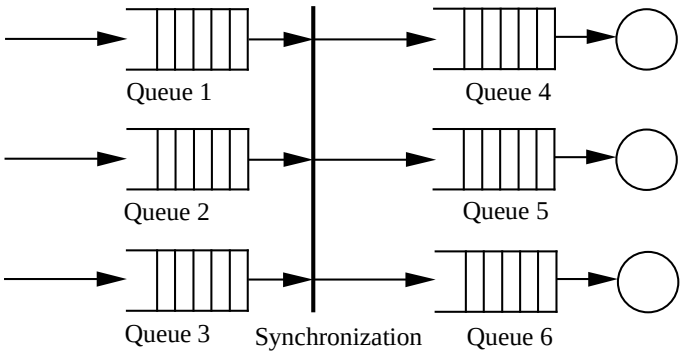


Figure 1.11: Illustration of a pure synchronization mechanism

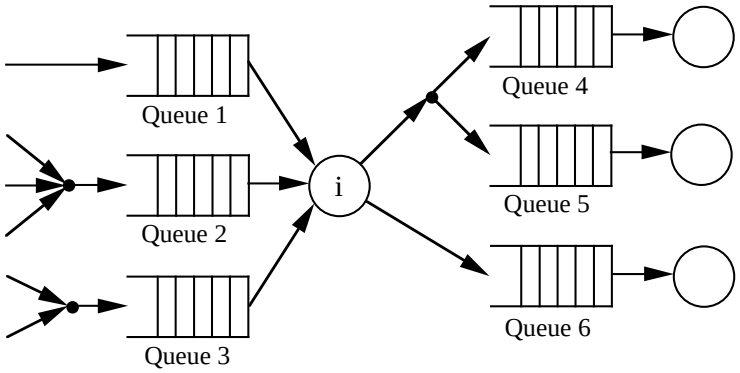


Figure 1.12: Illustration of the combination of A/D and routing mechanisms

queue 6, while the other one joins either queue 4 or queue 5. This routing decision may again be based on a probabilistic choice.

As for basic queueing networks, the notion of classes of customers may be introduced to differentiate between customers that have different behaviors in terms of the arrival processes, service processes, service disciplines and routing mechanisms. Also, finite capacity constraints as well as state dependent routing mechanisms can be incorporated. As for finite capacity A/D networks, various blocking mechanisms can be represented [6]. They are a generalization of those described for basic queueing networks in Section 1.1.5.

The notion of batch processing introduced in the basic queueing network formalism can be generalized to the extended queueing network formalism. Consider again the assembly station shown in Figure 1.8. So far, the assembly server takes one item in each of the upstream queues and assembles them into a single item. However, a more general situation arises in the case where to produce one single assembly, several items of the same upstream queue are required. We may for instance specify that to build an assembly, 2 items from queues 1, 1

item from queue 2, and 4 items from queue 3, are required. This multiplicity of items required in an assembly operation can be characterized by associating a *batch size* (or *weight*) with each of the upstream queues of an assembly station. More generally, for an A/D station, one may specify a batch size for each of the upstream queues as well as a batch size for each of the downstream queues. Consider for instance the A/D station shown in Figure 1.10. Suppose the batch sizes of queues 1 to 5 are: 2, 1, 4, 3, 1. That means that any time an A/D operation is performed, 2, 1 and 4 items are removed from queues 1, 2 and 3, respectively, and 3 and 1 items are added to queues 4 and 5, respectively. Of course, the A/D operation can only begin if there are at least 2 items in queue 1, 1 item in queue 2, and 4 items in queue 3.

Bibliography

[1] M. H. Ammar, S. B. Gershwin, "Equivalence Relations in Queuing Models of Fork/Join Queueing Networks with Blocking", *Performance Evaluation* **10**, pp. 233-245, 1989.

[2] R. G. Askin, C. R. Standridge, "Modeling and Analysis of Manufacturing Systems", John Wiley, 1993.

[3] F. Baskett, K. M. Chandy, R. R. Muntz, F. G. Palacios "Open, Closed and Mixed Networks of Queues with Different Classes of Customers," *Journal of the ACM*, **22**, 2, pp. 248-260, 1975.

[4] J. A. Buzacott, J. G. Shanthikumar, "Stochastic Models of Manufacturing Systems", Prentice Hall, Englewood Cliffs, N. J., 1993.

[5] C. H. Sauer, K. M. Chandy, "Computer Systems Performance Modling, Prentice Hall, 1981.

[6] Y. Dallery, Z. Liu, D. Towsley, "Properties of Fork/Join Queueing Networks with Blocking under Various Operating Mechanisms," *IEEE Transactions on Robotics and Automation*, **13**, 4, pp. 503-518, 1997.

[7] L. Kleinrock, "Queueing Systems", John Wiley, New York, 1975.

[8] E. D. Lazowska, J. Zahorjan, G. S. Graham, K. C. Sevcik, "Quantitative System Performance", Prentice-Hall, 1984.

[9] I. Mitrani, "Modelling of Computer and Communication Systems, Cambridge University Press, 1987.

[10] H. G. Perros, "Open Queueing Networks with Blocking", *Stochastic Analysis of Computer and Communication Systems"*, (ed. H. Takagi), North-Holland, Amsterdam, Netherlands, 1989.

Chapter 2

Untimed Petri Nets

2.1 Introduction

Typical discrete event dynamic systems (DEDS) exhibit parallel evolutions which lead to complex behaviours due to the presence of synchronisation and resource sharing pheonomena. *Petri nets (PN)* are a mathematical formalism which is well suited for modelling concurrent DEDS: it has been satisfactorily applied to fields such as communication networks, computer systems, discrete part manufacturing systems, etc. Net models are often regarded as self documented specifications, because their graphical nature facilitates the communication among designers and users. The mathematical foundations of the formalism allow both correctness (i.e., logical) and efficiency (i.e., performance) analysis. Moreover, these models can be (automatically) implemented using a variety of techniques from hardware to software, and can be used for monitoring purposes once the system is readily working. In other words, they can be used all along in the life cycle of a system.

Rather than a single formalism, PN are a family of them, ranging from low to high level, each of them best suited for different purposes. In any case, they can represent very complex behaviours despite the simplicity of the actual model, consisting of a few objects, relations, and rules. More precisely, a PN model of a dynamic system consists of two parts:

1. A *net structure*, an inscribed bipartite directed graph, that represents the static part of the system. The two kinds of nodes are called places and transitions, pictorially represented as circles and boxes, respectively. The places correspond to the state variables of the system and the transitions to their transformers. The fact that they are represented at the same level is one of the nice features of PN compared to other formalisms. The inscriptions may be very different, leading to various families of nets. If the inscriptions are simply natural numbers associated with the arcs, named weights or multiplicities, *Place/Transition (P/T) nets* are obtained. In this case, the weights permit the modelling of bulk services and arrivals.

A historically and conceptually interesting subclass of P/T nets is obtained when all weights are one. These nets are said to be *ordinary*, and they lead, for example, to a straightforward but important generalisation of automata models. More elaborate inscriptions, associated with places, transitions, and arcs, lead to the so called High Level Petri Net formalisms.

2. A *marking*, pictorially represented by tokens inside the places, that represents a distributed overall state on the structure. The marking of a place (state variable) is its state value. A net system is a net structure together with an initial marking. The system dynamics (i.e., the system behaviour) is given by the evolution rules for the marking. The basic rule allows the occurrence of a transition when the input state values fulfill some condition expressed by the arc inscriptions. The occurrence of a transition changes the values of its adjacent state variables, according again to the arc inscriptions.

The above separation allows one to reason on net based models at two different levels: *structural* and *behavioural*. From the former we may derive some "fast" conclusions on the possible behaviours of the modelled system. Purely behavioural reasonings can be more conclusive, but they may require costly computations, or even they may not be feasible. The structural reasoning can be regarded as an *abstraction* of the behavioural one: for instance, instead of studying whether a given system has a finite state space, we might investigate whether the state space is finite *for every* possible initial state; or we could study whether *there exists* an initial state that guarantees infinite activity rather than deciding this for a given initial state, etc.

The *interpretation* of a model precises the semantics of objects and their behaviour, eventually making explicit the connection of this model to the external world within a given type of application (i.e., the interpretation considers the environment in which the model will be exercised). An interpretation may give a "physical" meaning to the net's entities (places, transitions, tokens), evolution conditions and, possibly, will define the actions generated by the evolutions. Interpreted graphs, apart from PN, are common in the modelling of systems. Among many others, the following are classic (state based) graph interpretations: State Diagrams (SD) and Algorithmic State Machines (ASM) representing sequential switching systems, State Transition Rate Diagrams (STRD) representing continuous time homogeneous Markov Chains, Resource Allocation Graphs (RAG), Program Evaluation and Review Technique (PERT) graphs, etc. Some represent the global states (SD, ASM, STRD), others represent the state in a distributed fashion (RAG, PERT, PN).

The basic PN formalism can be further interpreted, in order to describe different perspectives of a given system along its life cycle. The variety of interpretations yields a major advantage of modelling with nets: Net systems provided with appropriate interpretations can be used along the design and operation of systems, using a single family of formalisms, and basic concepts and results are shared (can be reused) by the different interpretations, leading to some economy in the analysis and synthesis of models. Taking into account

the scope of this book, a particularly interesting family of net interpretations is obtained when time and probabilities are associated with the model (see Chapter 3). The present chapter is devoted to the introduction of the basic PN formalism, where the dynamics of the net system is governed only by the net evolution rules, and not by external events, timing, etc. This basic formalism is conventionally referred to as *autonomous* PN. Also the name *untimed* PN is used when the kind of further interpretations one has in mind have something to do with time. The autonomous evolution of a net model is said to be (completely) non deterministic in the sense that neither the resolution of conflicts nor the occurrence time of an enabled transition are specified at all.

The rest of the chapter is organised as follows: In Section 2.2 we introduce what we consider the basic PN formalism, and explore its abilities for the modelling of systems. Other PN formalisms are presented in Sections 2.3, 2.4, and 2.5.

2.2 Place/Transition Nets and Systems

In this section we introduce formally Place/Transition net systems, emphasising the distinction between net (the structure) and system (the net with a marking, and its evolution rules), and showing with an example how P/T systems are able to model DEDS. We describe in some detail several modelling features of P/T systems, namely the locality of states and actions, and the representation of the diverse phenomena found in parallel and distributed systems. This modelling features allow to represent in a natural way typical situations such us sinchronisation, mutual exclusion, conflicts, etc., and to apply both top-down and bottom-up synthesis methodologies to the design of systems.

2.2.1 Net Structure

In PN systems the state is described in a distributed fashion by means of a set of state variables. Places (represented as circles) are the support of these state variables. Individual actions, which transform the state variables they are connected to, are modelled by transitions (represented as bars or boxes). In the Place/Transition formalism, places and transitions are related through a weighted flow relation. Let us give now the formal definitions and see some examples.

Definition 2.1 (P/T net, graph oriented)
A Place/Transition (P/T) net *is a four-tuple:*

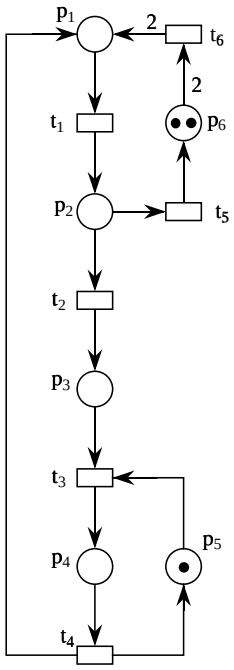
$$\mathcal{N} = \langle P, T, F, W \rangle$$

where:

1. *P and T are disjoint finite non empty sets, the* places *and* transitions *respectively.*

2. $F \subseteq (P \times T) \cup (T \times P)$ *is the* flow relation *(set of directed arcs). Without loss of generality isolated nodes are forbidden:* $\mathrm{dom}(F) \cup \mathrm{range}(F) = P \cup T$.

3. $W : F \to \mathbb{N}_+$ *assigns a* weight *or* multiplicity *to each arc.*



$$\mathbf{Pre} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

$$\mathbf{Post} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 2 \\ 1 & -1 & 0 & 0 & -1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 \end{bmatrix}$$

Figure 2.1: A Place/Transition net system and its incidence matrices.

The graph of Figure 2.1 — do not care about the black dots in places by now — is a net structure. Arcs are labelled with natural numbers, the arc weights or multiplicities. By convention, unlabelled arcs are weighted one. All the arc weights in the example net are one, except for $W(p_6, t_6)$ and $W(t_6, p_1)$ that are two. A place p is an input (resp. output) place of transition t if there exists an arc going from p to t (resp. from t to p). In Figure 2.1, $\{p_3, p_5\}$ are input places of t_3 while $\{t_2, t_5\}$ are the output transitions of p_2. Observe that the adjacency relations between nodes, which will determine later the relationship between states and actions, are fixed by the net structure, hence they are static.

There is an alternative representation of P/T nets where the flow relation is described in matrix form. It naturally leads to an interesting state equation-like description of the system evolution.

Definition 2.2 (P/T net, matrix oriented)
A P/T net is a four-tuple:

$$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$$

where:

1. *P and T are as in Definition 2.1.*
2. *$\mathbf{Pre}, \mathbf{Post} \in \mathbb{N}^{|P|\times|T|}$ are the* pre- *and* post- incidence matrices.

With the matrix notation, there is an arc going from place p to transition t when $\mathbf{Pre}[p,t] \neq 0$, and the value of $\mathbf{Pre}[p,t]$ is precisely the arc weight. Similarly, for post-incidence, $\mathbf{Post}[p,t] = W(t,p)$ when different from zero.

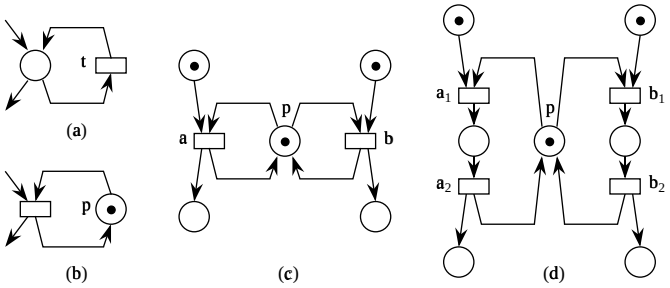


Figure 2.2: Self-loops.

The dot notation is used for pre- and post-sets of nodes: ${}^\bullet v = \{u \mid \langle u,v\rangle \in F\}$ and $v^\bullet = \{u \mid \langle v,u\rangle \in F\}$, a notation that is extended to sets. For instance, ${p_2}^\bullet = \{t_2, t_5\}$, and ${}^\bullet T \cup T^\bullet = P$. A transition t such that $|t^\bullet| > 1$ (resp. $|{}^\bullet t| > 1$) is called a *fork* (resp. a *join*). A place p such that $|{}^\bullet p| > 1$ (resp. $|p^\bullet| > 1$) is called an *collector* (resp. a *distributor*). In the example of Figure 2.1, t_4 is a fork, t_3 is a join, p_2 is a distributor, and p_1 is a collector. A pair made up of a place p and a transition t is called a *self-loop* if p is both input and output of t. Figures 2.2 (a), (b), and (c) show several self-loops. A net is said to be *pure* if it has no self-loop. Pure nets are completely characterised by a single matrix: $\mathbf{C} = \mathbf{Post} - \mathbf{Pre}$. This is called the *incidence matrix* of the (pure) net. Positive (negative) entries in $\mathbf{C}$ represent the post- (pre-) incidence function. If the net is not pure, $\mathbf{C}$ "does not see" the self-loops (thus it is not an incidence matrix but just a *token flow matrix* as will be shown later on).

By reversing arcs or interchanging places and transitions we get the *reverse net*, $\mathcal{N}^r$, or the *dual net*, $\mathcal{N}^d$, of $\mathcal{N}$. Both transformations together lead to the *reverse-dual* or *transpose net*, $\mathcal{N}^{rd}$. Sometimes in net theory relations are established between a net and its reverse, dual, or reverse-dual.

$\mathcal{N}$	$\langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$	$\mathbf{C}$
$\mathcal{N}^r$	$\langle P, T, \mathbf{Post}, \mathbf{Pre}\rangle$	$\mathbf{-C}$
$\mathcal{N}^d$	$\langle T, P, \mathbf{Post}^\perp, \mathbf{Pre}^\perp\rangle$	$\mathbf{-C}^\perp$
$\mathcal{N}^{rd}$	$\langle T, P, \mathbf{Pre}^\perp, \mathbf{Post}^\perp\rangle$	$\mathbf{C}^\perp$

A net $\mathcal{N}'$ is *subnet* of $\mathcal{N}$ (written $\mathcal{N}' \subseteq \mathcal{N}$) when $P' \subseteq P$, $T' \subseteq T$ and its pre- and post-incidence matrices are $\mathbf{Pre}' = \mathbf{Pre}[P', T']$ and $\mathbf{Post}' = \mathbf{Post}[P', T']$.

Subnets are generated by subsets of places *and* transitions. When a subnet is generated by a subset V of nodes of a single kind, it is assumed that it is generated by $V \cup {}^\bullet V \cup V^\bullet$. Subnets generated by a subset of places (transitions) are called P- (T-) subnets.

2.2.2 Net Systems: Marking and Token Game

The structure of a net is something static. Assuming that the behaviour of a system can be described in terms of the state and its changes, the dynamics on a net structure is created by defining its initial state and the state evolution rule.

Definition 2.3 (Marking and P/T system)
The marking *of a net $\mathcal{N}$ is a place indexed vector, $\mathbf{m} \in \mathbb{N}^{|P|}$, which assigns a non negative integer (number of tokens) to each place. A* P/T net system *is the pair $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ (or, more explicitly, $\mathcal{S} = \langle P, T, F, W, \mathbf{m_0} \rangle$ or $\mathcal{S} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$) where $\mathcal{N}$ is a P/T net and $\mathbf{m_0}$ is its* initial marking.

The number of tokens at a place represents the local state of the place (i.e., the value of the state variable represented by that place, which in the P/T formalism is an integer, so it can be interpreted as a *counter* or *store*). The state of the overall net system is defined by the collection of local states of the places. Therefore, the vector $\mathbf{m}$ is the state vector of the DEDS described by the net system. Pictorially, we put $\mathbf{m}[p]$ black dots (tokens) in the circle representing place p. The marking of the net system in Figure 2.1 is $\mathbf{m_0} = [0, 0, 0, 0, 1, 2]$. For the sake of convenience, we shall often use a bag-like notation for markings, e.g., $\mathbf{m} = p_5 + 2p_6$.

The evolution of the distributed state is defined through a firing or occurrence rule, informally named as the "token game". This is because net structures can be seen as a sort of "checkers", the tokens as "markers", and the firing rule as the "game rule". Transitions represent potential "moves" in the token game. (Observe, though, that tokens are not *moved* but *destroyed and created* at each "move", possibly in a different number.)

Definition 2.4 (Enabling and occurrence)
The marking in a net system evolves as follows:

1. *A transition is said to be* enabled *at a given marking when each input place has at least as many tokens as the weight of the arc joining them. Formally, t is enabled at $\mathbf{m}$ iff $\mathbf{m} \geq \mathbf{Pre}[P, t]$.*

   The number of simultaneous enablings of a transition t at a given marking $\mathbf{m}$ is called its enabling degree, *and is denoted by $\mathbf{e}(\mathbf{m})[t]$. Formally, $\mathbf{e}(\mathbf{m})[t] = \max\{k \in \mathbb{N}_+ \mid \mathbf{m} \geq k \cdot \mathbf{Pre}[P, t]\}$. (The enabling degrees at $\mathbf{m}$ of all the transitions are collected in the* enabling vector, $\mathbf{e}(\mathbf{m})$.*)*

2. *The* occurrence, *or* firing, *of an enabled transition is an atomic operation that removes from (adds to) each input (output) place a number of tokens*

equal to the weight of the arc joining the place (transition) to the transition (place). Formally, the occurrence of t at marking $\mathbf{m}$, denoted by $\mathbf{m} \xrightarrow{t} \mathbf{m}'$, yields the marking $\mathbf{m}' = \mathbf{m} + \mathbf{C}[P, t]$.

The pre-condition of a transition can be seen as the resources required for the transition to be fired. The post-condition represents the resources produced by the firing of the transition. The only transition enabled in the net system of Figure 2.1 is t_6. Its firing leads to the marking $2p_1 + p_5$, where t_1 is enabled with enabling degree two: t_1 could occur now two times "simultaneously", *self-concurrency* or *reentrancy* can be modelled. When a transition is required not to be self-concurrent, e.g., because it represents a single server, we can explicitly show it in the model by using a self-loop like that in Figure 2.2 (b). The same schema can be applied to model a k-servers transition by putting k tokens in place p.

Remark Observe that in the firing rule of our abstract model, enabled transitions *are never forced to fire*. This is a form of non determinism. In practical modelling the interpretation partially governs the firing of enabled transitions (e.g., depending on whether or not an external event associated to an enabled transition occurs). It must also be noticed that *it is not precised whether the occurrence of a transition takes some time*, since time has not been introduced yet. This is again dependant on the interpretation we give to the model. Generally speaking, a transition can *implement a system activity*, so its occurrence would take some time, or it can *represent the completion of a system activity*, thus being instantaneous. In this book, we mainly consider the latter interpretation. Anyway, transitions representing system activities can be implemented by a path instantaneous begin transition → activity in course place → instantaneous end transition, as in Figure 2.2 (d) with respect to (c).

Interleaving Semantics: Sequential Observations

A common way of describing the behaviour of a P/T system is by means of its sequential observations. So to say, the observer is supposed to "see" only single events, e.g., one transiton occurring at a time. The *interleaving semantics* of a net system is given by all possible sequences of individual transition occurrences that could be observed from the initial marking. If two transitions a and b are enabled simultaneously and the occurrence of one does not disable the other, in principle they could occur at the same time, but the sequential observer will see either a followed by b or viceversa. The name interleaving semantics comes from this way of seing simultaneous occurrences.

Definition 2.5 (Sequences, language, and reachability)
Let $\mathcal{S}$ be a P/T system.

1. *An* occurrence or firing sequence *from $\mathbf{m}$ is a sequence $\sigma = t_1 \cdots t_k \cdots \in T^\omega$ such that $\mathbf{m} \xrightarrow{t_1} \mathbf{m_1} \cdots \xrightarrow{t_k} \mathbf{m_k} \cdots$. If the firing of sequence σ yields the marking $\mathbf{m}'$, this is denoted by $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$.*

2. *The* language *of* $\mathcal{S}$*, denoted by* $\mathrm{L}(\mathcal{S})$*, is the set of all the occurrence sequences from* $\mathbf{m_0}$.

3. *The* reachability set *of* $\mathcal{S}$*, denoted by* $\mathrm{RS}(\mathcal{S})$*, is the set of all the markings reachable from* $\mathbf{m_0}$ *by firing some sequence in* $\mathrm{L}(\mathcal{S})$.

4. *The* reachability graph *of* $\mathcal{S}$*, denoted by* $\mathrm{RG}(\mathcal{S})$*, is a labelled graph where the vertices are the reachable markings, and there is an edge labelled* t *from vertex* $\mathbf{m}$ *to vertex* $\mathbf{m}'$ *iff* $\mathbf{m} \xrightarrow{t} \mathbf{m}'$.

(See more on the reachability graph and its use in the analysis in Chapter 6.)

Concurrent Semantics: Multiple Enablings and Steps

If firings take some time, the occurrences of several transitions could be simultaneous — or overlap in time. Physically, it will never be the case if firings are supposed to be instantaneous, unless a strong deterministic timing is assumed, but representing them in a *step* makes explicit the fact that they need not occurring in a precise order.

Definition 2.6 (Steps)
A step *enabled at* $\mathbf{m}$ *is a multiset of transitions such that they could occur "simultaneously". A step can be represented in vector form:* $\mathbf{s}[t]$ *denotes the number of times that transition* t *is in step* $\mathbf{s}$*. With this notation, the step* $\mathbf{s}$ *is enabled at* $\mathbf{m}$ *iff* $\mathbf{m} \geq \mathbf{Pre} \cdot \mathbf{s}$*. The occurrence of step* $\mathbf{s}$ *can be denoted by* $\mathbf{m} \xrightarrow{\mathbf{s}} \mathbf{m}'$*, or* $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$ *if* σ *is an arbitrary* sequentialisation *of* $\mathbf{s}$*. In fact, every sequentialisation of the step is fireable so, in practice, the reachable markings can be computed considering individual transition occurrences only.*

In the reachability graph we could represent an arc from a marking to another one labelled with the step, but this would be in some sense redundant, since all the possible sequentialisations of the step would appear too. For example, from $p_2 + p_3 + p_5$ in the net of Figure 2.1, $t_5 + t_3$ is a (maximal) enabled step, and therefore the sequences $t_5 t_3$ and $t_3 t_5$ are fireable.

Interleaving semantics assumes that only the language is important to describe the behaviour. Instead, if also the steps and/or the enabling degrees are important, we will speak of a *concurrent semantics*. Later on we shall show examples where the distinction between interleaving and concurrent semantics is important.

State Equation and Semiflows

The *firing count (or Parikh) vector* of a sequence σ is defined as $\boldsymbol{\sigma}[t] = \#(t, \sigma)$. Let $\mathcal{S}$ be a P/T system and let $\mathbf{m} \in \mathrm{RS}(\mathcal{S})$. Integrating the evolution equation in Definition 2.4.2 (i.e., $\mathbf{m} \xrightarrow{t} \mathbf{m}' \Rightarrow \mathbf{m}' = \mathbf{m} + \mathbf{C}[P, t]$) from $\mathbf{m_0}$ to an arbitrary $\mathbf{m}$ we get:

$$\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m} \Rightarrow \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \tag{2.1}$$

This is known as the *state equation* of the system, on which several structural analysis methods are based (see Chapter 6). In particular, left and right nonnegative annullers of the token flow matrix are called *P-* and *T-semiflows*, respectively. These dual structural objects lead to linear invariant laws on the possible behaviours. If $\mathbf{y}$ is a P-semiflow (i.e., $\mathbf{y} \geq \mathbf{0}$ and $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$), then $\mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0}$. For instance, $\mathbf{y} = p_4 + p_5$ is a P-semiflow of the net in Figure 2.1, and it induces the invariant $m_4 + m_5 = 1$. If $\mathbf{x}$ is a T-semiflow (i.e., $\mathbf{x} \geq \mathbf{0}$ and $\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$), then the firing of any sequence with firing count vector $\mathbf{x}$ leads from a marking back to itself. For instance, $\mathbf{x} = 2t_1 + 2t_5 + t_6$ is a T-semiflow corresponding to the cyclic sequence $t_6 t_1 t_1 t_5 t_5$. It is important to note that these invariant laws are *structural*, they abstract from the initial marking. A P-semiflow induces a marking invariant law *for every* initial marking, while a T-semiflow indicates that *there exist* initial markings for which a corresponding cyclic behaviour is possible.

Modelling DEDS with Net Systems

Petri nets, as introduced so far, are a mathematical formalism, in the same sense as differential equations are. While the latter are useful for describing continuous dynamical systems, the former are introduced for the description of (asynchronous) DEDS. Let us illustrate now a possible physical interpretation that PN can be given, by means of an example.

Example 2.1 (Multicomputer PLC) A Programmable Logic Controller (PLC) is a device to control a process or plant. It works in a cyclic fashion: plant variables are sampled, some calculations are performed, and actions to the plant are emitted, once and again. In order to control a complex or distributed plant we may have a PLC with multicomputer architecture. Assume we want to model a PLC consisting of two computers each of them containing a double-access memory which are connected by a shared bus. The two computers synchronise to start a control cycle. Then they perform their calculations independently, unless they need to read external data (i.e., residing in another computer's memory) using the common bus. At this level of detail, we disregard the bus control policy.

The net system of Figure 2.1 is an abstract model of such a multicomputer PLC, where the identity of the computers is disregarded (they are modelled as indistinguishable). The tokens in place p_6 model the two computers that will start a control cycle when t_6 fires. The tokens in place p_1 model the computers engaged in private calculations. The occurrence of t_1 models the fact that a computer finishes a piece of these computations. Then, either it has completed its actions in the present cycle and becomes idle again (t_5 fires) until a new cycle is started, or it requires external data (t_2 fires). If the bus is ready, what is modelled by p_5 being marked, then the computer can use it (t_3 fires and p_4 becomes marked). Computers queue up in p_3 for the bus when it is not ready. (Observe, though, that no queueing policy is specified.) Once the communication is completed, the computer goes on with its private computations and the bus is released (firing of t_4).

2.2.3 Locality and the Synthesis of Models

It is clear from the definitions of the net structure, state representation, and evolution rule, that there exists a locality principle on both states and actions. Place p_1 of the example net informs on the number of computers performing private calculations, independently of the bus or even of other informations we might have on the computers. Transition t_6 models the start of a control cycle, which changes only the state of the computers.

The locality both of the state representation and the actions is another crucial feature of PN, which permits the modelling of complex distributed behaviours. Its importance for the synthesis of models resides in the fact that nets can be locally modified, either refined or coarsened, with no alteration of the rest of the model. It is also possible that different nets, modelling different parts of a system, can be composed by sharing some nodes (representing common activities and states). In other words, nets can be synthesised using *top-down* and *bottom-up* approaches. Top-down synthesis is any procedure that starting with an initial — abstract — model, leads to the final model through stepwise refinements. In a bottom-up approach modules are produced, possibly in parallel by different groups of designers, and later composed.

As an example, assume we want to be more precise about "the private calculations", that consist on the parallel execution of some codes. Moreover, once the bus is released it must undertake some set-up operations before becoming ready for a new access. Figure 2.3 (a) shows a refined model of our multicomputer PLC, where both place and transitions refinements are illustrated. A modular way of constructing the net is shown in Figure 2.3 (b), as a sychronisation of the model of the computers and the model of the bus. (Place p'_4 modelling "bus busy" is redundant with place p_4 modelling "computer using bus", and can be removed.)

2.2.4 Causal Dependence, Conflicts, and Concurrency

Let us discuss now the adequacy of the formalism to modelling DEDS with concurrent or parallel activities. Informally, the three basic phenomena to be represented are *causal dependence*, i.e., some actions require that others are performed first, *conflicts*, i.e., some actions are alternative, a decision on which one will occur must be taken, and *concurrency*, i.e., there are actions that may occur simultaneously. These phenomena are readily represented by PN models in a very natural way. They are the basis for the modelling of typical schemas in parallel and distributed systems, like mutual exclusion, join-assembly, rendezvous, etc. Example 2.1 will be used for illustration purposes.

Concurrency

Causal dependence and conflicts are classical notions in sequential systems, like finite automata, although the presence of concurrency modifies them somehow, so we shall start with the latter. Two transitions are concurrent at a given marking if they can occur "simultaneously", that is, in a step:

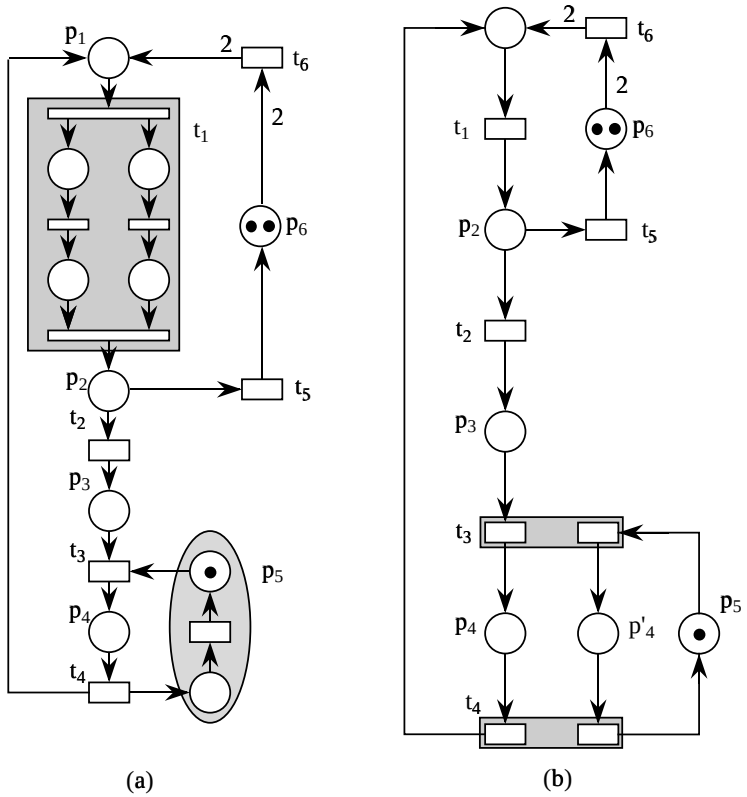


Figure 2.3: Synthesising the model of the multicomputer PLC of Example 2.1: (a) A refined model; (b) As a synchronisation of two sub-models.

Definition 2.7 (Concurrency relation)

Transitions t_i *and* t_j *are in* concurrency relation *at marking* $\mathbf{m}$*, denoted by* $\langle t_i, t_j \rangle \in \mathrm{Cc}(\mathbf{m})$*, when* $\mathbf{m} \xrightarrow{t_i} \mathbf{m}'$*,* $\mathbf{m} \xrightarrow{t_j} \mathbf{m}''$*,* $e_j(\mathbf{m}') > 0$*, and* $e_i(\mathbf{m}'') > 0$*.*

In other words, $\langle t_i, t_j \rangle \in \mathrm{Cc}(\mathbf{m})$ when $\mathbf{m} \geq \mathbf{Pre}[P, t_i] + \mathbf{Pre}[P, t_j]$. For instance, assume t_6 and then t_1 and t_2 are fired in the example, leading to $p_1 + p_3 + p_5$; therefore, transitions t_1 and t_3 are enabled and may occur simultaneously. Notice that steps allow to express *true concurrency*. In the case of interleaving semantics, as we mentioned, concurrency of two (or more) actions a and b is represented by the possibility of performing them in any order, first a and then b, or viceversa. Nevertheless, the presence of all possible sequentialisations of the actions does not imply that they are "truly" concurrent, as the example in Figure 2.2 (c) illustrates: a and b can occur in any order, but they cannot occur simultaneously, and in fact the step $a+b$ is not enabled. The distinction is specially important if transitions a and b were to be refined. In Figure 2.2 (d), a and b have been refined. It is clear that, for instance, a_1 and

b_2 are not concurrent, which is incongruent with a and b being concurrent.

Remark It is important at this point to distinguish between *concurrency of transitions* (as abstract model objects) and *concurrency of system activities*, which is again a matter of the interpretation of the model. In case transitions implement system activities, both things coincide: the occurrence of transitions would take some time, so their occurrences may overlap in time; in this sense, they could occur simultaneously. On the other hand, if transitions represent just the completion of some system activity (hence they are instantaneous) two system activities are concurrent when the transitions that represent their completion are simultaneously enabled. With the former interpretation the activities modelled by a and b in Figure 2.2 (c) are not concurrent. Nevertheless, if a and b represent the completion of two activities, these activities are concurrent (although the completion of one interrupts the other). Even the activities whose completion is modelled by t and t' in Figure 2.4 (a) would be concurrent, although these transitions are not concurrent either in an interleavings or a steps sense! With the interpretation of occurrences as completions, concurrency of two transitions means that the completion of one activity does not interrupt the other.

Causal Dependence

Roughly speaking, causal dependences are represented by the partial ordering of actions induced by the flow relation. For instance, it is clear in the example that t_4 occurs after t_3, and that to fire t_3, both t_2 and t_4 must occur before, in whichever order — apart from the first time. If intantaneous occurrences are considered, the (immediate or direct) causal dependence at a given marking can be formalised as the following relation between transitions:

Definition 2.8 (Causality relation)
Transition t_i *is in* direct causality relation *with* t_j *at marking* $\mathbf{m}$, *denoted by* $\langle t_i, t_j \rangle \in \mathrm{Cs}(\mathbf{m})$, *when* $\mathbf{m} \xrightarrow{t_i} \mathbf{m}'$ *and* $e_j(\mathbf{m}') > e_j(\mathbf{m})$.

If multiple transition occurrences were allowed, steps rather than individual occurrences of transitions should be considered in the definition.

Causal dependences appear in the form of sequences (e.g., t_4 follows t_3), as in sequential systems, but also in the form of synchronisations (e.g., t_3 is a synchronisation of a computer and the bus).

The very basic net construct used to model causal dependences is a place connecting two transitions. Transitions connected through a place are said to be in *structural causal relation* ($\langle t_i, t_j \rangle \in \mathrm{SCs}$ when $t_i{}^\bullet \cup {}^\bullet t_j \neq \emptyset$). The very basic net construct used to model synchronisations is a transition with more than one input place, i.e., a join transition (it takes its name from the fork-join schema, the reverse kind of transition — more than one output — is called a fork). It can also be said that a multiple arc from a place to a transition is a sort of synchronisation, because several "individuals" (tokens) must assemble in front

of the transition to enable it, as it happens with t_6, where the two computers synchronise to start a control activity.

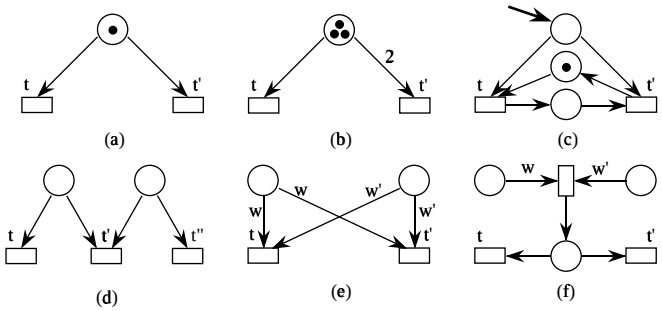


Figure 2.4: Conflicts and structural conflicts.

Conflicts

Regarding conflicts, in sequential systems they are clearly the situation in which two actions are enabled so one must be chosen to occur. For instance, Figure 2.4 (a) shows a conflict between t and t'. Things become more complicated in the case of concurrent systems, where the fact that two transitions are enabled does not necessarily imply that we must choose one. Sometimes, the "sequential" definition — there is a conflict when two transitions are enabled and the occurrence of one disables the other — is suitable, namely in 1-bounded systems. But in other cases a new definition is needed. Assuming the example net (Figure 2.1) was marked with $p_1 + p_2 + p_5$, t_5 and t_2 would obviously be in conflict, meaning that if the computer in p_2 is ready to finish it cannot need the bus, and viceversa. Consider now the marking $2p_2 + p_5$. Neither the occurrence of t_2 disables t_5 nor the converse, but the firing of one decreases the enabling degree of the other: so to say, each token must decide which way to go. Formally, *there is a conflict situation when the enabling vector is not an enabled step.* In Figure 2.4 (b), neither the occurrence of t or t' disables the other, but the firing of t' decreases the enabling degree of t from three to one. The enabling vector is $3t + t'$, while the (maximal) enabled steps are $3t$ and $t + t'$. By the way, this example shows that *conflict does not imply absence of concurrency*: t and t' are involved in a conflict, but they could occur concurrently, as in $t + t'$.

Once the notion of conflict has been clarified, we introduce a relation between transitions, to account for the transitions which are involved in a conflict situation at a given marking. Depending on whether we interpret occurrences as instantaneous or not, again we obtain slightly different definitions (and again we strengthen the former, due to the main orientation of the book):

Definition 2.9 (Conflict relation)

Transition t_i is said to be in effective conflict relation *with t_j at marking $\mathbf{m}$, denoted by $\langle t_i, t_j \rangle \in \mathrm{Cf}(\mathbf{m})$, when $\mathbf{m} \xrightarrow{t_i} \mathbf{m}'$ and $e_j(\mathbf{m}') < e_j(\mathbf{m})$.*

This relation is antisymmetric: In Figure 2.4 (b), t' is in effective conflict with t, but not the other way round, because a firing of t does not decrease the enabling degree of t'. If transition occurrences were not instantaneous, then we should take into account the possibility of multiple firings of a transition. Therefore, we should also say that t is in conflict with t', because firing t twice (or thrice) does change the enabling degree of t'. In such case, it would be better to say that two transitions t_i and t_j are in conflict relation at $\mathbf{m}$ when $\mathbf{m} \not\geq e_i(\mathbf{m}) \cdot \mathbf{Pre}[P, t_i] + e_j(\mathbf{m}) \cdot \mathbf{Pre}[P, t_j]$, which is symmetric. Observe that it has not been defined either how or when a given conflict should be solved, leading to non determinism in the behaviour.

The very basic net construct used to model conflicts is a place with more than one output transition, i.e., a distributor place. In fact, distributor places are needed to model conflicts, but the converse is not true. Due to the regulation circuit in Figure 2.4 (c), t and t' are never in effective conflict although they share an input place. The output transitions of a distributor place are said to be in *structural conflict relation* ($\langle t_i, t_j \rangle \in \mathrm{SCf}$ when ${}^\bullet t_i \cap {}^\bullet t_j \neq \emptyset$). This relation is reflexive and symmetric, but not transitive. Its transitive closure is named *coupled conflict relation*, and it partitions the transitions of a net into *coupled conflict sets* ($\mathrm{CCS}(t)$ denotes the coupled conflict set containing t). In Figure 2.4 (d) t and t'' are not in structural conflict relation but they are in coupled conflict relation, through t'.

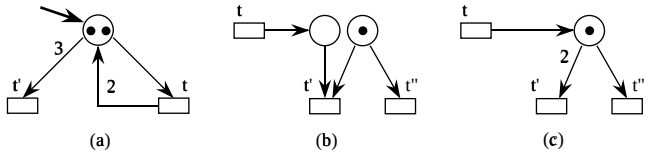


Figure 2.5: Non equal conflicts and confusion.

Remark Very often in the literature, our structural conflicts are called simply "conflicts", but we prefer to add the adjective structural to better distinguish from the behavioural, hence dynamical, notion of (effective) conflict, which depends on the marking. As we have noted, a structural conflict makes possible the existence of an effective conflict, but it does not guarantee it, e.g. Figure 2.4 (d), except for the case of *equal conflicts*, where all the transitions in structural conflict have the same precondition. Transitions t and t' are said to be in *equal conflict relation*, $\langle t, t' \rangle \in EQ$, when $t = t'$ or $\mathbf{Pre}[P, t] = \mathbf{Pre}[P, t'] \neq \mathbf{0}$. This equivalence relation partitions the transitions into *equal conflict sets*. The equal conflict set containing t is denoted by $\mathrm{EQS}(t)$. Figure 2.4 (e) shows an equal conflict set.

When structural conflicts are not equal, it may well be the case that the "conflicting" transitions become gradually enabled, as shown by the examples in Figure 2.5: none of them is in a conflict situation, although in the three cases one "conflicting" transition is enabled, namely t in (a) and t'' in (b) and (c); after firing t, a conflict appears in the three cases. An intriguing situation

arises when different sequentialisations of a step involve a conflict resolution or not. This is known as the *confusion* phenomenon, illustrated in Figure 2.5 (b) and (c): in both cases the step $t + t''$ can be fired. If we fire t first, a conflict between t' and t'' appears, that is solved in favour of t''. If we fire t'' and then t, no conflict appears. This phenomenon will be particularly annoying when considering priorities later on.

Control Flow and Synchronisation Schemas

It goes without saying that the habitual control flow structures (e.g., sequence, if-then-else, iteration, par-begin/par-end, etc.) are readily modelled using PN, as Figure 2.6 illustrates. (In this example the interpretation adds information on how to solve conflicts, or the actions that correspond to the firing of transitions.)

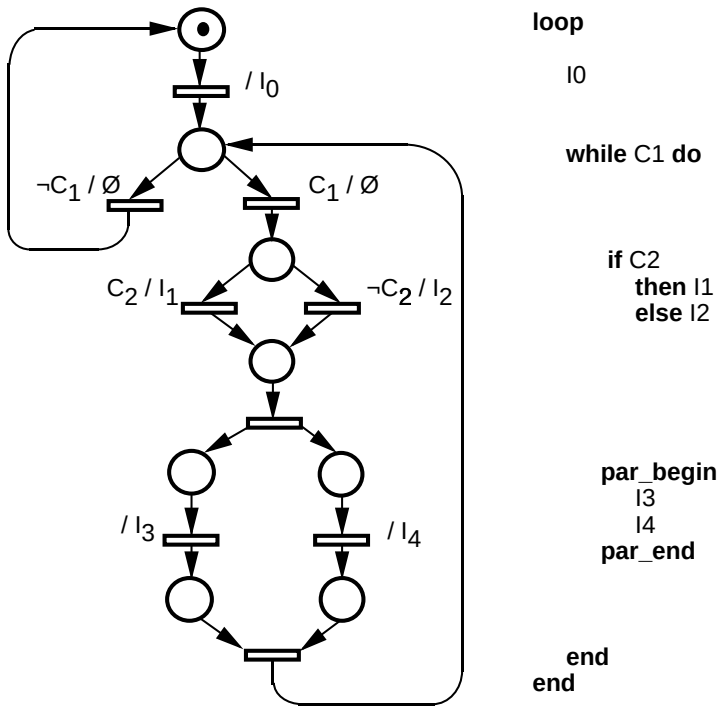


Figure 2.6: Control flow of a simple parallel-PASCAL-like program.

Also, conventional synchronisation schemas, like rendez-vous, semaphores, fork-join, mutex, etc., can easily be represented by net constructs, as Figure 2.7 illustrates.

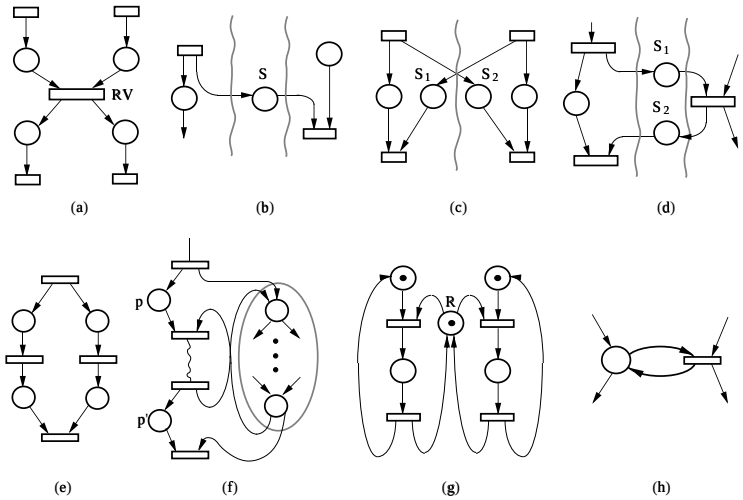


Figure 2.7: Conventional synchronisation schemas: (a) rendez-vous, (b) semaphore, (c) symmetric rendez-vous/semaphore, (d) asymmetric rendez-vous/semaphore (master/slave), (e) fork-join, (f) subprogram (p and p' must not be marked simultaneously), (g) mutex, (h) guard (condition reading).

2.2.5 Subclasses and Extensions of the Basic Formalism

Net systems are difficult to deal with. This reflects the inherent complexity of concurrent systems, where the subtle interactions between conflicts and synchronisations hinder the analysis and synthesis. A common approach to cope with difficult problems is to restrict to easier instances of them. (Linear differential equations with constant coefficients are easier to solve than general ones, yet they are very useful in practice.) A compromise must be found between modelling power and availability of results. Some subclasses of P/T net systems will be introduced in Section 2.3.

On the other hand, the basic P/T formalism is unable to represent certain system behaviours. Enriching this formalism will allow to model a larger class of systems, again at the price of possibly losing some analysis capabilities. In Sections 2.4 and 2.5, we will concentrate on two extensions, net systems with inhibitor arcs and/or priorities, and coloured net systems, respectively.

At this point, it is important to make a distinction between *modelling or expressive power* and *convenience*, in order to better appreciate the relative merits of the different formalisms, restrictions, and extensions. We shall say that a formalism has greater (theoretical) modelling power than anoher when the former can model systems that the latter cannot. For instance, PN have greater modelling power than finite automata since they can model systems with an infinite state space (even with such a simple net as a place where tokens arrive through the firing of a transition and depart through the firing of another one, a single queue-like net system).

When the systems that two formalisms can model are the same, i.e., when they have the same modelling power, the distinction of these formalisms is a matter of convenience, which is not necessarily a mere matter of taste or verbosity. Besides the size and clarity of a model in each formalism, other aspects are important in judging the relative convenience of two formalisms, e.g., the ability to costruct models by refinement and composition, the ability to adequately represent data and control, the analysability, or the degree of parametrization. Assume, for instance, coming back to Example 2.1, that instead of having a fixed system with two computers and a bus, we are designing the PLC. Then, to decide the number of computers (N) and buses ($B \leq N$), we might be interested on some performance indices versus those parameters. In our P/T model the only changes are the initial marking of places p_6 and p_5 and the weight, while the finite automaton (which is isomorphous to the reachability graph) has $\sum_{i=0}^{B} 4^{N-i}$ states, showing up the state space explosion problem. The fact that the structure of the model is not (or little) changed is interesting from a "readability" point of view, but also — and this is perhaps more important — from the perspective of the analysis, which is often based on the structure.

Rating a formalism as more or less convenient than another is not always easy or even possible, since convenience comprises aspects which are typically contradictory, such as compactness and analysability. In some cases, sound and complete (preferably syntactical) transformations exist to translate models from one formalism to another, which can be used to apply analysis techniques developped for a cumbersome formalism to a model produced in an abbreviated formalism, which is then seen as "more convenient" (more compact and identically analysable, leaving other aspects apart). Generally speaking, though, such transformations are impractical and it is highly desirable to develop the direct analysability of abbreviated formalisms, thus increasing their overall convenience.

2.3 Syntactical Subclasses

Subclasses of net systems can be defined either restricting the behaviour or the structure (or syntax) of the model. A possible way of obtaining syntactical subclasses is restricting the inscriptions (e.g., nets with every weight equal to one are ordinary) or the topology, usually aiming at limiting the interplay between conflicts and synchronisations. The latter can be achieved either by giving a general restriction, typically on distributor places and/or join transitions (e.g., there are no joins), or by giving rules to construct models (e.g., sequential functional entities are synchronised by some restricted message passing).

2.3.1 Ordinary Nets and Elementary Net Systems

A P/T net is said to be *ordinary* when every arc weight equals to one. Therefore, the occurrence of a transition consumes one token from each input place, and produces one token for each output place. In modelling terms, if the transition

performs some service and tokens represent clients queueing up in places, serving simultaneosly a bundle of clients, or the simultaneous arrival of a group of clients to a queue cannot be modelled — at least directly.

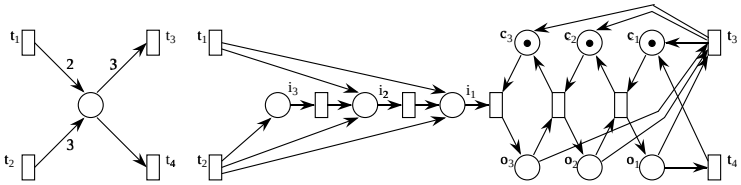


Figure 2.8: Ordinary implementation of a weighted net.

In fact, implementing a weighted P/T net by an ordinary one can be achieved by using a net transformation like the one shown in Figure 2.8. Basically, what the transformation does is collecting the input tokens in a pipe (places i_x) that drives them to place i_1. Then, they are pushed one by one (thanks to places c_x) through the places o_x, ... from which they are collected by the output transitions. Nevertheless, the size of the resulting model grows as the weights do, and it is artificially complex, eventually loosing its appealing clarity. Moreover, although the interleaving semantics is preserved, disregarding the occurrence of the added transitions (i.e., the languages of both systems are the same modulo a projection), if we consider a concurrent semantics, the behaviour is not preserved: After firing t_1 the two tokens in p enable the step $2t_4$, which is never enabled in the ordinary implementation. This is easily solved substituting each output transition by a sequence invisible transition $\rightarrow$ cumulating place $\rightarrow$ output transition, to cumulate the enablings. Doing so, the same steps can occur in both net systems (disregarding the added transitions). But still the concurrent semantics is not fully preserved when there are output arcs from a distributor place having different weights. For instance, in the original model, after the firing of t_2 both t_3 and t_4 become enabled at the same time (the latter with enabling degree three) and a conflict appears, whereas in the ordinary implementation such conflict might be hidden if tokens were directed one by one towards t_4 without ever enabling t_3. These subtle differences will be of some importance when time is incorporated to the model.

In the particular case that the bound of every place is one (i.e., the system is 1-bounded, or safe), places can be interpreted as *boolean conditions*, which hold or not depending on whether they are marked or not. A transition is an event that may occur when its pre-conditions hold. After its occurrence, the pre-conditions become false and the post-conditions are made true. If we want to adhere to this interpretation, we can modify the occurrence rule, so we need not worrying about 1-boundedness: A transition t is enabled when its pre-conditions hold and its post-conditions don't. This is usually known as *Elementary Net (EN) systems*. The idea of taking into account the marking of the output places when defining the enabling of a transition can be extended to general P/T nets by assigning a *capacity* to each place. Nevertheless, capacities are a minor

modelling convenience since they can be simulated using *complementary places* (i.e., the reverse of a given place p marked with the capacity of p minus its current marking, later on we shall show some examples) preserving both the interleaving and concurrent semantics.

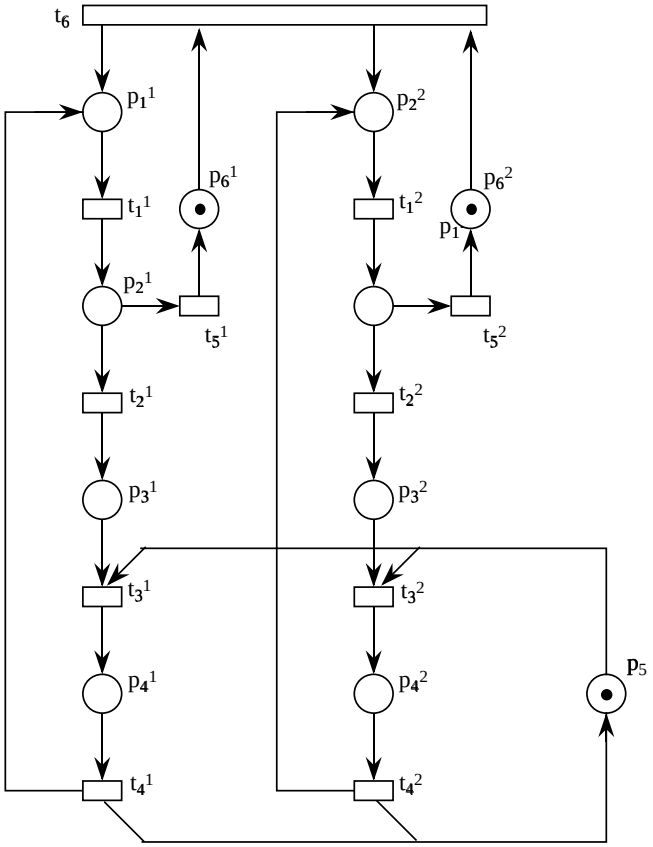


Figure 2.9: An EN model of the multicomputer PLC of Example 2.1.

Let us show how one would model the system in Example 2.1 using an EN system. Now places represent conditions, so we cannot model all the computers in a certain state by putting several tokens in a place. Instead, we model each computer individually as a sequential system (see Figure 2.9). The possible states are modelled by the places: waiting for a new cycle (p_6^i), calculating (p_1^i), deciding (p_2^i), waiting for the bus (p_3^i), and using the bus (p_4^i). The synchronisation of the two computers is represented by the merging of the transitions that model the start of a new cycle in each computer (t_6). The bus forces the computers' states "using the bus" to be mutually exclusive.

2.3.2 Topological Subclasses

Historically, subclasses of ordinary nets have received special attention because powerful results were early obtained for them. In this presentation some of them appear as subclasses of their weighted generalisations for the sake of concision. Regarding the modelling power, clearly some subclasses have less than others if the former are properly included in the latter. Also the weighted generalisations have more modelling power than their ordinary counterparts since, in general, the ordinary implementations of weights do not preserve the (topological) class membership.

Join-free and State Machines

A P/T net $\mathcal{N}$ is *join-free (JF)* when no transition is a join, i.e., $|{}^\bullet t| \leq 1$ for every t. With these nets, proper synchronisations cannot be modelled. $\mathcal{N}$ is a *weighted P-net* when every transition has one input and one output place, i.e., $|{}^\bullet t| = |t^\bullet| = 1$ for every t. An ordinary weighted P-net is a P-net or *state machine (SM)*, a name due to the fact that when marked *with only one token* each place represents a possible global state of the (sequential) system. With more than one token concurrency appears: an SM with k tokens represents k instances of the same sequential process evolving in parallel. Given an adequate stochastic interpretation, strongly connected SM correspond to closed Jackson Queueing Networks.

Distributor-free and Marked Graphs

A P/T net $\mathcal{N}$ is *distributor-free (DF)* when no place is a distributor, i.e., $|p^\bullet| \leq 1$ for every p. With these nets, conflicts cannot be modelled. They are also called *structurally persistent* because the structure enforces persistency, that is, the property that a transition can only be disabled by its own firing. $\mathcal{N}$ is a *weighted T-net* when every place has one input and one output transition, i.e., $|{}^\bullet p| = |p^\bullet| = 1$ for every p. An ordinary weighted T-net is a T-net or *marked graph (MG)*, a name due to a representation as a graph where the nodes are the transitions and the arcs joining them are marked (that is, places have been obviated). As some examples, MG can model activity ordering systems, generalising PERT graphs, job-shop systems with fixed production routing and machine sequencing, flow lines, Kanban systems, etc. For instance, the net in Figure 2.10 (a) is a MG. Given an adequate stochastic interpretation, strongly connected MG correspond to Fork/Join Queueing Networks with Blocking.

Equal Conflict and Free Choice

A P/T net $\mathcal{N}$ is *equal conflict (EQ)* when every pair of transitions in structural conflict are in equal conflict, i.e., they have the same pre-incidence function: ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$ implies $\mathbf{Pre}[P,t] = \mathbf{Pre}[P,t']$. An ordinary EQ net is an *(extended) free choice net (FC)*. Free choice nets play a central role in the theory of net systems because there are powerful results for their analysis and synthesis while

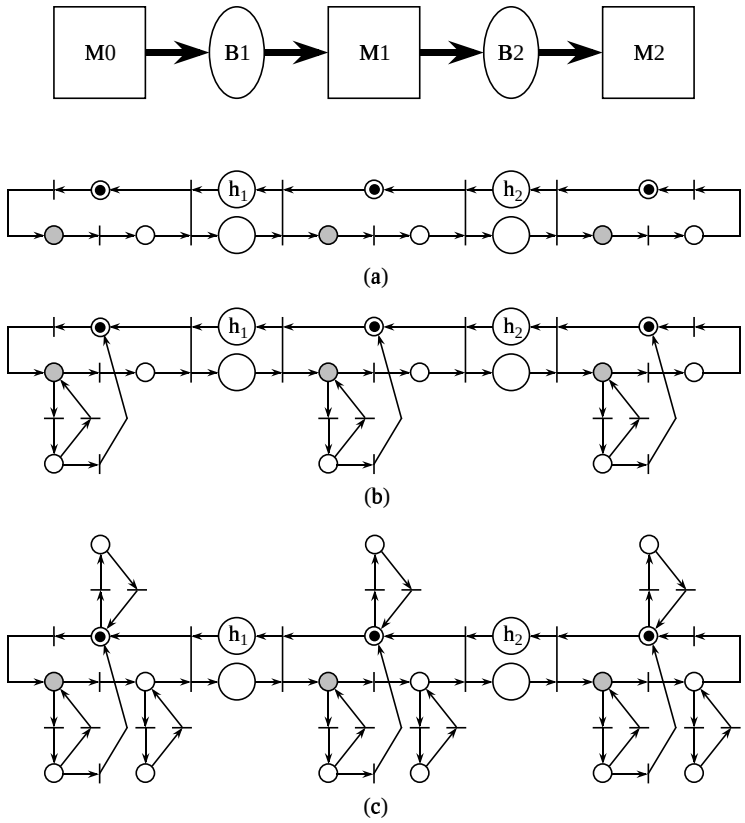


Figure 2.10: Modelling a flow line with three machines and two buffers. Each buffer is modelled by two places, for the parts and "holes", respectively (the later initially marked with h_i holes). Each machine is modelled by a state machine, initially idle, where the "working-state" is shaded; they follow a blocking after service policy (they start their work even if there are no holes in the output buffer, so they might stay blocked before unloading). The different models consider: (a) reliable machines, (b) machines with operation dependent failures (may fail only when working), and (c) machines with time dependent failures (may fail at any time). Scrapping (part is discarded) is possible in the case of unreliable machines.

they allow the modelling of systems allowing both conflicts and synchronisations. It is often said that FC can be seen as MG enriched with SM-like conflicts or, equivalently, SM enriched with MG-like synchronisations. However, they cannot model mutex semaphores or resource sharing, for instance. The nets in Figure 2.6 and Figure 2.10 (b) are FC. The net in Figure 2.1 is EQ. The fundamental property of EQ systems is that whenever a marking enables some transition t, then it enables every transition in $\mathrm{EQS}(t) = \mathrm{CCS}(t)$. It can be said

that the structural and behavioural notions of conflict coincide. It is also said that conflicts and synchronisations are neatly separated, because it is easy to transform the net so that no output of a distributor place is a join: Figure 2.4 (f) is the result of transforming (e).

Asymmetric Choice, or Simple

A P/T net $\mathcal{N}$ is *asymmetric choice (AC)*, sometimes called *Simple*, when it is ordinary and $p^\bullet \cap p'^\bullet \neq \emptyset$ implies $p^\bullet \subseteq p'^\bullet$ or viceversa. In these nets, the conflict relation is transitive. They generalise FC, and allow modelling to a certain extent resource sharing. The nets in Figure 2.9 and Figure 2.10 (c) are AC.

The above subclasses are defined through a global constraint on the topology. Their relations are illustrated in the graph of Figure 2.11, where a directed arrow connecting two subclasses indicates that the source properly includes the destination, and the constructs depicted illustrate the typical situations that distinguish each subclass.

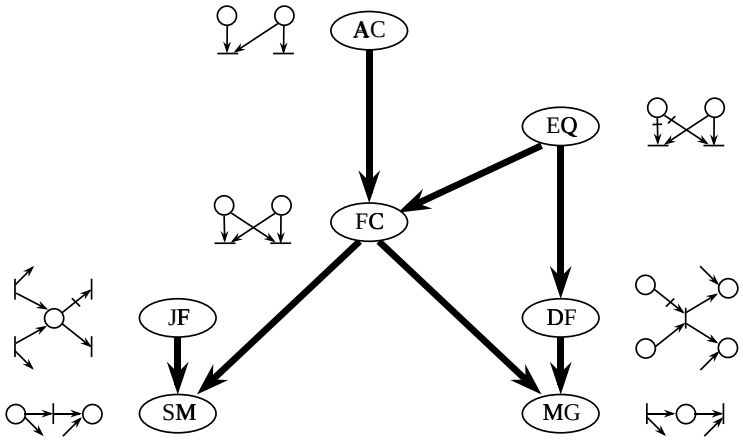


Figure 2.11: Relations between some basic syntactical subclasses.

Modular Subclasses

Subclasses can also be defined in a modular way, by giving some modules and how interconnecting them. Very often the modules are monomarked SM, representing sequential systems which run in parallel communicating in a restricted fashion. A few examples follow.

Superposed automata systems (SA) are composed by monomarked SM synchronised by transition merging, that is, via rendez-vous. They lead to general — although structured — bounded systems models. For instance, duplicating the places p_4^i in the net of Figure 2.9 an SA net is obtained: the four automata are the three computers (for each i: p_1^i = computing, p_1^i = deciding, p_3^i =

waiting bus, p_4^i = using bus, and p_6^i = waiting new cycle) and the bus (p_5 = ready, $p_4^{i\prime}$ = granted to i). The computers synchronise at t_6 to start a cycle, and each computer synchronises with the bus at t_3^i and t_4^i to take and release it, respectively.

Systems of buffer-cooperating functional entities are modules (depending on the kind of modules we obtain different subclasses) synchronised by message-passing through buffers in a restricted fashion. A P/T system $\mathcal{S}$ is in this class when:

- $P = B \uplus \biguplus_i P_i$, $T = \biguplus_i T_i$. The net systems $\mathcal{S}_i$ generated by P_i and T_i are the *functional entities* or modules, and the places of B are the *buffers*.
- For every $b \in B$, there exists i such that $b^\bullet \in T_i$, that is, buffers are output private. Moreover if $t, t' \in T_i$ are in EQ relation in $\mathcal{N}_i$, then $\mathbf{Pre}[b,t] = \mathbf{Pre}[b,t']$, that is, buffers do not modify the EQ relations of the modules. These restrictions on buffers prevent competition.

In case the modules are monomarked SM we obtain *deterministically synchronised sequential processes (DSSP)*. These can be buffer-interconnected again, leading to a hierarchical class of systems, recursively defined, that is called $\{DS\}^*SP$. They allow the modelling of hierarchically coupled cooperating systems. The net systems in Figure 2.10 (a) and (b) can be seen as — rather trivial — examples of buffer-cooperating systems, where the places modelling the buffers are precisely the buffers, while each machine is modelled by an SM.

Systems of Simple Sequential Processes with Resources (S^3PR) are SM synchronised by a restricted resource sharing. The restrictions impose that there is a place in each SM which is contained in every cycle and does not use any resource (an "unavoidable idle state"), and that every other place uses one (possibly shared) resource. They allow the modelling of rather general flexible manufacturing systems, or similar systems where resource sharing is essential.

2.4 Inhibitor Arcs and Priorities

According to their definition, P/T net systems do not allow modelling *zero tests*, that is, transitions that are enabled only if some place is empty. The extensions that we discuss in this subsection will allow the modelling of zero tests, so they clearly increase the theoretical modelling power, actually leading to Turing machines. On the other hand, the extended model is less amenable of analysis. For instance, differently from P/T systems, boundedness is undecidable in systems with inhibitor arcs or priorities [18, 26]. Let us first give the formal definitions.

Definition 2.10 (Inhibitor arcs and priorities)
A P/T net with inhibitor arcs and priorities is a six-tuple:

$$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{Inh}, \mathbf{pri} \rangle$$

where:

1. $\langle P, T, \mathbf{Pre}, \mathbf{Post}\rangle$ *is a P/T net.*
2. $\mathbf{Inh} \in \mathbb{N}^{|P|\times|T|}$ *is the* inhibition *incidence matrix.*
3. $\mathbf{pri} \in \mathbb{N}^{|T|}$ *is the* priority *vector.*

Nets where $\mathbf{Inh} = \mathbf{0}$ *or* $\mathbf{pri} = k \cdot \mathbf{1}$ *have no inhibitor arcs or no priorities, respectively, and the corresponding matrix or vector is not explicited.*

Inhibitor arcs and priorities modify the enabling condition of the occurrence rule: A transition t *is* enabled *at* $\mathbf{m}$ *iff* $\mathbf{m} \geq \mathbf{Pre}[P,t]$, $\mathbf{m} < \mathbf{Inh}[P,t]$, *and there is no* t' *such that* $\mathbf{pri}[t'] > \mathbf{pri}[t]$ *which is enabled. It is often convenient to distinguish whether a transition is not enabled only due to the priorities. We shall say that* t *has* concession *at* $\mathbf{m}$ *when* $\mathbf{m} \geq \mathbf{Pre}[P,t]$ *and* $\mathbf{m} < \mathbf{Inh}[P,t]$. *The enabling degree and the occurrence rule for enabled transitons are as in plain P/T systems. (Naturally, disabled transitions — even if it is due only to inhibitor arcs or priorities — have enabling degree zero.)*

Inhibitor arcs are depicted as circle-headed directed arcs from places to transitions, labelled with the corresponding arc weight (unless it is one). The inhibition set of a transition t is denoted by ${}^\circ t = \{p \mid \mathbf{Inh}[p,t] \neq 0\}$. The inhibited set of a place p is denoted by p°. The lowest priority transitions (conventionally priority zero) are depicted as boxes, while the rest are depicted as bars, labelled with the priority level when greater than one.

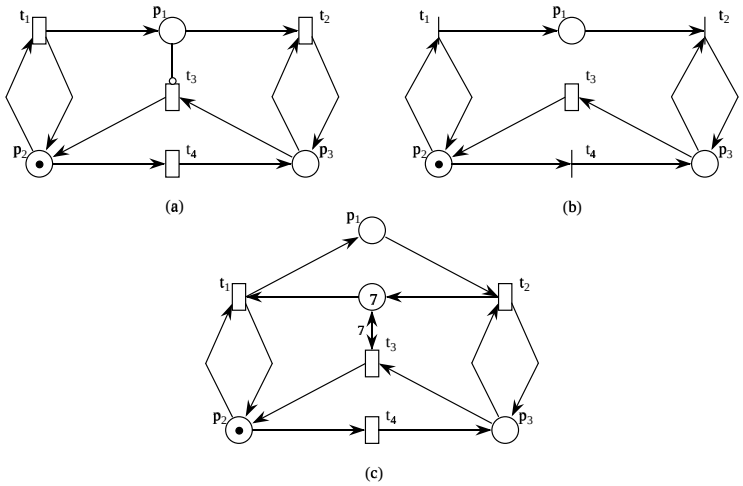


Figure 2.12: Three models of the lazy lad.

Let us illustrate the modelling capabilities of inhibitor arcs and priorities by means of a simple example. Assume we want to model a lazy lad living on his own. When there is nothing to eat he cooks several dishes until he gets desperately hungry and he starts eating. From then on, he eats what he has cooked until the fridge is empty, what makes him switch again to cooking mode.

Figure 2.12 shows two possible models of this behaviour. In (a) the inhibitor arc prevents switching from eating mode (p_3 marked) to cooking mode (p_2 marked) by inhibiting t_3 when the fridge (p_1) is not empty. In (b) the lower priority of the transition from eating to cooking (t_3) prevents its firing unless the fridge (p_1) is empty. The reader is invited to get convinced that plain P/T systems cannot model this behaviour because there is no way to check that the fridge, assumed to have unlimited capacity, is empty.

Inhibitor arcs from bounded places can be implemented (preserving the interleaving semantics) using the complementaries of these places, as shown in the example of Figure 2.12 (c), where we have supposed that at most seven dishes fit in the fridge. This simple transformation does not work so well, though, when concurrent semantics is considered. In the example of Figure 2.13, clearly (b) is an interleaving semantics preserving implementation of (a), but it does not preserve either the enabling degree or the steps. In [4] the complementary place schema is generalised to preserve the (sequentialisable) steps in the case of 1-bounded systems, but, to the best of our knowledge, there is no general technique to implement inhibitor arcs preserving the concurrent semantics, so their convenience in some cases should not be undervalued.

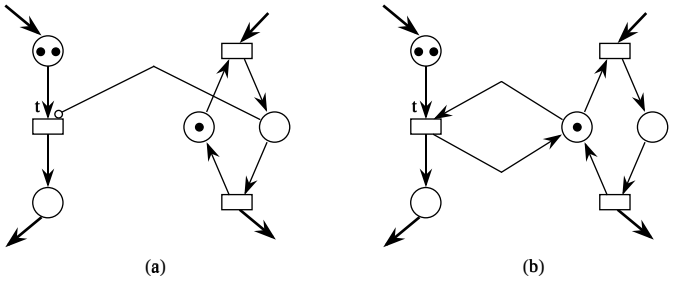


Figure 2.13: Inhibitor arcs and enabling degree. In (a) $\mathbf{e}(\mathbf{m})[t] = 2$, and in (b) $\mathbf{e}(\mathbf{m})[t] = 1$.

It is possible to implement a system with inhibitor arcs using priorities and viceversa, even in the unbounded case and preserving concurrent semantics, so both extensions can be interchanged except for modelling convenience (the transformations are rather cumbersome [9]). Inhibitor arcs have the advantage that they are graphically represented in the net structure, while the influence of a priorities definition on the enabling of some transition is not so clearly reflected, and is not so local. On the other hand, priorities arise naturally when a timing interpretation is considered. Therefore, despite their formal equivalence, both extensions are allowed on equal footing, because they have been introduced to cope with different situations.

Concurrent Semantics of Inhibitor Arcs and Priorities

Let us precise the concurrent semantics of inhibitor arcs and priorities by means of a couple of paradoxical examples. Look at the net systems in Figure 2.14 (a)

and (b), which are independent — somehow similar, though — examples, not the implementation of one another. In both cases, the enabling vector is $t + 2t'$ (this may account for the concurrency of some system activities whose completion was modelled by these transitions). If the occurrence of t and t' took some time, it would be possible to start their firing in parallel, leading to the (maximal) step $t + 2t'$, meaning that there is no conflict. The odd thing about this step is that not all its sequentialisations can occur: in (a) the occurrence of t disables t', while in (b) the occurrence of t' enables the higher priority t'', thus disabling t. This poses the problem that it is not possible to obtain the reachable markings considering individual transition occurrences only, as in plain P/T net systems. For instance, the marking $3p''$ is reachable in (a) by firing $t + 2t'$, and the marking $p + 2p''$ is reachable in (b) by firing $2t'$, markings that cannot be reached by individual transition occurrences, unless we substitute every transition whose firing takes some time by a sequence instantaneous transition → place → instantaneous transition, in order to explicitly manifest that their firing takes time, as depicted in Figure 2.14 (c) and (d). If firings are regarded as instantaneous, such steps make little sense, and *they will be considered as non fireable* (despite Definition 2.6 which was meant for plain P/T systems). Therefore, the step $t+2t'$ in the examples above is not fireable, so the enabling vector is not a fireable step, and then we would say there is a conflict.

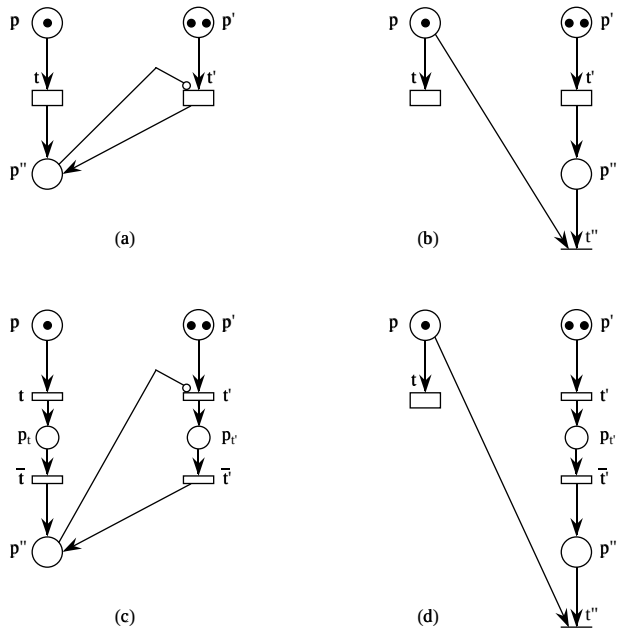


Figure 2.14: Inhibitor arcs, priorities, concurrency, and conflicts.

Generalisation of Inhibitor Arcs: Logical Guards

Following the line of inhibitor arcs, richer extensions can be devised, generalising the kind of condition that "guards" the enabling of a transition. A first step in such direction would be the addition of *test arcs* from places to transitions, which disable the transition if the marking of the place is less than the weight of the arc. Notice that they differ from self-loops such as that in Figure 2.13 (b), because when the transition is enabled the enabling degree does not depend on the marking in the source of the test arc. Clearly, *in the case of bounded systems*, using complementary places it is possible to implement test arcs with inhibitor arcs, and viceversa. Both inhibitor and test arcs guard the enabling with a simple condition on the marking of a place. More general conditions could be allowed (e.g., propositional logic predicates), but perhaps these would put too much information about the behaviour of the system away from the structure (i.e., net) of the model, which is somehow against the rationale of using nets. Since, *in the case of bounded nets*, these more general conditions can be implemented — preserving the concurrent semantics — by inhibitor arcs, after adding some places and possibly replicating some transitions (see Figure 2.15 for an example), they are mainly a matter of (minor) modelling convenience, and we refrain from introducing them formally.

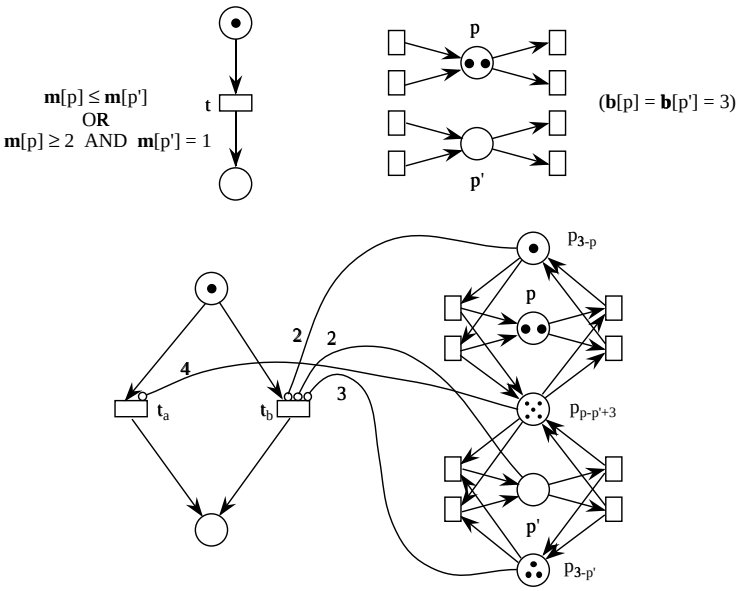


Figure 2.15: Implementing complex enabling conditions in bounded systems.

In Figure 2.15 (a), the enabling of transition t is conditioned by the marking of places p and p' (besides the input place of t): t is enabled when ${}^\bullet t$ is marked and $\mathbf{m}[p] \leq \mathbf{m}[p'] \vee \mathbf{m}[p] \geq 2 \wedge \mathbf{m}[p'] = 1$. It is assumed in this example that the marking bounds of p and p' are three. In Figure 2.15 (b) we duplicated

transition t to represent the disjunction, and used complementary (or similar) places as needed to obtain elementary conditions where the marking of places is upper-bounded (that is, expressible as inhibitor arcs). More precisely:

- In order to represent the disjunction, we duplicate transition t: the instance t_a is enabled when $\mathbf{m}[p] \leq \mathbf{m}[p']$, while the instance t_b is enabled when $\mathbf{m}[p] \geq 2 \wedge \mathbf{m}[p'] = 1$.
- The condition $\mathbf{m}[p] \geq 2$ is checked through the complementary place of p, p_{3-p}. Since $\mathbf{m}[p_{3-p}] = 3 - \mathbf{m}[p]$, $\mathbf{m}[p] \geq 2 \Leftrightarrow \mathbf{m}[p_{3-p}] < 2$.
- The condition $\mathbf{m}[p'] = 1$ is $\mathbf{m}[p'] < 2$ and $\mathbf{m}[p'] \geq 1 \Leftrightarrow \mathbf{m}[p_{3-p'}] < 3$, where $p_{3-p'}$ is the complementary of p'.
- The condition $\mathbf{m}[p] \leq \mathbf{m}[p']$, i.e., $\mathbf{m}[p] - \mathbf{m}[p'] \leq 0$, is checked through place $p_{p-p'+3}$, for which $\mathbf{m}[p_{p-p'+3}] = \mathbf{m}[p] - \mathbf{m}[p'] + 3$ (it is easily obtained generalising the complementary place idea as follows: $\mathbf{C}[p_{p-p'+3}, T] = \mathbf{C}[p, T] - \mathbf{C}[p', T]$, and $\mathbf{m_0}[p_{p-p'+3}] = \mathbf{m_0}[p] - \mathbf{m_0}[p'] + 3$; the addition of three is to force it to have a nonnegative marking). Clearly, $\mathbf{m}[p] - \mathbf{m}[p'] \leq 0 \Leftrightarrow \mathbf{m}[p_{p-p'+3}] < 4$.

2.5 Coloured Nets and Systems

In this section an important extension of PNs is presented. By way of introducing elaborated inscriptions associated with places, transitions, and arcs it allows to write compact models of complex systems. Among the many *High Level Petri Nets* (HLPN) formalisms, we shall restrict ourselves to the Coloured Petri Nets (CPN) formalism [22] and in particular to the subclass of Well-formed Nets (WN) [10].

The new interesting feature introduced by CPNs is the possibility of having distinguished tokens (this explains the adjective *coloured*: the tokens may be represented graphically as dots of different colour instead of being all black dots; actually the "colour" attached to a token may be any kind of information). Each token in CPNs may carry information whose *type* depends on the place where it is located, hence the definition of a CPN must include the specification of a *colour domain* for each place (denoted $cd(p)$, $p \in P$) that is the type of data attached to the tokens in that place.

If the number of possible colours is finite, CPNs have the same theoretical modelling power as P/T net systems, since an algorithmical transformation, called *unfolding*, permits to obtain an equivalent — typically far larger and more cumbersome — P/T model of any given CPN model, as we shall see in Subsection 2.5.3.

Let us immediately use this new feature in an example: assume we are modelling a communication system and we need to represent a (finite) buffer that may contain messages, possibly with different destination. The buffer could be represented by a place, named *buffer*, whose colour domain is the set of all possible destination site identifiers. The number of tokens "coloured" with a

given identifier represents the number of buffered messages for the corresponding destination site. A P/T representation of the same buffer, should comprise as many places as the number of possible destinations, and the number of messages for a given destination would be represented by the number of (black) tokens into the corresponding place. We shall see in Subsection 2.5.3 that this P/T system can be automatically obtained from the CPN representation by *unfolding* it. Observe that if the destination of a message is *irrelevant* in the description of the system behaviour, then the token colour is just *redundant* information and we could represent the buffer with a single place containing as many black tokens as the overall number of messages in the buffer. We shall discuss in more detail the issue of redundant colour information in Subsection 2.5.3.

The state, or marking, in CPNs is represented by the multiset of coloured tokens associated with each place. As in P/T systems the state change is performed by transition firing. Since the tokens in CPNs are distinguished, some additional information is needed to define the coloured tokens that are withdrawn from the input places and put into the output places of a given transition when it fires. This information, associated with each arc, defines a multiset over the colour domain of the place connected to the arc.

Coming back to the example above we may have a transition representing the reception of a message by destination site d_i (named t_rec_{di}) with input place *buffer* and with the corresponding input arc labelled $\{d_i\}$, meaning that t_rec_{di} may fire only if at least one token with associated colour d_i is in place *buffer*, and its firing withdraws a token of colour d_i from its input place. Notice that if there are N destination sites, then we need N transitions to represent the reception of a message by any possible destination. Since the behaviour of these N transitions is the same up to the identity of the receiver, it is convenient to "fold" them into one representative transition (say t_rec), parameterized with the identity of the receiver (let us use variable r to denote the transition parameter). The possible values of the transition parameter(s) define the so called *transition colour domain*. In this case, we need to define the enabling and firing rule of each "instance" of t_rec, denoted $\langle t\_rec, r = d_i \rangle$ or simply $\langle t\_rec, d_i \rangle$ with $d_i \in cd(t\_rec)$. The inscription associated with each arc of this new model is a function of the transition parameter(s) giving a multiset over the colour domain of the corresponding place. In our example, the arc connecting place *buffer* and transition t_rec (with parameter r) has an associated function returning the set $\{r\}$.

Observe that the evolution from P/T nets to CPNs resembles the evolution from the first assembly languages to the more recent typed programming languages.

CPNs with finite colour classes have the same modelling power than P/T nets, because an equivalent P/T net can always be built using the unfolding algorithm, but CPNs are more convenient, not only for their compactness and readability but also for their significantly higher degree of parameterization that can be exploited at the analysis level [21, 20, 19, 33, 11] (see Chapter 7).

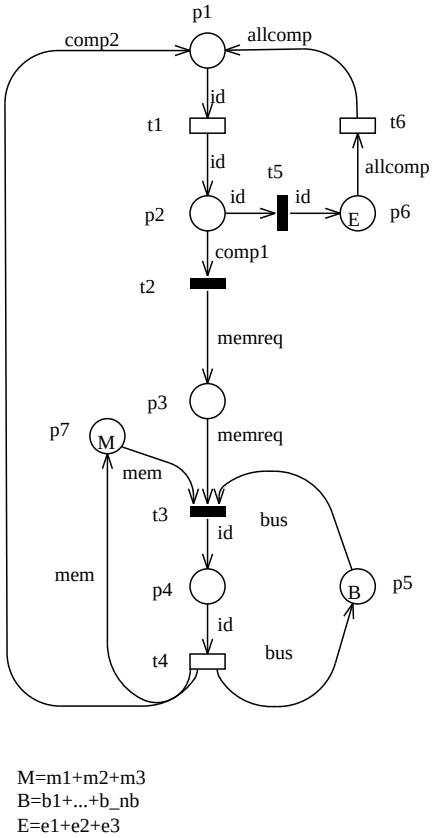


Figure 2.16: CPN model of a multicomputer PLC

2.5.1 Colored Petri Nets Formal Definition

Let us formally define the CPN structure, marking, and dynamics. As for P/T nets, it is possible to give both a graph and matrix oriented definition: we give only the matrix oriented version.

Definition 2.11 (CPN, matrix oriented) *A* Coloured Petri Net *is a six-tuple:*

$$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathcal{C}, cd \rangle$$

where:

1. *P and T are disjoint finite non empty sets (the places and transitions of $\mathcal{N}$),*
2. *$\mathcal{C}$ is the finite set of finite colour classes,*
3. *$cd : P \cup T \to \mathcal{C}$ is a function defining the colour domain of each place and transition,*
4. *$\mathbf{Pre}[p,t], \mathbf{Post}[p,t] : cd(t) \to \mathrm{Bag}(cd(p))$ are the* pre- *and* post- incidence matrices.

If $\forall c \in cd(t), \mathbf{Pre}[p,t](c) = \emptyset$ ($\forall c \in cd(t), \mathbf{Post}[p,t](c) = \emptyset$) then there isn't any arc from place p to transition t (from transition t to place p). The definition of incidence matrix naturally extends to CPNs: the elements of the incidence matrix are functions $\mathbf{C}[p,t] : cd(t) \to Bag(cd(p))$ defined as $\mathbf{C}[p,t] = \mathbf{Post}[p,t] - \mathbf{Pre}[p,t]$.

Remark Observe that some of the places and transitions in a CPN model may be "neutral", that is a place may contain undistinguished tokens (black dots) and a transition may have no parameters (i.e., transitions with only one possible instance). In this case we may define the "neutral" colour class $C_\bullet = \{\bullet\}$ and define the colour domain of any neutral place p (transition t) as $cd(p) = C_\bullet$ ($cd(t) = C_\bullet$).

The functions associated with arcs connecting neutral places/transitions have a simplified definition: if the arc connects a neutral place and a neutral transition the corresponding function may be represented by an integer constant function (the multiplicity in P/T nets); the function associated with an arc connecting a coloured transition t and a neutral place is a function $cd(t) \to \mathbb{N}$, finally the function connecting a neutral transition and a coloured place p is a (constant) multiset over $cd(p)$.

We may extend CPNs with priorities and inhibitor arcs:

Definition 2.12 (CPN with priorities and inhibitor arcs) *A* Coloured Petri Net *with priorities and inhibitor arcs is an eight-tuple:*

$$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{Inh}, \mathbf{pri}, \mathcal{C}, cd \rangle$$

where:

1. *P, T, $\mathcal{C}$, cd,* **Pre** *and* **Post** *are as in Definition 2.11,*

2. $\mathbf{Inh}[p,t] : cd(t) \to \mathrm{Bag}(cd(p))$ *is the matrix defining the inhibitor arcs and associated functions,*

3. $\mathbf{pri}[t] : cd(t) \to \mathbb{N}$ *is a vector that associates with each transition t a function defining the priority of each instance.*

Nets where $\mathbf{Inh}[p,t] = \emptyset$ (i.e., a constant function always returning an empty set) or $\mathbf{pri} = k \cdot \mathbf{1}$ (where $\mathbf{1}$ in this context represents a vector of constant functions always returning 1) have no inhibitor arcs or no priorities, respectively, and the corresponding matrix or vector is not explicited.

Definition 2.13 (CPN marking and system) *The marking of a CPN is a place indexed vector which assigns to each place p a multiset over $cd(p)$: $\mathbf{m}[p] \in Bag(cd(p))$.*

A CPN system is a couple $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ where $\mathcal{N}$ is a CPN and $\mathbf{m_0}$ its initial marking.

Using the vector notation for bags, we may denote $\mathbf{m}[p][c]$ the number of tokens of colour $c \in cd(p)$ in $\mathbf{m}[p]$. Nevertheless, in what follows the abbreviation $\mathbf{m}[p,c]$ will be used instead of $\mathbf{m}[p][c]$.

Pictorially, we write the multiset of coloured tokens in a place using a formal sum notation for bags into the circle representing the place. The initial marking of the system in Figure 2.16 is $\mathbf{m_0} = [\emptyset, \emptyset, \emptyset, \emptyset, b_1 + \ldots + b_{nb}, e_1 + \ldots + e_{nc}, m_1 + \ldots + m_{nc}]$, or in a more convenient formal sum notation $\mathbf{m_0} = p_5(b_1 + \ldots + b_{nb}) + p_6(e_1 + \ldots + e_{nc}) + p_7(m_1 + \ldots + m_{nc})$ (to avoid listing all the empty places).

The evolution of a CPN system is defined through a firing rule (once more we want to stress the fact that in this case the firing concerns a transition instance $\langle t, c \rangle$ rather than a transition).

Definition 2.14 (Enabling and occurrence in CPNs)
The marking in a CPN system evolves as follows:

1. *A transition instance $\langle t, c \rangle$ is said to be* enabled *at a given marking when the multiset of coloured tokens in each input place $p \in {}^\bullet(t)$ contains at least as many tokens of (any) colour $c' \in cd(p)$ as the multiplicity of c' in $\mathbf{Pre}[p,t](c)$. Formally, $\langle t, c \rangle$ is enabled at $\mathbf{m}$ iff $\mathbf{m} \geq \mathbf{Pre}[P,t](c)$.*

   The number of simultaneous enablings of a transition instance $\langle t, c \rangle$ at a given marking $\mathbf{m}$ is called its enabling degree, *and is denoted by $\mathbf{e}(\mathbf{m})[\langle t, c \rangle]$. Formally, $\mathbf{e}(\mathbf{m})[\langle t, c \rangle] = \max\{k \in \mathbb{N}_+ \mid \mathbf{m} \geq k \cdot \mathbf{Pre}[P,t](c)\}$. (The enabling degrees at $\mathbf{m}$ of all the transition instances are collected in the* enabling vector, *$\mathbf{e}(\mathbf{m})$.)*

2. *The* occurrence, *or* firing, *of an enabled transition instance $\langle t, c \rangle$ is an atomic operation that removes from (adds to) each input (output) place the multiset of tokens obtained by applying the function associated with*

the corresponding arc, to the transition colour c. Formally, the occurrence of $\langle t, c\rangle$ *at marking* $\mathbf{m}$*, denoted by* $\mathbf{m}\xrightarrow{\langle t,c\rangle}\mathbf{m}'$*, yields the marking* $\mathbf{m}' = \mathbf{m} + \mathbf{C}[P,t](c)$

3. *Inhibitor arcs and priorities modify the enabling condition and the occurrence rule: a transition instance* $\langle t, c\rangle$ *is enabled at* $\mathbf{m}$ *iff* $\mathbf{m} \geq \mathbf{Pre}[P,t](c)$, $\forall p \in P, \forall c' \in \mathbf{Inh}[p,t](c), m[p,c'] \leq (\mathbf{Inh}[p,t](c))[c']$, *and there is no transition instance* $\langle t', c'\rangle$ *such that* $\mathbf{pri}[t'](c') > \mathbf{pri}[t](c)$ *which is enabled.*

   The concepts of concession *and* enabling degree *introduced in Definition 2.10 for P/T with inhibitor arcs and priorities naturally extend to the CPN formalism.*

Remark Observe that in CPNs we may have different instances of the same transition that are concurrent, or different instances of the same transition that are in conflict with each other (e.g., because they need the same coloured tokens from a shared input place). Moreover, a given transition instance might be multiply enabled (self-concurrency).

Next we revisit the multicomputer PLC example. Taking advantage of the compactness of CPN models, we shall present in fact an extension of Example 2.1, with a net of comparable size. The aim of this example is twofold: on one hand we want to show how complex information can be included in a very compact form, on the other hand we want to highlight the fact that a model abstracting out the inessential details, is preferable than a model containing useless information both from the point of view of the model comprehension and correctness, and from the point of view of the analysis complexity: this latter issue will be discussed in Subsection 2.5.3.

Example 2.2 (Multicomputer PLC revisited)

Assume that instead of three computers and one bus, nb busses are available to serve the remote memory accesses of nc computers. Only one external access to the same memory can be served at a time, while a local and an external memory access can occur concurrently, since the memory of each computer in the system is dual-port.

We use colours to distinguish tokens representing different computers, memories and busses and to distinguish tokens representing different memory requests and accesses. The CPN model of the multi-computer PLC is depicted in Figure 2.16. Places and transitions have essentially the same meaning as those in the P/T example. Place p_7 has been added representing the memories that are not involved in any external access.

Another important difference between this model and the PN model of Fig. 2.1 is the additional priority structure: all transitions have priority[1] set to 0 except t_2, t_3 and t_5: $\forall c, \mathbf{pri}[t_2](c) = 1$, $\forall c, \mathbf{pri}[t_5](c) = 1$, $\forall c, \mathbf{pri}[t_3](c) = 2$.

[1]Transitions with priority 0 are pictorially represented as white boxes, while transitions with priority greater than 0 are represented as black thin bars.

The rationale for defining priorities in this way is the following: transitions t_1, t_4 and t_6 represent operations that take time while transitions t_2, t_5 and t_3 represent logical actions that take a negligible amount of time to complete: hence once a process has finished its local computation it immediately decides whether to perform a memory request or to end its computation, moreover, if it decides for a memory access and the needed resources are available, the access immediately starts. Therefore a firing sequence as t_1, t_2, t_3 or t_1, t_5 appears as an atomic action from the point of view of any other transition whose priority is set to 0.

The colour domains of the places and the transitions are defined hereafter:

- **Places:**
  - $cd(p_1) = cd(p_2) = cd(p_6) = \{e_i, i = 1, \ldots, nc\}$: the set of computers;
  - $cd(p_3) = \{e_i\_req\_m_j, i, j = 1, \ldots, nc\}$: the set of possible memory access requests,
  - $cd(p_4) = \{e_i\_acc\_m_j\_via\_b_k, i, j = 1, \ldots, nc, k = 1, \ldots, nb\}$: the set of possible memory accesses;
  - $cd(p_5) = \{b_i, i = 1, \ldots, nb\}$: the set of busses;
  - $cd(p_7) = \{m_i, i = 1, \ldots, nc\}$: the set of memories;
- **Transitions:**
  - $cd(t_1) = \{e_i, i = 1, \ldots, nc\}$: the possible local computation activity instances;
  - $cd(t_2) = \{e_i\_req\_m_j, i, j = 1, \ldots, nc, i \neq j\}$: the possible memory access request instances;
  - $cd(t3) = cd(t4) = \{e_i\_req\_m_j\_via\_b_k, i, j = 1, \ldots, nc, i \neq j, k = 1, \ldots, nb\}$: the possible memory access instances;
  - $cd(t_5) = \{e_i, i = 1, \ldots, nc\}$: the possible instances of an "end control cycle";
  - $cd(t_6) = \{\bullet\}$: the (unique) possible instance of a synchronization among all the computers composing the PLC.

The arc functions are defined as follows:

- id is the identity function;
- $comp1 : cd(t_2) \rightarrow Bag(cd(p_2))$, is defined as $comp1(e_i\_req\_m_j) = \{e_i\}$;
- $comp2 : cd(t_4) \rightarrow Bag(cd(p_1))$, is defined as $comp2(e_i\_req\_m_j\_via\_b_k) = \{e_i\}$;
- $memreq : cd(t_3) \rightarrow Bag(cd(p_3))$, is defined as $memreq(e_i\_req\_m_j\_via\_b_k) = \{e_i\_req\_m_j\}$;

- $mem : cd(t_4) \rightarrow Bag(cd(p_7))(= cd(t_3) \rightarrow Bag(cd(p_7)))$, is defined as $mem(e_i\_req\_m_j\_via\_b_k) = \{m_j\}$;
- $bus : cd(t_4) \rightarrow Bag(cd(p_5))(= cd(t_3) \rightarrow Bag(cd(p_5)))$, is defined as $bus(e_i\_req\_m_j\_via\_b_k) = \{b_k\}$;
- $allcomp : cd(t_6) \rightarrow Bag(cd(p_6))(= cd(t_6) \rightarrow Bag(cd(p_1)))$, is a constant function defined as $allcomp = \{e_i, i = 1, \ldots, nc\}$.

To better understand the enabling and firing rules for CPNs let us play the token game on this example. The initial marking comprises the set $E = \{e_i, i = 1, \ldots, nc\}$ of computers in place p_6, meaning that initially all the computers are ready to synchronize for starting a control cycle, the set $B = \{b_i, i = 1, \ldots, nb\}$ of busses in place p_5, meaning that all busses are initially idle, and the set $M = \{m_i, i = 1, \ldots, nc\}$ of all memories in place p_7 meaning that there are no memories initially involved in a remote access. The unique transition initially enabled is t_6. After firing t_6 the whole set E disappears from p_6 and appears in p_1. In this new marking, nc transition instances are enabled, namely $\langle t_1, e_i\rangle, i = 1, \ldots, nc$, meaning that each computer e_i may now perform a local computation (firing of $\langle t_1, e_i\rangle$): observe that the instances $\langle t_1, e_i\rangle$ and $\langle t_1, e_j\rangle, j \neq i$ are not in conflict with each other, because they withdraw tokens of different colour from the common input place p_1. Indeed the arc function id appearing on the input arc of t_1 returns a set containing a single token whose colour is the same as the transition colour instance.

Let us assume that $\langle t_1, e_2\rangle$ fires, then the token of colour e_2 is withdrawn from place p_1, and put into place p_2. In this new state there are three transitions that have some instance with concession, namely t_1, t_2 and t_5, but only the following nc instances $\langle t_2, e_2\_req\_m_j\rangle, j = 1, 3, \ldots, nc$ and $\langle t_5, e_2\rangle$ are actually enabled, because of their higher priority. Observe that all these transition instances are in conflict one with the other because they all need the token of colour e_2 in place p_2. Let us assume that $\langle t_2, e_2\_req\_m_1\rangle$ fires, meaning that computer e_2 wants to access memory m_1: function $comp1$ in this case returns a single token of colour e_2, that is withdrawn from p_2, while the id function on the output arc causes a token of the same colour as the transition instance to be put into place p_3. Again, in the new marking only the instances of the higher priority transition t_3 are enabled (i.e., $\langle t_3, e_2\_req\_m_1\_via\_b_k\rangle, k = 1, \ldots, nb$). The firing of transition instance $\langle t_3, e_2\_req\_m_1\_via\_b_1\rangle$ withdraws a token of colour $e_2\_req\_m_1$ from place p_3 (see definition of function $memreq$), a token of colour m_1 from p_7 and a token of colour b_1 from p_5 (see definition of functions mem and bus) and puts a token of colour $e_2\_req\_m_1\_via\_b_1$ into place p_4 (because of the id function on the output arc).

Summarizing, the main differences between this model and its P/T counterpart are the following: we have refined the P/T model to take into account the identity of the computers composing the system, and in particular we have used this identity to check that only one remote access at a time is allowed on the same memory. Observe however, that concurrent remote accesses to the same memory are possible only if more than one bus is available (an hypothesis

that is true only in the CPN example). We have also added a priority structure that allows to distinguish between transitions representing time consuming activities and transitions representing logical actions that can be completed into a negligible amount of time.

2.5.2 Well-formed Nets

Well-formed Nets (WN) are a subclass of CPNs whose peculiarity is a very structured syntax for the definition of the place and transition colour domains and of the arc functions. The motivation for such syntax is the POSSibility of automatically exploiting the intrinsic symmetries of the model to efficiently generate an "aggregated" reachability graph (the algorithm will be presented in Chapter 7).

The starting point in the structured definition of the WN colour syntax is the set of *basic colour classes* $\{C_1, \ldots, C_n\}$. A basic colour class C_i is a nonempty, finite (possibly circularly *ordered*) set of colours; intuitively, a basic colour class can be defined as a set of colours identifying objects of the same nature. A basic colour class is ordered if a *successor function* is defined on its elements, such that it induces a circular ordering on the class elements. Examples are the class of processors, the class of memories, the class of busses, etc. An example of ordered class is the class of processors connected in a ring topology. Basic colour classes are disjoint (i.e., $\forall i, j : i \neq j, C_i \cap C_j = \emptyset$), moreover, a class may be partitioned into several *static subclasses* ($C_i = C_{i,1} \cup \ldots \cup C_{i,n_i} \forall j, k : j \neq k, C_{i,j} \cap C_{i,k} = \emptyset$): colours belonging to different static subclasses represent objects of the same type but with different behaviour, for example the basic colour class of processors could be partitioned into two (disjoint) static subclasses, one containing the *fast* processors and the other containing the *slow* ones. We denote $\tilde{C}_i$ the set of static subclasses of C_i.

The place colour domains, are defined by composition through the Cartesian product operator of basic colour classes. The colour domain of a place is similar to a C-language structure declaration, i.e., the information associated with tokens comprises one or more *fields*, each field in turn has a type selected from the set of basic colour classes $\{C_1, \ldots, C_n\}$. The identification of the fields is positional (there is no name associated with a field).

With reference to the multicomputer PLC example, we may have two basic colour classes: the class E of computers, and the class B of busses. We do not define the class M of memories, instead we use the same colours in class E to identify memories (since there is a one-to-one correspondence among computers and memories): as we shall see in a moment, this is needed to be able to define the arc functions. In this new setting, let us give the new colour domain definition for the places of the multicomputer PLC model:

- $cd(p_1) = cd(p_2) = cd(p_6) = cd(p_7) = E$;
- $cd(p_3) = E \times E$.
- $cd(p_4) = E \times E \times B$;

- $cd(p_5) = B$;

The transition colour domains, are used to define the *parameters* of transitions and their type; each parameter has a type selected from the basic colour classes, moreover restrictions can be defined on the possible colour instances of a transition (i.e., on the possible values assigned to parameters) by means of a *transition predicate*, or *guard*. Therefore, the definition of a transition colour domain comprises two parts: a list of typed parameters, and the *guard*, defined as a Boolean expression of (a restricted set of) basic predicates on the parameters. Each parameter is associated with a *variable* appearing in the arc functions of the input, output and inhibitor arcs of the transition[2]. We shall denote $var_i(t)$ the subset of transition t parameters of type C_i, and $var(t)$ the whole set of transition t parameters.

Definition 2.15 (Standard Predicates) *A standard predicate (or guard) associated with a transition t is a boolean expression of* basic predicates. *The allowed basic predicates are: $x = y$, $x = !y$, $d(x) = C_{i,j}$, $d(x) = d(y)$, where $x, y \in var_i(t)$ are parameters of t of the same type, $!y$ denotes the successor of y (assuming that the type of y is an ordered class), and $d(x)$ denotes the static subclass x belongs to.*

Here follows the new colour domain definition for the transitions in our running example (the colour domain is defined as a pair ⟨transition parameters type, guard⟩):

- $cd(t_1) = cd(t_5) = \langle \langle x \rangle \in E, true \rangle$;
- $cd(t_2) = \langle \langle x, y \rangle \in E \times E, x \neq y \rangle$;
- $cd(t_3) = cd(t_4) = \langle \langle x, y, z \rangle \in E \times E \times B, true \rangle$;
- $cd(t_6) = \langle \langle \bullet \rangle, true \rangle$.

Observe that the definition of $cd(t_2)$ explains why we had decided to use the same class to represent both the memories and the computers: the guard $x \neq y$ makes sense only if x and y are parameters of the same type. If we had defined two basic classes, E and M, and if y in t_2 had type M instead of E, then the guard of t_2 should have been something like $((x = e_1) \wedge ((y = m2) \vee \ldots \vee (y = m_{nc}))) \vee ((x = e_2) \wedge ((y = m1) \vee (y = m3) \vee \ldots \vee (y = m_{nc}))) \vee \ldots \vee ((x = e_{nc}) \wedge ((y = m1) \vee \ldots \vee (y = m_{nc-1})))$ but this guard is made up by boolean composition of basic predicates not allowed by the WN formalism. The only way of implementing these predicates in WNs, is to partition both E and M in nc static subclasses, $E^i, i = 1, \ldots, nc$ and $M^j, j = 1, \ldots, nc$, each containing only one element, and rewrite the basic predicate $x = e_i$ $(y = m_j)$ as $d(x) =$

[2] Observe that the *scope* of a variable appearing on a given arc is the corresponding transition: instances of the same variable appearing in arcs of the same transition actually represent the same parameter, while different instances of the same variable associated with different transitions are independent.

E^i $(d(y) = M^j)$. As we shall see in Chapter 7, by so doing we destroy the symmetries of the model and prevent the state aggregation.

The arc functions are defined as weighted (and possibly guarded) sums of tuples, the elements composing the tuples are in turn weighted sums of *basic functions*, defined on basic colour classes and returning multisets of colours in the same class. Given this definition, it is more appropriate to refer to the arc inscriptions as *arc expressions* instead of arc functions.

Definition 2.16 (Arc expressions) *An arc expression associated with an arc connecting place p and transition t has the following form:*

$$\sum_k \delta_k.[pred_k]F_k$$

where δ_k is a positive integer, F_k is a function and $[pred_k]$ is a standard predicate. The value of function "$[pred]f$" is given by:

$$[pred]f(c) \quad = \quad If\ pred(c)\ then\ f(c)\ else\ 0.$$

Each $F_k : cd(t) \rightarrow Bag(cd(p))$ is a function of the form

$$F = \bigotimes_{C_i \in\ \mathcal{C}} \bigotimes_{j=1,\ldots,e_i} f_i^j = \langle f_1^1, \ldots, f_1^{e_1}, \ldots, f_n^1, \ldots, f_n^{e_n} \rangle$$

with e_i representing the number of occurrences of class C_i in colour domain of place p, i.e., $cd(p) = \bigotimes_{C_i \in \mathcal{C}} \bigotimes_{j=1,\ldots,e_i} C_i = \bigotimes_{C_i \in \mathcal{C}} C_i^{e_i}$.

Each function f_i^j in turn is defined as:

$$f_i = \sum_{q=1}^{n_i} \alpha_{i,q}.S_{C_{i,q}} + \sum_{x \in var_i(t)} (\beta_x.x + \gamma_x.!x)$$

where $S_{C_{i,q}}$, x and $!x$ are basic functions (defined hereafter), $\alpha_{i,q}$, β_x and γ_x are natural numbers.

The multiset returned by a tuple of basic functions is obtained by Cartesian product composition of the multisets returned by the tuple elements. As it can be observed in the formal definition of arc expressions, there are three types of basic functions: the *projection* function, the *successor* function and the *diffusion/synchronization* function. The syntax used for the projection function is x, where x is one of the transition variables (i.e., one of the transition parameters: it is called projection because it selects one element from the tuple of parameter values defining the transition colour instance), the syntax used for the successor function is $!x$ where x is again one of the transition variables, it applies only to ordered classes and returns the successor of the colour assigned to x in the transition colour instance. Finally, the syntax for the diffusion/synchronization function is S_{C_i} (or $S_{C_{i,j}}$): it is a constant function that returns the whole set of colours of class C_i (of static subclass $C_{i,j} \subset C_i$). It is called synchronization

when used on a transition input arc because it implements a synchronization among a set of coloured tokens contained into a place, while it is called diffusion when used on a transition output arc because it puts several tokens of different colour into a place.

Let us define some arc expression for the multicomputer PLC example: some trivial examples are function $allcomp = \langle S_E \rangle$, or the identity function on the input and output arcs of transition t_5 $id = \langle x \rangle$. Let us consider the arc expression on the input/output arcs of some transitions:

t_2: $comp1 = \langle x \rangle$, $id = \langle x, y \rangle$;

t_3: $memreq = \langle x, y \rangle$, $mem = \langle y \rangle$, $bus = \langle z \rangle$, $id = \langle x, y, z \rangle$;

t_4: $id = \langle x, y, z \rangle$, $mem = \langle y \rangle$, $bus = \langle z \rangle$, $comp2 = \langle x \rangle$.

Observe that the structure of the arc expressions (i.e., the position of the basic functions in a tuple) must be consistent with the position of the "fields" in the colour domain of the corresponding place, i.e., the type of the multiset returned by the j-th element in the arc expression is equal to the type of the j-th field in the corresponding place colour domain.

All basic ingredients have now been introduced: let us formally define WNs.

Definition 2.17 (Well-formed Nets) *A* Well-formed net *is an eight-tuple:*

$$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{Inh}, \mathbf{pri}, \mathcal{C}, cd \rangle$$

where:

1. *P and T are disjoint finite non empty sets (the places and transitions of $\mathcal{N}$),*

2. *$\mathcal{C} = \{C_1, \ldots, C_n\}$ is the finite set of finite* basic *colour classes, (we use the convention that classes with index up to h are not ordered, while classes with higher index are ordered),*

3. *cd is a function defining the colour domain of each place and transition; for places it is expressed as Cartesian product of basic colour classes (repetitions of the same class are allowed), for transitions it is expressed as a pair $\langle$ variable types, guard $\rangle$ defining the possible values that can be assigned to transition variables in a transition instance; guards must be expressed in the form of* standard predicates,

4. *$\mathbf{Pre}[p,t], \mathbf{Post}[p,t] : cd(t) \rightarrow \mathrm{Bag}(cd(p))$ are the* pre- *and* post- incidence matrices, *expressed in the form of* arc expressions,

5. *$\mathbf{Inh}[p,t] : cd(t) \rightarrow \mathrm{Bag}(cd(p))$ is the matrix defining the inhibitor arcs and associated arc expressions,*

6. $\mathbf{pri}[t] : cd(t) \rightarrow \mathbb{N}$ *is a vector that associates with each transition* t *a function defining the priority of each instance; there is a restriction on the priority functions: two instances of a given transition may be assigned different priorities only if there exists a standard predicate capable of distinguishing the two. In other words the specification of* $\mathbf{pri}[t]$ *could be given as:*
$\mathbf{pri}[t] : case$
$\Phi_1 : n_1$
$\Phi_2 : n_2$
...
$\Phi_k : n_k$
$default : n_{default}$

Since WNs are a subclass of CPNs, it is possible to define both the marking and dynamics of WNs as in CPNs. The initial marking of a WN is thus a place indexed vector which assigns to each place p a multiset over $cd(p)$: $\mathbf{m}[p] \in Bag(cd(p))$. For reasons that will become clear in a later chapter, it is useful to define a *symmetric* initial marking of a WN model:

Definition 2.18 (Symmetric Marking of a WN) *A marking* $\mathbf{m}$ *of a WN model is symmetric if it can be expressed as follows:*

$$\forall p \in P, \mathbf{m}[p] = \sum_{\tilde{c} \in \bigotimes_{C_i \in C} \tilde{C_i}^{e_i}} \alpha_{\tilde{c}}\ \tilde{c}$$

where $\tilde{c}$ *is a tuple of static subclasses (consistent with the colour domain of the corresponding place), and* $\alpha_{\tilde{c}} \in \mathbb{N}$ *is the coefficient of tuple* $\tilde{c}$*. As usual a tuple* $\langle A_1, \ldots, A_k \rangle$ *of sets represents the set of tuples obtained by composing the sets through the Cartesian product operator:*

$$\langle A_1, \ldots, A_k \rangle := \bigotimes_{i=1,\ldots,k} A_i$$

In summary the WN formalism poses more constraints on the modeller than the CPN formalism, so the more permissive CPN formalism appears to be more convenient at first sight. Nevertheless, in most cases the structured WN colour definition is natural and leads to "cleaner" models. Moreover, the availability of specific analysis algorithms (that will be presented in Chapter 7) that rely on the *symmetry properties* of WNs arc functions, transition predicates, and priority functions, represents a crucial advantage that supports the high convenience of the WN formalism.

2.5.3 Unfolding, Folding, and Decolouring: Choosing the Appropriate Detail Level

Given a CPN with finite colour domains, it is always possible to derive an equivalent P/T net by applying an *unfolding* algorithm. While the unfolding

of a CPN is unique, the inverse operation of *folding* a P/T net to obtain a more compact, coloured representation of the same model, may lead to several alternative CPN models, depending on the point of view of the folding and the desired degree of "compactation". In the extreme case, we may fold all the places into one place, whose marking would encode the whole system state, and all the transitions into one transition; in this case the arc functions on the arcs connecting the unique place and the unique transition would embed all the information on the system dynamics, resulting in the loss of any "visual" information. In this sense, a more restrictive formalism, can force the modeller to avoid hiding too much information in the arc functions (e.g., very complex functions are hardly expressed by composing simple basic functions as those allowed by the WN formalism),

Another important issue concerns the possibility of introducing *redundant* colour information in a CPN model: the possibility of including a lot of information in the model with very little effort, makes this eventuality more frequent than in P/T nets. The structured nature of the WN formalism, makes it relatively easy to automatically discover and remove some redundant colour information, helping the modeler in identifying the relevant details of the modeled system, and reducing the complexity of the model analysis. Later in this section we shall discuss how the CPN model of the multicomputer PLC example can be decoloured.

Definition 2.19 (Unfolding of a CPN) *The P/T net* $\mathcal{N}_{P/T} = \langle P', T', \mathbf{Pre}', \mathbf{Post}', \mathbf{Inh}', \mathbf{pri}' \rangle$ *resulting from the unfolding of a CPN* $\mathcal{N}_{CPN} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{Inh}, \mathbf{pri}, \mathcal{C}, cd \rangle$ *is defined as follows:*

- $P' = \{\langle p, c \rangle, p \in P, c \in cd(p)\}$
- $T' = \{\langle t, c \rangle, t \in T, c \in cd(t)\}$
- $\mathbf{Pre}'[\langle p, c \rangle, \langle t, c' \rangle] = (\mathbf{Pre}[p, t](c'))[c]$
- $\mathbf{Post}'[\langle p, c \rangle, \langle t, c' \rangle] = (\mathbf{Post}[p, t](c'))[c]$
- $\mathbf{Inh}'[\langle p, c \rangle, \langle t, c' \rangle] = (\mathbf{Inh}[p, t](c'))[c]$
- $\mathbf{pri}'[\langle t, c \rangle] = \mathbf{pri}[t](c)$

The initial marking $\mathbf{m_0}_{P/T}$ *of the unfolded P/T net is defined as follows:* $\mathbf{m_0}_{P/T}[\langle p, c \rangle] = \mathbf{m_0}_{CPN}[p, c]$.

Basically, the unfolding consists of replicating each place and each transition as many times as the cardinality of the corresponding colour domain, and by including an arc of weight w from place $\langle p, c \rangle$ to $\langle t, c' \rangle$ iff $\mathbf{Pre}[p, t](c')[c] = w$ (similarly for functions **Post** and **Inh**). There is a clear correspondence between a marking in the CPN and that of its unfolding: place $\langle p, c \rangle$ contains n tokens in marking $\mathbf{m}$ iff $\mathbf{m}[p, c] = n$.

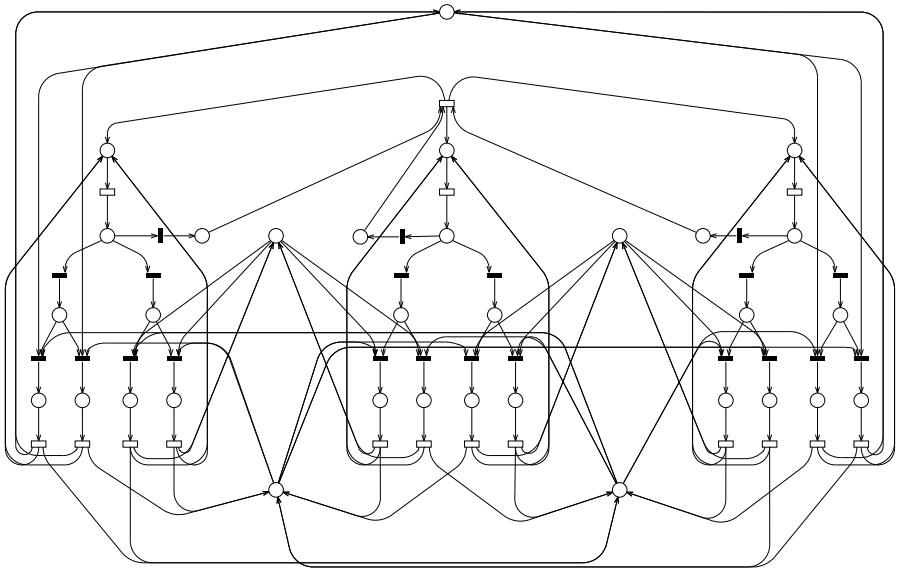


Figure 2.17: Unfolded model of a multicomputer PLC with three computers and two busses

Figure 2.17 shows the unfolding of the multicomputer PLC CPN assuming three computers and two busses. Comparing this net with its coloured counterpart it can be observed that in the CPN only the subnets with some similarity have been folded resulting in a CPN model with very simple arc functions, i.e., in a model that is parametric, has the same amount of visual information as its uncoloured counterpart, and is more readable (especially when the number of computers and busses in the system grows). It is possible to recognize the three copies of the place representing the memories, the two copies of the (place - transition - place) subnet representing the busses not being used in an external access, and three copies of the submodel of the computers behaviour, that in turn has some replicated parts to represent the possible alternative choices in issuing a memory request, and the possible alternative choices in acquiring a bus to perform an access. Needless to say, this model can become huge and difficult to read as the number of computers and busses increases, while the CPN model structure does not change.

Let us study the possible presence of redundancy in the colour specification of the CPN model of the multicomputer PLC: this discussion is very informal, it will be formalized later. The first observation concerns the colour class used for the busses: this colour is never actually used to influence the behaviour of the system (a memory access can start if any bus is available), so that the token rapresenting busses, could be "decoloured" (i.e., made all black tokens) without affecting the possible "event sequences" that can be observed by playing the

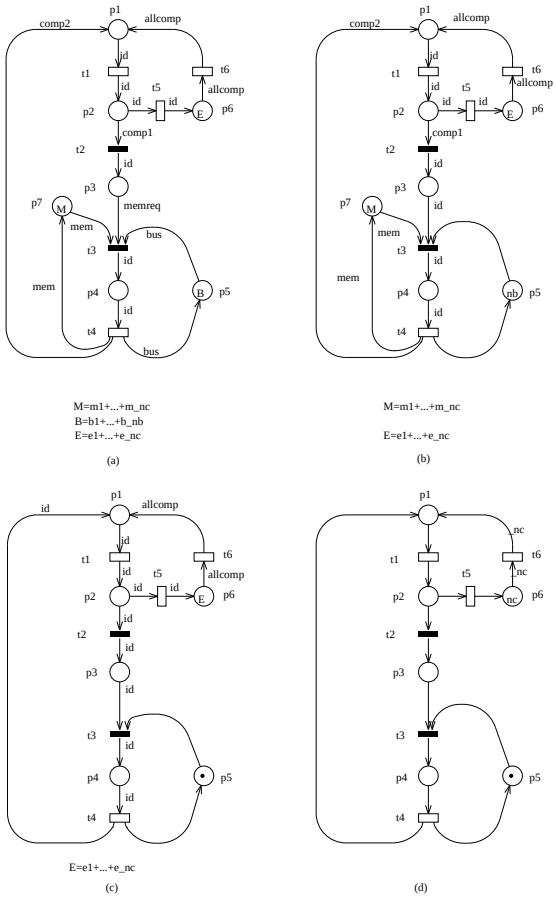
token game on the net (see Figure 2.18(b)). The second observation concerns the colour used to distinguish memories, in the particular case of $nb = 1$ (only one bus in the system): indeed in this situation, the place representing the memories not involved in any external access becomes *implicit* (i.e., its removal does not change the behaviour at all), and can therefore be removed. After this transformation the information of the identity of the memory requested for an external access becomes redundant (after place p_7 is removed, the synchronization on the memory colour in transition t_3 disappears, hence the memory identity information doesn't affect the possible event sequences) and can therefore be canceled from the net (see Figure 2.18(c)). The new CPN model after this two decolouring steps, has only one basic colour class, E, and the only transition that actually uses the colour information to fire is t_6, since it can fire only if the set E is present in p_6. Observe however, that due to the conservativity property of the net, there can never be more than one token of a given colour e_i in any place, so that the enabling condition of t_6 could be simply changed into a check for the presence of $|E| = nc$ tokens in p_6. This last decolourization step leads us back to a P/T model (see Figure 2.18(d)).

As we shall see in Chapter 7 this type of redundant colour simplification can be algorithmically defined, and hence automatized, for WN models.

2.6 Bibliographical Remarks

Petri nets were introduced in the Ph.D. thesis of C.A. Petri [27]. Today it is a very rich but relatively young field having impact on many different industrial sectors. More than in seminal/historical papers we mainly (but not only) refer to books, tutorials or surveys, where the specialised contributions are explicitely pointed. A bibliography on Petri nets is periodically gathered by the Gesellschaft für Informatik, Bonn University, Germany, and published in the Petri Net Newsletter. This bibliography and other informations on Petri nets can be accessed electronically via the Petri nets WWW page (`http://www.daimi.aau.dk/PetriNets/`), a service supplied by the Datalogisk Afdeling I Matematisk Institut (DAIMI), Aarhus University, Denmark. Introductory texts to Petri nets and their applications are [26, 5, 30, 31, 22, 1]. [25, 32, 13] are surveys. The material of two advanced courses on Petri nets is collected in [6, 7, 8]. The International Conference (formerly European Workshop) on Application and Theory of Petri Nets takes place every year, since 1980. Since 1992, the proceedings are published within Springer's Lecture Notes in Computer Science (LNCS). Before, selected papers appeared in Advances in Petri Nets, a subseries of LNCS edited by G. Rozenberg. Focused on High-Level Petri Nets, [23] is a selection of papers.

Some net models (including P/T and EN) and subclasses are surveyed in [3]. The basic subclasses (SM, MG, FC, and AC) and some qualitative analytical results can be found in several of the introductory texts, and also in the more specialised book [15]. Some net models (including P/T and EN) and subclasses are surveyed in [3]. The basic subclasses (SM, MG, FC, and AC) and some



(b) Changes in the colour definition:

- $cd(p_4) = E \times E$;
- $cd(t_3) = cd(t_4) = \langle\langle x, y\rangle \in E \times E\rangle$;

(c) Changes in the colour definition:

- $cd(p_4) = E$;
- $cd(t_3) = cd(t_4) = \langle\langle x\rangle \in E, true\rangle$;
- $id = \langle x\rangle$ on all arcs.

Figure 2.18: Removing redundant colours when $nb = 1$

qualitative analytical results can be found in several of the introductory texts, and also in the more specialised book [15]. Other subclasses are studied in research papers [35, 36, 14, 29, 34, 28, 16].

Other concepts of priorities have been proposed in order to model systems in which a local priority specification is introduced to compare only some pairs of transitions [4]. The two concepts of local and global priority have equivalent modelling power, in the sense that each one can simulate the other even considering a concurrent semantics [9]. However, the two mechanisms have different levels of modelling convenience, depending on the nature of the modelled system. Since global priority levels can be naturally related to a timing semantics (priority zero transitions model timed activities, while higher priority ones model instantaneous routing) they are preferred in the context of this book. Regarding the concurrent semantics of systems with priorities (and/or inhibitor arcs), compared to [4], we have taken the position to allow only the steps that can be linearised, the same as [24], a paper on the concurrent semantics of nets with inhibitor and test arcs.

Other extensions, besides inhibitor arcs and priorities, have been proposed in the literature. Basically, what extensions do is removing some basic assumptions of net models: inhibitor arcs and priorities concern the logic of enablement, which in basic net models is exclusively in terms of consumption (i.e., a transition for which enough resources are available is enabled). Another assumption of basic net models is that the effect of the occurrence of a transition on its neighbourhood is fixed, what is relaxed in nets with *reset arcs* [2], *self-modifying nets* [37], and, more generally, *nets with marking-dependent arc cardinalities* [12].

Bibliography

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* J. Wiley, 1995.

[2] T. Araki and T. Kasami. Some decision problems related to the reachability problem for Petri nets. *Theoretical Computer Science*, 3:85–104, 1977.

[3] L. Bernardinello and F. DeCindio. A survey of basic net models and modular net classes. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 304–351. Springer Verlag, 1992.

[4] E. Best and M. Koutny. Petri net semantics of priority systems. *Theoretical Computer Science*, 96:175–215, 1992.

[5] G. W. BRAMS. *Réseaux de Petri: Théorie et Pratique.* Masson, 1983.

[6] W. Brauer, editor. *Net Theory and Applications*, volume 84 of *Lecture Notes in Computer Science.* Springer Verlag, 1979.

[7] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Central Models and their Properties. Advances in Petri Nets 1986, Part I*, volume 254 of *Lecture Notes in Computer Science.* Springer Verlag, 1987.

[8] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Applications and Relationships to Other Models of Concurrency. Advances in Petri Nets 1986, Part II*, volume 255 of *Lecture Notes in Computer Science.* Springer Verlag, 1987.

[9] G. Chiola, S. Donatelli, and G. Franceschinis. Priorities, inhibitor arcs, and concurrency in P/T nets. In *Proc. 12^{th} Intern. Conference on Application and Theory of Petri Nets*, Aarhus, Denmark, June 1991.

[10] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11), November 1993.

[11] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A Symbolic Reachability Graph for Coloured Petri Nets. *Theoretical Computer Science B (Logic, semantics and theory of programming).* To appear in 1997.

[12] G. Ciardo. Petri nets with marking-dependent arc cardinality: Properties and analysis. In Valette, editor, *Application and Theory of Petri Nets 1994*, volume 815 of *LNCS*, pages 179–198. Springer Verlag, 1994.

[13] R. David and H. Alla. Petri nets for modeling of dynamic systems — a survey. *Automatica*, 30(2):175–202, 1994.

[14] F. De Cindio, G. De Michelis, L. Pomello, and C. Simone. Superposed automata nets. In C. Girault and W. Reisig, editors, *Application and Theory of Petri Nets*. IFB 52, New York and London, 1982.

[15] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

[16] J. Ezpeleta, J. Colom, and J. Martínez. A Petri net based deadlock prevention policy for flexible manufacturing systems. *IEEE Trans. on Robotics and Automation*, 11(2):173–184, 1995.

[17] A. Finkel. The minimal coverability graph for Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1993*, volume 674 of *Lecture Notes in Computer Science*, pages 210–243. Springer Verlag, 1993.

[18] M. Hack. *Decidability Questions for Petri Nets*. PhD thesis, M.I.T., Cambridge, MA, Dec. 1975. also tech. report 161, Lab. for Computer Science, June 1976.

[19] S. Haddad and J.M. Couvreur. Towards a general and powerful computation of flows for parametrized coloured nets. In *Proc. 9^{th} Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[20] P. Huber, A.M. Jensen, L.O. Jepsen, and K. Jensen. Towards reachability trees for high-level Petri nets. In G. Rozenberg, editor, *Advances on Petri Nets '84*, volume 188 of *LNCS*, pages 215–233. Springer Verlag, 1984.

[21] K. Jensen. Coloured Petri nets and the invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

[22] K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use. Volume 1.* Springer Verlag, 1992.

[23] K. Jensen and G. Rozenberg, editors. *High Level Petri Nets.* Springer Verlag, 1991.

[24] U. Montanari and F. Rossi. Contextual nets. *Acta Informatica*, 32:545–596, 1995.

[25] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[26] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, 1981.

[27] C. Petri. *Kommunication mit Automaten*. PhD thesis, Institut für Instrumentelle Mathematik, Univ. Bonn, 1962.

[28] L. Recalde, E. Teruel, and M. Silva. Modeling and Analysis of Sequential Processes that Cooperate Through Buffers. *IEEE Trans. on Robotics and Automation*, 14(2):267–277, 1998.

[29] W. Reisig. Deterministic buffer synchronization of sequential processes. *Acta Informatica*, 18:117–134, 1982.

[30] W. Reisig. *Petri Nets. An Introduction*. EATCS Monographs on Theoretical Computer Science. Springer Verlag, 1985.

[31] M. Silva. *Las Redes de Petri: en la Automática y la Informática*. AC, 1985.

[32] M. Silva. Introducing Petri nets. In *Practice of Petri Nets in Manufacturing*, pages 1–62. Chapman & Hall, 1993.

[33] M. Silva, J. Martinez, P. Ladet, and H. Alla. Generalized inverses and the calculation of symbolic invariants for coloured Petri nets. *Technique et Science Informatiques*, 4:113–126, 1985.

[34] Y. Souissi and N. Beldiceanu. Deterministic systems of sequential processes: Theory and tools. In *Concurrency 88*, volume 335 of *Lecture Notes in Computer Science*, pages 380–400. Springer Verlag, 1988.

[35] E. Teruel, J. M. Colom and M. Silva. Choice-free Petri Nets: A Model for Deterministic Concurrent Systems with Bulk Services and Arrivals. *IEEE Trans. on Systems, Man, and Cybernetics*, 27(1):73–83, 1997.

[36] E. Teruel and M. Silva. Structure theory of Equal Conflict systems. *Theoretical Computer Science*, 153(1-2):271–300, 1996.

[37] R. Valk. On the computational power of extended Petri nets. In *Proc.* 7^{th} *Sypm.Mathematical foundations of Computer Science*, volume 64 of *Lecture Notes in Computer Science*, pages 527–535. Springer, 1978.

Chapter 3

Timed Petri Nets

Petri nets were originally proposed as a causal model, explicitly neglecting time. The introdution of time into the basic net model entails many subtle difficulties in the definition of a coherent model. In particular, the specification of the behaviour in time in order to evaluate performance requires a great deal of *interpretation* to be added over the basic net system model. We approach the problem in two steps: first we define the idea of *observation* of the behaviour of an autonomous model in time; second we define a non-autonomous model in which each activity modelled by transition firing delays is specified in a proper way.

This chapter is composed of the following sections. Section 3.1 presents the mapping between Activities and Servers on the one hand, and transition enabling and firing on the other hand. Section 3.2 introduces the idea of *reduction of the non-determinism* which is inherent to the underlying autonomous Petri net model. As a particular case of non-determinism reduction, Section 3.3 discusses the idea of distinction between *timed* and *immediate* transitions, and how this distinctions can be related to the introduction of a *priority* interpretation. Section 3.4 introduces the idea of "servers" sitting inside transitions to perform the "firing work." This mapping of work to transition enabling is analysed from the point of view of queueing disciplines as defined in the Queueing systems setting. Also the implications of the presence of conflicts are thouroghly analysed, and different memory policies are introduced. Section 3.5 defines the problem of observing the evolution in time of the autonomous net system by means of the Occurrence Equations technique. Section 3.6 introduces then one application of the Occurrence observation, namely the Operational Analysis approach, that defines an "experimental" setting in which several performance indexes as well as model parameters may be identified and related one with the other. Specification of models and measurement of performance indexes are then related by means of the Operational Analysis approach in Section 3.7. The particular case of Memoryless stochastic timing is then presented in Section 3.8, and some of the most popular model definitions deriving from this approach are outlined in Sections 3.9, 3.10, and 3.11. Finally, concluding and bibliographical remarks

are summarized in Sections 3.12 and 3.13.

3.1 Transition Timing

One of the main objectives for a timed Petri net model of a distributed system is to describe the system activities in order to understand their mutual interactions and their timing.

Borrowing concepts from Queueing models, we may identify *servers* that carry out such activities in an asynchronous, independent fashion (except for interactions represented explicitly at the Petri net level). Broadly speaking, an activity may be defined as a *quantity of work* performed by a server in order to produce some results starting from a given pre-condition. The completion of such activity takes time, and the quantity of time needed depends on the *speed* of the server as well as on the quantity of work.

The completion of an activity may be viewed as a change in the state of the system: "the results produced by the activity become available." Since in Petri nets changes of states are determined by transition firing, it is natural and intuitive to relate the completion of activities to the firing of transitions. On the other hand, transitions may fire only after their enabling. A natural mapping of activities on Petri net models consists thus in identifying servers with enabled transitions: the enabling of a transition corresponds to putting a server to work on an activity, and the firing of the transition corresponds to the completion of the activity, with production of results.

More than one server may be started on activities of a given type. For instance, in a generic marking $\mathbf{m}$, the enabling degree $e_i(\mathbf{m})$ tells the number of servers that can work concurrently to carry out the activities of the type represented by transition $t_i \in T$ (i.e. we consider a notion of *self-concurrency*). The time needed to complete the activity is modelled by the time interval between the enabling and the firing of a transition *instance* (we'll provide a formal definition later on in Section 3.5). With this mapping in mind, we can easily keep the idea of *atomic firing* for transitions (which is a crucial assumption for untimed Petri net models), since the firing itself may be thought as an instantaneous event that concludes a timed activity. Using QNs terminology, transitions can thus be interpreted as *stations* that may have up to:

$$eb_i = \max\{\ e_i(\mathbf{m}) \mid \forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})\ \}$$

servers, where eb_i is called the *enabling bound* of transition t_i.

A problem arises when we consider the case of *conflicting transitions*. Let us concentrate on a free-choice conflict between two transitions. When a token is deposited in the input place, both transitions are enabled. With our interpretation, two servers are started (simultaneously) to work on the two conflicting activities modelled by the transitions. However, only one of them may come to its completion (modelled by the corresponding transition firing). The other activity must be preempted by the conflict. What do we do with the (portion of) work performed by the server before preemption? Several *memory* and

sampling policies may be defined to specify the behaviour of a timed Petri net model containing conflicting timed transitions. We delay the discussion of this problem to Section 3.5.4.

Let us now introduce a simple example in order to point out some of the problems that may arise in transforming an untimed model into a timed one. Consider the P/T system in Figure 3.1. The intended meaning of this model is to represent what usually happens in a restaurant offering a buffet for dessert including chocolate cakes. Only part of the cakes are available to customers in the form of sliced pieces on a small table. On the table there is room for 3 customers' trays as well as for a big dish that can contain at most 16 slices of cake. Each customer of the buffet must wait in queue for a free spot for his tray on the small table, then he/she can deposit his/her tray on the table, put one or two slices of cake on the tray, take the tray and come back to his/her table to eat the cake. It may happen that when a customer comes to the cake table no slice of cake is left. In this case customers wait at the table for a waiter to bring more slices. Usually, customers take one slice of cake, however sometimes they like chocolate so much that they help themselves with two slices if more than one is available on the table. However a customer never waits at the table for a second slice if the big dish is empty (he/she prefers to eat the first slice, and eventually come back later to the buffet to take on or two slices more). A waiter comes to whatch the cake buffet from time to time. If the waiter sees that there are less than 9 slices left on the big plate, he goes into the kitchen, takes a new full cake, cuts it in 8 slices, and brings the new 8 slices to the big dish on the table. One may easily follow the flow of all these activities by playing the token game of the P/T system and convince oneself that the model correctly represents the qualitative behaviour of the system we wanted to model, as described above.

Let us now try to use the rules outlined at the beginning of this section in order to associate activity interpretation to transition enablings, therefore coming to a model than can meaningfully describe our system's behaviour in time. Transition "cometobuffet" is an example of how the enabling degree can be meaningfully associated to the number of concurrent, independent activities that we want to model. Each customer autonomously decides when to go to the buffet to take his/her dessert, independently of the presence of other customers (this is a simplistic model where social habits and manners are not taken into accounts). Each enabling instance of the transition may be associated with its own, independent firing time. We might think of an ordered FIFO queue for the access to the three available places around the table, however we do not represent any queueing discipline explicitly in the model. The only feature that we do explicitly represent in the model is the fact that service is not interruptible (i.e., that once somebody has put his/her tray on the small table, all other waiting customers will not try to overtake the customer for the access to the slices).

Suppose now that only one slice of cake is left on the big dish and that 3 customers have put their trays on the table and are trying to get their first slice. In this case the enabling degree of transition "takeapiece" is 1 (due to the presence of a single token in place "slices"), meaning that only one of the customer is actually taking the only available piece, while the other two customers

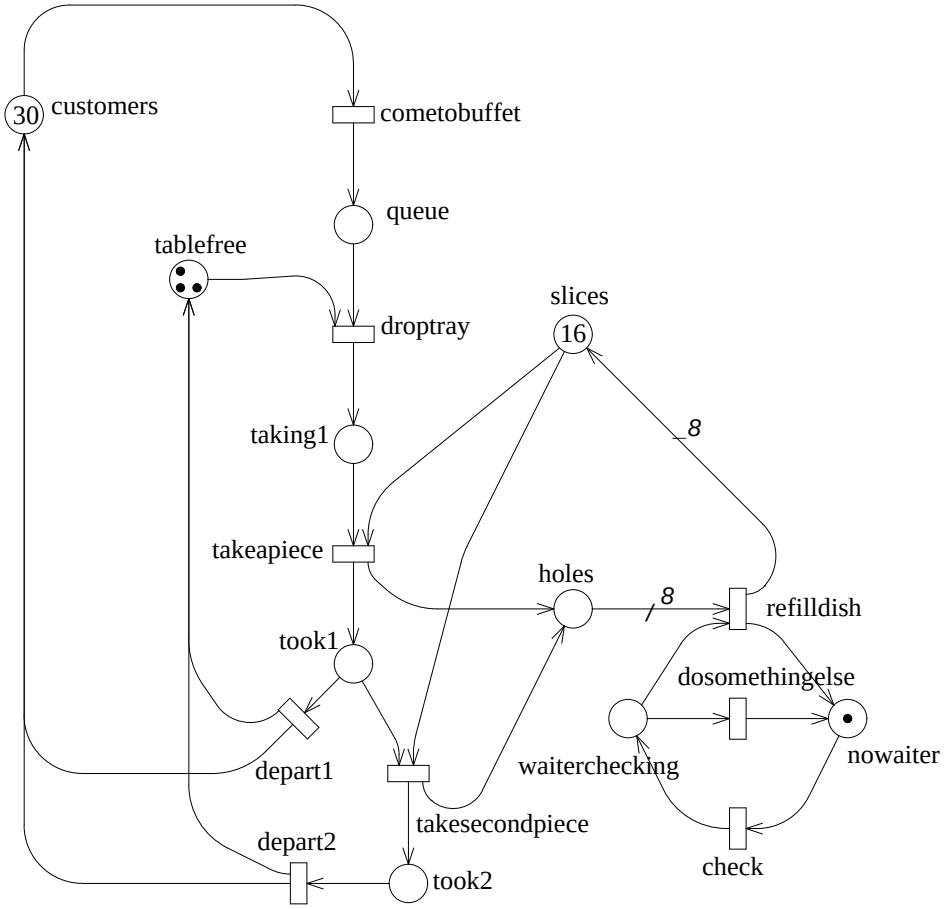


Figure 3.1: Untimed "dessert buffet" P/T system

are patiently waiting for other slices to be brought by the waiter. Besides the problem of not knowing who among the three is taking the piece of cake and who is waiting (due to the fact that tokens are not coloured), we have to admit that there must be some gentlemen's agreement or some strong social rule followed by all customers in order for this idealized model's behaviour to make sense, otherwise more than one of the three customers could make an attempt to take the only piece of cake at the same time. On the contrary, if we consider the situation in which one customer has already taken his/her first slice and wants a second slice while only one slice is left on the big dish and two other customers are about to take their first slice, than the two transitions "takeapiece" and "takesecondpiece" are enabled in conflict, thus modelling a race in which it appears obvious to expect that the quickest among the two wins the last available slices while the slowest has to abort his action and try again when the waiter has brought other slices (or leave the buffet if the loser is the one that already got his/her first slice of cake). Therefore we see that the introduction of a timing interpretation according to the rules set up above leads to a kind of inconsistency in the model: by using the above sketched time interpretation, we are implicitly assuming that strong rules have been defined to solve conflicts for taking the first slice of cake or the second slice of cake, but not to solve a-priori conflicts between people that already took one slice and people that had not taken any piece of cake, which appears to be quite strange.

Another source of model interpretation problems is also the conflict between transitions "depart1" and "takesecondpiece" that arises in case there are some slices left in the big dish. Does it make sense to think at two conflicting activities performed by the same customer? Is the customer using one hand to start taking the piece of cake, and at the same time the other hand to grab the tray from the table and base the decision whether to take a second slice or not on the relative speed of these two actions? And what happens in case of conflict between "takesecondpiece" and "takeapiece"? Moreover, what happens if "takesecondpiece" is enabled after "depart2" but before its firing, due to the re-filling of the big dish by the waiter? Does the customer change his/her mind and take a second piece, delaying his/her departure? A correct and consistent interpretation of the model should properly answer all questions of this type.

3.2 Reduction of the Non Determinism

The starting point for the definition of Timed Petri Nets is a non deterministic net system, in which several details concerning the behaviour are left undefined. A time interpretation of the underlying net system consists then in specifying the behaviour "in time" in such a way that:

1. it is "compatible" with the untimed, non deterministic model;

2. the "amount of non determinism" is reduced in order to take the timing constraints into account;

3. the behaviour of the system is specified precisely enough in order to derive performance measures (the degree of precision required depends on the analysis technique that is used).

The problem of reducing the non determinism may be attacked in several different ways. Here we identify three different techniques that may be combined one with the other, and whose combination covers all kinds of "timed Petri nets" proposed in the literature so far:

N) reduce the amount of non determinism, still keeping the model non deterministic (but to a lesser extent);

S) introduce a stochastic measure for the non determinism of the underlying model

D) define deterministic algorithms to substitute non deterministic specifications

Non determinism may assume two (slightly) different forms in a timed net system: choice among conflicting transitions and "duration" of the enabling for transitions (independently of the fact that they can be persistent or in conflict with other transitions). Both types of non determinism may be solved by any of the three above mentioned basic techniques (or combinations of them).

3.2.1 Non Deterministic Reduction

The classical example of this approach is the definition of the so called "interval timed" net systems (originally introduced by Merlin and Farber in the context of verification of communication protocols involving time-out mechanisms). A *minimal* and a *maximal* firing time are associated with each transition $t \in T$ as part of the net interpretation (we denote them $\tau_m(t)$ and $\tau_M(t)$, respectively). The firing rule is re-defined in time as follows: any transition $t \in T$ must remain enabled at least $\tau_m(t)$ units of time before being able to fire; a transition that remains enabled for $\tau_M(t)$ time units is forced to fire at the elapsing of the maximum period. Of course the inequality $\tau_m(t) \leq \tau_M(t)$ must always be satisfied in order for the time definition to make sense (otherwise we would specify deterministically dead transitions). Nothing is said concerning the actual firing time in case $\tau_m(t) < \tau_M(t)$: it can assume any value within the interval in a totally non deterministic way. Moreover, in case t is non-persistent, t can be disabled before firing and before reaching its maximum enabling time $\tau_M(t)$.

Indeed, non interpreted net systems may be redefined as interval timed nets with $\tau_m(t) = 0$ and $\tau_M(t) = \infty \; \forall t \in T$, without changing their non deterministic behaviour. On the contrary, the behaviour of a timed model with $\tau_m(t) > 0$ and/or $\tau_m(t) < \infty$ for some $t \in T$ may be different from the one of the uninterpreted net system, even from a purely qualitative point of view (existence of deadlocks, livelocks, boundedness, etc.). For example, consider the three P/T systems with interval time interpretation depicted in Figure 3.2. The P/T system depicted in Figure 3.2.a) is subject to deadlock without the timing

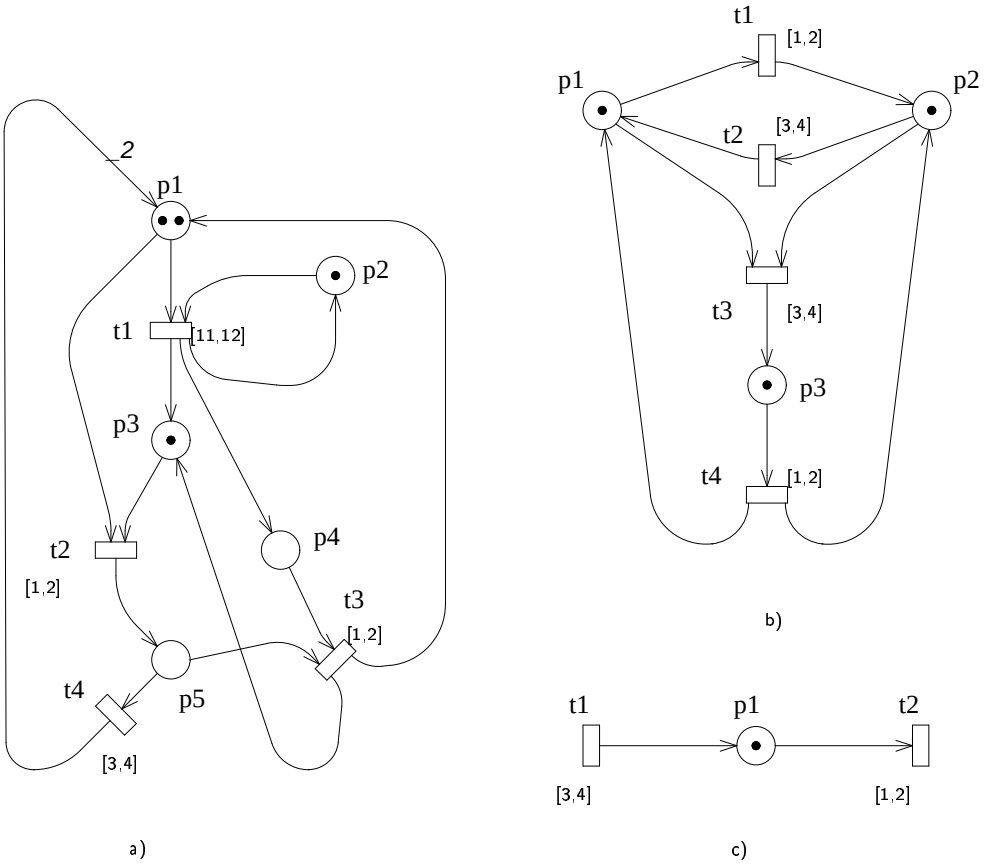


Figure 3.2: Effect of interval time interpretation on qualitative properties of P/T systems

interpretation (the deadlock is reached, e.g., by firing t1 twice from the initial marking), while the interval time interpretation prevents such transition firing sequences to occur, thus making the interpreted model live and bounded. The P/T system depicted in Figure 3.2.b) is live and bounded, while the addition of the timing interpretation creates a livelock and makes transitions t3 and t4 dead. The underlying P/T system depicted in Figure 3.2.c) is live and unbounded, while the addition of the time interpretation makes place p1 safe (still keeping the model live). In general we can conclude that the restrictions on firing sequences due to the timing interpretation may provide new safeness properties and may add, change, destroy or keep unchanged the liveness properties as compared to the non interpreted underlying Petri net.

Quite clearly, this type of non determinism reduction alone does not make the specified behaviour of the model sufficiently precise to compute "exact" performance (unless $\tau_m(t) = \tau_M(t)$ for all transitions). At most (upper and lower) bounds may be computed, due to the lack of knowledge on the real occurrence instances for transition firing. In fact this kind of timed models have been used essentially for (qualitative) property assessment, as an alternative to non interpreted net systems to take timing constraints into account.

The same kind of reduction can be applied to the non determinism involved in the choice among several conflicting transitions. Although never appeared in the literature (to the best of our knowledge), the obvious counterpart for τ_m and τ_M would be the introduction of "fairness constraints." Let us consider the case of a free-choice between two transitions $t1$ and $t2$. One might specify as an additional interpretation of the net system, that transition $t1$ cannot fire more than $k1$ times consecutively without firing $t2$ at least once (and vice-versa, $t2$ may not fire more than $k2$ times without firing $t1$ at least once). Provided that $1 < ki < \infty$, this kind of interpretation would restricts the amount of original non determinism of the underlying net system, while still not imposing any deterministic behaviour. Notice that this interpretation is different with respect to imposing a finite synchronic distance relation between the transitions at the net system level, so that the addition of such interpretation would entail a real change in the modelling power.

We do not elaborate any further on this possible interpretation since we are not going to use it in the following of this book. We just presented it with the purpose of clarification and illustration of basic concepts.

3.2.2 Stochastic Reduction

This is the "main" way used for the reduction of non determinism in case one wants to use the interpreted model for performance evaluation. The idea is to preserve the "possible variability" which is inherent to the underlying non deterministic net system, and at the same time make the description of the behaviour precise enough to be able to apply statistical analysis techniques to compute performance. In this case the reduction is "complete" in the sense that a mathematically precise description is given, nothing being left undefined.

This kind of reduction is obtained by adding a "probability measure" to

the space of variability of the non deterministic model. A probability density function (pdf) with infinite support $[0, \infty)$ is associated with each instance of transition firing delay, and a discrete probability is assigned to each conflicting transition for the choice within each effective conflict set. The actual value of each instance of firing delay becomes thus a *random variable*, that in principle may assume any value in the range $[0, \infty)$, as in the underlying net system. Differently from the uninterpreted net system, however, now we can apply statistical analysis (such as, e.g., the weak law of large numbers) to estimate probabilities of some events to occur. Notice that in fact in the non interpreted model it does not make sense to introduce the concept of probability and/or expected frequencies of events due to their pure non deterministic nature that does allow one to assume any kind of "typical" or "expected" behaviour for the firing times or outcome of conflict choices.

From a qualitative point of view, the stochastic interpretation does not change the behaviour of the model with respect to the non interpreted net system. The addition of a probability measure may only "complement" such behaviour with additional information that allows the application of the idea of expected behaviour from the observation of the model. If we think for instance at the reachability relation, reachability analysis of the non interpreted net system may tell whether a given marking *can* be reached from the initial marking or not. The reachability property is not affected by the stochastic interpretation. However the addition of a stochastic interpretation allows one to ask the additional question: "How likely is the event of reaching that marking after τ time units from the start of the observation when the model for sure was in its initial marking?"

Notice that the stochastic interpretation may follow two different approaches. We can either assume to know the stochastic processes governing the behaviour of the net system (we assume a mathematical model) or want to estimate their parameters based on the observation of the actual behaviour of such models (we experimentally measure the system, or its simulation model). We shall come back on the relation between these two attitudes in the last sections of this chapter.

3.2.3 Deterministic Reduction

This is again a way of reducing the non determinism in a complete way. This time the reduction is obtained eliminating the possibility of variation. In some sense this could be viewed as a special case of the non deterministic reduction, in which $\tau_m = \tau_M$. However, we prefer to keep this case separate in order to fully understand all the implications of this kind of reduction. For instance, the fact that the complete behaviour of the interpreted model is specified in a deterministic way allows one to compute all performance results (at least all transient time results, by simple "simulation"). On the other hand the interpreted model may enjoy more safeness properties and different liveness properties as compared to the underlying net system.

Deterministic conflict resolution policies can obviously be defined in addition

to the definition of deterministically fixed transition firing delays.

3.2.4 Mixing different Reduction techniques

Sometimes one may find in the literature proposals of timed interpretations for Petri nets in which deterministic firing delays and stochastic choice among conflicting transitions are mixed (see, e.g., DSPNs). This is probably the most common example of mixture of different techniques for the reduction of the non determinism.

Another example is the definition of "finite support" pdf for transition firing delays in a stochastic interpretation framework. Indeed this can be viewed as the superposition of the first technique (to reduce the support from $[0, \infty)$ to $[\tau_m, \tau_M]$) and of the second technique (to provide a probability measure on this range of variability). This way of presenting the interpretation makes it clear that this interpretation inherits all properties of the two reduction techniques, namely it allows "exact" performance analysis, and (in general) changes the qualitative behaviour with respect to the underlying net system without interpretation.

3.3 Timed and Immediate Transitions

Intuitively, before discussing all details of transition timing interpretations it makes sense to introduce a first rough distinction between zero time events and non-zero time activities. The motivations to make such a distinction are various. From the modelling point of view, sometimes it is useful to implement in form of net system some small algorithm to compute more complex results that cannot be directly expressed in terms of Petri net primitives. In these cases, we do not want the firing of these "additional transitions" to perturb the behaviour in time of our model, so that we can specify them as "immediate transitions," implementing all steps of our algorithm in zero time. For example, Figure 3.3 depicts a subnet composed by transitions "remove" and "endflush" and place "flushpp" which purpose is to implement an iterative algorithm to empty place "pp" as soon a token is put in place "flushpp," no matter how many tokens "pp" contains at this time. The termination of the algorithm is indicated by firing of "endflush" that removes the controlling token from place "flushpp."

In some other cases, the duration of some activities might be negligible compared to others. For example, in the "dessert buffet" model depicted in Figure 3.1 the duration of the activity modeled by transition "droptray" might be much shorter than the durations of the other activities, so that it could make sense to approximate it with a zero time activity. In these cases it could be conceptually simpler and/or more convenient/efficient from the analysis point of view to define the "short" activities as actually taking zero time.

In any case the interaction between timed and immediate transitions must be clearly defined. The usual way of defining this interaction is to introduce the concept of *priority*. This is intuitively justified in the following way. If we

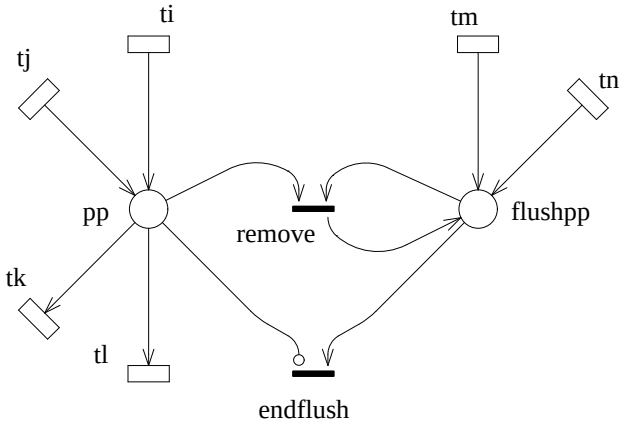


Figure 3.3: An example of marking manipulation algorithm implemented by means of a subnet

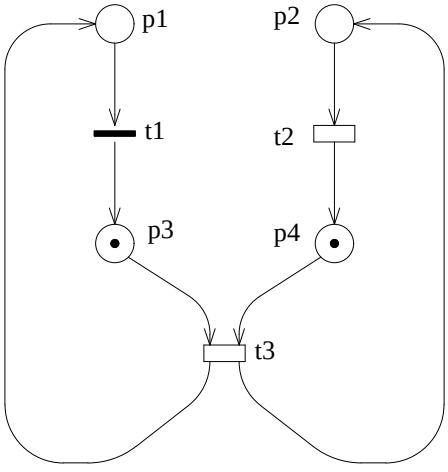


Figure 3.4: Immediate and timed transitions

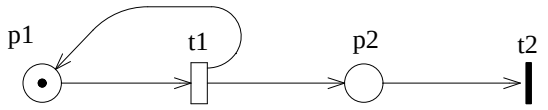


Figure 3.5: An unbounded system that becomes safe due to priority interpretation

observe the behaviour of two transitions $t1$ and $t2$, one immediate (such as, e.g., $t1$, in Figure 3.4) and the other one characterized by a firing delay $\epsilon > 0$ (such as $t2$ in Figure 3.4), once the two are enabled simultaneously (such as, e.g., in Figure 3.4 due to the firing of transition $t3$), the immediate transition will always fire *before* the timed one. Hence we might intuitively say that $t1$ has priority over $t2$ due to the fact that it is immediate rather than timed.

This intuitive idea of priority of immediate transitions over timed transitions may be formalized (and extended) as an additional and separate *interpretation* of the basic net system, independently of the concept of timing. This introduction of priorities into untimed net systems allows one to study the interaction between timed and immediate transitions without actually having to define the timing interpretation of the net, and see what are the effects on qualitative properties of the net system. For instance, the net system with priorities depicted in Figure 3.5 is bounded due to the priority interpretation, independently of the actual time interpretation, while the underlying non interpreted P/T net is structurally unbounded. After the introduction of a *consistent* priority interpretation, the introduction of a time interpretation (even including immediate transitions) does not change any more the qualitative properties of the model. In order to establish this consistency between the time interpretation and the use of priorities, we require that

Definition 3.1 (Timing and Priority consistency) *a timed model is a net system with a priority structure such that:*

timed *transitions (i.e., transition with non zero firing delay) have priority 0 (the lowest level priority). They are depicted as a rectangular boxes in the graphical representation;*

immediate *transitions (i.e., transition with zero firing delay) have priority* > 0*. They are depicted as thin bars in the graphical representation.*

Under these conditions, priority and timing specifications are complementary for the definition of the observable behaviour in time; moreover, immediate transitions always fire before timed ones. A classical example that can be found in literature of a consistent timed and prioritized model is Generalized Stochastic Petri Nets (GSPN).

3.4 Specification of Servers, Conflicts, and Policies

3.4.1 Finite and Infinite Servers

Let us consider the two related examples of PLC net systems in their coloured and non coloured versions depicted in Figures 2.16 and 2.18(d). The two models are intended to represent exactly the same behaviour at different levels of detail: the first model allows one to keep track of which processor is doing what; the second model corresponds to a more abstract view of the system, in which we can still observe the activity of the different processors but we cannot keep track of their identity. In order to have this kind of "equivalence" between coloured and decoulored models, we need to assume the *infinite-server* semantics for relating activities to transition enabling.

Definition 3.2 (Infinite server semantics) *In a timed model the enabling degree of each timed transition represents the number of concurrent activities modelled by the transition's enabling in a given marking.*

Assuming infinite-server semantics for timed transitions means that if we want to model the equivalent of a single-server queue, we have to make sure that the timed transition representing the completion of the server activity has a maximal enabling degree of 1. This may be obtained in a systematic way by adding a place always containing one token, connected with both an input and an output arc to the "server" transition. As a shortened notation in the graphical representation of net systems, we may avoid putting the additional place to reduce the maximum enabling degree and use instead the inscription "1-server" (or "k-server" in general) associated with the transition.

3.4.2 Queueing Policies

When we consider the case of a k-server transition which would be enabled more than k times (if we didn't add the server constraint to the net system), then the issue of *queueing policy* similar to the one explicitly addressed in queueing models could arise. In principle, if we have a limit of k servers available to carry out activities in parallel, and we have $n > k$ "customers" demanding activities, then the best we can do is to choose k customers and start their activity while the other $n - k$ customers remain waiting for a server to become free. The queueing policy is precisely the algorithm that we follow to select k out of n customers to give them the "privilege" of starting their service right away. In queueing models several queueing policies can be specified, some of them taking into account the order of arrival of requests (like, e.g., FCFS, LCFS, etc.), some other distinguishing customers in classes (e.g., Priority), some other implementing more "democratic" policies in order not to make distinctions among customers (e.g., PS), etc.

In our framework, since tokens (of the same colour) in the same place are indistinguishable it does not make sense to pose the problem in case of persis-

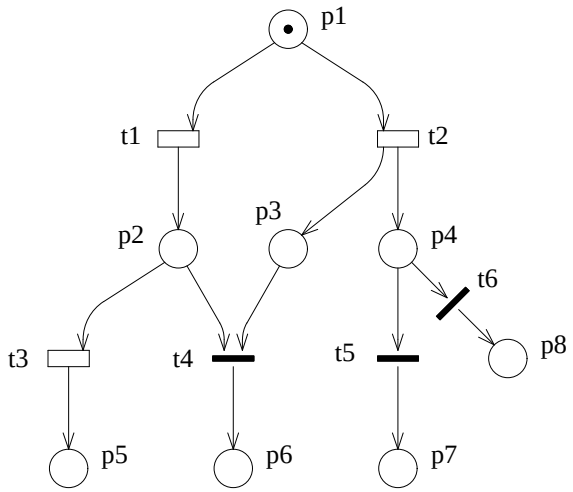


Figure 3.6: Several types of conflicts in a net system with priority

tent transitions: the behaviour that can be observed at the level of details of operational quantities wouldn't be affected by different choices. On the contrary, if conflicts may occur, then the choice of a particular queueing policy would affect the conflict resolution policy: assuming for instance FCFS would mean specifying a deterministic reduction of the non-determinism inherent to the choice at the Petri net level. Usually explicit queueing policies are not defined for timed Petri nets, and the problem is solved at the level of choice of the semantics for the case of conflicting transitions (i.e., at a finer grain level of the net system behaviour). In general Priority, RandomOrder, and PS queueing disciplines are straightforward to implement using this approach (especially in case of Markovian timing), while state-dependent queueing disciplines like FCFS, LCFS, longest job first, etc., result in very cumbersome net system representations.

3.4.3 Conflict resolution: Race vs. Preselection

Consider the net system (with priority) depicted in Figure 3.6. It contains three different types of conflicting situations:

a (free-choice) conflict between timed transitions $t1$ and $t2$;

b (non free-choice) conflict between timed transition $t3$ and immediate transition $t4$;

c (free-choice) conflict between immediate transitions $t5$ and $t6$.

Let us consider the three cases separately to explain the different choice policies that can be defined.

Case a. In principle we can imagine two different ways of solving the conflict, both perfectly reasonable from the modelling point of view: either take the timing definition into account (*Race Policy*) or assume a choice independent of the timing (*Preselection Policy*).

According to the Race Policy, each transition independently "decides" its own firing delay; the two enablings start at the same time, so that the transition that is associated with the shortest firing delay "wins the race" and fires; the other transition "loses the race," and its enabling is preempted by the conflict.

According to the Preselection Policy, instead, first an agreement is reached (ideally in zero time) between the conflicting transitions, according to a protocol or algorithm that does not necessarily take their firing delays into account; then the transition that is selected for firing "declares" its firing delay, and waits that amount of time before firing; on the contrary, the other transition (that lost the conflict resolution) behaves as if it were not enabled, not even thinking about what would have been its enabling time if it had won the conflict.

Case b. First of all notice that it does not make sense to consider free-choices in the case of conflicts between transitions at different priority level (otherwise the lower priority transitions would be dead). Hence, if we want to give the timed transition a chance of firing we have to consider cases in which $t3$ is enabled and $t4$ is not. The only meaningful conflict resolution policy is then the Race Policy in this case, where $t3$ starts counting its firing delay, and "wins the race" if and only if $t4$ does not become enabled (due to the firing of $t2$ in our example) before the firing delay of $t3$ expires.

Case c. In this case only a Preselection Policy makes sense, since the firing delay for both transitions is identically equal to 0 (so that the Race Policy would always result in a tie, and could not decide which one is going to fire).

We can thus realize that both Race and Preselection policies make sense, even mixed in the same model for solving different conflicts. The Preselection policy enforces a clear separation between the conflict resolution and the transition timing, so that in some sense it is theoretically cleaner. On the other hand, the Race Policy is needed to model a lot of interesting phenomena that are usually found in distributed systems (race conditions, timeouts, etc.) so that its introduction in timed Petri net models is unavoidable.

Finally, notice that Preselection among timed transitions may be simulated by a structurally different model in which Case a is substituted by Case c and immediate transitions are followed by *persistent* timed transitions. Hence, in a mixed model containing both timed and immediate transition, one could always assume Race policy among conflicting timed transitions without major loss of modelling power with respect to the case in which both Race and Preselection policies are allowed for conflicting timed transitions.

3.4.4 Conflict and preemption: memory policies

In case of non-persistent timed transitions the choice of the Race policy is not sufficient to determine the behaviour of the model. We have still to decide what we are going to do with the firing times of transitions that lost the race. Notice that this distinction does not make sense in case of Preselection policy, since only the selected transition instance is actually started, so that the issue of interrupted instances does not raise.

At least two main alternatives may be identified that yield to a timed behaviour consistent with the qualitative behaviour of the non-interpreted underlying net model for conflicting timed transitions with Race policy:

Age memory : a transition instance that lost the race, becoming disabled before its firing, is considered to be frozen by the "photo finish" of the race. As soon as the same transition is enabled again, the frozen instance is resumed and continues its activity until either firing or being frozen again by a faster conflicting transition instance.

Enabling memory : a transition instance that lost the race is considered lost. When the same transition is enabled again, the previously interrupted instances are "forgotten", and a new fresh transition instance starts its activity from scratch.

The two alternatives yield quite different behaviours from the modelling point of view: the former allows a trivial representation of "round-robin" scheduling disciplines and it makes it impossible to represent a time-out mechanism in a natural way; the latter has opposite characteristics in these respects. Notice that these policies are orthogonal with respect to the way in which non determinism is reduced (whether by restriction of the non determinism or by addition of a stochastic measure on the space of possibilities).

Both Age and Enabling memory policies must be further specified in case of multiple server transitions: when more than one transition instance is enabled, and the transition's enabling degree is reduced to a lesser value still greater than zero, an algorithm must be specified to select which transition instance has to continue to run and which has to be either frozen or discarded. Different disabling disciplines could be specified, exactly as in the case of the queueing disciplines for queueing models, such as, e.g.: FIFO, LIFO, Random choice, Largest value, Smaller value, etc. In case of multiple server Age memory transition, also a re-scheduling policy must be specified in a similar way to decide which one among the frozen instances have to resume when the transition's enabling degree increases of a value less than the total number of currently frozen instances.

Moreover, in case the firing time is subject to variability (either stochastic or non-deterministic) at least one "intermediate" alternative could be considered:

Reset memory : the work performed by a transition instance that lost the race is considered lost. However, trace is kept of the work that was discarded. When the same transition is enabled again, the previously inter-

rupted activity must be restarted from scratch, so that the same value will be used as the transition firing time. This policy is sometimes called "preempt-repeat-identical" (PRI) in the framework of stochastic queues, as opposed to "preempt-repeat-different" (PRD) which is our concept of Enabling memory.

The difference between Enabling and Reset memory is subtle: if the firing time is subject to variability (either stochastic or non-deterministic), the Enabling memory policy cannot guarantee that a transition instance restarts from scratch "the same activity" that was interrupted by disabling a previous enabling instance of the same transition, while the Reset memory policy does guarantee this condition. Reset and Enabling memory collapse to the same policy when the transition firing time is deterministic.

3.4.5 An Example

Let us consider again the example of the Dessert Buffet outlined in Figure 3.1 and try to assign it a consistent time interpretation following the considerations developed so far.

Most of the problems of time interpretation already noticed may be solved in a easy and elegant way by providing a Preselection policy interpretation for the conflict among transitions "takeapiece," "depart1," and "takesecondpiece." A second conflict that should be solved by Preselection is the one between transitions "refilldish" and "dosomethingelse." A modified Petri net model more consistent with the timing behaviour of our considered system is thus depicted in Figure 3.7. This different representation of the Dessert Buffet system has the nice property of having all timed transitions persistent, thus making much easier the definition of their firing policies. All timed transitions are interpreted as of infinite-server type, thus possibly modelling multiple, independent activities associated with different enabling instances.

Preselection is of course adopted for the solution of conflicts among immediate transitions. The current marking affects the choice in case of non free-choices such as, e.g. the conflicts between "take2" and "depart1" or between "refill" and "dosomethingelse." The structural conflict between "take1" and "take2" is deterministically solved in favour of "take1" by means of a priority specification, thus modelling a consistent behaviour of customers that always agree in letting people take a first slice refraining from taking a second slice in case only a last slice is available on the big dish.

3.5 Observation in Time: Occurrence Equations

The behaviour in time of a P/T or even coloured Petri net model can be completely represented in terms of *sequences of occurrence times* of the events of interest in the net, i.e., the creation and destruction of tokens in places and the enabling and firing of transitions. Let us consider the example depicted in Figure 3.8. Notice that transition $t4$ is drawn as an immediate transition, so that

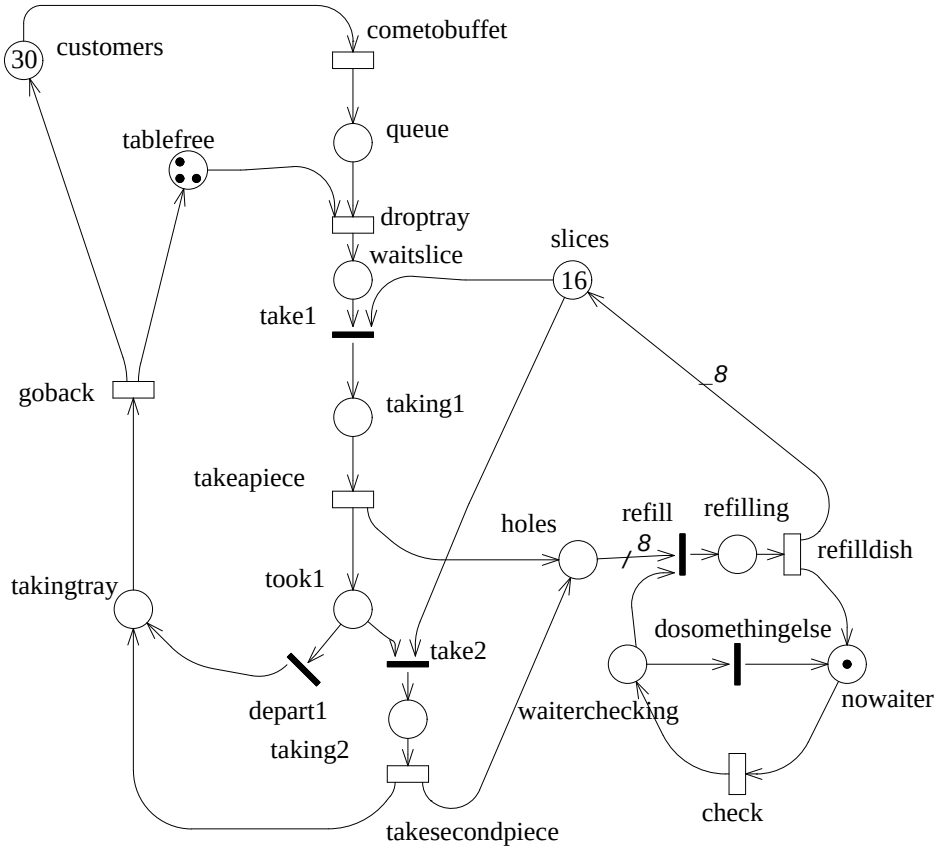


Figure 3.7: Timed "dessert buffet" Petri net model

we always observe a *firing delay* (as defined below) $r(t4, i) = 0$. If we take the autonomous system, and observe it starting at time $\tau = 0$ in the initial marking $\mathbf{m_0}$ (when we can assume the initial token "arrives" in $p1$), we could for instance notice the first firing of transition $t1$ at time instant $\tau_1 = 3$, and of course at the same time we would observe also the occurrence of the three related events "departure of the first token from $p1$," "arrival of the first token in $p2$," and "arrival of the first token in $p3$." At time $\tau_2 = 4$ we could then observe the first firing of transition $t3$ (together with its input token departure and output token arrival), and so on, as illustrated in the occurrence time diagram depicted in Figure 3.9.

In order to formalize the study of this kind of (extremely detailed) behavioural description we start by defining occurrence equations, then we show by means of examples how to use them in order to describe the behaviour in time of a Petri net model in a mathematically precise way.

3.5.1 Definitions

We start by defining the four basic observable events in the framework of a P/T net model. All these definitions could be trivially extended to coloured net models in which $\forall x \in P \cup T$ a colour domain $cd(x)$ is defined that assigns a "type" to tokens in places and to transition firing instances. We refer the reader to more specialized literature for such an extension, and concentrate on the main ideas for the sake of an introduction:

$\forall p \in P, \quad \forall i \in \mathbb{N},$

$\alpha(p,i)$ arrival time of the i-th token in p

$\delta(p,i)$ departure time of the i-th token from p

$\forall t \in T, \quad \forall i \in \mathbb{N},$

$\epsilon(t,i)$ time of the i-th enabling of t

$\phi(t,i)$ time of the i-th firing of t

For convenience we assume the following boundary conditions that take the initial marking into account:

$\forall p \in P, \quad \forall i \leq \mathbf{m_0}[p],$

$$\alpha(p,i) = 0$$

$\forall t \in T,$

$$\epsilon(t,0) = \phi(t,0) = 0$$

Sequences of events are ordered by non-decreasing *occurrence time* (also called *time-stamp*):

$\forall p \in P, \quad \forall i \in \mathbb{N},$

$$\alpha(p,i) \leq \alpha(p,i+1) \ \wedge \ \delta(p,i) \leq \delta(p,i+1)$$

$\forall t \in T, \quad \forall i \in \mathbb{N},$

$$\epsilon(t,i) \leq \epsilon(t,i+1) \ \wedge \ \phi(t,i) \leq \phi(t,i+1)$$

Moreover, for causality and marking consistency, such observable occurrence times always satisfy the following (obvious) inequalities:

$\forall p \in P, \quad \forall i \in \mathbb{N},$

$$\alpha(p,i) \leq \delta(p,i)$$

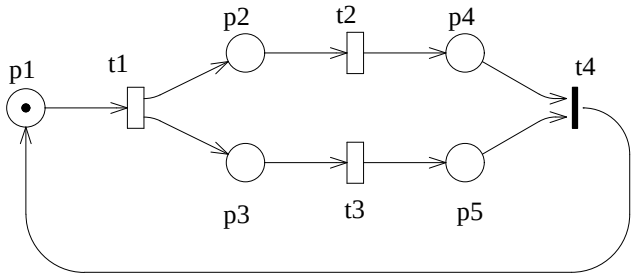


Figure 3.8: A Marked Graph example.

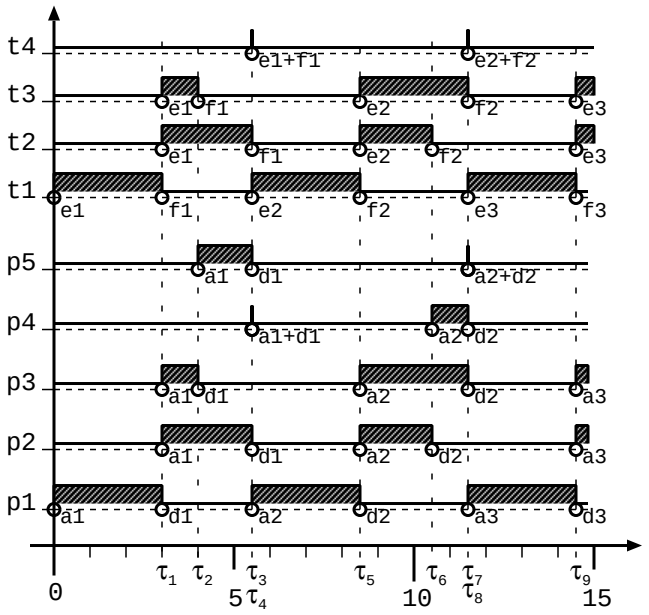


Figure 3.9: An occurrence time diagram of a possible experiment from time 0 to time 15.

$\forall t \in T, \quad \forall i \in \mathbb{N},$

$$\epsilon(t,i) \leq \phi(t,i)$$

Relations exist among such occurrence times since tokens are created and destroyed by transition firings, and vice-versa transitions are enabled by the availability of new tokens in their input places. However, the mathematical definition of such relations depends on the timing semantics of the model as well as on its structure.

For example, a very simple relation exists between enabling and firing times of transition instances in case of *persistent*, single-server transitions (i.e., transitions that may not be disabled after they become enabled until their firing, and such that the enabling is associated with a single activity): $\forall t \in T$ such that t is persistent and single-server, $\forall i \in \mathbb{N}$,

$$\phi(t,i) = \epsilon(t,i) + r(t,i)$$

where $r(t,i)$ denotes the i-th instance of the firing delay of transition t. In the following of this chapter we shall see that the firing delay may be specified in several different ways. In any case, usually the firing delays instances $r(t,i)$ can be assumed as (in some sense) known input parameters for the model.

In other words, if we limit ourselves to persistent models and if we assume to know the initial marking and the sequence of firing delay instances $r(t,i)$ for all transitions, then we are able to compute the occurrence times for all significant

events observable in the model. The way in which we define the sequence of firing delays for each transition depends on the way we chose for the non determinism reduction of the underlying net system: they will be constant predefined values in case of deterministic timing, or random variables with specified probability density function in case of stochastic timing.

On the other hand, if a transition is *not persistent*, the i-th enabling instance is not guaranteed to yield a transition firing event, since this could be preempted by the firing of a conflicting transition. In the general case, the j-th firing event may be related to the i-th enabling event with $i \geq j$ depending on the actual number of enablings that have been preempted before they could accomplish a firing event.

Similarly, the relation between enabling instances of a transition t and token arrivals in its input places $p \in {}^\bullet t$ is straightforward in case of persistent, single-server transitions with FIFO token scheduling and identity input functions (in which case the i-th enabling corresponds to the latest among all input places of the i-th token arrival), while it becomes intricated in the general case.

3.5.2 An MG, FIFO example

In order to clarify the meaning of the definitions introduced so far, let us consider the extremely simple example depicted in Figure 3.8. The initial conditions are in this case:
$\alpha(p1,1) = 0$ and $\epsilon(t1,1) = 0$.

The evaluation of all other functions is a simple mathematical representation of the concept of event simulation in the model. The occurrence functions for the following events can be evaluated in increasing time-stamp order as a function of previously evaluated event time-stamps. For example, we can write:

$$
\begin{array}{lcl}
\phi(t1,1) & = & \epsilon(t1,1) + r(t1,1) \;=\; r(t1,1) \\
\delta(p1,1) & = & \phi(t1,1) \;=\; r(t1,1) \\
\alpha(p2,1) & = & \phi(t1,1) \;=\; r(t1,1) \\
\alpha(p3,1) & = & \phi(t1,1) \;=\; r(t1,1) \\
\epsilon(t2,1) & = & \alpha(p2,1) \;=\; r(t1,1) \\
\epsilon(t3,1) & = & \alpha(p3,1) \;=\; r(t1,1) \\
\phi(t2,1) & = & \epsilon(t2,1) + r(t2,1) \;=\; r(t1,1) + r(t2,1) \\
\phi(t3,1) & = & \epsilon(t3,1) + r(t3,1) \;=\; r(t1,1) + r(t3,1) \\
\delta(p2,1) & = & \phi(t2,1) \;=\; r(t1,1) + r(t2,1) \\
\delta(p3,1) & = & \phi(t3,1) \;=\; r(t1,1) + r(t3,1) \\
\alpha(p4,1) & = & \phi(t2,1) \;=\; r(t1,1) + r(t2,1) \\
\alpha(p5,1) & = & \phi(t3,1) \;=\; r(t1,1) + r(t3,1)
\end{array}
$$

$$
\begin{array}{lcl}
\epsilon(t4,1) & = & \max\{\ \alpha(p4,1), \alpha(p5,1)\ \} \\
 & = & r(t1,1) + \max\{\ r(t2,1), r(t3,1)\ \} \\
\phi(t4,1) & = & \epsilon(t4,1) + 0 \\
 & = & r(t1,1) + \max\{\ r(t2,1), r(t3,1)\ \} \\
\delta(p4,1) & = & \phi(t4,1) \\
 & = & r(t1,1) + \max\{\ r(t2,1), r(t3,1)\ \} \\
\delta(p5,1) & = & \phi(t4,1) \\
 & = & r(t1,1) + \max\{\ r(t2,1), r(t3,1)\ \} \\
\alpha(p1,2) & = & \phi(t4,1) \\
 & = & r(t1,1) + \max\{\ r(t2,1), r(t3,1)\ \} \\
\epsilon(t1,2) & = & \alpha(p1,2) \\
 & = & r(t1,1) + \max\{\ r(t2,1), r(t3,1)\ \} \\
\phi(t1,2) & = & \epsilon(t1,2) + r(t1,2) \\
 & = & \sum_{i=1}^{2} r(t1,i) + \max\{\ r(t2,1), r(t3,1)\ \} \\
\ldots & &
\end{array}
$$

The technique appears straightforward to apply, and in this particular case it is even easy to infer the general form of equations as a function of the index of the event, such as, e.g.:

$$\phi(t1,k) \quad = \quad \sum_{i=1}^{k} r(t1,i) + \sum_{i=1}^{k-1} \max\{\ r(t2,i), r(t3,i)\ \}$$

3.5.3 FIFO vs. general cases: scheduling functions

The situation becomes substantially more intriguing if we change the initial marking in the example in Figure 3.8, adding for instance one token in place $p2$. With such a modification of the initial marking, the form of equations remains simple in case the sequences of firing times are such that the FIFO ordering of tokens is preserved for departure from places. In the general case where a token that arrived late in a place can leave before other tokens that arrived earlier than it, the relations between instances of firing events and instances of token arrivals cannot be identity even for a Marked Graph net, and a function implementing time-stamp reordering must be introduced.

Let us consider in some details what happens in our modified example without the FIFO hypothesis. The initial conditions become in this case:
$\alpha(p1,1) = \alpha(p2,1) = 0$, and $\epsilon(t1,1) = \epsilon(t2,1) = 0$.
A subset of occurrence equations can be written in a way very similar to the first example:

$$
\begin{array}{lcl}
\phi(t1,1) & = & \epsilon(t1,1) + r(t1,1) \;=\; r(t1,1) \\
\delta(p1,1) & = & \phi(t1,1) \;=\; r(t1,1) \\
\alpha(p2,2) & = & \phi(t1,1) \;=\; r(t1,1) \\
\alpha(p3,1) & = & \phi(t1,1) \;=\; r(t1,1) \\
\epsilon(t2,2) & = & \alpha(p2,2) \;=\; r(t1,1) \\
\epsilon(t3,1) & = & \alpha(p3,1) \;=\; r(t1,1) \\
\phi(t3,1) & = & \epsilon(t3,1) + r(t3,1) \\
 & = & r(t1,1) + r(t3,1) \\
\delta(p3,1) & = & \phi(t3,1) \;=\; r(t1,1) + r(t3,1) \\
\alpha(p5,1) & = & \phi(t3,1) \;=\; r(t1,1) + r(t3,1)
\end{array}
$$

However, a problem arises for the firing of $t2$, that complicates the subsequent occurrence equations:

$$
\begin{array}{l}
\phi(t2,1) = \min(\; \epsilon(t2,1) + r(t2,1),\; \epsilon(t2,2) + r(t2,2) \;) \\
\qquad = \min(\; r(t2,1),\; r(t1,1) + r(t2,2) \;)
\end{array}
$$

Depending on the actual values of $r(t2,1)$ and $(r(t1,1)+r(t2,2))$, the first firing of $t2$ may thus be related either to the first or the second enabling instance of $t2$. Hence the scheduling of the firing events depends in a complex way on the firing times of $t1$ and $t2$ (as well as of $t3$ and $t4$ if we consider subsequent instances with higher index value).

Analogous problems arise in the case of conflicting transitions if the conflict is not solved according to a pre-selection policy (which will be defined later on) or in the case of places with more than one input transitions if these transitions are not in strong synchronic relation (i.e., if one of the transitions may fire indefinitely without the others being forced to fire). Hence occurrence equations not only are difficult to obtain in general cases, but are also difficult to evaluate efficiently.

We introduce then an auxiliary function that we call *scheduling firing* function:
$\forall t \in T, \quad \forall i \in \mathbb{N},$

$$\text{sched.f}(t,i) = n \in \mathbb{N} \text{ (instance number of enabling)}$$

so that we can always retrieve the number of the enabling occurrence that yields the i-th firing of the transition. This is helpful to define the relation between enabling and firing for a transition.

If we stick with the restriction of considering only net systems with persistent transitions, the relation between enabling and firing becomes:

$$\phi(t,i) = \epsilon(t, \text{sched.f}(t,i)) + r(t, \text{sched.f}(t,i))$$

where it is the correct definition of the scheduling function that guarantees the condition:
$$
\begin{array}{ll}
\epsilon(t, \text{sched.f}(t,i)) + r(t, \text{sched.f}(t,i)) & \leq \\
\epsilon(t, \text{sched.f}(t,i+1)) + r(t, \text{sched.f}(t,i+1)) &
\end{array}
$$
Notice that, in order to satisfy this scheduling ordering for firing, in general the function sched.f(t,i) *must* depend both on $\epsilon(t,j)$ and on $r(t,j)$, $\forall j \leq k$, with $k \geq i$ depending on the server semantics and the boundedness of the net system.

Hence, in general, the introduction of the scheduling function is a technique that allows one to concentrate all the complexity of the system behaviour in the scheduling function itself, simplifying the rest of the occurrence equations.

3.5.4 Conflicts and Collectors

Consider the distinction between timed and immediate transitions. In case of immediate transitions the equations may be non-FIFO only due to conflict situations, while in case of timed transitions, permutations of firing instances with respect to enabling instances may occur also in case of persistent transitions with enabling degree > 1 (as already illustrated in Section 3.5.3) if the firing delay is subject to decrements from an instance to the following one.

For the case of immediate transitions (in which problems arise only for conflicting transitions) we propose to switch the point of view used for the definition of events from the individual transitions to a more "global" one, namely the set of conflicting transitions. Extended conflicts sets (ECS) as defined in the GSPN framework can be considered instead of individual transitions (see definition in Section **??**). In our context the idea of CCS (as introduced in Section **??**) is sufficient to characterize the set of transitions that are potentially in conflict one with the other.

Only preselection policy is meaningful to define conflict resolution in case of immediate transitions, and an ECS (or CCS) represents the minimal environment that should be taken into account in order to define which one among a set of conflicting transitions is going to fire. Since the effective conflict relation is intransitive more than one immediate transition may be selected to fire from the subset of transitions enabled within a given ECS in a given marking. This type of conflict resolution may be specified by defining events of types:

1. arrival of tokens in one of the places that are input of some transitions in a given ECS;

2. decision to fire the i-th instance of a transition in the considered ECS after a given set of arrival events to the input places of the ECS.

Events of type 2 are directly determined by the choice policy that is specified in the model to resolve the conflict, so that we can define a set of auxiliary functions $\gamma_{\mathrm{ECS}}()$ for each conflicting immediate transition that we can assume known to the modeler:
$\gamma_{\mathrm{ECS}}(t, a1, a2, \ldots, a_m) = j > 0$ if after $a1$ arrivals in the first input place and $\ldots$ and a_m arrivals in the m-th input place, the j-th instance of transition t is chosen to fire;
$\gamma_{\mathrm{ECS}}(t, a1, a2, \ldots, a_m) = 0$ if after $a1$ arrivals in the first input place and $\ldots$ and a_m arrivals in the m-th input place, transition t is not chosen to fire, either because not enabled or because the conflict has been solved in favor of another transition of the same ECS.

Coming back to timed transition, in order to define the relation between enabling and firing, we have already to distinguish the selection and memory

policies underlying the observation of transition timing. For a transition that is assumed to act according to the *race with enabling memory* policy (i.e., such that in case of preemption a new activity is started, forgotting about the possibly interrupted activities), the relation is:

$$\phi(t,i) = \epsilon(t, \text{sched.f}(t,i)) + r(t, \text{sched.f}(t,i))$$

On the other hand, in case of a transition that is assumed to act according to the *race with reset memory* policy the relation is:

$$\phi(t,i) = \epsilon(t, \text{sched.f}(t,i)) + r(t,i)$$

The formal expression for *race with age memory* (also called *preemption-resume* — PRe) transitions is more elaborated, since it must take into account the firing instances of conflicting transitions. Remember that timed transitions with *preselection* policy can be reduced to the case of conflicting immediate transitions followed by persistent timed transitions, for which we already know how to write Occurrence equations.

3.6 Operational Analysis

Operational Analysis of Timed Petri Net models can be considered as a particular case of use of occurrence equations, in which the fine-grain events and their relations are exploited to derive some higher-level performance measures and exact relations among them. The basic principles of Operational Analysis of discrete event dynamic systems are:

- Define basic quantities that in principle are observable and measurable in an experiment running from time 0 to time θ;
- Define derived quantities in terms of basic ones;
- Prove simple algebraic relations among basic and derived quantities that hold true for any possible experiment performed using a model;
- Give a consistent physical interpretation to the operational quantities;

Hence, obtain simple algebraic relations among physical parameters of a model that are always valid.

We start by giving some basic definitions for operational quantities, and by showing how they could be derived from the Occurrence Equation approach. We use the following notation to provide numerical values to predicates:

$$\#[\,\text{any\_predicate}\,] \stackrel{\triangle}{=} \quad 1 \;\text{ if TRUE} \quad 0 \;\text{ otherwise}$$

All definitions are presented for P/T systems, the extension to coloured nets being straightforward.

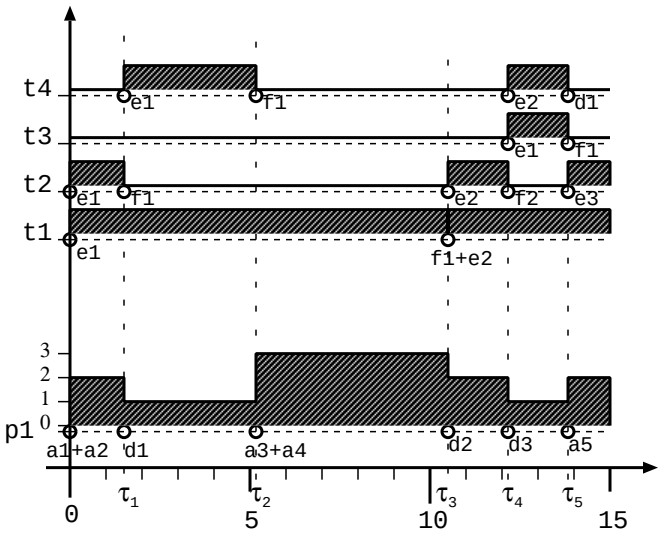


Figure 3.10: An occurrence time diagram and instantaneous marking for place $p1$ of a possible observation from time 0 to time 15.

Instantaneous Marking: $\forall p \in P$, $\forall$ instant τ of the experiment (i.e., such that $0 \leq \tau \leq \theta$)

$$\boldsymbol{\mu}[p](\tau) \triangleq \sum_{i=0}^{\infty} (\#[\,\alpha(p,i) \leq \tau\,] - \#[\,\delta(p,i) < \tau\,])$$

is the number of tokens present in place p at time τ. This definition is a straightforward mathematical way of expressing the current marking of a place p at a given time instant τ as the sum of the initial marking (by definition equal to $\sum_{i=0}^{\infty} \#[\alpha(p,i) = 0]$) plus the total number of tokens deposited by the firing of transitions occurring in the time interval $(0,\tau]$ (computed as $\sum_{i=0}^{\infty} \#[\,\alpha(p,i) > 0 \;\wedge\; \alpha(p,i) \leq \tau\,]$) minus the total number of tokens removed by the firing of transitions occurring in the time interval $(0,\tau)$ (computed as $\sum_{i=0}^{\infty} \#[\,\alpha(p,i) < \tau\,]$).

As an example, consider the time diagram depicted in Figure 3.10. This represents the occurrence of events that affect the marking of place $p1$ from time 0 to time 15 for the interval-timed model in Figure 3.2.a). Notice that at each time instant τ the definition of instantaneous marking corresponds the height of the dashed portion of the diagram representing the marking of $p1$.

Average Marking: $\forall p \in P$, if $\theta > 0$

$$\overline{\boldsymbol{\mu}}[p] \triangleq \frac{1}{\theta} \int_0^{\theta} \boldsymbol{\mu}[p](\tau)\, d\tau$$

With reference to the dashed portion of the diagram representing the marking of $p1$ in Figure 3.10, the definition of average marking computes the ratio between

the area of the dashed part of the diagram up to a certain time coordinate divided by the value of that time coordinate.

Instantaneous Enabling Degree: $\forall t \in T$ with at least one input place (i.e., such that ${}^\bullet t \neq \emptyset$),
$\forall$ instant τ of the experiment

$$ed_t(\tau) \triangleq \mathbf{e}(\boldsymbol{\mu}(\tau))[t] = \sup\{k \in \mathbb{N} : \forall p \in {}^\bullet t, \quad \boldsymbol{\mu}[p](\tau) \geq k \cdot pre_{pt}\}$$

The physical interpretation of the enabling degree is the number of servers that are currently active performing transition firing work. In the example depicted in Figure 3.10 the instantaneous value of the enabling degree for each transition is also represented by the height of the dashed diagram associated with the transition name at any instant within the observation interval. In this particular example the enabling degree for each transition is either 0 or 1 at any instant of time. From the diagram one can see that the enabling degree is 0 for transition $t2$ at time instants 2, 2.31415, and 13.001, while it is 1 for transition $t4$ at time instants 2, 3.217, and 4.5.

Total Enabling Work: $\forall t \in T$ such that ${}^\bullet t \neq \emptyset$)

$$\Theta_t \triangleq \int_0^\theta ed_t(\tau)\, d\tau$$

Physical interpretation: total active time for all servers associated with the transition firing during the experiment. In the example depicted in Figure 3.10 the total enabling work for each transition up to a given time instant corresponds to the area of the dashed portion of the diagram associated with that transition to the right with respect to the horizontal coordinate corresponding to the considered time instant.

Average Enabling Degree: $\forall t \in T$ such that ${}^\bullet t \neq \emptyset$), if $\theta > 0$

$$\overline{\mathbf{e}}[t] \triangleq \frac{\Theta_t}{\theta}$$

Physical interpretation: average number of active servers associated with the transition firing during the experiment.

Total Firing count: $\forall t \in T$, $\forall$ instant τ of the experiment

$$F_t(\tau) \triangleq \sum_{i=1}^{\infty} \#[\, \phi(t,i) < \tau \,]$$

is the total number of firings of t observed from 0 to τ. In the example depicted in Figure 3.10 the total firing count for the observation up to time instant 15 is 1 for transitions $t1$, $t3$ and $t4$ (the latter is enabled twice, but its enabling is preempted due to the race-type conflict with $t3$), and is 2 for transition $t2$.

Throughput: $\forall t \in T$, if $\theta > 0$

$$\boldsymbol{\chi}[t] \triangleq \frac{F_t(\theta)}{\theta}$$

Physical interpretation: actual firing frequency of transition t observed during the experiment. In the example depicted in Figure 3.10 the throughput is $1/15$ for transitions $t1$, $t3$ and $t4$ and is $2/15$ for transition $t2$, during the short observation time.

Average Firing Time: $\forall t \in T$ such that either t is persistent (i.e., never disabled before firing) or t has *Age Memory* policy, if $F_t(\theta) > 0$

$$\overline{S_t} \triangleq \frac{\Theta_t}{F_t(\theta)}$$

Physical interpretation: average time for a server to complete a firing during the experiment. the experiment.

As an example of an *operational law* for timed Petri nets we present the following:

Property 3.3 (Enabling Law) *$\forall t \in T$ with at least one input place and of type either persistent or age memory, $\forall$ conceivable experiment such that $\theta > 0$ and $F_t(\theta) > 0$*

$$\overline{\mathbf{e}}[t] = \boldsymbol{\chi}[t] \cdot \overline{S_t}$$

A formal proof for this property can be easily obtained by substitution of the definitions of the operational quantities and trivial algebraic simplifications.

This is the equivalent in Petri net terms of the classical "Utilization Law" in the domain of queueing networks for single-server stations: $U = X \cdot S$.

Several other useful operational laws will be presented in Chapter **??**.

3.7 Observation and Specification of Timed Models

With the introduction of Occurrence Equations and of Operational Analysis we addressed the problem of the observation of the behaviour of a Timed model in order to measure some of its performance related figures (for instance a particular transition's throughput). At the beginning of this chapter we addressed instead the problem of specifying the behaviour in time for a untimed, non interpreted Petri net system. The two aspects (specification and observation/computation of performance measures) are however strongly related, in the sense that measurements of operational quantities must be consistent with the timing interpretation/specification of the model.

The critical point in establishing a consistent relation between timing specification and performance measurements is the concept of transition enabling

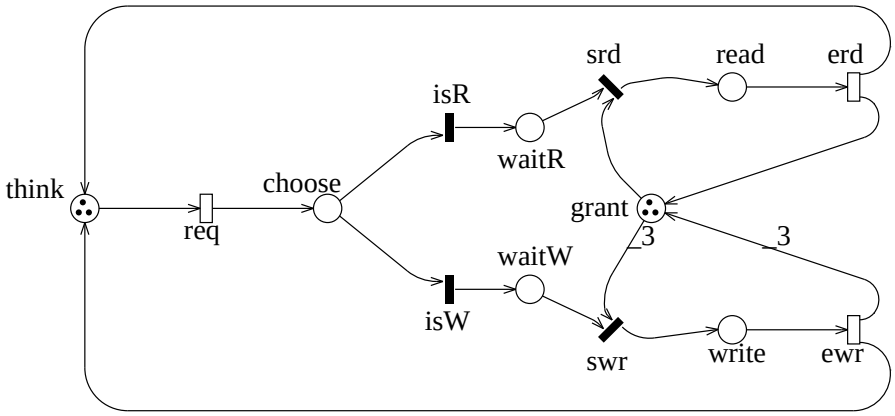


Figure 3.11: TTN model of the Concurrent Readers Exclusive Writers problem.

interval. Indeed, in case of a persistent, single-server transition $t \in T$ we have the consistency property:

$$\overline{S_t} = \frac{\sum_{i=1}^{F_t(\theta)} r(t,i)}{F_t(\theta)}$$

This consistency property allows us to identify the $\overline{S_t}$ measurement with the average specified firing delay for transition t. If we specify the transition enabling delay either deterministically or stochastically, then we can compute the operational quantity S_t without actually having to set up an "experimental measurement" of our model, and be able to apply operational laws to compute other performance measurements directly from the specification of our timed interpreted model.

At this point the ideas behind the use of Timed Petri Nets for performance measurements should be intuitively clear. Before turning our attention to the specification of timed models, let us present a motivating example.

3.7.1 The concurrent readers exclusive writers example

Example 3.1 CREW The problem is defined as a set of N customers that share the access to a single resource. Two access modes are defined: "read" and "write." Read accesses can be performed in parallel to other readers. Write accesses must be performed in mutual exclusion. A possible Timed Transition Net (TTN) representation of the access protocol is depicted in Figure 3.11.

The initial marking of place "think" contains one token per customer that needs access to the shared resource. Customers are indistinguishable one from

the other due to the lack of colour specification associated with place "think." The marking of place "choose" represents customers that are going to request access and are about to decide whether they need the read or write access mode. Such decision is taken by the firing of transitions "isR" or "isW": the two transitions are immediate and in free-choice relation; the preselection policy is defined according to the workload that one wants to represent; independent, identically distributed (iid) random choice could be a simple interpretation that is fully consistent with the non-determinism inherent to the non interpreted net. Instead immediate transitions "srd" and "swr" are in non-free choice conflict relation. In case both are enabled, an additional preselection policy specified by a proper interpretation may add constraints over the classical non deterministic choice of the underlying net model. If only one of the two transitions is enabled, then it will fire right away without further non determinism due to the priority structure.

The correct access protocol is implemented by means of place "grant": in the initial marking this place contains a number of tokens equal to the total number of customers; the firing of transition "srd" (start read) removes one token from place "grant" corresponding to the customer that is starting the read access; a token is put back in place "grant" by the firing of transition "erd" (end read); a write access requires the availability of all tokens initially in place "grant" (which implies that none of the customers is currently reading or writing), and the firing of "swr" (start write) removes all of them at once due to the arc weight; all access grants are restored by the firing of transition "ewr" (end write) due to the presence again of the same weight labeling the arc from "ewr" to "grant."

This example is not trivial to analyse by means of the Occurrence Equation techniques due to the following features:

1. the presence of a free-choice conflict between "isR" and "isW";
2. the presence of the non free-choice conflict between "srd" and "swr";
3. the presence of the "confluences" of tokens resulting from the firing of different transitions in non-FIFO order for places "think" and "grant" (the latter unless transitions "erd" and "ewr" are assumed to have identical and deterministic firing delay).

The model has however the following regularities that simplify the explicit writing of occurrence equations: conflicts are limited to immediate transitions, so that conflict resolution is always of type preselection and the chosen transition immediately fires upon the last token arrival in the input places; timed transitions are persistent. These regularities are sufficient conditions that allow us to assume the consistency between the measurement of the average transition firing times and their "real" firing delay. Moreover, the specification of a preselection policy at the level of immediate transition conflict sets allows us to derive switching functions of reasonable complexity, thus showing that a complete and correct stochastic interpretation can be found for this model, without

any further restrictive assumptions on the probability distributions for timed transitions' delay.

3.7.2 Operational observation of CREW

The operational approach allows us to define performance measures that are closely related to the interpretation of the physical system modelled by a timed Petri net.

For instance, transition "req" models the requests issued to access to the shared resource. Hence, his throughput χreq represents the total number of requests issued by customers per time unit. Since ${}^\bullet\text{req} = \{\text{think}\}$ and $pre_{\text{reqthink}} = 1$, we have that $\forall\tau\ ed_{\text{think}}(\tau) = \boldsymbol{\mu}[\text{think}](\tau)$, hence $\overline{\mathbf{e}}[\text{think}] = \overline{\boldsymbol{\mu}}[\text{think}]$. This allows us to interpret the average marking $\overline{\boldsymbol{\mu}}[\text{think}]$ as the average number of customers "thinking" about the issueing of a new access request.

By analogous reasons, we can easily find out that $\overline{\boldsymbol{\mu}}[\text{read}]$ is the average number of customers performing read access, and that their average access time is exactly the average firing delay of transition "erd." Similarly, we can convince ourselves that $\overline{\boldsymbol{\mu}}[\text{write}]$ is the average number of customers performing write access, and that their average access time is exactly the average firing delay of transition "ewr."

Moreover, the consistency property deriving from the fact that all timed transitions are persistent also guarantees the consistency between the average firing delay that we can measure during an experimental observation and the expected value of the firing delay in case of stochastic interpretation of the model. Hence, we can apply operational laws in order to compute performance measurements by using the specification of the model without having to resort to the observation of a real or simulated experiment involving the model behaviour.

3.8 Markov and semi-Markov cases

There are some particular cases for which the issues of memory, de-scheduling and re-scheduling policies simplify drastically with respect to the general case. This is the case called "memoryless," in which all different choices yield the same behaviour due to a particular property of the stochastic timing.

Using our Occurrence equation terminology, the memoryless property for a Markovian stochastic Petri net may be stated as follows:

Property 3.4 (absence of memory for the firing times) $\forall t \in T,\ \forall i > 0,$ $\forall\theta > 0,$

$$\forall\tau\ :\quad \epsilon(t,i) < \tau < \phi(t,i), \qquad \Pr\{\phi(t,i) \le \tau + \theta\} = \Pr\{r(t,i) \le \theta\}$$

This property is guaranteed by the assumption of random, independent, negative exponentially distributed firing times for all transition firing instances:
$\forall t \in T,\ \forall i > 0,$

$$r(t,i) = \chi_i \ \mid \forall\tau > 0, \left(\Pr\{\chi_i \le \tau\} = 1 - e^{-\mathcal{T}(t)\cdot\tau}\right)$$

3.8.1 Race policy simplifications

Due to the memoryless property of the exponential distribution, several simplifications may be introduced in the specification of a timed Markovian net system with respect to the general case. For example, in case of race among timed transitions age and enabling memory specify the same behaviour. This is not the case, however, for the reset memory policy, for which the distribution of a resumed firing time has not the same distribution as a fresh new instance of firing time (since we already know that the former has a duration greater than the largest interval of time for which the transition instance was enabled without firing).

Also the choice of the de-scheduling and re-scheduling disciplines in case of race with age memory or race with enabling memory policies becomes irrelevant under the hypothesis of Markovian transition firing times.

All these simplifications can be proven correct by manipulations of Occurrence equations exploiting Property 3.4.

3.8.2 Memoryless workloads

In some cases the memoryless assumption for some of the model parameters may induce substantial simplifications not only in the analysis of timed net models, but also in the structure of the underlying net system representation. Consider, for instance, The P/T system model of the PLC application depicted in Figure 2.1 in Chapter I.2. In this case, each token marking p_2 represents a processor that has finished its local computation phase, and is going to "decide" whether to access the common memory and continue the computation (by firing t_2) or to terminate its activity (by firing the free-choice transition t_5).

On the one hand, the non determinism of the choice naturally suggests an interpretation of iid random choice in order to convert the P/T system into a performance model. However, an i.i.d. assumption implies a memoryless distribution (actually, a geometrically distributed discrete mass function) for the number of memory accesses before task completion.

On the other hand, the interpretation of a non-memoryless distribution for the number of memory accesses before task completion appears impossible in general for this P/T system. Unless all transition firing delays are assumed to be deterministically fixed with infinite server semantics, place p_2 may contain more than one indistinguishable tokens modelling processors that have already performed different numbers of memory accesses, so that it would be impossible to relate the choice policy to the number of memory accesses already performed. For instance, a workload of the type "an equally likely number of memory accesses in the range from 3 and 5" would be impossible to specify as an interpretation for the choice policy if the firing delay of t_1 is assumed to be exponentially distributed. The specification of such a workload would require a different underlying net system (e.g., a coloured net) in which at least the different processor identities are kept. In this case, a time-dependent sequence of random choices might be assumed as the interpretation of the net system in

order to solve at least the specification problem (the analysis problem would still be very hard to attack due to the time-dependency that we would have to assume).

This example shows how the memoryless property can simplify not only the analysis but also the specification of a timed net model. A model that would be wrong in the general case due to an oversimplification of its state representation may be converted into a correct model if we restrict to the subclass of memoryless distributions.

3.8.3 Marking Dependency

The memoryless property may suggest a clever way of abstracting some of the details of a system model representation while keeping a performance model accurate. This is the idea of "equivalent server" taken from queueing models.

If a single class queue enjoys the memoryless property, then all the information about its present state can be summarized in a single number that counts the total number of customers inside. A detailed characterization of its behaviour may be avoided provided that for each value that the load parameter can assume, we specify the appropriate service rate, i.e., the speed of departure of the first customer. The departure of a customer will change the load parameter, therefore changing also the rate of departure of the next customer (if any). Also the arrival of a new customer will change the load parameter, inducing the selection of a different departure rate.

One way of exploiting this feature of memoryless queues is to perform hierarchical analysis, by first computing the load-dependent departure rates based on a detailed model of a single, complex queue. Then the queue is substituted by its "equivalent server" in a more abstract model, that takes into account the interaction of the queue whose behaviour is being abstracted with other queues (or even the "equivalent servers" of other queues).

In the framework of a Petri net, if the memoryless property holds then the complete state of the model can be summarized by just specifying the marking in each place. Following the same principle, one can thus substitute the explicit, detailed representation of a memoryless submodel with exponentially distributed transitions firing delays that are a function of the current marking. Notice, however, that the complexity of such a representation explodes in general, as the number of combinations of tokens distributed among the places of a subnet does (each combination requires its own specification of firing rate in order to be equivalent to the original memoryless subnet).

In some specific cases, however, the adoption of marking-dependent firing rates for transitions in a Markovian Petri net may induce substantial reductions in the model complexity with respect to a detailed representation of the mechanisms of the system to be modelled. Usually such "clever" exploitation of marking-dependent rates depend on the knowledge of the modeller of some intrinsic regularity and/or symmetry in the model behaviour. Such feature, though useful in some particular cases, should be exploited with great care and self-control by experienced users of the formalism. In general, marking-

dependent firing rates for transitions create dependencies among events that would not be related according to the analysis of the structure of the underlying net system, thus hampering one to exploit one of the fundamental characteristics of well-defined timed Petri nets, namely the orthogonality of the timing interpretation with respect to the causal properties of the Petri net model.

3.9 Generalized Stochastic Petri Nets (GSPNs)

As a very popular example of timed Petri net, we introduce now the definition of GSPNs. In this case the reduction of the non determinism is stochastic, with the additional restriction of considering only exponentially distributed firing times for timed transitions. As already discussed in this section, this choice has strong impact on the definition of the timing semantics, drastically simplifying the definition of a race model for timed transitions.

Not having to bother about the distinction between Age and Enabling memory, GSPNs allow conflicts for timed transitions without any restriction (free-choice or not) with a purely Race semantics. The natural choice for disabling policy in case of multiple-server transitions is of course Random, which is perfectly consistent with the memoryless property of all transition timings. In the early definition of GSPNs general marking dependency was allowed to define the firing rate parameter for each timed transition. In the revised definition only enabling dependence was allowed in order to force the modeller to increase the accuracy of the underlying untimed net representation for what concerns the explicit modelling of activities in terms of active transition servers.

Immediate transition are also allowed in GSPNs, and in case of conflict an iid random choice is assumed to define a firing order among so called "extended conflict sets." This definition implies a correctly defined preselection policy among conflicting immediate transitions. GSPNs imply the use of a consistent priority structure for transitions, in which all timed transitions have the lowest priority level, while immediate transitions are defined to have priority greater than timed transitions. The use of different priority levels for different transitions allows one to define deterministic "conflict" resolution that overcome the stochastic nature of conflict resolution within extended conflict sets. The combination of deterministic and stochastic preselection rules for the solution of conflicts between immediate transition is regulated by some correctness criteria (the subnets of immediate transitions must be "confusion-free") that provide sufficient conditions to make sure that the behaviour of the model is completely and correctly specified at the net level, without having necessarily to study the sequences of transition firings before defining the choice probabilities.

The modelling power and convenience of GSPNs for the representation of complex behaviours is also increased by the adoption of an extended definition for the underlying net system that includes multiple inhibitor arcs as well as multiple level priorities for immediate transitions. Figure 3.12 contains an example of a GSPN model of the CREW example already introduced. In this case priority is given to "writers" in case the resource is free and both readers

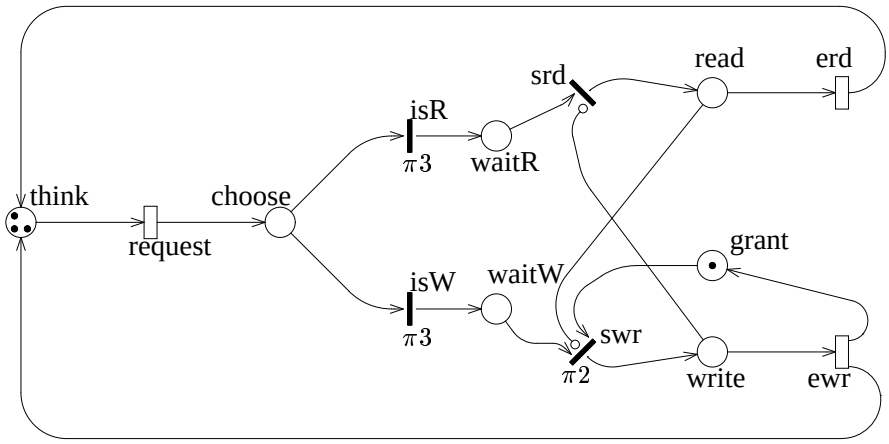


Figure 3.12: GSPN model of the Concurrent Readers Exclusive Writers problem with Writers' priority.

and writers are requesting it (this feature is due to the priority specification 2 associated with "swr" as compared to the default priority 1 associated with "srd").

The mutual exclusion is in this case obtained by disabling "swr" by means of an inhibitor arc of weight 1 in all markings containing at least one token in place "read". Transition "srd" is disabled by means of another inhibitor arc of weight 1 in all markings containing a token in place "write." Finally, transitions "srd" can never fire before or at the same time as "swd" in case they are both enabled, due to the strict order determined by the priority specification. Write accesses are serialized one at a time by means of the place invariant composed by "grant" and "write" that always contains exactly one token as in the initial marking.

Notice that in this way, the model is easier to modify in order to represent a different number of customers instead of 3. In the GSPN model depicted in Figure 3.12 only the initial marking of place "think" has to be changed for this purpose, while the original model presented in Figure 3.11 would require the consistent change of the markings of places "think" and "grant" and of the weight of the two arcs connecting place "grant" with transitions "swr" and "ewr."

Concerning the stochastic interpretation, a single numerical parameter has to be associated with each transition: the firing rate associated with each (exponentially distributed, infinite-server) timed transition, and a firing weight associated with each immediate transition that could be enabled in conflict with other transitions at the same priority level. In our example in Figure 3.12 this speci-

fication is obtained by providing three rate parameters, respectively associated with transitions "request," "erd," and "ewr," and two weight parameters associated with transitions "isR" and "isW." Due to the free-choice type of conflict between "isR" and "isW," the random choice for conflict resolution is always decided in favour of one transition with probability equal to the weight of that transition divided by the sum of the weights of the two conflicting transitions. The other two immediate transitions are never in conflict with others due to the peculiar priority structure of the model, so that they always immediately fire as soon as enabled, according to the rules defined for the underlying untimed model. The fact that the model is structurally confusion-free according to the new GSPN definition is a sufficient condition to conclude that such probability specification assigned in terms of relative weights within conflict sets is complete and correct.

3.10 Stochastic Well-Formed Nets (SWNs)

Another popular example of timed Petri net is a coloured net extension of GSPNs called Stochastic Well-Formed Nets. The idea that led to the introduction of this model was to extend the GSPN timing semantics so as to fit in the framework of an higher-level net model. The principal motivation for the introduction of SWNs was the exploitation of model symmetries that can be easily expressed in a concise way using Well-Formed net systems instead of P/T net systems, not only to draw more compact net models but also to reduce the complexity of stochastic models' analysis. From the time semantics specification point of view, an SWN model is completely specified by assigning a single numerical rate or weight parameter to each enabled colour instance of each transition. In order to preserve symmetry properties also in the time behaviour, the rate or weight parameter is assumed to be identical for all colour instances belonging to the same static colour subclass.

3.11 Deterministic and Stochastic Petri Nets (DSPNs)

This model was also introduced as an extension of the popular GSPN model. The motivation was to introduce a mixture of two different timing mechanisms for transitions in the same model: stochastic transitions with memoryless distribution for firing delays and deterministic transitions with fixed firing delay.

According to our taxonomy of non-determinism reduction, this is a mixed model in which complete deterministic reduction is assumed for some transitions while stochastic reduction is assumed for other transitions. In principle, an arbitrary definition of the subset of deterministic timed transitions could have a strong impact on the qualitative behaviour of the interpreted model as compared to the qualitative behaviour of the underlying net system. However, the definition of DSPN was introduced in such a way so as to avoid this kind of mismatch between the interpreted and the uninterpreted models.

A sufficient condition that guarantees the qualitative consistency in the behaviour after the introduction of the interpretation, that was assumed as part of the definition of DSPNs, is that the deterministic time interpretation is allowed only for single-server transitions that are never enabled concurrently with other deterministically timed transitions. Indeed this implies that, whenever a deterministic timed transition is enabled, all other enabled timed transitions behave according to the stochastic interpretation with infinite support probability distribution. Hence, whatever value is assigned as a fixed firing time for such deterministic transition (provided that it is greater than zero and less than infinity), each other timed transition will have a chance to fire before it with non-null probability, and a chance to fire after it with non-null probability. Therefore all firing interleavings are still allowed, as in the case of the uninterpreted, non deterministic net system.

Deterministically timed transitions must be further specified in terms of memory policy (Age or Enabling, the latter being equivalent to Reset due to the lack of variability in the firing delay). Such specification can be avoided only in case of persistent transitions. Due to problems of complexity in the mathematical study of the model, the original definition of DSPN allowed only Enabling memory policy associated with deterministic transitions, although from the specification point of view Age memory would not pose any additional problem.

As an example, we can consider again the GSPN model of the CREW system depicted in Figure 3.12 and notice that in this case only timed transition "ewr" satisfies the above condition (indeed it is the only one of single-server type, thanks to the property of place "write" to be safe). Such model can thus be transformed into a DSPN by simply changing the time interpretation for transition "ewr" to be deterministically timed. In this case the deterministic transition is persistent, so that we don't need to specify its memory policy.

3.12 Concluding Remarks

Our main concern in this chapter was to identify ways to introduce the concept of time within the Petri net formalism so as to satisfy the following requirements:

- be consistent with the definition and basic properties of untimed Petri net models
- define timing aspects consistent with our intuition about observation of dynamic systems and prototype measurements.

We approached the problem by first identifying the non determinism and different ways of reducing it in the underlying untimed Petri net model. Then we exploited techniques based on occurrence equations and operational analysis to relate Petri net models to the ideas of observation and measurement.

Occurrence Equations provide a clear semantics for the definition of the model behaviour taking the time specification into account. Starting from occurrence equations instead of the reachability graph (see further chapters) the

limitations of the Markovian approach can be overcome in a conceptually easy way, at least for what concerns the specification of the model behaviour. Trying to derive occurrence equations from a net system structure is a good way of identifying possible problems in the definition of a complete and correct timing interpretation for the model: not being able to correctly derive occurrence equations is usually a symptom of a lack of correspondence between the net system behaviour and the behaviour of the system that we want to model.

Some difficulties in defining a proper timing semantics for a given net system are apparently overcome by choosing a restricted modelling framework, such as for example assuming stochastic timing with memoryless properties. Indeed some net systems for which we cannot find a consistent time interpretation in the general case allow a consistent time interpretation using, e.g., the GSPN or the SWN interpretation. Such models are usually "abstract views" of the system operation that ignore a number of "details" of the real system behaviour, providing only a superficial model for it. Such simplified models are usually very useful to an experienced modeller in order to come to some rough "order of magnitude" ideas about the performance characteristics of a very complex system, without having to draw and analyse very complex and detailed models. However, one must be aware of the "simplifications" that were introduced in the model specification in order to be able to critically accept or reject some of the results obtained from the observation (simulation) or mathematical analysis of the model, depending on the impact that such simplifications may have on the performance characteristics to be studied.

The specification of particular queueing disciplines can affect the measurable behaviour of the model only when prescribing a choice policy among distinguishible tokens, such as tokens of different colours and/or tokens residing in different places. In particular, the choice of a particular queueing discipline will *not* affect any kind of measure that one can imagine to perform on the model according to the operational approach described in Section 3.6. Such indication can also be usefully exploited by experienced modellers in order to reduce the model complexity, for instance avoiding an explicit net representation of some complex queueing discipline in case such precise specification cannot affect the performance characteristics one is interested in.

The Operational approach is very useful in this setting to clarify and define performance measures in terms that are intuitive, and related to the measurement approach that is normally used in physics. The definition of average firing time, for instance, provides an intuive way of figuring out the meaning of the firing delay parameter of a transition, even starting from a non interpreted net system.

A second advantage of the Operational approach consists in defining "average" measures without necessarily resorting to statistics and probability theory. Our operational averages need not necessarily converge to any theoretical "expected" value, neither in transient nor in steady-state. The operational laws that can be derived from the analysis of the model structure are valid no matter the statistical characterization that the quantities might have: in particular the relations among "average" measures are valid for any probability distributions

and/or "higher moments" we might assume for them, thus providing a very robust framework for performance experiments.

3.13 Bibliographic Remarks

The early attempts to define consistent time interpretation for Petri net systems focused on deterministic approaches mainly target at performance evaluation [1, 2]. However this approach was not pursued very deeply because of some undecidability and complexity results [3, 4] that apparently prevented the development of automatic analysis tools for general net system. Essentially only Marked Graph structures were thoroughly investigated with this time interpretation approach [5].

An approach based on incomplete reduction of the non determinism by definition of ranges for firing time was proposed by Merlin and Farber [6]. Efficient ways for generating reachable states where proposed and implemented [7], but this approach was only followed for validation of logical properties, not for performance evaluation.

Stochastic interpretation for Petri nets where initially introduced in conjunction with memoryless probability distributions [8, 9, 10, 11]. Several extensions were then introduced [12, 13, 14, 15], but it was only in [16, 17] that the problem of a clear semantics for the time interpretation for non-exponential distributions started to be fully understood. The semantics of immediate transitions and its relation to transition priority was clarified in [18, 19].

An extensive literature has been devoted to DSPNs, that originated in [20, 21]. Several extensions have been proposed later on [22, 23, 24], as well as several applications [25, 26].

Several coloured extensions were proposed for the basic GSPN model, including [27, 28, 29, 30, 31]. Stochastic Well-formed nets were eventually defined [32] to exploit an efficient analysis algorithm based on the Symbolic Reachability Graph [33, 34]. The decolourization technique for SWN was introduced in [35, 36]. Although limited in its applicability to cases of strong symmetry, it allows one to establish in a formal and clean way different "points of view" on a given system, at different levels of "abstraction" with respect to its details in terms of identities of the parts that constitute the system.

Baccelli et al. first proposed a technique for the description of the timed behaviour of Petri net models that was called "recursive equations" [37, 38, 39, 40]. The material on occurrence equations presented in this chapter is derived from the paper [41], where a straightforward generalization of the notation was introduced to allow the description of the behaviour of well-formed net models. Some of the developments of the technique have been performed in the framework of the Esprit BRA Project No.7269 "QMIPS" [42]. In our framework, the FIFO requirement that was posed in previous works on this subject (e.g, [38, 39]) is more a convenience to simplify the derivation of the occurrence equations in algorithmic way rather than a theoretical limitation of the approach.

Operational analysis for timed Petri nets was proposed in [43, 44]. His-

torically, operational analysis approaches were developed independently of the study of occurrence equations. However, a-posteriori the operational approach may be interpreted as a particular case of exploitation of occurrence equations. In the operational approach the fine grain behaviour at the level of individual event occurrences is studied for once, in order to develop operational laws relating higher level performance measures. Once an operational law has been proven correct, its formulation may be used as it is on any net system on which the performance measures may be defined, without having to resort to the fine grain analysis any more.

Bibliography

[1] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed Petri Nets.* PhD thesis, MIT, Cambridge, MA, 1974. Ph.D. Thesis.

[2] C.V. Ramamoorthy and G.S. Ho. Performance evaluation of asynchronous concurrent systems using Petri nets. *IEEE Transactions on Software Engineering*, 6(5):440–449, September 1980.

[3] M. Hack. *Decidability Questions for Petri Nets.* PhD thesis, M.I.T., Cambridge, MA, December 1975. also tech. report 161, Lab. for Computer Science, June 1976.

[4] N.D. Jones, L.H. Landweber, and Y.E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299, 1977.

[5] J. Campos, G. Chiola, J.M. Colom, and M. Silva. Properties and performance bounds for timed marked graphs. *IEEE Transactions on Circuits and Systems—1: Fundamental Theory and Applications*, 39(5):386–401, May 1992.

[6] P.M. Merlin and D.J. Farber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Transactions on Communications*, 24(9):1036–1043, September 1976.

[7] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Enginnering*, 17(3):259–273, March 1991.

[8] S. Natkin. *Les Reseaux de Petri Stochastiques et leur Application a l'Evaluation des Systemes Informatiques.* PhD thesis, CNAM, Paris, France, 1980. These de Docteur Ingegneur.

[9] M.K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models.* PhD thesis, UCLA, Los Angeles, CA, 1981. Ph.D. Thesis.

[10] M.K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transaction on Computers*, 31(9):913–917, September 1982.

[11] G. Florin and S. Natkin. Les reseaux de Petri stochastiques. *Technique et Science Informatiques*, 4(1), February 1985.

[12] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.

[13] J.B. Dugan, K.S. Trivedi, R.M. Geist, and V.F. Nicola. Extended stochastic Petri nets: Applications and analysis. In *Proc. PERFORMANCE '84*, Paris, France, December 1984.

[14] M.A. Holliday and M.K. Vernon. A generalized timed Petri net model for performance analysis. In *Proc. Intern. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.

[15] M.K. Vernon and M.A. Holliday. Performance analysis of multiprocessor cache consistency protocols using generalized timed Petri nets. In *Proc. Performance '86 and 1986 SIGMETRICS Joint Conference*, pages 9–17, Raleigh, NC, USA, May 1986. ACM.

[16] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. On Petri nets with stochastic timing. In *Proc. Intern. Workshop on Timed Petri Nets*, pages 80–87, Torino,Italy, July 1985. IEEE-CS Press.

[17] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Transactions on Software Engineering*, 15(7):832–846, July 1989.

[18] M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte. Generalized stochastic Petri nets revisited: Random switches and priorities. In *Proc. Intern. Workshop on Petri Nets and Performance Models*, pages 44–53, Madison, WI, USA, August 1987. IEEE-CS Press.

[19] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, February 1993.

[20] M. Ajmone Marsan and G. Chiola. *On Petri Nets with Deterministic and Exponentially Distributed Firing Times*, volume 266 of *LNCS*, pages 132–145. Springer Verlag, 1987.

[21] M. Ajmone Marsan, G. Chiola, and A. Fumagalli. Improving the efficiency of the analysis of DSPN models. In G. Rozenberg, editor, *Advances on Petri Nets '89*, number 424 in LNCS. Springer Verlag, 1990.

[22] C. Lindemann. An improved numerical algorithm for calculating steady-state solutions of Deterministic and Stochastic Petri net models. In *Proc. 4^{th} Intern. Workshop on Petri Nets and Performance Models*, pages 176,185, Melbourne, Australia, December 1991. IEEE-CS Press.

[23] R. German and C. Lindemann. Analysis of stochastic Petri nets by the method of supplementary variables. In *Proc. of Performance 93*, Rome, Italy, September 1993.

[24] G. Ciardo and C. Lindemann. Analysis of deterministic and stochastic Petri nets. In *Proc. 5^{th} Intern. Workshop on Petri Nets and Performance Models*, pages 160–169, Toulouse, France, October 1993. IEEE-CS Press.

[25] M. Ajmone Marsan, G. Chiola, and A. Fumagalli. Timed Petri net model for accurate performance analysis of CSMA/CD bus LAN. *Computer Communications*, 10(6):304–312, December 1987.

[26] M. Lu, D. Zhang, and T. Murata. Analysis of self-stabilizing clock synchronization by means of stochastic Petri nets. *IEEE Trans. on Computers*, 39:597–604, 1990.

[27] A. Zenie. Colored stochastic Petri nets. In *Proc. Intern. Workshop on Timed Petri Nets*, pages 262–271, Torino, Italy, July 1985. IEEE-CS Press.

[28] C. Lin and D.C. Marinescu. Stochastic high level Petri nets and applications. *IEEE Transactions on Computers*, 37(7):815–825, July 1988.

[29] G. Chiola, G. Bruno, and T. Demaria. Introducing a color formalism into generalized stochastic Petri nets. In *Proc. 9^{th} Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[30] J.A. Carrasco. Automated construction of compound Markov chains from generalized stochastic high-level Petri nets. In *Proc. 3^{rd} Intern. Workshop on Petri Nets and Performance Models*, pages 93–102, Kyoto, Japan, December 1989. IEEE-CS Press.

[31] C. Dutheillet and S. Haddad. Regular stochastic Petri nets. In *Proc. 10th Intern. Conf. Application and Theory of Petri Nets*, Bonn, Germany, June 1989.

[32] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11), November 1993.

[33] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed coloured nets and their symbolic reachability graph. In *Proc. 11^{th} Intern. Conference on Application and Theory of Petri Nets*, Paris, France, June 1990. Reprinted in *High-Level Petri Nets. Theory and Application*, K. Jensen and G. Rozenberg (editors), Springer Verlag, 1991.

[34] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. A symbolic reachability grapgh for coloured Petri nets. *Theoretical Computer Science*, 176:39–65, 1997.

[35] G. Chiola and G. Franceschinis. A condition for behaviour-preserving colour simplification in Well-Formed coloured nets. In *Proc. ISCAS'91*, Singapore, June 1991. Special session on Net-based analysis and Design of Concurrent Distributed Event Systems.

[36] G. Chiola and G. Franceschinis. A structural colour simplification in Well-Formed coloured nets. In *Proc. 4^{th} Intern. Workshop on Petri Nets and Performance Models*, pages 144–153, Melbourne, Australia, December 1991. IEEE-CS Press.

[37] F. Baccelli. Ergodic theory of stochastic Petri networks. *Ann. Probab.*, 20:375–398, 1992.

[38] F. Baccelli and Z. Liu. Comparison properties of stochastic decision free Petri nets. *IEEE Transactions on Automatic Control*, 37(12), December 1992.

[39] F. Baccelli, G. Cohen, and B. Gaujal. Recursive equations and basic properties of timed Petri nets. *Discrete Event Dynamic Systems: Theory and Applications*, 1:415–439, 1992.

[40] F. Baccelli, S. Foss, and B. Gaujal. Structural, timed and stochastic properties of free-choice nets. Technical Report 2411, INRIA, Sophia Antipolis, France, November 1994.

[41] G. Chiola. Characterization of timed well-formed Petri nets behavior by means of occurrence equations. In *Proc. 6^{th} Intern. Workshop on Petri Nets and Performance Models*, Durham, NC, USA, October 1995.

[42] F. Baccelli, A. Jean-Marie, and I. Mitrani, editors. *Quantitative Methods in Parallel Systems*. Esprit Basic Research Series. Springer Verlag, 1995.

[43] G. Chiola, J. Campos, J.M. Colom, and M. Silva. Operational analysis of timed Petri nets. In *Proc. PERFORMANCE '93*, Roma, Italy, September 1993. extended abstract.

[44] G. Chiola, C. Anglano, J. Campos, J.M. Colom, and M. Silva. Operational analysis of timed Petri nets and applications to the computation of performance bounds. In *Proc. 5^{th} Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

Chapter 4

Petri Nets versus Queueing Networks: Similarities and Differences

This presentation is a first contribution to stimulate a debate among the partners on a "fair comparison" between the timed Petri net and the queueing network formalisms. Firstly graphic models are proposed to be classified according to their balance between the model structure and the model interpretation. Then some considerations are made on the trade off between modelling and analysis power of different formalisms. We then argue that Petri net formalisms allow more freedom in the choices regarding these trade offs than previous formalisms like discrete space Markov chains and (either BCMP or extended) queueing networks. Finally some considerations are made on coloured Petri nets that could suggest them as a superior modelling formalism.

4.1 Introduction: how can we compare formalisms?

During the last ten years a large number of proposals have been published for the utilization of Petri net based formalisms for performance evaluation. Most of these proposals are substantially different the one from the other from several points of view, the only unifying thing being probably the fact that all of them introduce in some way the concept of time that was purposely avoided in the original definition of Petri nets. Hence the term "timed Petri net" has assumed a broad sense to indicate a whole class of models rather than a particular choice proposed by a particular author. Moreover, so far each author has kept using his/her own developed version of the "timed Petri net" formalism rather than attempting any effort of unification with other slightly different proposals. Part of this lack of unification is probably due to the different modelling requirements raising from different application domains and from different cultural

backgrounds rather than from deliberate vicious attitude. This state of facts is however quite deleterious when attempting a survey of the characteristics of the "timed Petri net" formalism, since the statements that one would consider to hold true for a particular proposal may be completely unapplicable or false when applied to another (sometimes "only slightly") different formalism.

The situation is not substantially better when we try and survey queueing network models. BCMP queueing networks are sometimes assumed by default when the term queueing network is used. However many "extensions" have been proposed (and used) to increase the modelling power for the representation of concurrent systems with synchronizations and other "non product form" features. Sometimes special purpose analysis techniques are associated with these extended models; most of the times they are simply used as a more or less convenient representation for simulation models.

To the best of our knowledge the only attempt that has been published so far to make a comparison between timed Petri nets and (extended) queueing networks is the one by Vernon, Zahorjan, and Lazowska [1]. An example of the questions that they pose when attempting such a comparison is outlined in the following table:

Which QN models?
BCMP or extended (blocking, passive resources, fork-join, etc.)?
Single class or multiclass?
Which TPN models?
SPN with exponential distributions?
GSPN with immediate transitions?
Deterministic time?
ESPN with arbitrary distribution?
Basic P/T nets or arbitrarily extended (priority, inhibitor arcs, marking-dependent arcs, activity networks)?
Which analysis methods?
Discrete event simulation?
Exact numerical analysis?
Approximate numerical analysis?
Bounds?
Structural analysis and qualitative validation?

They tried to be as "fair" as possible by attempting a broad comparison between the two classes of models rather then sticking with one particular choice for each class. The results are of course fuzzy: they very generically suggest a substantial equivalence between the two classes of formalisms, the choice depending essentially on the "taste" of the modeller.

Such a conclusion was more or less convincing from a methodological point of view, but in our opinion had no practical impact. Indeed such a substantial equivalence cannot be established for any pair of actually proposed formalisms (supported by the relative software tools). It could only be taken as an indication that it is conceivable to define new proposals in one class that could be arbitrarily

Table 4.1: Comparison Summary: Similarities

TPNs	QNs
flow of tokens	flow of customers
abstract modelling	abstract modelling
parametric marking	parametric population
operational analysis	operational analysis
performance bounds	bottleneck and asymptotic analysis
flow equivalent servers	flow equivalent servers
surrogate delays	surrogate delays
discrete event simulation	discrete event simulation

Table 4.2: Comparison Summary: Differences

TPNs	QNs
detailed modelling possible	only abstract modelling
no special semantics	predefined semantics
state space analysis	MVA or convolution for PF
need behavioural validation	properties guaranteed for PF
non-monotonic behaviour	monotonic for PF
non parametric systems	population always parametric

close to an existing one in the other class.

In Tables 4.1 and 4.2 we sketch a summary of comparison using the same idea of the above paper [1] but with a different purpose. Instead of looking for some kind of "fair objective measures" to compare proposed formalisms we aim at identifying some of the strong and weak points of such formalisms with the idea of proposing yet another formalism that overcomes the difficulties.

4.2 Towards a practically useful PN formalism

Petri net based formalisms have been introduced in response to a real need in the domain of performance evaluation of parallel and distributed systems, where there is no question about the modelling need for constructs that are not included in BCMP QNs. The overwhelming number of timed Petri net (TPN) formalism variations historically derives from:

- the large flexibility of the PN formalism itself, that may encourage different reinterpretations and extensions;
- the use of the formalism within different application domains;
- the choice of different modelling and analysis methods and techniques.

None of the proposed TPN formalisms was specifically designed as a general purpose formalism to *substitute* QNs. Instead, some of them were designed to *complement* QN descriptions for large and complex applications [2, 3].

A better, unified formalism for TPNs is not necessarily a superset of the more powerful proposed ones. On the contrary some restrictions could be imposed in order to improve the analysis power of the model, provided that enough modelling power is retained to effectively cover several practical applications. Indeed a principle that is often accepted by researchers working in the field of modelling and evaluating complex systems is to maintain a coherence between the timing specification and the "qualitative properties" of the underlying, untimed Petri net model in order to exploit results and structural analysis algorithms not only for validation but also for performance evaluation [4]. A few examples of simultaneous use of the same Petri net model for validation and performance evaluation have already appeared in the literature (see, e.g., [5]). Examples of direct exploitation of structural analysis results for performance evaluation are also starting to appear in the literature (see, e.g., [6, 7, 8]).

BCMP QNs with associated MVA or convolution algorithms are examples of a modelling formalism that exhibits a very good trade-off between modelling and analysis power. Currently used TPN formalisms are instead still in a early stage of development and do not yet necessarily exhibit the best trade-off in this sense. A direct comparison of a generic TPN formalism with BCMP QNs would be sterile if we limited ourselves to the consideration that the former are more powerful from the modelling point of view while the latter can be analyzed more efficiently. What we really need is the development of new, efficient analysis techniques that can be applied to realistic problems naturally modelled by TPNs. One way of going into this direction is to start by considering restrictions of TPNs that allow efficient (structural) analysis, even if with such constraints they do not cover the generality of cases that one would like to model. Hopefully, starting from PNs in which synchronization is one of the basic constructs, one may hope to come up with a formalism that covers cases partially disjoint from those covered by BCMP QN models.

In order to identify such an ideal TPN formalism that could really complement BCMP QNs we propose to step back to the idea of a graphic performance model, and then try to identify the requirements for such formalisms.

4.2.1 Graphic Performance Models

We propose the following decomposition of a graphic performance modelling formalism in three main parts:

- Nodes (in graph terminology) that represent objects or entities or activities taken into account by the model (e.g., customers, resources, servers, inter-request delays, etc.)

- Arcs that represent interactions or relations or routing among "things" represented by nodes.

- (explicit or implicit) "interpretations" that are added to the graph representation in order to complete the specification of the model behaviour. Examples of such interpretations are:
  - the queueing discipline (FCFS, PS, IS, etc.) that is explicitly added to each queue representation in a BCMP QN model, while the state-independent probabilistic routing is implicitly assumed to hold in each model of this class;
  - the exponential probability distribution that is implicitly assumed for firing times in a SPN model, while the value of the firing rate is explicitly written next to each transition.

A given feature of a system to be modelled may be represented either structurally by means of nodes and arcs, or implicitly through an appropriate interpretation. One should never forget that, independently of the chosen formalism, a *complete* model description must be given in any case if we want to compute performance indexes that depend on the system characteristics that we are considering. Therefore, general purpose modelling formalisms must be designed in such a way that most (hopefully all) interesting features may be represented either through appropriate graphic structures or by proper specification of interpretations.

The choice of what to represent explicitly in graphic form and of what to describe by means of interpretations has several implications on the modelling as well as on the analysis power of the formalism.

Once the level of details of the system to be modelled is decided, if a general purpose formalism includes the possibility of a sophisticate interpretation the structure of a given model may be kept simple and small. If instead a general purpose formalism does not provide a given feature to be modelled at the interpretation level, then the model structure may become quite cumbersome in order to explicitly represent that feature. Special purpose formalisms dedicated to a restricted class of models may provide very powerful interpretations (for that specific domain) thus substantially reducing the required structural complexity of models belonging to the target class of applications. Examples of relatively large domains for which one would be willing to develop special purpose formalisms include, e.g., FMS cells, OSI-like protocols, and so on.

From the model analysis point of view, algorithms must take into account the complete model definition as well. However the distinction between graph complexity and interpretation affects the way in which analysis algorithms are defined. The algorithms must be prepared to handle all syntactically correct graph structures, while different algorithms may be invoked for different interpretations of the same model structure. Examples of the relation between interpretation and analysis algorithms include the following cases:

- BCMP restrictions on queueing disciplines that allow the use of MVA or Convolution algorithms for the solution of QN models. If some of the queues do not conform with the restrictions of the BCMP theorem,

standard Markov chain analysis can be applied (with worse time and space complexity).

- Exponential distribution assumption that allows the use of Markov chain numerical techniques and standard Reachability Graph algorithms for the analysis of SPN models.

- Deterministic time assumption that allows the use of Embedded Markov chain techniques and special purpose Reachability graph algorithms for the analysis of TPN models.

- Unrestricted probability distributions (neither Phase-type nor deterministic) that allow only the application of simulation and distribution insensitive results as general analysis algorithms.

The key for success of a given modelling formalism is the identification of a good trade-off between structural and interpretation complexity that corresponds to efficient analysis algorithms. In order to be practically useful, a modelling formalism with a rich interpretation requires sophisticate analysis algorithms that support it efficiently. A rich interpretation not supported by efficient algorithms is counter-productive: the modeller "measures" the complexity in terms of graphic model complexity (high solution cost for graphically simple models is frustrating). Examples of formalisms with interpretation complexity too high with respect to the efficiency of the analysis algorithms will be shown in Section 4.2.4. If several system features are modelled by sophisticate interpretations, the modeller will produce structurally simpler and/or smaller models so that the analysis algorithms will be applied to simpler and/or smaller cases.

4.2.2 QN models

According to the classification proposed in the previous section, QNs as graphic performance models can be characterized as follows.

QN models are intended to represent the *input/output* relation through the concept of customers visiting different service stations. The internal implementation of the individual service stations is purposely hidden. Only the effect in terms of service times is taken into account. Internal details of implementation that have an impact on the characterization of customer flows (e.g. FCFS service order or number of servers) are added to the model interpretation.

QN models are often used in a hierarchical way in order to model large systems. Notice however that even in case of hierarchical modelling, the refinement of a service center is always in terms of subnetworks that take only queueing and delay effects into account. A service center can never be replaced by a description of how the service is implemented. In this sense the formalism is closed in the performance evaluation environment, and can hardly be opened to other interesting environments such as system design, rapid prototyping, etc.

Analysis algorithms (exact for BCMP-type or approximate in case of non product-form nets) are usually based on the knowledge of a "characteristic function" for each service center, and their combination to produce net level performance results.

4.2.3 Markov chain models

Markov chains are the classic formalism used to represent memoryless systems at an arbitrary level of details. They can be considered a graphic model if we think at the graphic representation of the state transition diagram of a chain, much in the same way as in Computer Science finite state automata are used to characterize formal languages. A state variable is identified, and the behaviour is described by specifying state transitions from each possible state of the model.

Changing the definition of the state variable it is possible to decide whether to include or not details concerning the implementation of services. Contrarily to QN models, in MC the graphic structure is related to the state variable representing the whole system rather than to the input/output relation among system components. In this sense the graphic structure represents how the system behaves in terms of transitions from state to state rather than relationships and performance dependencies among system components. Performance measures may be defined for example in Markov-Reward models as functions related to the permanence in states or to the movement from a state to the other.

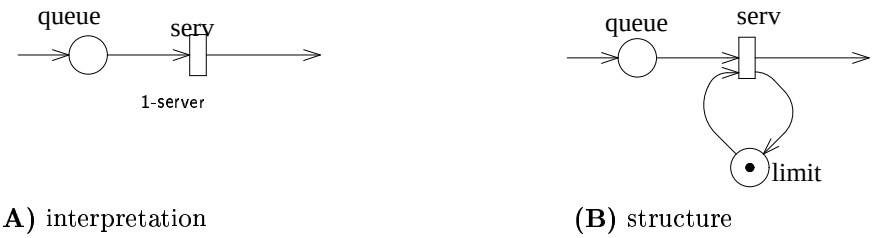
MCs are open to design and prototyping since a description of a system component in terms of finite state machine specification can be directly transformed into a MC model, and vice-versa. The structure of modules composing a complex system is however lost in the MC graphic representation since the global state variable of the complete system is considered.

The analysis of the graph structure provides information on the behaviour of the system. For example a strongly connected state transition graph corresponds to a system with repetitive behaviour (ergodicity).

4.2.4 Petri net models

Petri nets combine the capabilities of behavioural modelling at the level of state transition specification of finite state machines and of input/output relation of network models. The concept of *local state variable* allows the expansion of an abstract model containing a representation of service centers by substituting a more detailed description of how a server is implemented. This feature is not provided by QN or MC models in the same natural way. This unique characteristics of PN models can be exploited to arbitrarily decide what features of a given system must be represented explicitly by the model structures and what can be specified by the associated interpretation.

As a trivial example of this degree of freedom, consider the problem of modelling a single server queue. Figure 4.1 depicts two alternative ways of representing this system. In Figure 4.1.A the "single server semantics" is attached to transition "serv" as part of the net interpretation. In Figure 4.1.B the same



(A) interpretation **(B)** structure

Figure 4.1: Two ways of representing a single server queue.

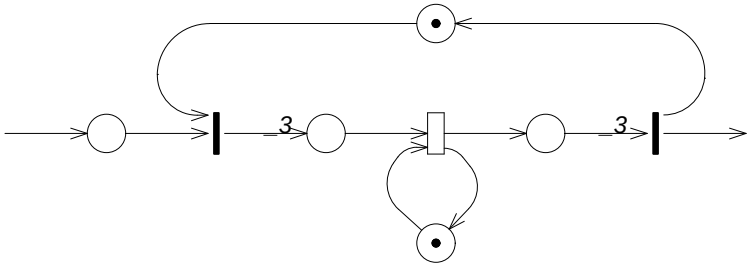


Figure 4.2: Structural GSPN representation of Erlang 3 distribution.

feature is modelled explicitly by the addition of place "limit," that reduces the enabling degree of transition "serv" to 0 or 1 (depending on the fact that place "queue" is empty or not). In the case depicted in Figure 4.1.B the server semantics for transition "serv" is irrelevant, and could be ∞-server for instance. The server is represented explicitly by the token in place "limit," and its activity is modelled in term of synchronization between the server and one customer's request.

Another less trivial example of these possibilities of trade-off between graphic complexity and interpretation of a TPN model consider the case of inclusion of Phase-type distributions in GSPN models. According to [9], a GSPN structural representation of a monoclass, non-preemptive, single server center with Erlang-3 probability distribution and random order queueing discipline is depicted in Figure 4.2. This model assumes an exponential distribution with infinite server semantics interpretation for all timed transitions (white, rectangular boxes). The same system can be modelled as shown in Figure 4.3 using the ESP formalism [10] in which Phase-type distributions are allowed directly as interpretation for timed transitions. Notice that, despite their higher modelling convenience, ESP nets were not successful because of the inefficiency of analysis algorithms, that required in any case a (automatic) "macro expansion" of the underlying Markov chain.

Finally, let us consider as another example the representation of queueing

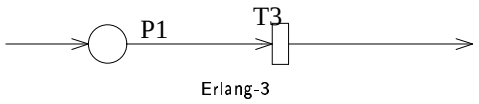


Figure 4.3: Structural GSPN representation of Erlang 3 distribution.

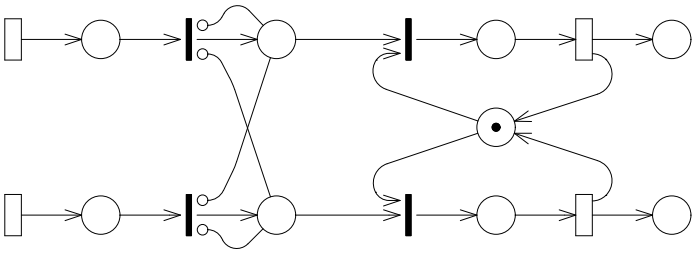


Figure 4.4: Structural GSPN representation of 2-bounded, 2-class FIFO queue.

disciplines. In the framework of non coloured Petri net models where tokens in a place are indistinguishable it makes sense to specify a queueing and a service discipline only for multiclass models where customers of different classes are represented by markings in different places. Figure 4.4 depicts a GSPN structural representation of a two class, non-preemptive, single server station with FCFS queueing discipline in case of 2-bounded queues. The explicit representation of the queueing discipline is obtained by forcing the places to contain at most one token at a time, and then choosing the marking of the place that corresponds to the desired position in the queue. In case of places that may contain more than one token the usual interpretation of the untimed PN model is that one token is considered, no matter which one. This absence of queueing discipline may of course be changed in different ways. One possible interpretation (the usual one) for the definition of TPNs is that one token is chosen at random with equal probability, thus implementing a "random order" queueing discipline. However, even a FCFS queueing discipline is compatible with the semantics of untimed PNs, so that nothing prevents us from defining a different queueing discipline interpretation with each place of a TPN model, in a similar way as we do for BCMP QNs. Figure 4.5 depicts a possible example of use of such an extended TPN formalism. Besides the (obvious) advantage in terms of graphic conciseness, the problem is of course the definition of appropriate analysis algorithms for such extended models. As in the previous case, the "obvious" algorithm consisting in an automatic translation from the TPN representation in Figure 4.5 to the one in Figure 4.4 (although feasible) would not be satisfactory due to the lack of efficiency. Special purpose algorithms with computational complex-

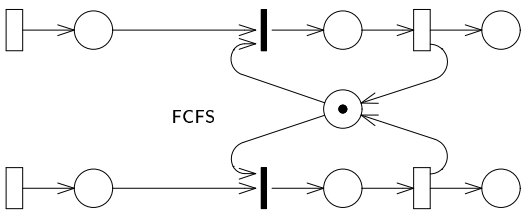


Figure 4.5: Example of TPN with FIFO queue interpretation.

ity related to the graphic complexity of the net in Figure 4.5 would be required in order for the use of the compact representation to be effective. As a very special and restricted example of direct exploitation of the FCFS interpretation of conflict resolution in a TPN, we propose the computation of the upper bound for the waiting time: in case of FCFS it is finite and given by the sum of the waiting times of the other customers arrived before plus the service time of the customer itself. This value is peculiar of the FCFS queueing discipline, and does not apply for instance to the usual interpretation of random order for TPNs. In this sense, the specification of FCFS might *reduce* the complexity of the computation of some performance indices instead of increasing it as it would be the case if the automatic translation to explicit structural representation was applied.

4.2.5 Coloured TPNs

From a theoretical point of view, coloured Petri nets [11] may be considered a special case of interpretation of Petri net structures. Tokens are no longer indistinguishable: they carry information that may be structured like "records of fields," each one composed by an "enumerated type" of an high-level programming language. Places and transitions are also typed accordingly. Values of the record fields are manipulated by transition firings. The modelling power increases substantially if no restriction is posed on the kind of manipulation of data associated with tokens.

Well-formed Coloured nets [12] are a special case in which the purpose of the colour introduction is to describe and analyze symmetric systems. Timed versions of Well-formed nets have been proposed as well with the same goal of symmetry exploitation [13, 5, 14, 15]. The manipulation of symmetric models is supported by special purpose analysis algorithms [16, 17].

A broad parallel may be established between coloured PNs in general (and WNs in particular) and multiclass QNs. They are somewhat similar in terms of the basic idea of labelling customers or tokens so that one can distinguish them even if they are sitting in the same queue or marking the same place. The effect of moving from basic PNs to WNs is however different from that of moving from single class to multiclass BCMP QNs, due to the differences between the two graphic formalisms and their interpretations. A summary of the comparison

between TWNs and multiclass QNs is reported in the following table:

- Multiclass BCMP QNs cannot be represented by (larger) monoclass BCMP QNs.
- Multiclass QNs can be represented by non-coloured TPNs.
- The use of Well-formed TPNs for the representation of monoclass QNs yields models with structural complexity similar to that of non-coloured TPNs.
- The use of Well-formed TPNs for the representation of multiclass QNs yields structurally simpler models than non-coloured TPNs.
- The availability of efficient algorithms for the symmetry interpretation allows one to relate the graphic simplification of the model description with a lower analysis cost.

4.3 Conclusions

Petri nets allow both "net level" and "state level" modelling using the same formalism. This provides freedom in the definition of what must be expressed by the net structure and what can be expressed by the interpretation. This freedom may be exploited to tune the modelling power of the formalism to the needs of different application domains. This tuning is always guided by the trade-off between modelling and analysis power. In particular one must be very careful in avoiding the introduction of formalisms with very high modelling power in terms of graphic conciseness if this conciseness has no counterpart in the computational complexity of the associated analysis algorithms. Complex interpretations should always be supported by efficient dedicated analysis algorithms, so that the analysis of graphically simple models does not require "macro expansions" in terms of more primitive constructs.

From the modelling point of view one would like to express all basic mechanisms that characterize a given application domain by the interpretation, so that the model structure is kept simple. Following this principle the model structure is focused on the specification of the peculiar features of a model rather than on the features that are common to most models in the considered application domain. In this sense different application domains may probably require different kinds of interpretations associated with the Petri net structure.

Colours increase the modelling power of the structural part of a TPN identifying and exploiting symmetries. This can be seen as a special form of interpretation. The exploitation of algorithms that take symmetry into account for the efficient analysis of TPN models is an example of introduction of useful interpretation supported by special purpose algorithms.

Bibliography

[1] M. Vernon, J. Zahorjan, and E. D. Lazowska. A comparison of performance Petri nets and queueing network models. In *Proc. 3^{rd} Intern. Workshop on Modelling Techniques and Performance Evaluation*, pages 181–192, Paris, France, March 1987. AFCET.

[2] G. Balbo, S. C. Bruell, and S. Ghanta. Combining queueing network and generalized stochastic Petri net models for the analysis of some software blocking phenomena. *IEEE Transactions on Software Engineering*, 12(4):561–576, April 1986.

[3] G. Balbo, S. C. Bruell, and S. Ghanta. Combining queueing network and generalized stochastic Petri nets for the solution of complex models of system behavior. *IEEE Transactions on Computers*, 37(10):1251–1268, October 1988.

[4] M. Ajmone Marsan, G. Balbo, G. Chiola, G. Conte, S. Donatelli, and G. Franceschinis. An introduction to Generalized Stochastic Petri Nets. *Microelectronics and Reliability*, 31(4):699–725, 1991. Special issue on Petri nets and related graph models.

[5] G. Balbo, G. Chiola, S.C. Bruell, and P. Chen. An example of modelling and evaluation of a concurrent program using coloured stochastic Petri nets: Lamport's fast mutual exclusion algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):221–240, March 1992.

[6] J. Campos, G. Chiola, and M. Silva. Ergodicity and throughput bounds for Petri nets with unique consistent firing count vector. *IEEE Transactions on Software Engineering*, 17(2):117–125, February 1991.

[7] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):1368–1382, December 1991.

[8] J. Campos, G. Chiola, J. M. Colom, and M. Silva. Properties and performance bounds for timed marked graphs. *IEEE Transactions on Circuits and Systems—1: Fundamental Theory and Applications*, 39(5):386–401, May 1992.

[9] P. Chen, S.C. Bruell, and G. Balbo. Alternative methods for incorporating non-exponential distributions into stochastic Petri nets. In *Proc. 3rd Intern. Workshop on Petri Nets and Performance Models*, pages 187–197, Kyoto, Japan, December 1989. IEEE-CS Press.

[10] A. Cumani. Esp - a package for the evaluation of stochastic Petri nets with phase-type distributed transition times. In *Proc. Int. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.

[11] K. Jensen and G. Rozenberg, editors. *High-Level Petri Nets. Theory and Application.* Springer Verlag, 1991.

[12] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. In *Proc.* 11^{th} *International Conference on Application and Theory of Petri Nets*, Paris, France, June 1990. Reprinted in *High-Level Petri Nets. Theory and Application*, K. Jensen and G. Rozenberg (editors), Springer Verlag, 1991.

[13] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets and multiprocessor modelling applications. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets. Theory and Application.* Springer Verlag, 1991.

[14] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 1993. accepted for publication.

[15] G. Chiola, R. Gaeta, and M. Sereno. A simulation model of a double ring protocol based on timed well-formed coloured Petri nets. In H. Schwetman, J. Walrand, K. Bagchi, and D. DeGroot, editors, *Intern. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, volume 25 of *Simulation Series*, pages 259–264, San Diego, California, January 1993. Society for Computer Simulation.

[16] G. Chiola, G. Franceschinis, and R. Gaeta. A symbolic simulation mechanism for well-formed coloured Petri nets. In *Proc. 25th SCS Annual Simulation Symposium*, Orlando, Florida, April 1992.

[17] G. Chiola, R. Gaeta, and M. Ribaudo. Designing an efficient tool for Stochastic Well-Formed Coloured Petri Nets. In R. Pooley and J. Hillston, editors, *Proc.* 6^{th} *Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 391–395, Edinburg, UK, September 1992. Antony Rowe Ltd.

Chapter 5

Stochastic Process Algebras

5.1 Introduction

In recent decades parallel and distributed systems have been extensively used in many application areas, and there has been significant effort in developing formal methodologies for the specification and analysis of such systems; unfortunately not always completely satisfactorily. The presence of concurrency, communication, synchronisation and nondeterminism makes the study of the correctness of concurrent systems particularly difficult, especially when the process interaction structure is not regular.

We have already presented in the previous chapters two approaches to modelling complex systems. Here we introduce a new one, known as *stochastic process algebras*, which provides a formal apparatus to reason about the qualitative and quantitative behaviour of these systems.

Following the style of previous chapters the presentation will be rather informal: we first start from the basics of the formalism and then describe some of the extensions which have been recently proposed in the literature to make it suitable for performance modelling. Stochastic process algebras, as was the case for the previously presented formalisms, are suitable for both the modelling and the analysis of discrete event dynamic systems, but here we are mainly concerned with the modelling aspects, referring to chapters which will appear later in this book for discussion of the analysis techniques. In the second part of the chapter we briefly discuss the relationship between stochastic process algebras and stochastic Petri nets.

5.2 Process Algebras

Process algebras (PA) are abstract languages which have been introduced for the specification and understanding of complex systems characterised by *communication* and *concurrency* [35]. These systems are seen as collections of entities that are acting independently most of the time, and which sometimes have to

interact through communication in order to achieve a common goal. Process algebras, as we will discuss, provide a formal framework in which such systems can be defined, interpreted, and analysed.

5.2.1 Classical process algebras

Throughout this chapter we will refer to process algebras which have not been extended with notions of probability or time, as *classical process algebras*. Here we briefly overview the fundamentals of process algebra in terms of two classical process algebras which have been strongly influential in the field: CCS and CSP.

In a process algebra concurrent systems are described as collections of entities, called *agents*, that execute atomic *actions* which constitute the building blocks of the language. Complex systems are built starting from these building blocks and by applying the constructors of the algebra. In addition to the compositional operators, mechanisms for abstraction are provided which allow internal details to be disregarded. Legal terms in the algebra are defined by a grammar. The actions can be visible or not. We assume there is a set of visible actions, denoted Act. By convention, actions are represented by lower case letters.

The Calculus of Communicating Systems (CCS) [35] is considered the starting point in the field of PA. In CCS, Act consists of two sets, $\Lambda = \{a, b, c, \ldots\}$, the set of names, and $\overline{\Lambda} = \{\overline{a}, \overline{b}, \overline{c}, \ldots\}$, the set of co-names. An additional action, called the *silent* or *perfect* action τ, is introduced to represent internal (not visible) activity. The syntax of CCS is defined by the following grammar:

$$P ::= Nil \mid a.P \mid P + P \mid P|P \mid P \setminus L \mid P[f] \mid rec(X = P)$$

where $a \in \mathcal{A}ct \cup \tau$, $L \subseteq \mathcal{A}ct$, and X is a process variable.

The names and the intuitive meanings of the operators are the following:

- **Inactivity** Nil represents the agent that cannot perform any action.
- **Prefix** This operator constitutes the basic mechanism by which the behaviours of components are constructed: $a.P$ is the agent that can perform an action a and subsequently behaves like P.
- **Choice** $P + Q$ is a composite process that can behave either like P or like Q. When one component is selected the other is discarded, the choice being performed nondeterministically.
- **Parallel Composition** $P|Q$ is a process whose components P and Q might proceed concurrently and independently or they might synchronise. Two parallel agents can engage in a communication when one agent performs an action, say a, and the other agent performs complementary action $\overline{a}$. The result of the communication between the two partners is a τ action which is no longer visible in the environment.
- **Restriction** $P \setminus L$ represents the process that cannot perform any actions $\{a, \overline{a}\} \in L$.

- **Relabelling** $P[f]$ is an agent that behaves like P, but with the actions relabelled by the function f.

- **Recursion** The recursive term $rec(X = P)$ may describe an infinite behaviour: for example, the expression $rec(X = a.X)$ may be read as *"the agent X such that $X = a.X$"* and represents an agent that can perform an infinite number of actions a. In this so-called *rec* expression, X is a bound variable and can be renamed without any semantic effect.

An alternative way for describing infinite behaviours makes use of *constants*, i.e. names that can be assigned to patterns of behaviour associated with components. It is possible to specify equations like $A \stackrel{def}{=} P$, which gives the constant A the behaviour of the component P; if the definition of P contains A then an infinite behaviour is obtained. In the following we will use constant names instead of the *rec* operator.

Another well known process algebra is the Communicating Sequential Processes (CSP) originally introduced in [28] and in a more abstract version in [40]. In contrast to CCS, this language does not include the notion of complementary actions and a new form of communication between components is proposed. Together these allow the specification of multiway synchronisations rather than the strict pairings which are allowed in CCS.

The expression $P \|_S Q$ represents the parallel composition of P and Q with respect to the set S of joint actions (τ cannot belong to S). $P \|_S Q$ behaves like P or Q running independently of each other except for all actions contained in the set S on which they *must* synchronise and communicate. By varying the synchronisation set S, parallel composition $\|_S$ ranges from arbitrary interleaving, when S is the empty set (in this case the concise notation $P \| Q$ is usually used), to full synchrony, when S comprises all the possible actions. During the communication the joint action remains visible to the environment and it can be reused by other concurrent processes. This rule leads to a new kind of communication (multiway synchronisation) in which more than two processes can be involved.

In CSP it is also possible to abstract from internal details using a new operator called *hiding*. The process P/b represents a process that behaves like P except for the action b that is hidden from the external environment. The hiding operator differs from the CCS restriction operator. Restriction actually stops the actions with a given label from occurring, whereas hiding allows the actions to proceed invisibly.

CSP provides several other operators which will not be explained in this introductory description. The operators which have been presented here form the core set of operators found in most process algebras. In the following section we will introduce the process algebra which we will use for the remainder of this chapter, which contains elements from both CCS and CSP.

5.2.2 A process algebra: syntax and formal semantics

In this subsection we define the process algebra on which we will base the remainder of our presentation. As well as the syntax and semantics of this language we discuss some important features of process algebra, namely compositionality and abstraction mechanisms, as exemplified in our language.

Syntax As already explained the language we will use is an amalgamation of features from CCS and CSP. In particular we use some CCS operators plus the CSP synchronisation and hiding mechanisms. The motivation for choosing such a language is mainly that some PA extensions that we will discuss later in the chapter use this set of operators. Additionally, we feel that this language contains a sufficient set of operators to be expressive whilst not overawing the reader with details.

The grammar of our PA language is:

$$P ::= Nil \mid a.P \mid P + P \mid P \|_S P \mid P/L \mid A$$

where $a \in \mathcal{A}ct \cup \tau, S \subseteq \mathcal{A}ct, L \subseteq \mathcal{A}ct$. The intuitive meanings of the operators coincide with those given in the previous section (Section 5.2.1).

Once we have defined our set of language operators we need to associate a precise meaning with each of them, i.e. we need to define their semantics.

Interleaving semantics The formal semantic rules for the operators, which are usually presented in the Structural Operational Semantics (SOS) style of Plotkin [44], allow one to associate with each agent a Labelled Transition System (LTS). In general a LTS $= (S, T, \rightarrow)$ is defined by a set of states S, a set of transition labels T and a transition relation $\rightarrow \subseteq S \times T \times S$. In the case of PA the set of states is given by the set of language terms, the set of transition labels is given by the set of (atomic) actions, and the transition relation is given by the operational rules.

An example of SOS may be found in Figure 5.1 where the rules for the prefix and parallel composition operators are shown. These rules are read as follows: if the transition above the line is possible, then we can infer the transition below the line.

No precondition is necessary for prefix and the term $a.P$ can evolve into P by executing action a. Three different rules are needed to specify precisely the meaning of parallel composition. The first two rules define the behaviour of P and Q when executing independent actions. If P, by executing a, evolves into P' and $a \notin S$, then the composite process $P \|_S Q$, by executing a, can evolve into $P' \|_S Q$ (similarly for Q). The third rule defines the communication between the two components when executing an action in the synchronisation set S. If P executes a and Q executes a, an interaction can take place. The resulting activity is a joint action a and the whole process evolves into $P' \|_S Q'$.

By applying the semantic rules it is possible to associate with each language expression a *transition tree* that can be viewed as a way of describing the be-

Prefix

$$\frac{}{a.P \xrightarrow{a} P}$$

Parallel Composition

$$\frac{P \xrightarrow{a} P'}{P \parallel_S Q \xrightarrow{a} P' \parallel_S Q} \; a \notin S \qquad \frac{Q \xrightarrow{a} Q'}{P \parallel_S Q \xrightarrow{a} P \parallel_S Q'} \; a \notin S$$

$$\frac{P \xrightarrow{a} P' \;\; Q \xrightarrow{a} Q'}{P \parallel_S Q \xrightarrow{a} P' \parallel_S Q'} \; a \in S$$

Figure 5.1: Prefix and parallel composition semantic rules.

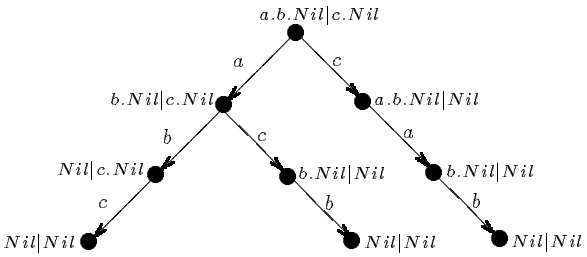


Figure 5.2: Transition tree of P.

haviour of the modelled system. The nodes of the tree are labelled with language expressions and the arcs with the actions that cause their evolution.

An example of a transition tree for the term $P = a.b.Nil \parallel c.Nil$ is shown in Figure 5.2. This tree corresponds to the interleaving semantics of P and gives all the possible sequences of actions that could be observed during the evolution of the model. Notice that in the interleaving semantics we abstract from the fact that the system is composed of separate components (only two in this simple example, $a.b.Nil$ and $c.Nil$) because we consider a *global* state without considering its distributed nature. Actions of independent components are merged with actions of the others in such a way that all possible interleavings are represented.

Some nodes in the tree of Figure 5.2 are labelled with the same syntactic expression. When we consider an equivalence notion between nodes for which the nodes characterised by the same label are considered to be equivalent, we can collapse them into a single one and obtain the *transition diagram* (also called *derivation graph*) of Figure 5.3. This notion of syntactic equivalence is

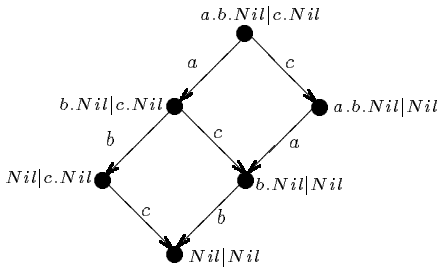


Figure 5.3: Transition diagram of P.

quite simple, probably the simplest notion of equivalence that could be defined. In the theory of PA significant effort has been given to the study of equivalence relations between states and between processes, and we will present some of them later in this chapter. We end this discussion by inviting the reader to observe the close relationship between the reachability graph underlying a net model and the derivation graph underlying a process algebra model.

Concurrent semantics A major drawback of interleaving models is that they abstract from the independence of the actions executed in independent subsystems. Noninterleaving models capture the fact that a system consists of a set of (partially) independent components and do not refer to the notion of a global state. The system's behaviour is modelled in terms of sequences of actions which are not required to be totally ordered, but which are only partially ordered. This partial order reflects the causal dependences between actions.

There is a strong debate around the choice of interleaving versus noninterleaving models for the definition of the semantics of PA and this chapter is not the right place to discuss this problem. We simply recall that some examples of noninterleaving models are Petri nets, event structures [52], and partial order traces [33] and that there is a branch of the research in this field related to the definition of a concurrent semantics for PA by means of noninterleaving models [9, 18, 41, 48, 31, 43].

Compositionality Process algebra models have been used extensively to establish the correct behaviour of complex systems [10, 6, 1]. These models are built following a *compositional* approach which constitutes a central feature of model construction.

Each subsystem is modelled in isolation and then the submodels are composed using the operators provided by the calculus in order to obtain the model of the whole system. Model components can be developed by different modellers and libraries of re-usable components may be established.

This leads to a hierarchical approach to model construction: the resulting model has a structure which reflects the structure of the system itself, is easy to understand, and readily modifiable. Moreover, this structure may be exploited

during analysis.

The profound benefits of compositionality become apparent when we consider equivalence relations over models. If the relation can be shown to be a *congruence*—meaning that it respects the operators of the language in such a way that equivalence of components can be considered in isolation—analysis of the model via the relation can be carried out component by component. In general, this greatly reduces the complexity of the models which need to be tackled, during model verification for example.

Abstraction mechanism The abstraction mechanism allows some behaviour of the system to be abstracted away, for example because it is more detailed than necessary for the current model. In our PA this is provided by the hiding operator which allows us to abstract some aspect of the behaviour of a component so that it is not visible to an external observer, or to other components. The component P/L in fact behaves as P, except that the activities of types within the set L are *hidden*. They appear as the silent type τ and they are considered to be internal to the component in which they occur. Thus hiding allows an interface to a model or component to be defined.

This is particularly powerful when used in conjunction with the parallel composition combinator as it may restrict the interactions of a component. Components of a system may be modelled individually in detail, but subsequently in a more abstract form as the interactions between them are developed. For example, let us consider the term $P = a.b.Nil$ which offers the two actions a and b to the environment. By hiding one of the two actions, say a, we have $P' = P/a$ and we change the system interface since now P' offers only b. This means that if we want to compose P' with other components, or with new replicas of P, we can only require synchronisation on b, the action a now being internal.

5.2.3 The PLC multicomputer example

To better understand how PA may be used to describe system behaviours we now consider the simple multicomputer PLC example discussed earlier in the book (see Example 2.1 in Chapter 2). We recall that the example consists of a multicomputer architecture responsible for the control of a distributed plant. Each computer has a double access memory which is connected to a common bus. The behaviour of the PLC is cyclic: first the computers synchronise to start a control cycle, then they perform their calculation independently as long as they need to read external data.

The actions executed by the PLC components form the set:

$$Act_{PLC} = \{start\_cycle, local\_comp, end\_cycle, acq\_bus, read\_ext\_data, req\_ext\_data\}$$

The entities involved in the system are first modelled in isolation. Each computer is a *sequential* component whose behaviour may be expressed by means

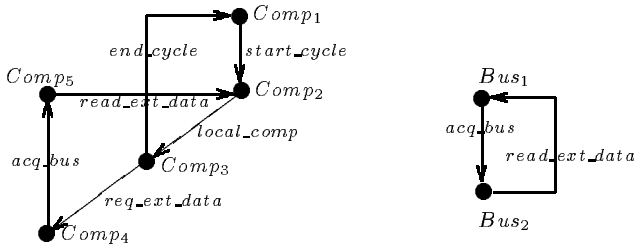


Figure 5.4: Transition diagrams of the computer (left) and the bus (right) components.

of the prefix and choice operators as follows:

$$\begin{aligned} Comp_1 &= start\_cycle.Comp_2 \\ Comp_2 &= local\_comp.Comp_3 \\ Comp_3 &= end\_cycle.Comp_1 + req\_ext\_data.Comp_4 \\ Comp_4 &= acq\_bus.Comp_5 \\ Comp_5 &= read\_ext\_data.Comp_2 \end{aligned}$$

The bus is acquired and used for reading external data; a possible specification could be the following:

$$\begin{aligned} Bus_1 &= acq\_bus.Bus_2 \\ Bus_2 &= read\_ext\_data.Bus_1 \end{aligned}$$

By applying the semantic rules we can derive the transition diagrams of $Comp_1$ and Bus_1 which are shown in the left and right parts of Figure 5.4 respectively.

After having specified the behaviour of the single entities, we need to express the behaviour of the complete system. This is done by composing the entities and by establishing their interactions.

As in the Example 2.1 discussed in Chapter 2, let us suppose the system is formed by two computers which synchronise to start a control cycle. This system is modelled by two instances of the entity $Comp_1$ which synchronise on the action $start\_cycle$, representing the beginning of the control cycle. Moreover, both computers need the bus in order to read external data. This behaviour is achieved by adding one instance of the Bus_1 component with an appropriate synchronisation set. The specification of the PLC example is thus the following:

$$PLC = (Comp_1 \|_{\{start\_cycle\}} Comp_1) \|_{\{acq\_bus, read\_ext\_data\}} Bus_1$$

Once we have specified the complete model, we can derive the underlying transition diagram. Investigation of this diagram may prove several qualitative properties of the system as we will see in Chapter 6 when discussing the analysis methods based on the construction of the reachability (derivative) graph. Figure 5.5 shows a portion of the transition diagram underlying the PLC specification where, for simplicity, we have used short labels for the states (the name

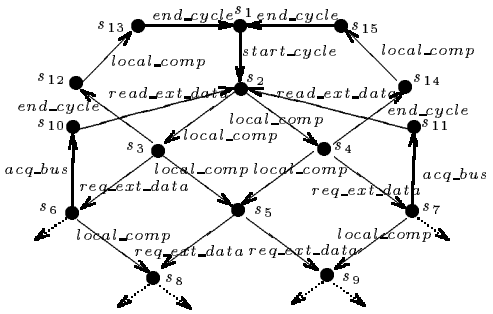


Figure 5.5: Portion of the transition diagram underlying the PLC example.

s_1	$(Comp_1 \|\|_S Comp_1) \|\|_L Bus_1$
s_2	$(Comp_2 \|\|_S Comp_2) \|\|_L Bus_1$
s_3	$(Comp_3 \|\|_S Comp_2) \|\|_L Bus_1$
s_4	$(Comp_2 \|\|_S Comp_3) \|\|_L Bus_1$
s_5	$(Comp_3 \|\|_S Comp_3) \|\|_L Bus_1$
s_6	$(Comp_4 \|\|_S Comp_2) \|\|_L Bus_1$
s_7	$(Comp_2 \|\|_S Comp_4) \|\|_L Bus_1$
s_8	$(Comp_4 \|\|_S Comp_3) \|\|_L Bus_1$
s_9	$(Comp_3 \|\|_S Comp_4) \|\|_L Bus_1$
s_{10}	$(Comp_2 \|\|_S Comp_5) \|\|_L Bus_2$
s_{11}	$(Comp_5 \|\|_S Comp_2) \|\|_L Bus_2$
s_{12}	$(Comp_1 \|\|_S Comp_2) \|\|_L Bus_1$
s_{13}	$(Comp_1 \|\|_S Comp_3) \|\|_L Bus_1$
s_{14}	$(Comp_2 \|\|_S Comp_1) \|\|_L Bus_1$
s_{15}	$(Comp_3 \|\|_S Comp_1) \|\|_L Bus_1$

Table 5.1: Labels of the states in the transition diagram of Figure 1.4 (where $S = \{start\_cycle\}, L = \{acq\_bus, read\_ext\_data\}$).

of each state is listed in Table 5.1). The complete transition diagram has 24 states and 47 transitions.

Let us now suppose that we want to add new computers to our system. This extension may be achieved compositionally by combining more instances of the $Comp_1$ component. For example, in the case of four computers and one common bus we have:

$$PLC' = (Comp_1 \|_{\{start\_cycle\}} Comp_1 \|_{\{start\_cycle\}} Comp_1 \|_{\{start\_cycle\}} Comp_1) \|_{\{acq\_bus, read\_ext\_data\}} Bus_1$$

Finally, let us suppose that we do not want to observe the acquisition of the global bus, but only the data exchange. We can model this situation by taking advantage of the abstraction mechanism and by hiding the action *acq_bus*:

$$PLC'' = ((Comp_1 \|_{\{start\_cycle\}} Comp_1 \|_{\{start\_cycle\}} Comp_1 \|_{\{start\_cycle\}} Comp_1) \|_{\{acq\_bus, read\_ext\_data\}} Bus_1)/\{acq\_bus\}$$

5.2.4 Modelling features

Causal dependences, Concurrency, and Conflicts Bearing in mind the informal meanings of the algebraic operators, let us briefly discuss now the adequacy of the formalism for modelling causal dependences, concurrency, and conflicts between actions, modelling features that we have already discussed in Chapter 2 in the context of PN models.

Causal dependences appear in the form of sequences of actions which can be easily modelled by means of the prefix operator. In the PN context we said that a transition t_j follows a transition t_i when they are connected through a place. In the PA context we can say that an action b follows an action a when they are in the form $a.b$. Paraphrasing the net terminology, we could say that *"a and b are connected through the prefix operator."*

Concurrency between actions is syntactically represented by means of the parallel composition operator. For instance, $a.Nil \parallel b.Nil$ denotes an expression in which the two actions a and b are simultaneously enabled and independent from each other. However, as we have already discussed, when we consider the interleaving semantics of this term, we have that a and b can occur in any order, first a and then b, or vice versa, but never simultaneously.

Moreover, in the interleaving approach causality information may be lost. Let us consider, for example, the two agents $P = a.Nil \parallel b.Nil$ and $Q = a.b.Nil + b.a.Nil$. Under the interleaving assumption the external behaviour of P and Q is the same: an external observer will either see first an a and then a b, or first a b and then an a, and therefore will not be able to distinguish between them. The observer is not able to detect that in $a.Nil \parallel b.Nil$ the actions a and b occur independently, while in $a.b.Nil + b.a.Nil$ there is a causal dependence between them: a occurs before b or b occurs before a. In contrast, in the noninterleaving approach to semantics, these two agents are no longer indistinguishable because a distinction occurs between P, in which a and b can occur simultaneously, and Q in which this option is not present.

Conflicts in sequential systems represent situations in which two or more actions are enabled but only one can occur. The basic mechanism to model conflicts in PA makes use of the alternative or choice operator: the expression $P = a.Nil + b.Nil$ models a situation in which both actions a and b are enabled but only one of them can occur, the choice being performed nondeterministically. This simple form of conflict can be assimilated to the free-choice conflict already discussed. When we move to concurrent systems things become more complicated. Consider again the term P and suppose it is composed in parallel with another term Q, requiring that they have to synchronise on one of the two actions, for example a (i.e. $P \|_{\{a\}} Q$). Depending on the specification of Q we can have a conflict or not: indeed, if Q does not offer any action a, the choice will be always resolved in favour of b. The reader is invited to observe the close relation between this situation and the non free-choice conflicts of PN.

Structure and state As already discussed in Chapter 2, a PN model of a dynamic system consists of a net structure and a marking and it is possible to reason at two different levels, structural and behavioural.

There is no syntactic distinction between structure and state in PA. However, at the semantic level, in the LTS, a state is associated with each syntactic term. Thus for a PA model each derivative of the initial expression representing the model is considered to be a state. Observe that, since the model and each derivative are specified in the algebraic language, it is impossible to distinguish syntactically whether a term is a derivative of a model or a model itself.

Model analysis Due to the lack of an explicit structure it is not generally possible to prove properties which hold for any initial state, as in the case of PN models. Qualitative properties, like the absence of *deadlock* or the *reachability of a given state*, are then investigated by inspecting the transition diagram associated with the model as will be discussed in detail in Chapter 6. Recent work by Gilmore *et al.* [16] has established a structural theory for PA and this is currently under development.

5.2.5 Equivalence notions and equational laws

One of the most attractive features of PA is their compositional nature, but it is not the only one. Another important aspect of the formalism is the definition of equivalence relations [35], which can be used to compare agents (*model verification*) and to replace one agent by another which exhibits an equivalent behaviour, but has a simpler representation (*model simplification*). Such notions of equivalence are considered part of the semantics of the language, and therefore their definition is an integral part of its development. Another use of equivalence relations is over the states within a model. When a set of states are found to have equivalent behaviour we can simplify analysis by using the relation to partition the state space and considering only one representative of each partition (*model aggregation*). This is an important means of state space reduction.

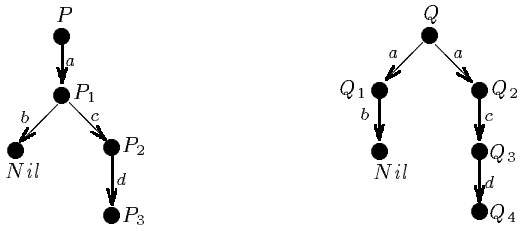


Figure 5.6: Transition diagrams of the two agents P (left) and Q (right).

We have already briefly discussed the notion of syntactic equivalence between the states forming the transition tree of a model. Unfortunately this relation has no benefits for model aggregation since the number of states remains unchanged. However, other equivalence notions between states have been proved to be very powerful for model aggregation and will be discussed in Chapter 16 while here we limit the discussion to the equivalence notions between agents.

The semantic theory of PA is an observational theory: the behaviour of a system corresponds to what is observable, and to observe a system corresponds to communicating with it. This approach is based on a single observer (interleaving) model in which the occurrence of events is serialised even for independent processes. Let us consider the agents:

$$\begin{array}{lcl} P & = & a.P_1 \\ P_1 & = & b.Nil + c.P_2 \\ P_2 & = & d.P_3 \end{array} \qquad \begin{array}{lcl} Q & = & a.Q_1 + a.Q_2 \\ Q_1 & = & b.Nil \\ Q_2 & = & c.Q_3 \\ Q_3 & = & d.Q_4 \end{array}$$

and imagine an observer R trying to interact with them, i.e. $R\|_S P$ and $R\|_S Q$, with $S = \{a, b, c, d\}$. At the beginning both agents P and Q offer a to R (see their transition diagrams in Figure 5.6). But after the action a is performed, a difference emerges between them. P is deterministic: after the execution of action a, it always evolves into P_1 and it subsequently offers to the observer both actions b and c. The agent Q is instead nondeterministic due to the presence of the choice operator in its description. After the execution of the action a, Q can evolve into Q_1 *or* into Q_2 and it offers to the observer sometimes an action b and sometimes an action c. Thus, even though P and Q offer the same set of actions to R, they do not exhibit an equivalent behaviour.

We need to define some criteria to decide when two agents can be considered equivalent. As the meaning of agents is expressed by their corresponding transition diagrams we could think of two agents being equivalent when these graphs are isomorphic. However, isomorphism has been proved to be too strong a requirement and new notions of equivalence have been proposed [35]. In CCS for example, two agents are considered to be equivalent when their externally observed behaviours appear to be the same or, in other words, when any action by one can be matched by an action of the other and afterwards they remain equivalent. This notion of equivalence is known as *strong bisimulation*; it was

first introduced in [42] and formally defined for CCS in [35] and it is fundamental in the PA theory.

Definition 5.1 *A binary relation $\mathcal{R}$ over agents is a* **strong bisimulation** *if $(P, Q) \in \mathcal{R}$ implies, for all $a \in \mathcal{A}ct \cup \tau$:*

1. *Whenever $P \xrightarrow{a} P'$, then for some Q', $Q \xrightarrow{a} Q'$ and $(P', Q') \in \mathcal{R}$;*
2. *Whenever $Q \xrightarrow{a} Q'$, then for some P', $P \xrightarrow{a} P'$ and $(P', Q') \in \mathcal{R}$.*

Any relation which satisfies Definition 5.1 is a strong bisimulation. The notion of *strong equivalence* ($\sim$), also called *strong bisimilarity*, is then introduced as the largest strong bisimulation.

To establish a bisimilarity between two agents P and Q it is necessary to set up a correspondence between the states in the respective transition diagrams. We may build such a correspondence starting with the pair (P, Q) and seeing which pairs must be in correspondence given that (P, Q) is such a pair. The procedure is then repeated with each of the new pairs so introduced until no more new pairs exist. In the end, if all the states in the transition diagrams are contained in some pair, it can be concluded that the two agents are strongly bisimilar.

The agents P and Q described earlier (see Figure 5.6) do not satisfy Definition 5.1. When we start from the pair of initial states (P, Q), both processes can execute action a evolving into P_1 and Q_1 or Q_2. The new pairs (P_1, Q_1) and (P_1, Q_2) seem to be the candidates for setting up the correspondence. However, P_1 and Q_1 (resp. P_1 and Q_2) are not equivalent because they do not offer the same actions. The computation stops and, since not all the states belong to some pair, we can conclude that the two agents are not strongly bisimilar.

As already mentioned, another fundamental notion in the PA theory is the notion of congruence. An equivalence relation is a congruence when it is preserved by all combinators of the language. Strong equivalence is a congruence and therefore allows the full advantages of compositionality. In fact, whenever two agents P and Q are strongly equivalent, they can be interchanged in any complex system S, with confidence that its behaviour remains the same. This means that model verification and model simplification can be carried out on the components within a model, rather than across the whole model at once.

Strong bisimilarity is rather restrictive because every action an agent may perform must be matched by an action of the same type in the equivalent agent — even τ actions. However, we would like to consider equivalent terms like $a.Nil$ and $a.\tau.Nil$ since their observable behaviours are the same: they both execute an action a and then terminate. The latter term in fact has an internal state change that we wish to consider invisible. For this purpose, a weaker notion of bisimulation, called *weak bisimulation* ($\approx$), has been defined [35]. In this case it is required that each τ action in one term must be matched by zero or more τ actions in the other term. Under weak bisimulation it can be proved that $a.Nil \approx a.\tau.Nil$ (while $a.Nil \not\sim a.\tau.Nil$). On the other hand, $\approx$ is not preserved by the choice operator: when we compose an agent in choice with

two terms which are equivalent under weak bisimulation, we do not necessarily obtain two expressions which are still equivalent. For instance we have:

$$b.Nil \approx \tau.b.Nil \quad \text{but} \quad a.Nil + b.Nil \not\approx a.Nil + \tau.b.Nil$$

because the second term can autonomously (by executing τ) reach a state in which only b is possible while the first agent must always choose between a and b. This fact implies that $\approx$ is not a congruence. However, a refinement of $\approx$, *observational congruence*, is known to have this desirable property.

A lot of work has been done in the definition of different equivalence notions and we refer the reader to the literature (see for example [49, 50]) for an extensive discussion. Chapter 16 also provides more insight into this topic.

Equational Laws The equivalence notions allow the proof of a set of equational laws that can be used to manipulate the language expressions. If an algebraic characterisation of the equivalence (axiomatisation) is found, the axioms may be used to apply it at the level of the syntax rather than at the level of the underlying transition system. We do not intend to give a complete list of all the equational laws here, but we show some of them below to give an idea of the algebraic axiomatisation of the strong bisimulation relation. More details can be found in the literature; for example, the complete list of the CCS laws can be found in [35].

1) $P + Q \sim Q + P$	2) $P \parallel_S Q \sim Q \parallel_S P$
3) $P + Nil \sim P$	4) $P \parallel_S Nil \sim P$
5) $P + P \sim P$	

The equations 1) and 2) show the commutativity of the " $+$ " and " $\parallel_S$ " operators. Equations 3) and 4) are related to the Nil operator: no external observer can detect if an agent is composed with the Nil agent using " $+$ " or " $\parallel_S$ ". The last law, $P + P \sim P$, states that no external observer can detect if the agent he is observing is composed with one or more copies of itself in a nondeterministic choice: the two agents $P + P$ and P cannot be distinguished in terms of the actions they offer to the environment.

The most important law is called the *expansion principle* or *expansion law* and it expresses the behaviour of a complex system by means of the behaviours of the composite subsystems. Generally, due to the interleaving semantics, the parallel composition of a finite number of agents can be transformed into an equivalent specification in which parallel composition is replaced by the choice and the prefix operators. For example the term $P = a.b.Nil \parallel c.Nil$ can be rewritten as $P = a.b.c.Nil + a.c.b.Nil + c.a.b.Nil$.

In the case of a complex system $P = P_1 \parallel P_2 \parallel \ldots \parallel P_n$ in which each component P_i can proceed independently the overall behaviour of P can be seen as follows:

$$P_1 \parallel P_2 \parallel \cdots P_n \;\sim\; \sum\{a.(P_1 \parallel \cdots \parallel P_i' \parallel \cdots \parallel P_n) : 1 \leq i \leq n, P_i \xrightarrow{a} P_i'\}$$

In its most general form the expansion law is more complicated and we refer the reader to the literature for more details.

Notice that the expansion law makes interleaving between actions explicit since all possible sequences of actions are represented.

When an equivalence relation has been identified which is useful for model simplification, if an axiomatisation exists it opens the possibility of the simplification procedure being automated via a rewriting system, see for example [29].

5.2.6 The PLC multicomputer example revisited

Let us consider again the PLC model presented in Section 5.2.3. Using axiom 2) from the previous section we can see that the derivatives:

$$(Comp_2 \|_{\{start\_cycle\}} Comp_4) \|_{\{acq\_bus, read\_ext\_data\}} Bus_1$$
$$(Comp_4 \|_{\{start\_cycle\}} Comp_2) \|_{\{acq\_bus, read\_ext\_data\}} Bus_1$$

are strongly bisimilar, because

$$(Comp_2 \|_{\{start\_cycle\}} Comp_4) \sim (Comp_4 \|_{\{start\_cycle\}} Comp_2)$$

and $\sim$ is a congruence. Intuitively we can see that this makes sense, from the point of view of an external observer, if one computer is engaged in local computation ($Comp_2$) and the other is trying to acquire the bus ($Comp_4$) it does not matter which computer is in which state because the observable behaviour of the system will be the same in either case.

5.3 Probabilistic Process Algebras

Various extensions of PA have been proposed in the literature. We briefly present here *probabilistic process algebras* [30, 32, 45], a class of languages in which nondeterministic choice has been replaced by probabilistic choice to capture uncertainty about the behaviour of the modelled system. A new operator, let us use the notation $\oplus$, has been introduced for specifying the probabilistic choice between two or more alternatives. For example, the expression $P = a \oplus_p b$ represents a process that can perform action a with probability p or action b with probability $1 - p$. The semantics is given in terms of probabilistic labelled transition systems in which the transition labels can be action types, probabilities, or both.

In [45] a classification of some probabilistic models is presented, distinguishing *reactive, generative* and *stratified* models. In a reactive system the probabilities of the transitions of an agent may depend on the environment in which the agent is placed: the choice of the action to be performed next is driven by what is offered externally. Once the environment has provided one action, the choice becomes internal and probabilistic. In each state in the transition diagram the

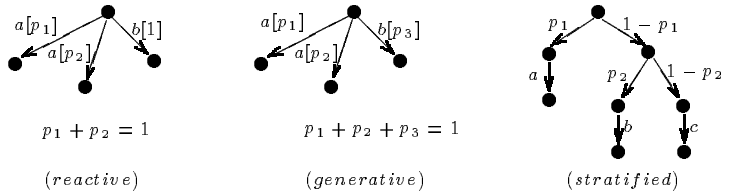


Figure 5.7: Different semantic models for probabilistic processes.

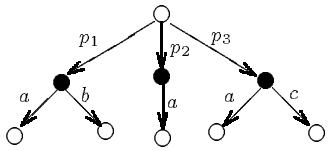


Figure 5.8: Alternating semantic model.

sum of the probabilities of outgoing transitions for each action type must be one (see the left part of Figure 5.7).

In a generative system the transition probabilities are independent of the environment and for each state the sum of the probabilities of the outgoing transitions must be (globally) one (see Figure 5.7 in the middle). If only a subset of actions are offered from the environment, new probabilities are computed by renormalisation.

Stratified models are a generalisation of generative models in which probabilistic and action transitions are kept separate. First a probabilistic choice is solved and then one action is performed. Notice that it is not possible to have nondeterministic transitions because after each probabilistic choice is performed, at most one action is enabled. The right part of Figure 5.7 shows an example of transition diagram for a stratified model.

In [20] a different approach is taken and a language that combines nondeterminism and probability is discussed. A probabilistic choice operator is proposed such that the designer can abstract away from the details of how choices are made but still provides information on the outcome of the choice itself. Probabilistic choice is independent from the environment: it is used for specifying an internal behaviour which cannot be influenced by synchronisation. The semantics is given in terms of labelled transition systems with different types of states, probabilistic and nondeterministic. Figure 5.8 shows an example of transition diagram: since there is a strict alternation between probabilistic states (white circles) and nondeterministic states (black dots) this semantic model has been called the *alternating* model.

Of course a key feature of these process algebras is again the equivalence relations that can be used to compare and analyse models which have been constructed. A probabilistic form of bisimulation has been proposed which aims to capture a notion of indistinguishability when it is assumed that the observer

can witness the probability that an action occurs [32]. For this purpose, a probability measure, μ is defined over the transitions of a labelled transition system, $\mu : \mathcal{P} \times Act \times \mathcal{P} \longrightarrow [0,1]$ ($\mathcal{P}$ is the set of process terms). If we consider all the transitions into a set of process terms, via a given action, this can be extended to a probability measure $\nu : \mathcal{P} \times Act \times 2^{\mathcal{P}} \longrightarrow [0,1]$, such that

$$\nu(P \xrightarrow{a} S) = \sum_{P' \in S} \mu(P \xrightarrow{a} P').$$

The bisimulation for CCS is an equivalence relation and thus it generates equivalence classes over the set of all process terms $\mathcal{P}$. Exploiting this idea, a *probabilistic bisimulation* is defined to be an equivalence relation such that, for any two agents within an equivalence class, for any action $a \in Act$ and any equivalence class S, the probability measure ν of each of the agents performing an a action and resulting in an agent within S, is the same.

Definition 5.2 *A* **probabilistic bisimulation** $\mathcal{R}$*, is an equivalence relation over* $\mathcal{P}$ *such that whenever* $(P,Q) \in \mathcal{R}$ *then for all* $a \in \mathcal{A}ct \cup \tau$*, and for all* $S \in \mathcal{P}/\mathcal{R}$

$$\nu(P \xrightarrow{a} S) = \nu(Q \xrightarrow{a} S)$$

The definition of the probability measure μ, and consequently also ν, depends on whether the process algebra is reactive or generative. Larsen and Skou, [32] define $\mu(P \xrightarrow{a} P')$ for a reactive system, as the probability, given that P performs and action a, that P' is the derivative. In contrast, for a generative system, Jou and Smolka [47], define $\mu(P \xrightarrow{a} P')$ to be the probability that the transition $\xrightarrow{a} P'$ is the one that P performs.

Probabilistic bisimulation is a *strong* relation, in the sense that the silent action, τ, is treated in the same manner as any other action. Weak probabilistic bisimulation can be defined similarly to weak bisimulation for CCS [3].

5.4 Timed Process Algebras

The languages we have discussed so far (untimed and probabilistic) cannot express time delays between events: action executions take zero time and only relative ordering is represented via the traces of the processes. Over the years, various proposals for introducing time into process algebras have been made. These attempts follow two main lines: time can be either *deterministic* or *stochastic*. In the deterministic approach the possibility of finding a calculus for real time communicating systems in the style of CCS or CSP has been investigated [38, 2, 34, 53, 54, 36, 39]. The original motivation in the stochastic approach was that of combining system design and performance evaluation, starting from the first steps of the design [25].

The remaining part of this section is devoted to a brief description of some deterministic time extensions to CCS while languages which follow the stochastic approach will be described in more details in a following section.

Usually in deterministically timed PA some specific constructs are added to an untimed language and/or it is assumed that actions may be delayed. An explicit notion of time is defined by introducing a temporal domain T that can be either discrete (for example $\mathbb{N}$) or dense (for example $\mathbb{R}^+$). Systems are collections of cooperating components that may modify their states either by executing some actions or by letting time progress.

Probably the first timing extension to process algebra is Synchronous CCS [34]. In this language a discrete time domain is assumed and agents proceed in lockstep—i.e. at every instant each agent performs a single action. The expression $P = a.P'$, which in CCS represents an agent P which becomes P' after executing action a, has the following meaning in Synchronous CCS: *"the agent P, existing at time t, executes action a and then becomes P' at time $t+1$."*

In Temporal CCS [36] a discrete temporal domain is assumed and some operators are added to those of standard CCS. For instance, $(t).P$ represents the process that will evolve into P after exactly t units of time (*bounded* idling), $\delta.P$ represents the process that behaves as the process P, but is willing to wait any amount of time before actually proceeding (*unbounded* idling).

The language is given an operational semantics with two different types of transitions: action transitions, similar to those of standard CCS, which describe the functional aspects of the processes and time transitions which describe their temporal aspects. The underlying model is a transition system whose labels are either atomic actions or elements of an appropriate time domain.

In addition to delay operators which postpone the execution of an action by a given (or an unbounded) amount of time, other operators have been added to some timed PA to augment their modelling expressivity. For example, a *timeout* operator has been introduced. It uses two arguments P (the body) and Q (the exception) and a parameter t, and behaves as P if an initial action of P is performed within time t, otherwise it behaves as Q.

Yi's Timed CCS [54] is an extension of CCS which uses a set of idling actions defined as

$$\delta_T \;=\; \{\epsilon(t) \mid t \in T - \{0\}\}$$

Processes can evolve by executing CCS-like transitions or they can idle for a certain amount of time: $P \xrightarrow{\epsilon(t)} Q$ means that *"P will idle for t units of time and then will behave like Q."* The semantic rules of the language are those of CCS plus new ones that specify the idling behaviour of agents. A notion of strong bisimulation has been introduced to determine whether two agents are considered equivalent. This equivalence depends on the *capability* of the observer: an observer who cannot measure time will never tell the difference between two agents that execute their actions one in one second and the other in one year. This is a typical observer in CCS, but in Yi's Timed CCS a more powerful observer, who can also record the time delay between events, is required. In this context in fact, two agents are considered equivalent if they witness the same sequences of actions and the same sequences of delays. Notice that the duration of an activity is not associated with the action that represents it (which is executed in zero time) but it is modelled by a delay that precedes

the action itself.

In Chen's Timed CCS [8], another timed extension of CCS, action prefix captures timing constraints which may apply to the action. The process $a(t)_e^{e'}.P$ can perform the action a between the times e and e' (inclusive); e and e' are called the lower bound and upper bound of action a respectively. For example an action a which can delay indefinitely is represented by $a(t)_0^{\infty}$. Any occurrences of the time variable t which appear in the process P refer to the happening time of a. When the action is performed, say at time u, the variable t becomes bound to the value u. As well as a structured operational semantics, Chen gives a true concurrent semantics to his language in terms of timed synchronisation trees.

Some important properties which influence description capabilities have been defined for timed process algebras. Among them we mention *time determinism* and *action urgency*. The first property states that the progress of time should be deterministic: if a process P evolves into P' after a time t, and the same process evolves into P'' after the same amount of time, then P' and P'' must be the same process. Action urgency states that a process may block the progress of time and enforce the execution of an action before some delay (in some languages only the invisible action is urgent).

Another deterministic extension of PA has been proposed in [19]. In this language there is no distinction between the delay associated with an action and the action itself. The basic assumption is that actions are time-consuming and any action has an associated duration denoted by a natural number. Each sequential subsystem is equipped with a local clock that records the units of elapsed time due to the execution of actions which are local to the subsystem. When two subsystems interact through synchronisation they have to perform the same action at the same time. For this reason a sort of "busy waiting" is necessary when one subsystem is able to execute a synchronising action while the other is not. A notion of *performance equivalence* equates systems that perform the same actions with the same amount of time is discussed. Moreover the necessity of replacing deterministic time durations with time probabilistic distribution functions is recognised as a way to provide an uniform integration of the theories of process algebras and performance evaluation.

5.5 Stochastic Process Algebras

Recently the benefit of associating probabilistically distributed delays with the actions of a process algebra has been recognised, and introduced into several algebraic languages. Modelling the timing behaviour of systems by random variables rather than deterministic times allows the randomness of the real world to be captured. This is fundamental to performance evaluation. Incorporating this idea into a process algebra which facilitates compositional reasoning based on well-defined notions of equivalence, merges two previous distinct approaches to system representation. The result is a constructive methodology for the *specification* and *evaluation* of complex systems.

These new algebraic formalisms are known collectively as *stochastic* or *Marko-*

vian process algebras. Most of them adhere to Markovian assumptions, i.e. only random variables with a negative exponential distribution are used, but a few allow generally distributed random variables. However we will not consider these more general stochastic process algebras in this book, and will only discuss the Markovian process algebras. There are variations between the Markovian process algebras which have appeared in the literature but here we will concentrate on their common features. Therefore we will base our discussion on a *generic* language which we will simply call SPA.

SPA is based on an untimed process algebra in which the basic action has been extended with an exponentially distributed delay. Models in the language can be used to generate an underlying Markov process (see Chapter 9 which can then be used to derive performance measures for the modelled system.

5.5.1 Syntax and informal semantics

The basic process algebra of SPA is derived from CCS and CSP. In fact it is the language presented in Section 5.2.3. In this language systems are modelled as interactions of *components* that can perform a set of actions. Each action is given an associated random duration and is now termed an *activity*. An activity a is described by a pair (α, r) where α is the *type* of the activity and $r \in \mathbb{R}^+$ is the parameter of the negative exponential distribution governing its duration. Whenever a component P can perform an activity an instance of the given probability distribution is sampled. The resulting number specifies how long the component will take to *complete* the action.

The syntax of SPA, our abstract language, is defined as follows:

$$P ::= Nil \mid (\alpha, r).P \mid P + Q \mid P \|_S Q \mid P/L \mid A$$

The functional meaning of each operator is the same as in CCS and CSP (see Section 5.2), but we now also need to consider the durations of the activities.

- **Nil** As before, *Nil* represents the component which is not capable of performing any activities: a deadlocked component. In some timed process algebras such a component also cannot witness the passing of time and so results in the whole model being deadlocked. This is not the case in SPA—although this component cannot progress other components may be able to.

- **Prefix** This gives a component a designated first activity, i.e. the component $(\alpha, r).P$ performs the activity which has type α and a duration which is negative exponentially distributed with parameter r (mean $1/r$) and then evolves as P.

- **Choice** As in the untimed case, the component $P + Q$ represents a system which may behave either as component P or as Q. $P + Q$ enables all the current activities of P and Q and a *race condition* governs the resolution of the choice. This means that we may think of all the activities

attempting to proceed but only the "fastest" succeeding. Thus the first activity to complete identifies one of the components which is selected as the component that continues to evolve; the other component is discarded. The outcome of the race will be random but dependent of the rates of the associated random variables. Thus the non-deterministic choice of CCS and CSP becomes probabilistic in SPA.

Whenever an activity completes the model evolves, now taking on the behaviour of the resulting component. Any other activity which was simultaneously enabled may remember the time for which it was enabled and start from that point whenever it is next enabled. Alternatively it may abandon its spent lifetime and start another lifetime whenever it is next enabled. Under the exponential assumption there is no difference between these two possibilities.

- **Parallel Composition** The component $P \|_S Q$ represents a system in which components P and Q work together to perform activities in the set S. The set S is called the *synchronising* or cooperation set. Both components proceed independently with any activities whose types do not occur in the set S. However, activities with action types in the set S are assumed to require the simultaneous involvement of both components. The resulting activity will have the same action type as the two contributing activities and a rate reflecting their rates. As we will discuss later, several possibilities exist for defining this resultant rate.

- **Hiding** As before, the component P/L behaves as P except that any activities of types within the set L are *hidden*, meaning that their type is not visible outside the component upon completion. Instead they appear as the unknown type τ and can be regarded as *internal delays* by the component. The timing characteristics of the hidden activities are unaffected.

- **Constant** Just as described for CCS, constants are components whose meaning is given by equations such as $A \stackrel{def}{=} P$. Here the constant A is given the behaviour of the component P. Constants can be used to describe infinite behaviours, via mutually recursive defining equations.

Sometimes a component will leave the rate of an activity *unspecified* (this is denoted $\top$, 0, or 1, depending on the language—here we will use $\top$). In this case we say that the component is *passive* with respect to that action type. Such a passive action must be shared with another component via synchronisation. In a final or complete model every passive action must be synchronised with at least one component active with respect to the action type and this component will determine the rate at which the shared activity occurs.

Analogously to GSPN, some of the stochastic process algebra languages also admit *immediate actions* which are completed in zero time with *priority* over timed actions (see Chapter 3). The implications of the inclusion of immediate actions are briefly discussed in Section 5.5.5.

5.5.2 Operational semantics

The deduction rules of the language operators, shown in Figure 5.9, outline the activities that a component can witness. Time is not represented explicitly, but it is assumed that timed activities take some time to complete and consequently the corresponding transitions represent some advance of time.

The rule for the prefix operator can be read as follows: in the component $(\alpha, r).P$ an activity of type α is enabled and the component behaves like P after its execution. The delay associated with the activity is exponentially distributed with rate r.

The other rules are straightforward and are presented without comment except for the rules concerning parallel composition. The first and the second rules represent the independent evolution of the two components while they are executing activities not in the cooperation set S. The third rule represents the cooperation between the two components to achieve a common task. In general, both components of the parallel composition will need to complete some work, as reflected by their own version of the activity, for the common activity to be completed. The rate R of the common activity is given by a function $\phi(r_1, r_2, P, Q)$ which reflects the rates of the individual activities. The different languages adopt different formulae for the definition of the new rate, all satisfying some algebraic requirements that are necessary in order to ensure that the equivalence notions which are defined are also congruences.

In PEPA [27] we have the following expression

$$\phi(r_1, r_2, P, Q) = \frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} \min(r_\alpha(P), r_\alpha(Q))$$

where $r_\alpha(P)$ is the *apparent* rate of action type α in the P component and it is the rate at which an action type α appears to an external observer.

For any component P we can regard its total capacity for carrying out activities of a given type as the sum of the rates of the activities of that type which are enabled by P. This will be the apparent rate of the action type in P. We assume that when two components carry out α in cooperation their total capacity to complete the activity of that type is limited to the capacity of the slower component, i.e. the apparent rate of the synchronising activity is the minimum of the apparent rate of α in the two contributing components. Since each component may enable several α activities we assume that each independently chooses which activity instance takes part in the cooperation. Thus the rate of the synchronising activity is adjusted to reflect these probabilities, i.e. it is the product of the probability that each contributing activity is selected by its enabling component and of the apparent rate of the action type in the cooperation.

In MTIPP [22] the synchronisation between two or more processes leads to the observation of an action with a rate equal to the *product* of the rates of the individual processes i.e. $\phi(r_1, r_2, P, Q) = r_1 \cdot r_2$. We can distinguish two different situations: both actions are active or one action is active and the other is passive. In the first case (pairs of active actions) different *meanings* can be

Prefix

$$\frac{}{(\alpha, r).P \xrightarrow{(\alpha,r)} P}$$

Choice

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P + Q \xrightarrow{(\alpha,r)} P'} \qquad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P + Q \xrightarrow{(\alpha,r)} Q'}$$

Parallel Composition

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P \|_S Q \xrightarrow{(\alpha,r)} P' \|_S Q} (\alpha \notin S) \qquad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P \|_S Q \xrightarrow{(\alpha,r)} P \|_S Q'} (\alpha \notin S)$$

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q'}{P \|_S Q \xrightarrow{(\alpha,R)} P' \|_S Q'} (\alpha \in S) \quad \text{where } R = \phi(r_1, r_2, P, Q)$$

Hiding

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P/H \xrightarrow{(\alpha,r)} P'/H} (\alpha \notin H) \qquad \frac{P \xrightarrow{(\alpha,r)} P'}{P/H \xrightarrow{(\tau,r)} P'/H} (\alpha \in H)$$

Constant

$$\frac{P \xrightarrow{(\alpha,r)} P'}{A \xrightarrow{(\alpha,r)} P'} (A = P)$$

Figure 5.9: Operational semantics of SPA.

associated with the rates which can be interpreted as rates or as *scaling factors*. Let us consider two synchronising components representing a *Processor* and a *Task* executing an action α with rates r_1 and r_2 respectively. The rate r_1 may be interpreted as a measure of the processor speed while the rate r_2 may be interpreted as a scaling factor representing the dimension of the task, i.e. the number of work units composing it. By convention, a standard task has an associated rate equal to 1, values of $r_2 > 1$ represent "small" tasks and values of $r_2 < 1$ represent "big" tasks. Thus the resulting rate of the shared activity captures the amount of work to be done.

The synchronisation between passive and active agents is in general interpreted as a service provided by one agent and required by the other. In this case the rate of the passive action is equal to one (which is the neutral element with respect to the product) and therefore it is the active action that determines the global rate during the synchronisation.

In EMPA [5] synchronisation between processes requires that at most one active action is involved, while all the other actions must be passive. In this language passive actions have a rate equal to zero and the global rate of the synchronisation is given by the *maximum* of the individual rates. In this way the rate of the global action resulting from the synchronisation is determined by the rate of the active action only.

Notice that this choice has implications for compositionality since once an interaction involves one active participant only passive agents may be added later to the system.

In MPA [7] each activity is represented by a pair (α, r) where α, as usual, is the action type but r does not represent a rate; instead it describes the number of invocations of the action α. Normally an action is invoked only once by an agent, but r can also have other values, including real numbers. Moreover, it is assumed that each action α has associated a fixed exponential distribution with rate μ_α and that the invisible action τ has a rate μ_τ equal to one.

Each operation (action) needs a basic time to be performed. If an agent is faster, the parameter r associated with the action is greater than one, if it is slower, the parameter r is smaller than one. Thus, in a certain sense, r can be related to the speed of the agent.

Since each action has a fixed rate the interaction of two α actions results in an action with the same rate μ_α. However, the number of invocations of action α equals the *product* of the number of invocations of both involved activities. In this way, each invocation in one agent is combined with all the invocations in the other agent.

Transition diagram/Derivation graph We have seen in the previous sections that PA terms can be mapped onto transition diagrams whose arcs are labelled with action names, probabilities, and elements of an appropriate time domain.

In SPA actions are time consuming and we can derive a transition diagram whose arcs are labelled with pairs consisting of the action type and the corre-

sponding rate. However, we must take some care. Consider a simple agent P which will repeatedly carry out the action (α, r). For a classical process algebra (and for qualitative analysis) we need only consider which actions are possible in an agent. Thus the agent $P + P$ has the same behaviour as the agent P — both are capable of an action α and subsequently behave as P — so these agents are considered equivalent (see the equational law in Section 5.2). In SPA multiple instances of an action become apparent because the duration of an action of that type will appear to be the minimum of the corresponding random variables (race policy). In the case of exponentially distributed durations this means that the global rate of an action will be the sum of the rates. Thus $P + P$ appears to carry out the first α action at twice the rate of the agent P. Consequently P and $P + P$ cannot be regarded as equivalent. As a consequence in the semantics of the language the number of instances of a transition between states, together with their rates, has to be recognised.

The underlying Markov process There is a clear correspondence between the labelled transition system of an SPA model and a continuous time Markov process. The Markov process underlying any finite SPA component can be obtained directly from the transition diagram: a state of the Markov process is associated with each node of the diagram and the transitions between states are defined by the arcs of the diagram. Since all activity durations are exponentially distributed, the total transition rate between two states will be the sum of the activity rates labelling arcs connecting the corresponding nodes in the transition diagram.

5.5.3 The PLC multicomputer example in SPA

Our PLC example can be specified with SPA by extending the previous untimed specification associating a rate with each action. We obtain the following description:

$$
\begin{array}{lcl}
Comp_1 & = & (start\_cycle, r_1).Comp_2 \\
Comp_2 & = & (local\_comp, r_2).Comp_3 \\
Comp_3 & = & (end\_cycle, r_3).Comp_1 + (req\_ext\_data, r_4).Comp_4 \\
Comp_4 & = & (acq\_bus, r_5).Comp_5 \\
Comp_5 & = & (read\_ext\_data, r_6).Comp_2 \\
 & & \\
Bus_1 & = & (acq\_bus, \top).Bus_2 \\
Bus_2 & = & (read\_ext\_data, \top).Bus_1 \\
 & & \\
PLC & = & (Comp_1 \|_{\{start\_cycle\}} Comp_1) \|_{\{acq\_bus, read\_ext\_data\}} Bus
\end{array}
$$

The rates of the actions *acq_bus* and *read_ext_data* are left unspecified in Bus_1, meaning that this component is passive with respect to them.

5.5.4 Equivalence relations in SPA

Equivalence relations have been used in untimed, probabilistic and timed PA to compare components and to replace a component by another which exhibits an equivalent behaviour, but has a simpler representation. This technique still applies in SPA where different notions of equivalence that cover the functional aspects, the temporal aspects and both of them, have been defined.

As previously, we need to specify the capabilities of the observer. Can the observer record the rate of each activity? Can he record the relative frequency with which alternative activities occur in a given component? Usually it is assumed that the observer has no memory of the past history of the components and bases his comparison only on the current behaviour.

The definition of equivalences for SPA should conservatively extend the notion of bisimilarity introduced in Section 5.2, since this has proved to be fundamental for PA. The difficulty is that now we have to consider not only qualities but also *quantities*, i.e. the rates associated with the action types. In contrast, bisimilarity only considers a (logical) quality: either there is a move between two states or it is impossible.

A fundamental equivalence notion for SPA, called *strong equivalence* [26] or *Markovian bisimulation* [22], considers both qualities and quantities and it has been introduced to ensure indistinguishability under experimentation. We briefly introduce this equivalence here and the reader is invited to see Chapter 16 for further details.

Some definitions are necessary. If P can, by some action, evolve to become Q then Q is said to be a *derivative* of P. The *transition rate* between two components P and Q is denoted by $q(P, Q)$ and is the sum of the activity rates labelling arcs connecting node P to node Q in the transition diagram. The *conditional transition rate* from P to Q via an action type α is denoted by $q(P, Q, \alpha)$. This is the sum of the activity rates labelling arcs connecting the corresponding nodes in the transition diagram which are also labelled by the action type α. The conditional transition rate is thus the rate at which a system behaving as component P evolves to behaving as component Q as the result of completing an activity of type α.

If we consider a *set* of possible derivatives S, the *total conditional transition rate* from P to S, denoted $q[P, S, \alpha]$, is equal to the sum of the conditional transition rates from P to components Q_i belonging to S:

$$q[P, S, \alpha] = \sum_{Q_i \in S} q(P, Q_i, \alpha)$$

The concept of total conditional transition rate is the basis for the definition of strong equivalence since two components are considered strongly equivalent if for any action type α, the total conditional transition rates from those components to any equivalence class, via activities of this type, are the same.

Definition 5.3 *A binary relation $\mathcal{R}$ over components is a* **strong equivalence** *if whenever $(P, Q) \in \mathcal{R}$ then for all α and for all equivalence classes S induced*

by $\mathcal{R}$,

$$q[P, S, \alpha] = q[Q, S, \alpha]$$

In this definition the total conditional transition rate is used analogously to the probability measure ν in Definition 5.2 and [32].

We illustrate the notion of strong equivalence by considering again the PLC example already discussed in Sections 5.2.3 and 5.5.3. Let us take the term $Comp_1 \|_{\{start\_cycle\}} Comp_1$ consisting of two computers synchronising on *start_cycle*, without considering any interaction with the global bus. It is possible to show that $Comp_1 \|_{\{start\_cycle\}} Comp_1$ is strongly equivalent to the term written below:

$$\begin{array}{lcl}
Comp^{eq} & = & (start_cycle, r_1).Comp_1^{eq} \\
Comp_1^{eq} & = & (local_comp, 2r_2).Comp_2^{eq} \\
Comp_2^{eq} & = & (end_cycle, r_3).Comp_3^{eq} + (local_comp, r_2).Comp_4^{eq} \\
 & & \qquad + (req_ext_data, r_4).Comp_5^{eq} \\
Comp_3^{eq} & = & (local_comp, r_1).Comp_6^{eq} \\
Comp_4^{eq} & = & (end_cycle, 2r_3).Comp_6^{eq} + (req_ext_data, 2r_4).Comp_7^{eq} \\
Comp_5^{eq} & = & (local_comp, r_1).Comp_7^{eq} + (acq_bus, r_5).Comp_8^{eq} \\
Comp_6^{eq} & = & (end_cycle, r_3).Comp^{eq} + (req_ext_data, r_4).Comp_9^{eq} \\
Comp_7^{eq} & = & (end_cycle, r_3).Comp_9^{eq} + (req_ext_data, r_4).Comp_{10}^{eq} \\
 & & \qquad + (acq_bus, r_5).Comp_{11}^{eq} \\
Comp_8^{eq} & = & (read_ext_data, r_6).Comp_3^{eq} + (local_comp, r_2).Comp_{11}^{eq} \\
Comp_9^{eq} & = & (acq_bus, r_5).Comp_{12}^{eq} \\
Comp_{10}^{eq} & = & (acq_bus, r_5).Comp_{13}^{eq} \\
Comp_{11}^{eq} & = & (read_ext_data, r_6).Comp_6^{eq} + (end_cycle, r_3).Comp_{12}^{eq} \\
 & & \qquad + (req_ext_data, r_4).Comp_{13}^{eq} \\
Comp_{12}^{eq} & = & (read_ext_data, r_6).Comp^{eq} \\
Comp_{13}^{eq} & = & (read_ext_data, r_6).Comp_9^{eq} + (acq_bus, r_5).Comp_{14}^{eq} \\
Comp_{14}^{eq} & = & (read_ext_data, r_6).Comp_{12}^{eq}
\end{array}$$

Essentially we have taken advantage of the symmetry between the two computers which was discussed in Section 5.2.6, and formed a single component which preserves their combined behaviour from the point of view of an external observer. The transition diagrams underlying $Comp_1 \|_{\{start\_cycle\}} Comp_1$ and $Comp^{eq}$ have 25 states and 51 transitions, and 15 states and 26 transitions, respectively. Portions of the corresponding transition diagrams are shown in Figures 5.10 and 5.11 where for readability we have omitted the states names and we have associated the rates only with some arcs labels.

These two terms are strongly equivalent but the second has a smaller state space. Now we can take advantage of the fact that strong equivalence is a congruence and we can substitute $Comp^{eq}$ for $Comp_1 \|_{\{start\_cycle\}} Comp_1$ in a more complex system. For example we can transform

$$\begin{array}{lcl}
PLC' = & (Comp_1 \|_{\{start_cycle\}} Comp_1 \|_{\{start_cycle\}} Comp_1 \|_{\{start_cycle\}} Comp_1) \\
 & \qquad \|_{\{acq_bus, read_ext_data\}} Bus_1
\end{array}$$

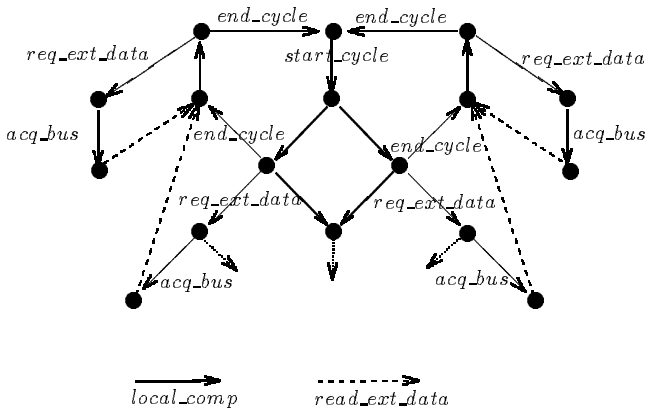


Figure 5.10: Portion of the transition diagram of $Comp_1 \|_{\{start\_cycle\}} Comp_1$.

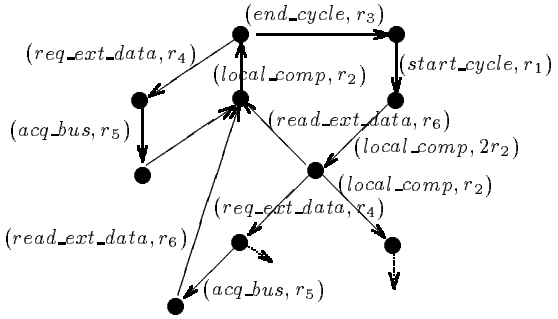


Figure 5.11: Portion of the transition diagram of $Comp^{eq}$.

into

$$PLC^{eq} = (Comp^{eq} \|_{\{start\_cycle\}} Comp^{eq}) \|_{\{acq\_bus, read\_ext\_data\}} Bus_1$$

being sure that they have an equivalent behaviour. The advantage is that the transition diagram of PLC^{eq} has 180 states and 557 transitions while in the case of PLC' we obtain 512 states and 1857 transitions

This notion of equivalence can be used to compare and replace components by simpler ones but it also has interesting stochastic properties which can be exploited for model aggregation and for the efficient analysis of SPA models. These features will be discussed in Chapter 16.

5.5.5 Immediate actions

Some SPA languages allow the use of *immediate* actions which are executed in zero time with priority over timed actions. Immediate actions have been introduced in EMPA [5] taking inspiration from the work on GSPN. Exponentially timed actions have an associated rate $\lambda \in \mathbb{R}^+$ while immediate actions have a rate $\top_{l,w}$ where $l \in \mathbb{N}^+$ is the priority level and $w \in \mathbb{R}^+$ is the associated weight. Different types of choices can be expressed within this language. When the choice involves exponentially timed actions only, the race policy determines which action will be performed. In the case of a state enabling both timed and immediate actions, the immediate actions with the highest priority level are the only actions actually executable, the choice being performed on the basis of their associated weights.

A different approach has been followed in [23] where the basic MTIPP language has been enriched with immediate actions which are represented by their names only. These actions can be *external* or *internal* (denoted τ) referring to whether the environment can influence their execution.

Again, a crucial aspect of the language is the choice operator since the behaviour of a term like $P + Q$ depends on the nature of the choice alternatives. If the choice involves Markovian actions only, as in Figure 5.12 (left), the race policy is used to determine the future behaviour of the term. If the choice involves immediate actions only, the decision is taken on the basis of what is offered from the environment; when the environment offers several of the actions enabled in P and Q, the choice becomes nondeterministic (see middle of Figure 5.12, under the hypothesis that both a and b are offered).

Finally, if immediate and Markovian actions appear in a choice, immediate actions *might* happen instantaneously if they are not blocked from the environment. In particular, since internal actions cannot be prevented, they always happen with priority over timed actions as shown in the right part of Figure 5.12.

Notice that in this language the hiding operator plays a special role since it can be used to give priority to immediate external actions. In the expression $P = (\alpha, r).P_1 + b.P_2$ the action b is executed instantaneously only if the environment also offers the same action instantaneously; otherwise, the environment governs when action b may happen. The environment may also determine that action

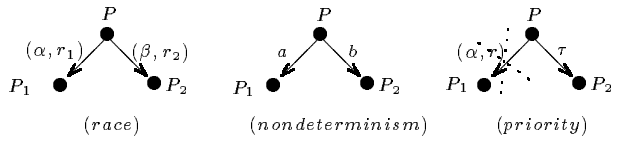


Figure 5.12: Different types of choice: race, nondeterminism, priority.

α occurs first. However, by hiding action b, we can be sure that it cannot be blocked and therefore will always (invisibly) prevent the occurrence of action α. In other words, a priority has been associated with action b by means of hiding.

Reduction of the Nondeterminism In Section 3.2 we discussed how the definition of a TPN starts from a *nondeterministic*, untimed PN model. The inclusion of time is done in such a way that the amount of nondeterminism is reduced, eventually ensuring that the behaviour of the system is specified precisely enough to derive performance measures. In the PN setting, different ways to attack the problem of reducing nondeterminism have been identified. Two of them re-appear in the context of SPA. The incorporation of exponentially distributed delays into an untimed process algebra leads to a stochastic reduction of nondeterminism in the sense of Section 3.2.2. Due to the race condition, nondeterminism is replaced by assigning a probability to each alternative.

In the presence of immediate actions the situation is different: nondeterminism between timed transitions is reduced stochastically, while nondeterminism between immediate transitions is not. This implies that the amount of nondeterminism can make the specification too imprecise to allow the derivation of performance measures. To enhance the precision of such specifications one relies on a variant of nondeterminism reduction in the style of Section 3.2.1. A reduction is performed by postulating some *external influence*. As an example, consider the middle example in Figure refchoice. The term $a.P_1 + b.P_2$ describes a nondeterministic choice, when (and only when) both action a and action b may happen. By postulating an additional external influence we may resolve this choice. For instance, a synchronisation on action b with the inactive process Nil prevents the occurrence of the b branch of the choice. In this way, the amount of nondeterminism is reduced.

Deterministic reduction in the style of Section 3.2.3 has also received some attention in the more general context of models with nondeterminism and probabilities [51]. For this purpose the notion of a *scheduler* is introduced. A scheduler is a function that for any state determines the next transition to be executed, taking into account the history of the system. If the scheduler has the possibility to "roll dice", this form of deterministic reduction turns into a stochastic reduction.

5.6 Comparison with Petri nets

As we have seen throughout this chapter GSPN and SPA have undergone similar developments. In both cases the original definitions did not include any temporal information, and they were used only for the qualitative analysis of concurrent systems. The stochastically timed extensions allowed the study of the systems' quantitative properties too, since the two formalisms now formed high-level description languages for Markov processes. In both cases also, as well as the stochastically timed actions there have been subsequent developments to include immediate actions.

In this section we consider similarities and differences between the two formalisms. We start with the definition of a mapping between them which provides a formal means of investigating the relationship between them. We show how to map the basic operators of our SPA language into GSPN models. In the remainder of the section we compare the formalisms in terms of model construction, qualitative and quantitative analysis. Our comparison is largely informal as we focus on the facilities that are offered to the modeller, rather than theoretical definitions of modelling power etc.

Many comparisons of untimed Petri nets and process algebras have appeared in the literature, concentrating on their different representations of causality and concurrency. Our motivation is different: our intention is to help the reader develop a better understanding of the formalisms, individually and in relation to each other.

5.6.1 Mapping SPA operators into GSPN models

We have already seen in Section 5.5.2 that operational rules are associated with each algebraic operator of SPA, to provide a structured operational semantics. Now we take a different approach to semantics and we associate a net model with each algebraic operator. We follow the *compositional* approach already proposed in [18, 48] for untimed models, and in [4, 46] for timed models.

This compositional approach to (net) model construction requires that the net of a complex language expression is obtained starting from the subnets of the composite subterms. For each operator *op* in the language, we will describe the operation $op_{\mathcal{N}}$ that has to be performed at the net level to reproduce its effect. These operations are based on the well-known net operations of transition and place superposition.

We consider *labelled* GSPN defined as 7-tuples $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{w}, \mathbf{m_0}, l \rangle$ where $P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{w}, \mathbf{m_0}$ are the same of the previous chapters and l is the labelling function $l : T \to Act$ that maps transition names into action names, whose meaning will be immediately clear in the translation examples below. We are not introducing a formal definition here, but rather explaining everything in terms of simple examples; the reader is invited to consult [46] for further details.

The most straightforward mapping is that of the Nil operator which can be translated into a single marked place, not connected to any transition.

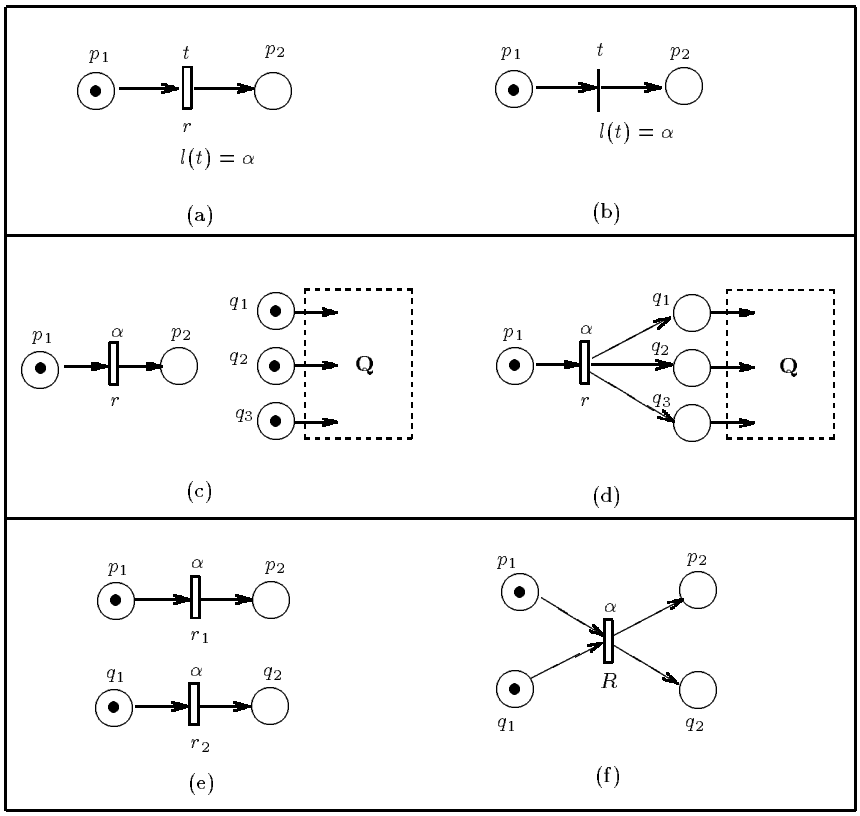


Figure 5.13: GSPN mapping of SPA operators.

The (timed) basic element in our SPA is the activity (α, r) composed of two pieces of information: the action type α and the rate r. Its translation is shown in Figure 5.13(a): p_1 and p_2 are the input (initial) and the output (final) places of the basic element and the timed transition t models action α. In the case of immediate actions we have the same net structure, but the transition t is now immediate, as shown in Figure 5.13(b).

Note the difference between *transitions* and *actions*: since in SPA different terms can perform actions of the same type, several transitions may represent the same action type. However, by definition, transitions need to be distinguishable, i.e., they must have distinct names. Therefore, we use the labelling function l to map transition names into action names[1] so that it is possible that $l(t_i) = l(t_j) = \alpha$ but $t_i \neq t_j$.

Consider the term $P = (\alpha, r).Q$, constructed by the prefix operator. Its mapping into a net model is obtained by first translating the components (α, r) and Q in isolation (see Figure 5.13(c)); then, to represent the "$\cdot$" operator, the

[1] In the following figures we associate the action name $l(t)$ with each transition t.

final place of (α, r) is superposed on all the places which are initially marked in the net modelling Q (see Figure 5.13(d)). Notice that the initial places of Q (q_1, q_2, q_3) are no longer marked.

The translation of the term $P \|_S Q$ requires superposition of transitions and is again performed in two steps. First we map P and Q into GSPN models; then we use an appropriate scheme to translate the $\|_S$ operator. We distinguish two cases:

action types $\alpha \notin S$: we consider the union of the nets representing P and Q[2];

action types $\alpha \in S$: we superpose transitions with the same label α and compute appropriate synchronisation rates.

The net depicted in Figure 5.13(f) shows the translation of the term $P = (\alpha, r_1).P_1 \|_{\{\alpha\}} (\alpha, r_2).Q_1$. Both processes execute an α activity that belongs to the synchronising set; their parallel composition is obtained by superposing the two α transitions. The value of the rate R depends from the choice of the function $\phi(r_1, r_2, P_1, Q_1)$ discussed in Section 5.5.2.

It is only possible to synchronise actions of the same type, i.e. timed with timed, immediate with immediate. This means that at the net level we will (correctly) superpose only transitions of the same type.

The translation of choice uses place superposition. As previously, for the term $P + Q$ we first translate the components P and Q, and then we translate the operator "+" using the Cartesian product of their input places. For example, consider $R = P + Q$ where $P = (\alpha, r_1).P_1 \| (\beta, r_2).P_2$ and $Q = (\gamma, r_3).Q_1$.

The nets representing the two components, P and Q, and the one representing R are shown in Figure 5.14(a) and Figure 5.14(b). The initial places in the composed net are obtained by considering the Cartesian product of the input places of the nets of P and Q (see the places p_1, p_2 and q_1 and the places $p_1 \times q_1$ and $p_2 \times q_1$ in the figure).

The translation of the term P/H is obtained by mapping the term P into the corresponding GSPN model, and by relabelling all the transitions whose associated labels belong to the set H to τ.

In our SPA there is no explicit recursion operator and infinite behaviours are obtained using constants. For example, the expression $P \stackrel{def}{=} (\alpha, r_1).(\beta, r_2).P$ describes a component that can perform an infinite number of α and β actions. The translation of recursive terms like P can again be performed in two steps. First, we translate P as shown in Figure 5.14(c); then we recognise that we have to repeat the same behaviour. Instead of repeating the same net structure, we can close the net obtaining a GSPN model in which the transitions labelled α and β can fire infinitely many times (Figure 5.14(d)). Here we have restricted ourselves to a simple form of recursion in which the recursive term is composed of a sequence of prefixing operators. In [18] a more general translation for the CCS recursion is discussed and several problems which arise when translating recursive terms are presented; details can be found in the literature.

[2] When the set S is empty, we have the (pure) parallel composition of independent components which is translated as the union of the starting nets (Figure 5.13(e)).

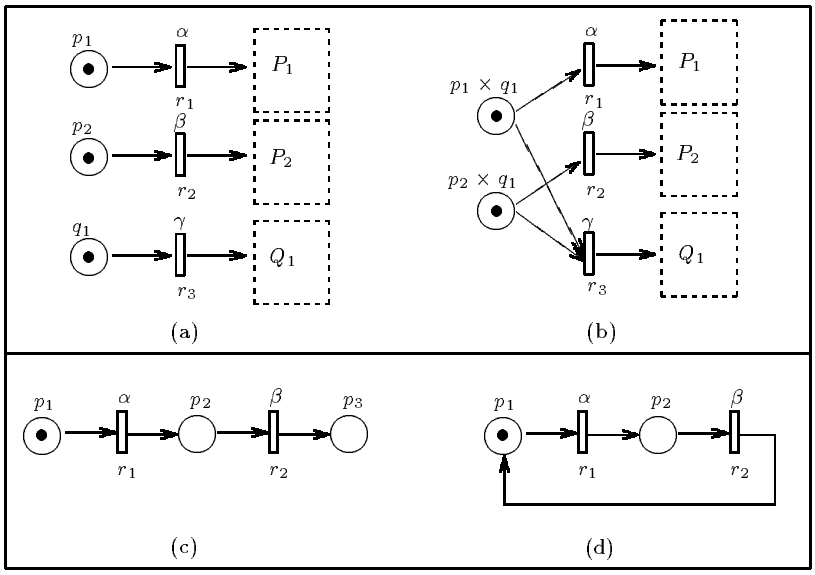


Figure 5.14: GSPN mapping of SPA operators (continue).

We end this section by inviting the reader to observe that, by applying the semantics rules just presented to the PLC model of Section 5.2.3, we obtain a net which is isomorphic[3] to that shown in Figure 2.9 of Section 2.3.1.

5.6.2 Comparison of model construction styles

In this section we examine the different styles in which GSPN and SPA, as high-level description languages for Markov processes, express the behaviour of systems.

At a notational level the difference seems significant since SPA is a textual language and GSPN has a graphical notation. However, at the other extreme, if we consider the class of Markov processes which can be expressed, it is clear that *any* Markov process can be expressed as a degenerate form using either formalism. In GSPN a place is associated with each state and appropriate transitions are inserted between the places to represent the transitions in the Markov process. The place corresponding to the initial state is marked by a single token. Similarly for SPA: a component is associated with the initial state of the Markov process, with a derivative for each subsequent state; activities capture the transitions. Neither of these comparisons reflect the modelling styles of the languages, which is what we wish to compare: for example, in the degenerate GSPN the notions of distributed state and local evolution are lost.

Five different aspects of modelling style, which highlight the differences between the formalisms, are considered below.

[3] Ignoring timing information.

State vs action orientation GSPNs have a very clear notion of state, the distribution of tokens in the places of the net. We can regard a GSPN model as an association of a state (the initial marking) with a graph structure, where the graph structure specifies how the state is modified (state evolution). Note that there are two distinct languages: one to define the structure, and one to record the state.

There is no explicit notion of state in the syntax of SPA. However, at the semantic level, in the labelled transition system, a state is associated with each syntactic term. Thus for an SPA model each derivative of the initial expression representing the model is considered to be a state.

Observe that, in general, it is not possible to distinguish by notation between the derivatives of a model (its states) and the complete model specification—they are all just language terms. The model definition will usually consist of a set of equations: at least one for each component and at least one to specify the interactions between the components. This last equation defines the structure of the model, a structure which will be reflected in all the derivatives, or states. So in some senses this is analogous with the graph structure of the GSPN. But note that it is also analogous to the initial marking since this equation specifies how many instances of each component are active in the model.

From a modelling point of view, GSPN is focussed on both states and actions, while SPA is focussed on actions. Given an arbitrary marking of a GSPN model it is usually possible, by considering the number of tokens in a given place, to immediately infer the state of the system, e.g. "bus is free." In contrast, the information that can be immediately extracted from an SPA model during its evolution is in terms of the actions which the model (system) could perform, e.g. "acq_bus" action is enabled. This may implicitly tell us that the bus is currently free. This distinction has consequences for the definition of performance measures which will be discussed in Chapter 8.

To see how the generation of states in the two paradigms differs, consider a simple system in which a job has a choice between two possible evolutions: it may perform action α at rate r_1 followed by action β at rate r_3, or it may choose to perform action α at rate r_2 followed by action β at rate r_3. A representation of this system in the two formalisms is given in Figure 5.15, by SPA on the left and GSPN[4] on the right. The derivative set of the SPA model has two elements, $\{P_1 + P_2,\ (\beta, r_3).(P_1 + P_2)\}$, while the reachability set of the GSPN model has three different states, $\{(1,0,0),\ (0,1,0),\ (0,0,1)\}$.

In SPA, after action (α, r_1) $((\alpha, r_2))$ takes place, action (β, r_3) is executed, without making any distinction of whether β is executed as an action of the first component (P_1) or of the second one (P_2). In the GSPN model instead the two β actions are represented by the two distinct transitions t_3 and t_4 triggered by a condition on two different places. Thus there is a "history" of which component executed the α action remembered by the model.

This example shows how, in SPA, terms with a common future are considered to represent the same state, whereas in GSPNs this is not necessarily so. We

[4] This GSPN model is obtained by applying the semantic rules introduced in Section 5.6.1.

$$
\begin{aligned}
Q &\stackrel{def}{=} P_1 + P_2 \\
P_1 &\stackrel{def}{=} (\alpha, r_1).(\beta, r_3).Q \\
P_2 &\stackrel{def}{=} (\alpha, r_2).(\beta, r_3).Q
\end{aligned}
$$

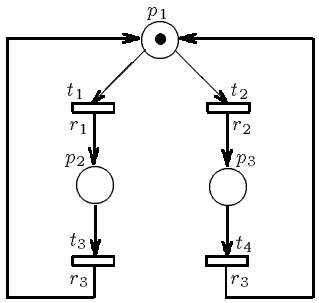


Figure 5.15: SPA and GSPN models of a simple system.

could have modelled the system with a GSPN where p_2 and p_3 (t_3 and t_4) are fused in a single place (transition), but this does not appear a very "natural" model for the given system, since the two possible evolutions of the job are described separately. Notice that this folding would be possible only because both transitions t_3 and t_4 have the same rate. Nevertheless, when the two β actions (the two transitions t_3 and t_4) have different rates, the two corresponding states are kept separate in both formalisms.

Static vs dynamic model entities Process algebras distinguish between dynamic and static combinators. Choice is dynamic since after a choice has been made the syntactic form of the component will, in general, be different: e.g. $P+Q \longrightarrow P'$. Parallel composition is a static combinator since the syntactic form of the component is the same after evolution regardless of whether a shared or an individual activity was completed, e.g. $P \|_S Q \longrightarrow P' \|_S Q$.

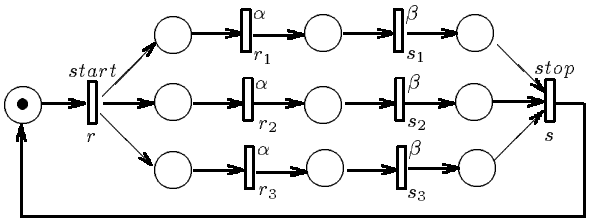
In SPA, entities within the system are represented as components in the model. If a model is to generate an ergodic Markov process it is necessary that the parallel components of a model are static—they are not created or destroyed as the model evolves. The initial term of an SPA model shows all the parallel components which are going to exist during the life of the model, and therefore an ergodic SPA model will have the same number of such terms throughout its evolution. Choices may occur within such components and these represent alternative modes of behaviour but not alternative structures.

In GSPN there is no notion of static components. However, a related concept is that of *P-semiflows* of the net (see Chapter 6). These are subsets of places such that a weighted sum of tokens in those places is constant for all reachable states. We observe that if an SPA term is built as the composition of N distinct components, then the equivalent GSPN has at least N P-semiflows where all the places have weight equal to 1. We refer to Chapter 6 for the discussion about *P- and T-semiflows*.

In a GSPN model, entities within the system are associated with the tokens of the Petri net. As the net evolves the number of tokens may vary, reflecting the dynamic behaviour of the system, according to the firing rule. This models the interactions between the entities in the system. For example, in the case of

the GSPN model of the PLC example, certain tokens represent the computers and others the buses. Tokens representing computers move from state to state (e.g. performing a local computation, reading external data) through the firing of transitions; when a computer is using the bus this is represented as a single token, although two entities are involved.

As another example, consider a simple parallel program containing a parbegin/parend section. In the GSPN this can be simply modelled using a fork & join structure as shown in the upper part of Figure 5.16. Representing this system in SPA produces an alternative view of the system (see the lower part of Figure 5.16). The GSPN representation has one process which becomes split into three and then recombined to form a single process again. The static nature of the SPA models forces a representation with three processes which are initially and finally constrained to act together, only free to act independently during the middle phase of their execution.



$$
\begin{aligned}
P_{i0} &\stackrel{def}{=} (start, r).P_{i1} \\
P_{i1} &\stackrel{def}{=} (\alpha, r_i).P_{i2} \\
P_{i2} &\stackrel{def}{=} (\beta, s_i).P_{i3} \\
P_{i3} &\stackrel{def}{=} (stop, s).P_{i0} \\
System &\stackrel{def}{=} ((P_{10} \|_{\{start,stop\}} P_{20}) \|_{\{start,stop\}} P_{30})
\end{aligned}
$$

Figure 5.16: GSPN and SPA models of the fork & join structure.

Modelling abstraction One of the skills of an experienced modeller is choosing an appropriate level of abstraction at which to construct a model. Although this is largely a question of judgement it is aided by flexibility in the way a system may be presented in a paradigm.

For example, consider a system which is comprised of two identical instances of an entity, which are independent. In SPA this would be modelled as the component $Q \stackrel{def}{=} P \| P$. The component P could have any behaviour but for simplicity we assume that it repeatedly carries out activity (α, r) followed by (β, s): $P \stackrel{def}{=} (\alpha, r).(\beta, s).P$.

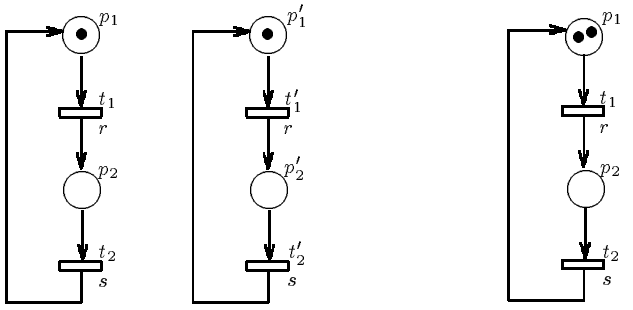


Figure 5.17: Alternative GSPN representations of two instances of the same entity.

There are two possible GSPN models of this behaviour. In the first, the net representing the behaviour of the entity is constructed, and repeated instances of the component are represented by repeated instances of the same net (Figure 5.17(a)). Alternatively, the single net structure representing the entity may be marked by two tokens in its initial place to represent the repeated structure (Figure 5.17(b)), if we assume an infinite server discipline for the transition t_1.

In SPA repeated instances of the same entity are usually distinguished so all possible interleavings of states are represented, i.e. we distinguish between $P' \| P$ and $P \| P'$. However, recent work [17, 24] has aimed to take advantage of the fact that the identity of the component which has completed the activity (α, r) is unimportant. This is analogous to the compact GSPN representation of Figure 5.17(b) in which the marking $(1, 1)$ models a such a situation.

Compositionality Compositionality is a central feature of model construction in SPA, resulting in models which are easy to understand and readily modified. The expression $Q \stackrel{def}{=} P \| P$ shows that the system is comprised of two identical but independent components, without detailed information about the behaviour of P. It follows that an SPA term may have a lot of embedded behaviour, defined in a separate expression. This leads to a hierarchical approach to model construction. The resulting model has a structure which reflects the structure of the system itself. Moreover this structure may be exploited during analysis. Model components can be developed by different modellers and libraries of re-usable components may be established.

In contrast, with GSPN the model is constructed as a *flat* representation of the system, all modelling primitives conveying the same amount of information. In fact, both the notions of state and firing can be viewed as compositional since their are based on the knowledge of local information. Hence there is nothing within the GSPN formalism which makes compositionality impossible. Indeed, although Petri nets (timed or untimed) are not compositional in nature, there

have been efforts to add composition operators to the formalism: composition is based on superposition of places and transitions (see Chapter 12), and more recently it has been defined in the more structured approach of the Box Calculus [12], and in the case of Well Formed nets in [37]. Chapter 16 will discuss in detail the notion of compositionality at the net level.

Operator abstraction We can consider two forms of operator abstraction in performance modelling paradigms: *functional* and *temporal* abstraction. Functional abstraction allows some behaviour of the system to be abstracted away because it is more detailed than necessary for the current model. Temporal abstraction allows some of the timing information within a system to be abstracted away as irrelevant in the current model.

Temporal abstraction is provided by immediate transitions or actions when they are used to represent timed actions, the duration of which is negligible compared with that of other actions within the model. It is assumed that these events do not have a significant impact on the performance of the system. Note that, at a conceptual level, this is distinct from the use of immediate transitions to represent logical actions, to which no time can be associated.

Functional abstraction in SPA is provided by the hiding operator. Activities of type τ are considered to be internal to the component in which they occur. Thus hiding allows an interface to a model or component to be defined as we have already discussed. This is particularly powerful when used in conjunction with parallel composition as it may restrict the interactions of a component. We invite the reader to see Chapter 16 for a discussion on functional abstraction in the context of (compositional) SPN.

5.6.3 Qualitative and quantitative analysis

Since GSPN and SPA have both evolved from formal system description techniques, models in either paradigm can be regarded as a functional representation of the system, as well as a performance representation. The choice, in both GSPN and SPA, to use a distribution with infinite support for the delays, allows functional properties of the timed models to be proved with the same techniques used for their untimed counterpart.

Graph-based analysis, also called state space analysis, may be used to answer many questions about system behaviour when applied to the reachability or derivation graph. For example, the presence of a *dead* state, i.e. a node with no output arc, can be checked on the graph, as well as the reachability of a given state M' starting from a state M can be checked by looking for a direct path in the graph connecting the corresponding nodes.

State space analysis techniques are very powerful, since they allow the proof of many properties of interest by inspection of the graph which contains all possible evolutions of the model. In general, they are considered to be very expensive because the space and time complexity of the graph construction algorithm can exceed acceptable limits. However, in the case of GSPN and SPA models, intended for performance evaluation, construction of this graph is

essential in any case to generate the underlying Markov process. State space analysis techniques as well as other analysis techniques are discussed in Chapter 6.

The solution techniques applied to compute quantitative results in GSPN and SPA are identical, based on the numerical solution of the underlying Markov process. Starting from a GSPN (SPA) model, the associated reachability (derivation) graph is obtained and then reduced to the corresponding continuous time Markov process. This is then numerically solved to compute the steady state probabilities.

Being a more mature formalism, there has been more work on efficient algorithms for finding and solving this Markov process in the case of GSPN, and there has been a certain effort towards "less expensive" solution methods. Some of the efficient algorithms for the construction and solution of the associated Markov process have already been imported into SPA, and work is progressing on others. Again, we invite the reader to see the next chapters for major details about quantitative analysis techniques and efficient solution methods.

5.7 Conclusions

In this chapter we have given an introduction to stochastic process algebra languages describing their development from classical process algebras. In many ways their development has been similar to the development of stochastic Petri net formalisms such as GSPN. In both cases in fact the starting point was an untimed formalism which has been extended to also take into account probabilistic and/or timing information. We have restricted ourselves to the discussion of the main modelling features provided by SPA; (most of) the associated analysis techniques presented later in this book can be applied to SPA too.

In Section 5.6 we informally compared SPA and GSPN and before ending this chapter we briefly summarise some modelling features observing both similarities and important differences.

The model construction facilities of the two formalisms have contrasting strengths and weaknesses. One of the strengths of Petri nets is that causality, conflict and concurrency are clearly depicted within a model and this is true for GSPN models as well. GSPN offer the modeller an explicit notion of state and the graphical notation, which can give an intuitive understanding, and can also be easier to grasp initially. Tool support for GSPN is much more sophisticated largely due to the fact that this formalism has been established for a longer time. This has several consequences: clearly the user is given greater support when using the tool, and the greater support will generally lead to higher confidence on the part of the user.

In contrast, in process algebra based formalisms causality is not exhibited and there is no explicit notion of state. However, the structure within an SPA model is very clearly represented since a compositional structure is apparent in the model. Compositionality provides the possibility of reusability of model components. Modelling is generally an expensive and time-consuming task:

building a new model by composing previously developed components could offer substantial savings. Moreover, the structure makes the model easier to understand, more manageable, and can be exploited for both qualitative and quantitative analysis. Having equivalence relations which are developed and considered to be an integral part of the language means that SPA models can be reasoned about and, in some circumstances, automatically transformed into simpler forms.

If the question is - *what formalism should we use?* - we answer that we do not advocate the adoption of one and the discarding of the other. Instead, we feel that some problem domains will be better suited to expression in one formalism while other problem domains will be better expressed by the other. Thus a modeller would be wise to be familiar with both techniques.

We hope that this introductory presentation gave an intuitive idea of what SPA is, and we invite the reader to read the next chapters bearing in mind that many of the analysis techniques that will be presented in the GSPN context could be (although not always easily) adapted to the SPA domain.

5.8 Bibliographic remarks

Compared to untimed Petri nets, classical process algebras are a more recent formalism since they first appeared in the literature at the end of eighties when the algebraic languages, the Calculus of Communicating Systems (CCS) and the Communicating Sequential Processes (CSP), were introduced. A general introduction to process algebra and CCS may be found in Milner's book [35], probably the best known book on this subject. A good reference for examples and case studies is [14]; a practical introduction to formal methods for specifying concurrent system (based on CCS) may also be found in the recently published book [13]. Concerning CSP, the interested reader may refer to Hoare's book [28].

After the introduction of classical process algebras several extensions, concerning the addition of probabilities and/or timing information, appeared in the literature as we have discussed throughout this chapter.

For timed process algebras, the survey paper by Nicollin and Sifakis [39] provides an overview of the different proposals. More details about the specific languages may be found in the papers [38, 2, 34, 53, 54, 36] already cited in Section 5.4. To the best of our knowledge there is no survey on probabilistic process algebras, and we recommend the papers [32, 45] and [30] for more details about these extensions. Concerning the stochastic extensions, we suggest different references, one for each language we have briefly mentioned in Section 5.5, i.e. [27] for PEPA, [22] for MTIPP, [5] for EMPA, and [7] for MPA.

The development of software tools to support the (S)PA analysis has allowed the usability of the formalism for studying relatively complex systems to be demonstrated. Information about the Concurrency Workbench, a software tool for classical process algebras, may be found in [10]; in the case of SPA we recall the papers [15, 21] which describe more recently developed tools.

The material presented in this chapter covers only some of the basic aspects

of the (untimed and timed) algebraic formalism. For a complete overview of its potential, extensions and applications the interested reader can either see the list of papers cited in the bibliography (which is not exhaustive) or the papers published in the proceedings of conferences and workshops on concurrency and related topics in which sections on process algebras are organised. For example, we highlight the International Conference on Concurrency Theory (CONCUR), whose proceedings are published in the series Lecture Notes in Computer Science. Recently, some sections devoted to (stochastic) process algebras have also been organised within the Petri nets conferences (the comparison section is based on the two papers [46, 11] which have been presented at the Petri Nets and Performance Modelling Workshop). As with Petri nets, more abstract work on process algebras also appears in various conferences devoted to theoretical computer science, for example ICALP and LICS.

Starting in 1992, the Process Algebra and Performance Modelling workshop takes place every year with the aim of bringing together researchers and practitioners interested in the development and application of process algebras to performance modelling. The focus is on the interplay between functional and performance analysis in a process algebra setting. Since its inception, the workshop has taken place in Edinburgh, Erlangen, Torino, Twente and Verona. The 1998 workshop, organised by the University of Verona, was a satellite of CONCUR. The 1999 workshop will be held in Zaragoza in conjunction with the International Workshop on Petri Nets and Performance Modelling (PNPM) and the International Conference on Numerical Solution of Markov Chains (NSMC).

Bibliography

[1] J.C.M. Baeten, editor. *Applications of Process Algebra*, volume 17. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1990.

[2] J.C.M. Baeten and J.A. Bergstra. Real time Process Algebra. *Formal Aspects of Computing*, 3, 1981.

[3] C. Baier and H. Hermanns. Weak Bisimulation for Fully Probabilistic Processes. In O. Grumberg, editor, *CAV '97: Computer Aided Verification (Haifa, Israel)*, volume 1254 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 1997.

[4] M. Bernardo, L. Donatiello, and R. Gorrieri. Giving a Net Semantics to Markovian Process Algebras. In *Proc. 6th International Workshop on Petri Nets and Performance Models*, Durham, NC, 1995.

[5] M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, 1998. To appear.

[6] G. Brebner. A CCS-based investigation of deadlock in a multi-process electronic mail system. Technical Report CSR-17-91, The University of Edinburgh, April 1991.

[7] P. Buchholz. Markovian Process Algebra: Composition and Equivalence. In U. Herzog and M. Rettelbach, editors, *Proc.* 2^{nd} *Workshop on Process Algebra and Performance Modelling*, Erlangen, 1994.

[8] L. Chen. *Timed Processes: Models, Axioms and Decidability.* PhD thesis, University of Edinburgh, 1993. CST-101-93.

[9] R. Cleaveland and M. Hennessy. On the consistency of truly concurrent operational semantics. In *Proc. of LICS.* IEEE, 1988.

[10] R. Cleaveland, J. Parrow, and B. Steffen. The Concurrency Workbench: a semantics-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, 15:36–72, January 1993.

[11] S. Donatelli, J. Hillston, and M. Ribaudo. A comparison of Performance Evaluation Process Algebra and Generalized Stochastic Petri Nets. In *Proc. 6^{th} Intern. Workshop on Petri Nets and Performance Models*, Durham, NC, USA, October 1995.

[12] R. Devillers E. Best and J.G. Hall. The Box Calculus: a New Causal Algebra with Multi-level Communication. In *Advances in Petri Nets*, volume 609 of *LNCS*, 1992.

[13] C. Fencott. *Formal Methods for Concurrency.* International Thomson Publishing, 1996.

[14] G.Bruns. *Distributed Systems Analysis with CCS.* Prentice Hall, UK, 1997.

[15] S. Gilmore and J. Hillston. The PEPA workbench: A tool to support a Process Algebra based approach to performance modelling. In *Proc. Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Vienna, 1994.

[16] S. Gilmore, J. Hillston, and L. Recalde. Elementary Structural Analysis for PEPA. Technical Report ECS-LFCS-97-377, Laboratory for Foundations ofComputer Science, University of Edinburgh, December 1997.

[17] S. Gilmore, J. Hillston, and M. Ribaudo. An Efficient Algorithm for Aggregating PEPA Models. Technical report, University of Edinburgh, 1998. Submitted for publication.

[18] U. Goltz. On representing CCS programs by finite Petri nets. In *Mathematical Foundations of Computer Science*, volume 324 of *LNCS*, 1988.

[19] R. Gorrieri, M. Roccetti, and E. Stancampiano. A theory of processes with durational actions. *Theoretical Computer Science*, 1994.

[20] Hans A. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Dept. of Computer Science, University of Uppsala, September 1991.

[21] H. Hermanns, U. Herzog, U. Klehmet, M.Siegle, and V. Mertsiotakis. Compositional Performance Modelling with the TIPPtool. In *Proc. of 10th Int. Conference on Modelling Techniques and Tool for Computer Performance Evaluation*, LNCS. Springer-Verlag, 1998.

[22] H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras - Between LOTOS and Markov Chains. *Computer Networks and ISDN Systems*, 30 (9-10):901–924, 1998.

[23] H. Hermanns, M. Rettelbach, and T. Weiß. Formal characterisation of immediate actions in SPA with nondeterministic branching. *The Computer Journal*, 38(6):530–541, 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.

[24] H. Hermanns and M. Ribaudo. Exploiting Symmetries in Stochastic Process Algebras. In *Proc. of 12th European Simulation Multiconference (Manchester, UK)*. SCS Europe, 1998.

[25] U. Herzog. EXL — Syntax, Semantic and Examples. Technical Report 16/90, Friedrich-Alexander University Erlangen-Nürnberg, IMMD VII, 1990.

[26] J. Hillston. Compositional Markovian modelling using a process algebra. In *Proc. 2nd International Workshop on the Numerical Solution of Markov Chains*, Raleigh, North Carolina, January 1995.

[27] Jane Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[28] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[29] P. Inverardi and M. Nesi. Deciding Observational Congruence of finite state CCS expressions by rewriting. *Theoretical Computer Science*, 139:315–354, 1995.

[30] C-C. Jou and S.A. Smolka. Equivalences, Congruences and Complete Axiomatizations of Probabilistic Processes. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR '90*, volume 458 of *LNCS*, pages 367–383. Springer-Verlag, August 1990.

[31] Joost-Pieter Katoen. *Quantitative and Qualitative Extensions of Event Structures*. PhD thesis, Centre for Telematics and Information Technology, April 1996.

[32] K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1), 1991.

[33] A. Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationship to Other Models of Concurrency*, volume 255 of *LNCS*. Springer Verlag, 1987.

[34] R. Milner. Calculi for Synchrony and Asynchrony. *Theoretical Computer Science*, 25, 1983.

[35] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[36] F. Moller and C. Tofts. A Temporal Calculus of Communicating Systems. In *CONCUR '90*, volume 458 of *LNCS*. Springer Verlag, 1990.

[37] I. Rojas Mujica. *Compositional contruction and analysis of Petri nets systems*. PhD thesis, University of Edinburgh, 1998. CST-142-98.

[38] X. Nicollin, J.L. Richier, J. Sifakis, and J. Voiron. ATP: An algebra for timed processes. In *Proc. of the IFIP TC 2 Working Conference on Programming Concepts and Methods*, Sea of Galilee, Israel, April 1990.

[39] X. Nicollin and J. Sifakis. An overview and synthesis on Timed Process Algebra. In *Real-Time: Theory in Practice*, volume 600 of *LNCS*. Springer Verlag, 1991.

[40] E.-R. Olderog and C.A.R. Hoare. Specification Oriented Semantics for Communicating Processes. *Acta Informatica*, 23:9–66, 1986.

[41] E.R. Olderog. Operational Petri nets semantics for CCSP. In *Advances in Petri Nets*, volume 266 of *LNCS*. Springer Verlag, 1987.

[42] D. Park. Concurrency and Automata on Infinite Sequences. In *Proc.* 5^{th} *GI Conf. on Theoretical Computer Science*, volume 104 of *LNCS*, 1981.

[43] Doron Peled, Vaughan Pratt, and Gerard Holzmann. *Partial Order Methods in Verification*, volume 29 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, chapter Debate '90: An electronic discussion on true concurrency. American Mathematical Society, 1997.

[44] G.D. Plotkin. An operational semantics for CSP. Technical Report CSR-114-82, The University of Edinburgh, May 1982.

[45] B. Steffen R. van Glabbeek, S.A. Smolka and C. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proc.* 5^{th} *IEEE Int. Symp. on Logic in Computer Science*, 1990.

[46] M. Ribaudo. Stochastic Petri nets semantics for stochastic process algebras. In *Proc.* 6^{th} *Intern. Workshop on Petri Nets and Performance Models*, Durham, NC, USA, October 1995.

[47] S.A. Smolka and B. Steffen. Priority as extremal probability. In J.C.M. Baeten and J.W. Klop, editors, *CONCUR '90*, volume 458 of *LNCS*. Springer Verlag, 1990.

[48] R. van Glabbeek and F. Vaandrager. Petri nets models for algebraic theory of concurrency. In *PARLE Parallel Architectures and languages Europe*, volume 259 of *LNCS*, 1987.

[49] R.J. van Glabbeek. The linear time – branching time spectrum (extended abstract). In J. C. M. Baeten and J. W. Klop, editors, *Theories of Concurrency: Unification and Extension (CONCUR '90, Amsterdam, The Netherlands)*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990. Full version available as CWI Amsterdam Report CS-R9029.

[50] R.J. van Glabbeek. The Linear Time – Branching Time Spectrum II: The semantics of sequential systems with silent moves (Extended Abstract). In Eike Best, editor, *Fourth International Conference on Concurrency Theory (CONCUR '93, Hildesheim, Germany)*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81. Springer, 1993.

[51] M. Vardi. Automatic Verification of Probabilistic Concurrent Finite-State Programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science*, pages 327–338, 1985.

[52] G. Winskel. Event structures. In *Petri Nets: Applications and Relationship to Other Models of Concurrency*, volume 255 of *LNCS*. Springer Verlag, 1987.

[53] Wang Yi. Real-Time Behaviour of Asynchronous Agents. In *CONCUR '90*, volume 458 of *LNCS*. Springer Verlag, 1990.

[54] Wang Yi. CCS + Time = an Interleaving Model for Real-Time Systems. In *Proc. of the Eighteenth Colloquium on Automata Languages and Programming*, volume 510 of *LNCS*. Springer Verlag, 1991.

Part II

Properties and Qualitative Analysis

Chapter 6

Logical properties of P/T systems and their analysis

6.1 Basic logical properties

Only a few qualitative properties will be considered in this introductory chapter. They are general in the sense that they are meaningful for any concurrent system, not only for those modeled with Petri nets. Nevertheless, their statements using Petri net concepts and objects make them specially "easy to understand" in many cases. The properties to be considered are:

1) *boundedness*, characterising finiteness of the state space.

2) *liveness*, related to potential fireability in all reachable markings. *Deadlock-freeness* is a weaker condition in which only global infinite activity (i.e. fireability) of the net system model is guaranted, even if some parts of it do not work at all.

3) *reversibility*, characterizing recoverability of the initial marking from any reachable marking.

4) *mutual exclusion*, dealing with the impossibility of simultaneous *submarkings* (p-mutex) or *firing concurrency* (t-mutex).

Consider the net in Figure 6.1.a. Firing t_2 leads to $\mathbf{m} = p_3 + p_4$. Firing now t_4, $\mathbf{m}_1 = p_1 + p_3$ is reached. Repeating ω times the sequence $t_2 t_4$ the marking $\mathbf{m}_\omega = p_1 + \omega\, p_3$ is reached. So the marking of p_3 can be arbitrarily large, place p_3 is said to be *unbounded*. In practice, the capacity of the physical element represented by p_3 should be finite, so an *overflow* can appear, which is a pathological situation.

The maximum number of tokens a place may contain is its (marking) *bound*. A place is bounded if its bound is finite. A net system is bounded if each place is

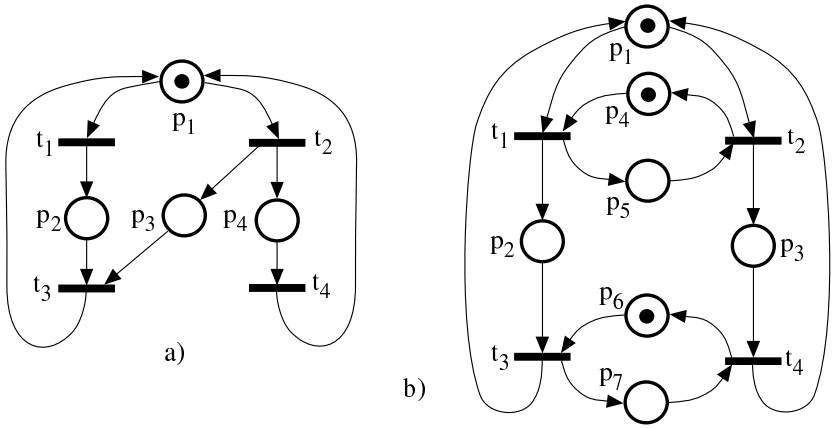


Figure 6.1: On qualitative pathological behaviors: (a) an unbounded, deadlockable (non-live), non-reversible net system; (b) increasing the initial marking (e.g. $\mathbf{m_0}[p_5] = 1$) the live net system can reach a deadlock state!

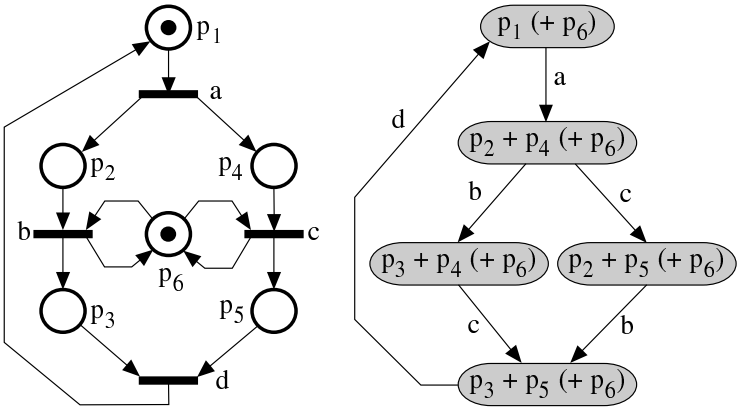


Figure 6.2: Bounded, live and reversible system and its reachability graph

bounded. System boundedness (i.e. all places bounded) is a generally required behavioural property.

For any initial marking we can define on the net structure of Figure 6.2a the following token conservation laws:

$$\mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_3] = \mathbf{m_0}[p_1] + \mathbf{m_0}[p_2] + \mathbf{m_0}[p_3] = k_1(\mathbf{m_0})$$
$$\mathbf{m}[p_1] + \mathbf{m}[p_4] + \mathbf{m}[p_5] = \mathbf{m_0}[p_1] + \mathbf{m_0}[p_4] + \mathbf{m_0}[p_5] = k_2(\mathbf{m_0})$$
$$\mathbf{m}[p_6] = \mathbf{m_0}[p_6] = k_3(\mathbf{m_0})$$

where $\mathbf{m_0}$ is the initial marking and $\mathbf{m}$ any reachable marking. Therefore:

$$\mathbf{m}[p_1] \leq \min(k_1(\mathbf{m_0}), k_2(\mathbf{m_0}))$$
$$\mathbf{m}[p_i] \leq k_1(\mathbf{m_0}); i = 2, 3$$
$$\mathbf{m}[p_j] \leq k_2(\mathbf{m_0}); j = 4, 5$$
$$\mathbf{m}[p_6] = k_3(\mathbf{m_0})$$

The above inequalities mean that *for any* $\mathbf{m_0}$ the net system is bounded. This property, stronger than boundedness, is called *structural boundedness* because it holds independently of the initial marking (only finiteness of $\mathbf{m_0}$ is assumed).

Let us consider now a different scenario where we fire t_1 from the marking in Figure 6.1a. After that, no transition can be fired: a *total deadlock* situation has been reached. A net system is said to be *deadlock-free* if always (i.e. from any reachable marking) at least one transition can occur. A stronger condition than deadlock-freeness is *liveness*. A transition t is potentially fireable at a given marking $\mathbf{m}$ if there exists a transition firing sequence σ leading to a marking $\mathbf{m}'$ in which t is enabled (i.e. $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}' \geq \mathbf{Pre}[P, t]$). A transition is *live* if it is potentially fireable in all reachable markings. In other words, a transition is live if it never loses the possibility of firing (i.e. of performing some activity). A net system is live if all the transitions are live.

For any initial marking we can define on the net structure in Figure 6.1a, non liveness holds (in fact, a total deadlock can always be reached). Non-liveness for arbitrary initial markings reflect a pathology of the net structure: *structural non-liveness*. A net is structurally live if there exists at least one live initial marking.

At first glance it may be accepted as intuitive that increasing the initial marking (e.g. increasing the number of resources) of a net system "helps" in making it live. A paradoxical behaviour of concurrent systems is the following: The net system in Figure 6.1b shows that increasing the number of resources can lead to deadlock situations (Adding a token to p_5, t_2 can be fired and a deadlock is reached!). In other words, in general, liveness is not *monotonic* with respect to the initial marking. Nevertheless, it can be pointed out that liveness can be marking-monotonic on certain net subclasses.

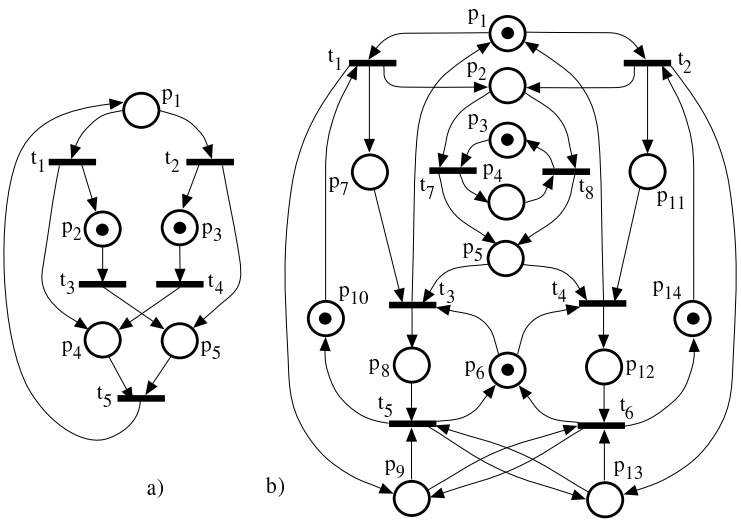


Figure 6.3: On home states: (a) The initial marking is not a home state, but all successor markings are home states; (b) Net system that presents two livelocks, so there are no home states.

A marking is a *home state* if it is reachable from any other reachable marking. The initial marking of the net system in Figure 6.3a is not a home state: after the firing of transition t_3 or t_4 it is not possible to recover this initial marking. Nevertheless, each one of the reachable markings from the initial one is a home state. For some subclasses of net systems the existence of home states is guaranteed [45, 40], but in general the existence of home states does not hold. The net system in Figure 6.3.b [6] is live and bounded but there are no home states. In fact, there exist two different terminal live behaviours that are mutually unreachable. Each one of these terminal live behaviours is called a *livelock*. The set of home states of a net system is called the *home space*. The existence of a home space for a net system is a desirable property because it is strongly related to properties such as ergodicity, of crucial importance in the context of performance evaluation or system simulation.

In the particular case that the initial marking is a home state, the net system is reversible, so it is always possible to return to the initial marking.

Liveness, boundedness, and reversibility are just three different "good" (often required) behavioural properties that may be interesting to study in a net system. Figure 6.4 shows that they are independent of each other, giving examples of the eight cases we may have.

The last basic property we introduce in this section is *mutual exclusion*. This property captures constraints like the impossibility of a simultaneous access by two robots to a single store. Two places (transitions) are in mutual exclusion if can they never be simultaneously marked (fired). For instance, in the net system in Figure 6.2 we can write: $\mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_3] = 1$, so p_1, p_2, and

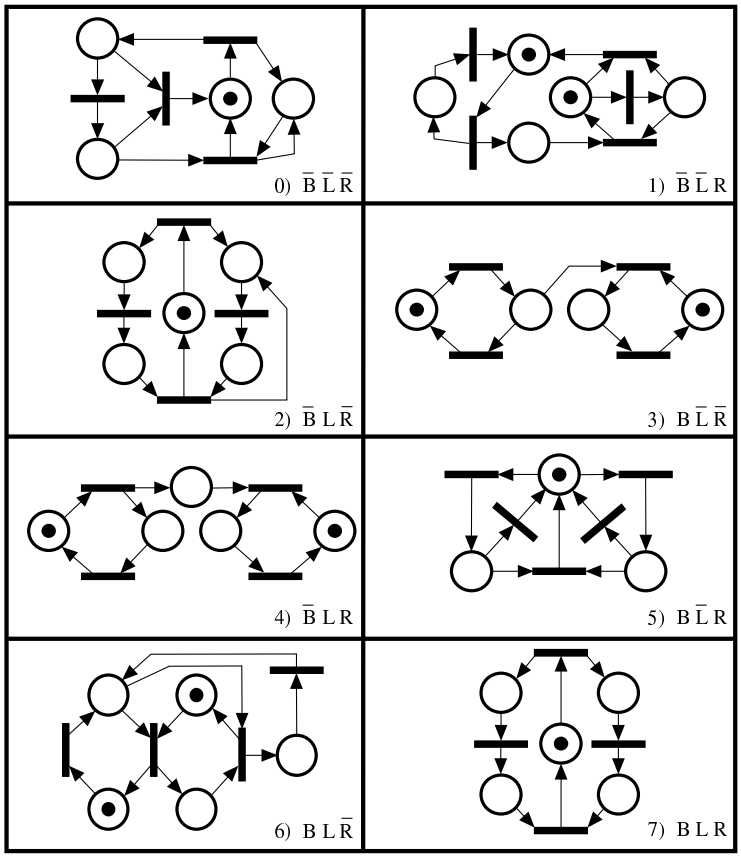


Figure 6.4: Boundedness (B), liveness (L) and reversibility (R) are independent properties

(1)	Bound of place p in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ $\mathbf{b}(p) = \sup\{\mathbf{m}[p] \mid \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})\}$
(2)	p is bounded in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ iff $\mathbf{b}(p) < \infty$
(3)	$\langle \mathcal{N}, \mathbf{m_0} \rangle$ is bounded if all places are bounded
(4)	$\langle \mathcal{N}, \mathbf{m_0} \rangle$ is deadlock-free iff $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})\ \exists t \in T$ such that t is fireable at $\mathbf{m}$
(5)	t is live in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ iff $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})\ \exists \sigma$ such that $\mathbf{m} \xrightarrow{\sigma t} \mathbf{m}'$
(6)	$\langle \mathcal{N}, \mathbf{m_0} \rangle$ is live if all transitions are live
(7)	$\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ is a home state iff $\forall \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})\ \exists \sigma$ such that $\mathbf{m}' \xrightarrow{\sigma} \mathbf{m}$
(8)	$\langle \mathcal{N}, \mathbf{m_0} \rangle$ is reversible iff $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})\ \exists \sigma$ such that $\mathbf{m} \xrightarrow{\sigma} \mathbf{m_0}$
(9)	Mutual exclusion in $\langle \mathcal{N}, \mathbf{m_0} \rangle$: p_i and p_j are in marking mutual exclusion iff $\nexists\ \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that $(\mathbf{m}[p_i] > 0)$ and $(\mathbf{m}[p_j] > 0)$ t_i and t_j are in firing mutual exclusion iff $\nexists\ \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ such that $\mathbf{m} \geq \mathbf{Pre}[P, t_i] + \mathbf{Pre}[P, t_j]$
(10)	Structural properties (abstractions of behavioural properties): $\mathcal{N}$ is structurally bounded iff $\forall \mathbf{m_0}$ (finite) $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is bounded $\mathcal{N}$ is structurally live iff $\exists \mathbf{m_0}$(finite) making $\langle \mathcal{N}, \mathbf{m_0} \rangle$ a live system

Table 6.1: Summarising some basic logical properties

p_3 are in mutual exclusion.

Table 6.1 summarises the definitions of the different properties we introduced in this section.

6.2 Basic analysis techniques for P/T net systems

Conventionally, analysis techniques for Petri nets, are classified as: (1) Enumeration; (2) Transformation; and (3) Structural analysis. Simulation methods have also been applied to study systems modeled with P/T nets. They proceed playing the token game (firing enabled transitions) on the net system model under certain strategies. In general, simulation methods do not allow to prove properties, but they might be of great help for understanding the modeled system or to fix the manifested problems during simulation. Simulation methods are extremely useful when time is associated with the net evolution (timed systems), or when we wish to know the response of the system described with a net in an environment which is also defined by simulation (see Part VII of this book). In this section we do not consider simulation methods and we will only overview the three previously mentioned analysis techniques on P/T nets without interpretation.

Enumeration methods are based on the construction of a *reachability graph* (RG) which represents, individually, the net markings and single transition fir-

ings between them. If the net system is bounded, the reachability graph is finite and the different qualitative properties can be easily verified. If the net system is unbounded, the RG is infinite and its construction is not possible. In this case, finite graphs known as *coverability graphs* can be constructed (see, for example, [31, 33, 18]). In spite of its power, enumeration is often difficult to apply, even in small nets, due to its computational complexity (it is strongly combinatory).

Analysis by transformation proceeds transforming a net system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ into a net system $\mathcal{S}' = \langle \mathcal{N}', \mathbf{m_0}' \rangle$ preserving the set of properties Π to be verified (i.e. $\mathcal{S}'$ satisfies the properties Π iff $\mathcal{S}$ satisfies them). The final goal is to verify the properties Π in $\mathcal{S}'$ in a more easy way than in $\mathcal{S}$. The state space of $\mathcal{S}'$ may be bigger than that of $\mathcal{S}$, but $\mathcal{S}'$ may belong to a subclass for which state enumeration can be avoided.

Reduction methods are a special class of transformation methods in which a sequence of net systems preserving the properties to be studied is constructed. The construction is done in such a way that the net system $\langle \mathcal{N}_{i+1}, \mathbf{m_{0}}_{i+1} \rangle$ is "smaller" (less markings or maintaining the reachability set it has less places or transitions) than the previous in the sequence, $\langle \mathcal{N}_i, \mathbf{m_{0}}_i \rangle$.

The applicability of reduction methods is limited by the existence of irreducible net systems. Practically speaking, the reductions obtained are normally considerable, and can allow the desired properties to be verified directly. Because of the existence of irreducible systems, this method must be complemented by others.

Finally, *structural analysis techniques* investigate the relationships between the behaviour of a net system and its structure (hence their name), while the initial marking acts, basically, as a parameter. In this last class of analysis techniques, we can distinguish two subgroups:

1) *Linear algebra / Linear programming based techniques*, which are based on the net state equation. In certain analysis they permit a fast diagnosis without the necessity of enumeration.

2) *Graph based techniques*, in which the net is seen as a bipartite graph and some "ad hoc" reasonings (frequently derived from the firing rule) are applied. These methods are especially effective in analysing restricted subclasses of ordinary nets.

The three groups of analysis techniques outlined above are by no means exclusive, but complementary. Normally the designer can use them according to the needs of the ongoing analysis process. Obviously, although we have distinguished between transformation/reduction and structural analysis methods, it must be pointed out that most popular reduction techniques act basically on the net structure level and thus can be considered also as structural techniques.

For what concerns the qualitative analysis of interpreted systems, it should be pointed out that conclusions about the properties of the underlying autonomous model can be only sufficient (e.g. for boundedness), necessary (e.g. for reachability) or neither sufficient nor necessary (e.g. for liveness). For par-

ticular net subclasses under "reasonable assumptions" on the behaviour of the environment necessary and sufficient conditions exist.

6.3 Analysis based on the reachability graph

Given a net system, $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$, its reachability graph (recall definition 2.5.4) is a directed graph, $\mathrm{RG}(\mathcal{S}) = (V, E)$, where $V = \mathrm{RS}(\mathcal{S})$ and $E = \{\langle \mathbf{m}, t, \mathbf{m}' \rangle | \mathbf{m}, \mathbf{m}' \in \mathrm{RS}(\mathcal{S})$ and $\mathbf{m} \xrightarrow{t} \mathbf{m}'\}$ are the sets of nodes and edges, respectively.

If the net system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ is bounded, the $\mathrm{RG}(\mathcal{S})$ is finite and it can be constructed, for example, by the algorithm 6.1. It finishes when all the possible firings from the reachable markings have been explored. The tagging scheme in step 2.1 ensures that no marking is visited more than once. Each marking visited is tagged (step 2.1), and step 2.2.3 ensures that the only markings added to V are ones that have not previously added. When a marking is visited, only those edges representing the firing of an enabled transition are added to the set E in 2.2.4.

Let us consider, for example, the net system in Figure 6.2 without the place p_6 and its reachability graph, obtained by applying the algorithm 6.1,. The net system has five markings, thus it is bounded. It is also easy to conclude that all places are 1-bounded. A closer look allows to state that p_1, p_2 and p_3 (p_1, p_4 and p_5) are in mutual exclusion. Moreover, considering RS and the net structure (the pre-function), firing concurrency between transitions b and c can be decided. Observe at this point that introducing p_6 in our net system, the reachability graph does not change, but transitions b and c become in firing mutual exclusion. This example shows that the obtained RG is a *sequentialised observation* of the net system behaviour.

For unbounded net systems, $\mathcal{S}$, $\mathrm{RS}(\mathcal{S})$ is not a finite set and therefore the construction of $\mathrm{RG}(\mathcal{S})$ never ends. Karp and Miller [23] showed how to detect unboundedness of a net system by means of the following condition (incorporated in step 2.2.2 of algorithm 6.1 as a break condition): the system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ is unbounded iff there exists $\mathbf{m}'$ reachable from $\mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\mathbf{m} \xrightarrow{\sigma} \mathbf{m}'$, such that $\mathbf{m} \lneq \mathbf{m}'$ (the repetition of σ allows to conclude on unboundedness because the occurrence of σ strictly increases the content of tokens of the starting marking $\mathbf{m}$).

Coverability graphs allow to obtain finite representations of the RG of unbounded net systems [23, 31, 33, 18]. Roughly speaking, in a coverability graph the set of nodes is a finite set of marking vectors (called the coverability set) that covers all the markings of the reachability set. There is an edge, representing the firing of a transition t, between two nodes, $\mathbf{m}$ and $\mathbf{m}'$ if and only if t is fireable from $\mathbf{m}$ and a marking covered by $\mathbf{m}'$ is reached. The loss of information in the computation of a coverability graph makes that many important properties (e.g. marking reachability or deadlock freeness) cannot be decided on it.

In order to analyse a given property in a bounded net system, the reachability graph is used as the basis for the corresponding *decision procedure*. It allows to

Algorithm 6.1 (Computation of the Reachability Graph)

Input - The net system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$
Output - The directed graph $\mathrm{RG}(\mathcal{S}) = (V, E)$ for bounded net systems

1. Initialize $\mathrm{RG}(\mathcal{S}) = (\{\mathbf{m_0}\}, \emptyset)$; $\mathbf{m_0}$ is untagged;
2. **while** there are untagged nodes in V **do**
   - 2.1 Select an untagged node $\mathbf{m} \in V$ and tag it
   - 2.2 **for** each enabled transition, t, at $\mathbf{m}$ **do**
     - 2.2.1 Compute $\mathbf{m}'$ such that $\mathbf{m} \xrightarrow{t} \mathbf{m}'$;
     - 2.2.2 **if** there exists $\mathbf{m}'' \in V$ such that $\mathbf{m}'' \xrightarrow{\sigma} \mathbf{m}'$ and $\mathbf{m}'' \not\leq \mathbf{m}'$
       then the algorithm fails and exits;
       (the unboundedness condition of $\mathcal{S}$ has been detected)
     - 2.2.3 **if** there is no $\mathbf{m}'' \in V$ such that $\mathbf{m}'' = \mathbf{m}'$
       then $V := V \cup \{\mathbf{m}'\}$; ($\mathbf{m}'$ is an untagged node)
     - 2.2.4 $E := E \cup \{\langle \mathbf{m}, t, \mathbf{m}' \rangle\}$
3. The algorithm succeds and $\mathrm{RG}(\mathcal{S})$ is the reachability graph

decide whether the net system satisfies a given property. All procedures are, in general, of exponential complexity in the size of the net (measured, for example, by the number of places) but they are of polynomial complexity on the size of the reachability graph (measured, for example, by the number of nodes and arcs). The focus of the rest of this section is in two general decision procedures.

In the sequel we will call *marking predicate* to a propositional formula whose atoms are inequalities of the form: $\sum_{p \in A} k_p \mathbf{m}[p] \leq k$, where k_p and k are rational constants and A is a subset of places. Let us consider a net system $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$.

The first group of properties are the so called *marking invariance properties*. A given marking predicate, Π, must be satisfied for all reachable markings (hence the name of marking invariance property): $\forall \mathbf{m} \in \mathrm{RS}(\langle \mathcal{N}, \mathbf{m_0} \rangle)$, $\mathbf{m}$ satisfies Π. Examples of this are:

1) *k-boundedness of place p*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\mathbf{m}[p] \leq k$.

2) *Marking mutual exclusion between p and p'*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $(\mathbf{m}[p] = 0) \vee (\mathbf{m}[p'] = 0)$.

3) *Deadlock-freeness*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\bigvee_{t \in T} (\mathbf{m} \geq \mathbf{Pre}[P, t])$.

Marking invariance properties can be decided through Algorithm 6.2, which is linear in the size of $\mathrm{RS}(\mathcal{S})$: each node is visited no more than once. If the algorithm succeeds, then all reachable markings from $\mathbf{m_0}$ satisfy Π. If the algorithm fails at step 2.2, there is a path in the $\mathrm{RG}(\mathcal{S})$ from $\mathbf{m_0}$, containing at least a marking that does not satisfy Π.

Example 6.1 (Analysis of marking invariances) Let us consider the net system in figure 6.2 for which $\mathrm{RS}(\mathcal{S}) = \{p_1 + p_6,\ p_2 + p_4 + p_6,\ p_3 + p_4 + p_6,$

Algorithm 6.2 (Decision procedure for marking invariances)

Input - The reachability set $\mathrm{RS}(\mathcal{S})$. The property Π.
Output - TRUE if the property is verified.

1. Initialise all elements of $\mathrm{RS}(\mathcal{S})$ as untagged.
2. **while** there is an untagged node $\mathbf{m} \in \mathrm{RS}(\mathcal{S})$ **do**
   2.1 Select an untagged node $\mathbf{m} \in \mathrm{RS}(\mathcal{S})$ and tag it
   2.2 **if** $\mathbf{m}$ does not satisfy Π
   then return FALSE (the property is not verified).
3. Return TRUE

Algorithm 6.3 (Decision procedure for liveness invariances)

Input - The reachability graph $\mathrm{RG}(\mathcal{N}, \mathbf{m_0})$. The property Π
Output - TRUE if the property is verified.

1. Decompose $\mathrm{RG}(\mathcal{N}, \mathbf{m_0})$ into its strongly connected components $C_1, \ldots, C_r$
2. Obtain the graph $\mathrm{RG}^c(\mathcal{S}) = (V_c, E_c)$ by shrinking $C_1, \ldots, C_r$ to a single node, i.e. $V_c = \{C_1, \ldots, C_r\}$. $\langle C_i, t, C_j \rangle \in E_c$ iff there exists $\langle \mathbf{m}, t, \mathbf{m}' \rangle \in E$, such that $\mathbf{m}$ is in the SCC C_i, $\mathbf{m}'$ is in the SCC C_j, and $i \neq j$.
3. Compute the set F of terminal strongly connected components from $\mathrm{RG}^c(\mathcal{S})$
4. **while** there is a $C_i \in F$ **do**
   3.1 **if** C_i it does not contain a $\mathbf{m}'$ satisfying Π
   then return FALSE
   3.2 Remove C_i from F
5. Return TRUE

$p_2 + p_5 + p_6$, $p_3 + p_5 + p_6\}$. The execution of Algorithm 6.2 to verify the mutual exclusion property between places p_5 and p_6 ($\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $(\mathbf{m}[p_5] = 0) \vee (\mathbf{m}[p_6] = 0)$) starts initialising all elements of $\mathrm{RS}(\mathcal{S})$ as untagged (step 1). Then the markings are visited one by one (e.g. in the previous order) until $p_2 + p_5 + p_6$ is visited, where the predicate Π is false, hence the algorithm stops and return FALSE.

The second group of properties are the so called *liveness invariance properties*. For each reachable marking of a net system, $\mathbf{m}$, there exists at least a reachable marking from it satisfying the property Π: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\exists \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$, $\mathbf{m}'$ satisfies Π. Examples of this are:

1) *Liveness of t*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\exists \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$ such that $\mathbf{m}' \geq \mathbf{Pre}[P, t]$.

2) $\mathbf{m}_H$ *is home state*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\exists \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$ such that $\mathbf{m}' = \mathbf{m}_H$.

3) *Reversibility*: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\exists \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$ such that $\mathbf{m}' = \mathbf{m_0}$.

These properties cannot be verified by an exclusive linear inspection of the reachability set (as in algorithm 6.2). The verification requires to find a reach-

able marking, satisfying Π, from each one of the markings in $\mathrm{RS}(\mathcal{S})$. In order to verify the property we will classify the markings of $\mathrm{RS}(\mathcal{S})$ into subsets of mutually reachable markings through the concept of strongly connected component of a directed graph. Therefore, the property will be easily verified checking that each terminal strongly connected component contains at least a marking satisfying Π. We recall now some basic concepts.

A *path* in a reachability graph $\mathrm{RG}(\mathcal{S})$ is any sequence $\mathbf{m}_1 \ldots \mathbf{m}_i \mathbf{m}_{i+1} \ldots \mathbf{m}_k$ of nodes of $\mathrm{RG}(\mathcal{S}) = (V, E)$ where all succesive nodes $\mathbf{m}_i$ and $\mathbf{m}_{i+1}$ in the path satisfy that $\langle \mathbf{m}_i, t, \mathbf{m}_{i+1} \rangle \in E$ for some t. The reachability graph, $\mathrm{RG}(\mathcal{S})$, is *strongly connected* iff there is a path from each node in V to any other node in V. A *strongly connected component* (SCC) of a reachability graph is a maximal strongly connected subgraph. A strongly connected component of a graph will be called *terminal* if no node in the component has an edge leaving the component. The strongly connected components of a digraph (V, E) can be found in order $(|V| + |E|)$ steps (e.g. [26]).

When computing the SCCs $C_1, \ldots, C_r$ of a reachability graph $\mathrm{RG}(\mathcal{S}) = (V, E)$, a new graph $\mathrm{RG}^c(\mathcal{S}) = (V_c, E_c)$ is induced by shrinking the strongly connected components to a single node, i.e. $V_c = \{C_1, \ldots, C_r\}$. For each edge $\langle \mathbf{m}, t, \mathbf{m}' \rangle \in E$, such that $\mathbf{m}$ is in a SCC C_i, and $\mathbf{m}'$ is in a different SCC C_j, there is an induced edge $\langle C_i, t, C_j \rangle \in E_c$. The graph $\mathrm{RG}^c(\mathcal{S})$ is an acyclic digraph. Therefore the terminal SCCs of $\mathrm{RG}(\mathcal{S})$ can be identified with linear complexity in the size of $\mathrm{RG}^c(\mathcal{S})$. This fact will be exploited in the algorithm 6.3 for liveness invariance checking.

Algorithm 6.3 allows to decide liveness invariance properties. The algorithm is of linear complexity in the size of the $\mathrm{RG}(\mathcal{S})$. If the algorithm succeeds, all terminal SCCs contain at least one marking satisfying the property Π, and therefore for all reachable marking there exists at least a successor marking satisfyig the property Π. If the algorithm fails, there exists at least one terminal SCC that does not contain markings satisfying the property Π, and therefore the reachable markings belonging to this SCC (at least) do not satisfy the liveness invariance property.

Remark It is possible to design more specific (efficient) decision procedures for the analysis of a property if we know, a priori, some characteristics of the property to be verified or we know some other properties of the net system to be analysed. For the first case we can consider as an example the reversibility property. It is easy to see that if a net system is reversible then all terminal SCCs must contain the initial marking, i.e. the reachability graph must be strongly connected. For the second case, for example, we may know that the net system is reversible; then liveness of a transition t can be decided checking the existence of an edge in the reachability graph labeled t (since the reachability graph is SC and therefore always is possible to reach the marking from which t can be fired).

Example 6.2 (Analysis of liveness invariances) Let us consider the net system in Figure 6.3.b for which the reachability graph is depicted in Figure

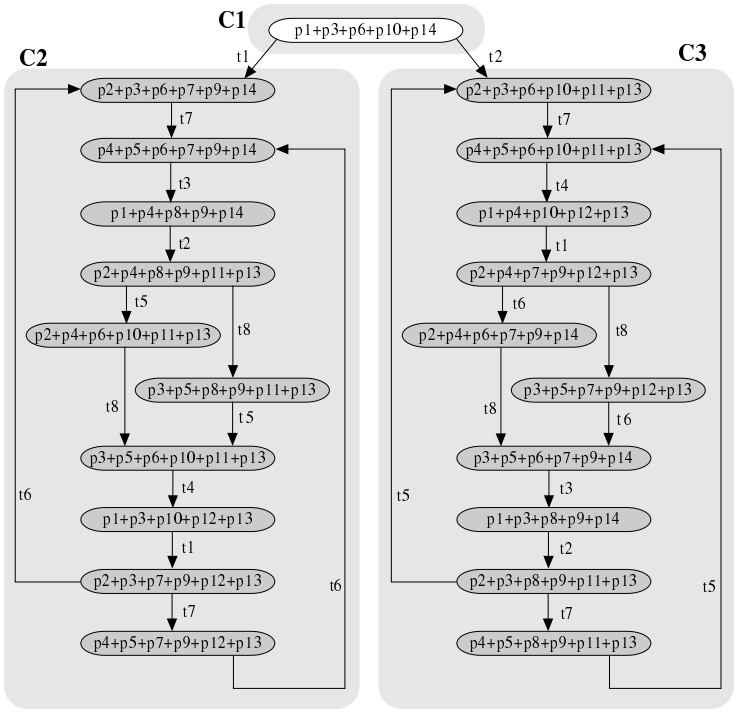


Figure 6.5: Reachability graph of the net system in Figure 6.3.b

6.5. The execution of the algorithm 6.3 to verify the liveness property of this net system ($\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\bigwedge_{t \in T} [\exists \mathbf{m}^t \in \mathrm{RS}(\langle \mathcal{N}, \mathbf{m} \rangle), \mathbf{m}^t \geq \mathbf{Pre}[P, t]]$) requires the computation of the strongly connected components of the $\mathrm{RG}(\mathcal{S})$ (step 1). In this case, there are three SCCs depicted in Figure 6.5 and named C_1, C_2 and C_3. The SCCs C_2 and C_3 are the terminal ones. The step 4 of the algorithm will verify that each one of these two SCCs contains for each transition t a marking $\mathbf{m}^t$ satisfying $\mathbf{m}^t \geq \mathbf{Pre}[P, t]$ (equivalently, contains edges labelled with all transitions of the net). The reader can observe by inspection of the figure that all transitions appear in some edge of C_2 and C_3, therefore the answer of the algorithm will be TRUE.

The execution of the algorithm 6.3 to verify that the marking $\mathbf{m}_H = p_2 + p_3 + p_6 + p_7 + p_9 + p_{14}$ is a home state ($\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\exists \mathbf{m}' \in \mathrm{RS}(\mathcal{N}, \mathbf{m})$ such that $\mathbf{m}' = \mathbf{m}_H$) gives as result FALSE, because the terminal SCC C_2 contains the marking $\mathbf{m}_H$, but the terminal SCC C_3 does not. Therefore, step 3.1 returns FALSE.

From a practical point of view, it is commonly accepted today that systems are too complex to be verified by hand. As a result, analysis increasingly is becoming synonymous with *computer-aided verification* [1]. Computer-aided verification means using a computer, for increased speed and reliability, to perform the analysis steps. For instance, the following example considers the analysis of a property belonging to the group of the so called *synchronic properties* [37], pointing out that an analysis by hand can be very hard.

Example 6.3 Figure 6.6 shows a very simple net system: Parts are sent from *STORE 1* to *STORE 2* and *STORE 3*. The subnet generated by places $\{B, C, E, F\}$ imposes some restrictions on the way parts are distributed to the destination stores (i.e. partially schedule the distribution). The reachability graph is, even if it has been "structured" for more clear presentation, difficult to understand and manage. The reader can try to check on the reachability graph (!) that the imposed distribution strategy is: parts are sent in a 1:1 relation to the destination stores, but allowing sometimes until four consecutive sendings to a given store (i.e. locally adjusting the possible demand, but maintaining the overall fair distribution).

Summarising, analysis techniques based on the reachability graph are only theoretically possible for bounded systems. They are very simple from a conceptual point of view. The problem that makes this approach not practical (impossible) in many cases is its computational complexity: *the state space explosion problem.*

On the other hand, it must be pointed out that the reachability/coverability graphs are computed for a given initial marking. This means that a parametric analysis of a net system (needed in earlier phases of the system design) where the initial marking of some places (e.g. representing the number of resources in the system) is a parameter, is not possible since for each value of the parameter a (completely different) new reachability graph must be computed. Moreover, the reachability graph presents some difficulties in order to analyze properties

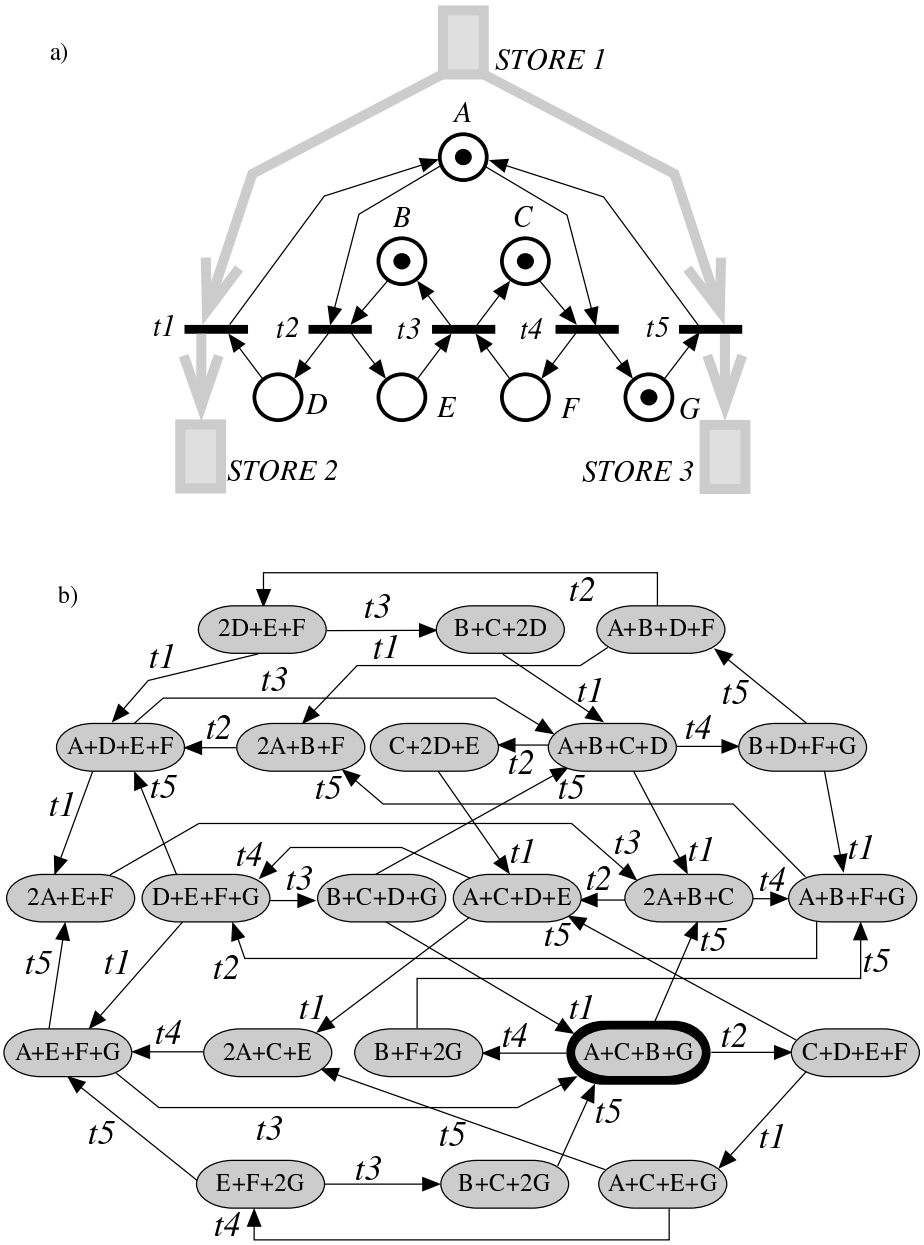


Figure 6.6: Parts of *STORE 1* are sent to *STORE 2* and *STORE 3* according to the strategy defined by the subnet generated by $\{B, C, E, F\}$: (a) the net system; (b) the reachability graph.

where the distinction between conflict and concurrency plays a fundamental role (recall the net in figure 6.2, the reachability graph is the same with place p_6 and without it!). This is because the reachability graph gives a sequentialized view of the behaviour of the net system.

Although these analysis techniques present the drawbacks above mentioned, for bounded net systems they are the more general ones and, in some cases, the only way to verify a given property.

6.4 Net system reductions

In order to paliate the state space explosion problem several techniques has been introduced to obtain *reduced state spaces*. As an example we can cite the stubborn set method [43, 44]. These techniques work directly in the construction phase of the reachability graph maintaining the original net model. In this section we review a different kind of reduction techniques named *net system reductions*. These reductions proceed transforming the net structure and, sometimes, the initial marking.

From an operational point of view, the approach is based on the definition of a kit or catalog of *reduction rules*, each one preserving a subset of properties (liveness, boundedness, reversibility, etc) to be analysed. A reduction rule characterises a type of subnet system (*locality principle*) to be substituted by another (simpler) subnet system.

The preconditions to be fulfilled have a *behavioural* and/or *structural* formulation. Behavioural preconditions can be more powerful for a given initial marking, but their verification is usually much more complex. So preconditions presented here are based on structural considerations and properties of the initial marking (i.e. the initial marking is considered as a parameter).

The design of a catalog of reduction rules is based on a tradeoff between completeness (i.e. transformation capabilities) and usefulness (i.e. applicability).

Given a catalog of reduction rules, analysis by reduction (the transformation procedure) is iterative by nature: Given the property (or properties) to be analyzed, the subset of rules that preserve it (them) is applied until the reduced system becomes irreducible. The irreducible system may be so simple that the property under study is trivially checked (see Figure 6.9.d). In other cases, the irreducible net is just "simpler" to analyse using another analysis technique (e.g. we can obtain a reduced state space on which it is possible to analyse the property that has been preserved in the reduction process). In other words, techniques to analyse net system models are complementary, not exclusive.

Reduction rules are transformation rules interesting for net analysis. When considered in the reverse sense they become expansion rules, interesting for net synthesis: stepwise refinements (or top-down) approach. Examples of this approach can be found in the context of synthesis of live and bounded Free Choice systems [17] or in the definition of subclasses of nets by the recursive application of classical expansion rules as the case of Macroplace/Macrotransition systems

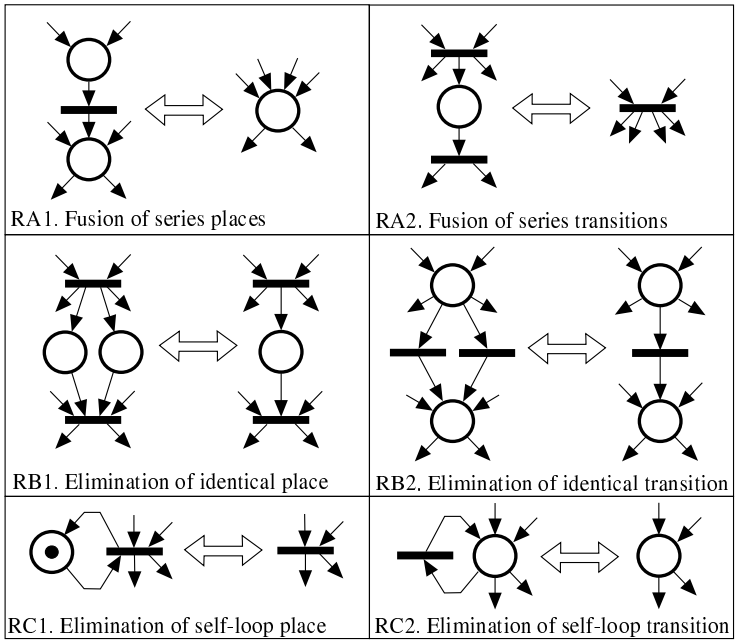


Figure 6.7: A basic reduction kit.

[15]. Using this approach, with adequate expansion rules, the model will verify by construction the specification. This is interesting when compared with the more classical approach based on the iteration of the design and analysis phases until the specification is satisfied. The iterative process has two basic disadvantages:

1) the lack of general criteria for modifying (correcting) a model which does not meet the requirements.

2) the operational difficulty inherent to the validation phase.

Nevertheless, since no kit of reduction rules is complete (i.e. able to fully reduce any system), it is not possible to synthesize an arbitrary system by such stepwise refinements.

A very basic kit of reduction rules is presented. Additional details are given only for the rule of implicit places, which are redundancies in the net system model: if an implicit place is removed, then illusory synchronizations disappear and other reduction rules can be applied.

6.4.1 A basic kit of reduction rules

Figure 6.7 presents graphically the structural and marking conditions for a kit of very particular cases of reduction rules. It is not difficult to observe that they

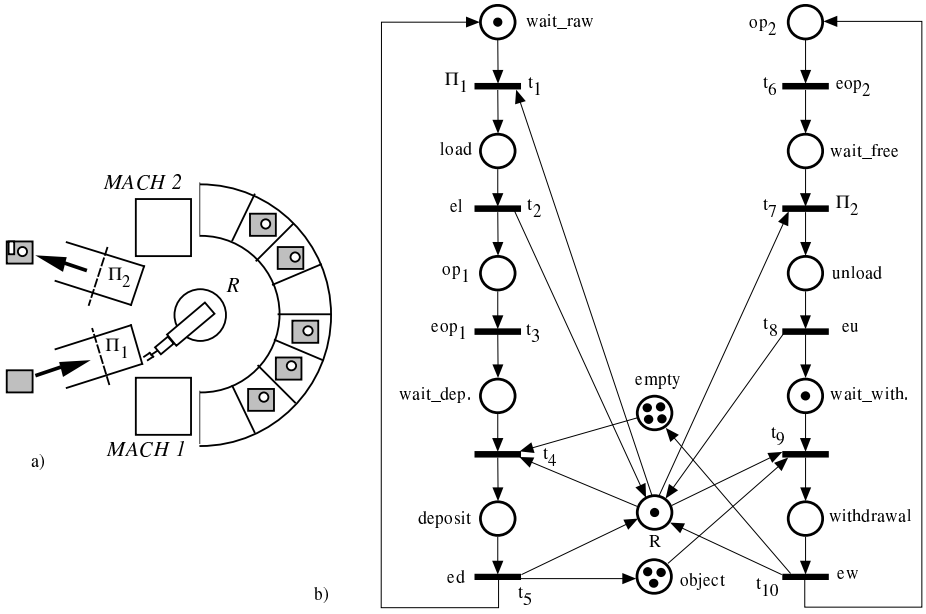


Figure 6.8: a) A production cell with two machines, one robot and a store. b) Net system specifying its behavior.

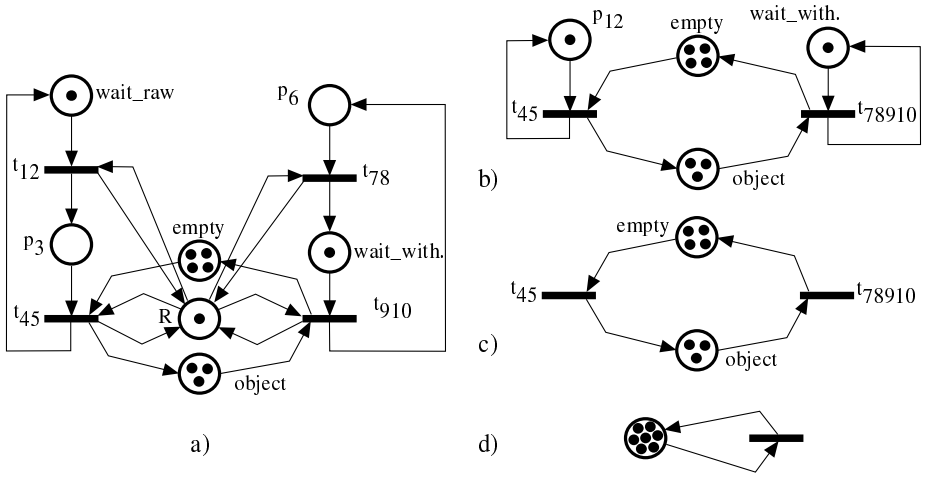


Figure 6.9: The reduction process shows (see (d)) that the net system in figure 6.8 is live, 7-bounded and reversible.

preserve properties such as liveness, the bound of places (thus boundedness), and the existence of home states (but they do not preserve reversibility because the rule *RA1*)

- *RA1* is a particular case of the *macroplace rule* [34]. If the output place of the transition has only this transition as input transition, then the entire kit preserves the reversibility property.
- *RA2* is a particular case of the *transition fusion rule* [3]
- *RB1* and *RC1* are particular cases of the *implicit place rule* [35, 37] (later considered in more detail). Observe that *RC1* can be trivially generalized creating several self-loops in which the place always appears. Liveness, the bound of places, and reversibility are preserved. Moreover if the place contains several tokens, liveness, boundedness (in general not the bound of the net system) and reversibility are preserved.
- *RB2* and *RC2* are particular cases of *identical* and *identity transition rules* [3], respectively.

An interesting remark is the analogy between rules at the same row in Figure 6.7: Basically rules *RX2* are obtained from rules *RX1* by changing the role of places and transitions (*duality*) and reversing the arrows (important only for rules *RA*).

Example 6.4 The local controller attached to the production cell depicted in Figure 6.8.a can be described by the given PN model. The places *wait_load*, *load*, op_1, *wait_dep.*, and *deposit* represent the possible states of *MACH 1*; The place R is marked when the robot is available; The places *empty* and *parts* contain as many tokens as empty slots or parts are available in the temporary buffer, etc. In this model actions are associated to places, e.g., *MACH 2* performs its operations while place op_2 is marked, and transitions represent atomic instantaneous changes of state. External inputs (from plant sensors) condition these possible changes of state, e.g., a load operation is initiated (transition t_1 is fired) when *MACH 1* is waiting for a raw part (*wait_load* marked), the robot is available (R marked), and a raw part is detected by the sensor Π_1 (Π_1 is true). As an example of constraint that is reflected in the model, a deposit operation cannot be initiated unless an empty slot is available. If the self-loops between *empty* and t_4 and between *object* and t_9 were deleted the system could reach a deadlock situation, e.g., *MACH 2* is "withdrawing" a part when there are none available but *MACH 1* cannot deposit any because the robot is busy.

Let us consider now the net system in Figure 6.8.b. The subnet defined by $\mathrm{op}_1 - t_3 -$ wait_dep. verifies the precondition of rule *RA1* (but in the constrained form that preserves reversibility). Thus it can be reduced to a place, p_3 (Fig. 6.9.a). The same holds for $\mathrm{op}_2 - t_6 -$ wait_free that is reduced to p_6 (Fig. 6.9.a). The subnets $t_1 -$ load $- t_2$, $t_4 -$ deposit $- t_5$, $t_7 -$ unload $- t_8$, and $t_9 -$ withdraw $- t_{10}$ can be reduced according to *RA2* (see t_{12}, t_{45}, t_{78} and t_{910} in Fig. 6.9.a). Place R in Fig. 6.9.a is implicit (one of the trivial generalizations

mentioned for *RC1*). Thus it can be removed, and wait_draw $- t_{12} - p_3$ and $t_{910} - p_6 - t_{78}$ can be reduced to p_{12} and t_{78910}, respectively (see Figure 6.9.b). Places p_{12} and wait_with. are implicit (*RC1*) in Figure 6.9.b, thus the net system in Figure 6.9.c is obtained. Playing the token game, a place (e.g. object) can became empty in Figure 6.9.c and t_{45} – object $- t_{78910}$ can be reduced (*RA2*) to a single transition (Fig. 6.9.d). Therefore, the original net system is live, 7-bounded and reversible.

6.4.2 Implicit places

A place in a net system is a constraint to the firing of its output transitions. If the removal of a place does not change the behaviour of the original net system, it represents a redundancy in the system and it can be removed. A place whose removal preserves the behaviour of the system is called an *implicit place*. Two notions of behaviour equivalence are used to define implicit places. The first one considers that the two net systems have the same behaviour if they present the same fireable sequences. That is, this place can be removed without changing the *sequential observation* of the behaviour of the net system (i.e. the set of fireable sequences: interleaving semantics). Implicit places under this equivalence notion are called *sequential implicit places* (SIP). The second notion of equivalence imposes that the two net systems must have the same sequences of steps. In this case the implicit places are called *concurrent implicit places* (CIP) and their removal does not change the possibilities of simultaneous occurrences of transitions in the original net system. Implicit places model illusory synchronisations on their output transitions.

Definition 6.1 *Let $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system and $\mathcal{S}' = \langle \mathcal{N}', \mathbf{m_0}' \rangle$ the net system resulting from removing place p from $\mathcal{S}$. The place p is a*

1. *Sequential Implicit Place (SIP) iff* $\mathrm{L}(\mathcal{N}, \mathbf{m_0}) = \mathrm{L}(\mathcal{N}', \mathbf{m_0}')$, *i.e., the removing of place p preserves all firing sequences of the original net.*
2. *Concurrent Implicit Place(CIP) iff* $\mathrm{LS}(\mathcal{N}, \mathbf{m_0}) = \mathrm{LS}(\mathcal{N}', \mathbf{m_0}')$, *i.e., the removing of place p preserves all sequences of steps of the original net.*

It is easy to see that if a place p is a CIP then it is also a SIP (since the preservation of the sequences of steps implies the preservation of the firing sequences). Nevertheless, the contrary is not true in general. Let us consider, for example, the net in Figure 6.2. The place p_6 is a SIP since its removal does not change the set of firing sequences (the reachability graphs of the original net system and the net system without place p_6 are the same), but the place p_6 is not a CIP because after its removal transitions b and c can occur simultaneously and in the original net system they are sequentialised (i.e. the steps are not preserved). A SIP with self-loops, in order to be a CIP may require more tokens in its initial marking than those making it a SIP (in our example p_6 to be CIP requires two tokens in the initial marking). In [9] it is proven that a self-loop free SIP is also a CIP.

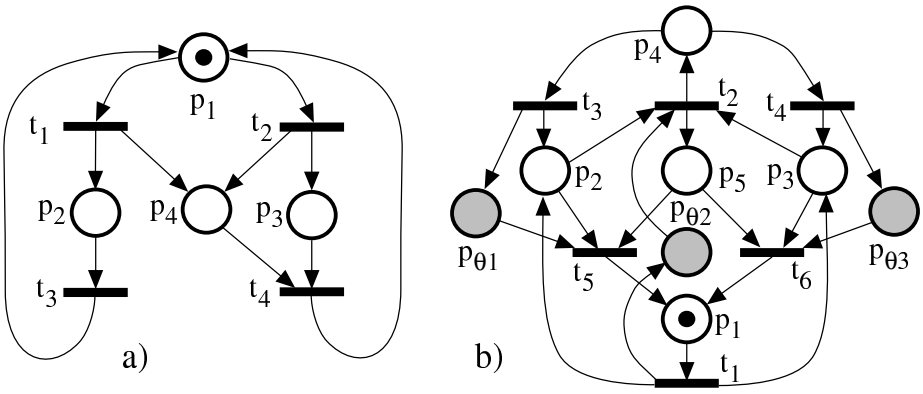


Figure 6.10: a) Place p_4 is firing implicit but not marking implicit. Removing p_4 the "false" synchronisation in t_4 disappears. b) The places in the set $\{p_{\theta 1}, p_{\theta 2}, p_{\theta 3}\}$ (or $\{p_2, p_3, p_5\}$) are CIPs.

Let p be an CIP of the net system $\mathcal{S}$ and $\mathcal{S}'$ the net system $\mathcal{S}$ without p. Let σ_s be a fireable sequence of steps in $\mathcal{S}$, such that $\mathbf{m_0} \xrightarrow{\sigma_s} \mathbf{m}$. The sequence σ_s is also fireable in the net system $\mathcal{S}'$, i.e., $\mathbf{m_0}' \xrightarrow{\sigma_s} \mathbf{m}'$. A trivial consequence of this is that the reached markings in $\mathcal{S}$ and $\mathcal{S}'$, firing the same sequence σ_s, are strongly related: $\forall q \in P \setminus \{p\}$, $\mathbf{m}[q] = \mathbf{m}'[q]$. Moreover, if $\mathbf{s}$ is a step enabled at $\mathbf{m}'$ the following holds: $\mathbf{m}' \geq \mathbf{Pre}' \cdot \mathbf{s} \Longrightarrow \mathbf{m}[p] \geq \sum_{t \in (p^\bullet \cap ||\mathbf{s}||)} \mathbf{s}[t] \cdot \mathbf{Pre}[p,t]$. If p is a SIP the previous property can be writen in the following way: $\forall t \in p^\bullet$, $\mathbf{m}' \geq \mathbf{Pre}'[P',t] \Longrightarrow \mathbf{m}[p] \geq \mathbf{Pre}[p,t]$.

The elimination of a CIP or a SIP preserves: deadlock-freeness, liveness and marking mutual exclusion properties; but it does not preserve: boundedness or reversibility. Moreover, the elimination of a CIP preserves the firing mutual exclusion property, but this is not true for SIPs.

Example 6.5 The net system in Fig. 6.10.a is unbounded (p_4 is the unique unbounded place) and non-reversible (also because of p_4). Place p_4 is a CIP. Removing p_4 the system becomes bounded and reversible! On the other hand, place p_6 in Figure 6.2 imposes firing mutual exclusion between b and c. Being p_6 a SIP, the reduction rule does not preserve firing mutual exclusion. According to the definition, fireable sequences are preserved.

Sometimes it is practical to impose an additional condition to the definition of implicit places, asking their marking to be redundant (computable) with respect to (from) the marking of the other places in the net (i.e. a marking redundancy property). Let us consider the CIP $p_{\theta 1}$ of the net system, $\mathcal{S}$, depicted in Figure 6.10.b. This place is CIP and its marking can be computed from the marking of places p_1, p_2 and p_5: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, $\mathbf{m}[p_{\theta 1}] = \mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_5] - 1$. This class of places will be called *marking implicit places*. Nevertheless, the marking of some implicit places cannot be exclusively computed from the marking of the other places in the net. These places will be called *firing implicit*

places. As an example consider the CIP p_4 in Figure 6.10.a: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$, such that $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$, $\mathbf{m}[p_4] = \mathbf{m}[p_3] + \boldsymbol{\sigma}[t_1]$). The classification of the implicit places into marking and firing implicit places can be applied to the two previously defined classes: CIP and SIP. Because of the additional condition, marking implicit places preserve the state space (i.e., the reachability graph of the net system with and without p are isomorphous), therefore they preserve boundedness and reversibility, too.

Implicit places presented until now are in a behavioural setting. In order to do the verification we must resort to algorithms based on the reachability graph with the inherent limitations and the high associated computational complexity. *Structurally implicit places* is a class of places that become implicit provided they are marked with enough tokens. The characterization of these places and a good bound of the minimum initial marking needed to be implicit can be done efficiently by means of Linear Programming techniques, avoiding the construction of the reachability graph.

Definition 6.2 *Let $\mathcal{N}$ be a net. A place p of $\mathcal{N}$ is a structurally implicit place iff there exists a subset $I_p \subseteq P \setminus \{p\}$ such that $\mathbf{C}[p,T] \geq \sum_{q \in I_p} y_q \cdot \mathbf{C}[q,T]$, where y_q is a nonnegative rational number (i.e. $\exists \mathbf{y} \geq 0$, $\mathbf{y}[p] = 0$ such that $\mathbf{y} \cdot \mathbf{C} \leq \mathbf{C}[p,T]$ and $I_p = ||\mathbf{y}||$.*

Obviously, the above structural condition can be checked in polynomial time. The next property gives the initial marking conditions to be satisfied by a structurally implicit place to become a SIP or a CIP. This condition is based on the solution of a Linear Programming Problem (the LPP in 6.1) that computes an upper bound of the minimal initial marking of a structurally implicit place to be SIP or CIP in the net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$. Because, LPPs are of polynomial time complexity [29], the evaluation of this condition has this complexity.

Property 6.3 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system. A structurally implicit place p of $\mathcal{N}$, with initial marking $\mathbf{m_0}[p]$, is a SIP (CIP) if $\mathbf{m_0}[p] \geq z$, where z is the optimal value of the LPP 6.1 with $\alpha = 1$ ($\alpha = \max\{\sum_{t \in p^\bullet} \mathbf{s}[t] | \mathbf{s} \in \mathrm{LS}(\mathcal{N}, \mathbf{m_0})\}$).*

$$
\begin{array}{lll}
z = & \mathit{min.} & \mathbf{y} \cdot \mathbf{m_0} + \alpha \cdot \mu \\
& \mathit{s.t.} & \mathbf{y} \cdot \mathbf{C} \leq \mathbf{C}[p,T] \\
& & \mathbf{y} \cdot \mathbf{Pre}[P,t] + \mu \geq \mathbf{Pre}[p,t] \quad \forall t \in p^\bullet \\
& & \mathbf{y} \geq 0, \mathbf{y}[p] = 0
\end{array}
\tag{6.1}
$$

If the optimal solution of the LPP 6.1, for a structurally implicit place p, verifies that $\mathbf{y} \cdot \mathbf{C} = \mathbf{C}[p,T]$, then p is a marking implicit place and the following holds: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, $\mathbf{m}[p] = \mathbf{y} \cdot \mathbf{m} + \alpha \cdot \mu$.

Observe that a structurally implicit place, p, can become implicit for any initial marking of places $P \setminus \{p\}$, if we have the freedom to select an adequate initial marking for it. This property is not true for CIPs (or SIPs) that are not structurally implicit places. For example, the place p_{10} in Figure 2.4.a is a CIP but it is not a structurally implicit place. Moreover, the place p_{10} is not implicit if we change the initial marking of place p_4 from 0 to 1.

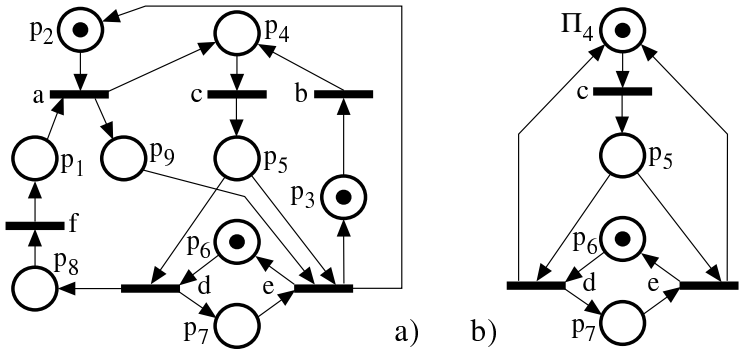


Figure 6.11: Places p_9 and p_2 (or p_2 and p_7) are implicits

Example 6.6 Solving the LPP in 6.1 for the place p_9 in Fig. 6.11.a with $\alpha = 1$ we obtain $z = 0$, for the optimal solution: $\mathbf{y} = [0,\ 0,\ 1,\ 1,\ 1,\ 0,\ 1,\ 0,\ 0]$ and $\mu = -1$. Moreover, $\mathbf{C}[p_9, T] = \mathbf{C}[p_3, T] + \mathbf{C}[p_4, T] + \mathbf{C}[p_5, T] + \mathbf{C}[p_7, T]$. Because $\mathbf{m_0}[p] \geq z = 0$, p_9 is a SIP (since p_9 is self-loop free place it is also a CIP) and can be removed. Being p_9 a marking implicit place we can write: $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, $\mathbf{m}[p_9] = \mathbf{m}[p_3] + \mathbf{m}[p_4] + \mathbf{m}[p_5] + \mathbf{m}[p_7] - 1$.

Once p_9 is removed, a similar computation for p_2 can be done and p_2 is also shown to be a CIP. Figure 6.11.b shows a reduced net system. It can be obtained reducing $p_3 - b - p_4$ into a place (say p_{34}) (*RA1*) and finally $p_8 - f - p_1 - a - p_{34}$ into Π_4. The rule *RA1* allows to fuse Π_4 and p_5. The new place is implicit, so it can be removed. Then a cycle with $p_6 - d - p_7 - e - p_6$ remains. Finally it can be reduced to a basic net, $p_6 - t_{de} - p_6$, with one token. Therefore the original net system is live, bounded. It is also reversible, but we cannot guarantee this because of the fusion of $p_3 - b - p_4$ into p_{34}.

6.5 Linear algebraic techniques

Analysis techniques based on linear algebra allow the verification of properties of a general net system. The key idea is simple, and it has been already commented previously: Let $\mathcal{S}$ be a net system with incidence matrix $\mathbf{C}$. If $\mathbf{m}$ is reachable from $\mathbf{m_0}$ by firing sequence σ, then $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$. Therefore the set of natural solutions, $(\mathbf{m}, \boldsymbol{\sigma})$, of this state equation defines a linearisation of the reachability set $\mathrm{RS}(\mathcal{S})$ denoted $\mathrm{LRS}^{SE}(\mathcal{S})$. This set can be used to analyse properties like marking and submarking reachability and coverability, firing concurrency, conflict situations, deadlock-freeness, mutual exclusion, k-boundedness, existence of frozen tokens (they never leaves a place), synchronic relations, etc. To do so, the properties are expressed as formulas of a first order logic having linear inequalities as atoms, where the reachability or fireability conditions are relaxed by satisfiability of the state equation. These formulas are verified checking existence of solutions to systems of linear equations that are automatically obtained

from them [9]. For instance, if $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S}) : \mathbf{m}[p] = 0 \vee \mathbf{m}[p'] = 0$; then places p and p' are in mutual exclusion. This is verified checking absence of (natural) solutions to $\{\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \wedge \mathbf{m}[p] > 0 \wedge \mathbf{m}[p'] > 0\}$. *Integer Linear Programming Problems* [30] where the state equation is included in the set of constraints can be posed to express optimization problems, like the computation of marking bounds, synchronic measures, etc. [9, 37]. This approach is a generalization of the classical reasoning using linear invariants [25, 27], and it deeply bridges the domains of net theory and convex geometry resulting in a unified framework to understand and enhance structural techniques [9] (see subsection 6.5.1).

Unfortunately, it usually leads to only semidecision algorithms (i.e., only necessary or only sufficient conditions) because, in general, $\mathrm{RS}(\mathcal{S}) \subset \mathrm{LRS}^{SE}(\mathcal{S})$. The undesirable solutions are named *spurious*.

Example 6.7 (Existence of spurious solutions and their consequences in the analysis) Let us consider the net system depicted in Figure 2.3. The corresponding net state equation has the following marking spurious solutions: $\mathbf{m}_1 = 2 \cdot p_4$, $\mathbf{m}_2 = 2 \cdot p_2$, $\mathbf{m}_3 = 2 \cdot p_3$, $\mathbf{m}_4 = 2 \cdot p_5$, $\mathbf{m}_5 = p_2 + p_4$, $\mathbf{m}_6 = p_3 + p_4$. The first four solutions allow to conclude that p_2, p_3, p_4 and p_5 are 2-bounded, while they are really 1-bounded (check it). The solutions $\mathbf{m}_2$, $\mathbf{m}_3$ and $\mathbf{m}_4$ are total deadlocks. Then using the state equation we cannot conclude that the system in Fig. 2.3 is deadlock-free.

Spurious solutions can be removed using certain structural techniques, consequently improving the quality of the linear description of the system [11]. For example, it is clear that adding implicit places, a new system model with identical behaviour is obtained. For some net systems, if the implicit places are chosen carefully, the state equation of the new system may have no integer spurious solution preventing to conclude on the bound of a place or the deadlock freeness of the system.

Example 6.8 (Elimination of spurious solutions) The net system in Figure 6.10.b has been obtained adding the implicit places $p_{\theta 1}$, $p_{\theta 2}$ and $p_{\theta 3}$ to that in Figure 2.3. The above mentioned spurious solutions, $\mathbf{m}_i, i = 1 \ldots 6$; are not solutions of the new state equation. Moreover, we can conclude now that the new (and original) net system(s) were 1-bounded and deadlock-free!

Anyway the algorithms based on linear algebra do decide in many situations, and they are relatively efficient, specially if the integrality of variables is disregarded. (This further relaxation may spoil the quality, although in many cases it does not [13, 37].) Moreover, these techniques allow in an easy way an initial marking *parametric* analysis (e.g. changing the number of customers, size of resources, initial distribution of customers and/or resources, etc). The application of these techniques to the analysis of boundedness and deadlock-freeness properties is illustrated in subsections 6.5.2 and 6.5.3, repectively.

In temporal logic terms, the above outlined approach is well suited for *safety* properties ("some bad thing never happens"), but not so much for *liveness* properties ("some good thing will eventually happen"). For instance, the formula expressing reversibility would be $\forall \mathbf{m} \in \mathrm{LRS}^{SE}(\mathcal{S}) : \exists \boldsymbol{\sigma}' \gneq 0 : \mathbf{m_0} = \mathbf{m} + \mathbf{C} \cdot \boldsymbol{\sigma}'$,

but this is neither necessary nor sufficient for reversibility. The general approach to linearly verify these liveness properties is based on the verification of safety properties that are necessary for them to hold, together with some inductive reasoning [20]. For instance, deadlock-freeness is necessary for transition liveness, and the existence of some *decreasing potential function* proves reversibility [36] (see subsection 6.5.5).

Another important contribution of linear techniques to liveness analysis has been the derivation of *ad hoc* simple and efficient semidecision conditions. In subsection 6.5.4, we present one of these conditions based on a rank upper bound of the incidence matrix, which was originally conceived when computing the *visit ratios* in certain subclasses of net models [8].

The following subsections study linear invariants, marking bounds and boundedness, deadlock-freeness, structural liveness and liveness, and reversibility.

6.5.1 Linear invariants

A *p-flow* (*t-flow*) is a vector $\mathbf{y} : P \to \mathbb{Q}$ such that $\mathbf{y} \cdot \mathbf{C} = 0$ ($\mathbf{x} : T \to \mathbb{Q}$ such that $\mathbf{C} \cdot \mathbf{x} = 0$), where $\mathbf{C}$ is the incidence matrix of the net. The set of p-flows (t-flows) is a vector space, orthogonal to the space of rows (columns) of $\mathbf{C}$. Therefore, the flows can be generated from a *basis* of the space. Natural and non-negative flows are called *semiflows*: vectors $\mathbf{y} : P \to \mathbb{N}$ such that $\mathbf{y} \cdot \mathbf{C} = 0$ ($\mathbf{x} : T \to \mathbb{N}$ such that $\mathbf{C} \cdot \mathbf{x} = 0$). The following terminology is used with semiflows [27]: The *support of a p-semiflow (t-semiflow)*, $\mathbf{y}$ ($\mathbf{x}$): $||\mathbf{y}|| = \{p \in P | \mathbf{y}[p] > 0\}$ ($||\mathbf{x}|| = \{t \in T | \mathbf{x}[t] > 0\}$). A semiflow is *cannonical* iff the g.c.d. of its non-null elements is equal to one. A net is *conservative* (*consistent*) iff there exists a p-semiflow (t-semiflow) such that $||\mathbf{y}|| = P$ ($||\mathbf{x}|| = T$).

The set of cannonical semiflows of a given net can be infinite, since the weighted sum of any two semiflows is also a semiflow. Consider now the case of p-semiflows. A *generator set* of p-semiflows, $\Psi = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_q\}$, is made up of the least number of them which will generate any p-semiflow as follows:$\mathbf{y} = \sum_{\mathbf{y}_j \in \Psi} k_j \cdot \mathbf{y}_j$, $k_j \in \mathbb{Q}$ and $\mathbf{y}_j \in \Psi$. The p-semiflows of Ψ are said to be *minimal*. The following result characterizes the generator set of the semiflows of a net.

Proposition 6.4 *A semiflow is minimal iff it is cannonical and its support does not contain strictly the support of any other p-semiflow. Moreover, the set of minimal semiflows of a net is finite and unique.*

From the above result, the number of minimal semiflows is less than or equal to the number of incomparable vectors of dimension k ($k = |P|$ or $k = |T|$): Number of minimal semiflows $\leq \begin{pmatrix} k \\ \lceil k/2 \rceil \end{pmatrix}$. Where $\begin{pmatrix} \star \\ \star \end{pmatrix}$ denotes a combinatory number and $\lceil \star \rceil$ denotes rounding up to an integer. In practice this number is still too gross a bound for the number of minimal semiflows.

Algorithm 6.4 presents a simple version allowing the computation of the set of minimal p-semiflows, Ψ, from the incidence matrix of the net. Each row of matrix $\mathbf{\Psi}$ memorizes the coefficients of the positive linear combination of rows

Algorithm 6.4 (Computation of the minimal p-semiflows)

Input - The incidence matrix $\mathbf{C}$. A fixed but arbitrary order in P is supposed.
Output - The p-semiflows' matrix, $\mathbf{\Psi}$, where each row is a minimal p-semiflow.

1. $\mathbf{A} = \mathbf{C}$; $\mathbf{\Psi} = \mathbf{I_n}$; { $\mathbf{I_n}$ is an identity matrix of dimension n }
2. **for** $i = 1$ **to** m **do** { $m = |T|$ }
   2.1 Add to the matrix $[\mathbf{\Psi}|\mathbf{A}]$ all rows which are natural linear combinations of pairs of rows of $[\mathbf{\Psi}|\mathbf{A}]$ and which annul the i-th column of $\mathbf{A}$
   2.2 Eliminate from $[\mathbf{\Psi}|\mathbf{A}]$ the rows in which the i-th column of $\mathbf{A}$ is non-null.
3. Transform the rows of $\mathbf{\Psi}$ into canonical p-semiflows and to remove all non-minimal p-semiflows from $\mathbf{\Psi}$ using the characterization of proposition 6.4.

of matrix $\mathbf{C}$ which generated the row of $\mathbf{A}$ with the same index. In step 3 of the algorithm, the rows of $\mathbf{A}$ are null and therefore each row $\mathbf{\Psi}[i]$ is a p-semiflow: $\mathbf{\Psi}[i] \cdot \mathbf{C} = 0$. The same algorithm can be used to compute the set of minimal t-semiflows if the input of the algorithm is the transpose of the incidence matrix.

The computation of minimal p-semiflows ($\mathbf{y}$) and minimal t-semiflows ($\mathbf{x}$) has been extensively studied [10]. Anyhow an *exponential number* of minimal semiflows may appear. Therefore the time complexity of this computation cannot be polynomial.

P- and T- semiflows are dual structural objects leading to linear invariant laws on the possible behaviours. These invariant laws arise from the structure of the net, and the initial marking plays the role of a parameter specifying a particular behaviour for the net. The two following classes of linear invariants can be obtained,

1) From p-semiflows: $\mathbf{y} \in \mathbb{N}^n$, $\mathbf{y} \cdot \mathbf{C} = 0 \Longrightarrow \forall \mathbf{m_0}$, $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, $\mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0}$ (token conservation law)

2) From t-semiflows: $\mathbf{x} \in \mathbb{N}^m$, $\mathbf{C} \cdot \mathbf{x} = 0 \Longrightarrow \exists \mathbf{m_0}$, $\exists \sigma \in \mathrm{L}(\mathcal{N}, \mathbf{m_0})$ such that $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m_0}$ and $\boldsymbol{\sigma} = \mathbf{x}$ (cyclic behaviour law)

Classical reasoning to prove logical properties uses these *linear invariants* on the behaviour of a net system [25, 27]. The key idea is similar to that presented for the analysis of properties from the net state equation: Let $\mathcal{S}$ be a net system and $\mathbf{\Psi}$ a matrix where each row is a p-semiflow: $\mathbf{\Psi}[i] \cdot \mathbf{C} = 0$. If $\mathbf{m}$ is reachable from $\mathbf{m_0}$, then $\mathbf{\Psi} \cdot \mathbf{m} = \mathbf{\Psi} \cdot \mathbf{m_0}$. Therefore the set of natural solutions, $\mathbf{m}$, of this equation defines a linearisation of the reachability set $\mathrm{RS}(\mathcal{S})$ denoted $\mathrm{LRS}^{\Psi}(\mathcal{S})$. This set can be used to analyze properties in a similar way to the method based on the state equation. Moreover, as in the case of the net state equation, it usually leads to only semidecision algorithms because, in general, $\mathrm{RS}(\mathcal{S}) \subset \mathrm{LRS}^{SE}(\mathcal{S}) \subset \mathrm{LRS}^{\Psi}(\mathcal{S})$.

Example 6.9 (Analysis based on linear invariants) The marking linear invariants induced by the minimal p-semiflows of the net system in Figure 6.8

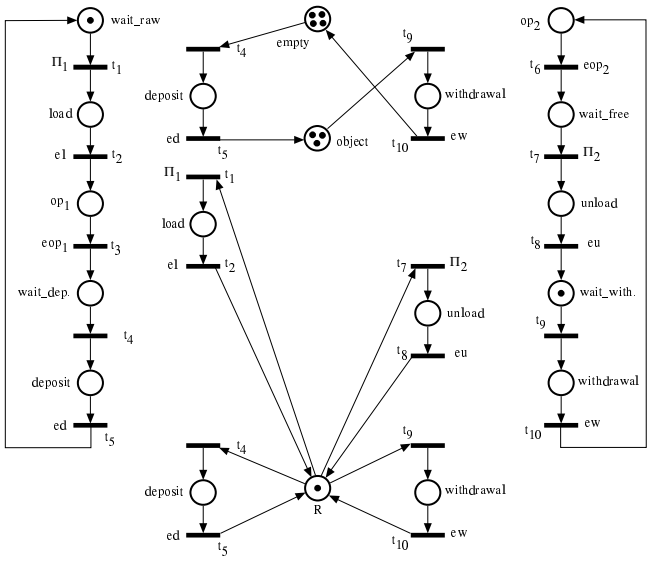


Figure 6.12: A decomposed view of the net system in Figure 6.8.

are the following:

$$\mathbf{m}[\text{wait\_raw}]+\mathbf{m}[\text{load}]+\mathbf{m}[\text{op}_1]+\mathbf{m}[\text{wait\_dep.}]+\mathbf{m}[\text{deposit}]=1 \quad (6.2)$$

$$\mathbf{m}[\text{op}_2]+\mathbf{m}[\text{wait\_free}]+\mathbf{m}[\text{unload}]+\mathbf{m}[\text{wait\_with.}]+\mathbf{m}[\text{withdrawal}]=1 \quad (6.3)$$

$$\mathbf{m}[\text{empty}]+\mathbf{m}[\text{deposit}]+\mathbf{m}[\text{object}]+\mathbf{m}[\text{withdrawal}]=7 \quad (6.4)$$

$$\mathbf{m}[\text{R}]+\mathbf{m}[\text{load}]+\mathbf{m}[\text{unload}]+\mathbf{m}[\text{deposit}]+\mathbf{m}[\text{withdrawal}]=1 \quad (6.5)$$

Because markings are non negative integers (i.e. $\forall p \in P$, $\mathbf{m}[p] \geq 0$), the following can be easily stated from the previous equalities:

1. Bounds: $\forall p_i \in P \backslash \{\text{empty}, \text{object}\}$, $\mathbf{m}[p_i] \leq 1$; $\mathbf{m}[\text{empty}] \leq 7$; and $\mathbf{m}[\text{object}] \leq 7$.

2. The places in each one of the following sets are in marking mutual exclusion:

   a) {wait_raw, load, op_1, wait_dep., deposit}

   b) {op_2, wait_free, unload, wait_with., withdrawal}

   c) {R, load, unload, deposit, withdrawal}

Finally, from a conceptual point of view, the consideration of semiflows provides *decomposed views* of the structure of the net model. In Figure 6.12 the decomposition induced by the minimal p-semiflows of the system in Figure 6.8

is graphically presented. The decomposed view of a net system is even useful to derive an *implementation*. For example, the net system in Figure 6.8 can be implemented using two sequential processes (for *Machine1* and *Machine2*) and three semaphores (*object*, *empty* and R), where R is a mutual exclusion semaphore.

Remark Other structural objects generalizing P- or T- semiflows have been defined [27] leading to other kind of linear invariants. A first type to consider are vectors $\mathbf{y} \in \mathbb{N}^n$ such that $\mathbf{y} \cdot \mathbf{C} \lneq 0$. A vector, $\mathbf{y}$, of this kind leads to the following marking law: $\forall \mathbf{m_0}$, $\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$, $\mathbf{y} \cdot \mathbf{m} \leq \mathbf{y} \cdot \mathbf{m_0}$. A second type are vectors $\mathbf{x} \in \mathbb{N}^m$ such that $\mathbf{C} \cdot \mathbf{x} \gneq 0$. In this case, a vector $\mathbf{x}$ of this kind leads to: $\exists \mathbf{m_0}$, $\exists \sigma \in \mathrm{L}(\mathcal{N}, \mathbf{m_0})$ such that $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m} \geq \mathbf{m_0}$ and $\boldsymbol{\sigma} = \mathbf{x}$. This linear invariants (expressed as inequalities) can be used for analysis purposes in the same way that presented previously for linear invariants obtained from semiflows.

6.5.2 Bounds and boundedness

The study of the bound of a place p, $\mathbf{b}(p)$, through linear algebraic techniques, requires the linearisation of the reachability set in the definition of $\mathbf{b}(p)$ by means of the state equation of the net. In this subsection we assume that $\mathbf{m} \in \mathbb{R}^n$ and $\boldsymbol{\sigma} \in \mathbb{R}^m$. This linearisation of the definition of $\mathbf{b}(p)$ leads to a new quantity called the *structural bound* of p, $\mathbf{sb}(p)$:

$$\mathbf{sb}(p) = \sup\{\mathbf{m}(p) | \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \geq 0, \boldsymbol{\sigma} \geq 0\} \tag{6.6}$$

Let $\mathbf{e_p}$ be the *characteristic vector* of p: $\mathbf{e_p}[q] :=$ if $q = p$ then 1 else 0. The structural bound of p, $\mathbf{sb}(p)$, can be obtained as the optimal solution of the following Linear Programming Problem (LPP):

$$\begin{array}{rlll} \mathbf{sb}(p) = & \max. & \mathbf{e_p} \cdot \mathbf{m} & (6.7) \\ & \text{s.t.} & \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \geq 0 & \\ & & \boldsymbol{\sigma} \geq 0 & \end{array}$$

Therefore $\mathbf{sb}(p)$ can be computed in polynomial time. In sparse-matrix problems (matrix $\mathbf{C}$ is usually sparse), good implementations of the classical *simplex method* leads to quasi-linear time complexities.

Because $\mathrm{RS}(\mathcal{S}) \subset \mathrm{LRS}^{SE}(\mathcal{S})$, in general, we have that $\mathbf{sb}(p) \geq \mathbf{b}(p)$ (recall example 6.7). Therefore, if we are investigating the k-boundedness of a place (i.e. $\mathbf{m}[p] \leq k$), we have a sufficient condition in polynomial time: if $\mathbf{sb}(p) \leq k$ then $\mathbf{b}(p) \leq k$ (i.e. p is k-bounded).

In the sequel we argue on classical results from linear programming and convex geometry theories. We assume the reader is aware of these theories (see, for example, [28, 29]); otherwise all the needed arguments are compiled and adapted in [37]. The important point here is to convey the idea that other theories are helpful to understand in a deep and general framework many sparse

results on net systems' behaviours. The dual linear programming problem of 6.7 is the following (see any text on linear programming to check it):

$$\mathbf{sb}(p)' = \begin{array}{ll} \text{min.} & \mathbf{y} \cdot \mathbf{m_0} \\ \text{s.t.} & \mathbf{y} \cdot \mathbf{C} \leq 0 \\ & \mathbf{y} \geq \mathbf{e_p} \end{array} \tag{6.8}$$

The LPP in 6.7 has always a feasible solution ($\mathbf{m} = \mathbf{m_0}$, $\boldsymbol{\sigma} = 0$). Using duality and boundedness theorems from linear programming theory, both LPPs presented in 6.7 and 6.8 are bounded (thus p is structurally bounded) and $\mathbf{sb}(p) = \mathbf{sb}(p)'$ iff there exists a *feasible solution* for the LPP 6.8: $\mathbf{y} \geq \mathbf{e_p}$ such that $\mathbf{y} \cdot \mathbf{C} \leq 0$.

The reader can easily check that the LPP in 6.8 makes in polynomial time an "implicit search" for the structural bound of p on a set of structural objects including all the p-semiflows. In this sense, we can say that analysis methods based on the state equation are more general than those based on linear invariants. That is, the dual LPPs of those based on the state equation consider not only the p-semiflows but other structural objects as $\mathbf{y} \geq 0$ such that $\mathbf{y} \cdot \mathbf{C} \not\leq 0$. On the other hand, we must say that the computational effort using the linear invariants is greater than using the state equation, since the computation of the minimal p-semiflows (in some cases, an exponential number!) must be done previously to the study of the property.

From the above discussion and using the *alternatives theorem* (an algebraic form of the *Minkowski-Farkas lemma*) the following properties can be proved:

Property 6.5 *The following three statements are equivalent:*

1. *p is structurally bounded, i.e. p is bounded for any $\mathbf{m_0}$.*
2. *There exists $\mathbf{y} \geq \mathbf{e_p}$ such that $\mathbf{y} \cdot \mathbf{C} \leq 0$. (place-based characterization)*
3. *For all $\mathbf{x} \geq 0$ such that $\mathbf{C} \cdot \mathbf{x} \geq 0$, $\mathbf{C}[p, T] \cdot \mathbf{x} = 0$. (transition-based characterization)*

Property 6.6 *The following three statements are equivalent:*

1. *$\mathcal{N}$ is structurally bounded, i.e. $\mathcal{N}$ is bounded for any $\mathbf{m_0}$.*
2. *There exists $\mathbf{y} \geq \mathbf{1}$ such that $\mathbf{y} \cdot \mathbf{C} \leq 0$. (place-based characterization)*
3. *For all $\mathbf{x} \geq 0$ such that $\mathbf{C} \cdot \mathbf{x} \geq 0$, $\mathbf{C} \cdot \mathbf{x} = 0$; i.e. $\nexists\, \mathbf{x} \geq 0$ s.t. $\mathbf{C} \cdot \mathbf{x} \gneq 0$. (transition-based characterization)*

6.5.3 Deadlock-freeness (and liveness)

Deadlock-freeness concerns the existence of some activity from any reachable state of the system. It is a necessary condition for liveness, although in general not sufficient. When no part of the system can evolve, it is said that the system has reached a state of total deadlock (or *deadlock* for short). In net system

terms, a deadlock corresponds to a marking from which no transition is fireable. In order to study deadlock-freenes by means of linear algebraic techniques, the property must be expressed as a formula of a first order logic having linear inequalities as atoms, where the reachability or fireability conditions are relaxed by satisfiability of the state equation. The formula to express that a marking is a deadlock consists of a condition for every transition expressing that it is disabled at such marking. This condition consists of several inequalities, one per input place of the transition (expressing that the marking of such place is less than the corresponding weight) linked by the "$\vee$" connective (because lack of tokens in a single input place disables the transition). We give below a basic general sufficient condition for deadlock-freeness based on the absence of solutions satisfying simultaneously the net state equation and the formula expressing the total deadlock condition commented above.

Proposition 6.7 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system. If there doesn't exist any solution $(\mathbf{m}, \boldsymbol{\sigma})$, for the system*

$$\begin{array}{l} \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \\ \mathbf{m} \geq 0, \boldsymbol{\sigma} \geq 0 \\ \bigvee_{p \in {}^\bullet t} \mathbf{m}[p] < \mathbf{Pre}[p,t]; \forall t \in T \end{array} \tag{6.9}$$

then $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is deadlock-free.

Obviously, the deadlock conditions are non linear, because they are expressed using the "$\vee$" connective. Anyway we can express the above condition by means of a set of linear systems as follows. Let $\alpha : T \rightarrow P$ be a mapping that assigns to each transition one of its input places. If there doesn't exist α such that the system

$$\begin{array}{l} \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \\ \mathbf{m} \geq 0, \boldsymbol{\sigma} \geq 0 \\ \mathbf{m}[\alpha(t)] < \mathbf{Pre}[\alpha(t), t]; \forall t \in T \end{array} \tag{6.10}$$

has a solution, then $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is deadlock-free. The problem is that we have to check it for *every* mapping α of input places to transitions so we have to check $\prod_{t \in T} |{}^\bullet t|$ systems of linear inequalities. If every transition has exactly one input place (e.g. State Machines) then only one system needs to be checked, but in general the number might be large. Nevertheless it is possible to reduce the number of systems to be checked, preserving the set of *integer solutions*. For this purpose, in [39] four simplification rules of the deadlock condition are presented using information obtained from the net system, and a simple net transformation obtaining an equivalent one wrt. the deadlock-freeness property where the enabling conditions of transitions can be expressed linearly. As a result, deadlock-freeness of a wide variety of net systems can be proven by verifying absence of solutions to a *single system of linear inequalities*. Even more, in some subclasses it is known that there are no spurious solutions being deadlocks, so the method decides on deadlock-freeness [40]. The following example

presents the deadlock-freenes analysis of the net system in figure 6.8 applying this technique.

Example 6.10 (Deadlock-freeness analysis and simplification rules) Let us consider the net system in Figure 6.8. The direct application of the method described in proposition 6.7 requires to check $\prod_{t\in T} |{}^\bullet t| = 36$ linear systems as that presented in 6.10. Nevertheless, below we show that we can reduce the deadlock-freeness analysis on this net to check a unique linear system applying the simplification rules presented in [39]. Solving the LPP 6.7 for the places of the net system we obtain the following: $\mathbf{sb}(\mathrm{p}) = 1$, for all $p \in P \setminus \{\text{empty}, \text{object}\}$; and $\mathbf{sb}(\text{empty}) = \mathbf{sb}(\text{object}) = 7$ (the same can be obtained from the linear invariants in Eqs 6.2-6.5). The transitions t_1, t_4, t_7 and t_9 are those presenting complex conditions giving rise to the large number of linear systems. The simplification of these consitions is as follows:

a) The non-fireability condition of t_1 is $(\mathbf{m}[\text{wait\_raw}] = 0) \vee (\mathbf{m}[\mathrm{R}] = 0)$. Taking into account that $\mathbf{sb}(\text{wait\_raw}) = \mathbf{sb}(\mathrm{R}) = 1$, we can apply a particularization of rule 3 in [39] to replace the previous complex condition by a unique linear inequality: Let t be a transition such that each input place verifies that its structural bound is equal to the weight of its output arc joining it to t. The non fireability condition for transition t at a marking $\mathbf{m}$ is $\sum_{p\in{}^\bullet t} \mathbf{m}[p] \leq \sum_{p\in P} \mathbf{Pre}[p,t] - 1$. That is, the amount of tokens in the input places of t is less than the needed. Therefore, for the transition t_1 this linear condition is: $\mathbf{m}[\text{wait\_raw}] + \mathbf{m}[\mathrm{R}] \leq 1$.

b) The non-fireability condition of t_7 is $(\mathbf{m}[\text{wait\_free}] = 0) \vee (\mathbf{m}[\mathrm{R}] = 0)$. In a similar way to the case of transition t_1 we replace this condition by $\mathbf{m}[\text{wait\_free}] + \mathbf{m}[\mathrm{R}] \leq 1$, since $\mathbf{sb}(\text{wait\_free}) = \mathbf{sb}(\mathrm{R}) = 1$ and rule 3 in [39] can be applied.

c) The non-fireability condition of t_4 is $(\mathbf{m}[\text{wait\_dep.}] = 0) \vee (\mathbf{m}[\mathrm{R}] = 0) \vee (\mathbf{m}[\text{empty}] = 0)$. Since $\mathbf{sb}(\text{wait\_dep.}) = \mathbf{sb}(\mathrm{R}) = 1$ and $\mathbf{sb}(\text{empty}) = 7$ (i.e. only one input place of t_7 has a $\mathbf{sb}$ greater than the weight of the arc) rule 4 of [39] can be applied. Then, the previous complex condition is replaced by the following linear condition:

$$\mathbf{sb}(\text{empty}) \cdot (\mathbf{m}[\text{wait\_dep.}] + \mathbf{m}[\mathrm{R}]) + \mathbf{m}[\text{empty}] \leq \mathbf{sb}(\text{empty}) \cdot (\mathbf{Pre}[\text{wait\_dep.}, T] + \mathbf{Pre}[\mathrm{R}, T]) + \mathbf{Pre}[\text{empty}, T] - 1$$

i.e. $7(\mathbf{m}[\text{wait\_dep.}] + \mathbf{m}[\mathrm{R}]) + \mathbf{m}[\text{empty}] \leq 14$.

d) The non-fireability condition of t_9 can be reduced to the following linear condition by similar reasons to the case of transition t_4: $7(\mathbf{m}[\text{wait\_with.}] + \mathbf{m}[\mathrm{R}]) + \mathbf{m}[\text{object}] \leq 14$.

Applying the previously stated simplifications, the deadlock-freeness analysis for the net system in Figure 6.8 is reduced to verify that there doesn't exist any

solution $(\mathbf{m}, \boldsymbol{\sigma})$, for the following single linear system (the reader can check that the system has no solutions).

$$
\begin{array}{ll}
\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} & (6.11) \\
\mathbf{m} \geq 0, \boldsymbol{\sigma} \geq 0 & \\
\mathbf{m}[\text{wait\_raw}] + \mathbf{m}[\text{R}] \leq 1; & \text{for } t_1 \\
\mathbf{m}[\text{load}] = 0; & \text{for } t_2 \\
\mathbf{m}[\text{op}_1] = 0; & \text{for } t_3 \\
7(\mathbf{m}[\text{wait\_dep.}] + \mathbf{m}[\text{R}]) + \mathbf{m}[\text{empty}] \leq 14; & \text{for } t_4 \\
\mathbf{m}[\text{deposit}] = 0; & \text{for } t_5 \\
\mathbf{m}[\text{op}_2] = 0; & \text{for } t_6 \\
\mathbf{m}[\text{wait\_free}] + \mathbf{m}[\text{R}] \leq 1; & \text{for } t_7 \\
\mathbf{m}[\text{unload}] = 0; & \text{for } t_8 \\
7(\mathbf{m}[\text{wait\_with.}] + \mathbf{m}[\text{R}]) + \mathbf{m}[\text{object}] \leq 14; & \text{for } t_9 \\
\mathbf{m}[\text{withdrawal}] = 0; & \text{for } t_{10}
\end{array}
$$

Linear invariants may also be used to prove *deadlock-freeness*. Using the linear invariants in Eqs. (6.2-6.5), we shall prove that our net system in Figure 6.8 is deadlock-free.

If there exists a deadlock, no transition can be fired. Let us try to construct a marking in which no transition is fireable. When a unique input place of a transition exists, that place must be unmarked. So $\mathbf{m}[\text{load}] = \mathbf{m}[\text{op}_1] = \mathbf{m}[\text{deposit}] = \mathbf{m}[\text{op}_2] = \mathbf{m}[\text{unload}] = \mathbf{m}[\text{withdrawal}] = 0$, and the linear invariants in Eqs (6.2-6.5) reduce to:

$$\mathbf{m}[\text{wait\_raw}] + \mathbf{m}[\text{wait\_dep.}] = 1 \tag{6.12}$$

$$\mathbf{m}[\text{wait\_free}] + \mathbf{m}[\text{wait\_with.}] = 1 \tag{6.13}$$

$$\mathbf{m}[\text{empty}] + \mathbf{m}[\text{object}] = 7 \tag{6.14}$$

$$\mathbf{m}[\text{R}] = 1 \tag{6.15}$$

Since R should always be marked at the present stage, to prevent the firing of t_1 and t_7, places wait_raw and wait_free should be unmarked. The linear invariants are reduced once more, leading to:

$$\mathbf{m}[\text{wait\_dep.}] = 1 \tag{6.16}$$

$$\mathbf{m}[\text{wait\_with.}] = 1 \tag{6.17}$$

$$\mathbf{m}[\text{empty}] + \mathbf{m}[\text{object}] = 7 \tag{6.18}$$

$$\mathbf{m}[\text{R}] = 1 \tag{6.19}$$

Since $\mathbf{m}[\text{wait\_dep.}] = \mathbf{m}[\text{wait\_with.}] = 1$, to avoid the firing of t_4 and t_9, $\mathbf{m}[\text{empty}] + \mathbf{m}[\text{object}] = 0$ is needed. This contradicts Eq (6.18), so the net system is deadlock-free. A more compact, algorithmic presentation of the above deadlock-freeness proof is:

if $\mathbf{m}[\text{load}] + \mathbf{m}[\text{op}_1] + \mathbf{m}[\text{deposit}] + \mathbf{m}[\text{op}_2] + \mathbf{m}[\text{unload}] + \mathbf{m}[\text{withdrawal}] \geq 1$
then one of t_2,t_3,t_5,t_6,t_8 or t_{10} is fireable
else if $\mathbf{m}[\text{wait\_raw}] + \mathbf{m}[\text{wait\_free}] \geq 1$
then one of t_1 or t_7 is fireable
else one of t_4 or t_9 is fireable

As a final remark, we want to point out that liveness can be proved for the net system in Figure 6.8. Liveness implies deadlock-freeness, but the reverse is not true in general. Nevertheless, if the net is consistent and it has only one minimal t-semiflow, as it happens in the example, where the unique minimal t-semiflow is $\mathbf{1}$; then any infinite behaviour must contain all transitions with relative firings given by such t-semiflow. Thus deadlock-freeness implies, in this case, liveness.

6.5.4 Structural liveness and liveness

A necessary condition for a transition t to be live in a system $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is its eventual infinite fireability, i.e. the existence of a firing repetitive sequence σ_R containing t: $\exists \sigma_R \in \mathrm{L}(\mathcal{N}, \mathbf{m_0})$ such that $\mathbf{m_0} \xrightarrow{\sigma_R} \mathbf{m} \geq \mathbf{m_0}$ and $\boldsymbol{\sigma}_R[t] > 0$.

Using the state equation as a linearisation of the reachability set, an *upper bound* of the number of times t can be fired in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is given by the following LPP ($\mathbf{e_t}[u] := $ if $u = t$ then 1 else 0):

$$\begin{array}{rlll} \mathbf{sr}(t) = & \max. & \mathbf{e_t} \cdot \boldsymbol{\sigma} & (6.20) \\ & \text{s.t.} & \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \geq 0 & \\ & & \boldsymbol{\sigma} \geq 0 & \end{array}$$

The dual of (LPP 6.20) is:

$$\begin{array}{rlll} \mathbf{sr}(t)' = & \min. & \mathbf{y} \cdot \mathbf{m_0} & (6.21) \\ & \text{s.t.} & \mathbf{y} \cdot \mathbf{C} \leq -\mathbf{e_t} & \\ & & \mathbf{y} \geq 0 & \end{array}$$

We are interested on characterizing when $\mathbf{sr}(t)$ goes to infinity. The LPP 6.20 has $\mathbf{m} = \mathbf{m_0}$ and $\boldsymbol{\sigma} = 0$ as a feasible solution. Using first duality and unboundedness theorems from linear programming and later the alternatives theorem, the following properties can be stated:

Property 6.8 *The following three statements are equivalent:*

1. *t is structurally repetitive (i.e. there exists a "large enough" $\mathbf{m_0}$ such that t can be fired infinitely often).*
2. *There does not exist $\mathbf{y} \geq 0$ such that $\mathbf{y} \cdot \mathbf{C} \leq -\mathbf{e_t}$ (place-based perspective)*
3. *There exists $\mathbf{x} \geq \mathbf{e_t}$ such that $\mathbf{C} \cdot \mathbf{x} \geq 0$ { transition-based perspective }*

Property 6.9 *The following three statements are equivalent:*

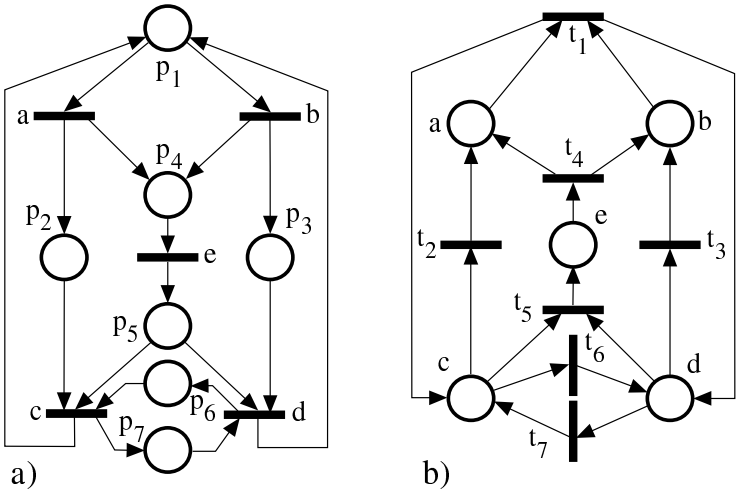


Figure 6.13: Two conservative and consistent, struturally non-live nets: (a) rank($\mathbf{C}$) = 4, |EQS| = 3, thus $\mathcal{N}$ is not structurally live; (b) rank($\mathbf{C}$) = 4, |EQS| = 4, |CCS| = 3, thus no answer.

1. *$\mathcal{N}$ is structurally repetitive (i.e. all transitions are structurally repetitive).*
2. *There does not exist* $\mathbf{y} \geq 0$ *such that* $\mathbf{y} \cdot \mathbf{C} \leqneq 0$
3. *There exists* $\mathbf{x} \geq \mathbf{1}$ *such that* $\mathbf{C} \cdot \mathbf{x} \geq 0$

Aditionally, the following classical results can be stated [27, 7, 35]:

Property 6.10 *Let $\mathcal{N}$ be a net and $\mathbf{C}$ its incidence matrix.*

1. **if** *$\mathcal{N}$ is structurally live* **then** *$\mathcal{N}$ is structurally repetitive.*
2. **if** *$\mathcal{N}$ is structurally live and structurally bounded* **then** *$\mathcal{N}$ is conservative* ($\exists \mathbf{y} \geq \mathbf{1}$ *such that* $\mathbf{y} \cdot \mathbf{C} = 0$) *and consistent* ($\exists \mathbf{x} \geq \mathbf{1}$ *such that* $\mathbf{C} \cdot \mathbf{x} = 0$).
3. **if** *$\mathcal{N}$ is connected, consistent and conservative* **then** *it is strongly connected.*
4. **if** *$\mathcal{N}$ is live and bounded* **then** *$\mathcal{N}$ is strongly connected and consistent.*

Net structures in Figure 6.13 are consistent and conservative, but there does not exist a live marking for them. A more careful analysis allows to improve the above result with a *rank condition* on the incidence matrix of $\mathcal{N}$, $\mathbf{C}$. This and other results are summarized in the next property. Recall, from section I.2.2.4, that SEQS and SCCS denote the sets of Equal Conflict Sets and Coupled Conflict Sets, respectively.

Property 6.11 *Let $\mathcal{N}$ be a net and $\mathbf{C}$ its incidence matrix.*

1. **if** $\mathcal{N}$ *is live and bounded* **then** $\mathcal{N}$ *is strongly connected, consistent, and* $\text{rank}(\mathbf{C}) \leq |\text{SEQS}| - 1$.

2. **if** $\mathcal{N}$ *is conservative, consistent, and* $\text{rank}(\mathbf{C}) = |\text{SCCS}| - 1$ **then** $\mathcal{N}$ *is structurally live and structurally bounded.*

The condition in property 6.11.1 has been proven to be also sufficient for some subclasses of nets [12, 40]. Observe that, even for structurally bounded ordinary nets, we do not have a complete characterization of structural liveness. Since $|\text{SCCS}| \leq |\text{SEQS}|$, there is still a range of nets which satisfy neither the necessary nor the sufficient condition to be structurally live and structurally bounded! The added rank condition allows to state that the net in Figure 6.13.a is structurally non-live. Nevertheless, nothing can be said about structural liveness of the net in Figure 6.13.b.

Property 6.11 is purely structural (i.e., the initial marking is not considered at all). Nevertheless, it is clear that a too small initial marking (e.g. the empty marking) make non live any net structure. A less trivial lower bound for the initial marking based on marking linear invariants is based on fireability of every transition. If $t \in T$ is fireable at least once, for any p-semiflow $\mathbf{y}$, $\mathbf{y} \cdot \mathbf{m_0} \geq \mathbf{y} \cdot \mathbf{Pre}[P, t]$. Therefore:

Property 6.12 *If* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *is a live system, then* $\forall \mathbf{y} \geq 0$ *such that* $\mathbf{y} \cdot \mathbf{C} = 0$, $\mathbf{y} \cdot \mathbf{m_0} \geq \max_{t \in T}(\mathbf{y} \cdot \mathbf{Pre}[P, t]) \geq 1$

Unfortunately no characterization of liveness exists in linear algebraic terms for general nets. The net system in Figure 6.1.b adding a token to p_5 is consistent, conservative, fulfills the rank condition and all p-semiflows are marked, but it is non live.

6.5.5 Reversibility (and liveness)

Let us use now a *Liapunov-stability-like* technique to prove that the net system in Figure 6.8 is reversible. It serves to illustrate the use of marking linear invariants and some inductive reasonings to analyze liveness properties.

As a preliminary consideration that makes easier the rest of the proof, the following simple property will be used: Let $\langle \mathcal{N}, \mathbf{m}_1 \rangle$ be a reversible system and $\mathbf{m_0}$ reachable from $\mathbf{m}_1$ (i.e., $\exists \sigma \in \text{L}(\mathcal{N}, \mathbf{m}_1)$ such that $\mathbf{m}_1 \xrightarrow{\sigma} \mathbf{m_0}$). Then $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is reversible.

Assume $\mathbf{m}_1$ is like $\mathbf{m_0}$ (Figure 6.8), but making: $\mathbf{m}_1[\text{wait\_raw}] = \mathbf{m}_1[\text{empty}] = 0$, $\mathbf{m}_1[\text{wait\_dep.}] = 1$ and $\mathbf{m}_1[\text{object}] = 7$.

Let us prove first that $\langle \mathcal{N}, \mathbf{m}_1 \rangle$ is reversible. Let $\mathbf{w}$ be a non-negative place weighting such that $\mathbf{w}[p_i] = 0$ iff p_i is marked in $\mathbf{m}_1$. Therefore, $\mathbf{w}[\text{wait\_dep.}] = \mathbf{w}[\text{R}] = \mathbf{w}[\text{object}] = \mathbf{w}[\text{wait\_with.}] = 0$ and $\mathbf{w}[p_j] > 0$ for all the other places. The function $\mathbf{v}(\mathbf{m}) = \mathbf{w} \cdot \mathbf{m}$ has the following properties: $\mathbf{v}(\mathbf{m}) \geq 0$ and $\mathbf{v}(\mathbf{m}_1) = 0$

For the system in Figure 6.8 a stronger property holds: $\mathbf{v}(\mathbf{m}) = 0 \iff \mathbf{m} = \mathbf{m}_1$. This can be clearly seen because $\mathbf{w} \cdot \mathbf{m} = 0 \iff \mathbf{m}[\text{wait\_raw}] = \mathbf{m}[\text{load}] =$

$\mathbf{m}[\text{op}_1] = \mathbf{m}[\text{deposit}] = \mathbf{m}[\text{empty}] = \mathbf{m}[\text{op}_2] = \mathbf{m}[\text{wait\_free}] = \mathbf{m}[\text{unload}] = \mathbf{m}[\text{withdrawal}] = 0$. Even more, it is easy to check the following: $\mathbf{m}_1$ is the present marking $\Longleftrightarrow t_9$ is the unique fireable transition.

If there exists (warning: in Liapunov-stability criteria the universal quantifier is used!) a finite firing sequence (i.e., a finite trajectory) per reachable marking $\mathbf{m}_i$ such that $\mathbf{m}_i \xrightarrow{\sigma_k} \mathbf{m}_{i+1}$ and $\mathbf{v}(\mathbf{m}_i) > \mathbf{v}(\mathbf{m}_{i+1})$, in a finite number of transition firings $\mathbf{v}(\mathbf{m}) = 0$ is reached. Because $\mathbf{v}(\mathbf{m}) = 0 \Longleftrightarrow \mathbf{m} = \mathbf{m}_1$, a proof that $\mathbf{m}_1$ is reachable from any marking has been obtained (i.e, $\langle \mathcal{N}, \mathbf{m}_1 \rangle$ is reversible).

Premultiplying the net state equation by $\mathbf{w}$ we obtain the following condition:**if** $\sigma_k = t_j$ **then** $[\mathbf{w} \cdot \mathbf{m}_{i+1} < \mathbf{w} \cdot \mathbf{m}_i] \Longleftrightarrow \mathbf{w} \cdot \mathbf{C}[P, t_j] < 0$

Now, removing in Figure 6.8 the places marked at $\mathbf{m}_1$ (i.e., wait_dep., R, object, wait_with.) and fireable transitions (i.e., t_9) an acyclic net is obtained, so there exists an $\mathbf{w}$ such that $\mathbf{w} \cdot \mathbf{C}[P, t_j] < 0$, $\forall j \neq 9$.

For example, taking as weights the levels in the acyclic graph we have:

$$\mathbf{w}[\text{op}_1] = \mathbf{w}[\text{unload}] = 1 \tag{6.22}$$

$$\mathbf{w}[\text{load}] = \mathbf{w}[\text{wait\_free}] = 2 \tag{6.23}$$

$$\mathbf{w}[\text{wait\_raw}] = \mathbf{w}[\text{op}_2] = 3 \tag{6.24}$$

$$\mathbf{w}[\text{deposit}] = \mathbf{w}[\text{withdrawal}] = 4 \tag{6.25}$$

$$\mathbf{w}[\text{empty}] = 5 \tag{6.26}$$

and $\mathbf{w} \cdot \mathbf{C} = [-1, -1, -1, -1, -1, -1, -1, -1, +4, -1]$. In other words, the firing of any transition, except t_9, decreases $\mathbf{v}(\mathbf{m}) = \mathbf{w} \cdot \mathbf{m}$.

Using the algorithmic deadlock-freedom explanation in previous sections, the reversibility of $\langle \mathcal{N}, \mathbf{m}_1 \rangle$ is proven (observe that the p-invariants in Eqs (6.2-6.3-6.4-6.5) remain for $\mathbf{m}_1$):

if $\mathbf{m}[\text{load}] + \mathbf{m}[\text{op}_1] + \mathbf{m}[\text{deposit}] + \mathbf{m}[\text{op}_2] + \mathbf{m}[\text{unload}] + \mathbf{m}[\text{withdrawal}] \geq 1$
then $\mathbf{v}(\mathbf{m})$ can decrease firing t_2,t_3,t_5,t_6,t_8 or t_{10}
else if $\mathbf{m}[\text{wait\_raw}] + \mathbf{m}[\text{wait\_free}] \geq 1$
then $\mathbf{v}(\mathbf{m})$ can decrease firing $t_1$or t_7
else $\mathbf{v}(\mathbf{m})$ can decrease firing t_4 or t_9 is the unique fireable transition
(iff $\mathbf{m}_1$ is the present marking)

Because $\mathbf{m_0}$ is reachable from $\mathbf{m}_1$ (e.g. firing $\sigma = (t_9 t_{10} t_6 t_7 t_8)^5 t_4 t_5$), $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is a reversible system.

Once again liveness of the system in Figure 6.8 can be proved, because the complete sequence (i.e. containing all transitions) $\sigma = t_1 t_2 t_3 t_4 t_5 t_9 t_{10} t_6 t_7 t_8$ can be fired. Since the system is reversible, no transition loses the possibility of firing (i.e., all transitions are live).

6.6 Siphons and traps

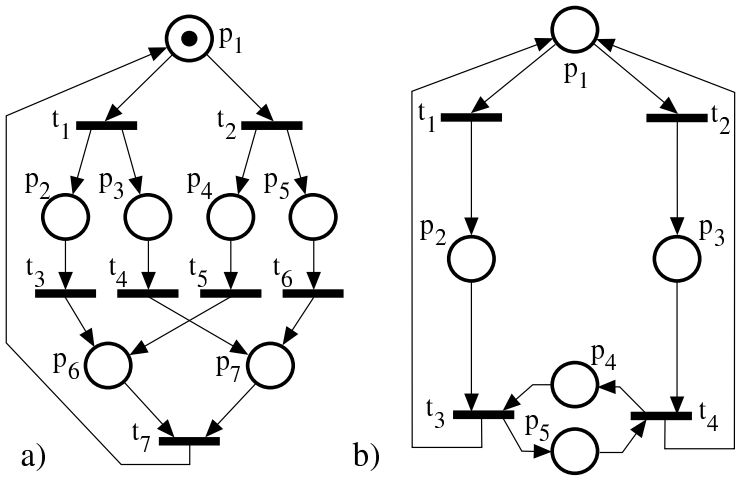


Figure 6.14: Two consistent and conservative free choice nets: (a) Structurally live rank($\mathbf{C}$) = 5, |EQS| = 5; (b) Structurally non-live rank($\mathbf{C}$) = 3, |EQS| = 2.

By means of graph theory based reasoning it is possible to characterize many properties of net subclasses. *Siphons* (also called *structural deadlocks*, or more simply - but ambiguously - *deadlocks*) and *traps* are easily recognizable subsets of places that generate very particular subnets.

Definition 6.13 *Let $\mathcal{N} = \langle P, T, F \rangle$ be an ordinary net.*

1. *A* siphon *is a subset of places, Σ, such that the set of its input transitions is contained in the set of its output transitions: $\Sigma \subseteq P$ is a siphon $\Longleftrightarrow$ ${}^\bullet\Sigma \subseteq \Sigma^\bullet$.*

2. *A* trap *is a subset of places, θ, such that the set of its output transitions is contained in the set of its input transitions: $\theta \subseteq P$ is a trap $\Longleftrightarrow \theta^\bullet \subseteq {}^\bullet\theta$.*

$\Sigma = \{p_1, p_2, p_4, p_5, p_6\}$ is a siphon for the net in Figure 6.14.a: ${}^\bullet\Sigma = \{t_7, t_1, t_2, t_3, t_5\}$, while $\Sigma^\bullet = {}^\bullet\Sigma \cup \{t_6\}$. Σ contains a trap, $\theta = \Sigma \setminus \{p_5\}$. In fact θ is also a siphon (it is minimal: removing any number of places no siphon can be obtained).

Siphons and traps are reverse concepts: A subset of places of a net $\mathcal{N}$ is a siphon iff it is a trap on the reverse net, $\mathcal{N}^{-1}$ (i.e. that obtained reversing the arcs, its flow relation, F).

The following property "explains" why structural deadlocks or siphons (think on "soda siphons") and traps are the names of the above concepts.

Property 6.14 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be an ordinary net system.*

1. *If $\mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0})$ is a deadlock state, then $\Sigma = \{p | \mathbf{m}[p] = 0\}$ is an unmarked (empty) siphon.*

2. *If a siphon is (or becomes) unmarked, it will remain unmarked for any possible net system evolution. Therefore all its input and output transitions are dead. So the system is not-live (but can be deadlock-free).*

3. *If a trap is (or becomes) marked, it will remain marked for any possible net system evolution (i.e. at least one token is "trapped").*

If a trap is not marked at $\mathbf{m_0}$, and the system is live, $\mathbf{m_0}$ will not be recoverable from those markings in which the trap is marked. Thus:

Corollary 6.15 *If a live net system is reversible, then* $\mathbf{m_0}$ *marks all traps.*

Remark For live and bounded free choice systems a stronger property holds: Marking all traps is a necessary and sufficient condition for reversibility [5]. The net system in Figure 6.14.a is reversible. Nevertheless, if $\mathbf{m_0} = [0, 1, 0, 0, 1, 0, 0]$, the new system is live and bounded but non reversible: The trap $\theta = \{p_1, p_3, p_4, p_6, p_7\}$ is not marked at $\mathbf{m_0}$.

A siphon which contains a marked trap will never become unmarked. So this more elaborate property can be of helpful for some liveness characterizations.

Definition 6.16 *Let* $\mathcal{N}$ *be an ordinary net. The system* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *has the* Marked-Siphon-Trap property, *MST-property, if each siphon contains a marked trap at* $\mathbf{m_0}$.

A siphon (trap) is *minimal* if it does not contain another siphon (trap). Thus, siphons in the above statement can be constrained to be minimal without any loss of generality.

The MST-property guarantees that all siphons will be marked. Thus no dead marking can be reached, according with property 6.14.1. Therefore:

Property 6.17 *If* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *has the MST-property, the system is deadlock-free.*

Figure 6.15 presents some limitations of the MST-property for liveness characterization.

Remark The MST-property is sufficient for liveness in simple net systems and necessary and sufficient for free-choice net systems. As a corollary, the *liveness monotonicity* result is true for the case of live free-choice systems: If $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is a live free-choice system, then for all $\mathbf{m_0}' \geq \mathbf{m_0}$, $\langle \mathcal{N}, \mathbf{m_0}' \rangle$ is also live. The previous result does not apply to Simple Net systems. The system in Figure 6.1.b is simple, $\Sigma = \{p_1, p_2, p_7\}$ is a siphon ($^\bullet\Sigma = \{t_3, t_4, t_1\}$, $\Sigma^\bullet = {}^\bullet\Sigma \cup \{t_2\}$) that does not contain any trap. If we assume $\mathbf{m_0}[p_5] = 1$, t_2 can be fired and Σ becomes empty, leading to non-liveness.

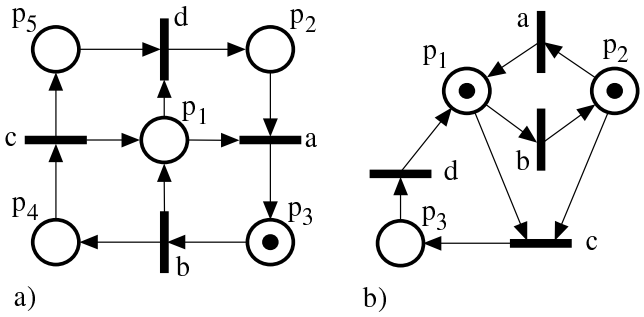


Figure 6.15: For the two nets, the MST-property does not hold, but: (a) The simple net is live and bounded; (b) The non-simple net is non-live (although deadlock-free) and bounded.

6.7 Analysis of net subclasses

In this section we quickly overview some of the analytical results for the subclasses defined in Chapter 2. We organise the material around properties instead of describing the results for each subclass, what would lead to abundant redundancies. (Of course, properties of large subclasses such as EQ systems, are inherited by their subclasses such as FC or DF systems.)

Our intention is to show how the restrictions imposed by subclasses' definitions, at the price of losing some modelling capabilities, facilitate the analysis. The designer must find a compromise between modelling power and availability of powerful analysis tools, while one of the theoretician's goals is obtaining better results for increasingly larger subclasses.

The general idea behind the structure theory of net subclasses is to investigate properties that every net system in the subclass enjoys, instead of analysing each particular system. These general properties are useful in two ways:

- The designer knows that her/his system (if it belongs to an appropriate subclass) behaves "well" (e.g., liveness monotonicity, existence of home states).
- General analysis methods become more applicable or more conclusive (e.g., model checking for FC, liveness analysis for all the subclasses considered).

The technical development of the presented results, and many other details that are out of the scope of this very succint presentation, can be found in [17], [14], [32], [38], [40].

6.7.1 Fairness and monopolies

In some systems, *impartiality* (or *global fairness*, that is, every transition appears infinitely often in infinite sequences) can be achieved *locally* (every solution of a — local — conflict that is effective infinitely often is taken infinitely often):

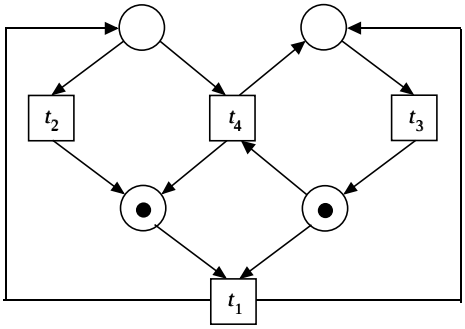


Figure 6.16: A net system where local fairness does not guarantee impartiality, and which can exhibit monopoly situations.

Theorem 6.18 *Let $\mathcal{S}$ be a bounded strongly connected EQ system or DSSP. A sequence $\sigma \in \mathrm{L}(\mathcal{S})$ is globally fair iff it is locally fair.*

This property is not true in general. Take for instance the net system in Figure 6.16. The sequence $\sigma = \{t_1\, t_2\, t_3\}^\omega$ is locally fair (actually, during the occurrence of σ no conflict is effective at all), but it is not globally fair since t_4 never occurs. Conversely, the sequence $\sigma = \{t_1\, t_3\, t_4\, t_3\, t_1\, t_2\, t_3\}^\omega$ is globally fair but not locally since whenever t_2 and t_4 are in conflict t_4 wins.

The equivalence of local and global fairness has two important consequences. The first one is equivalence of liveness and deadlock-freeness, what facilitates the analysis of liveness because it suffices to check the weaker property of deadlock-freeness:

Theorem 6.19 *Let $\mathcal{S}$ be a bounded strongly connected EQ system or DSSP. Then $\mathcal{S}$ is live iff it is deadlock-free.*

The second consequence is relevant for the eventual interpretation of the model. Assume, for instance, that the system in Figure 6.16 is interpreted so that transitions occur after a deterministic delay equal to their index. Then, the system behaves repeating the occurrence of $t_1\, t_2\, t_3$, never giving a chance to t_4, despite it was perfectly live in the autonomous model: the interpretation has destroyed liveness leading to a *monopoly* situation (the "resources" needed by t_4 are "monopolized" by t_2).

This can never happen to a bounded strongly connected EQ system or DSSP, assuming the interpretation allows progress (i.e., a transition that is continuously enabled eventually occurs): by imposing a fair conflict resolution policy, which can be done in a distributed fashion provided structurally conflicting transitions are allocated together, it is guaranteed that no action in the system becomes permanently disabled if the autonomous model was live.

6.7.2 Confluence and directedness

Persistent systems, which include structurally persistent ones (DF) enjoy a strong *confluence* property: whenever from a given marking we reach two different markings by firing two distinct sequences, then we can complete both such sequences, each with the firings left with respect to the other, reaching in any case the same marking [24]. Confluence is closely related to determinacy [22]: interpreting sequences as executions and transition occurrences as operations, when from a given point two different executions may occur, depending on operation times or other external matters, each operation in one execution will eventually occur in the other (assuming progress), possibly in a different order and with a different timing.

Moreover, confluence facilitates checking liveness (non-termination) of persistent systems: it suffices to find a repeatable sequence that contains every transition. This is because such a repeatable sequence allows to construct a sequence greater than any given sequence σ fireable from the initial marking, and this proves that σ can be continued to enable the repeatable sequence.

Stepping out from persistent systems, the presence of effective conflicts may destroy confluence. *Directedness* is a weaker property that states that a common successor of arbitrary reachable markings always exist, and which holds for some subclasses:

Theorem 6.20 *Let $\mathcal{S}$ be a live EQ system or DSSP. Let $\mathbf{m_a}, \mathbf{m_b} \in \mathrm{RS}(\mathcal{S})$. Then $\mathrm{RS}(\mathcal{N}, \mathbf{m_a}) \cap \mathrm{RS}(\mathcal{N}, \mathbf{m_b}) \neq \emptyset$.*

Informally, directedness means that the effect of a particular resolution of a conflict is not "irreversible": there is a point where the evolution joints with that which would have been if the decision had been other. The existence of *home states*, i.e., states that can be ultimately reached after whichever evolution, follows from directedness and boundedness:

Theorem 6.21 *Live and bounded EQ systems or DSSP have home states.*

The system in Figure 6.3.b is an example of a live and 1-bounded system without home states.

The existence of home state is an important property for many reasons:

- The system is known to have states to return to, which is often required in reactive systems. Chosing one such state as the initial one makes the system *reversible*, i.e., $\mathbf{m_0}$ can always be recovered.
- Model checking is largely simplified, since there is only one terminal strongly connected component in the reachability graph.
- Under a Markovian interpretation (e.g., as in *generalized stochastic Petri nets* [2]), *ergodicity* of the marking process is guaranteed; otherwise, simulation or computation of steady state performance indices could be meaningless.

6.7.3 Reachability and the state equation

As it was discussed in Section 6.5, reachable markings are solutions to the state equation but, in general, not conversely: some solutions of the state equation may be "spurious". This limits the use of the state equation as a convenient algebraic representation of the state space.

Fortunately stronger relations between reachable markings and solutions to the state equation are available for some subclasses:

Theorem 6.22 *Let $\mathcal{S}$ be a P/T system with reachability set* RS *and linearised reachability set wrt. the state equation* $\mathrm{LRS}^{\mathrm{SE}}$.

1. *If $\mathcal{S}$ is a live weighted T-system, or a live and consistent source private DSSP, then* $\mathrm{RS} = \mathrm{LRS}^{\mathrm{SE}}$ *(i.e. no spurious solutions). Moreover, if it is a live MG, then the integrality constraints can be disregarded (because in this case* $\mathbf{C}$ *is unimodular)*
2. *If $\mathcal{S}$ is a bounded, live, and reversible DF system, then* $\mathbf{m} \in \mathrm{RS}$ *iff* $\mathbf{m} \in \mathrm{LRS}^{\mathrm{SE}}$ *and the unique minimal T-semiflow of the net is fireable at* $\mathbf{m}$.
3. *If $\mathcal{S}$ is a live, bounded, and reversible FC system, then* $\mathbf{m} \in \mathrm{RS}(\mathcal{S})$ *iff* $\mathbf{m} \in \mathrm{LRS}^{\mathrm{SE}}(\mathcal{S})$ *(integrality constraints on $\boldsymbol{\sigma}$ can be disregarded) and every trap is marked at* $\mathbf{m}$.
4. *If $\mathcal{S}$ is a live EQ system or a live and consistent DSSP, and* $\mathbf{m_a}, \mathbf{m_b} \in \mathrm{LRS}^{\mathrm{SE}}$, *then* $\mathrm{RS}(\mathcal{N}, \mathbf{m_a}) \cap \mathrm{RS}(\mathcal{N}, \mathbf{m_b}) \neq \emptyset$.

We can take advantage of the above statements in a diversity of situations. For instance, the reachability characterisation for live MG allows to analyse some of their properties through linear programming. Even the last, and weakest, statement in the above theorem — a directedness result at the level of the linearised reachability graph — can be very helpful. In particular, it implies that there are no spurious deadlocks in live EQ systems, or live and consistent DSSP. Therefore, the deadlock-freeness analysis technique presented in Subsection 6.5.3 — which in these cases requires a single equation system — allows to decide liveness.

Figure I.2.3 shows an example of a live and 1-bounded system with spurious deadlocks.

6.7.4 Analysis of liveneness and boundedness

One of the properties that supports the claim that "good" behavior should be easier to achieve in some subclasses than in general systems is liveness *monotonicity* wrt. the initial marking. This means that liveness, provided that the net is "syntactically" correct as we shall precise later, is a matter of having enough tokens in the buffers (customers, resources, initial data, etc.), differently to what happens in general systems where the addition of tokens may well cause deadlocks due to poorly managed competition. For instance, in the net system of Figure 6.1.b adding a token (in p_5) to the initial marking destroys liveness.

Theorem 6.23 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a live EQ system or DSSP. The EQ system or DSSP $\langle \mathcal{N}, \mathbf{m_0} + \Delta\mathbf{m_0} \rangle$, where $\Delta\mathbf{m_0} \geq \mathbf{0}$ is live too.*

Very often, a net system is required to be live and bounded. As we saw in Section 6.3 the verification of liveness can be very hard. In some cases we are able to decide using structural methods alone; in other cases we can characterise the nets that can be lively and boundedly marked, so the costful enumeration analysis needs to be used only when there is a chance of success.

Theorem 6.24 *Let $\mathcal{N}$ be an EQ or DSSP net. A marking $\mathbf{m_0}$ exists such that $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is a live and bounded EQ system or DSSP iff $\mathcal{N}$ is strongly connected, conservative (or consistent), and* $\text{rank}(\mathbf{C}) = |\text{SEQS}| - 1$. *Moreover, in EQ systems, liveness of the whole system is equivalent to liveness of each P-component (the P-subnets generated by the minimal P-semiflows).*

Particular cases of the above result are well-known in net theory. For instance, in the ordinary case, the P-components of a FC net are strongly connected SM, which are live iff they are marked, so the liveness criterion can be stated as "there are no unmarked P-semiflows". In the case of MG, which are always consistent and $\text{rank}(\mathbf{C}) = |\text{SEQS}| - 1 = |T| - 1$, the existence of a live and bounded marking is equivalent to strong connectedness. Since their P-components are their circuits, liveness can be checked removing the marked places and verifying that the remaining net is acyclic.

6.8 Logical properties in time constrained models

It must be noticed that the interpretations concerning timing or synchronization with external events restrict the behavior of the underlying autonomous model, so they should be taken into account for the analysis. On the one hand this may become extremely complicated in some cases because the notion of state must be enlarged, e.g., time PNs [4]. On the other hand performing analysis of the autonomous system only may not be conclusive except for some particular properties and subclasses of systems. For instance, the autonomous PN in Figure 6.17 (a) is not bounded unless the interpretation ensures that t' fires as often as t; the autonomous PN in (b) is not live (fire t twice) unless the interpretation precises that the conflict is resolved in alternating fashion; in (c), if t takes always more time than t' to fire then the system will not return to the initial marking and t'' will die, although the autonomous model is live and reversible. In general it can be said that *safety* properties of the autonomous system are preserved under any interpretation while *liveness* properties are neither necessary nor sufficient [35].

Fortunately, in some subclasses the the interpretation is not so disturbing. For instance, we shaw that in strongly connected and bounded EQ or DSSP liveness of the autonomous model is preserved under any "reasonable" interpretation (i.e., allowing progress and fairly resolving local conflicts). Similarly,

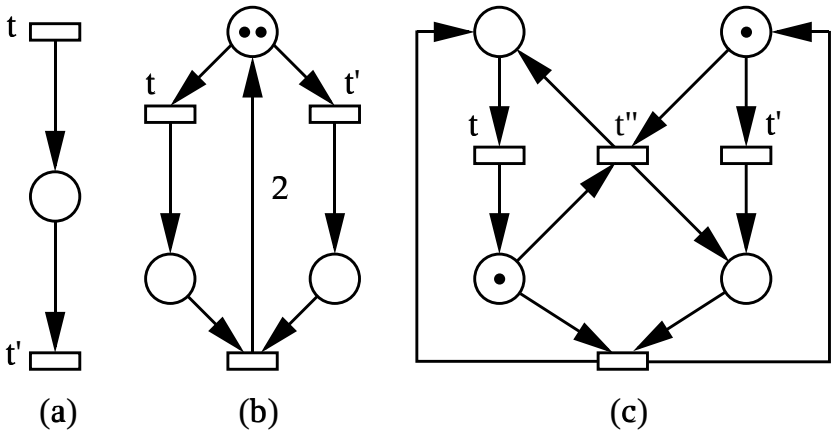


Figure 6.17: The interpretation affects qualitative properties.

some interpretations preserve the properties for every net system. For instance, under interpretations where the firing delay of transitions may range from zero to infinity (e.g., stochastic PN), the interpreted model has the same logical properties as the underlying autonomous model.

Bibliography

[1] *International Conference on Computer-Aided Verification.* The proceedings to date have been published as *Lecture Notes in Computer Science (LNCS)* **407** (1989), **531** (1990), **575** (1991), **663** (1992), **697** (1993), **818** (1994).

[2] M. Ajmone-Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* Wiley, 1995.

[3] G. Berthelot. Transformations and decompositions of nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986. Part I*, volume 254 of *Lecture Notes in Computer Science*, pages 359–376. Springer Verlag, Berlin, 1987.

[4] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Trans. on Software Engineering*, 17(3):259–273, 1991.

[5] E. Best, L. Cherkasova, J. Desel, and J. Esparza. Characterization of home states in free choice systems. Berichte 7/90, Hildesheimer Informatik, Hildesheim, Germany, July 1990.

[6] E. Best and K. Voss. Free choice systems have home states. *Acta Informatica*, 21:89–100, 1984.

[7] G.W. Brams. *Réseaux de Petri: théorie et pratique (2 vols.).* Masson, Paris, 1983.

[8] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):x–y, December 1991. [Special issue on *Multidimensional Queueing Networks*].

[9] J.M. Colom. *Análisis estructural de Redes de Petri, programación lineal y geometría convexa.* PhD thesis, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, Zaragoza, España, June 1989.

[10] J.M. Colom and M. Silva. Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal P-semiflows.

In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 79–112. Springer Verlag, Berlin, 1991.

[11] J.M. Colom and M. Silva. Improving the linearly based characterization of P/T nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 113–145. Springer Verlag, Berlin, 1991.

[12] J. Desel. A proof of the rank theorem for Extended Free Choice nets. In K. Jensen, editor, *Application and Theory of Petri Nets 1992*, volume 616 of *Lecture Notes in Computer Science*, pages 134–153. Springer Verlag, Berlin, 1992.

[13] J. Desel and J. Esparza. Reachability in cyclic Extended Free Choice nets. *Theoretical Computer Science*, 114:93–118, 1993.

[14] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1995.

[15] A. Desrochers, H. Jungnitz, and M. Silva. An approximation method for the performance analysis of manufacturing systems based on GSPNs. In *Procs of the Third International Conference on Computer Integrated Manufacturing and Automation Technology (CIMAT'92)*, pages 46–55. IEEE Computer Society Press, 1992.

[16] J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *J. Inform. Process. Cybernet*, 30(3):143–160, 1994.

[17] J. Esparza and M. Silva. On the analysis and synthesis of free choice systems. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 243–286. Springer Verlag, Berlin, 1991.

[18] A. Finkel. The minimal coverability graph for Petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1993*, volume 674 of *Lecture Notes in Computer Science*, pages 210–243. Springer Verlag, Berlin, 1993.

[19] M. Jantzen. Complexity of Place/Transition nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Central Models and Their Properties. Advances in Petri Nets 1986. Part I*, volume 254 of *Lecture Notes in Computer Science*, pages 413–434. Springer Verlag, Berlin, 1987.

[20] C. Johnen. Algorithmic verification of home spaces in P/T systems. In *Procs IMACS 1988. 12th World Congress on Scientific Computation*, pages 491–493, 1988.

[21] N.D. Jones, L.H. Landweber, and Y.E. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4:277–299, 1977.

[22] R. M. Karp and R. E. Miller. Properties of a model for parallel computations: Determinacy, termination, queueing. *SIAM Journal on Applied Mathematics*, 14(6):1390–1411, 1966.

[23] R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer Sciences*, 3:147–195, 1969.

[24] L. H. Landweber and E. L. Robertson. Properties of conflict-free and persistent Petri nets. *Journal of the ACM*, 25(3):352–364, 1978.

[25] K. Lautenbach. Linear algebraic techniques for Place/transition nets. In W. Brauer et al., editor, *Petri Nets: Central Models and their Properties. Advances in Petri Nets 1986*, volume 254 of *Lecture Notes in Computer Science*, pages 142–167. Springer Verlag, Berlin, 1987.

[26] K. Mehlhorn. *Graph algorithms and NP-completeness.* Springer-Verlag, Berlin, 1984.

[27] G. Memmi and G. Roucairol. Linear algebra in net theory. In W. Brauer, editor, *Net Theory and Applications*, volume 84 of *Lecture Notes in Computer Science*, pages 213–223. Springer Verlag, Berlin, 1980.

[28] K.G. Murty. *Linear Programming.* John Wiley and Sons, New York, 1983.

[29] G.L. Nemhauser, A.H. Rinnoy Kan, and M.J. Todd. *Optimization*, volume 1 of *Handbook in Operations Research and Management Science.* North Holland, Amsterdam, 1989.

[30] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization.* John Wiley and Sons, 1988.

[31] J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, New York, 1981.

[32] L. Recalde, E. Teruel, and M. Silva. Modeling and Analysis of Sequential Processes that Cooperate Through Buffers. *IEEE Trans. on Robotics and Automation*, 14(2):267–277, 1998.

[33] W. Reisig. *Petri Nets - An Introduction*, volume 4 of *Springer EATCS Monographs in Theoretical Computer Science.* Springer-Verlag, Berlin Heidelberg, 1985.

[34] M. Silva. Sur le concept de macroplace et son utilisation pour l'analyse des réseaux de Petri. *R.A.I.R.O. Automatique/Systems Analysis and Control*, 15(4):335–345, Avril 1981.

[35] M. Silva. *Las redes de Petri en la Automática y la Informática.* Editorial AC, Madrid, 1985.

[36] M. Silva. Logical controllers. In *Proceedings of IFAC Symposium on Low Cost Automation*, pages 157–166, 1989.

[37] M. Silva and J.M. Colom. On the computation of structural synchronic invariants in P/T nets. In G. Rozenberg, editor, *Advances in Petri Nets 1988*, volume 340 of *Lecture Notes in Computer Science*, pages 386–417. Springer Verlag, Berlin, 1988.

[38] E. Teruel, J. M. Colom and M. Silva. Choice-free Petri Nets: A Model for Deterministic Concurrent Systems with Bulk Services and Arrivals. *IEEE Trans. on Systems, Man, and Cybernetics*, 27(1):73–83, 1997.

[39] E. Teruel, J.M. Colom, and M. Silva. Linear analysis of deadlock-freeness of Petri net models. In *Procs. of the European Control Conference, ECC'93*, pages 513–518, Groningen, The Netherlands, June 1993.

[40] E. Teruel and M. Silva. Structure theory of equal conflict systems. *Theoretical Computer Science*, 153(1-2):271–300, 1996.

[41] V. Valero, D. Frutos, and F. Cuartero. Simulation of timed Petri nets by ordinary Petri nets and applications to decidability of the timed reachability problem and other related problems. In *Procs of the Fourth IEEE International Workshop on Petri Nets and Performance Models (PNPM'91)*, pages 154–163. IEEE Computer Society Press, 1991.

[42] V. Valero, D. Frutos, and F. Cuartero. Decidability of the strict reachability problem for timed Petri nets with rational and real durations. In *Procs of the Fifth International Workshop on Petri Nets and Performance Models (PNPM'93)*, pages 138–147, Toulouse, France, October 1993. IEEE Computer Society Press.

[43] A. Valmari. Stunnorn sets for reduced state space generation. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer Verlag, Berlin, 1991.

[44] A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4):297–322, 1992.

[45] W. Vogler. Live and bounded free choice nets have home states. *Petri Net Newsletter*, 32:18–21, April 1989.

Chapter 7

Analysis Techniques for Coloured Well-Formed Systems

The use of P/T nets for representing complex systems may result in models with quite many places and transitions. Part of this complexity can be hidden when higher-level models, such as coloured nets, are used for the representation. Yet, whatever the model, the goal is to analyze the system and the questions remain the same : is the system live ? reversible ? etc. Techniques for answering these questions if the system has been modelled with a P/T net have been presented in Chapter 6.

The first strategy for deciding such properties for a coloured net would be to transform the net into an equivalent P/T net, namely to *unfold* it, and then apply the techniques presented in Chapter 6. There are actually two important drawbacks to this approach : it will multiply the number of places, transitions and arcs, hence the cost for computing the results will be high. Also, the handling of these results may not be easy. Consider the simple case of a set of processes that can be in two states modeled by places $P1$ and $P2$ respectively. To check that the model is correct, i.e., that places $P1$ and $P2$ are never marked simultaneously for the same colour, the designer will have to check as many invariants as the number of processes. Of course, it would be easier to have some kind of generic relation that can be instantiated with the identities of processes and such that an instantiation corresponds to an invariant of the P/T net. Knowing that the relation holds for every process, the verification could even be done without actually performing the instantiation.

The idea is thus to adapt to coloured nets the analysis techniques defined for P/T nets, which actually means that we must be able to work directly at the net level in the coloured case. This implies to handle colour functions instead of numerical values. If there is no constraint in the definition of the functions, again the cost will be very high. But even if the functions are constrained, some

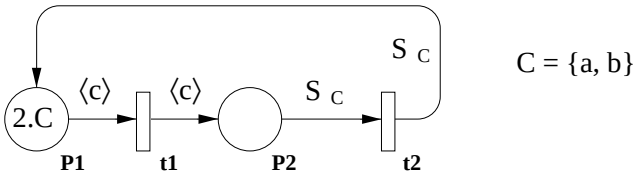


Figure 7.1: A simple well-formed net

analysis results defined for P/T nets do not extend automatically to coloured nets. This is the case for instance of the computation of siphons and traps.

The aim of this chapter is to show how the syntax of Well-Formed Nets can be exploited to derive efficient analysis techniques. It comprises three parts: the first part describes a technique that takes advantage of the intrinsic symmetries of a system, which are reflected in the basic colour classes, to derive a reduced representation of the reachability graph. This reduced representation, called Symbolic Reachability Graph, is based on the notion of equivalent markings and equivalent firings. It is obtained directly, i.e., without constructing first the complete reachability graph, and preserves most of the properties that are usually checked on the reachability graph.

The second part extends two structural analysis techniques to Well-formed nets, namely P and T-semiflows computation and net reduction. In particular, it addresses the problem of what the definition of a linear invariant should be in a coloured net.

The third part defines a structural technique that using the well-structured syntax of WNs, allows to simplify the colour structure of a WN model when redundancies are recognized in it. Both the check of redundancy and the *de-colourization* procedures can be performed automatically working at the level of the net structure.

7.1 Symbolic Reachability Graph

The aim of this section is not to present in a complete and very formal way the SRG approach, but rather to explain intuitively the concept of symmetry in nets and show how we can take advantage of symmetries to build a reduced representation of the reachability graph on which basic properties such as liveness or reversibility can be checked.

7.1.1 Introductory example

Let us consider the WN model in Figure 7.1.

There is one basic class C that contains two colours a and b. Let s be a permutation on C. There are actually two possibilities : either s is the identity or it commutes a and b. We can notice the following properties on the model :

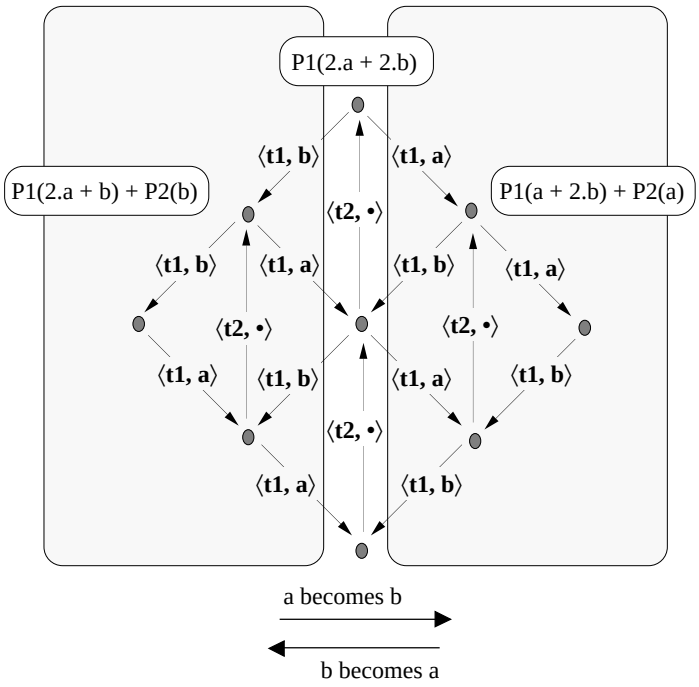


Figure 7.2: Reachability graph of the WN in fig. 7.1

whatever the reachable marking $\mathbf{m}$ under consideration, if $\mathbf{m} \xrightarrow{\langle t,c \rangle} \mathbf{m}'$ holds, where t is either t_1 or t_2 and c is either a or b, then by applying s on :

$\mathbf{m}$: we obtain another reachable marking, that we denote by $s.\mathbf{m}$. For instance, with $\mathbf{m}_1 = p_1(2.a + b) + p_2(b)$, we have $s.\mathbf{m}_1 = p_1(a + 2.b) + p_2(a)$, which is another reachable marking.

$\mathbf{m}$ and c : we obtain another legal firing $s.\mathbf{m} \xrightarrow{\langle t,s.c \rangle} \mathbf{m}''$, with $\mathbf{m}'' = s.\mathbf{m}'$. For instance, from the firing $\mathbf{m}_1 \xrightarrow{\langle t_1,b \rangle} p_1(2.a) + p_2(2.b)$, we deduce the firing $s.\mathbf{m}_1 \xrightarrow{\langle t_1,a \rangle} p_1(2.b) + p_2(2.a)$.

This properties are actually related to the structure of the colour functions that ensures potentially equivalent behaviours for a and b, and can easily be checked on the reachability graph presented in Figure 7.2. Based on these observations, we can make the following remarks :

1. it is not necessary to include both $\mathbf{m}$ and $s.\mathbf{m}$ in the graph : once $\mathbf{m}$ is in the graph, we know that $s.\mathbf{m}$ is also reachable. Moreover, as $\mathbf{m}'' = s.\mathbf{m}'$, by repeatedly applying the former property, we deduce that the whole subgraph originating at $s.\mathbf{m}$ can be derived from the subgraph originating at $\mathbf{m}$.

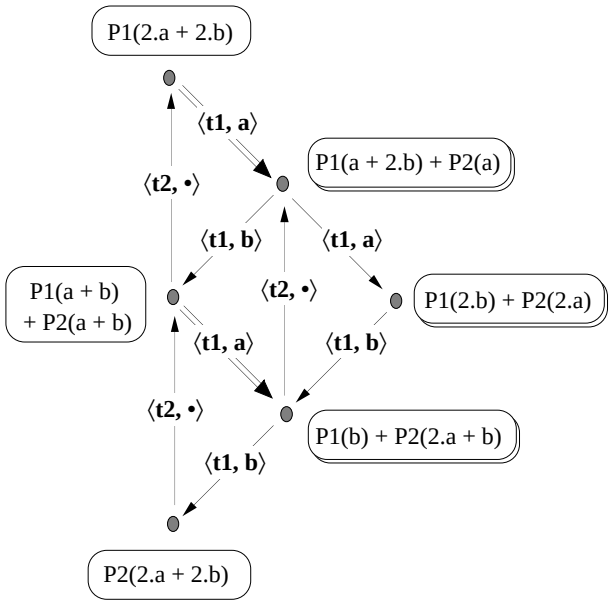


Figure 7.3: Symbolic reachability graph of the WN in fig. 7.1

2. if $\mathbf{m}$ and $s.\mathbf{m}$ are identical but c and $s.c$ are different, which happens for instance in marking $\mathbf{m} = p_1(2.a + 2.b)$ if s permutes a and b, it is not necessary to have both firings in the graph : having one ensures that the other also exists.

Based on the first remark, we introduce the notion of symbolic marking : once a group of symmetries has been defined, e.g., the set of permutations s in our example, all the markings that are identical within the application of a symmetry are grouped in an equivalence class. An equivalence class is called *symbolic marking* and only one representative of the class, for instance an element of this class, is included in the reachability graph.

The second remark leads to the notion of symbolic firing. If the application of a symmetry s (different from the identity) in a marking $\mathbf{m}$ does not modify $\mathbf{m}$, it means that several colours have the same distribution of tokens in places. These colours can be used interchangeably in a transition firing instance. Hence, the corresponding firings are not all represented in the graph, but only one is included.

By applying the proposed reductions, we obtain a smaller reachability graph, called *Symbolic Reachability Graph* (SRG). The SRG for the example in Figure 7.1 is given in Figure 7.3. Symbolic markings are represented by one element of the class. Double boxes stand for classes with more than one element. The same holds for symbolic firings.

However, having a reduced representation of the reachability graph is useful only if this representation can be used to check properties on the system. In

the next section, we present some interesting properties of the SRG that will be exploited in a later chapter for a performance evaluation of WN models. Other properties of the SRG that are useful for a qualitative analysis of the model can be found in [].

7.1.2 Verification of properties from the SRG

Let $(\mathcal{N}, \mathbf{m_0})$ be a coloured net system. Throughout this section, we make the assumption that a finite group ξ of symmetries s exists, such that the following proposition holds true :

$$\forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}), \forall c \in cd(t),\ \forall s \in \xi, \qquad \mathbf{m} \xrightarrow{\langle t,c \rangle} \mathbf{m}' \quad \Leftrightarrow \quad s.\mathbf{m} \xrightarrow{\langle t,s.c \rangle} s.\mathbf{m}' \quad (7.1)$$

We use $\widehat{\mathbf{m}}$ to denote the symbolic marking to which $\mathbf{m}$ belongs. Symbolic marking $\widehat{\mathbf{m}}'$ is reachable from symbolic marking $\widehat{\mathbf{m}}$ iff $\exists \mathbf{m}' \in \widehat{\mathbf{m}}'$, $\exists \mathbf{m} \in \widehat{\mathbf{m}}$ such that $\mathbf{m}'$ is reachable from $\mathbf{m}$.

We now analyze how the SRG can be used to check some basic logical properties of the system.

Boundedness From Assumption 7.1 and the definition of symbolic reachability, it is straightforward that, if $\widehat{\mathbf{m_0}}$ is a symbolic marking and $\mathrm{SRS}(\mathcal{N}, \widehat{\mathbf{m_0}})$ is the set of symbolic markings reachable from $\widehat{\mathbf{m_0}}$, then

$$\bigcup_{\mathbf{m_0} \in \widehat{\mathbf{m_0}}} \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) \ = \ \mathrm{SRS}(\mathcal{N}, \widehat{\mathbf{m_0}})$$

This property leads to the following observation : if RS is finite, so is SRS. Now, if SRS is finite, there is a finite number of classes of markings. Hence, there is a finite number of markings, unless a class contains an infinite number of markings. Such a situtation cannot happen if all the sets of basic colour classes are finite. As finiteness of the reachability set is equivalent to boundedness, we can conclude that in the case of finite colour sets, boundedness of the system is equivalent to finiteness of the symbolic reachability set.

Reversibility As it has been said in Chapter 6, the reversibility of the system is equivalent to the strong connection of the reachability graph. From the definition of symbolic reachability, it is easy to deduce that if $\mathrm{RG}(\mathcal{N}, \mathbf{m_0})$ is strongly connected, so is $\mathrm{SRG}(\mathcal{N}, \widehat{\mathbf{m_0}})$. Vice versa, the strong connection of SRG ensures that we have the situation depicted in Figure 7.4 (a). From Assumption 7.1 and the finiteness of ξ, we deduce the situations in Figure 7.4 (b) and (c). The grey arrows in Figure 7.4 (c) represent the existence of a firing sequence between two markings. Hence, if $\widehat{\mathbf{m_0}}$ is reduced to one element, which is often the case, the strong connection of SRG also implies the strong connection of RG.

What may happen however is that $\widehat{\mathbf{m_0}}$ is a set of markings. In this case, we may have a set of disconnected RGs, like in Figure 7.4 (c). Nevertheless, each RG is strongly connected. If the SRG is strongly connected, a sufficient condition

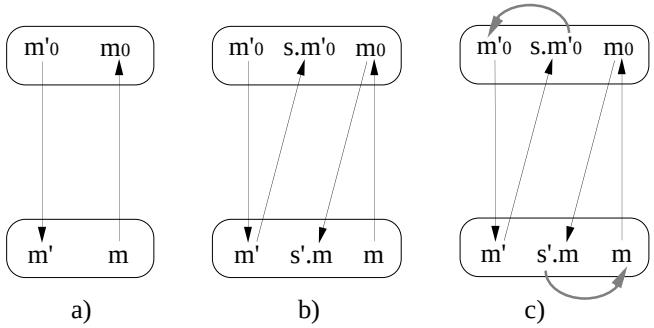


Figure 7.4: Strong connection of the reachability graph. a) SRG is strongly connected. b) Assumption 7.1 holds. c) ξ is finite.

for having one strongly connected RG is that the two following propositions hold true.

- $\xi = \xi_1 \circ \ldots \circ \xi_n$, i.e., $s \in \xi \Leftrightarrow s = s_1 \circ \ldots \circ s_n$ with $s_i \in \xi_i$.
- $\forall i, \exists \widehat{\mathbf{m}} \in \mathrm{SRG}(\mathcal{N}, \widehat{\mathbf{m_0}}), \forall m \in \widehat{\mathbf{m}}, \forall s_i \in \xi_i, s_i.m = m$.

However, for general classes of coloured nets, this condition is not useful in practice.

Home state The problem of reversibility is actually a particular case of the search for home states. The loss of information when building the SRG instead of the RG is thus of the same order.

To compare the SRG and the RG, we introduce the notion of *home set* : a home set is a set of markings such that, from every reachable marking, at least one element of the set can be reached. The two following propositions are equivalent :

- $\widehat{\mathbf{m}}$ is a home state for $(\mathcal{N}, \widehat{\mathbf{m_0}})$;
- $\forall \mathbf{m_0} \in \widehat{\mathbf{m_0}}$, $\{\mathbf{m} \in \widehat{\mathbf{m}}\}$ is a home set for $(\mathcal{N}, \mathbf{m_0})$.

A consequence of this property is that there may be only one terminal strongly connected component in the SRG whereas there are several in the RG. Such a situation occurs for the Petri net in Figure 7.5, for which the RG and SRG are given in Figures 7.6 and 7.7. The grey boxes underline a home set and a home state respectively.

Marking mutual exclusion Before considering how this property transposes from the RG to the SRG, we first define what marking mutual exclusion means for a coloured net. Let p and p' be two places, $c \in cd(p), c' \in cd(p')$. $p(c)$ and

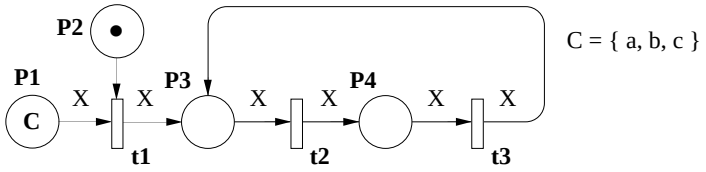


Figure 7.5: There is one terminal SCC in the SRG but several in the RG

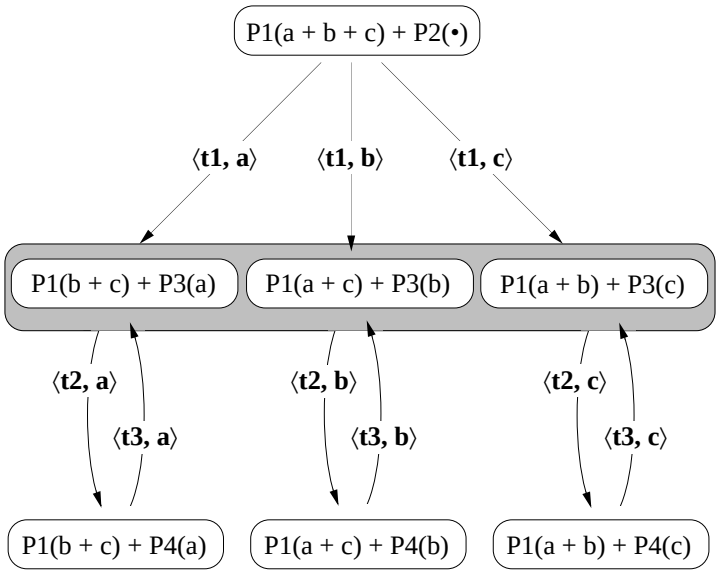


Figure 7.6: RG of the model in Figure 7.5

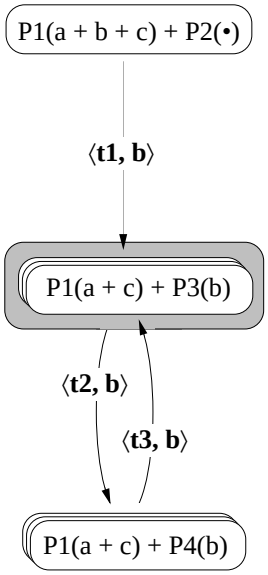


Figure 7.7: SRG of the model in Figure 7.5

$p'(c')$ are in mutual exclusion if they can never be simultaneously marked. In other words,

$$\forall \mathbf{m} \in RS(\mathcal{N}, \mathbf{m_0}), \quad \mathbf{m}[p, c] \times \mathbf{m}[p', c'] = 0.$$

Without loss of generality, we consider now that a symbolic marking is represented in the SRG by one element of the class. We use $\widehat{\mathbf{m}}$ to denote a reachable symbolic marking, as well as its representative (unless confusion may arise). If, for all $s \in \xi$, we define marking $s.\mathbf{m}$ as $s.\mathbf{m}[p, s.c] = \mathbf{m}[p, c]$ for every place p and every colour c in $cd(p)$, which is actually the definition that we have used in the introductory example, then we have the following property :

$$\begin{array}{lll} \forall \widehat{\mathbf{m}} \in & SRS(\mathcal{N}, \widehat{\mathbf{m_0}}), & \forall s \in \xi, \widehat{\mathbf{m}}[p, s.c] \times \widehat{\mathbf{m}}[p', s.c'] = 0 \\ & \Rightarrow & \forall \mathbf{m_0} \in \widehat{\mathbf{m_0}}, \forall \mathbf{m} \in RS(\mathcal{N}, \mathbf{m_0}), \mathbf{m}[p, c] \times \mathbf{m}[p', c'] = 0 \end{array}$$

where p, p' are any places of the net and c and c' are any colours in $cd(p)$ and $cd(p')$ respectively.

The example in Figure 7.8 illustrates the use of the SRG for checking marking mutual exclusion. At first glance, we could suppose that relation $p_1(a) \times p_3(a) = 0$ holds for every marking. But there exists a symmetry, namely any permutation mapping c onto a in $\widehat{\mathbf{m}}$, which invalidates the property. Actually, this symmetry is correct because a and c have identical behaviours, and it is easy to check that there is a reachable marking of the net for which both p_1 and p_3 are marked for a.

Deadlock-freeness Deadlock-freeness is checked on the RG by verifying that there is no terminal state. From the definition of symbolic reachability, and

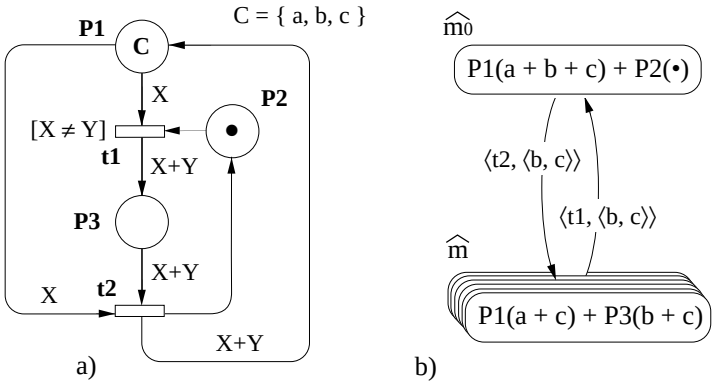


Figure 7.8: a) P1 and P3 are not in mutual exclusion for colour a. b) SRG of the model in a)

from Assumption 7.1, the equivalence between the two following propositions is straightforward.

- $(\mathcal{N}, \widehat{\mathbf{m_0}})$ is deadlock-free ;
- $\forall \mathbf{m_0} \in \widehat{\mathbf{m_0}}, (\mathcal{N}, \mathbf{m_0})$ is deadlock-free.

Liveness In a coloured net, this property is defined at the transition instance level. Transition t will be live for colour c if $\langle t, c\rangle$ is potentially fireable in all reachable markings.

Potential fireability of $\langle t, c\rangle$ can be tested directly on the RG by checking whether there is an arc labelled by $\langle t, c\rangle$. However, not all transition firings are represented in the SRG. There are actually two reasons for a firing not to be represented : either the marking at which it takes place is not in the SRG, i.e., it is not the representative of its class. Or there are equivalent firings taking place at the same marking, and the firing under consideration is not the representative of its class.

Hence, potential fireability cannot be checked by just looking at the SRG. However, we have the following property :

- $\langle t, c\rangle$ is potentially fireable at $\widehat{\mathbf{m_0}} \Rightarrow \forall s \in \xi, \exists \mathbf{m_0} \in \widehat{\mathbf{m_0}}, \langle t, s.c\rangle$ is potentially fireable at $\mathbf{m_0}$.

It follows a sufficient condition for testing liveness :

- $\langle t, c\rangle$ is potentially fireable at $\widehat{\mathbf{m_0}}$ and $\widehat{\mathbf{m_0}}$ is a home state $\Rightarrow \forall s \in \xi, \exists \mathbf{m_0} \in \widehat{\mathbf{m_0}}, \langle t, s.c\rangle$ is live in $(\mathcal{N}, \mathbf{m_0})$.

This condition partly takes advantage of the SRG as both the test of potential fireability and the characterization of home states can be performed on the SRG. But we have no simple condition on the graph like for the RG. The reason

is that, as we have seen for home states, it is not possible to know whether a terminal strongly connected component of the SRG corresponds to one or more SCCs in the RG. Hence, a transition that appears in all SCCs of the SRG may not appear in all SCCs of the RG.

The conclusion of this section is however that most important properties of the system can be checked directly on the SRG, which may be much smaller than the RG. We give some hints in the next section on when the construction of the SRG should be prefered to the RG.

7.1.3 Applicability of the SRG

We consider here the following question : when can the SRG technique be used, and when is it worth using it ? Most analysis techniques based on the reachability graph can be used only if the net is bounded, even if some questions can be answered by using the covering graph. However, there exists no such thing as a symbolic covering graph. The first condition for using the SRG is thus to consider only *bounded coloured nets*.

From a theoretical point of view, it is then possible to build an SRG as soon as Assumption 7.1 holds. This is actually the case for any coloured net if the group ξ of symmetries is limited to the identity function. In this case however, it is clear that the SRG is the same as the RG. The symbolic approach in the construction of the reachability graph is valuable only if a reasonably large set of symmetries can be defined. As we have seen in the introductory example, symmetries occur when different components of the system potentially have the same behaviour.

Anyway, the reduction ratio obtained when using the SRG instead of the RG is very difficult to estimate because for a same model, the cardinality of symbolic markings is very variable : it is one when all equivalent components are in the same state, and the largest cardinalities are obtained when all equivalent components are in different states. Of course, the state space reduction obtained with the SRG induces an increased computation cost but for reachability analysis, time is usually less critical than space.

Knowing that the system has symmetric components, there are still two possible approaches for building an SRG : either model the system with a general coloured net and define the set of symmetries, or use a model with a more constrained definition but for which the symmetries are known a priori. It is actually easy to get convinced that Assumption 7.1 is directly related to the structure of the colour functions. Hence, the introduction of a syntax in the model, and in particular in the definition of the colour functions, makes it possible to apply the SRG approach without having to define the set of symmetries. The next section shows how the syntax of Well-formed nets can be used to build a symbolic reachability graph in a completely automatic way.

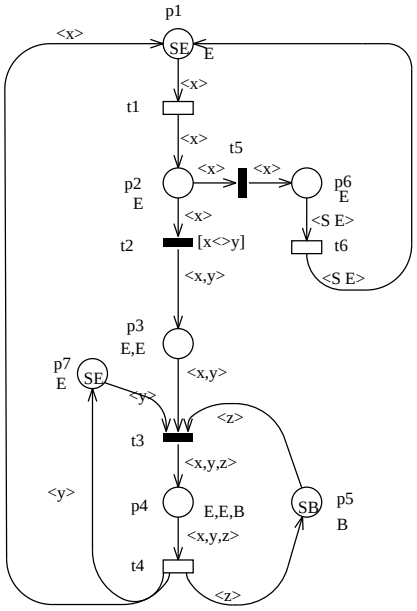


Figure 7.9: WN representation of the multicomputer PLC

7.1.4 Construction of the SRG

In this section, the different steps of the construction of the SRG for WNs are presented in a semi-formal way, and are illustrated on the example of the multicomputer PLC, with 4 computers and 2 busses. We recall the model in Figure 7.9. A more formal and complete presentation can be found in [3] or [10].

The following steps can be identified in the construction of the SRG :

- definition/construction of the set of symmetries for which Assumption 7.1 holds ;
- definition of the (unique) representative of a symbolic marking ;
- definition of a symbolic firing rule that operates directly on the representation of a symbolic marking.

Symmetries In Well-formed nets, objects of the same type are grouped in a same basic colour class. This class is possibly divided in several static subclasses. The equivalence between the behaviours of objects belonging to the same static subclass is ensured by the structure of the colour functions that does not make it possible to distinguish a particular object of the static subclass. For WNs,

the set of symmetries is thus defined as $\xi = \{s = \langle s_1, \ldots, s_h, s_{h+1}, \ldots, s_n \rangle\}$, where :

- for non-ordered classes $(0 < i \leq h)$, s_i is a permutation on C_i that respects static subclasses, i.e., $\forall C_{i,j}, s_i(C_{i,j}) = C_{i,j}$;
- for ordered classes $(h < i \leq n)$, s_i is a rotation on C_i that respects static subclasses. This condition implies that if the number n_i of static subclasses of C_i is greater than 1 then the only allowed rotation s_i is the identity.

The use of $C_1, \ldots, C_n$ for denoting basic colour classes is a formal notation used in definitions and proofs. But most often, classes are denoted by more significant names, such as B for the class of busses in the PLC example. Here, we will set $C_1 = B$ and $C_2 = E$. As neither B nor E is ordered or divided in static subclasses, any permutation s_B on B or s_E on E is acceptable as a colour permutation. The application of a symmetry on a non-basic colour is performed by applying the corresponding permutation or rotation on each basic colour class. Let us consider the following marking of the model :

$$\mathbf{m} = p_1(e_3) + p_3(\langle e_1 + e_4, e_2 \rangle) + p_4(\langle e_2, e_3, b_1 \rangle) + p_5(b_2) + p_7(e_1 + e_2 + e_4)$$

and a symmetry $s = \langle s_B, s_E \rangle$, with $s_B(b_1) = b_2, s_B(b_2) = b_1$ and $s_E(e_1) = e_1, s_E(e_2) = e_3, s_E(e_3) = e_4, s_E(e_4) = e_2$. The application of s on $\mathbf{m}$ gives marking :

$$s.\mathbf{m} = p_1(e_4) + p_3(\langle e_1 + e_2, e_3 \rangle) + p_4(\langle e_3, e_4, b_2 \rangle) + p_5(b_1) + p_7(e_1 + e_2 + e_3)$$

The main advantage of starting from a WN model for building the SRG is that it is not necessary to explicitly define the symmetries of the model. They are directly deduced from the structure of the colour classes and functions.

Representation of a symbolic marking By observing $\mathbf{m}$ and $s.\mathbf{m}$, it is obvious that the same interpretation can be given to both markings : one computer is currently accessing a remote memory, two others have issued a request for the same memory, which is different from the one currently accessed, and the fourth one is performing local computations. Actually, the application of an admissible symmetry to a marking only changes objects for other objects belonging to the same static subclass.

As these objects have the same potential behaviours, it appears convenient to abstract from their actual identities. A first proposition is to substitute the identity of objects with variables. The symbolic definition of classes B and E becomes $B = \{x_1, x_2\}$ and $E = \{y_1, y_2, y_3, y_4\}$ and

$$\widehat{\mathbf{m}} = p_1(y_1) + p_3(\langle y_2 + y_3, y_4 \rangle) + p_4(\langle y_4, y_1, x_1 \rangle) + p_5(x_2) + p_7(y_2 + y_3 + y_4)$$

is a possible representation of the symbolic marking of our example.

To define the semantics of this symbolic marking we need the definition of *valid assignment*. An assignment of objects from a basic class to the associated variables (x_i and y_i in our case) is said to be valid iff the following three conditions are verified :

1. every variable is assigned an object;
2. the same object is not assigned to more than one variable;
3. if the class is ordered, adjacent objects are assigned to subsequently numbered variables.

To each valid assignment corresponds an ordinary marking, and changing from a valid assignment to another one is equivalent to applying a symmetry in ξ on the associated markings. Hence, the new representation of $\widehat{\mathbf{m}}$ does stand for a class of equivalent markings. Actually, it is even possible to obtain a more compact description of the marking by grouping those variables that have the same distribution of tokens in places into *dynamic subclasses*. A dynamic subclass is characterized by its cardinality (i.e., the number of different objects represented by the dynamic subclass), and by the static subclass to which the represented objects belong (i.e., we can only group variables belonging to the same static subclass). In case of ordered basic classes, only contiguous objects can be represented by the same dynamic subclass and the ordering relation among objects is reflected by the ordering of the indexes of the dynamic subclasses.

In our example $\widehat{\mathbf{m}}$, the two variables y_2 and y_3 have the same distribution of tokens in the places ; they both appear in place p_7, and they are both associated with y_4 in p_3. Hence, they can be grouped in the dynamic subclass Z_E^1 of cardinality 2. All other dynamic subclasses have cardinality one. The new symbolic marking representation can thus be written as :

$$\widehat{\mathbf{m}} = p_1(Z_E^2) + p_3(\langle Z_E^1, Z_E^3\rangle) + p_4(\langle Z_E^3, Z_E^2, Z_B^1\rangle) + p_5(Z_B^2) + p_7(Z_E^1 + Z_E^3)$$

with $|Z_E^1| = 2$ and a cardinality one for other dynamic subclasses.

Yet, with such a representation, it is not guaranteed that we have a unique representative for each class. For instance,

$$\widehat{\mathbf{m}} = p_1(Z_E^2) + p_3(\langle Z_E^1, Z_E^3\rangle) + p_4(\langle Z_E^3, Z_E^2, Z_B^1\rangle) + p_5(Z_B^2) + p_7(Z_E^1 + Z_E^3)$$

with $|Z_E^1| = |Z_E^4| = 1$ is another valid representation of the same symbolic marking. This representation is said to be non-minimal, as subclasses Z_E^1 and Z_E^4 have the same marking and could be grouped in a single subclass. Another way of modifying the representation is to permutate the superscripts of the dynamic subclasses. For instance, we could also represent $\widehat{\mathbf{m}}$ as follows :

$$\widehat{\mathbf{m}} = p_1(Z_E^3) + p_3(\langle Z_E^2, Z_E^1\rangle) + p_4(\langle Z_E^1, Z_E^3, Z_B^2\rangle) + p_5(Z_B^1) + p_7(Z_E^1 + Z_E^2)$$

This representation is *ordered*, i.e., it is minimal for the lexicographical order if we represent the marking in a matrix form. More detail on the construction of the matrix and the computation of the minimum in the lexicographical order can be found in [3]. The minimal and ordered representation of a symbolic marking is unique. We call it *canonical representation* of the symbolic marking.

The next step in the construction of the SRG is to define a firing rule that applies on this representation.

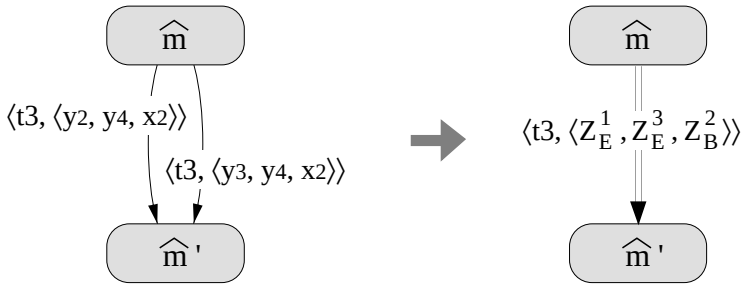


Figure 7.10: Grouping of ordinary firings

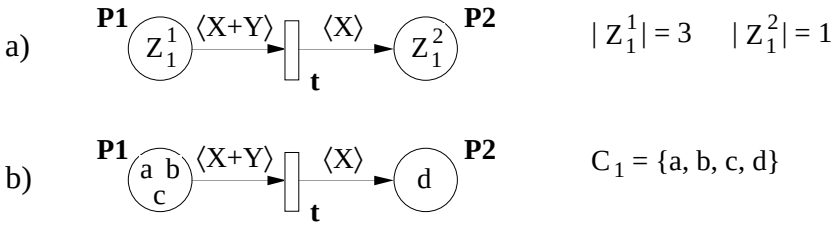


Figure 7.11: a) Firing from a symbolic marking. b) Firing from a valid assignment of this symbolic marking

Symbolic firing Due to the structure of the colour functions, objects that have the same distribution of tokens in places can be used interchangeably in a transition instance. By grouping these objects in dynamic subclasses, it is possible to know in advance which are the symbolic instances that lead to the same new symbolic marking. In our example, instead of testing transition firings for variables y_2 and y_3, we need to test it only for dynamic subclass Z^1_E, thus replacing two tests by one. Also, the result is a factorization of the arcs : instead of having one arc per element of the class, there is a single arc representing the set of firings. This situation is illustrated in Figure 7.10.

However, for this grouping to be efficient, it is necessary that the representation based on dynamic subclasses can be used directly for the firing. We thus need to define a *symbolic firing rule* that operates directly on this representation. Unfortunately, it is not possible to assign directly dynamic subclasses to the transition parameters instead of objects. Let us consider the symbolic marking in Figure 7.11a), a valid assignment of which is represented in Figure 7.11b). $\langle t_1, \langle a, b\rangle\rangle$ is a possible firing. The corresponding symbolic firing assigns Z^1_1 to X and also to Y. Hence, we would need $2.Z^1_1$ in p_1 to fire the transition. The proposed solution is to *split* a dynamic sublcass when it is assigned to a transition variable, so that the selected element of the class is isolated from the others. In our example, when Z^1_1 is assigned to X, it is split in two subclasses, namely $Z^{1,1}_1$, $|Z^{1,1}_1| = 1$, that represents the selected object, and $Z^{1,0}_1$, $|Z^{1,0}_1| = 2$ that represents the others.

Symbolic instance		Ordinary instance	Enabling of t
$\lambda = \langle \lambda_1(1), \lambda_1(2) \rangle$	$\mu = \langle \mu_1(1), \mu_1(2) \rangle$		
$\langle 1, 1 \rangle$	$\langle 1, 1 \rangle$	$\langle a, a \rangle, \langle b, b \rangle, \langle c, c \rangle$	NO
$\langle 1, 1 \rangle$	$\langle 1, 2 \rangle$	$\langle a, b \rangle, \langle a, c \rangle, \langle b, a \rangle$ $\langle b, c \rangle, \langle c, a \rangle, \langle c, b \rangle$	YES

Table 7.1: Correspondance between symbolic and ordinary firing for the example in Fig. 7.11

Now, when assigning a dynamic subclass to Y, there are two possibilities. Choosing $Z_1^{1,1}$ means that the same object is assigned to X and Y. Choosing $Z_1^{1,0}$ means that a different object is selected. $Z_1^{1,0}$ is split in two dynamic subclasses of cardinality one, namely $Z_1^{1,2}$ and $Z_1^{1,0}$.

Formally, a symbolic firing is denoted by means of two functions λ and μ. $\lambda = \langle \lambda_1, \ldots, \lambda_n \rangle$, where λ_i is used to determine, for the transitions variables that are assigned elements of C_i, in which dynamic subclass the assignment should be chosen. In the example, λ is reduced to λ_1. There are two variables assigned in C_1. $\lambda_1(1) = 1$ means that the first variable is assigned in Z_1^1, and $\lambda_1(2) = 1$ means the same for the second variable. Function $\mu = \langle \mu_1, \ldots, \mu_n \rangle$ is used to determine whether two assignments in the same dynamic subclass concern the same object or not. We first consider the case of non-ordered classes. Suppose that the x^{th} transition variable is assigned in Z_i^j, i.e., $\lambda_i(x) = j$. If it is the first one assigned in Z_i^j, then $\mu_i(x) = 1$. If not, and if it is assigned the same object as the y^{th} variable $(y < x)$ then $\mu_i(x) = \mu_i(y)$. Else, $\mu_i(x) = [max_{(y<x)}\ \mu_i(y)] + 1$. For ordered classes, $\mu_i(x)$ simply denotes the position of the selected object in dynamic subclass $Z_i^{\lambda_i(x)}$. The symbolic and the corresponding ordinary transition bindings for the example in Figure 7.11 are summarized in Table 7.1.

At this point, we introduce a notation that will be used later on for computing performance measures on the model. The cardinality of a symbolic firing, denoted with $|\widehat{\mathbf{m}}^{\langle t,\lambda,\mu \rangle}\!\!\rightarrow|$, is the number of ordinary firings from a single ordinary marking of the class that are represented by the symbolic firing. In the example presented in Table 7.1, this cardinality is equal to 6 for the enabled firing. It can be automatically computed by means of the following formula :

$$\prod_{i=1}^{h} \prod_{j=1}^{m(i)} \frac{card(Z_i^j)!}{(card(Z_i^j) - \mu_i^j)!}$$

where $\mu_i^k = \sup_{x\ :\ \lambda_i(x)=k}\ \mu_i(x)$, and $m(i)$ is the number of dynamic subclasses of C_i in $\widehat{\mathbf{m}}$.

Similarly, there is a formula for computing the number of ordinary markings represented by a symbolic marking. It can be found in [3].

We summarize the four steps for computing the canonical representation of

the symbolic marking $\widehat{\mathbf{m}}'$ obtained by firing $\langle t, \lambda, \mu \rangle$ in $\widehat{\mathbf{m}}$ (i.e., $\widehat{\mathbf{m}} \xrightarrow{\langle t,\lambda,\mu \rangle} \widehat{\mathbf{m}}'$).

1. Splitting $\widehat{\mathbf{m}}$ with respect to $[\lambda, \mu]$
   - if C_i is not ordered, a new dynamic subclass $Z_i^{j,k}$ of cardinality 1 is created for every couple $\langle j, k \rangle$ such that $\exists x, \lambda_i(x) = j$ and $\mu_i(x) = k$.
   - if C_i is ordered, the instanced dynamic subclasses in C_i are split into a set of new cardinality 1 dynamic subclasses.
2. Actual Firing $\widehat{\mathbf{m}} \xrightarrow{\langle t,\lambda,\mu \rangle}$
   The incidence functions are applied on the split representation, with dynamic subclasses replacing objects in the binding of the functions. The representation $\widehat{\mathbf{r}}_f$ of $\widehat{\mathbf{m}}'$ obtained after the firing is possibly non canonical.
3. Grouping $\widehat{\mathbf{m}}'$
   Compute a minimal representation $\widehat{\mathbf{r}}_m$ of $\widehat{\mathbf{m}}'$ by grouping dynamic subclasses of $\widehat{\mathbf{r}}_f$;
4. Ordering $\widehat{\mathbf{m}}'$
   Compute the canonical representation of $\widehat{\mathbf{m}}'$ by transforming $\widehat{\mathbf{r}}_m$ into an ordered representation.

7.1.5 Example of application : modeling and analysis of a multiserver random polling system

A single-server cyclic polling system (the simplest and most common polling system) comprises a set of waiting lines that receive arrivals from external world, and one server that cyclically visits the queues, providing service to customers that, after service completion, depart from the system. Such systems can be extended to the case of multiple servers. We consider here a *multiserver* polling system with a set $Q = \{q_1, \ldots, q_N\}$ of queues ($N \geq 1$) and a set $R = \{r_1, \ldots, r_S\}$ of servers ($1 \leq S \leq N$). A WN model of the system is given in Figure 7.12.

In our example, the *storage capacity* of each queue is equal to 1, so that no more than one customer can be at a queue at anytime. A token with colour q_i in place p_a means that queue i is empty. The arrival of a customer at queue i is represented by the firing of transition T_a for colour q_i. A token with colour q_i in place p_q thus stands for a customer waiting for service at queue i. This customer remains waiting until a server j polls queue i, which is represented by the arrival of a token with colour $\langle q_i, r_j \rangle$ in place p_p. In this case, transition $\langle t_s, \langle q_i, r_j \rangle \rangle$ fires because server j finds a customer waiting at queue i. A token with colour $\langle q_i, r_j \rangle$ is put in place p_s, representing the customer being served. Once the service is completed, transition $\langle T_s, \langle q_i, r_j \rangle \rangle$ fires, a token q_i comes back to place p_a meaning that queue i is now empty and a token $\langle q_i, r_j \rangle$ arrives at place p_r, representing the server moving to the next queue. Such a token is also produced when transition $\langle t_w, \langle q_i, r_j \rangle \rangle$ fires, which happens if queue i is empty when server j polls it : in this case, the server directly moves to the next queue.

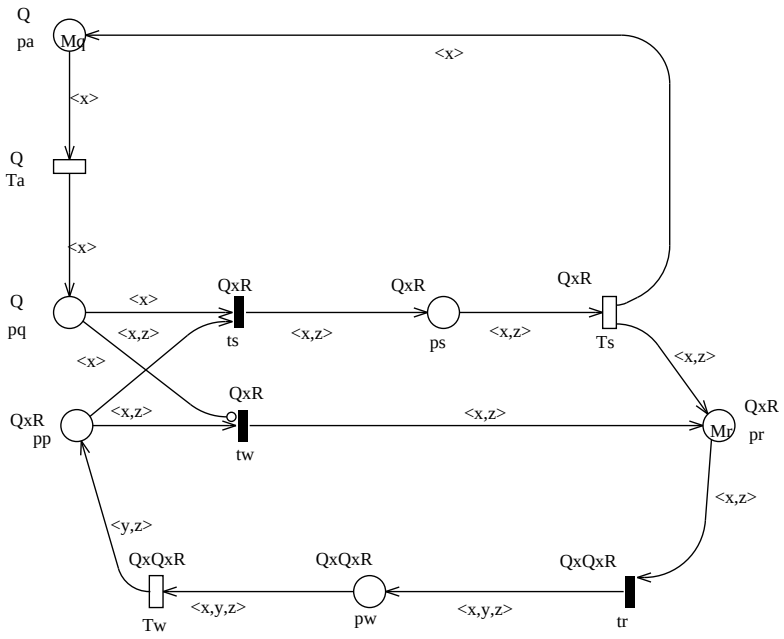


Figure 7.12: A WN model of a multiserver random polling system

For the model to be consistent with this description, we must guarantee that when a token $\langle q_i, r_j \rangle$ arrives in p_p, the decision to fire t_s or t_w is taken immediately, i.e., before the marking of Q can be modified by the arrival of a customer at queue i. This is done by setting the priority level of (all instances of) t_s and t_w to 1, while other transitions have priority 0.

The firing of t_r for colour $\langle q_i, q_k, r_j \rangle$ represents the choice of server j to visit queue k when leaving queue i. k is chosen randomly among all the queues including the one just visited. This is represented in the model by a free-choice conflict among all the instances $\langle q_i, x, r_j \rangle$ of transition t_r, where x can take any value in Q. The token produced in place p_w remains there until the firing of transition $\langle T_w, \langle q_i, q_k, r_j \rangle \rangle$, which models the time needed for the server for walking from queue i to queue k.

Let us come back to the action represented by t_r. As soon as a server leaves a queue, it *instantaneously* chooses the next queue to be visited. Hence, a token arriving in place p_r should leave it immediately. By assigning a higher priority to t_r, we take into account the fact that the choice of the next queue to be visited is a timeless action. Hence, we also assign priority 1 to all the instances of t_r. By doing so, we decrease the size of the reachability graph because place p_r will never contain more than one token, whereas if t_r had priority 0, it could contain up to $|R|$ tokens.

Initially, all the queues are empty, and we arbitrarily consider that all the servers are walking from a queue to this same queue. This is actually a possible state as every server can immediately come back to the queue they just visited. Hence, $\widehat{\mathbf{m_0}} = p_a(Z_Q^1 + Z_Q^2) + p_w(\langle Z_Q^1, Z_Q^1, Z_R^1 \rangle)$, with $card(Z_Q^1) = 1$, $card(Z_Q^2) = N - 1$ and $card(Z_R^1) = S$.

Table 7.2 compares the cardinality of the symbolic state space wih the number of equivalent ordinary markings for different values for the number N of queues and the number S of servers. The results have been computed using the software GreatSPN 2.0 [6].

Like for Place/Transition nets, techniques based on the reachability graph are the most general ones for analyzing a model. The symbolic firing rule can be easily cast into the usual algorithm structure for the computation of the Reachability Graph of a Petri net provided that the symbolic canonical representation is used to store the markings of the reachability set.

The space efficiency and time complexity of the symbolic approach strongly depend on the intrinsic symmetry of the model : the more objects with equivalent behaviours, the more markings in an equivalence class, the higher the space reduction degree. Compared with the RG construction, the most expensive step is the computation of a unique representative for each class of markings. It can be performed in a time of the same order of magnitude as the application of a symmetry. Hence, if the system is not symmetric, the time complexity for the SRG construction is the same order as for the RG.

Other analysis techniques can be extended from Place/Transition nets to Well-formed nets by taking advantage of the structure of the model.

	S = 2	S = 3
N = 2	31 / 31 (98 / 124)	-
N = 3	91 / 69 (876 / 768)	348 / 384 (9024 / 12186)
N = 4	161 / 107 (5168 / 3488)	904 / 822 (92284 / 94704)
N = 5	231 / 145 (24160 / 13280)	1569 / 1290 (661550 / 550080)
N = 6	301 / 183 (97248 / 45120)	2255 / 1758 ($3.78\ 10^6$ / $2.65\ 10^6$)
N = 7	371 / 221 (352576 / 141568)	-
N = 8	441 / 259 ($1.18\ 10^6$ / 418816)	-
N = 9	511 / 297 ($3.74\ 10^6$ / $1.18\ 10^6$)	-
N = 10	581 / 335 ($11.29\ 10^6$ / $3.23\ 10^6$)	-

Table 7.2: Symbolic state space cardinality (tangible / vanishing) for the WN model of the random polling system with S servers, N queues and queue capacities equal to 1. Numbers in parenthesis refer to non-symbolic states.

7.2 Invariants and Reductions for Well-formed Nets

7.2.1 Invariants

As in ordinary Petri nets, one of the main aspects of the structural verification of WNs is the generation of invariants. But before developping techniques for such a problem, some points must be clarified :

- how can we express an invariant of WNs and especially a linear invariant ?
- how can a family of (linear) invariants be characterized as a generative family of invariants ?
- how can we build a (generative) family of (linear) invariants ?

The aim of this section is to successively answer to these three questions in a concise way.

Presentation of linear invariants

The choice of an adequate definition of invariants should meet the following requirements. At first, an invariant of a high-level Petri net must be a high-level invariant. For instance, if we model processes by colours, it means that an invariant should express properties of the behaviour of one particular process but also of the behaviour of any process, or of the behaviour of any process except a particular one, etc. On the other hand, the definition should enable mathematical developments leading to efficient algorithms for computing invariants.

One can try to follow the definition of ordinary invariants, i.e. : an invariant is a weighted sum of the marking of the places left invariant by the firing of any transition. However such a definition involves two hidden extensions :

- what can be the weights on place markings ?
- there are multiple ways to fire a transition (as many as the size of the colour domain of the transition)

The essential point in the definition below is that the weights are colour functions. It can be interpreted as applying the colour function on the place marking corresponds to extract the relevant part of information contained in this marking for a given invariant. In order to be mathematically sound this function must have for domain the colour domain of the place and for codomain a common domain for the weights of the same invariant. This codomain may be viewed as the interpretation domain of the invariant and this requirement ensures that the weighted sum of the place markings is well defined.

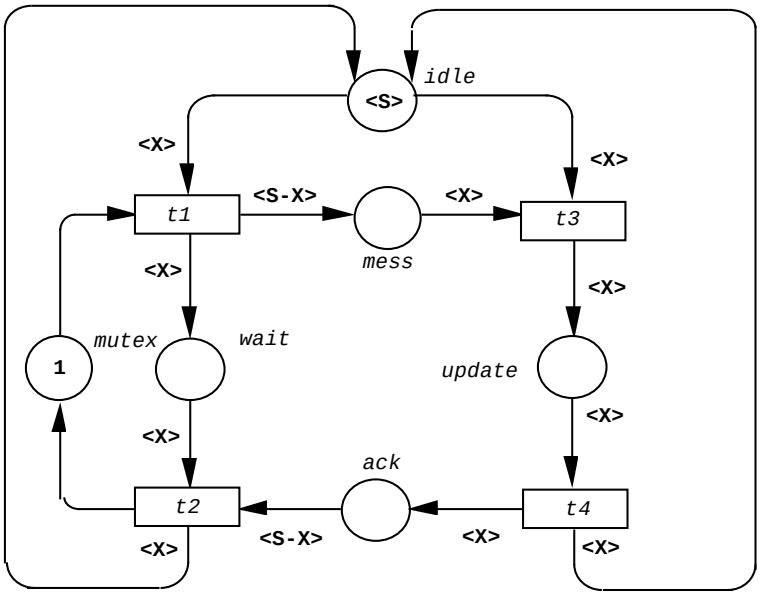


Figure 7.13: *A WN model of a replicated database*

Definition 7.1 *A linear invariant* $\mathbf{v}$ *of a well-formed net* $\mathcal{N}$ *is defined by :*

- $cd(\mathbf{v})$ *the colour domain of the invariant,*
- $\forall p \in P$, $\mathbf{v}(p)$ *a function from* $\mathbb{Z}^{cd(p)}$ *to* $\mathbb{Z}^{cd(\mathbf{v})}$,

such that :

$$\forall \mathbf{m}\ reachable\ marking, \sum_{p \in P} \mathbf{v}(p)(\mathbf{m}(p)) = \sum_{p \in P} \mathbf{v}(p)(\mathbf{m_0}(p))$$

The net of figure 7.13 models a database management with multiple copies. The access grant of the database is centralized and submitted to mutual exclusion.

The database is shared by a set of sites represented by the colour domain *Sites*. In order to modify the database an idle site (in place *idle*) must get the grant (a neutral token in place *mutex*) and once it has modified the file, it sends messages to the others sites. The whole action is modelled by the transition t_1 and the contents of the message is not modelled. Then the other sites update their own database (transition t_3) and send an acknowledgment (transition t_4). Once the active site has received all the acknowledgments, it releases the grant (transition t_2).

In order to simplify the net, accessing and modifying the database is modelled by a single transition (indivisible step) while the updating of the other sites is modelled by a place (divisible step).

Initially there is a token per site in place *idle* and a neutral token in place *mutex*.

Let us give a first example of a linear invariant :

$$\begin{array}{lcl} cd(\mathbf{v}) & = & Sites \\ \mathbf{v} & = & \langle x \rangle.idle + \langle x \rangle.wait + \langle x \rangle.update \end{array}$$

This linear invariant describes the behaviour of any site : either the site is idle, either it waits for the acknowledgments or it updates its database.

Computation of linear invariants

Now we focus on the computation of family of linear invariants. Ideally, we want a generative family of invariants, i.e., any linear invariant should be obtained by a linear combination of the invariants of the family. Keeping in mind what a linear invariant of a well-formed net is, we allow the coefficients of the combination to be functions with the same requirements on domain and codomain for these functions. It should be mentionned that this is the only way to keep a generative family of reasonnable size and moreover as we shall see in what follows to obtain significant flows (directly like items of the family or by linear combination).

We will not give in this subsection any algorithm but instead we will show on the previous example how to compute a generative family of invariants. Then we will interpret our family of invariants.

The corner stone of all the algorithms is to handle the incidence matrix in a similar way as it is done in the Gaussian elimination. However the elimination rules must be applied under conditions which ensure that no linear invariant will be "forgotten".

We start with the incidence matrix **C** of our example :

$$\mathbf{C} = \begin{array}{c} \begin{array}{cccc} \text{t1} & \text{t2} & \text{t3} & \text{t4} \end{array} \\ \left(\begin{array}{cccc} \langle x \rangle & -\langle x \rangle & 0 & 0 \\ -\langle x \rangle & \langle x \rangle & -\langle x \rangle & \langle x \rangle \\ \langle s-x \rangle & 0 & -\langle x \rangle & 0 \\ \text{-1} & 1 & 0 & 0 \\ 0 & 0 & \langle x \rangle & -\langle x \rangle \\ 0 & -\langle s-x \rangle & 0 & \langle x \rangle \end{array} \right) \end{array} \begin{array}{l} \langle x \rangle.wait \\ \langle x \rangle.idle \\ \langle x \rangle.mess \\ 1.mutex \\ \langle x \rangle.update \\ \langle x \rangle.ack \end{array}$$

On the right, the initial family of invariants is shown i.e. each place weight by the identity function of its colour domain. Now we proceed by a standard rule : adding to a line another one which has been premultiplied by a function. the only restriction with this rule is the consistency of domains and codomains

of functions. The result is shown below :

$$\begin{array}{cccc} \text{t1} & \text{t2} & \text{t3} & \text{t4} \\ \left(\begin{array}{cccc} \langle x\rangle & -\langle x\rangle & 0 & 0 \\ 0 & 0 & -\langle x\rangle & \langle x\rangle \\ 0 & \langle s-x\rangle & -\langle x\rangle & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \langle x\rangle & -\langle x\rangle \\ 0 & -\langle s-x\rangle & 0 & \langle x\rangle \end{array}\right) & \begin{array}{l} \langle x\rangle.wait \\ \langle x\rangle.idle + \langle x\rangle.wait \\ \langle x\rangle.mess - \langle s-x\rangle.wait \\ 1.mutex + 1.wait \\ \langle x\rangle.update \\ \langle x\rangle.ack \end{array} \end{array}$$

We apply on this new matrix a second rule which eliminates a line for which one of its coefficient is the only non nul coefficient of its column (here the first line). We require that this coefficient is an injective mapping.

$$\begin{array}{cccc} \text{t1} & \text{t2} & \text{t3} & \text{t4} \\ \left(\begin{array}{cccc} 0 & 0 & -\langle x\rangle & \langle x\rangle \\ 0 & \langle s-x\rangle & -\langle x\rangle & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & \langle x\rangle & -\langle x\rangle \\ 0 & -\langle s-x\rangle & 0 & \langle x\rangle \end{array}\right) & \begin{array}{l} \langle x\rangle.idle + \langle x\rangle.wait \\ \langle x\rangle.mess - \langle s-x\rangle.wait \\ 1.mutex + 1.wait \\ \langle x\rangle.update \\ \langle x\rangle.ack \end{array} \end{array}$$

The third rule we apply is identical to a Gaussian elimination rule i.e. delete a null column (here $t1$).

$$\begin{array}{ccc} \text{t2} & \text{t3} & \text{t4} \\ \left(\begin{array}{ccc} 0 & -\langle x\rangle & \langle x\rangle \\ \langle s-x\rangle & -\langle x\rangle & 0 \\ 0 & 0 & 0 \\ 0 & \langle x\rangle & -\langle x\rangle \\ -\langle s-x\rangle & 0 & \langle x\rangle \end{array}\right) & \begin{array}{l} \langle x\rangle.idle + \langle x\rangle.wait \\ \langle x\rangle.mess - \langle s-x\rangle.wait \\ 1.mutex + 1.wait \\ \langle x\rangle.update \\ \langle x\rangle.ack \end{array} \end{array}$$

We can iterate this process for eliminating the column $t2$ using the property that $\langle s-x\rangle$ is injective.

$$\begin{array}{ccc} \text{t2} & \text{t3} & \text{t4} \\ \left(\begin{array}{ccc} 0 & -\langle x\rangle & \langle x\rangle \\ 0 & -\langle x\rangle & \langle x\rangle \\ 0 & 0 & 0 \\ 0 & \langle x\rangle & -\langle x\rangle \end{array}\right) & \begin{array}{l} \langle x\rangle.idle + \langle x\rangle.wait \\ \langle x\rangle.mess - \langle s-x\rangle.wait + \langle x\rangle.ack \\ 1.mutex + 1.wait \\ \langle x\rangle.update \end{array} \end{array}$$

The elimination of $t3$ ends the algorithm as it eliminates at the same time $t4$. We obain the following family of invariants :

- The state of any site : either the site is idle, either it waits for the acknowledgments or it updates its database.

$$\langle x\rangle.wait + \langle x\rangle.idle + \langle x\rangle.update$$

- The state of the database : either the grant is present or a site is waiting for completing its transaction.

$$1.wait + 1.mutex$$

- The synchronization between sites : if a site is waiting then either any other site has a message for the current transaction, either it updates its copy, or it has send its acknowledgment.

$$\langle x \rangle.mess + \langle x \rangle.ack + \langle x \rangle.update - \langle s - x \rangle.wait$$

Some other significant linear invariants may also be obtained by combining the previous invariants. For instance, the following invariant says that either the grant is present and all the sites are idle or the grant is absent and the idle sites are exactly those who will receive a message or have sent their acknowledgment.

$$\langle x \rangle.idle - \langle s \rangle.mutex - \langle x \rangle.mess - \langle x \rangle.ack$$

Additional remarks

The computation of linear invariants we have described in the previous section can not be straightforwardly extended to general coloured nets or even to WNs. We emphasize below the three problems one must cope to provide a general algorithm :

- How to compose lines in order to cancel items of the matrix ?
- How to ensure that the last coefficient of a column is injective ?
- How to handle the two previous operations in a parametrized way i.e. independantly of the size of the colour domains (in our example, the number of sites) ?

A general algorithm for coloured nets (and also applicable to WNs) has been proposed in [9]. The key point of the algorithm is the intensive use of generalized semi-inverses. Its only restriction is to fix the size of the colour domains.

On the other hand, with restrictions to subclasses of coloured nets, it is possible to obtain algorithms which handle the parameters [15], [14] [17]. These algorithms transform the incidence matrix of functions into a set of matrices with coefficients taken in a ring of poynomials. The variables correspond to the parameters of the net. At this point, it is enough to apply a Gaussian-like elimination on these matrices. Each vector solution is finally transformed (in an inverse way) to a linear invariant.

7.2.2 Reductions

A reduction of nets is defined by some conditions of application and a method of transformation such that, if the orignal net fulfills the conditions, the reduced net has the same behaviour as the original one w.r.t. to generic properties. Reduction theory has been mainly developed by G. Berthelot [1] who has proposed ten reductions covering a wide area of application cases. Introducing a new reduction is interesting if :

- it covers a frequent behavioural situation (as will be described in the next subsection),
- the application conditions can be checked in a efficient way (e.g. by examination of the net structure or by linear invariant computation).

The generalization to high-level nets has been proposed by different authors [8] , [12] and [13]. We will follow the last reference to introduce some reductions. Then we will illustrate them on the example of the database management and at last we will discuss about a methodology to define new reductions.

Some reductions

We limit ourselves to two reductions for WNs (more information may be found in the previous reference) which are enough for our example. The style of definition will be unformal : the general interpretation of the reduction, the definition of application conditions and the method of transformation with for each item a corresponding interpretation. The set of properties preserved by these reductions is essentially the same one for the two reductions and covers liveness, boundedness, home state existence, unavoidable state existence, etc.

IMPLICIT PLACE SIMPLIFICATION

The reduction deletes a place which is *never the only one* to prevent the firing of transition. Such a place appears in two cases : either it is a redundant place which explicitely models an information implicitely contained in the other places or the place was not originally implicit but other reductions have transformed it.

The existence of an implicit place is ensured by a particular invariant. Such reduction illustrates an indirect use of the invariants for the model verification.

Definition 7.2 *Let $\mathcal{N}$ be a well-formed net, a place p is implicit iff :*

1. *there exists a linear invariant* $\mathbf{v}$ *such that :*
   - $cd(\mathbf{v}) = cd(p)$
   - $\mathbf{v}(p)$ *is the identity function of* $\mathrm{Bag}(cd(p))$
   - $\forall p' \neq p, \forall c \in cd(p'), -\mathbf{v}(p')(c) \in \mathrm{Bag}(cd(p'))$

2. $\forall t \in T, \forall c \in cd(t), \sum_{p' \in P} \mathbf{v}(p')(\mathbf{Pre}(p', t)(c)) \geq \sum_{p' \in P} \mathbf{v}(p')(\mathbf{m_0}(p'))$

Condition *2* ensures that in the initial marking, p will not be the only place to prevent the firing of any transition due to the positivity constraints included in Condition *1*. The fact that $\mathbf{v}$ is a linear invariant ensures that Condition *2* is also true for all reachable markings.

The transformation deletes the place and the bordering arcs.

Definition 7.3 *The reduced net* $\mathcal{N}_r = \langle P', T', \mathbf{Pre}', \mathbf{Post}', \mathcal{C}', cd' \rangle$ *with initial marking* $\mathbf{m_0}'$ *obtained from the net* $\mathcal{N}$ *by the simplification of the implicit place* p *is defined by :*

- $P' = P - \{p\}$
- $T' = T$
- $\mathcal{C}' = \mathcal{C}$
- $\forall t \in T', \forall p' \in P', cd'(t) = cd(t)$ *and* $cd'(p') = cd(p')$
- $\forall t \in T', \forall p' \in P', \mathbf{Pre}'(p', t) = \mathbf{Pre}(p', t)$ *and* $\mathbf{Post}'(p', t) = \mathbf{Post}(p', t)$
- $\forall p' \in P', \mathbf{m_0}'(p') = \mathbf{m_0}(p')$

POST-AGGLOMERATION OF TRANSITIONS

The principle of the post-agglomeration is the following. Suppose we are given a set H of transitions which represent global actions and which lead to an intermediate state represented by a token in place p. Suppose that the way of going out of this intermediate state is to fire a transition f which represents a local transition (i.e. without synchronization). Then the firing of any transition of H could be immediatly followed by the firing of f without modifying the global behaviour of the net. Doing this, the place p may be deleted and the firings of transition f may be included in the firing of any transition of H. However one must take some care in order to define the new items of the **Pre** and the **Post** matrices. For simplicity, we present here a restricted version of this reduction.

We begin by introducing the concept of a safe colour function required by the next definition. A safe function produces (or consumes depending on the arc) at most one token per colour of the place. Most of the time, the functions appearing in a model are safe functions.

Definition 7.4 *A colour function from* Bag*(E) to* Bag*(F) is safe iff :*

$$\forall c \in E, \forall c' \in F, f(c)(c') \leq 1$$

Definition 7.5 *Let* $\mathcal{N}$ *be a well-formed net, p a place which has for output the transition f and for inputs the set H of transitions with* $f \notin H$*, one can post-agglomerate f with H iff :*

1. $\forall h \in H, \mathbf{Post}(p,h)$ *is a safe function*
2. $cd(f) = cd(p)$ *and* $\mathbf{Pre}(p,f)$ *is the identity function*
3. *Initially* p *is unmarked*
4. *Any firing of* f *produces tokens*
5. *The only input place of* f *is* p

Condition *1* ensures that for a given colour, a transition of H produces one token per firing. Condition *2* ensures that such a token can be consumed by the firing of f with the same colour. Condition *3* could be overcome by substituting $\mathbf{m_0}$ by a set of initial markings after emptying place p. Nevertheless, place p is seldom marked. Condition *4* is necessary for preserving boundedness equivalence. The last condition with condition *2* is the crucial one as it ensures the immediate firing of f once p is marked.

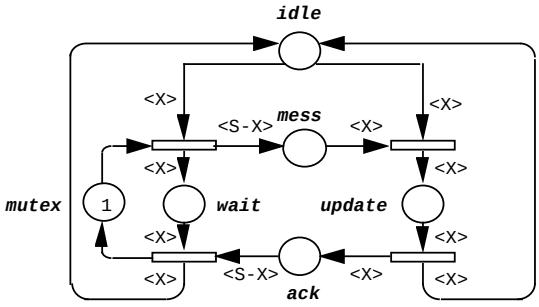
The transformation deletes the place p and the bordering arcs. Moreover any function on output arcs of a transition of H is obtained as the sum of the old function and the combined effect of the output of this transition followed by adequate firings of f (this explains the composition of functions).

Definition 7.6 *The reduced net* $\mathcal{N}_r = \langle P', T', \mathbf{Pre}', \mathbf{Post}', \mathcal{C}', cd' \rangle$ *with initial marking* $\mathbf{m_0}'$ *obtained from the net* $\mathcal{N}$ *by the post-agglomeration of the set of transitions* H *and the transition* f *is defined by :*

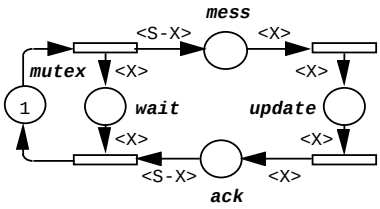
- $P' = P - \{p\}$
- $T' = T - \{f\}$
- $\mathcal{C}' = \mathcal{C}$
- $\forall t \in T', \forall p' \in P', cd'(t) = cd(t)$ *and* $cd'(p') = cd(p')$
- $\forall t \in T' - H, \forall p' \in P', \mathbf{Pre}'(p',t) = \mathbf{Pre}(p',t)$ *and* $\mathbf{Post}'(p',t) = \mathbf{Post}(p',t)$
- $\forall h \in H, \forall p' \in P', \mathbf{Pre}'(p',h) = \mathbf{Pre}(p',h)$ *and* $\mathbf{Post}'(p',h) = \mathbf{Post}(p',h) + \mathbf{Post}(p',f) \circ \mathbf{Post}(p,h)$
- $\forall p' \in P', \mathbf{m_0}'(p') = \mathbf{m_0}(p')$

Application to the example

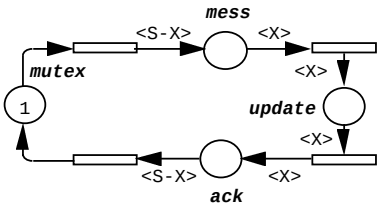
Figure 7.14 shows the reduction process of the model of database management. We could have reduced the initial net until we obtain a single transition using other reductions than the ones presented here. Nevertheless the final net presented in the figure is small enough to analyze its behaviour. The net language



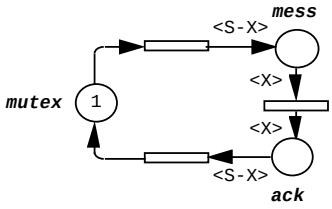
Simplification of the implicit place *idle*



Simplification of the implicit place *wait*



Post-agglomeration around *update*



Post-agglomeration around *mess*

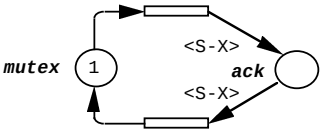


Figure 7.14: *The reduction of the WN model of a replicated database*

is the set of prefixes of $(\bigcup_{c \in Sites} t1(c).t2(c))^\star$. Its interpretation is straightforward : the behaviour of the net is an infinite sequence of database modifications done in mutual exclusion.

Let us give some explanations on the applications of the reductions :

- The flow associated to the implicit place *idle* is the one we have obtained by the combination of the generative family in subsection 7.2.1.
- The flow associated to the implicit place *wait* is :

$$1.wait - \frac{1}{n-1}.mess - \frac{1}{n-1}.update - \frac{1}{n-1}.ack$$

  where n is the number of sites
- During the post-agglomeration around *mess*, the valuation of the arc from $t1$ to *ack* is obtained by composition of the functions $\langle s - x \rangle$ and $\langle x \rangle$.

Methodology for obtaining high-level reductions

In this part, we give the approach one can use to define a new sound reduction for WNs starting from a reduction of Petri nets. This methodology includes two steps : the specification and the validation of a new reduction.

Specification of a high-level reduction

As for the ordinary reduction, one must specify application conditions and transformation rules. During this specification, the conditions must be decomposed in two kinds :

- Structural conditions as close as possible to the ones for the ordinary reduction.
- Functionnal conditions that must be the weakest possible in order to obtain a good structure for the unfolded Petri net corresponding to the WN.

The transformation rule must follow these two principles :

- It must not increase the size of the colour domain (since in this case there is a hidden extension of the net).
- It enables only significant operations for the colour functions (like for instance composition or inverse) in order to keep the new functions manageable.

Validation of a high-level reduction

Once defined the reduction, the proof that it is sound must be done in the following way :

Unfolding Show that in the unfolded net a set of ordinary reductions fulfill their conditions.

Reductions sequence Find an order of these reductions such that a reduction is still applicable once the previous ones have been applied.

Folding Show the ordinary reduced net may be folded in order to give the reduced WN given by the transformation rule.

Unlike the techniques presented so far, the last we present is not an extension of techniques existing for Place/Transition nets but it is completely specific to coloured nets.

7.3 Decolourization of WN systems

The presence of redundancies in the model description is quite natural in High Level Petri Nets, and derives from the methodology followed in the model construction. Once the basic elements that flow in the model are defined, two extreme choices may be considered *natural* in their representation: either each element has its own identity, or no distinction is made between elements (like in P/T nets). The best approach would probably be that of considering all elements undistinguishable at first, introducing distinctions only when it becomes actually necessary to more conveniently represent some part of the modelled system. However, since it is difficult in general to conceive a representation that is both correct and minimal, the possibility of automatically discovering some colour redundancies in a model allows the modeller to put less effort in achieving minimality, concentrating more on the correct representation of the system behaviour.

A first example of colour redundancy elimination was presented in Subsection ??; in this section the notion of redundant colour in WNs is formalized, and a WN model decolourization algorithm is presented. The contents of this section are a synthesis of the work on decolourization presented in [11, 4, 16]; some of the presented notions have been slightly modified and renamed with the aim of making them more easy to understand.

We need to introduce a few basic notions. A *component* of a (place/transition) colour domain is one of the colour classes occurring in the Cartesian product that defines the colour domain itself. For transitions, a colour component corresponds to one of the transition variables; for places, a colour component is identified by its position in the tuples associated with the tokens in that place. Observe that a place colour component can be uniquely identified as the h-th occurrence of a given class C_i (denoted C_i^h) in the place colour domain. Examples of colour components for the WN in Figure 7.9 are: the second occurrence of class E in $cd(p_3)$ and variable y of transition t_3.

A *colour flow* is a collection of colour components associated with a subset of places and transitions in the model: the components in a colour flow are

homogeneous, i.e., they are occurrences of the same colour class C_i, moreover they can be related by using the definition of the model arcs and associated functions, in a way that will be explained later: intuitively we can say that the components in a flow define a possible path that colours can follow within the net. For example, one colour flow in the WN of Figure 7.9 is that associated with the subnet comprising p_4, p_5, t_3 and t_4, composed of the following colour components: first (and unique) occurrence of B in $cd(p_4)$ and $cd(p_5)$, variable z of t_3 and t_4; the interpretation of this colour flow is the path followed by the colours representing the busses in the subnet representing a bus acquisition and release cycle.

Using these notions it is possible to partition the colour inscriptions of a WN model into a set of (*minimal*) colour flows; each colour flow is then *checked for redundancy*. The colour specification of the WN in Figure 7.9 can be partitioned into two colour flows: the first one has been described in the previous paragraph, the second one represents the path followed by the colours representing the nc computers and corresponding memories, and is composed of the first occurrence of E in $cd(p_1)$, $cd(p_2)$, $cd(p_6)$ and $cd(p_7)$, the first and second occurrence of E in $cd(p_3)$ and $cd(p_4)$, variable x of transitions t_1, t_2 and t_4, and variables x and y of transitions t_3 and t_4. Observe that in this colour flow it is possible to identify two subflows: one representing the computers and one representing the memories, however they need to be considered together as a single flow because the predicate $[x <> y]$ associated with transition t_2 creates a dependence between the two.

The redundancy of a colour flow can be defined at the level of the RG by introducing a proper notion of RG decolourization with respect to a *colour flow* and a *colour shrinking function*. The simpler type of *shrinking function* from colour class C_i to the neutral class $C_\bullet$ is a function that maps each element of C_i in the unique element in $C_\bullet$, i.e., a function that makes all the elements of a given class undistinguishable. In our running example we shall use such simple shrinking function in conjunction with the colour flow representing the busses colour flow. We may have more complex functions from C_i to a reduced class C_i' (containing the same number of static subclasses as C_i) that make undistinguishable only the elements of a given static subclass of C_i, leaving the other subclasses unchanged.

Observe that the colour components that constitute a given colour flow allow to select some elements in the token tuples of any place marking and some parameter values of any transition instance. The RG decolourization consists of substituting in each marking of the reachability set the selected token elements with the new colours obtained by applying the shrinking function to them, and similarly substituting in each arc of the RG the selected transition instance parameter values with the new values obtained by applying the shrinking function to them. Observe that this may make several nodes and arcs of the RG undistinguishable, hence causing a reduction in the RG size. For example, in Fig. 7.15(a) is depicted a portion of RG of the multicomputer PLC coloured model (for the case $nc = 3$ and $nb = 2$), while in Fig. 7.15(b) is shown its decolourization with respect to the busses colour flow and the simple shrinking

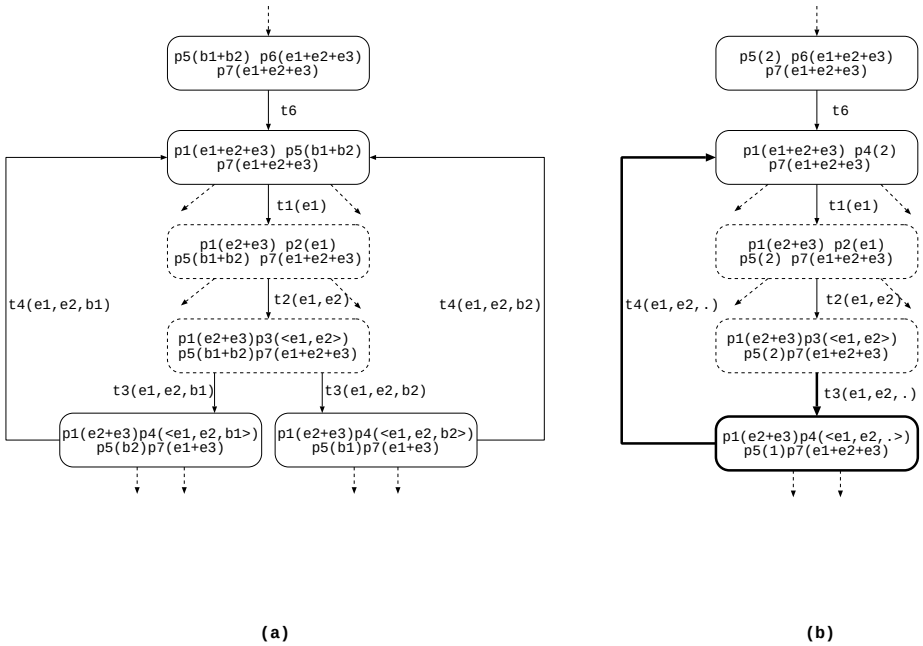


Figure 7.15: A portion of multicomputer PLC model RG (a) and its decolourization with respect to the busses colour flow (b).

function previously defined $sh : B \rightarrow C_\bullet$; the arcs and markings drawn with a thick line correspond to several arcs and markings in the original RG.

Based on the properties of the decolourized RG, it is possible to define different notions of redundancy: we consider as redundant those colour flows that allow to recognize in the decolourized RG a *lumping* of the original RG similar to that induced by the SRG algorithm, that is, all the markings that become undistinguishable after the decolourization have the same set of decolourized arcs, directed towards the same decolourized marking, or, stated in different terms, the possible transition firing sequences in the decolourized RG are the same as in the original RG. It is intuitively easy to see that the colour flow representing the busses satisfies this redundancy criteria.

Although the first definition of redundant colour flow is based on the RG, we are actually interested in recognizing and reducing the redundant colour flows at the level of the system structure. To this purpose we need to define the *projection* of a WN system onto a colour flow, and then some structural condition on the projected system to decide whether the colour flow is redundant; intuitively a colour flow is structurally redundant if its colour components are never used in any synchronization. For example the computers-memories colour flow of the multicomputer PLC example *is not* structurally redundant because the colour is used by transition t_3 in the synchronization of a token in p_3 and a token in p_7.

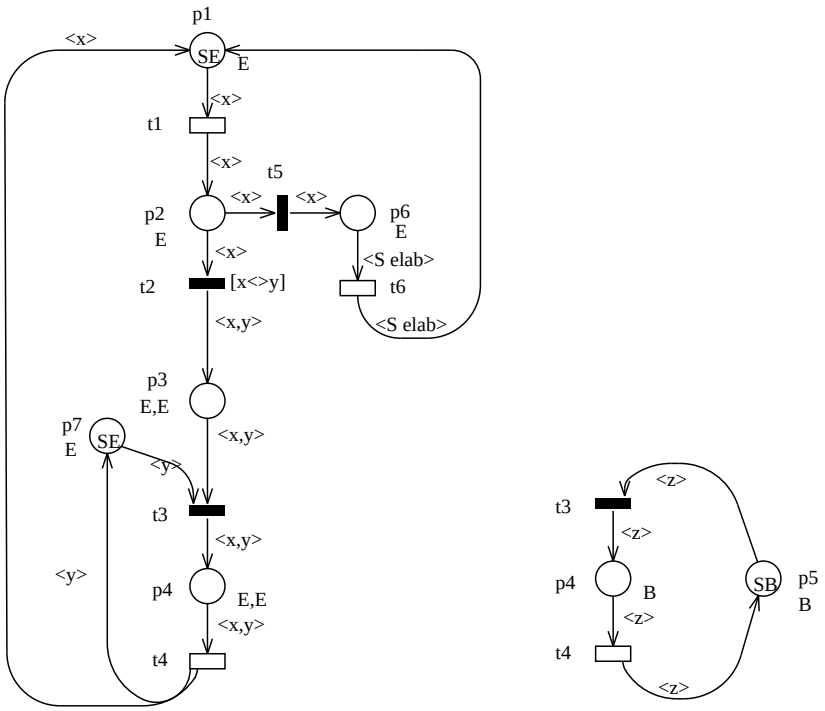


Figure 7.16: Projection of the multicomputer PLC WN model over its two colour flows

Finally we need a *WN system structural decolourization algorithm* to decolourize a WN system with respect to a structurally redundant colour flow. Let us point out that it might be the case that just a *portion* of a colour flow can be decolourized: we shall explain this particular case on an example suitable to this purpose.

In Fig. 7.16 is depicted the projection of the multicomputer PLC WN model on its two colour flows (computers-memories colour flow on the left, busses colour flow on the right).

As a further example, let us consider the random polling system of Figure 7.12. It contains two colour flows corresponding to the servers and the queues respectively. The first colour flow comprises the (unique) class R component of the colour domains of nodes p_p, p_r, p_s, p_w, t_s, T_s, t_w, t_r, and T_w; the second colour flow comprises all class Q components of the colour domain of all nodes in the net. The projection of the random polling WN model on its two colour components is shown in Figure 7.17.

Definition 7.7 (Place/Transition colour component) *The colour components of a place p are all the components of its colour domain $cd(p)$. We shall*

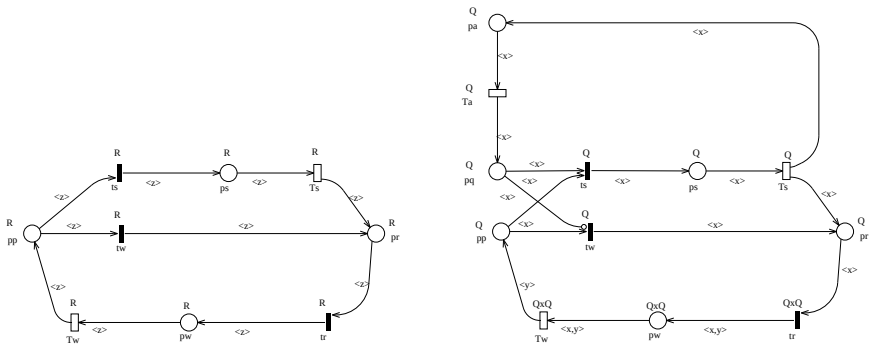


Figure 7.17: Projection of the random polling WN model over its two colour flows

denote $p \Downarrow_{C_i^j}$ *the colour component of* p *corresponding to the* j*-th occurrence of* C_i *in* $cd(p)$.

The colour components of a transition t *are all the components of its colour domain* $cd(t)$ *and are in a one-to-one correspondence with the transition variables. We shall denote* $t \Downarrow_x$ *the colour component of* t *corresponding to variable* $x \in var(t)$. *It is possible to extend the concept of colour component to the expressions associated with the input/output/inhibitor arcs of a given place* p, $(p \rightarrow t) \Downarrow_{C_i^j}$ $((p \leftarrow t) \Downarrow_{C_i^j}/\ (p \multimap t) \Downarrow_{C_i^j})$: *it corresponds, for each tuple in the formal sum defining the arc expression, to the element which is in the same position as* C_i^j *(the* j*-th occurrence of* C_i*) in* $cd(p)$.

Definition 7.8 (Colour flow) *A colour flow* $\mathcal{F}$ *consists of a set* $cf\_nodes(\mathcal{F})$ *of nodes and a set* $cf\_comp(\mathcal{F})$ *of colour components for each node in* $cf\_nodes(\mathcal{F})$ *(the colour components of the arc expressions belonging to the flow can be automatically derived from those of the corresponding places).*

A colour flow $\mathcal{F}$ *is well defined iff it satisfies the following properties:*

- *if* $p \in cf\_nodes(\mathcal{F})$ *then* ${}^\bullet p \cup p^\bullet \subset cf\_nodes(\mathcal{F})$;
- *if* $cf\_comp(\mathcal{F})$ *contains one or more colour components of place* p *then* $p \in cf\_nodes(\mathcal{F})$, *and vice versa;*
- *if* $cf\_comp(\mathcal{F})$ *contains one or more colour components (variables) of transition* t, *then* $t \in cf\_nodes(\mathcal{F})$ *(instead a transition* t *can belong to* $cf\_nodes(\mathcal{F})$ *even if there are not variables of* t *in* $cf\_comp(\mathcal{F})$*: this may happen when constant colour functions appear on the input/output/inhibitor arcs of* t*);*
- *if* $p \Downarrow_{C_i^j} \in cf\_comp(\mathcal{F})$ *and* $t \in p^\bullet$ *(or* $t \in {}^\bullet p$ *or* $t \in p^\circ$*) and* $x \in var(t)$ *appears in* $(p \rightarrow t) \Downarrow_{C_i^j}$ *(or* $(p \leftarrow t) \Downarrow_{C_i^j}$ *or* $(p \multimap t) \Downarrow_{C_i^j}$*), then* $t \Downarrow_x \in cf\_comp(\mathcal{F})$, *and vice versa;*

- *if* $t \Downarrow_x \in cf\_comp(\mathcal{F})$ *and a basic predicate of type* $[x = y]$, *or* $[d(x) = d(y)]$ *or* $[x = !y]$ *appears in* **guard**$[t]$, *then* $t \Downarrow_y \in cf\_comp(\mathcal{F})$.

A colour flow is minimal *if there is no well defined colour flow strictly included in it.*

Observe that all the colour components of a minimal colour flow are homogeneous, i.e. they all refer to the same basic class C_i.

Definition 7.9 (Colour shrinking function) *A colour shrinking function sh is a function from colour elements in a basic colour class* C_i *to colour elements in a shrunk colour class* C_i'*:*

$$sh \ : \ C_i \rightarrow C_i'$$

such that $|C_i'| \leq |C_i|$ *and* C_i' *has a number of static subclasses less than or equal to the one of* C_i *and*

$$\forall c_1, c_2 \in C_i, \quad d(c_1) = d(c_2) \quad \Rightarrow \quad d(sh(c_1)) = d(sh(c_2))$$

The function sh establishes a correspondence between a static subclass $C_{i,q}$ *in* C_i *and a static subclass* $C_{i,q}'$ *in the shrunk colour class* C_i' *(we also use* $sh(C_{i,q})$ *to denote the new static subclass corresponding to* $C_{i,q}$*). The shrunk static subclasses can be either of cardinality one or of the same cardinality of the originating static subclass* $C_{i,q}$*. The function must satisfy the property that each ordered class is either completely decolourized or not decolourized at all.*

A colour shrinking function *sh* is *compatible* with a minimal colour flow $\mathcal{F}$ if it is defined over the same basic class C_i common to all colour components in $\mathcal{F}$.

7.3.1 Redundancy check based on the system behaviour

Definition 7.10 (Decolourization of the RG**)** *The decolourization of a* RG *of a WN system with respect to a given colour flow* $\mathcal{F}$ *and a (compatible) shrinking function sh is defined through the function* $\mathcal{D}_{\mathcal{F},sh}$ *as follows:*

Marking decolourization *there is a one to one correspondence between the tokens in* $\mathbf{m}[p]$ *and those in the decoloured marking* $\mathcal{D}_{\mathcal{F},sh}(\mathbf{m})[p]$*. In particular for all places not contained in* $cf\_nodes(\mathcal{F})$, $\mathcal{D}_{\mathcal{F},sh}(\mathbf{m})[p] = \mathbf{m}[p]$, *while for all places contained in* $cf\_nodes(\mathcal{F})$, *each tuple in* $\mathbf{m}[p]$ *is substituted in* $\mathcal{D}_{\mathcal{F},sh}(\mathbf{m})$ *by a tuple in which the elements* c_i *corresponding to components not belonging to* $cf\_comp(\mathcal{F})$ *remain unchanged, while each element* c_i *corresponding to a component belonging to* $cf\_comp(\mathcal{F})$ *is substituted by* $sh(c_i)$.

The markings that become identical after the decolourization, are merged in the decolourized RG.

Transition firing instance decolourization *each assignment of colours to variables corresponding to a firing instance $\langle t, c\rangle$ of transition $t \in cf\_nodes(\mathcal{F})$ is modified so that in $\mathcal{D}_{\mathcal{F},sh}(\langle t, c\rangle)$ the colours assigned to variables that do not belong to $cf\_comp(\mathcal{F})$ remain the same as in $\langle t, c\rangle$ while the colours c_i assigned to variables that belong to $cf\_comp(\mathcal{F})$ are substituted with $sh(c_i)$. The transition firing instances that after the decolourization are identical and connect the same pair of decolourized markings are merged in a single decolourized transition instance.*

Definition 7.11 (Colour flow redundancy with respect to the RG**)** *A colour flow $\mathcal{F}$ of a WN system $\mathcal{S}$ is redundant with respect to the* RG$(\mathcal{S})$ *if a (compatible) shrinking function sh exists such that:*

$$\begin{aligned}
&\forall \mathbf{m}_i, \mathbf{m}_j \in \mathrm{RG}(\mathcal{S}) \ : \ \mathcal{D}_{\mathcal{F},sh}(\mathbf{m}_i) = \mathcal{D}_{\mathcal{F},sh}(\mathbf{m}_j) \\
&\quad \textit{if } \mathbf{m}_i \xrightarrow{\langle t,c\rangle} \mathbf{m}_k \textit{ then} \\
&\quad\quad \exists c' \in cd(t) \ : \ \mathbf{m}_j \xrightarrow{\langle t,c'\rangle} \mathbf{m}_l \\
&\quad\quad\quad\quad \wedge\ \mathcal{D}_{\mathcal{F},sh}(\langle t, c\rangle) = \mathcal{D}_{\mathcal{F},sh}(\langle t, c'\rangle) \\
&\quad\quad\quad\quad \wedge\ \mathcal{D}_{\mathcal{F},sh}(\mathbf{m}_k) = \mathcal{D}_{\mathcal{F},sh}(\mathbf{m}_l)
\end{aligned}$$

7.3.2 Redundancy check based on the WN structure

In this subsection structural redundant colour flows are formally defined and a structural reduction algorithm that modifies a WN model to eliminate the colour redundancy due to a given colour flow is defined.

Definition 7.12 (Projection of a WN model on a colour flow) *Given a WN net $\mathcal{N}$ and a colour flow $\mathcal{F}$, the projection of $\mathcal{N}$ over $\mathcal{F}$, $\mathcal{N}\downarrow_{\mathcal{F}}$ is defined as follows:*

$$\mathcal{N}\downarrow_{\mathcal{F}} = \langle P', T', \mathbf{Pre}', \mathbf{Post}', \mathbf{Inh}', \mathbf{pri}', \mathcal{C}, cd'\rangle$$

where

- $P' = \{p : p \in cf\_nodes(\mathcal{F})\}$
- $T' = \{t : t \in cf\_nodes(\mathcal{F})\}$
- *let*

$$\mathbf{Pre}[p,t] = \sum_k \delta_k \bigotimes_{C_i \in\ \mathcal{C}} \bigotimes_{j=1,..,e_i} \left\{ \sum_{q=1}^{n_i} \alpha_{i,q}^j . S_{C_{i,q}} + \sum_{x \in var_i(t)} \left(\beta_x^j . x + \gamma_x^j . !x\right) \right\}$$

be the arc expression associated with an arc from $p \in P'$ to $t \in T'$ in the original net, then

$$\mathbf{Pre}'[p,t] = \sum_k \delta_k \ \ \eta_k \bigotimes_{C_i \in\ \mathcal{C}} \bigotimes_{j : p \Downarrow_{C_i^j} \in cf\_comp(\mathcal{F})} \left\{ \sum_{q=1}^{n_i} \alpha_{i,q}^j . S_{C_{i,q}} + \sum_{x \in var_i(t)} \left(\beta_x^j . x + \gamma_x^j . !x\right) \right\}$$

where

$$\eta_k = \prod_{C_i \in \mathcal{C}} \prod_{j:p\Downarrow_{C_i^j} \notin cf\_comp(\mathcal{F})} \left| \sum_{q=1}^{n_i} \alpha_{i,q}^j . S_{C_{i,q}} + \sum_{x \in var_i(t)} \left(\beta_x^j . x + \gamma_x^j . !x\right) \right|$$

$$= \prod_{C_i \in \mathcal{C}} \prod_{j:p\Downarrow_{C_i^j} \notin cf\_comp(\mathcal{F})} \left(\sum_{q=1}^{n_i} \alpha_{i,q}^j |C_{i,q}| + \sum_{x \in var_i(t)} \left(\beta_x^j + \gamma_x^j\right)\right)$$

- $\mathbf{Post}'$ *and* $\mathbf{Inh}'$ *are derived from* **Post** *and* **Inh** *in the same way* $\mathbf{Pre}'$ *is derived from* **Pre**.
- *place and transition colour domains are defined as follows:*

$$\forall p \in P', cd'(p) = \bigotimes_{C_i^j : p\Downarrow_{C_i^j} \in cf\_comp(\mathcal{F})} C_i$$

$$\forall t \in T', var(t) = \{x : t \Downarrow_x \in cf\_comp(\mathcal{F})\}$$

, moreover the guard associated with transition t *is obtained from* $\mathbf{guard}[t]$ *in the original net by substituting each atomic predicate involving only variables in* $cf\_comp(\mathcal{F})$ *with a symbol* Φ_i *acting as a place holder in the boolean expression defining the guard.*

Let us point out that we are assuming that the model to be decolourized does not have any arc expression including predicates.

Based on the definition of model projection over a colour flow, a set of structural redundancy conditions shall now be defined: they check whether a given colour flow is structurally redundant and allow to derive a compatible colour shrinking function for a structurally redundant colour flow.

The colour shrinking function is constructed by performing the redundancy check on each static subclass $C_{i,q}$ of the colour flow class C_i independently: the function completely decolourizes the static subclasses that satisfy the redundancy conditions while it leaves the static subclasses that do not satisfy the conditions unchanged. We assume that the partition of each basic class into static subclasses is minimal, that is there are no more static subclasses than those strictly needed to properly define the transition enabling conditions.

There are cases in which the structural redundancy conditions are not fulfilled, but the colour flow can still be considered as redundant using some further information that can be derived from the model P-semiflows and by constraining the set of allowed initial markings.

Definition 7.13 (Colour flow structural redundancy) *Redundancy check of* $\mathcal{F}$ *with respect to static subclass* $C_{i,q}$. *For each transition* t *of* $\mathcal{N}' = N \downarrow_{\mathcal{F}}$ *the following properties must be satisfied:*

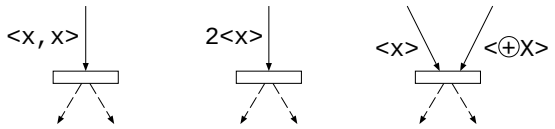


Figure 7.18: Examples of situations that prevent decolourization

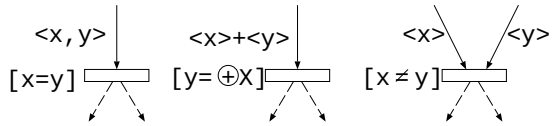


Figure 7.19: Other examples of situations that prevent decolourization

1. *the projected net does not contain inhibitor arcs* [1]*;*

2. *the set of functions that label the input arcs of* t *can only be functions of type* x, $!x$ *and linear combinations of the two. This condition can be relaxed by allowing synchronization functions (*S_i *or* $S_{i,q}$*) if the class is not ordered and the place connected to the arc is colour-safe*[2]*.*

3. *Let* x *be a variable that can be assigned an element in* $C_{i,q}$ *(i.e. the transition guard does not imply that* $d(x) \neq C_{i,q}$*). No more than one occurrence of either* x *or* $!x$ *is allowed in the labelling functions of all input arcs of* t*. Examples of forbidden situations are depicted in Figure 7.18.*

4. *If two or more variables* x, y, z, *etc. that can be assigned elements in the same static subclass* $C_{i,q}$ *occur in the input arc expressions of* t*, then the transition guard* $\mathbf{guard}[t]$ *must satisfy any one of the following constraints:*

   - *it is either a tautology (this comprises the particular case in which the predicate is omitted with the meaning that it is true);*
   - *or every predicate* ϕ'_t *such that* $\mathbf{guard}[t] \rightarrow \phi'_t$ *and composed exclusively of standard predicates of equality type over* x, y, z, *etc., is always true.*

   Examples of forbidden situations are depicted in Figure 7.19.

If these properties are satisfied, then either the colour components under study are never used in a synchronization (due to the conditions on the functions combination on input arcs), or the synchronizations can be solved just by counting the number of tokens in the place (due to the colour-safeness property of the place). Hence the static subclass $C_{i,q}$ can be decolourized by a function

[1] This condition could be relaxed in case the corresponding arc function is $< S_{Ci} >$ because this corresponds to check that there are no tokens, which can still be implemented after decolourization.

[2] A place is colour-safe iff it never contains two copies of the same tuple in any marking.

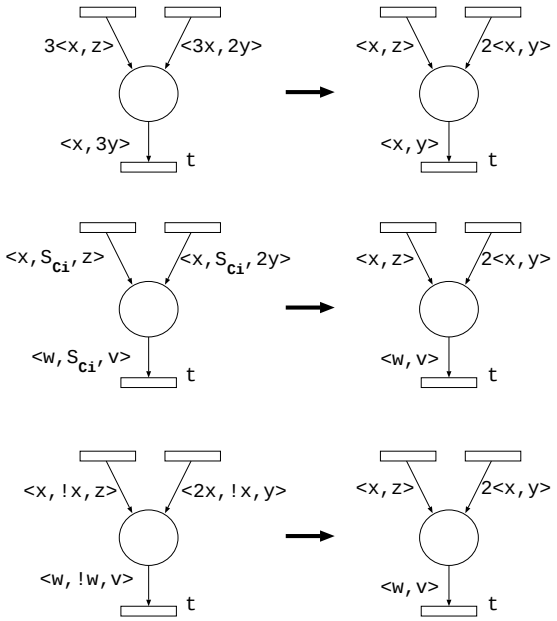


Figure 7.20: Some examples of arc expressions simplification

sh that maps each element $c \in C_{i,q}$ onto the same element $c'_{i,q}$, the unique element of the shrunk subclass $C'_{i,q}$. Otherwise $C'_{i,q}$ is set equal to $C_{i,q}$ and the shrinking function *sh* is the identity for the elements in $C_{i,q}$.

Observe that sometimes the redundancy check may fail because of the presence of *fictitious* synchronizations in the net definition that may be removed by applying *simplification* rules to the net before considering the redundancy check. In Fig. 7.20 three examples of simplification are shown where an apparent synchronization on transition t is removed from the net (see [5] for further details on nets simplification). Similar simplifications could apply also at the level of the transition guards, for example a transition guard like $(d(x) <> d(y))and(x <> y)and....$ could be simplified by eliminating the basic predicate $(x <> y)$ without affecting its truth value (because $(d(x) <> d(y)) \Rightarrow (x <> y)$).

As an example of decolourized net, in Fig. 7.21 the multicomputer PLC WN model is depicted after the decolourization of the busses; observe that since in this case the colour class B is completely decolourized, then the corresponding colour components have been deleted from the model.

Similarly, in the random polling WN model the colour flow corresponding to servers can be completely decolourized.

Property 7.14 (Structural versus RG redundancy) *Structural redundancy implies redundancy with respect to the* RG.

When the decolourization conditions are satisfied for some of the static subclasses of a class C_i the net $\mathcal{N}$ can be transformed using the following procedure.

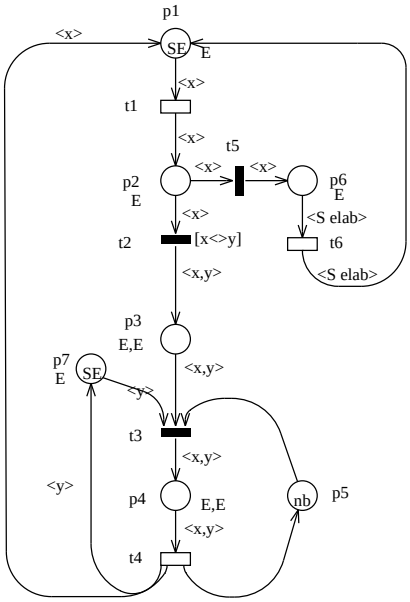


Figure 7.21: WN model of the multicomputer PLC after the decolourization of the busses colour flow.

The decolourized net is denoted $\mathcal{D}_{\mathcal{F},sh}(\mathcal{N})$.

Definition 7.15 (WN model structural decolourization) *with respect to a minimal colour flow $\mathcal{F}$ and a shrinking function sh:*

- *The newly defined, shrunk colour class C_i' is added to the set of colour classes.*
- *For each place and transition colour domain, the colour components in $cf\_comp(\mathcal{F})$ are substituted by the new class C_i'.*
- *The arc expression components selected by $cf\_comp(\mathcal{F})$ are transformed as follows:*

  x remains unchanged

  $S_{i,q}$ is substituted by $\frac{|C_{i,q}|}{|C'_{i,q}|}S_{i,q}$

  S_i is substituted by $\sum_q \frac{|C_{i,q}|}{|C'_{i,q}|}S_{i,q}$
- *Transition guards that refer to completely decolourized classes are substituted by "true." Predicates referring to classes that are only partially decolourized remain unchanged, unless the predicate is of the form $x \neq y$ (that relate functions on input or inhibitor arcs —say x— to functions on output arcs, otherwise the decolourization would not be allowed). In that case it should be transformed taking into account the cardinality of the static subclasses in the original system. If all the static subclasses in the original net have cardinality greater than one then the predicate is changed to true otherwise it becomes*[3]

$$\left(\bigvee_{\forall q:|C_{i,q}|>1 \ \wedge \ C_{i,q} \text{ has been shrinked}} [d(x) = q] \right) \quad \vee \quad [x \neq y] \tag{7.2}$$

Since the predicate in Equation (7.2) depends on the cardinality of the original static subclasses $C_{i,q}$, it is not completely parametric. The possible solution to this problem could be to consider for each static subclass that is involved in the predicate the two cases $|C_{i,q}| = 1$ and $|C_{i,q}| > 1$ and build the resulting reduced nets for all the possible cases. When the net is instantiated, it is possible to choose the correct reduced net among those generated by the reduction algorithm.

We have already mentioned that when a class is completely decolourized (i.e. $|C_i'| = 1$) the corresponding components can be "removed" from the net. When performing this cancellation however, the coefficients of the eliminated function

[3] The reason for using the above predicate is that different objects may belong to the same static subclass provided that it has a cardinality greater than one. When a static subclass of cardinality greater than one is shrunk, different colour elements in the original class could become equal in the decolourized class.

component must be taken into account: the cardinality of each eliminated function component becomes a factor multiplying the function tuple. For example consider the following function tuple:

$$\langle f_{j_1}, \ldots, f_{j_{k-1}}, S_i - X_i, f_{j_{k+1}}, \ldots, f_{j_m} \rangle$$

after the complete decolourization of C_i the above tuple is transformed into:

$$\langle f_{j_1}, \ldots, f_{j_{k-1}}, |C_i| S_i - x, f_{j_{k+1}}, \ldots, f_{j_m} \rangle$$

(the "$|C_i|$" coefficient in front of function S_i derives from the decolourization rule for functions described previously). The component "$|C_i|S_i - x$" can be eliminated by multiplying the tuple by $|C_i| - 1$, that is the sum of the basic functions coefficients of the component. Indeed this gives the cardinality of the function since after the complete decolourization of the class both S and X return a single element. The resulting function tuple of reduced arity is:

$$(|C_i| - 1)\langle f_{j_1}, \ldots, f_{j_{k-1}}, f_{j_{k+1}}, \ldots, f_{j_m} \rangle$$

7.3.3 Partial decolourization and other related issues

There are cases in which a colour flow is not structurally decolourizable as a whole, but there are some parts of it that instead can be decolourized locally. The intuition behind this type of local decolourization can be illustrated as follows: the general structure of the net obtained by projection of a WN model on a colour flow is depicted in Fig. 7.22; observe that it is possible to identify some *cyclic subflows*, some *colour generation subflows* and some *colour absorption subflows*.

The colour generation subflows can be decoloured (in the sense that the colour components are cancelled from the colour domains of the places and transition in the subflow) in such a way that the colour generation is delayed as much as possible (up to the point where it is needed for synchronization purposes). The colour absorption subflows can be decoloured in such a way that the colour cancellation is anticipated as much as possible.

These types of reduction have been formalized by defining two reduction rules (first introduced in [16]) that decolourize colour generation/absorption subflows one place at a time starting from the first/last place in the subflow.

Definition 7.16 (Elimination of a colour if not immediately used) *This reduction rule applies when a new colour is "created" by transition t but it is not immediately used in a synchronization. The creation of the new colour can therefore be postponed.*

This reduction is illustrated in Fig. 7.23: the colour to be eliminated corresponds to variable x. This variable is "created" by t_1, since it appears only once in the function associated with the arc from t_1 to p, and it does not appear in the predicate of t_1 (this last constraint could be relaxed in case it is possible to move the terms of the predicate involving t_1 on the output transitions of p: we do not give a general rule for this transformation here).

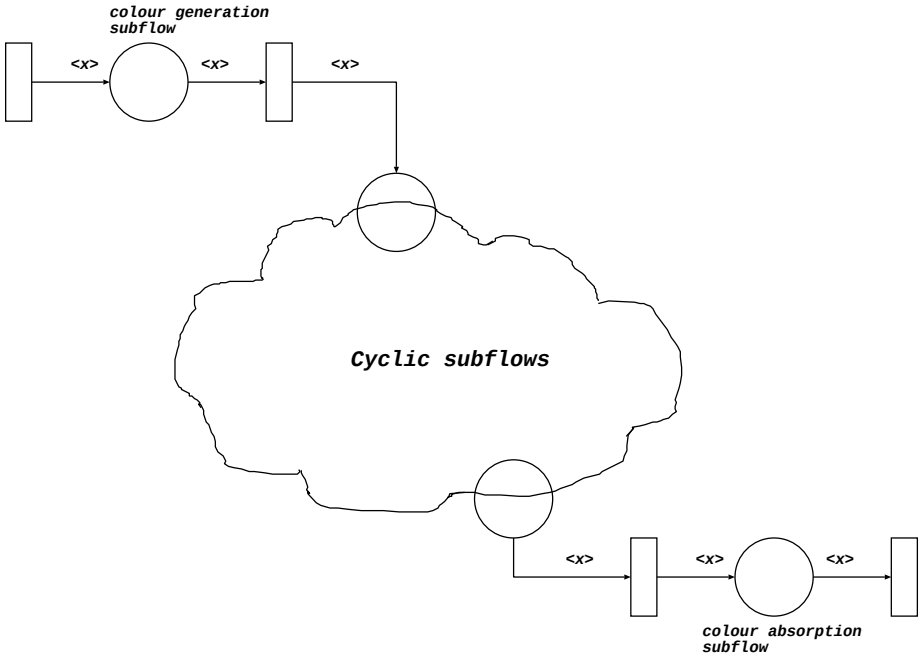


Figure 7.22: Structure of a projection over a colour flow: cyclic, colour generation and colour absorption subflows.

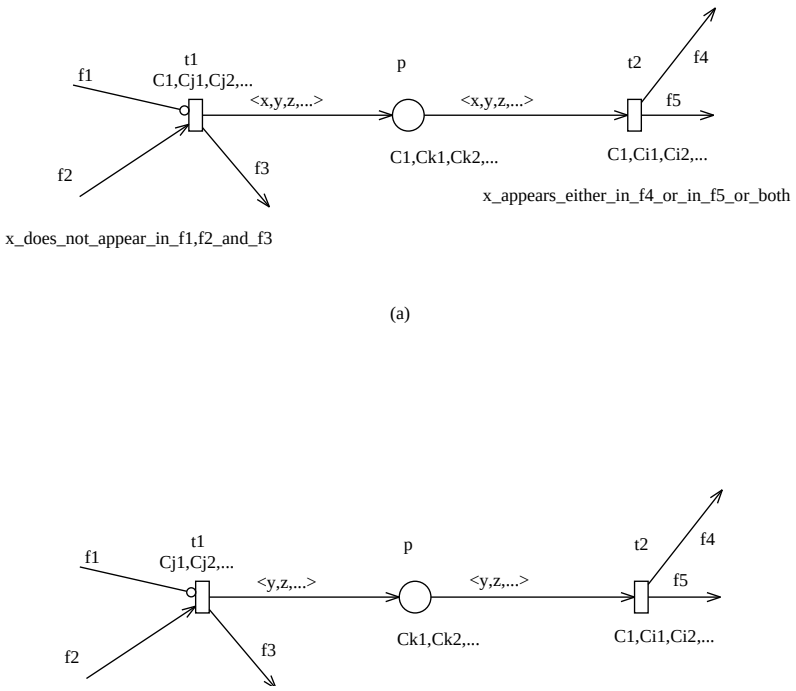


Figure 7.23: The effect of eliminating a colour not immediately used after its generation.

We are assuming (without loss of generality) that x corresponds to the first component in the colour domain of p, that x appears only once in the input and output functions of p, and that this colour is not used by transition t_2 in a synchronization: indeed t_2 has only one input arc from p, with an associated function that is simple[4] *and contains only one occurrence of x. This same variable appears in at least one output arc function of t_2.*

The reduction removes the colour from the colour domain of t_1 and of p; as a consequence, variable x disappears from the input and output arc functions of p. The effect of this reduction is to "merge" all the colour instances of t_1 with the same value of the variable x, and "merge" all the colour instances of p with the same first component.

On the contrary, by merging the instances of p, we may increase the number of (possibly) enabled instances of t_2 (x is no longer constrained by the marking of p).

[4] We call "simple" the arc expressions composed of only one term with coefficient 1 and whose elements are all projection functions.

Definition 7.17 (Elimination of a colour "swallowed" by a transition) *This reduction rule applies when a colour is "destroyed" (or swallowed) by a transition t, but it could have been already eliminated before by the firing that put tokens into the input place of t. The reduction anticipates the colour elimination of one step. The reduction is illustrated in Fig. 7.24: the colour to be eliminated corresponds to variable x.*

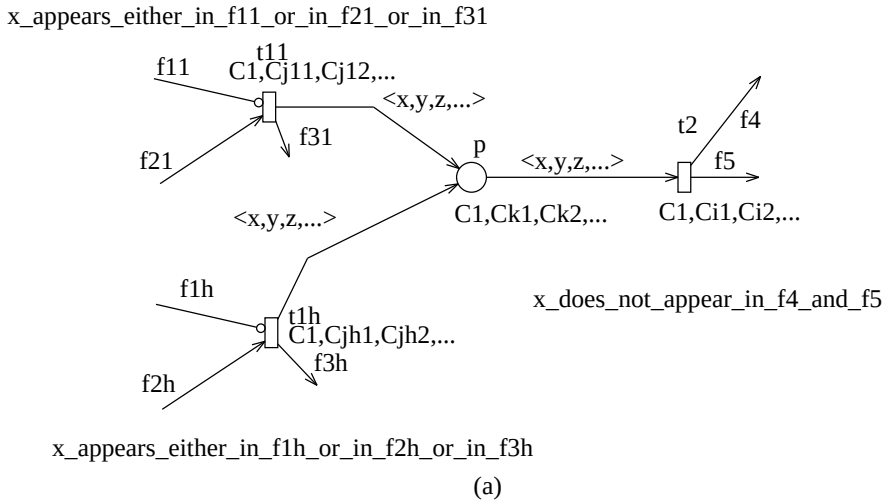
We are again assuming (without loss of generality) that x corresponds to the first component in the colour domain of p (the input and output arc functions of p are simple functions, and x appears only once in all of them). The variable x is "swallowed" by t_2 (it doesn't appear in any output arc function), however, since it is not used by t_2 (the only output transition of p) in a synchronization, it can be removed from the colour domain of t_2; as a consequence, the corresponding colour component of p should be removed as well. After the reduction, x may be "swallowed" by some input transition of p (all those transitions that do not have variable x in any output arc function). In Fig. 7.24 we consider a case in which x does not disappear from the colour domain of the input transitions $(t_{11}, \ldots, t_{1h})$ of p. Note that transitions $t_{11}, \ldots, t_{1h}$ do not have necessarily the same number of input, output and inhibition places as the picture may suggest.

The effect of this reduction is to merge all the instances of p with equal first component, and all the instances of t_2 with the same value of x.

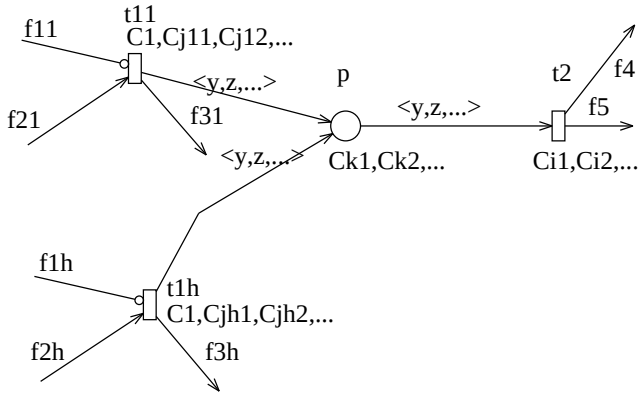
The transitions $t_{11}, \ldots, t_{1h}$ are not affected by the reduction (their colour domains and enabling conditions remain unchanged).

Example 7.1 (Decolourization of the multicomputer PLC model) In the multicomputer PLC example it is possible to apply the decolourization rules in the case $nb = 1$. Observe that the computers-memories colour flow contains a colour generation subflow corresponding to the generation of the identity of the (remote) memory that a given computer wants to access, the transition generating the colour is t_2. Observe that the second occurrence of E in p_3 cannot be decolourized because it is used in a synchronization over t_3, however, in the particular case of $nb = 1$, this is just a fictitious synchronization since when t_3 is enabled, all memories are present in p_7, hence the decision of the memory to be accessed could be delayed to the firing of t_3 so that the colour component $p_3 \Downarrow_{E^2}$ can be eliminated from $cd(p_3)$, and as a consequence colour component $t_2 \Downarrow_y$ can be cancelled from $cd(t_2)$. Observe that in this case the guard of t_2 must be moved on to t_3.

In the situation $nb = 1$ just considered, place p_7 happens to be an implicit place [7] so that it can be removed from the model without affecting its behaviour. After this model reduction, the subflow representing the path followed by the identity of the memory in a remote access is both a colour generation and absorption subflow that doesn't synchronize with any other subflow: hence it can be successively decoloured using either one of the above partial decolourization rules until it is completely cancelled (this last example shows the mechanism of partial decolourization although it is not very significant due to the fact that after p_7 is removed the whole colour flow becomes completely decolourizable).



(a)



(b)

Figure 7.24: The effect of eliminating a colour "swallowed" by a transition.

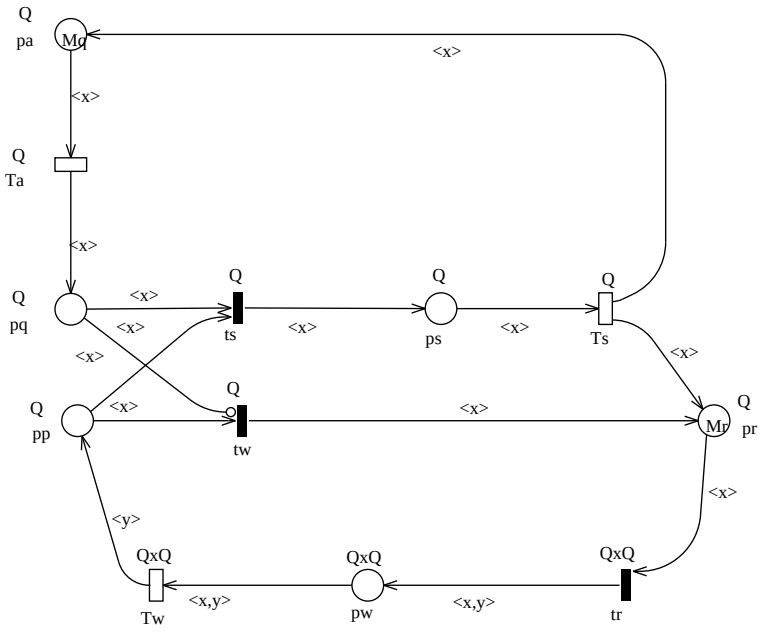


Figure 7.25: Decolourization of the servers colour flow in the random polling WN model

Example 7.2 (Decolourization of the random polling model) The application of the decolourization rules to the random polling model example presented in Section 7.1.5, allows to fully exploit the possibilities of this reduction technique. In [16] it has been shown that under a number of hypothesis it is possible to transform the random polling system SWN model into a GSPN model[5] from which it is possible to verify the same qualitative and quantitative properties with much less computational effort.

The random polling model in Fig. 7.12 contains two colour flows: the colour flow of the servers, whose nodes are p_p, p_s, p_r, p_w, t_s, T_s, t_w, T_w, t_r, and for each node, colour component R^1, and the colour flow of the queues that comprises all the nodes in the net, and for each node, all colour components corresponding to occurrences of colour class Q in the colour domain.

It is easy to verify that the servers colour flow satisfies the structural re-

[5] SWN and GSPN are extensions of the WN and PN formalisms, to include information on the durations of the modelled activities, with the aim of deriving performance measures form the model. Both formalisms will be defined later in this book.

N	Fig. 7.25	Fig. 7.26	Fig. 7.27
2	31, 31	12, 18	6, 13
	(57, 62)	(21, 34)	(9, 24)
3	91, 69	21, 30	9, 20
	(474, 384)	(90, 141)	(26, 80)
4	161, 107	30, 42	12, 27
	(2712, 1744)	(312, 476)	(72, 240)
5	231, 145	39, 54	15, 34
	(12480, 6640)	(960, 1435)	(192, 672)
6	301, 183	48, 66	18, 41
	(49776, 22560)	(2736, 4027)	(496, 1792)
7	371, 221	57, 78	21, 48
	(179424, 70805)	(7932, 10745)	(1248, 4608)
8	441, 259	66, 90	24, 55
	(599808, 209436)	(19200, 27640)	(3072, 11520)
9	511, 297	75, 102	27, 62
	$(1.9{\cdot}10^6, 0.6{\cdot}10^6)$	(46384, 69111)	(7424, 28160)
10	581, 335	84, 114	30, 69
	$(5.7{\cdot}10^6, 1.6{\cdot}10^6)$	(119040, 168950)	(17664, 67584)
20	–	174, 234	60, 139
	–	$(479{\cdot}10^6, 660{\cdot}10^6)$	$(61{\cdot}10^6, 242{\cdot}10^6)$
30	–	264, 354	90, 209
	–	$(1.1{\cdot}10^9, 1.5{\cdot}10^9)$	$(133{\cdot}10^9, 532{\cdot}10^9)$
100	–	894, 1194	300, 699
	–	(–)	(–)

Table 7.3: Symbolic and ordinary state space cardinality (tangible/vanishing) for WN models of a random polling system with two servers and queue capacities equal to one. Numbers in parenthesis refer to ordinary markings.

dundancy check of Definition 7.13. Since colour class R contains just one static subclass, the corresponding shrinking function reduces class R to a single element so that the colour flow can be completely removed from the net. The resulting decoloured net is shown in Fig. 7.25.

The first column of Table 7.3 shows the number of symbolic and ordinary markings (tangible/vanishing) of the model in Fig. 7.25 as a function of the number of queues and assuming a fixed number of servers $S = 2$. Observe that the number of symbolic markings of the decoloured model and of the initial WN model is the same (of course the number of ordinary markings is instead larger in the model where server identities are preserved), however the computation of the SRG of the model that keeps the server identities is more space and time consuming.

The next decolourization step consists of applying the reduction rule of Definition 7.17 twice, the first time to the subnet t_r, p_w, T_w (the colour component corresponding to variable x is "swallowed" by T_w: this subnet matches the schema of Figure 7.24), the second time to the subnet T_s, t_w, p_r, t_r. Intuitively,

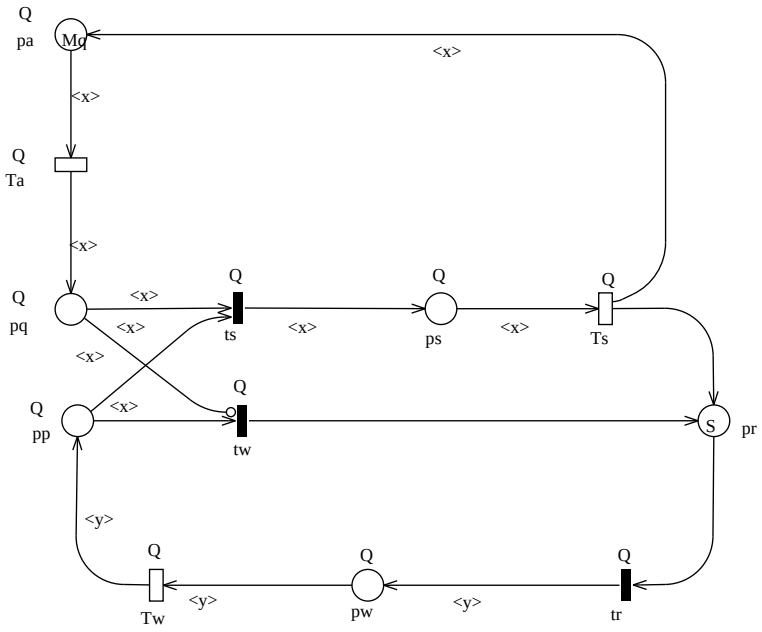


Figure 7.26: Partial decolourization of the queues colour flow in the random polling WN model (case of a colour component "swallowed" by a transition)

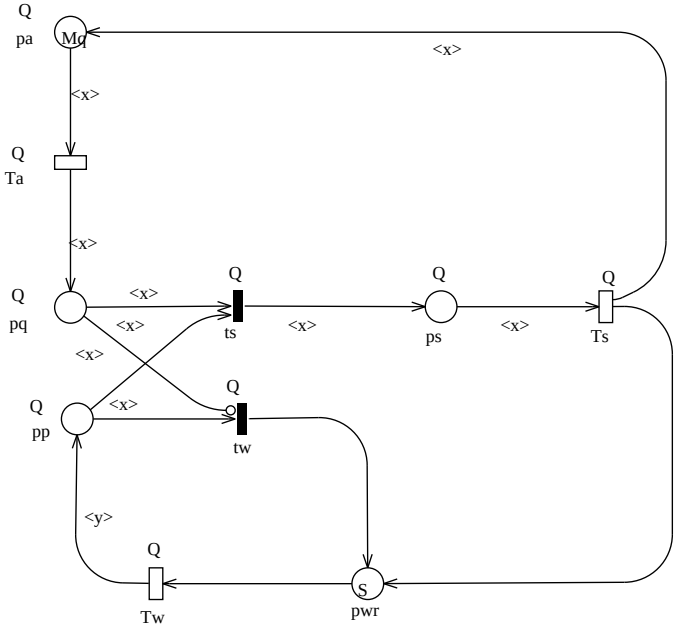


Figure 7.27: Partial decolourization of the queues colour flow in the random polling WN model (case of a colour component generated by a transition and not immediately used)

the effect of this reduction rule is to "forget" the identity of the last queue visited by a server as soon as it leaves the queue: this doesn't cause any loss of information because we are assuming that neither the choice of the next queue to be visited, nor the walk time depend on the previously visited queue. The resulting WN is shown in Figure 7.26, observe that the initial marking of place p_r has been decoloured as well, in fact it now consists of S uncoloured tokens; the number of symbolic and ordinary markings of the new model are shown in the second column of Table 7.3.

The last partial decolourization step, consists of applying the reduction rule of Definition 7.16 to the subnet t_r, p_w, T_w ((the colour component corresponding to variable y is generated by t_r but not immediately used in any synchronization: this subnet matches the schema of Figure 7.23). Intuitively, this corresponds to delaying the decision of which queue a server is directed to, only at the end of the walk. After having applied this last decolourization step, immediate transition t_r becomes redundant and the subnet p_r, t_r, p_w can be substituted by a single place, that we call p_{wr} (this simplification is a special case of "immediate transitions reduction" that will be briefly discussed in a later chapter; immediate transitions reduction rules can be found in [2], [18], and [16]). The resulting reduced WN is shown in Figure 7.27; the number of symbolic and ordinary markings of the last model are shown in the last column of Table 7.3.

Bibliography

[1] G. Berthelot. Transformations and decompositions of nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 359–376. Springer Verlag, Bad Honnef, West Germany, February 1987.

[2] G. Chiola, S. Donatelli, and G. Franceschinis. GSPN versus SPN: what is the actual role of immediate transitions ? In *Proc. 4^{th} Int. Workshop on Petri Nets and Performance Models*, Melbourne, Australia, December 1991.

[3] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, 1993.

[4] G. Chiola and G. Franceschinis. Structural colour simplification in Well-Formed coloured nets. In *Proc. 4^{th} Int. Workshop on Petri Nets and Performance Models*, Melbourne, Australia, December 1991.

[5] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. Markovian Analysis of symmetric CGSPNs: Symmetry Exploitation in Stochastic Well-Formed Colored Nets. Technical report, D W2.T2.13.v1 of the ESPRIT BRA QMIPS (Quantitative Modeling In Parallel Systems) Project, September 1994.

[6] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation, special issue on Performance Modeling Tools*, 24(1&2):47–68, November 1995.

[7] J.M. Colom. *Analisis estructural de Redes de Petri, programacion lineal y geometria convexa.* PhD thesis, Departamento de Ingenieria Electrica e Informatica, Universidad de Zaragoza, Espana, 1989.

[8] J.M. Colom, J. Martinez, and M. Silva. Packages for validating discrete production systems modeled with Petri nets. In P. Borne and S. Tzafestas, editors, *Applied Modelling and Simulation of Technological Systems*, pages 529–536. Elsevier Science Publ. (North Holland), 1987.

[9] J.M. Couvreur. The general computation of flows for coloured nets. In *Proc.* 11^{th} *Intern. Conference on Application and Theory of Petri Nets*, Paris, France, June 1990.

[10] C. Dutheillet. *Symétries dans les réseaux colorés : définition, analyse et application à l'évaluation de performance*. PhD thesis, Université Paris 6, 1991.

[11] G. Franceschinis. *Modellazione e Valutazione delle Prestazioni di Sistemi Concorrenti: le Reti di Petri Stocastiche Ben-Formate e loro Tecniche di Analisi*. PhD thesis, Università di Torino, Dipartimento di Informatica, 1993.

[12] H.J. Genrich. Equivalence transformations of Pr/T nets. In *Proc.* 9^{th} *Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[13] S. Haddad. Generalization of reduction theory to coloured nets. In *Proc.* 9^{th} *Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[14] S. Haddad and J.M. Couvreur. Towards a general and powerful computation of flows for parametrized coloured nets. In *Proc.* 9^{th} *Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[15] S. Haddad and C. Girault. Algebraic structure of flows of a regular coloured net. In *Proc.* 7^{th} *European Workshop on Application and Theory of Petri Nets*, Oxford, England, June 1986.

[16] M. Ajmone Marsan, S. Donatelli, G. Franceschinis, and F. Neri. Reductions in Generalized Stochastic Petri Nets and Stochastic Well-formed Nets: An Overview and an Example of Application. In J. Walrand, K. Bagchi, and G. Zobrist, editors, *The State-of-the-art in Performance Modeling and Simulation: Network Theory, Tools and Tutorials*, volume accepted for publication. Gordon and Breach Publishers INC, 1996.

[17] G. Memmi and J. Vautherin. Advanced algebraic techniques. In W. Brawer, W. Reisig, and G. Rozenberg, editors, *Advances on Petri Nets '86 - Part I*, volume 254 of *LNCS*. Springer Verlag, Bad Honnef, West Germany, February 1987.

[18] C. Simone and M. Ajmone Marsan. The application of EB-Equivalence rules to the structural reduction of GSPN models. *Journal of Parallel and Distributed Computing*, 15(3):171–187, 1992.

Chapter 8

Performance Measures and Basic Properties

A goal of performance modelling with timed Petri nets (TPN's) is the estimation of some quantifiable performance measures of the system under consideration by the simulation or analysis of the model of the system behaviour. In order to do that, *responsiveness* and *utilization* performance measures of the system must be described in terms of average values of operational quantities defined on the TPN model, like the *average marking* of a place or the *firing frequency* of a transition.

In Section 8.1, the usual average performance indices for TPN models are derived operationally from the basic observable events that were defined in Chapter 3.

Sections 8.2 and 8.3 are devoted to exploit the operational approach for the definition of performance measures. In Section 8.2, *operational analysis* techniques are used to partially characterize the behaviour of TPN models. Classical operational laws, like *Little's law* are stated in Petri net terms. Additionally, some *operational inequalities* are derived that are typical of the presence of synchronization and that have not been considered before in the framework of queueing networks models.

Section 8.3 illustrates another aspect of the deep gap, from the analytical point of view, existing between classical monoclass product-form queueing networks and TPN's. While in *Gordon-Newell* networks the vector of *visit ratios* to the stations can be easily (efficiently) computed from the routing information, in the case of bounded TPN's the computability of such index is a quite complex problem. This problem leads to a classification of net models attending to the dependency of the visit ratios on the net structure and the probabilistic routing, on the initial marking, and on the firing time of transitions.

Since few operational analysis techniques can presently be used to compute the performance indices of interest, the classical approach based on *stochastic processes* must be considered. Section 8.4 introduces also the definition of

performance parameters on the TPN model as expected values of the random variables that characterize the behaviour of the model, under some stochastic assumptions. As the reader surely knows, under *ergodicity* conditions the average (operationally defined) performance indices are equal to the expected (stochastically defined) indices, thus the techniques that will be presented later on for the computation of expected values will be useful to compute the average performance measures of interest.

Some bibliographic remarks are summarized in Section 8.5.

8.1 Performance Measures Defined Operationally

The basic idea for the operational definition of average performance indices is to derive them algebraically from the atomic events observed in the behaviour of the TPN model. For instance, from the instantaneous marking in place p at time τ, $\boldsymbol{\mu}[p](\tau)$, the *average marking* in place p during the experiment interval $(0, \theta)$ is defined as[1]:

$$\overline{\boldsymbol{\mu}}[p] = \frac{1}{\theta} \int_0^\theta \boldsymbol{\mu}[p](\tau)\, d\tau$$

Let us assume in the rest of this chapter an *infinite server semantics* for the timed transitions. The *instantaneous enabling degree* of transition t at instant τ represents the internal concurrency of that transition at time τ:

$$\mathbf{e}[t](\tau) = \sup\{k \in \mathbb{N} : \forall p \in {}^\bullet t, \boldsymbol{\mu}[p](\tau) \geq k\ \mathbf{Pre}[p,t]\}$$

The following relation holds by definitions:

$$\mathbf{e}[t](\tau) = \min_{p \in {}^\bullet t} \left\lfloor \frac{\boldsymbol{\mu}[p](\tau)}{\mathbf{Pre}[p,t]} \right\rfloor, \quad \forall t \in T, \forall \tau \tag{8.1}$$

The *average enabling degree* of transition t during the interval $(0, \theta)$ is defined as:

$$\overline{\mathbf{e}}[t] = \frac{1}{\theta} \int_0^\theta \mathbf{e}[t](\tau)\, d\tau$$

The average enabling degree represents the average number of active servers associated with the transition firing during the experiment.

In cases where transition t is persistent (i.e., never disabled before firing) or t has *age memory* policy, the *average firing time* of transition t, $\overline{\mathbf{s}}[t]$, can be expressed as the quotient between the *total enabling work* of that transition and the total number of firings, $F_t(\theta)$, of t observed from 0 to θ (assuming

[1] The reader should notice that we write $\overline{\boldsymbol{\mu}}[p]$ for short, while for being more precise we should write $\overline{\boldsymbol{\mu}}[p](\theta)$, since the average is computed between 0 and θ.

$F_t(\theta) > 0$). The total enabling work of transition t is obtained by integration over the experiment interval of the instantaneous enabling degree of t at instant τ, $\mathbf{e}[t](\tau)$.

$$\overline{\mathbf{s}}[t] = \frac{\int_0^\theta \mathbf{e}[t](\tau)\, d\tau}{F_t(\theta)}$$

The *throughput* of transition t is the firing frequency of that transition observed during the interval $(0, \theta)$:

$$\boldsymbol{\chi}[t] = \frac{F_t(\theta)}{\theta}$$

With queueing systems terminology, the *relative throughput* between transitions is expressed by the vector of *visit ratios*, normalized, for instance, for transition t_1:

$$\mathbf{v}[t] = \frac{\boldsymbol{\chi}[t]}{\boldsymbol{\chi}[t_1]}$$

The quantity $\mathbf{v}[t]$ represents the number of times that transition t is fired per each firing of trasition t_1 (in queueing systems terminology, it represents the number of "visits" of a customer to station t per each visit to the reference station t_1). A detailed analysis of the computability of visit ratios for different net classes is presented in Section 8.3.

If $\alpha(p, i)$ denotes the arrival time of the i-th token in p, then the *total number of tokens visiting place p* during the experiment interval is:

$$N_p(\theta) = \sum_{i=1}^{\infty} \#(\alpha(p, i) \leq \theta)$$

where for an arbitrary predicate P, the function $\#$ is defined as:

$$\#(P) = \begin{cases} 1 & \text{if } P = \text{true} \\ 0 & \text{otherwise} \end{cases}$$

If $\delta(p, i)$ denotes the departure time of the i-th token from p, then for all place p such that $N_p(\theta) > 0$, the *average token residence time* in place p during the interval $(0, \theta)$ is the following:

$$\overline{\mathbf{r}}[p] = \frac{1}{N_p(\theta)} \sum_{k=1}^{N_p(\theta)} \int_0^\theta (\#(\alpha(p, k) \leq \tau) - \#(\delta(p, k) < \tau))\, d\tau$$

An alternative definition of the above indices may be given in a unified way in terms of *reward functions*, or index functions defined over the reachable markings of the PN system. First, let us denote by $\pi_i(\theta)$ the *proportion of time spent by the system in a given reachable marking,* $\mathbf{m_i}$*, during the interval* $(0, \theta)$:

$$\pi_i(\theta) = \frac{1}{\theta} \int_0^\theta \#(\boldsymbol{\mu}(\tau) = \mathbf{m_i})\, d\tau$$

A reward function over reachable markings is a function:

$$\begin{array}{rccc} r: & \mathrm{RS} & \longrightarrow & \mathbb{R}^+ \\ & \mathbf{m} & \rightsquigarrow & r(\mathbf{m}) \end{array}$$

From that function, an *operational average reward* between 0 and θ can be computed as the weighted sum:

$$\overline{r}(\theta) = \sum_{\mathbf{m_i} \in \mathrm{RS}} r(\mathbf{m_i})\, \pi_i(\theta)$$

Different average performance indices can be computed as operational average rewards by giving different interpretations to the reward function. For instance, the *proportion of time during the experiment interval that a given predicate on the marking, $P(\mathbf{m})$, holds* can be defined using the following reward function:

$$r(\mathbf{m}) = \begin{cases} 1 & \text{if } P(\mathbf{m}) = \text{true} \\ 0 & \text{otherwise} \end{cases}$$

The average marking in place p during the interval $(0, \theta)$ can be redefined using the following reward function:

$$r(\mathbf{m}) = n \text{ if and only if } \mathbf{m}[p] = n$$

The equivalence of both definitions of average marking can be easily shown:

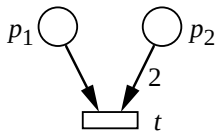
$$\begin{aligned} \overline{r}(\theta) &= \sum_{\mathbf{m_i} \in \mathrm{RS}} r(\mathbf{m_i})\, \pi_i(\theta) = \sum_{\mathbf{m_i} \in \mathrm{RS}} \mathbf{m_i}[p]\, \frac{1}{\theta} \int_0^\theta \#(\boldsymbol{\mu}(\tau) = \mathbf{m_i})\, d\tau = \\ &= \frac{1}{\theta} \int_0^\theta \sum_{\mathbf{m_i} \in \mathrm{RS}} \mathbf{m_i}[p]\, \#(\boldsymbol{\mu}(\tau) = \mathbf{m_i})\, d\tau = \frac{1}{\theta} \int_0^\theta \boldsymbol{\mu}[p](\tau)\, d\tau = \\ &= \overline{\boldsymbol{\mu}}[p](\theta) \end{aligned}$$

In a similar way, other average performance indices can be defined as operational average rewards.

8.2 From Performance Measures to Operational Laws

Operational analysis is a conceptually very simple way of deriving mathematical equations relating observable quantities in queueing systems. In this section operational analysis techniques are applied to derive linear equations and inequalities relating interesting performance measures in timed Petri net models.

The first result is the *enabling law* which is the Petri net counterpart of the well-known "utilization law" in queueing systems literature.



$$\boldsymbol{\chi}[t] \cdot \overline{\mathbf{s}}[t] = \overline{\mathbf{e}}[t] \leq \left(\overline{\boldsymbol{\mu}}[p_1], \frac{\overline{\boldsymbol{\mu}}[p_2]}{2} \right)$$

Figure 8.1: Maximum throughput law.

Property 8.1 (Enabling Law) *For all transition $t \in T$ with at least one input place and of type either persistent or age memory or immediate and for all experiment interval $(0, \theta)$ with $\theta > 0$ and $F_t(\theta) > 0$*

$$\overline{\mathbf{e}}[t] = \boldsymbol{\chi}[t] \cdot \overline{\mathbf{s}}[t]$$

Proof Sketch:
Trivial substitution of the definitions. ♢

From the enabling law it follows that if the average firing time of a transition is known, then its throughput is proportional to its average enabling degree. Of course in case of immediate transitions $\overline{\mathbf{s}}[t] = 0$, so immediate transitions are never enabled for non-null intervals of time.

The main conceptual difference between *basic queueing networks* (using the terminology introduced in Chapter 1) and Petri net models (or *extended queueing networks*) is the presence of a synchronization primitive in the latter, therefore the following operational inequalities derived for synchronization elements have no counterpart in operational laws for basic queueing networks.

Property 8.2 (Maximum Throughput Law) *For all transition $t \in T$ with at least one input place and of type either persistent or age memory or immediate and for all experiment interval $(0, \theta)$ with $\theta > 0$ and $F_t(\theta) > 0$*

$$\boldsymbol{\chi}[t] \cdot \overline{\mathbf{s}}[t] \leq \min_{p \in {}^\bullet t} \left(\frac{\overline{\boldsymbol{\mu}}[p]}{\mathbf{Pre}[p,t]} \right)$$

Proof:
Equation (8.1), that is valid in each instant of the experiment, implies that $\forall p \in {}^\bullet t$, $\forall \tau : 0 \leq \tau \leq \theta$, $\mathbf{e}[t](\tau) \leq \frac{\boldsymbol{\mu}[p](\tau)}{\mathbf{Pre}[p,t]}$.

Therefore $\forall p \in {}^\bullet t$, $\overline{\mathbf{e}}[t] \leq \frac{\overline{\boldsymbol{\mu}}[p]}{\mathbf{Pre}[p,t]}$, and applying the enabling law the result follows. ♢

This inequality establishes an upper bound for the average enabling (hence for the transition throughput once the firing time is defined) in the case of transitions with more than one input place that model a synchronization. As depicted in figure 8.1, the instantaneous enabling degree of a transition cannot be bigger than the marking of its input places (divided by the arc weight), therefore the same applies for the average values.

Property 8.3 (Minimum Throughput Law for single input transitions) *For all transition $t \in T$ with exactly one input place p and of type either persistent or age memory or immediate,*

$$\boldsymbol{\chi}[t] \cdot \overline{\mathbf{s}}[t] \geq \frac{\overline{\boldsymbol{\mu}}[p] - \mathbf{Pre}[p,t] + 1}{\mathbf{Pre}[p,t]} \tag{8.2}$$

Proof:
First define some auxiliary punctual marking functions:

$$\forall p \in P, \quad \forall \tau, \quad \boldsymbol{\mu}^v[p](\tau) \;=\; \max(0, \boldsymbol{\mu}[p](\tau) - v)$$

$$\forall p \in P, \quad \forall \tau, \quad \boldsymbol{\mu}_l^u[p](\tau) \;=\; \boldsymbol{\mu}^l[p](\tau) - \boldsymbol{\mu}^u[p](\tau)$$

Consider now the following properties of the auxiliary function $\forall k \in \mathbb{N} : k > 0$,

$$0 \;\leq\; \boldsymbol{\mu}_{kw-1}^{(k+1)w-1}[p](\tau) \;\leq\; w$$

Moreover, notice that the k-th server in transition t is enabled if and only if $\boldsymbol{\mu}_{kw-1}^{(k+1)w-1}[p](\tau) \geq 1$ in case $w = \mathbf{Pre}[p,t]$. Therefore,

$$\forall t \in T : {}^\bullet t = \{p\}, \quad \forall k \geq 1, \quad \forall \tau, \quad \mathbf{e_k}[t](\tau) \;\geq\; \frac{1}{\mathbf{Pre}[p,t]}\, \boldsymbol{\mu}_{k\mathbf{Pre}[p,t]-1}^{(k+1)\mathbf{Pre}[p,t]-1}[p](\tau)$$

where $\mathbf{e_k}[t](\tau)$ is the instantaneous enabling of k-th server in t, i.e., the characteristic function that evaluates to 1 if and only if the k-th server in transition t is busy at time τ ($\mathbf{e_k}[t](\tau) =$ if $\mathbf{e}[t](\tau) \geq k$ then 1 else 0).

Hence, $\forall t \in T : {}^\bullet t = \{p\}$:

$$\forall \tau, \; \mathbf{e}[t](\tau) \;\geq\; \frac{1}{\mathbf{Pre}[p,t]} \sum_{k=1}^{\infty} \boldsymbol{\mu}_{k\mathbf{Pre}[p,t]-1}^{(k+1)\mathbf{Pre}[p,t]-1}[p](\tau) \;=\; \frac{\boldsymbol{\mu}[p](\tau) - \mathbf{Pre}[p,t] + 1}{\mathbf{Pre}[p,t]}$$

Finally, taking the average over the experiment interval and applying the enabling law, the result follows. ♢

Observe that in the case that the right-hand expression in (8.2) is negative, a trivial inequality holds: $\boldsymbol{\chi}[t] \cdot \overline{\mathbf{s}}[t] \geq 0$.

Corollary 8.4 (Throughput Law for ordinary, single input transitions) *For all transition $t \in T$ with exactly one input place p and of type either persistent or age memory or immediate, and such that* $\mathbf{Pre}[p,t] = 1$,

$$\boldsymbol{\chi}[t] \cdot \overline{\mathbf{s}}[t] = \overline{\boldsymbol{\mu}}[p]$$

Proof Sketch:
Combining Properties 8.2 and 8.3. ♢

The above result is nothing more than the well-known expression for the average number of customers at delay stations in queueing networks terminology (remember that since we are considering infinite server semantics, the case of a transition with only one input place with ordinary incidence arc models a delay server).

In the particular case of bounded nets, the above minimum throughput law can be improved, as stated in the next results.

Property 8.5 (Minimum Throughput Law for single input transitions in bounded nets) *For all transition $t \in T$ with exactly one input place p and of type either persistent or age memory or immediate, if $\forall \tau$: $\boldsymbol{\mu}[p](\tau) \leq b_p$ and $\exists k \in \mathbb{N} : k\ \mathbf{Pre}[p,t] \leq b_p < (k+1)\mathbf{Pre}[p,t]$, then*

$$\boldsymbol{\chi}[t] \cdot \overline{\mathbf{s}}[t] \geq\ k\ \frac{\overline{\boldsymbol{\mu}}[p] - k\ \mathbf{Pre}[p,t] + 1}{b_p - k\ \mathbf{Pre}[p,t] + 1}$$

Proof:
Firstly note that $\forall t \in T$, $\forall j \in \mathbb{N}$

$$\overline{\mathbf{e}}[t] \geq\ j\ \frac{1}{\theta} \int_0^\theta \mathbf{e_j}[t](\tau)\ d\tau$$

where $\mathbf{e_j}[t](\tau)$ is defined as in the proof of Property 8.3.

Secondly, note that the marking in the input place p can be expressed as the sum of two components:

$$\forall \tau, \quad \boldsymbol{\mu}[p](\tau)\ =\ \mathbf{Pre}[p,t] \sum_{j=1}^{\infty} \mathbf{e_j}[t](\tau)\ + \boldsymbol{\nu}[p](\tau)$$

where the component $\boldsymbol{\nu}[p](\tau) \leq \mathbf{Pre}[p,t] - 1$ represents the portion of marking not used to enable the transition. Now taking the integral and dividing by θ one obtains:

$$\overline{\boldsymbol{\mu}}[p]\ =\ \mathbf{Pre}[p,t]\ \overline{\mathbf{e}}[t] + \overline{\boldsymbol{\nu}}[p]$$

This equation shows that the average enabling depends only on the mean values of the input place marking and of the unused portion of the marking.

The worst case from the point of view of enabling the transition k times occurs when the place is marked with $\mathbf{Pre}[p,t]k-1$ tokens most of the time and with b_p tokens for the rest of the time, since this case maximizes the unused portion of the average marking in the input place. From these considerations the result follows. ♢

Property 8.6 (Minimum Throughput Law for binary synchronizations with ordinary arcs in bounded nets) *For all transition $t \in T$ of type either persistent or age memory or immediate with ${}^\bullet t = \{p_1, p_2\}$ and $\mathbf{Pre}[p_1,t] = \mathbf{Pre}[p_2,t] = 1$, if $\forall \tau$: $\boldsymbol{\mu}[p_1](\tau) \leq b_{p_1}$ and $\boldsymbol{\mu}[p_2](\tau) \leq b_{p_2}$, then*

$$\boldsymbol{\chi}[t] \cdot \overline{\mathbf{s}}[t] \geq\ \overline{\boldsymbol{\mu}}[p_1] + \frac{b_{p_1}}{b_{p_2}} \overline{\boldsymbol{\mu}}[p_2] - b_{p_1}$$

Proof:
Similarly to the previous case, two equations can be written relating the average marking, the average enabling, and the average portion of unused marking for each of the two input places:

$$\overline{\mathbf{e}}[t] = \overline{\boldsymbol{\mu}}[p] - \overline{\boldsymbol{\nu}}[p_1]$$

$$\overline{\mathbf{e}}[t] = \overline{\boldsymbol{\mu}}[p_2] - \overline{\boldsymbol{\nu}}[p_2]$$

Now, upper bounds on the unused part of the marking can be computed as follows. The maximum fraction of time during which $\boldsymbol{\nu}[p_1](\tau)$ may be greater than zero is equal to the minimum time during which $\boldsymbol{\mu}[p_2](\tau) = 0$ (otherwise the transition would be enabled and the marking of p_1 would contribute to the enabling instead); since place p_2 is b_{p_2}–bounded, this fraction of time is less than or equal to $1 - \frac{\overline{\boldsymbol{\mu}}[p_2]}{b_{p_2}}$; moreover during this maximum time, the maximum value of the marking in p_1 is less than or equal to b_{p_1}. Hence

$$\overline{\boldsymbol{\nu}}[p_1] \leq b_{p_1} \left(1 - \frac{\overline{\boldsymbol{\mu}}[p_2]}{b_{p_2}}\right)$$

and from this the result trivially follows. ♢

Property 8.7 (Minimum Throughput Law in bounded nets) *For all transition $t \in T$ of type either persistent or age memory or immediate with ${}^\bullet t = \{p_1, p_2, \ldots, p_k\}$, if $\forall j : 1 \leq j \leq k$, $\forall \tau :$ $\boldsymbol{\mu}[p_j](\tau) \leq b_{p_j}$ and $b_{p_1} \leq b_{p_j}$, then*

$$\boldsymbol{\chi}[t] \cdot \overline{\mathbf{s}}[t] \geq \frac{\overline{\boldsymbol{\mu}}[p_1] - \mathbf{Pre}[p_1, t] + 1 - b_{p_1} \ \max_j(f_j)}{\mathbf{Pre}[p_1, t]}$$

where $\forall j : 2 \leq j \leq k$, $f_j = 1 - \frac{\overline{\boldsymbol{\mu}}[p_j] - \mathbf{Pre}[p_j, t] + 1}{b_{p_j} - \mathbf{Pre}[p_j, t] + 1}$

Proof Sketch:
Similar to the previous ones writing the upper bound for the quantity $\overline{\boldsymbol{\nu}}[p_1]$. ♢

The next property is the analogous of Little's law in queueing systems.

Property 8.8 (Little's Law) *For all place $p \in P$:*

1. *For all experiment interval $(0, \theta)$ with $\theta > 0$ and $F_t(\theta) > 0$:*

$$\overline{\mathbf{r}}[p] \left(\frac{\mathbf{m_0}[p]}{\theta} + \sum_{t \in {}^\bullet p} \mathbf{Post}[p, t] \boldsymbol{\chi}[t] \right) = \overline{\boldsymbol{\mu}}[p]$$

Notice that, even if it is not made explicit in the notation, $\overline{\mathbf{r}}[p]$, $\boldsymbol{\chi}[t]$, and $\overline{\boldsymbol{\mu}}[p]$ are functions of the duration of the experiment interval θ.

2. *If the limits* $\overline{\boldsymbol{\mu}}^*[p] = \lim_{\theta\to\infty} \overline{\boldsymbol{\mu}}[p]$ *and* $\boldsymbol{\chi}^*[t] = \lim_{\theta\to\infty} \boldsymbol{\chi}[t]$ *exist, then:*

$$\overline{\mathbf{r}}^*[p] = \lim_{\theta\to\infty} \overline{\mathbf{r}}[p] = \frac{\overline{\boldsymbol{\mu}}^*[p]}{\sum_{t\in {}^\bullet p} \mathbf{Post}[p,t]\boldsymbol{\chi}^*[t]}$$

Proof Sketch:
It follows taking into consideration the corresponding definitions and exchanging sum and integral signs in the definition of average token residence time. ◊

8.3 Visit Ratios Computability: A Net Hierarchy

In a classical monoclass queueing network, the following system can be derived by equating the rate of *flow of customers* into each station to the rate of flow out of the station:

$$\chi_j = \chi_{0j} + \sum_{i=1}^{m} \chi_i \, r_{ij}, \qquad j = 1, \ldots, m$$

where χ_i is the limit *throughput* of station i, i.e., the average number of service completions per unit time at station i, $i = 1, \ldots, m$; r_{ij}, is the probability that a customer exiting center i goes to j $(i, j = 1, \ldots, m)$; and χ_{0j} is the external arrival rate of customers to station j $(j = 1, \ldots, m)$, i.e., the average number of customers arriving at station j per unit of time.

If the network is open (i.e., if there exists a station j with positive external arrival rate, $\chi_{0j} > 0$ and also customers can leave the system), then the above m equations are linearly independent, and the exact throughput of stations can be derived (independently of the service times, s_i, $i = 1, \ldots, m$). This is not the case for closed networks. If $\chi_{0j} = 0$, $j = 1, \ldots, m$, then only $m - 1$ equations are linearly independent, and thus only relative throughputs can be determined, i.e., the *visit ratios*, denoted as v_i, for each station i.

Unfortunately, concerning timed Petri net models, the introduction of synchronization schemes can lead to the "pathological" behaviour of models reaching a total deadlock, thus with null visit ratios for all transitions, in the limit. In other words, for these models it makes no sense to speak about steady-state behaviour. Therefore, in the rest of this chapter only deadlock-free Petri nets are considered. Even more, in most systems that will be studied, deadlock-freeness implies liveness of the net, in other words, the existence of an infinite activity of all the transitions is assured.

8.3.1 The Visit Ratios Computation Problem

The counterpart of routing of customers in queueing networks consists both on the *net structure* $\mathcal{N}$ and the relative *routing rates at conflicts*, denoted $\mathbf{R}$, in

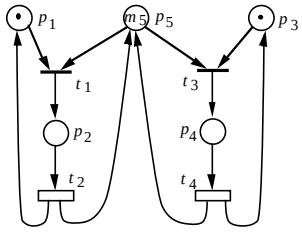


Figure 8.2: A net system whose visit ratios depend on the structure, on the routing at conflicts, on the initial marking, and on the firing times.

strongly connected marked graphs	$\mathbf{v} = \mathbf{1}$ {constant}
mono-T-semiflow nets	$\mathbf{v} = \mathrm{f}(\mathcal{N})$
live and bounded free choice nets	$\mathbf{v} = \mathrm{f}(\mathcal{N}, \mathbf{R})$
simple nets	$\mathbf{v} = \mathrm{f}(\mathcal{N}, \mathbf{R}, \mathbf{m_0}, \overline{\mathbf{s}})$

Table 8.1: Computability of the vector of visit ratios and net subclasses.

timed Petri nets. Unfortunately, in the general Petri net case it is not possible to derive the visit ratios only from $\mathcal{N}$ and $\mathbf{R}$. Net systems can be constructed such that the visit ratios for transitions do depend on the net structure, on the routing rates at conflicts, but also on the initial marking (distribution of customers), and on the distribution firing time (thus, in particular, on the average firing time) of transitions:

$$\mathbf{v} = \mathrm{f}(\mathcal{N}, \mathbf{R}, \mathbf{m_0}, \overline{\mathbf{s}})$$

where $\mathbf{v}$ is normalized, for instance, for transition t_1.

As an example, let us consider the net system depicted in Figure 8.2. Transitions t_1 and t_3 are *immediate* (i.e., they fire in zero time). The constants $r_1, r_3 \in \mathbb{N}^+$ define the conflict resolution policy, i.e., when t_1 and t_3 are simultaneously enabled, t_1 fires with relative rate $r_1/(r_1+r_3)$ and t_3 with $r_3/(r_1+r_3)$. Let s_2 and s_4 be the average firing times of t_2 and t_4, respectively. If $m_5 = 1$ (initial marking of p_5) then p_1 and p_3 are *implicit*, hence they can be deleted without affecting the behaviour! Thus a closed queueing network topology is derived. A product form queueing network can be obtained and the visit ratios, normalized for transition t_1 can be computed: $\mathbf{v} = (1, 1, r_3/r_1, r_3/r_1)$. If $m_5 = 2$ (different initial marking for p_5) then p_5 is now implicit, hence it can be deleted; two isolated closed tandem queueing networks are obtained and $\mathbf{v}' = (1, 1, s_2/s_4, s_2/s_4)$. Obviously $\mathbf{v} \neq \mathbf{v}'$, in general.

In this section, we consider those nets whose associated vector of visit ratios for transitions can be computed from the net structure and the routing rates at conflicts. Since the existence of strictly positive visit ratios requires liveness of the net and we are looking for an structural computation, the presentation is restricted to *structurally live* Petri nets. On the other hand, unless otherwise explicitly stated, *structurally bounded* nets are considered. Under these

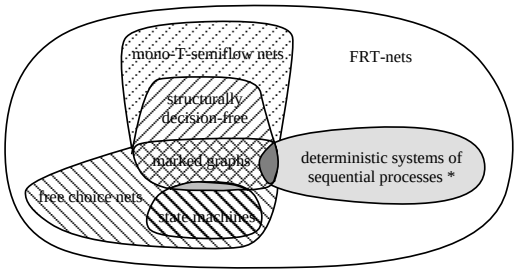


Figure 8.3: Inclusion relations among FRT-net subclasses (* these are marked nets).

restrictions, a characterization of the class of nets for which the vector of visit ratios can be computed *without any behavioural analysis* (i.e., from structure and routing rates at conflicts) in terms of a *rank condition* over the incidence matrix of the net is presented. These nets are defined as having *freely related T-semiflows* and denoted, for short, as *FRT-nets*. This means that they can have several independent T-semiflows but the vector of visit ratios, which is always a linear combination of the minimal T-semiflows, is computed as an "average T-semiflow" from local free choices among transitions, governed by the routing rates.

A particular class of FRT-nets is that of *mono-T-semiflow nets*, which have a unique minimal T-semiflow. They include *choice-free nets* (if strongly connected and consistent) and, in particular, the well-known net subclass of *marked graphs*.

FRT-nets include also *free choice nets*. This last class includes marked graphs, thus intersecting with mono-T-semiflow nets. *State machines* are also included in the class of free choice nets and constitute the Petri net counterpart of monoclass queueing networks.

Another well-known subclass of marked FRT-nets is that of *deterministic systems of sequential processes*, that is, 1–bounded state machines communicating through private buffers non-disturbing local decisions at state machines. In fact, *FRT-nets communicating through private buffers non-disturbing local decisions* are also FRT-nets. In this sense, FRT-nets can be recursively defined.

Inclusion relations among above mentioned subclasses are depicted in Figure 8.3.

8.3.2 FRT-Nets

In the following paragraphs the class of structurally live and structurally bounded nets whose vector of visit ratios for transitions can be computed just from the net structure and the routing rates at conflicts is defined. Nets belonging to this subclass are said to have *freely related T-semiflows* and called *FRT-nets*.

Before giving a formal definition, let us remark that the vector of visit ratios

for transitions of any net should verify the two following conditions:

1. The vector of visit ratios $\mathbf{v}$ (normalized, for instance, for transition t_1) must be a non-negative right annuller of the incidence matrix:

$$\mathbf{C} \cdot \mathbf{v} = \mathbf{0}$$

2. The components of $\mathbf{v}$ must verify the following relations with respect to the routing rates for each subset of transitions $T_i = \{t_1, \ldots, t_k\} \subset T$ in *generalized free (or equal) conflict* (i.e., having equal pre-incidence function: $\mathbf{Pre}[\cdot, t_1] = \ \ldots \ = \mathbf{Pre}[\cdot, t_k]$):

$$\begin{aligned} r_2\mathbf{v}[t_1] - r_1\mathbf{v}[t_2] &= 0 \\ r_3\mathbf{v}[t_2] - r_2\mathbf{v}[t_3] &= 0 \\ &\ldots \\ r_k\mathbf{v}[t_{k-1}] - r_{k-1}\mathbf{v}[t_k] &= 0 \end{aligned}$$

Expressing the former homogeneous system of equations in matrix form: $\mathbf{R}[T_i] \cdot \mathbf{v} = \mathbf{0}$, where $\mathbf{R}[T_i]$ is a $(k-1) \times m$ matrix. Now, by considering all generalized free conflicts $T_1, \ldots, T_r$: $\mathbf{R} \cdot \mathbf{v} = \mathbf{0}$, where $\mathbf{R}$ is a matrix:

$$\mathbf{R} = \begin{pmatrix} \mathbf{R}[T_1] \\ \vdots \\ \mathbf{R}[T_r] \end{pmatrix}$$

$\mathbf{R}$ is a *routing matrix* with δ rows and $m = |T|$ columns, where δ is the difference between the number of transitions in generalized free conflict and the number of subsets of transitions in generalized free conflict ($\delta < m$) or, in other words, the number of independent relations fixed by the routing rates at conflicts. Given that $r_i \neq 0$ for all i, it can be observed that, by construction, $\text{rank}(\mathbf{R}) = \delta$. The above remarked conditions together with the normalization constraint for transition t_1, $\mathbf{v}[t_1] = 1$, characterize a unique vector if and only if the number of independent rows of the matrix

$$\begin{pmatrix} \mathbf{C} \\ \mathbf{R} \end{pmatrix}$$

is $m - 1$.

The class of structurally live and structurally bounded nets verifying the previous rank condition is introduced now. In order to do that, an equivalence relation on the set of T-semiflows of the net is defined. After that, the class of FRT-nets will be defined as nets having only one equivalence class for this relation.

Definition 8.9 (Freely connected T-semiflows) *Let $\mathcal{N}$ be a Petri net and $\mathbf{x_a}$, $\mathbf{x_b}$ two different T-semiflows of $\mathcal{N}$. $\mathbf{x_a}$ and $\mathbf{x_b}$ are said to be freely connected by places $P' \subset P$, denoted as $\mathbf{x_a} \overset{P'}{\wedge} \mathbf{x_b}$, iff $\exists t_a \in \|\mathbf{x_a}\|, t_b \in \|\mathbf{x_b}\|$ such that: $\mathbf{Pre}[\cdot, t_a] = \mathbf{Pre}[\cdot, t_b]$ and ${}^\bullet t_a = {}^\bullet t_b = P'$.*

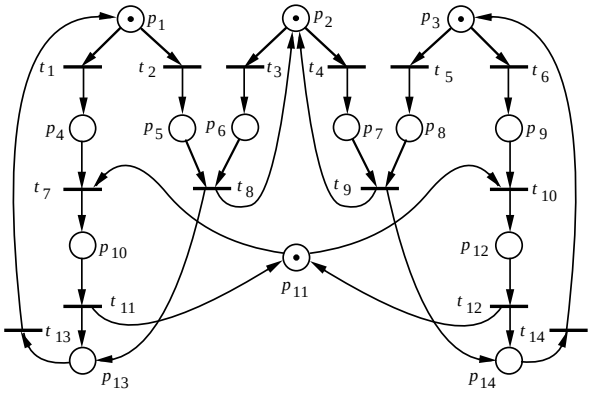


Figure 8.4: A live and structurally bounded FRT-net.

Definition 8.10 (Freely related T-semiflows) *Let $\mathcal{N}$ be a Petri net and $\mathbf{x_a}$, $\mathbf{x_b}$ two T-semiflows of $\mathcal{N}$. $\mathbf{x_a}$ and $\mathbf{x_b}$ are said to be freely related, denoted as $\langle \mathbf{x_a}, \mathbf{x_b} \rangle \in \mathrm{FR}$, iff one of the following conditions holds:*

1. $\mathbf{x_a} = \mathbf{x_b}$,
2. $\exists P' \subset P$ *such that* $\mathbf{x_a} \overset{P'}{\wedge} \mathbf{x_b}$, *or*
3. $\exists \mathbf{x_1}, \ldots, \mathbf{x_k}$ *T-semiflows of $\mathcal{N}$ and* $P_1, \ldots, P_{k+1} \subset P$, $k \geq 1$, *such that* $\mathbf{x_a} \overset{P_1}{\wedge} \mathbf{x_1} \overset{P_2}{\wedge} \ldots \overset{P_k}{\wedge} \mathbf{x_k} \overset{P_{k+1}}{\wedge} \mathbf{x_b}$.

From the above definition the next property trivially follows:

Property 8.11 FR *is an equivalence relation on the set of T-semiflows of a net.*

The introduction of this equivalence relation on the set of T-semiflows induces a partition into equivalence classes. FRT-nets are defined as follows:

Definition 8.12 (FRT-nets) *A Petri net $\mathcal{N}$ is a net with freely related T-semiflows (FRT-net, for short) iff the introduction of the freely relation on the set of its T-semiflows induces only one equivalence class.*

Note that FRT-nets are necessarily connected. Therefore, in what follows, unless otherwise explicitly stated, we consider only connected nets.

As an example, let us consider the net depicted in Figure 8.4. It is a live and structurally bounded net. Its minimal T-semiflows are:

$$\begin{array}{lcl} \mathbf{x_1} & = & (1,0,0,0,0,0,1,0,0,0,1,0,1,0) \\ \mathbf{x_2} & = & (0,1,1,0,0,0,0,1,0,0,0,0,1,0) \\ \mathbf{x_3} & = & (0,0,0,1,1,0,0,0,1,0,0,0,0,1) \\ \mathbf{x_4} & = & (0,0,0,0,0,1,0,0,0,1,0,1,0,1) \end{array} \tag{8.3}$$

Then, the net is an FRT-net because:

$$\mathbf{x_1} \overset{\{p_1\}}{\wedge} \mathbf{x_2} \overset{\{p_2\}}{\wedge} \mathbf{x_3} \overset{\{p_3\}}{\wedge} \mathbf{x_4}$$

From the definition of FRT-nets, it may appears that a direct checking of the membership of a given net to this net subclass is not a polynomial problem on the net size. This is because the number of T-semiflows of a net can grow exponentially with the number of places and transitions. However, if structural liveness and structural boundedness are assumed, a nice characterization of the FRT-nets subclass can be obtained and checked in polynomial time. Before the presentation of that result, a second equivalence relation, now on the set of transitions of the net, is introduced.

Definition 8.13 (Equality conflict relation) *Two transitions t_a and t_b are said to be in equality conflict relation, denoted by $\langle t_a, t_b \rangle \in$ ECR, iff $\mathbf{Pre}[\cdot, t_a] = \mathbf{Pre}[\cdot, t_b]$.*

Since the equality conflict relation is based on the equality of vectors, the next property follows:

Property 8.14 ECR *is an equivalence relation on the set of transitions.*

Each equivalence class will be called *equality conflict set*, ECS for short. Let D be an ECS, the number $\delta_D = |D| - 1$ is called *number of non-redundant free conflicts of D*. The reason of the name lies on the fact that δ_D is exactly the number of independent relations among the throughput of transitions belonging to D that can be derived from the routing rates defining the resolution of the conflict. The *number of non-redundant free conflicts of a net*, denoted as δ, is the sum of all δ_D corresponding to the ECS's of the net: $\delta = \sum_{D \in T/ECR} \delta_D$.

Theorem 8.15 *Let $\mathcal{N}$ be a structurally live and structurally bounded net. Then $\mathcal{N}$ is a FRT-net if and only if* $\text{rank}(\mathbf{C}) = m - \delta - 1$, *where $\mathbf{C}$ is the incidence matrix of $\mathcal{N}$, $m = |T|$, and δ is the number of non-redundant free conflicts of the net.*

Due to its considerable extension, we do not include here the proof of the above Theorem. The interested reader is referred to the bibliography.

An important conclusion is stated in the next Corollary.

Corollary 8.16 *If $\mathcal{N}$ is structurally live and structurally bounded, deciding if $\mathcal{N}$ belongs to the class of FRT-nets is polynomial on the net size.*

Structural boundedness of a net can always be checked in polynomial time (iff $\exists \mathbf{y} \geq \mathbf{1}$ such that $\mathbf{y} \cdot \mathbf{C} \leq \mathbf{0}$). Unfortunately, structural liveness of FRT-nets cannot be decided (so far) efficiently. Nevertheless, a *necessary condition* for a net to be structurally live structurally bounded and FRT-net can be checked in polynomial time, looking for the consistency, conservativeness, and rank condition over the incidence matrix, because structural liveness and structural boundedness implies consistency and conservativeness.

The next result gives a *polynomial time* method for the computation of the vector of visit ratios for transitions of a live and structurally bounded FRT-net, from the knowledge of the net structure and the routing rates at conflicts.

Theorem 8.17 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a live and structurally bounded FRT-net system. Let $\mathbf{C}$ be the incidence matrix of $\mathcal{N}$ and $\mathbf{R}$ the previously introduced routing matrix. Then, the vector of visit ratios $\mathbf{v}$ normalized for transition t_1 can be computed from $\mathbf{C}$ and $\mathbf{R}$ by solving the following linear system of equations:*

$$\begin{pmatrix} \mathbf{C} \\ \mathbf{R} \end{pmatrix} \cdot \mathbf{v} = \mathbf{0}, \quad \mathbf{v}[t_1] = 1 \tag{8.4}$$

Proof:
We only have to check that the above system has a unique solution. By Theorem 8.15, the number of independent rows of matrix $\mathbf{C}$ is $m - \delta - 1$. Therefore, the $m - \delta - 1$ independent conditions given by $\mathbf{C} \cdot \mathbf{v} = \mathbf{0}$ plus the δ independent conditions given by $\mathbf{R} \cdot \mathbf{v} = \mathbf{0}$ plus the normalization condition $\mathbf{v}[t_1] = 1$ are enough to determine exactly the m components of the vector $\mathbf{v}$. ♢

The reader can notice that a *rank condition* over the incidence matrix $\mathbf{C}$ exists underlying Theorem 8.17: the system of equations (8.4) has a unique solution $\mathbf{v}$ if and only if $\text{rank}(\mathbf{C}) = m - \delta - 1$, where δ is the rank of $\mathbf{R}$.

From the above theorem, for structurally live and structurally bounded FRT-nets we have:

$$\mathbf{v} = \text{f}(\mathcal{N}, \mathbf{R})$$

and the next complexity result follows:

Corollary 8.18 *The computation of the vector of visit ratios for transitions of a structurally live and structurally bounded FRT-net is polynomial on the net size.*

As an example, let us consider again the net depicted in Figure 8.4. The vector of visit ratios must be a right annuller of the incidence matrix, hence a linear combination of a basis of T-semiflows:

$$\mathbf{v} = \sum_{i=1}^{4} \alpha_i \mathbf{x_i} \tag{8.5}$$

where $\mathbf{x_i}$, $i = 1, \ldots, 4$, are the minimal T-semiflows (8.3) of the net. If r_1, r_2 are the routing rates of t_1, t_2 in the conflict at p_1; r_3, r_4 the routing rates of t_3, t_4 in the conflict at p_2; and r_5, r_6 the routing rates of t_5, t_6 in the conflict at p_3, then $\mathbf{v}$ must satisfy:

$$\begin{aligned} r_2 \mathbf{v}[t_1] &= r_1 \mathbf{v}[t_2] \\ r_4 \mathbf{v}[t_3] &= r_3 \mathbf{v}[t_4] \\ r_6 \mathbf{v}[t_5] &= r_5 \mathbf{v}[t_6] \end{aligned}$$

And together with the normalization requirement:

$$\mathbf{v}[t_1] = 1 \tag{8.6}$$

the four parameters α_i, $i = 1, \ldots, 4$, can be determined.

Another interesting qualitative property follows from Theorem 8.17, that does not hold for general nets:

Property 8.19 *Let $\mathcal{N}$ be a structurally live and structurally bounded FRT-net. Then $\mathcal{N}$ is live if and only if it is deadlock-free.*

Proof Sketch:
The "only if" direction is always true by definitions of liveness and deadlock-freeness. The other direction follows from Theorem 8.17. The vector of visit ratios for transitions is univocally determined in that theorem and all its components are non-null. If an infinite behaviour of the net always occurs (i.e., if the net is deadlock-free) the limit throughput of transitions must be proportional to the vector of visit ratios, which is strictly positive. Therefore, the net is live. ♢

8.4 Ergodicity and Stochastic Definition of Performance Indices

In Section 8.1, performance measures were defined *operationally* from the basic observable events in the evolution of the model behaviour. In particular, we saw how *transient* performance indices can be defined from the following *time average*:

$$\pi_i(\theta) = \frac{1}{\theta} \int_0^\theta \#(\boldsymbol{\mu}(\tau) = \mathbf{m_i})\, d\tau \tag{8.7}$$

The above value represents the proportion of time spent by the system in a given reachable marking, $\mathbf{m_i}$, during the interval $(0, \theta)$.

If instead of a single execution of the model, an infinite number of sample paths is considered, a *sample average* between 0 and θ, $\tilde{\pi}_i(\theta)$, can be defined as the average of $\pi_i(\theta)$ over all the sample paths.

Even if the above operational definition is the natural way of defining performance indices, few analysis techniques for the computation of such indices are presently known (one of them will be presented in Chapter 17). An alternative (classical) approach is to quantify the ocurrence of the observable events as *random variables*, therefore the sequences of observable events are *stochastic processes* for which, under certain assumptions, efficient analysis techniques exist (they will be presented in Chapters 9, 10, and 11). Thus, the stochastic counterpart of the above time average (8.7) can be defined as:

$$\eta_i(\theta) = \mathrm{E}[\boldsymbol{\mu}(\theta)] \tag{8.8}$$

where $\eta_i(\theta)$ represents the *expected value* (mathematical expectation operator) of $\boldsymbol{\mu}(\theta)$, the state distribution of the system at time θ.

An important fact is that the sample average and the expected value are equal:

$$\eta_i(\theta) = \tilde{\pi}_i(\theta)$$

The above values correspond to the transient behaviour, i.e., to the state of the system in a finite interval $(0, \theta)$. The corresponding limit or *steady-state* behaviours can be defined also in both the operational and the stochastic framework. Concerning the operational framework, the limit $(\theta \to \infty)$ of the time average can be defined as:

$$\pi_i = \lim_{\theta \to \infty} \pi_i(\theta) \tag{8.9}$$

On the other hand, within the stochastic framework, the existence of the following limit expected value can be considered:

$$\eta_i = \lim_{\theta \to \infty} \eta_i(\theta) \tag{8.10}$$

If the system is *ergodic*, the above limit exists and it is independent of the initial state. Moreover, if the system is ergodic the limit time average (8.9) and the limit expected value (8.10) are equal:

$$\pi_i = \eta_i$$

An important consequence of the above equality is that a single (infinite) sample path characterizes the expected behaviour of the system.

The steady-state probability distribution η_i can be used to compute the expected performance measures as average rewards, if the corresponding reward functions are properly defined.

The *expected marking* in place p can be computed as the following average reward:

$$\overline{\boldsymbol{\mu}}[p] = \sum_{\mathbf{m_i} \in \mathrm{RS}} r(\mathbf{m_i})\, \eta_i$$

where the reward function is:

$$r(\mathbf{m}) = n \text{ if and only if } \mathbf{m}[p] = n$$

The *throughput* of transition t can be derived from the following reward function

$$r(\mathbf{m}) = \begin{cases} \lambda_t & \text{if } t \text{ is enabled in } \mathbf{m} \\ 0 & \text{otherwise} \end{cases}$$

(where λ_t is the firing rate of transition t, i.e., the inverse of its average service time) as the average reward:

$$\boldsymbol{\chi}[t] = \sum_{\mathbf{m_i} \in \mathrm{RS}} r(\mathbf{m_i})\, \eta_i$$

In general, the *steady-state probability of a given predicate on the marking*, $P(\mathbf{m})$, can be computed from the corresponding reward function:

$$r(\mathbf{m}) = \begin{cases} 1 & \text{if } P(\mathbf{m}) = \text{true} \\ 0 & \text{otherwise} \end{cases}$$

In Chapters 9, 10, and 11, the computation of the steady-state distribution η_i will be addressed, for the particular case where the marking process of the model is a Continuous Time Markov Chain.

8.5 Bibliographic Remarks

Operational analysis is a conceptually very simple way of deriving mathematical equations relating observable quantities in queueing systems. A classical work is [13]. In [12] the reader can find some nice examples of how the application of operational analysis techniques can help in explaining and providing fundamental results in queueing network analysis.

The first specific work on the operational analysis topic within the timed Petri nets context appeared in [10] and it was applied to the computation of performance bounds in [9] (see Chapter 17). Both the operational definition of performance indices and the derivation of operational laws presented in Sections 8.1 and 8.2 have been taken from those papers.

The study of the computability of the visit ratios for Petri nets and the definition of the FRT-nets class, presented in Section 8.3, were considered firstly in [2] and later in [8]. The particular cases of marked graphs, mono-T-semiflow nets, and free choice nets were previously studied in [3, 4, 5, 6, 7].

Concerning ergodicity concept and ergodic theory, the reader is referred to the literature on stochastic processes. A good introduction is the book by Karlin and Taylor [14]. With respect to the definition of performance measures as reward functions over the marking, the reader is referred to [11] or [1].

Bibliography

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* J. Wiley, 1995.

[2] J. Campos. *Performance Bounds for Synchronized Queueing Networks.* PhD thesis, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, Spain, October 1990. Research Report GISI-RR-90-20.

[3] J. Campos, G. Chiola, J. M. Colom, and M. Silva. Tight polynomial bounds for steady-state performance of marked graphs. In *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models*, pages 200–209, Kyoto, Japan, December 1989. IEEE Computer Society Press.

[4] J. Campos, G. Chiola, J. M. Colom, and M. Silva. Properties and performance bounds for timed marked graphs. *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications*, 39(5):386–401, May 1992.

[5] J. Campos, G. Chiola, and M. Silva. Properties and steady-state performance bounds for Petri nets with unique repetitive firing count vector. In *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models*, pages 210–220, Kyoto, Japan, December 1989. IEEE Computer Society Press.

[6] J. Campos, G. Chiola, and M. Silva. Ergodicity and throughput bounds of Petri nets with unique consistent firing count vector. *IEEE Transactions on Software Engineering*, 17(2):117–125, February 1991.

[7] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):1368–1382, December 1991.

[8] J. Campos and M. Silva. Structural techniques and performance bounds of stochastic Petri net models. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 352–391. Springer-Verlag, Berlin, 1992.

[9] G. Chiola, C. Anglano, J. Campos, J. M. Colom, and M. Silva. Operational analysis of timed Petri nets and application to the computation of performance bounds. In *Proceedings of the* 5^{th} *International Workshop on Petri Nets and Performance Models*, pages 128–137, Toulouse, France, October 1993. IEEE-Computer Society Press.

[10] G. Chiola, J. Campos, J. M. Colom, and M. Silva. Operational analysis of timed Petri nets. In *Proceedings of the* 16^{th} *International Symposium on Computer Performance Modelling, Measurement and Evaluation (Performance'93)*, Roma, Italy, September 1993.

[11] G. Ciardo, J. Muppala, and K. S. Trivedi. On the solution of GSPN reward models. *Performance Evaluation*, 12(4):237–253, July 1991.

[12] Y. Dallery and X. R. Cao. Operational analysis of stochastic closed queueing networks. *Performance Evaluation*, 14(1):43–61, January 1992.

[13] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, September 1978.

[14] S. Karlin and H. M. Taylor. *A First Course in Stochastic Processes*. Academic Press, New York, NY, 1975.

Part III

Markov Chain Generation

Chapter 9

EXPONENTIAL STOCHASTIC PETRI NETS

In this chapter we show how SPN systems can be converted into Markov chains and how their analysis can be performed to compute interesting performance indices. The construction of the Markov chain associated with a Stochastic Petri Net (SPN) system is described first, to set the ground for the subsequent derivation of the probabilistic model associated with a GSPN. Only SPNs with finite state space are considered in this chapter, as they yield Markov chains that are solvable with standard numerical techniques. More advanced solution methods based on matrix-geometric theory [16] will be discussed in the next chapter. Some more advanced material is discussed in the second part of the chapter where we illustrate reduction techniques that can be employed to obtain smaller reachability graphs.

9.1 The Stochastic Process Associated with a SPN

We have already discussed the motivations behind the work of several authors that led to the proposal of Stochastic Petri Nets (SPNs) [14, 15, 19]. Due to the memoryless property of the exponential distribution of firing delays, it is relatively easy to show [14, 15] that SPN systems are isomorphic to continuous

time Markov chains (CTMCs). In particular, a k-bounded SPN system can be shown to be isomorphic to a finite CTMC.

Finite State Machine and Marked Graph SPNs

This can be easily seen when the structure of the SPN is that of both a *finite state machine* (no transition has more than one input and one output place) and of a *marked graph* (no place has more than one input and one output transition) with only one token in its initial marking. In this case each place of the net univocally corresponds to a state of the net and the position of the token at a given instant of time identifies the state the net is in at that same time. Each place of the net maps into a state of the corresponding CTMC and each transition maps into an arc annotated by the rate of the corresponding firing time distribution. Moreover, if the firing times of the transitions have negative exponential distributions and if the structure of the net is that of a marked graph, the time spent in each place by the net is completely identified by the characteristics of the only transition that may withdraw the token from that place. When the net has the structure of a finite state machine, conflicts among simultaneously enabled transitions arise since several transitions may share the same input place. In this situation, several possibilities exist for the selection of the transition to fire next. A preselection policy may be applied in which one of the transitions is initially selected and then an exponentially distributed amount of time elapses before the transition actually fires. If this is the way in which the net operates, a probabilistic specification of the preselection mechanism is needed in order to construct the corresponding probabilistic model. Another possibility is that of assuming that a race takes place among the enabled transitions and that the transition associated with the activity that ends first is that selected to fire. As the race is won by one of the transitions, the others are interrupted and a choice must be made on how to deal with these partially completed activities. Several possibilities exist in this case and will be addressed in details in Chapter 10. For the time being, we can just observe that the simplest possibility is that of discarding the work associated with interrupted transitions, so that they start anew the next time they are enabled again (and a new race begins). If all the activities have negative-exponentially distributed durations, the CTMC corresponding to the SPN is obtained from the net in a straightforward manner. Again each place of the SPN maps into a state of the corresponding CTMC and each transition of the SPN maps into an arc of the CTMC annotated with the rate of the corresponding firing time distribution. Also in this case the time spent by the net in each place has a

negative-exponential distribution, but its rate is given by the sum of the firing rates of all the transitions that withdraw tokens from that place.

More complex situations arise when several tokens are allowed in the initial markings of SPNs of the types mentioned above. These are due to the fact that places non longer correspond to states of the associated probabilistic models. Moreover, in these cases the behaviours of the probabilistic models are also affected by the service selection policies from the input places as well as by the token selection policies adopted at transition firing moments.

Even the very simple case of two tokens waiting in the only input place of a transition rises a set of interesting questions that must be addressed in developing the corresponding probabilistic model. The first has to do with the speed at which the transition withdraws the tokens from its input place in this situation. Several possibilities exist. One is that of considering the input place as a waiting room: the enabled transition "serves" only one token at the time and its serving speed is not affected by the number of tokens "sitting" in the waiting room. Another is that of assuming a "multiple service" policy in which each token starts being served as soon as it arrives in the input place. This corresponds to several instances of service being simultaneously active: as many as there are tokens in the input place. Finally, a last one is that of assuming a form of *load dependency* in which the firing rate of the transition is a function of its enabling degree. In this case an additional specification must be introduced in the model to define the load dependency function associated with each transition. The second question refers to the selection of the token that is removed from the input place upon the firing of the transition. From a "classical" Petri net point of view, this selection policy is inessential since tokens do not carry any identity (except in the case of coloured nets that is of no interest in this chapter) and no tokens are "moved" throughout the net during its operation. In many applications however, it is convenient to associate a physical meaning with the tokens (e.g., customers), so that questions on their flow through the net can be answered. In these situations, when several tokens are simultaneously present in the input place of a transition, if this is assumed to operate with a single server policy, a question on the queueing policy applied to these tokens becomes interesting. The most natural policy (from a Petri net point of view) that is assumed in these situations is a *random order*. When the firing times are exponentially distributed and when the performance figures of interest are only related to the moments of the number of tokens in the input place of a transition, it is possible to show that many queueing policies yield the same results (e.g., random, FIFO, LCFS). It must however be observed that in other

cases the choice of the token selection policy may be important and that policies different from the random one must be explicitly implemented through appropriate net constructions.

SPNs with general structure

In general, the CTMC associated with a given SPN system is obtained by applying the following simple rules:

1. The CTMC state space $S = \{s_i\}$ corresponds to the reachability set $RS(\boldsymbol{m}_0)$ of the PN associated with the SPN ($\boldsymbol{m}_i \leftrightarrow s_i$).

2. The transition rate from state s_i (corresponding to marking $\boldsymbol{m}_i$) to state s_j ($\boldsymbol{m}_j$) is obtained as the sum of the firing rates of the transitions that are enabled in $\boldsymbol{m}_i$ and whose firings generate marking $\boldsymbol{m}_j$.

Based on these simple rules, it is possible to devise algorithms for the automatic construction of the infinitesimal generator (also called the state transition rate matrix) of the isomorphic CTMC, starting from the SPN description.

Assuming that all the transitions of the net operate with a single-server semantics and marking-independent speeds, and denoting with $\mathbf{Q}$ this matrix, with w_k the firing rate of T_k, and with $\boldsymbol{e_j}(\boldsymbol{m}_i) = \{h : T_h \in \boldsymbol{e}(\boldsymbol{m}_i) \wedge \boldsymbol{m}_i[T_h\rangle \boldsymbol{m}_j\}$ the set of transitions whose firings bring the net from marking $\boldsymbol{m}_i$ to marking $\boldsymbol{m}_j$, the components of the infinitesimal generator are:

$$q_{ij} = \begin{cases} \sum_{T_k \in \boldsymbol{e_j}(\boldsymbol{m}_i)} w_k & i \neq j \\ -q_i & i = j \end{cases} \tag{9.1}$$

where

$$q_i = \sum_{T_k \in \boldsymbol{e}(\boldsymbol{m}_i)} w_k \tag{9.2}$$

Let $\pi(\boldsymbol{m}_i, \tau)$ be the probability that the SPN is in marking $\boldsymbol{m}_i$ at time τ. The Chapman-Kolmogorov equations for the CTMC associated with an SPN are specified by:

$$\frac{d\pi(\boldsymbol{m}_i, \tau)}{d\tau} = \sum_{T_k \in T} q_{kj} \pi(\boldsymbol{m}_k, \tau) \tag{9.3}$$

In matrix notation this becomes

$$\frac{d\boldsymbol{\pi}(\tau)}{d\tau} = \boldsymbol{\pi}(\tau)\boldsymbol{Q} \tag{9.4}$$

whose solution can be formally written as

$$\boldsymbol{\pi}(\tau) = \boldsymbol{\pi}(0) e^{\boldsymbol{Q}\tau} \tag{9.5}$$

where $\boldsymbol{\pi}(0)$ is the probability of the initial distribution (in our case we usually have $\pi_i(0) = 1$ if $\boldsymbol{m}_i = \boldsymbol{m}_0$ and $\pi_i(0) = 0$ otherwise) and $e^{\boldsymbol{Q}\tau}$ is the matrix exponentiation defined by

$$e^{\boldsymbol{Q}\tau} = \sum_{k=0}^{\infty} \frac{(\boldsymbol{Q}\tau)^k}{k!} \tag{9.6}$$

In this chapter we consider only SPNs originating homogeneous and ergodic CTMC. A k-bounded SPN system is said to be ergodic if it generates an ergodic CTMC; it is possible to show that a SPN system is ergodic if $\boldsymbol{m}_0$, the initial marking, is a home state (see Section ??).

If the SPN is ergodic, the steady-state probability distribution on markings of the SPN exists and is defined as the limit $\boldsymbol{\pi} = \lim_{\tau \to \infty} \boldsymbol{\pi}(\tau)$. Its value can be computed solving the usual system of linear equations:

$$\begin{cases} \boldsymbol{\pi}\ \boldsymbol{Q} = \mathbf{0} \\ \\ \boldsymbol{\pi}\ \mathbf{1}^{\mathrm{T}} = 1 \end{cases} \tag{9.7}$$

where $\mathbf{0}$ is a vector of the same size as $\boldsymbol{\pi}$ and with all its components equal to zero and $\mathbf{1}^{\mathrm{T}}$ is a vector (again of the same size as $\boldsymbol{\pi}$) with all its components equal to one, used to enforce the normalization condition typical of all probability distributions.

To keep the notation simple, in the rest of this chapter we will use $\pi_i(\tau)$ and π_i instead of $\pi(\boldsymbol{m}_i, \tau)$ and $\pi(\boldsymbol{m}_i)$ to denote the transient and steady state probabilities of marking $\boldsymbol{m}_i$. The *sojourn time* is the time spent by the PN system in a given marking M. As we already observed, the fact that a CTMC can be associated with the SPN system ensures that the sojourn time in the ith marking is exponentially distributed with rate q_i. The pdf of the sojourn time in a marking corresponds to the pdf of the minimum among the firing times of the transitions enabled in the same marking; it thus follows that the probability that a given transition $T_k \in \boldsymbol{e}(\boldsymbol{m}_i)$ fires (first) in marking $\boldsymbol{m}_i$ has the expression:

$$P\{T_k | \boldsymbol{m}_i\} = \frac{w_k}{q_i} \quad . \tag{9.8}$$

Using the same argument, we can observe that the average sojourn time in marking $\boldsymbol{m}_i$ is given by the following expression:

$$SJ_i = \frac{1}{q_i} \quad . \tag{9.9}$$

9.1.1 SPN performance indices

The steady-state distribution $\boldsymbol{\pi}$ is the basis for a quantitative evaluation of the behaviour of the SPN that is expressed in terms of performance indices. These results can be computed using a unifying approach in which proper index functions (also called *reward functions*) are defined over the markings of the SPN and an average reward is derived using the steady-state probability distribution of the SPN. Assuming that $r(\boldsymbol{m})$ represents one of such reward functions, the average reward can be computed using the following weighted sum:

$$R = \sum_{\boldsymbol{m}_i \in RS(\boldsymbol{m}_0)} r(\boldsymbol{m}_i)\ \pi_i \tag{9.10}$$

Different interpretations of the reward function can be used to compute different performance indices. In particular, the following quantities can be computed using this approach.

(1) The *probability of a particular condition* of the SPN. Assuming that *condition* $\Upsilon(\boldsymbol{m})$ is *true* only in certain markings of the PN, we can define the following reward function:

$$r(\boldsymbol{m}) = \begin{cases} 1 & \Upsilon(\boldsymbol{m}) = true \\ & \\ 0 & otherwise \end{cases} \tag{9.11}$$

The desired probability $P\{\Upsilon\}$ is then computed using equation 9.10. The same result can also be expressed as:

$$P\{\Upsilon\} = \sum_{\boldsymbol{m}_i \in A} \pi_i \tag{9.12}$$

where $A = \{\boldsymbol{m}_i \in RS(\boldsymbol{m}_0) : \Upsilon(\boldsymbol{m}_i) = true\}$.

(2) The *expected value of the number of tokens in a given place.* In this case the reward function $r(\boldsymbol{m})$ is simply the value of the marking of that place (say place j):

$$r(\boldsymbol{m}) = n \text{ iff } m(p_j) = n \tag{9.13}$$

Again this is equivalent to identifying the subset $A(j,n)$ of $RS(\boldsymbol{m}_0)$ for which the number of tokens in place p_j is n ($A(j,n) = \{\boldsymbol{m}_i \in RS(\boldsymbol{m}_0) : \boldsymbol{m}_i(p_j) = n\}$); the expected value of the number of tokens in p_j is given by:

$$\boldsymbol{e}[m(p_j)] = \sum_{n>0} [n\ P\{A(j,n)\}] \tag{9.14}$$

where the sum is obviously limited to values of $n \leq k$, if the place is k-bounded.

(3) The *mean number of firings per unit of time of a given transition.* Assume that we want to compute the firing frequency of transition T_j (the *throughput* of T_j); observing that a transition may fire only when it is enabled, we have that the reward function assumes the value w_j in every marking that enables T_j:

$$r(\boldsymbol{m}) = \begin{cases} w_j & T_j \in \boldsymbol{e}(\boldsymbol{m}) \\ 0 & otherwise \end{cases} \tag{9.15}$$

The same quantity can also be computed using the more traditional approach of identifying the subset A_j of $RS(\boldsymbol{m}_0)$ in which a given transition T_j is enabled ($A_j = \{\boldsymbol{m}_i \in RS(\boldsymbol{m}_0) : T_j \in \boldsymbol{e}(\boldsymbol{m}_i)\}$). The mean number of firings of T_j per unit of time is then given by:

$$f_j = \sum_{M_i \in A_j} w_j \ \pi_i \tag{9.16}$$

These results show that Petri nets can be used not only as a formalism for describing the behaviour of distributed/parallel systems and for assessing their qualitative properties, but also as a tool for computing performance indices that allow the efficiency of these systems to be evaluated.

To illustrate the details of this last analysis step, a simple example is presented in the following subsection, with the explicit derivation of the CTMC infinitesimal generator, of the steady-state probability distribution of the different markings, and of some performance indices.

9.1.2 An example SPN system

Consider a simple shared memory multiprocessor system in which two processors must occasionally access a common shared memory. All the processors are assumed to have identical behaviours characterized by a cyclic sequence of local activities, followed by requests for the common memory and by accesses of the common memory. All these actions last for certain amount of times and additional delays are experienced by a processor that requets to access the common memory while it is busy serving the other processor. Assuming that all timings considered in the example have negative exponential distributions, the SPN model of this system is depicted in Fig. 9.1. The net comprises seven places and six timed transitions with single-server semantics; the parameters of the different firing time distributions are contained in Table 9.1.

Starting from the initial marking shown in Fig. 9.1, in which the two processors are both in a locally active state and the memory is idle (p_{act1}+p_{act2}+p_{idle}),

Table 9.1: Specification of the transitions of the SPN of Fig. 9.1

transition	rate	semantics
T_{req1}	λ_1	single-server
T_{req2}	λ_2	single-server
T_{str1}	α_1	single-server
T_{str2}	α_2	single-server
T_{end1}	μ_1	single-server
T_{end2}	μ_2	single-server

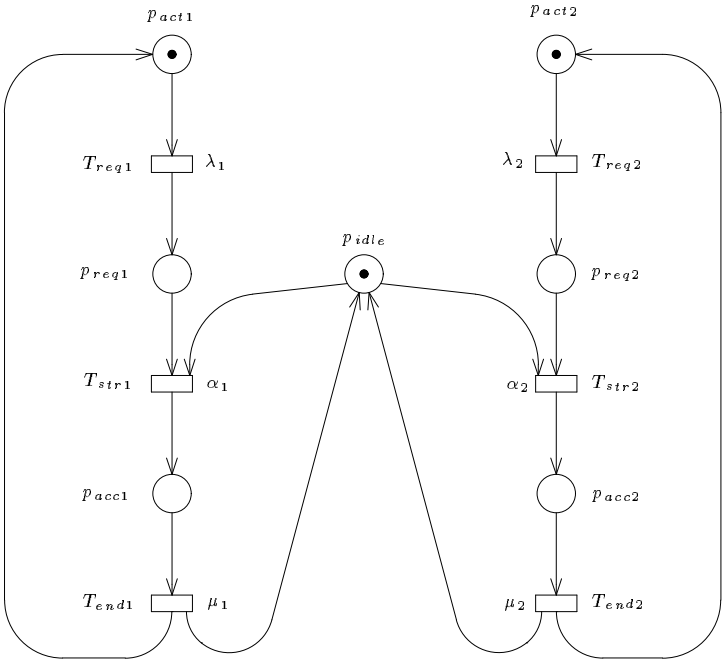


Figure 9.1: The SPN description of shared memory system

a possible evolution of the SPN marking that focuses on the processing cycle of processor 1, may be the following. Processor 1 works locally for an exponentially distributed random amount of time with average $1/\lambda_1$, and then requests an access to the common memory. Transition T_{req1} fires, and the token contained in p_{act1} is removed while a token is added in p_{req1}. Since the common memory is available (place p_{idle} is marked), the acquisition of the memory starts immediately and takes an average of $1/\alpha_1$ units of time to complete; this is represented by the firing of transition T_{str1} whose associated delay has a negative-

Table 9.2: Reachability set of SPN of Fig. 9.1

$\boldsymbol{m}_0 =$	p_{act1}	+			p_{idle}	+ p_{act2}		
$\boldsymbol{m}_1 =$		p_{req1}	+		p_{idle}	+ p_{act2}		
$\boldsymbol{m}_2 =$			p_{acc1}	+		p_{act2}		
$\boldsymbol{m}_3 =$			p_{acc1}	+			p_{req2}	
$\boldsymbol{m}_4 =$		p_{req1}	+		p_{idle}	+	p_{req2}	
$\boldsymbol{m}_5 =$	p_{act1}	+			p_{idle}	+	p_{req2}	
$\boldsymbol{m}_6 =$	p_{act1}	+						p_{acc2}
$\boldsymbol{m}_7 =$		p_{req1}	+					p_{acc2}

exponential distribution with rate α_1; when transition T_{str1} fires, one token is removed from place p_{req1} and another token is deposited into place p_{acc1}, where it stays for the entire time required by the first processor to access the common memory. Such a time lasts on the average $1/\mu_1$ units, and ends when transition T_{end1} fires returning the net to its initial state. Obviously, a similar processing cycle is possible for processor 2 and many interleavings between the activities of the two processors may be described by the evolution of the net.

A conflict exists in the behaviour of this system when both processors want to simultaneously access the common memory, i.e., when transitions T_{str1} and T_{str2} are both enabled. According to equation (9.1), in this situation, transition T_{str1} fires with probability:

$$P\{T_{str1}\} = \frac{\alpha_1}{\alpha_1 + \alpha_2} \tag{9.17}$$

whereas transition T_{str2} fires with probability:

$$P\{T_{str2}\} = \frac{\alpha_2}{\alpha_1 + \alpha_2} \tag{9.18}$$

Notice that when the two transitions T_{str1} and T_{str2} are both enabled in a given marking M, the speed at which the PN model exits from that marking is the sum of the individual speeds of the two transitions and the conflict is actually resolved only at the moment the first of them fires.

The reachability set of the SPN system of Fig. 9.1 is depicted in Table 9.2. The reachability graph is shown in Fig. 9.2, while the corresponding CTMC state transition rate diagram is presented in Fig. 9.3.

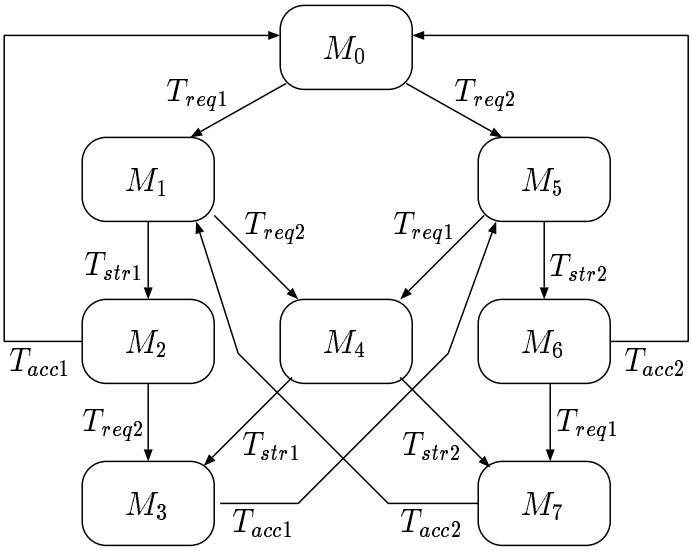


Figure 9.2: The reachability graph of the SPN system in Fig. 9.1

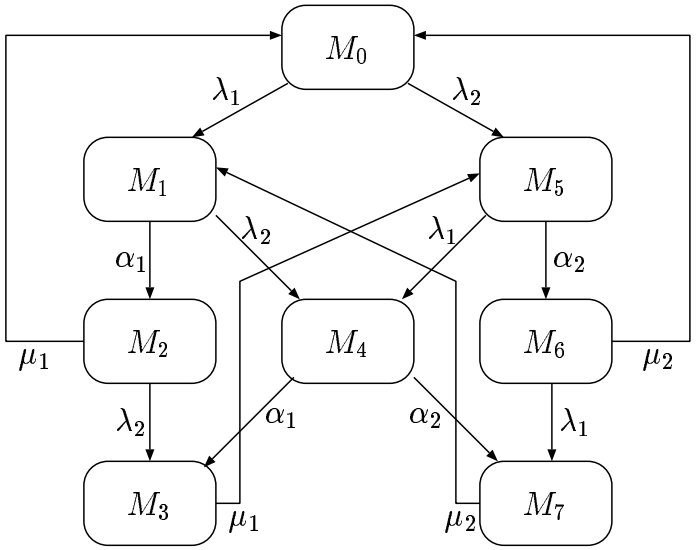


Figure 9.3: The state transition rate diagram of the Markov chain associated with the SPN in Fig. 9.1

9.2 The Stochastic Process Associated with a GSPN

In Chapter 3 we described the qualitative effect that immediate transitions have on the behaviour of a GSPN system. GSPNs adopt the same firing policy of SPNs; when several transitions are enabled in the same marking, the probabilistic choice of the transition to fire next depends on parameters that are associated with these same transitions and that are not functions of time. The general expression for the probability that a given (timed or immediate) transition t_k, enabled in marking $\boldsymbol{m}_i$, fires is:

$$P\{t_k|\boldsymbol{m}_i\} = \frac{w_k}{q_i} \tag{9.19}$$

where q_i is the quantity defined by equation (9.2). Equation (9.19) represents the probability that transition t_k fires first, and is identical to equation (9.8) for SPNs, with a difference in the meaning of the parameters w_k. When the marking is vanishing, the parameters w_k are the weights of the immediate transitions enabled in that marking and define the selection policy used to make the choice. When the marking is tangible, the parameters w_k of the timed transitions enabled in that marking are the rates of their associated negative exponential distributions. The average sojourn time in vanishing markings is zero, while the average sojourn time in tangible markings is given by equation (9.9).

Observing the evolution of the GSPN system, we can notice that the distribution of the sojourn time in an arbitrary marking can be expressed as a composition of negative exponential and deterministically zero distributions: we can thus recognize that the marking process $\{\mathcal{M}(\tau), \tau \geq 0\}$ is a semi-Markov stochastic process.

When several immediate transitions are enabled in the same vanishing marking, deciding which transition to fire first makes sense only in the case of conflicts. If these immediate transitions do not "interfere" they could be fired simultaneously and the choice of firing only one of them at a time becomes an operational rule of the model that hardly relates with the actual characteristics of the real system. In this case the selection is inessential from the point of view of the overall behaviour of the net. The concept of ECS introduced in Chapter 2 represents a basic element for identifying these situations and for making sure that both the specification of the model and its operational rules are consistent with the actual behaviour of the system we want to analyse.

Assuming, as we have already pointed out in the previous chapter, that the

GSPN is not confused (see Section ?? for a definition), the computation of the ECSs of the net corresponds to partitioning the set of immediate transitions into equivalence classes such that transitions of the same partition may be in conflict among each other in possible markings of the net, while transitions of different ECSs behave in a truly concurrent manner.

When transitions belonging to the same ECS are the only ones enabled in a given marking, one of them (say transition t_k) is selected to fire with probability:

$$P\{t_k|\boldsymbol{m}_i\} = \frac{w_k}{\omega_k(\boldsymbol{m}_i)} \tag{9.20}$$

where $\omega_k(\boldsymbol{m}_i)$ is the weight of ECS(t_k) in marking $\boldsymbol{m}_i$ and is defined as follows:

$$\omega_k(\boldsymbol{m}_i) = \sum_{t_j \in [\mathrm{ECS}_{(t_k)} \wedge \boldsymbol{e}(\boldsymbol{m}_i)]} w_j \tag{9.21}$$

Within the ECS we may have transitions that are in direct as well as in indirect conflicts. This means that the distributions of the firing selection probabilities computed for different markings may assign different probabilities of firing to the same transition in different markings. Equation (9.20) however ensures that if we have two transitions (say transitions t_i and t_j), both enabled in two different markings (say markings $\boldsymbol{m}_r$ and $\boldsymbol{m}_s$), the ratios between the firing probabilities of these two transitions in these two markings remain constant and in particular equal to the ratio between the corresponding weights assigned at the moment of the specification of the model.

During the evolution of a GSPN, it may happen that several ECSs are simultaneously enabled in a vanishing marking. According to the usual firing mechanism of Petri nets, we should select the transition to fire by first non-deterministically choosing one of the ECSs and then a transition within it. The assumption that the GSPN is not confused guarantees that the way in which the choice of the ECS is performed is irrelevant with respect to the associated stochastic process. One possibility is that of computing the weight of the ECS by adding the parameters of all the enabled transitions that belong to that ECS and of using this weight to select the ECS with a method suggested by equation (9.19). A simple derivation shows that the use of this method implies that the selection of the immediate transition to be fired in a vanishing marking can be performed with the general formula (9.19) that was originally derived for timed transitions (and thus for tangible markings) only [1, 9]. Moreover, it is possible to show that if we consider the probabilities associated with the many different sequences of immediate transitions whose firings lead from a given vanishing marking to a target tangible one, they turn out to be all equal [1].

Table 9.3: Specification of the transitions of the SPN of Fig. 9.4

transition	weight	priority	ECS
t_0	w_0	1	-
t_1	β	1	-
t_2	α	1	-

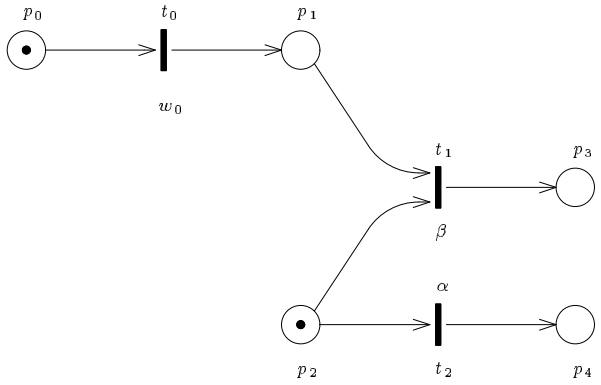


Figure 9.4: Example of a simple confused GSPN system

This last property strongly depends on the absence of confusion in the GSPN; the fact that the presence of confused subnets of immediate transitions within a GSPN is an undesirable feature of the model can also be explained considering the following example.

Suppose that a subnet is identified as confused using the structural confusion condition of Section ??. Suppose also that the subnet has the structure depicted Chapter 2 and in Fig. 9.4. Given the initial marking $\boldsymbol{m}_0 = (p_0 + p_2)$ and the parameters listed in Table 9.3, markings $\boldsymbol{m}_1 = p_3$ and $\boldsymbol{m}_2 = (p_1 + p_4)$ are reached with total probabilities

$$P\{\boldsymbol{m}_0, \boldsymbol{m}_1\} = \frac{w_0}{(w_0 + \alpha)} \frac{\beta}{(\alpha + \beta)} \qquad P\{\boldsymbol{m}_0, \boldsymbol{m}_2\} = \frac{\alpha}{(w_0 + \alpha)} \quad .$$

From these expressions we can see that, although the picture suggests transitions t_0 and t_2 as concurrent, and transitions t_1 and t_2 as members of a conflict set, the first choice of firing either t_0 or t_2 is actually crucial for the possibility of reaching the desired markings. In this case the value assigned by the analyst to w_0 becomes fundamental for the quantitative evaluation of the model.

Table 9.4: Specification of the transitions of the SPN of Fig. 9.5

transition	weight	priority	ECS
t_0	w_0	1	-
t_1	β	1	-
t_2	α	1	-
t_a	w_a	1	-

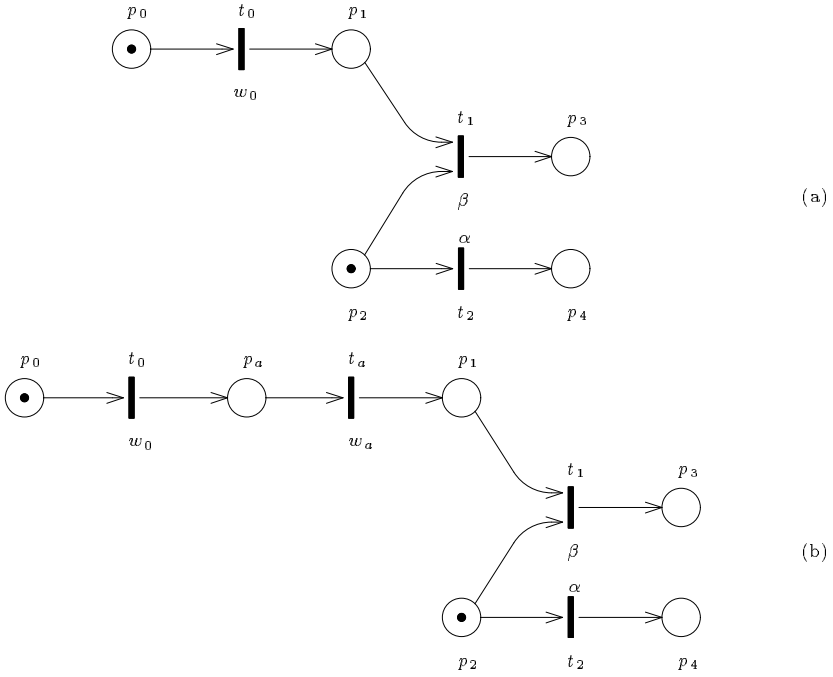


Figure 9.5: Comparison of two simple confused GSPN systems

Much more intriguing however is the behaviour of the subnet in Fig. 9.5(b) that differs from that of Fig. 9.5(a) for the simple addition of place p_a and transition t_a between transition t_0 and place p_1. Given the initial marking $\boldsymbol{m}_0 = (p_0 + p_2)$ and the parameters listed in Table 9.4, markings $\boldsymbol{m}_1 = p_3$ and $\boldsymbol{m}_2 = (p_1 + p_4)$ are reached in the case of the subnet of Fig. 9.5(b) with total probabilities

$$P\{\boldsymbol{m}_0, \boldsymbol{m}_1\} = \frac{w_0}{(w_0 + \alpha)} \frac{w_a}{(w_a + \alpha)} \frac{\beta}{(\alpha + \beta)}$$

and

$$P\{\boldsymbol{m}_0, \boldsymbol{m}_2\} = \frac{\alpha}{(w_0 + \alpha)} + \frac{w_0}{(w_0 + \alpha)} \frac{\alpha}{(w_a + \alpha)} \quad .$$

From a modelling point of view, there are good reasons to consider the two subnets of Figs. 9.5(a) and 9.5(b) equivalent since the sequence of immediate actions represented by t_0 and t_a of Fig. 9.5(b) should be reducible to transition t_0 of Fig. 9.5(a) without affecting the behaviour of the model. Instead, the difference among the total probabilities that we have just computed shows that, in the case of confused models, the trivial action of splitting an atomic (and instantaneous) action in two has drastic effects not only on the graphical description of the model, but also (and more important) on the values of the results obtained from its quantitative evaluation.

9.2.1 Marking dependency

The firing times and the weights that we have considered so far were assumed to be independent of the marking of the GSPN system. In principle, however, it is possible to work with transition parameters that are marking-dependent as pointed out at the beginning of this chapter. When one or more timed transitions are enabled in a given marking, we can compute the distribution of the sojourn time in that marking, as well as the probability of the transition that fires first, using negative exponential distributions whose firing rates may depend on that specific marking. Similarly, the selection of the immediate transition that fires in a vanishing marking enabling several immediate transitions can be computed using weighting factors that may be marking-dependent. In all these cases, equations (9.9), (9.19), (9.20), and (9.21) can be generalized assuming that all the parameters are functions of the marking for which they are computed.

In practice, the generality allowed by this extension (which was assumed by the original GSPN proposal [2]) is in contrast with the claim of GSPNs as a high-level language for the description of complex systems. In fact, while the construction of a GSPN model requires a local view of the behaviour of the real system, the specification of (general) marking-dependent parameters requires the analyst to be aware of the possible (global) states of the system. Moreover, a dependency of the weights of conflicting immediate transitions on the marking of places that are not part of the input set of any of the transitions comprised in their ECS could lead to a new form of "confusion" that is not captured by the definitions contained in Chapter 2 and discussed in the previous section.

For this reason, a restriction of the type of marking-dependency allowed in GSPN models was informally proposed in [9]. A definition of marking dependency that satisfies these restrictions can be obtained by allowing the specifi-

cation of marking dependent parameters as the product of a nominal rate (or weight in the case of immediate transitions) and of a dependency function defined in terms of the marking of the places that are connected to a transition through its input and inhibition functions.

Denote with $\boldsymbol{m}_{/t}$ the restriction of a generic marking M to the input and inhibition sets of transition t:

$$\boldsymbol{m}_{/t} \;=\; \bigcup\nolimits_{p \in (\bullet t \cup \circ t)} m(p) \tag{9.22}$$

Let $f(\boldsymbol{m}_{/t})$ be the marking dependency function that assumes positive values every time transition t is enabled in M; using this notation, the marking dependent parameters may be defined in the following manner:

$$\begin{cases} \mu_i(\boldsymbol{m}) \;=\; f(\boldsymbol{m}_{/T_i})\; w_i & \text{in the case of firing rates} \\ \\ \omega_j(\boldsymbol{m}) \;=\; f(\boldsymbol{m}_{/t_j})\; w_j & \text{in the case of weights} \end{cases} \tag{9.23}$$

Multiple-servers and infinite-servers can be represented as special cases of timed transitions with marking dependent firing rates that are consistent with this restriction. In particular, a timed transition T_i with a negative-exponential delay distribution with parameter w_i and with an infinite-server policy, has a marking dependency function of the following form:

$$f(\boldsymbol{m}_{/T_i}) \;=\; ED(T_i, \boldsymbol{m}) \tag{9.24}$$

where $ED(T_i, \boldsymbol{m})$ is the enabling degree of transition T_i in marking M (see Section ??). Similarly, a timed transition T_i with multiple-server policy of degree K has a marking dependency function defined in the following way:

$$f(\boldsymbol{m}_{/T_i}) \;=\; min\;(ED(T_i, \boldsymbol{m}),\; K) \tag{9.25}$$

Other interesting situations can be represented using the same technique, Fig. 9.6 depicts two such cases. In Fig. 9.6(a), we have a situation of two competing infinite servers: transition T_1 fires with rate $w_1 m(p_1)$ if place p_0 is marked; similarly for transition T_2. Obviously both transitions are interrupted when the first of the two fires removing the token from place p_0.

Using $\delta(x)$ to represent the following step function:

$$\delta(x) \;=\; \begin{cases} 0 & x \;=\; 0 \\ \\ 1 & x \;>\; 0 \end{cases} \tag{9.26}$$

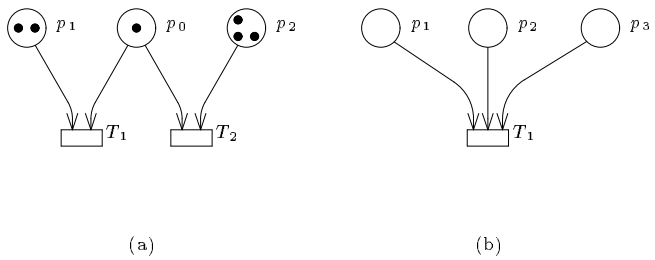


Figure 9.6: Examples of complex marking dependency situations

we can define the marking dependency function as follows:

$$\begin{cases} f(\boldsymbol{m}_{/T_1}) \;=\; \delta(ED(T_1,\boldsymbol{m}))m(p_1) \\ \qquad\qquad \text{and similarly} \\ f(\boldsymbol{m}_{/T_2}) \;=\; \delta(ED(T_2,\boldsymbol{m}))m(p_2) \end{cases} \tag{9.27}$$

Fig. 9.6(b) represents instead a server whose speed depends on a linear combination of the markings of all its input places. In this case the marking dependency function may assume the form:

$$f(\boldsymbol{m}_{/T_1}) \;=\; \delta(ED(T_1,\boldsymbol{m})) \sum_{p\in {}^\bullet T_1} \alpha_p m(p) \qquad \alpha_p \geq 0, \quad \sum_{p\in {}^\bullet T_1} \alpha_p \;=\; 1 \tag{9.28}$$

9.3 Numerical Solution of GSPN Systems

The stochastic process associated with a k-bounded GSPN system with $\boldsymbol{m}_0$ as its home state can be classified as a finite state space, stationary (homogeneous), irreducible, and continuous-time semi-Markov process.

Semi-Markov processes can be analysed identifying an embedded (discrete-time) Markov chain that describes the transitions from state to state of the process. In the case of GSPNs, the embedded Markov chain (EMC) can be recognized disregarding the concept of time and focusing the attention on the set of states of the semi-Markov process. The specifications of a GSPN system are sufficient for the computation of the transition probabilities of such a chain.

Let RS, TRS, and VRS indicate the state space (the reachability set), the set of tangible states (or markings) and the set of vanishing markings of the stochastic process, respectively. The following relations hold among these sets:

$$RS = TRS \bigcup VRS, \qquad TRS \bigcap VRS = \emptyset.$$

The transition probability matrix $\boldsymbol{U}$ of the EMC can be obtained from the specification of the model using the following expression:

$$u_{ij} = \frac{\sum_{T_k \in \boldsymbol{e_j}(\boldsymbol{m}_i)} w_k}{q_i} \tag{9.29}$$

Assuming for the sake of simplicity, that all enabled transitions produce a different change of marking upon their firing, the sum contained in equation (9.29) reduces to a single element. In this way, except for the diagonal elements of matrix $\boldsymbol{U}$, all the other transition probabilities of the EMC can be computed using equation (9.19) independently of whether the transition to be considered is timed or immediate, according to the discussion contained at the end of Section 9.2.

By ordering the markings so that the vanishing ones correspond to the first entries of the matrix and the tangible ones to the last, the transition probability matrix $\boldsymbol{U}$ can be decomposed in the following manner:

$$\boldsymbol{U} = \boldsymbol{A} + \boldsymbol{B} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{D} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \boldsymbol{E} & \boldsymbol{F} \end{bmatrix} \tag{9.30}$$

The elements of matrix $\boldsymbol{A}$ correspond to changes of markings induced by the firing of immediate transitions; in particular, those of submatrix $\boldsymbol{C}$ are the probabilities of moving from vanishing to vanishing markings, while those of $\boldsymbol{D}$ correspond to transitions from vanishing to tangible markings. Similarly, the elements of matrix $\boldsymbol{B}$ correspond to changes of markings caused by the firing of timed transitions: $\boldsymbol{E}$ accounts for the probabilities of moving from tangible to vanishing markings, while $\boldsymbol{F}$ comprises the probabilities of remaining within tangible markings.

Indicating with $\boldsymbol{\psi}(n)$ the probability distribution of the EMC at step n (i.e., after n (state-) transitions performed by the EMC), we can compute this quantity using the following expression

$$\boldsymbol{\psi}(n) = \boldsymbol{\psi}(0)\boldsymbol{U}^n \tag{9.31}$$

where, as usual, $\boldsymbol{\psi}(0)$ represents the initial distribution of the EMC. The steady-state probability distribution $\boldsymbol{\psi}$ can be obtained as the solution of the system of linear equations

$$\begin{cases} \boldsymbol{\psi} &= \boldsymbol{\psi}\, \boldsymbol{U} \\ \\ \boldsymbol{\psi}\, \mathbf{1}^{\mathrm{T}} &= 1 \end{cases} \tag{9.32}$$

The steady-state probability distribution of the EMC, can be interpreted itself in terms of numbers of (state-) transitions performed by the EMC. In fact, $1/\psi_i$ is the mean recurrence time for state s_i (marking $\boldsymbol{m}_i$) measured in number of transition firings. The steady-state probability distribution of the stochastic process associated with the GSPN system is thus obtained by weighting each entry ψ_i with the sojourn time of its corresponding marking SJ_i and by normalizing the whole distribution.

The solution method outlined so far, is computationally acceptable whenever the size of the set of vanishing markings is small (compared with the size of the set of tangible markings). However, this method requires the computation of the steady-state probability of each vanishing marking that does not increase the information content of the final solution since the time spent in these markings is known to be null. Moreover, vanishing markings not only require useless computations, but, by enlarging the size of the transition probability matrix $\boldsymbol{U}$, tend to make the computation of the solution more expensive and in some cases even impossible to obtain.

In order to restrict the solution to quantities directly related with the computation of the transient and steady-state probabilities of tangible markings, we must reduce the model by computing the total transition probabilities among tangible markings only, thus identifying a Reduced EMC (REMC).

To illustrate the method of reducing the EMC by removing the vanishing markings, consider first the example of Fig. 9.7. This system contains two free-choice conflicts corresponding to transitions T_1 , T_2, and t_1, t_2, respectively. The transition characteristics are specified in Table 9.5. From the initial marking $\boldsymbol{m}_i = p_1$, the system can move to marking $\boldsymbol{m}_j = p_3$ following two different paths. The first corresponds to the firing of transition T_1, that happens with probability $\frac{\mu_1}{(\mu_1+\mu_2)}$, and that leads to the desired (target) marking $\boldsymbol{m}_j$ in one step only. The second corresponds to selecting transition T_2 to fire first, followed by transition t_1. The first of these two events happens with probability $\frac{\mu_2}{(\mu_1+\mu_2)}$, and the second with probability $\frac{\alpha}{(\alpha+\beta)}$. The total probability of this second path from $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$ amounts to $\frac{\mu_2}{(\mu_1+\mu_2)}\frac{\alpha}{(\alpha+\beta)}$. Notice that firing transition T_2 followed by transition t_2 would lead to a different marking (in this case

Table 9.5: Specification of the transitions of the SPN of Fig. 9.7

transition	rate	semantics
T_1	μ_1	single-server
T_2	μ_2	single-server

transition	weight	priority	ECS
t_1	α	1	1
t_2	β	1	1

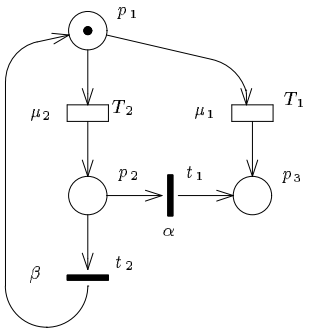


Figure 9.7: A GSPN system with multiple paths between tangible markings

the initial one). Firing transition T_2 leads the system into an intermediate (vanishing) marking $\boldsymbol{m}_r$. The total probability of moving from marking $\boldsymbol{m}_i$ to marking $\boldsymbol{m}_j$ is thus in this case:

$$u'_{ij} = \frac{\mu_1}{(\mu_1+\mu_2)} + \frac{\mu_2}{(\mu_1+\mu_2)}\,\frac{\alpha}{(\alpha+\beta)} \tag{9.33}$$

In general, upon the exit from a tangible marking, the system may "walk" through several vanishing markings before ending in a tangible one. To make the example more interesting, and to illustrate more complex cases, let us modify the net of Fig. 9.7 as depicted in Fig. 9.8 and specified in Table 9.6. In this new case the system can move from marking $\boldsymbol{m}_1 = p_1$ to marking $\boldsymbol{m}_2 = p_2$ following four different paths. The first corresponds again to firing transition T_1 and thus to a direct move from the initial marking to the target one. This happens with probability $\frac{\mu_1}{(\mu_1+\mu_2+\mu_3)}$. The second path corresponds to firing transition T_2 followed by t_1 (total probability $\frac{\mu_2}{(\mu_1+\mu_2+\mu_3)}\,\frac{\alpha}{(\alpha+\beta)}$); the third path corresponds to firing transition T_3 followed by transition t_4 (total probability $\frac{\mu_3}{(\mu_1+\mu_2+\mu_3)}\,\frac{\gamma}{(\gamma+\delta)}$). Finally, the last path corresponds to firing transition T_3 followed by transition t_3 and then by transition t_1 which happens with probability $\frac{\mu_3}{(\mu_1+\mu_2+\mu_3)}\,\frac{\delta}{(\gamma+\delta)}\,\frac{\alpha}{(\alpha+\beta)}$.

In this case the total probability of moving from marking $\boldsymbol{m}_i$ to marking

Table 9.6: Specification of the transitions of the SPN of Fig. 9.8

transition	rate	semantics
T_1	μ_1	single-server
T_2	μ_2	single-server
T_3	μ_3	single-server

transition	weight	priority	ECS
t_1	α	1	1
t_2	β	1	1
t_3	δ	1	1
t_4	γ	1	1

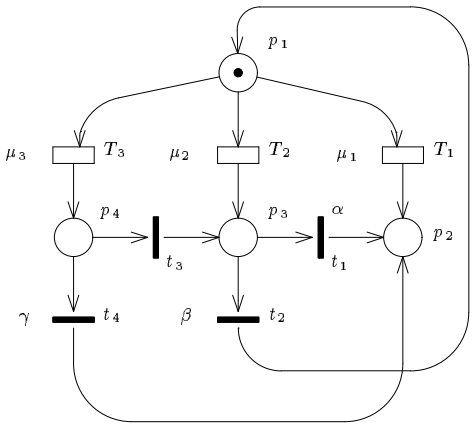


Figure 9.8: A GSPN system with several multiple paths between tangible markings

$\boldsymbol{m}_j$ becomes:

$$u'_{ij} = \frac{\mu_1}{(\mu_1+\mu_2+\mu_3)} + \frac{\mu_2}{(\mu_1+\mu_2+\mu_3)}\frac{\alpha}{(\alpha+\beta)} + \frac{\mu_3}{(\mu_1+\mu_2+\mu_3)}\frac{\gamma}{(\gamma+\delta)} + \frac{\mu_3}{(\mu_1+\mu_2+\mu_3)}\frac{\alpha}{(\alpha+\beta)}\frac{\delta}{(\gamma+\delta)} \quad (9.34)$$

In general, recalling the structure of the $\boldsymbol{U}$ matrix, a direct move from marking $\boldsymbol{m}_i$ to marking $\boldsymbol{m}_j$ corresponds to a non-zero entry in block $\boldsymbol{F}$ ($f_{ij} \neq 0$), while a path from $\boldsymbol{m}_i$ to $\boldsymbol{m}_j$ via two intermediate vanishing markings corresponds to the existence of

1. a non-zero entry in block $\boldsymbol{E}$ corresponding to a move from $\boldsymbol{m}_i$ to a generic intermediate marking $\boldsymbol{m}_r$;

2. a non-zero entry in block $\boldsymbol{C}$ from this generic state $\boldsymbol{m}_r$ to another arbitrary vanishing marking $\boldsymbol{m}_s$;

3. a corresponding non-zero entry in block $\boldsymbol{D}$ from $\boldsymbol{m}_s$ to $\boldsymbol{m}_j$.

These informal considerations are precisely captured by the following formula:

$$u'_{ij} \;=\; f_{ij} \;+\; \sum_{r:\boldsymbol{m}_r \in VRS} e_{ir}\, P\,\{r \to s\,\}\; d_{sj} \tag{9.35}$$

where $P\{r \to s\}\; d_{sj}$ is the probability that the net moves from vanishing marking $\boldsymbol{m}_r$ to tangible marking $\boldsymbol{m}_j$ in an arbitrary number of steps, following a path through vanishing markings only.

In order to provide a general and efficient method for the computation of the state transition probability matrix $\boldsymbol{U}'$ of the REMC, we can observe that equation (9.35) can be rewritten in matrix notation in the following form:

$$\boldsymbol{U}' \;=\; \boldsymbol{F} \;+\; \boldsymbol{E}\;\boldsymbol{G}\;\boldsymbol{D} \tag{9.36}$$

where each entry g_{rs} of matrix $\boldsymbol{G}$ represents the probability of moving from vanishing marking $\boldsymbol{m}_r$ to vanishing marking $\boldsymbol{m}_s$ in any number of steps, but without hitting any intermediate tangible marking. Observing that each entry c^n_{rs} of matrix $\boldsymbol{C}^n$ represents the probability of moving from vanishing marking $\boldsymbol{m}_r$ to vanishing marking $\boldsymbol{m}_s$ in exactly n steps without hitting any intermediate tangible marking, we can conclude that

$$\boldsymbol{G} \;=\; \sum_{n=0}^{\infty} \boldsymbol{C}^n$$

In the computation of $\boldsymbol{C}^n$, two possibilities may arise. The first corresponds to the situation in which there are no loops among vanishing markings. This means that for any vanishing marking $\boldsymbol{m}_r \in VRS$ there is a value n_{0r} such that any sequence of transition firings of length $n \geq n_{0r}$ starting from such marking must reach a tangible marking $\boldsymbol{m}_j \in TRS$. In this case

$$\exists n_0 \;:\quad \forall\;\; n \geq n_0 \qquad \boldsymbol{C}^n \;=\; 0$$

and

$$\boldsymbol{G} \;=\; \sum_{k=0}^{\infty} \boldsymbol{C}^k \;=\; \sum_{k=0}^{n_0} \boldsymbol{C}^k$$

The second corresponds to the situation in which there are possibilities of loops among vanishing markings, so that the GSPN may remain "trapped" within a set of vanishing markings. In this case the irreducibility property of the semi-Markov process associated with the GSPN system ensures that the following results hold [20]:

$$\lim_{n \to \infty} \boldsymbol{C}^n \;=\; 0$$

so that

$$\boldsymbol{G} \;=\; \sum_{k=0}^{\infty} \boldsymbol{C}^k \;=\; [\boldsymbol{I} \;-\; \boldsymbol{C}]^{-1}.$$

We can thus write (see [3, 2] for details):

$$\boldsymbol{H} \;=\; \begin{cases} \left(\sum_{k=0}^{n_0} \boldsymbol{C}^k \right) \boldsymbol{D} & \textit{no loops among vanishing states} \\ [\boldsymbol{I} \;-\; \boldsymbol{C}]^{-1} \boldsymbol{D} & \textit{loops among vanishing states} \end{cases}$$

from which we can conclude that an explicit expression for the desired total transition probability among any two tangible markings is:

$$u'_{ij} \;=\; f_{ij} \;+\; \sum_{r \in VRS} e_{ir}\, h_{rj} \quad \forall \quad i,j \in TRS$$

The transition probability matrix of the REMC can thus be expressed as

$$\boldsymbol{U'} \;=\; \boldsymbol{F} \;+\; \boldsymbol{E}\,\boldsymbol{H} \tag{9.37}$$

Denoting again with $\boldsymbol{\psi'}(n)$ the probability distribution of the REMC at step n (i.e., after the firing of n timed transitions), we can compute this quantity using the following expression

$$\boldsymbol{\psi'}(n) \;=\; \boldsymbol{\psi'}(0)\boldsymbol{U'}^n \tag{9.38}$$

The steady-state probability distribution $\boldsymbol{\psi'}$ can be obtained as the solution of the system of linear equations

$$\begin{cases} \boldsymbol{\psi'} \;=\; \boldsymbol{\psi'}\,\boldsymbol{U'} \\ \boldsymbol{\psi'}\,\mathbf{1}^{\mathrm{T}} \;=\; 1 \end{cases} \tag{9.39}$$

in which $\boldsymbol{\psi'}$ allows the direct computation of the mean number of visits performed by the net to tangible markings only, between two subsequent visits to a reference marking. By selecting one of the markings of the REMC as a reference marking for the chain (say marking $\boldsymbol{m}_i$), the mean number of visits to an arbitrary marking $\boldsymbol{m}_j$, between two subsequent visits to marking $\boldsymbol{m}_i$, is obtained as:

$$v_{ij} \;=\; \frac{\psi'_j}{\psi'_i} \tag{9.40}$$

The stationary probability distribution associated with the set of tangible markings is thus readily obtained by means of their average sojourn times (see equation (9.9)) using the following formula:

$$\pi'_j \;=\; \frac{v_{ij}\; SJ_j}{CY(\boldsymbol{m}_i)} \tag{9.41}$$

where $CY(\boldsymbol{m}_i)$ represents the mean cycle time of the GSPN with respect to state $\boldsymbol{m}_i$, i.e., the average amount of time that the GSPN takes to return to marking $\boldsymbol{m}_i$. $CY(\boldsymbol{m}_i)$ has the following expression:

$$CY(\boldsymbol{m}_i) \;=\; \sum_{k:\boldsymbol{m}_k \in T} v_{ik}\; SJ_k \tag{9.42}$$

The construction of the REMC defined over the set of tangible markings TRS implies that a transformation exists of the semi-Markov process associated with every GSPN system into a CTMC. The steady-state probability distribution over the tangible markings can thus be also obtained by a direct solution of this CTMC. In our case, the infinitesimal generator $\boldsymbol{Q}'$ of the CTMC associated with a GSPN can be constructed from the transition probability rate matrix $\boldsymbol{U}'$ of the REMC by dividing each of its rows by the mean sojourn time of the corresponding tangible marking. To conform with the standard definition of the infinitesimal generators, the diagonal elements of $\boldsymbol{Q}'$ are set equal to the negative sum of the off-diagonal components:

$$q'_{ij} \;=\; \begin{cases} \frac{1}{SJ_i}\; u'_{ij} & i \neq j \\ \\ -\sum_{j \neq i} q'_{ij} & i = j \end{cases} \tag{9.43}$$

This result shows that GSPNs, like SPNs, can be analysed by solving properly associated CTMCs. This obviously implies that the transient state probability distribution is the solution of the following differential equation

$$\frac{d\boldsymbol{\pi}'(\tau)}{d\tau} \;=\; \boldsymbol{\pi}'(\tau)\boldsymbol{Q}' \tag{9.44}$$

while the steady-state probability distribution over the tangible markings requires the solution of the following system of linear equations:

$$\begin{cases} \boldsymbol{\pi'}\; \boldsymbol{Q'} \;=\; \mathbf{0} \\ \\ \boldsymbol{\pi'}\; \mathbf{1}^{\mathrm{T}} \;=\; 1 \end{cases} \tag{9.45}$$

The computation of the performance indices defined over GSPN models can be performed using the reward method discussed in Section 9.1 without any additional difficulty.

The advantage of solving the system by first identifying the REMC is twofold. First, the time and space complexity of the solution is reduced in most cases,

since the iterative methods used to solve the system of linear equations tend to converge more slowly when applied with sparse matrices and an improvement is obtained by eliminating the vanishing states thus obtaining a denser matrix [11, 6]. Second, by decreasing the impact of the size of the set of vanishing states on the complexity of the solution method, we are allowed a greater freedom in the explicit specification of the logical conditions of the original GSPN, making it easier to understand.

The method outlined in this section exploits the elegant mathematical structure of the problem to overcome the difficulties due to the presence of loops of immediate transitions. The loops considered in this derivation are of the "transient" type [11] and correspond to situations in which a steady-state analysis of the model is possible. The REMC is instead impossible to construct following this approach when the loop of immediate transitions is of the "absorbing" type so that during its evolution the net can be trapped into a situation from which it can not [11] exit. Except for very pathological cases [11] in which the model makes sense despite the presence of such absorbing loops, GSPNs of this type are considered non-well formed and their analysis is stopped once the existence of absorbing loops of immediate transitions is discovered during the construction of the infinitesimal generator of the REMC.

In practice, loops of immediate transitions seldom appear in GSPN models. This observation lead to the development of solution methods that discard the vanishing states "on the fly" thus trading a save of space with some additional computational effort due to the fact that in some situations sequences of vanishing states are repeatedly generated to compute the transition rates among different pairs of tangible markings. Usually this method is computationally convenient, even if some extra work must be performed to preanalyze the model in order to discover the presence of situations that can not be treated with this technique. The choice of not saving vanishing markings has also the drawback of making certain performance indices related with the firing of immediate transitions impossible to compute. Details on the comparison of the advantages and disadvantages of storing vanishing markings can be found in [11].

9.3.1 Computational Considerations

The mathematically elegant solution techniques outlined so far in this section suffer in practice of the difficulties deriving from the size of the CTMCs associated with these models and of the time-scale differences usually existing among the specifications of the activities represented by transitions. In this subsec-

tion we will discuss the main difficulties encountered for the computation of the transient and steady-state probability distributions and we will outline some solution techniques, while we will refer to the specialized literature for a deep discussion of the problem.

Transient Solution - Uniformization Method

The first difficulty in the analysis of the CTMC associated with GSPN models comes from the evaluation of the transient probability distribution on the markings of the net.

The apparently simple solution

$$\boldsymbol{\pi}'(\tau) \;=\; \boldsymbol{\pi}'(0)e^{\boldsymbol{Q}'\tau} \;=\; \boldsymbol{\pi}'(0)\sum_{k=0}^{\infty}\frac{(\boldsymbol{Q}'\tau)^k}{k!} \tag{9.46}$$

of the Chapman-Kolmogorov differential equations

$$\frac{d\boldsymbol{\pi}'(\tau)}{d\tau} \;=\; \boldsymbol{\pi}'(\tau)\boldsymbol{Q}' \tag{9.47}$$

are unfortunately often rather difficult and unstable to compute [18]. Moler and Van Loan [8] discuss nineteen "dubious" ways to compute the exponential of relatively small order matrices whose accuracy of the approximations heavily depend on the norms of the matrices. One of the most commonly used methods for computing this result is called the *Uniformization* technique that is based on the following simple derivation.

Assume that the diagonal elements of the infinitesimal generator $\boldsymbol{Q}'$ are bounded, so that there exist a Γ such that:

$$|q'_{ii}| \;\leq\; \Gamma \;<\; \infty, \quad \forall \boldsymbol{m}_i \;\in\; TRS \tag{9.48}$$

A (discretized) transition probability matrix $\boldsymbol{R}$ can be defined as follows

$$\boldsymbol{R} \;=\; \boldsymbol{I} \;+\; \frac{1}{\Gamma}\boldsymbol{Q}' \tag{9.49}$$

From this definition we have that $\boldsymbol{Q}' = \Gamma(\boldsymbol{R} - \boldsymbol{I})$, so that

$$\boldsymbol{\pi}'(\tau) \;=\; \boldsymbol{\pi}'(0)e^{\boldsymbol{Q}'\tau} \;=\; \boldsymbol{\pi}'(0)e^{\tau\Gamma\boldsymbol{R}-\tau\Gamma\boldsymbol{I}} \;=\; \boldsymbol{\pi}'(0)e^{-\tau\Gamma}e^{\tau\Gamma\boldsymbol{R}} \tag{9.50}$$

since $e^{-(\tau\Gamma)\boldsymbol{I}} = e^{-\tau\Gamma}\boldsymbol{I}$ and $\boldsymbol{R}$ and $\boldsymbol{I}$ commute. The transient distribution at time τ is thus obtained by computing an approximation to the infinite summation

$$\boldsymbol{\pi}'(\tau) \;=\sum_{k=0}^{\infty}\boldsymbol{\pi}'(0)\boldsymbol{R}^k e^{-\Gamma\tau}\frac{(\Gamma\tau)^k}{k!} \tag{9.51}$$

which is called the *uniformization equation*. The advantages of this technique are the simplicity of its implementation and the possibility to control the approximation error. In fact, it is possible to easily identify the limit after which to truncate the series in order to reduce the consequent error below any predefined threshold ϵ [18]. The approximated result can thus be expressed in the following form:

$$\boldsymbol{\pi}'(\tau) = \sum_{k=0}^{K} \boldsymbol{\pi}'(0)\boldsymbol{R}^k e^{-\Gamma\tau} \frac{(\Gamma\tau)^k}{k!} \tag{9.52}$$

Finally, it is worth to observe that, if the values of Γ and τ are such that the computation of $e^{-\Gamma\tau}$ produces an underflow, the evaluation of the expression at the desired τ can be obtained subdividing the interval in sufficiently many subintervals (say l) of equal length h so that

$$\boldsymbol{\pi}'(\tau i + 1) = \boldsymbol{\pi}'(\tau_i)\Delta_i, \quad i = 0,\ 1,\ l-1 \tag{9.53}$$

where

$$\Delta_i = \sum_{k=0}^{K} \boldsymbol{R}^k e^{-\Gamma h} \frac{(\Gamma h)^k}{k!} \tag{9.54}$$

and where $\tau_{i+1} = \tau_i + h$.

Steady-State Solution - Time Scale Decomposition

As we have seen at the beginning of this section, the steady state probability distribution of the markings of a GSPN model is obtained from the solution of a system of linear equations. This problem, that is mathematically simple and well understood, may pose considerable difficulties in the case of GSPN due to the size of their reachability set and to the possibility of having within the infinitesimal generator of the corresponding CTMC rates that are order of magnitude different from each other. In this chapter we will not survey the many methods that can be employed for the solution of these large system of linear equations; the interested reader is referred to [18] for a comprehensive discussion of direct as well as iterative techniques that can be employed to obtain the desired solution keeping under control the storage requirements as well as the computational costs. We will instead briefly outline a technique that can be conveniently used to obtain approximate results when the GSPN models are of a special type and when transitions of different speeds are included in the net.

Consider the case of the operation of a simple processor/memory system in which the memory may fail while it is not accessed. Fig. 9.9 depicts the GSPN model of such a system in which failures happen quite rarely and repairs require

Table 9.7: Specification of the transitions of the SPN of Fig. 9.9

transition	rate	semantics
T_{req}	λ	single-server
T_{end}	μ	single-server
T_{fail}	γ	single-server
T_{rep}	δ	single-server

transition	weight	priority	ECS
t_{str}	1	1	1

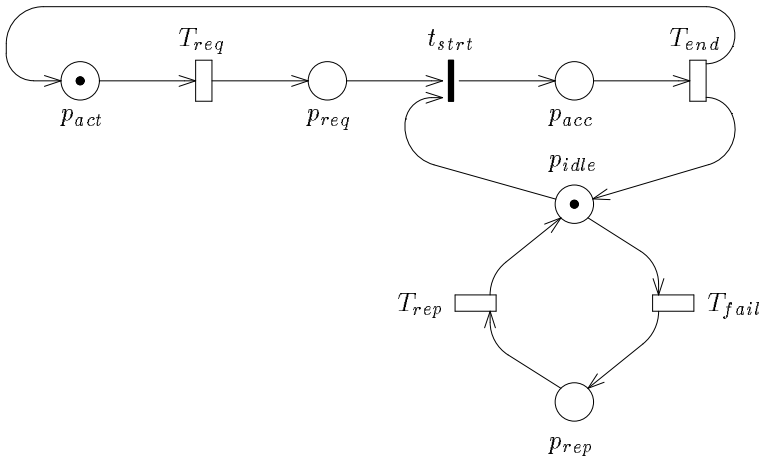


Figure 9.9: A simple processor/memory system with failures

a considerable amount of time to be completed. In a model of this type, besides the distinction between immediate and timed transitions, we can further classify timed transitions as *fast* and *slow*. In Fig. 9.9, transitions T_{req} and T_{str} can be considered fast, while transitions T_{fail} and T_{rep} can be assumed to be slow.

A method for solving these models in a computationally convenient manner has been proposed by Ammar and Islam [4] that is based on the theory of the *Near Complete Decomposability* due to Simon and Ando [17] and further investigated by Courtois [12]. The theoretical basis of this method is treated at some length in Chapter 13. Here we only observe that (always referring to our Processor/Memory model with failures) at fast time scale failures and repairs are so far apart in time, during these intervals the system can be assumed to

Table 9.8: Specification of the transitions of the SPN of Fig. 9.10

transition	rate	semantics
T_{req}	λ	single-server
T_{end}	μ	single-server

transition	weight	priority	ECS
t_{str}	1	1	1

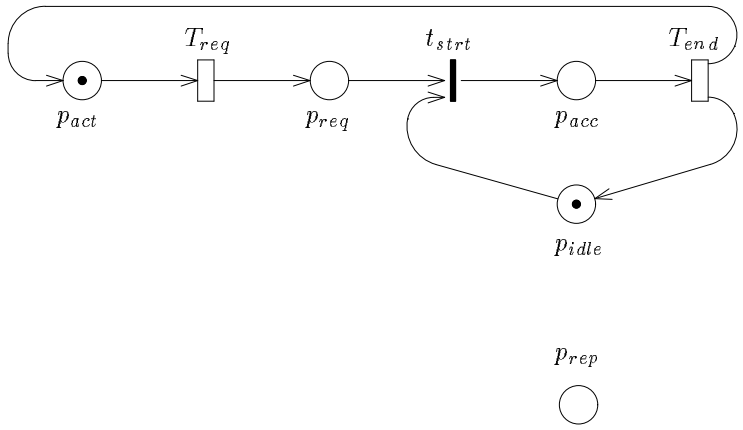


Figure 9.10: Fast subnets derived form the simple processor/memory system with failures of Fig. 9.9

be fault-free. The theory of near decomposable systems says that as the process approaches the long-term equilibrium, a short-term equilibrium is reached between rare events, so that the short-term analysis can be approximately separated from the analysis of the long-run dynamics of such systems. The theory is developed by identifying within the Markov chains representing these systems, groups of states with (internal) strong interaction compared with the weak interaction existing among states of different groups. The method of Ammar tries to identify these groups at the GSPN level by removing the slow transitions so that the net decomposes into several "fast" subnets that describe the short-run dynamics of the system. The individual solution of the CTMCs associated with these subnets provides the basis for the computation of aggregated representations that are used in the construction of a reduced subnet, called the aggregated GSPN (AGSPN), used for the analysis of the long-run dynamics of the system.

Table 9.9: Specification of the transitions of the SPN of Fig. 9.11

transition	rate	semantics
T_{a1}	α	single-server
T_{a2}	β	single-server

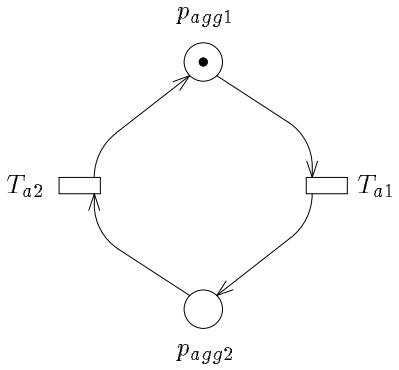


Figure 9.11: Aggregated subnet derived form the simple processor/memory system with failures of Fig. 9.9

Following the technique proposed by Ammar, we remove the slow transitions from the net of Fig. 9.9 and we obtain the two subnets of Fig. 9.10, only one of which is interesting for the construction of the aggregated net of Fig. 9.11. Computing the solution of this first subnet (which is very simple since it has only two tangible markings corresponding to the processor active locally and to the processor accessing the memory), we observe that the firing rate of transition T_{a1} of the aggregated net is different from zero only when this fast submodels is in its first state.

Indicating with $\boldsymbol{\pi}$ the solution of the original model of Fig. 9.9, with $\boldsymbol{\pi}^*$ its approximation computed with the TSDM of Ammar, with $\boldsymbol{\pi}^1$ the solution of the fast submodel of Fig. 9.10, and with P the solution of the aggregated model

of Fig. 9.11, we have:

$$\begin{aligned}
\pi^1(p_{act} + p_{idle}) &= \frac{1}{1 + \rho} \\
\alpha &= \frac{\gamma}{1 + \rho} \\
\beta &= \delta \\
\mathcal{P}(p_{agg1}) &= \frac{1 + \rho}{1 + \rho + \sigma} \\
\pi^*(p_{act} + p_{idle}) &= \frac{1}{1 + \rho + \sigma} \\
\pi(p_{act} + p_{idle}) &= \frac{1}{1 + \rho + \sigma + \rho\sigma\frac{\delta}{\lambda + \delta}}
\end{aligned} \tag{9.55}$$

where $\rho = \lambda/\mu$ and $\sigma = \gamma/\delta$. The difference between $\boldsymbol{\pi}$ and $\boldsymbol{\pi}^*$ is due to the approximate computation of the aggregated rate α of transition T_{a1} of the model of Fig. 9.10. The exact aggregation procedure discussed in Chapter 13 (but that unfortunately cannot be applied, since it requires the "a priori" knowledge of the final distribution) would have produced the following value for α:

$$\alpha = \frac{\gamma}{1 + \rho\,(1 + \frac{\gamma}{\lambda+\delta})} \tag{9.56}$$

that differs from the previous one only for the ratio $\gamma/(\lambda + \delta)$ that obviously becomes negligible when $\gamma \ll \lambda$ as it is the case for our model.

In general the technique presented in [4] cannot be applied in such a straightforward manner and requires some clever intervention from the analyst, thus making it unsuitable for automatic applications. Under the condition of dealing with a somehow more restricted class of models, a variation of the above technique has been presented by Blakemore and Tripathi [7] that, working at the level of the state space of the CTMC underlying the GSPN model, first define the groups of states used for the aggregation, and then identify the transitions (of the GSPN model) that make the CTMC moving among these groups of states. Depending on the choice of the groups of states, the transitions that induce a change of state between groups (called *cross transitions*) can be either fast or slow. An implementation of this method should make sure that cross transitions are also slow in order for the TSD technique to be accurate, issuing a warning when this condition is not met. The interested reader will find

the details of this new method, together with an algorithm for its automatic application, in the original paper [7].

9.4 Reducing GSPNs to SPNs

As we saw in the previous sections, immediate transitions, that have quite a beneficial impact on the modelling power of GSPNs, unfortunately make the overall analysis of GSPNs more complex than that of SPNs. Indeed, while for the latter the reachability graph is isomorphic to the state transition rate diagram of the CTMC underlying the SPN, such isomorphism does not exist for GSPNs, but is obtained only after the elimination of vanishing markings. An analysis procedure different from that described in the previous sections consists in the elimination of all immediate transitions from a given GSPN before starting the generation of its reachability set, so as to produce an SPN whose underlying continuous-time Markov chain is identical to that corresponding to the GSPN.

This section provides a concise and informal overview of the rules for the reduction of a GSPN model to an equivalent SPN by eliminating all immediate transitions. A complete and formal description of these reduction rules and of the class of GSPNs for which it is possible to compute an equivalent SPN, can be found in [10, 13].

The basic idea behind the elimination of immediate transitions is quite simple and can be easily explained in its natural form by means of an example. The model depicted in Fig. 9.12(a) is a free-choice GSPN subnet whose SPN counterpart is shown in Fig. 9.12(b). The two systems are equivalent as far as tangible states are concerned.

To understand how we can transform the GSPN into its corresponding SPN, consider what happens in the GSPN when T_a fires: the three immediate transitions t_1, t_2, and t_3 become enabled, due to the arrival of one token in place p_b; they are the only transitions that are enabled in the subnet, and the choice about which one to fire is made according to their associated weights. The basic behaviour of the subnet in Fig. 9.12(a) is therefore that of "moving" tokens from place p_a to one of the three places p_1, p_2, and p_3. The net in Fig. 9.12(b) is clearly equivalent to that of Fig. 9.12(a) from the point of view of the flow of tokens (token flow equivalence was defined in [5]).

When time is involved, token flow equivalence is not enough to ensure the possibility of freely interchanging the two types of models. In particular, the equivalence notion we are interested in must preserve the underlying stochastic behaviour of the net. We must therefore take into account not only the possible

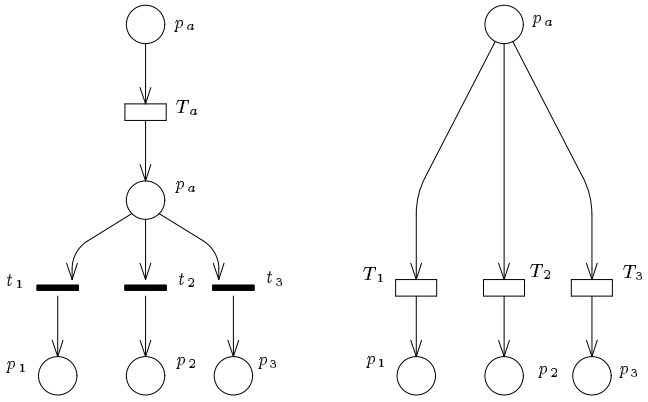


Figure 9.12: Removing immediate transitions: the free-choice case

final states of the subnet, but also the rates at which the model transits from state to state.

The operations of the GSPN subnet induce a movement of a token from p_a to p_k, $k = 1, 2, 3$, with rate

$$\frac{w_a w_k}{\sum_{j=1,2,3} w_j} \tag{9.57}$$

where w_a is the rate of the exponential distribution associated with timed transition T_a, and w_k is the weight of immediate transition t_k, so that

$$\frac{w_k}{\sum_{j=1,2,3} w_j} \tag{9.58}$$

is the probability that t_k is the transition that actually fires when t_1, t_2, and t_3 are all enabled.

The timed transition T_k in the SPN model represents the firing of timed transition T_a followed by immediate transition t_k. If we define its firing rate as in equation (9.57), then the rate at which the SPN subnet in Fig. 9.12(b) induces a movement of a token from p_a to p_k, given that transition T_a is enabled, is exactly the same as for the GSPN subnet, and therefore the rates of the underlying Markov processes are the same. Note that place p_b was deleted in the reduction process: this is not surprising because p_b is a vanishing place, a place that is never marked in a tangible marking.

The elimination procedure is somewhat more complex if the set of immediate transitions enabled by the firing of a timed transition does not form a free-choice conflict. Indeed, in this case the probability of firing an immediate transition

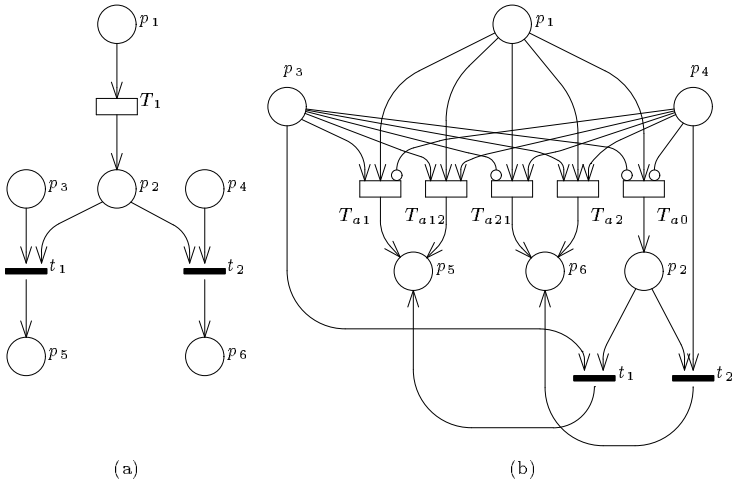


Figure 9.13: Removing immediate transitions: the non-free-choice case

may depend also on the other simultaneously enabled immediate transitions. We are thus forced to consider all possible cases. As an example, consider the subnet in Fig. 9.13(a).

The firing of timed transition T_a may enable either only t_1, or only t_2, or both t_1 and t_2, or neither of them, depending on the marking of p_3 and p_4. In any case, at most one of the two immediate transitions can fire (only one token is in p_2 after the firing of T_a). The probability that, for example, t_1 is the transition that actually fires is different in the two cases where t_1 is enabled alone, or together with t_2: we have therefore to distinguish the two cases. A similar consideration holds for t_2, so that we have two more cases, plus the additional case in which the firing of the timed transition cannot be followed by that of any immediate transition. We hence have five possible situations:

1. T_a is followed by t_1 when only t_1 is enabled (transition T_{a1} with rate w_a);
2. T_a is followed by t_1 when both t_1 and t_2 are enabled (transition T_{a12} with rate $w_a w_1/(w_1 + w_2)$);
3. T_a is followed by t_2 when only t_2 is enabled (transition T_{a2} with rate w_a);
4. T_a is followed by t_2 when both t_2 and t_1 are enabled (transition T_{a21} with rate $w_a w_2/(w_1 + w_2)$);
5. T_a is not followed by any immediate transition (transition T_0 with rate w_a).

In the GSPN subnet resulting from the reduction of t_1 and t_2 over T_a, shown in Fig. 9.13(b), five timed transitions take the place of T_a. No vanishing place existed in the original GSPN, so that no place could be deleted. The size of the state space has been reduced, as all vanishing markings of the original GSPN are not reachable any longer (for example, the marking $\{p_2, p_3, p_4\}$ is not generated by the reduced net of Fig. 9.13(b)). Observe that the subnet depicted in Fig. 9.13(a) includes only one of the timed transitions that may deposit tokens in the input places of the ECS composed by t_1 and t_2, but in general there can be more than one. This is the reason why the two immediate transitions t_1 and t_2 are still present in the reduced subnet of Fig. 9.13(b), since they shall also be reduced over all timed transitions that add tokens to p_3 and/or p_4.

Bibliography

[1] M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte. Generalized stochastic Petri nets revisited: Random switches and priorities. In *Proc. Intern. Workshop on Petri Nets and Performance Models*, pages 44–53, Madison, WI, USA, August 1987. IEEE-CS Press.

[2] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.

[3] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. MIT Press, Cambridge, USA, 1986.

[4] H.H. Ammar and S.M. Rezaul Islam. Time scale decomposition of a class of generalized stochastic Petri net models. *IEEE Transactions on Software Engineering*, 15(6):809–820, June 1989.

[5] G. Berthelot. Transformations and decompositions of nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 359–376. Springer Verlag, Bad Honnef, West Germany, February 1987.

[6] A. Blakemore. The cost of eliminating vanishing markings from generalized stochastic Petri nets. In *Proc. 3^{rd} Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.

[7] A. Blakemore and S.K. Tripathi. Automated time scale decomposition and analysis of stochastic Petri nets. In *Proc. 5^{th} Intern. Workshop on Petri Nets and Performance Models*, pages 248–257, Toulouse, France, October 1993. IEEE-CS Press.

[8] Moler C. and C. Van Loan. Nineteen doubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–836, 1978.

[9] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, February 1993.

[10] G. Chiola, S. Donatelli, and G. Franceschinis. GSPN versus SPN: what is the actual role of immediate transitions? In *Proc. 4^{th} Intern. Workshop on Petri Nets and Performance Models*, pages 20–31, Melbourne, Australia, December 1991. IEEE-CS Press.

[11] G. Ciardo. *Analysis of large Petri net models.* PhD thesis, Department of Computer Science, Duke University, Durham, NC, USA, 1989. Ph. D. Thesis.

[12] P.J. Courtois. *Decomposability: Queueing and Computer Systems Applications.* Academic Press, New York, 1977.

[13] S. Donatelli. *L'uso delle reti di Petri per la valutazione e la validazione di sistemi di grandi dimensioni.* PhD thesis, Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10149, Torino, Italy, February 1990. (in italian).

[14] G. Florin and S. Natkin. Les reseaux de Petri stochastiques. *Technique et Science Informatiques*, 4(1), February 1985.

[15] M.K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models.* PhD thesis, UCLA, Los Angeles, CA, 1981. Ph.D. Thesis.

[16] M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models.* Johns Hopkins University Press, Baltimore, MD, 1981.

[17] H.A. Simon and A. Ando. Aggregation of variables in dynamic systems. *Econometrica*, 29, 1961.

[18] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, Princeton, New Jersey, USA, 1994.

[19] F.J.W. Symons. *Modeling and Analysis of Communication Protocols Using Numerical Petri Nets.* PhD thesis, University of Essex, May 1978.

[20] R.S. Varga. *Matrix Iterative Analysis.* Prentice-Hall, Englewood Cliffs, NJ, 1962.

Chapter 10

NON-EXPONENTIAL STOCHASTIC PETRI NETS

Many activities in computer, communications, and manufacturing systems are difficult to characterize with negative-exponentially distributed durations and a need exists for including in their Petri net representations random firing delays with low variability (or even constant duration). Moreover, some significant measures, introduced to characterize a stochastic system over an interval rather than at a time instant (like the distribution function of cumulative measures), cannot be evaluated by solving a set of linear first-order differential equations typical of Markovian systems [35, 7] and require a more complex stochastic formulation.

In recent years, several classes of Stochastic Petri Net (*SPN)* models have been proposed which incorporate some non-Markovian characteristics in their definition. The semantics of *SPN*s with generally distributed transition times has been discussed in [2]. We refer to this model as *Generally Distributed SPN (GDSPN).*

In this chapter we discuss the *GDSPN* models recently proposed in the literature with an emphasis on their modelling power and analytical tractability. For this purpose, the main features of the different formulations are briefly described with the intent of stressing their basic assumptions and the complexity of their analytical solutions.

10.1 Generally Distributed SPN

In order to properly define the marking process associated with a *GDSPN*, each timed transition should be assigned a memory policy chosen among three possible alternatives: *resampling, enabling*, and *age memory*. Memory policies account for different behaviours that may arise in the model depending on whether the represented activities are reset at each state change (resampling), are reset whenever they are interrupted (enabling), or continue from the point in which they were interrupted when they are started again (age).

With the aim of providing a modeler with a formalism yielding representations that can be automatically translated into analytical models, various restrictions of the general *GDSPN* language have been discussed in the literature [16, 10] which are a compromise between representation power and model tractability.

The semi-Markov *SPN*s, studied in [33, 6], are of little practical interest since in these models the firing of any transition forces a resetting of all the other transitions in the net thus making the representation of parallel activities quite difficult. A semi-Markov type of *SPN*, more suited for applications, has been discussed in [22] where the transitions are partitioned into three classes: *exclusive, competitive, and concurrent*. Informally speaking, we can say that exclusive transitions are always enabled alone; competitive transitions are instead in conflict and change their enabling conditions when one of them fires; finally, concurrent transitions act independently and their enabling conditions are unaffected by the firing of one of them. In this early proposal, only exclusive or competitive transitions can be non-exponential. An embedded Markov chain that describes the evolution of the model at firing instants of the non-exponential transitions can be identified, provided that concurrent transitions are all exponentially distributed and that competitive transitions resample a new firing delay whenever they are enabled.

An alternative approach is that proposed by Cumani [21] in which non-exponential distributions are allowed to be of the *phase-type* only [34]. Without changing the semantics of the model, transitions with phase-type distributions have their firing divided into exponential stages. *SPN*s with phase-type distributions (that we call *PHSPN*) can still be mapped onto Markov chains at the expense of an expansion of their state space that accounts, not only for the distribution of tokens over the places, but also for the current stage of firing of the enabled transitions. The peculiar feature of the *PHSPN* model is that it supports any memory policy combination, always yielding Markov models that

can be analyzed with standard techniques.

A particular case of non-Markovian *SPN*, is the class of *Deterministic and SPN (DSPN)*, where timed transitions may have either constant or exponential firing times. A *DSPN* is defined in [4] as a Markovian *SPN* where, in each marking, only one deterministic transition can be enabled. Only the steady-state solution was provided in [4]. An improved steady-state algorithm was presented in [31], and some structural extensions were investigated in [17]. Choi et al. [14] observed that the marking process underlying a *DSPN* is a *Markov Regenerative Process (MRP)* [18] for which a closed-form expression for the transition probability matrix can be derived both as a function of time and in steady-state.

This observation opened a very fertile line of research aimed at the definition of solvable classes of *SPN* models whose underlying marking process is a *MRP*, and therefore referred to as *Markov Regenerative SPN (MRSPN)*. Choi et al. [15] extended the *DSPN* model by allowing the presence in each marking of at most one enabled transition with a generally distributed firing time. The solution proposed in [15] is based on the derivation of the time-dependent transition probability matrix in the Laplace transform domain, followed by a numerical inversion. German and Lindemann [26] proposed to derive the steady-state solution of the same model by resorting to the method of supplementary variables [19]. The possibility of applying the supplementary variable technique to the transient analysis of *DSPN*s is explored in [24], where, however, only very special cases are taken into consideration.

The main limitation of the models proposed by these authors is that the generally distributed (or deterministic) transitions must be assigned a firing policy of enabling memory type [1].

A semantic generalization of the *DSPN* model, by including the possibility of modeling preemptive mechanisms of resume type, has been proposed in [12]. This modeling extension is crucial in connection with fault tolerant and parallel computing systems, where a single task may be interrupted either during a fault/recovery cycle or for the execution of a higher priority activity, provided that it is resumed from the point it was interrupted, when the cause originating the interruption is completed. Finally, a more general class of solvable *MRSPN* is defined in [11], where both enabling and age memory policies can be combined into a single model.

[1] The enabling memory assumption is relaxed in [17] where a deterministic transition can be disabled in vanishing markings only. Since vanishing markings are traversed in zero time, this assumption does not modify the behaviour of the marking process versus time.

10.2 The Stochastic Process Associated with a GDSPN

The complexity of the stochastic models associated with *GDSPN* requires that the analysis discussed in Chapter 9 in the case of exponential Petri nets is refined and that some additional attention is given to the way in which stochastic timing is introduced in PN models. Since we are considering causal systems, we want to be able to describe (at least in a probabilistic sense) the future behaviour of a system from the knowledge of its past history.

We begin by recalling some basic definitions relating to *SPN*s in order to introduce the notation that will be used throughout the chapter.

A timed execution sequence $\mathcal{T}_E$ is a connected path in the reachability graph $RG(\boldsymbol{m}_0)$ augmented by a non-decreasing sequence of real non-negative values representing the epochs of firing of each transition; consecutive transition firings correspond to ordered epochs $\tau_i \leq \tau_{i+1}$ in $\mathcal{T}_E$ [23].

$$\mathcal{T}_E = \{(\tau_0, \boldsymbol{m}_{(0)}); (\tau_1, \boldsymbol{m}_{(1)}); \ldots; (\tau_i, \boldsymbol{m}_{(i)}); \ldots\} \tag{10.1}$$

The time interval $\tau_{i+1} - \tau_i$ between consecutive epochs represents the period of time that the PN sojourns in marking $\boldsymbol{m}_{(i)}$ [2].

A timed execution $\mathcal{T}_E$ truncated at the k-th epoch τ_k is called the *history* of the PN up to the k-th epoch τ_k and is denoted by $\boldsymbol{Z}(k)$. In the following we always assume (without loss of generality) $\tau_0 = 0$ as the initial epoch.

For the durations of the activities (firing delays) that determine these state changes, we make the following

Assumption 1. *Let $\boldsymbol{Z} = \boldsymbol{Z}(i)$ be a history of the PN up to the i-th epoch, and $\boldsymbol{m} = \boldsymbol{m}_{(i)}$ be the marking entered by firing transition $t_{(i)}$. We assume that for all i, $\boldsymbol{Z}$, and $\boldsymbol{m}$, the* firing distribution *$D(x|\boldsymbol{m}, \boldsymbol{Z})$ can be uniquely determined in such a way that its k-th component is defined (interpreted) in the following way:*

$$D_k(x|\boldsymbol{m}, \boldsymbol{Z}) = Pr\{t_k \; fires, \; firing \; delay \leq x \mid \boldsymbol{m}, \boldsymbol{Z}\} \tag{10.2}$$

The random variable "firing delay" represents the time that elapses from entering $\boldsymbol{m}$ up to the next firing epoch, i.e., the time interval $\tau_{i+1} - \tau_i$. The above distribution must be defined over all the transitions enabled in $\boldsymbol{m}$ (i.e., for all

[2] The notation $\boldsymbol{m}_{(i)}$ is used to indicate the marking entered by the net after the i-th transition firing (i-th epoch). Notice that $\boldsymbol{m}_{(0)} = \boldsymbol{m}_0$.

t_k in $\boldsymbol{e}(\boldsymbol{m})$) so that

$$\sum_{k:\, t_k \in \boldsymbol{e}(\boldsymbol{m})} \lim_{x->\infty} D_k(x|\boldsymbol{m}, \boldsymbol{Z}) \;=\; 1. \tag{10.3}$$

Note that $\boldsymbol{m}$ is known from $\boldsymbol{Z}$, but we have explicitly indicated the dependence on $\boldsymbol{m}$ since in many cases $\boldsymbol{m}$ is the only element that actually influences this distribution function.

The probability $p_k(\boldsymbol{m}, \boldsymbol{Z})$ of selecting t_k to be the next transition to fire is obtained as:

$$p_k(\boldsymbol{m}, \boldsymbol{Z}) \;=\; \lim_{x->\infty} D_k(x|\boldsymbol{m}, \boldsymbol{Z}) \;=\; Pr\{t_k \; fires \mid \boldsymbol{m}, \boldsymbol{Z}\} \tag{10.4}$$

and the distribution of the time spent in marking $\boldsymbol{m}$ before the next epoch is found as:

$$F(x|\boldsymbol{m}, \boldsymbol{Z}) \;=\; \sum_{k:t_k \in \boldsymbol{e}(\boldsymbol{m})} D_k(x|\boldsymbol{m}, \boldsymbol{Z}) \;=\; Pr\{firing\ delay \;\leq x \;|\boldsymbol{m}, \boldsymbol{Z}\} \tag{10.5}$$

Having introduced the notation and the basic concepts that underly the behaviour of the model, we can now give the following

Definition 1. *A GDSPN is a marked SPN in which:*

- *A random variable θ_k is associated with any timed transition $t_k \in T$. θ_k models the time needed by the activity represented by t_k to complete, when considered in isolation.*
- *Each random variable θ_k is characterized by its (possibly marking dependent) cumulative distribution function $G_k(x, \boldsymbol{m})$.*
- *A set of specifications is given for unequivocally defining the stochastic process associated with the ensemble of all the timed execution sequences $\mathcal{T}_E$. This set of specifications is called the* execution policy.
- *An initial probability is given on the reachability set.*

With the above definition, the set of possible executions of a $GDSPN$, together with the probability measure induced on it by assigning the firing distribution of Assumption 1, constitutes a continuous-time discrete-state stochastic point process.

The state space of the stochastic process is not necessarily isomorphic with the reachability graph of the underlying PN, since the timing constraints superimposed on the firing rules may alter the set of possible execution sequences,

as in the case that some of the firing probabilities of (10.4) might reduce to zero. In order to avoid these phenomena, we restrict the class of allowed distribution functions by introducing the following [3]:

Assumption 2. *The probabilities $p_k(\boldsymbol{m}, \boldsymbol{Z})$ of eq. (10.4) satisfy the following condition:*

$$p_k(\boldsymbol{m}, \boldsymbol{Z}) \; > \; 0 \quad \forall \; k : t_k \in \boldsymbol{e}(\boldsymbol{m}) \tag{10.6}$$

The definition of $GDSPN$ together with Assumptions 1 and 2 guarantee the isomorphism between the state space of the stochastic process and the reachability graph of the underlying PN. In the following, we will often refer to this stochastic process as the marking process $\mathcal{M}(\tau)$ of the $GDSPN$. For what concerns the initial probability distribution, we assume that the net is in marking $\boldsymbol{m}_0$ at $\tau = 0$ with probability one as we already (implicitly) did in Chapter 9.

An *execution policy* is a set of specifications for unequivocally defining the stochastic process underlying the $GDSPN$, given the PN topology and the set of distribution functions associated with the timed transitions. Indeed, the inclusion of non-exponential timings destroys the memoryless property and forces us to specify how the system is conditioned upon its past history. The execution policy comprises two specifications: a criterion to choose the next timed transition to fire (the *firing policy*), and a criterion to keep memory of the past history of the process (the *memory policy*).

As in the case of the exponential Petri nets discussed in Chapter 9, we assume that the association of time with transitions can be intuitively described as the definition of a timer for each timed transition. When the transition becomes enabled, the timer is set and from then on it is decremented with constant speed until it becomes zero. When this happens, the transition fires and with an atomic action tokens are removed from the input place(s) and new tokens are deposited in the output place(s).

When several transitions are enabled in the same marking, the transition that fires is that the one associated delay is statistically the minimum. This firing policy is called a *race* policy and this type of model is called a *race* (or concurrent [28]) model.

[3] All the derivations that follow in this chapter apply to the case of $GDSPN$ with timed and immediate transitions. To keep them simple, however, we will develop the analysis for the case of $GDSPN$ models without immediate transitions and the reader is referred to the specialized literature for details concerning the case of more general models.

10.2.1 Race SPN Definition

When the net enters marking $\boldsymbol{m}$, a random sample is extracted from the joint distribution:

$$\phi(x_1, x_2, ...|\boldsymbol{m}, \boldsymbol{Z}) = Pr\{\theta_1 \leq x_1, \theta_2 \leq x_2, ...|\boldsymbol{m}, \boldsymbol{Z}\} \tag{10.7}$$

where the θ_k are random variables representing, for each transition $t_k \in \boldsymbol{e}(\boldsymbol{m})$, the time till firing, measured from the epoch at which $\boldsymbol{m}$ was entered. The θ_k for which the sampled value is minimum determines which transition will actually fire, and the sojourn time in $\boldsymbol{m}$ equals this minimum sampled value. We only consider the case in which all random variables θ_k are independent, so that (10.7) is uniquely determined by the marginal distributions:

$$\phi_k(x|\boldsymbol{m}, \boldsymbol{Z}) = Pr\{\theta_k \leq x|\boldsymbol{m}, \boldsymbol{Z}\} \tag{10.8}$$

In order to satisfy conditions (10.3) and (10.6), it is sufficient to assume that these distribution functions are derivable (at least in the sense of generalized functions), honest distributions whose derivative with respect to x is a probability density function with infinite support $[0, \infty)$.

The k-th component of the firing distribution (10.2) is then computed as:

$$D_k(x|\boldsymbol{m}, \boldsymbol{Z}) \;=\; \int_0^x \prod_{\substack{j:t_j \in \boldsymbol{e}(\boldsymbol{m}) \\ j \neq k}} [1 - \phi_j(u|\boldsymbol{m}, \boldsymbol{Z})] d_u \phi_k(u|\boldsymbol{m}, \boldsymbol{Z}) \tag{10.9}$$

In this case,

$$p_k(\boldsymbol{m}, \boldsymbol{Z}) \;=\; \int_0^\infty \prod_{\substack{j:t_j \in \boldsymbol{e}(\boldsymbol{m}) \\ j \neq k}} [1 - \phi_j(x|\boldsymbol{m}, \boldsymbol{Z})] d_x \phi_k(x|\boldsymbol{m}, \boldsymbol{Z}) \tag{10.10}$$

and

$$F(x|\boldsymbol{m}, \boldsymbol{Z}) \;=\; 1 - \prod_{k:t_k \in \boldsymbol{e}(\boldsymbol{m})} [1 - \phi_k(x|\boldsymbol{m}, \boldsymbol{Z})] \tag{10.11}$$

For the solution of the model it is necessary to obtain the distributions $\phi_k(x|\boldsymbol{m}, \boldsymbol{Z})$ for each transition of the SPN, given the $G_k(x, \boldsymbol{m})$.

10.2.2 Conditioning on Past History $\boldsymbol{Z}$

Different ways for keeping track of the past behaviour of the net are possible. Assume that an *age variable* a_k, associated with each timed transition t_k, is defined that increases with the time in which the corresponding transition is

enabled. The way in which a_k is related to the past history determines the different memory policies. We consider the following three alternatives:

RESAMPLING: The age variable a_k is reset to zero at any change of marking. The firing distribution $\{D_k(x|\boldsymbol{m}, \boldsymbol{Z})\}$ is independent of $\boldsymbol{Z}$, but it may depend upon the current marking $\boldsymbol{m}$.

ENABLING MEMORY: The age variable a_k accounts for the time elapsed from the last epoch in which t_k has been enabled. The firing distribution depends on the past history $\boldsymbol{Z}$ through a_k. The k-th component $D_k(x|\boldsymbol{m}, \boldsymbol{Z})$ of the firing distribution depends on the distribution of the residual time needed for this activity to complete, given a_k. When transition t_k is disabled (even without firing), a_k is reset.

AGE MEMORY: The age variable a_k accounts for the total time in which t_k has been enabled from its last firing. The firing distribution depends on the past history $\boldsymbol{Z}$ through a_k. The k-th component $D_k(x|\boldsymbol{m}, \boldsymbol{Z})$ of the firing distribution depends on the distribution of the residual time needed for this activity to complete.

The race model with resampling (we shall refer to this case as R-R) can be used to describe the behaviour of a set of parallel competing activities (modeled by conflicting transitions) such that the first one to terminate determines a change in the system state. The work performed by those activities that do not complete is lost; the only work that is relevant for the model is that performed by the activity corresponding to the transition that fired leading to a change of state of the system. An example might be the parallel execution of hypotheses tests (see Fig. 10.1) where the R-R policy may be specified for transitions t_{test1}, t_{test2}, and t_{test3}. The process that terminates first is the one that has verified an hypothesis. Those hypotheses whose verifications were not completed need not be remembered; furthermore, it is not important to keep track of the point at which these other tests were interrupted since they are not going to be resumed. This model appears to be interesting only in the case of conflicting transitions. When we use this model in the case of concurrent transitions we easily obtain paradoxical system behaviours [1]. Even if the R-R policy is clearly of little interest in practical applications, we mentioned it because it was implicitly assumed in early attempts to extend the class of SPN to general time distributions in the framework of semi-Markov processes [33, 6].

The race model with enabling memory (R-E) is again used to describe the behaviour of a set of simultaneous activities such that the first activity that terminates determines a change in the system state. In this case the work performed by those activities that do not complete is lost, unless the transitions to

Table 10.1: Specification of the transitions of the SPN of Fig. 10.1

transition	rate	semantics	firing policy
t_{submit}	λ	single-server	-
t_{test1}	μ_1	single-server	R-R
t_{test2}	μ_2	single-server	R-R
t_{test3}	μ_3	single-server	R-R
$t_{action1}$	γ_1	single-server	-
$t_{action2}$	γ_2	single-server	-
$t_{action3}$	γ_3	single-server	-

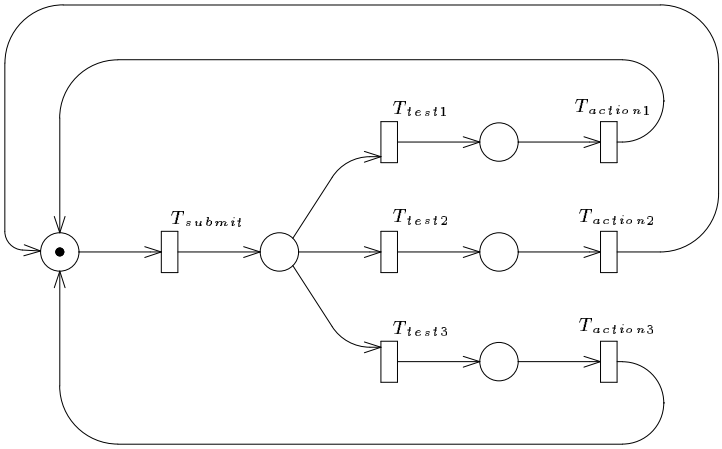


Figure 10.1: The SPN description of parallel hypothesis tests

which the activities correspond remain enabled in the new marking generated by the change of state. An example in this case can be provided by the behaviour of a concurrent program where several activities are performed in parallel through several intermediate steps up to a synchronization. Results from the parallel execution of independent tasks are collected together before the subsequent execution of a similar computational cycle. Each parallel activity is carried-on to its end independently of any change of state that derives from the completion of any intermediate step of the other computations (see Fig. 10.2) where the R-E firing policy may be specified for transitions t_{act1}, t_{act21}, t_{act22}, t_{act3}.

Table 10.2: Specification of the transitions of the SPN of Fig. 10.2

transition	rate	semantics	firing policy
t_{distr}	λ	single-server	-
t_{ch1}	μ_1	single-server	R-R
t_{ch2}	μ_2	single-server	R-R
t_{act11}	γ_{11}	single-server	R-E
t_{act12}	γ_{12}	single-server	R-E-
t_{act2}	γ_2	single-server	R-E-
t_{act31}	γ_{31}	single-server	R-E-
t_{act32}	γ_{32}	single-server	R-E-
$t_{collect}$	δ	single-server	-

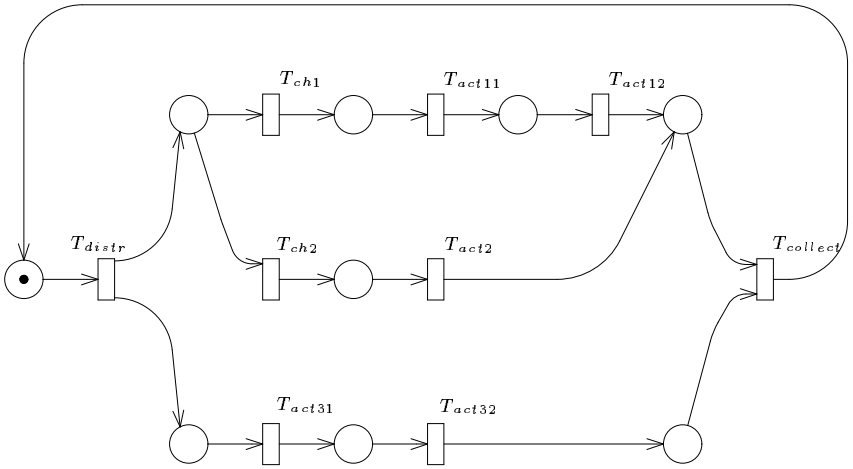


Figure 10.2: The SPN description of a parallel computation

The race model with age memory (R-A) may be used to describe the behaviour of a set of simultaneous activities such that the first activity to terminate determines a change in the system state. In this case, however, the work performed by those activities that do not complete is not lost. This implies that simultaneous activities are such that, after the firing of the first transition that completes, they will resume from the point at which they were interrupted, in the first marking that enables them; they may continue in the new state if they are still enabled. An example in this case can be provided by the execution of

Table 10.3: Specification of the transitions of the SPN of Fig. 10.3

transition	rate	semantics	firing policy
t_{submit}	λ	single-server	-
$t_{produce}$	μ	single-server	R-A
t_{fail}	γ	single-server	R-R
t_{repair}	δ	single-server	-

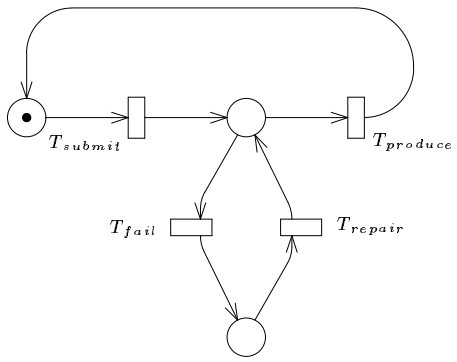


Figure 10.3: The SPN description of parallel hypothesis tests

an activity on a machine tool that may break-down. The work performed before the break down is continued after the repair has been completed (see Fig. 10.3) where the R-A policy, may be specified for transition $t_{produce}$.

For conflicting transitions R-R and R-E behave identically, however R-E allows the modeling of parallel activities with preemption disciplines.

The conditioning on $\boldsymbol{Z}$ is a property of each single transition of the net and a $GDSPN$ model may contain transitions belonging to all the three types of conditioning.

At the entrance in a new marking, the residual firing time is computed for each enabled timed transition given its age variable. The next marking is determined by the minimal residual firing time among the enabled timed transitions (*race policy*). Under an enabling memory policy the firing time of a transition is resampled from the original distribution each time the transition becomes enabled so that the time possibly spent without firing in prior enabling periods is lost. The memory of the underlying stochastic process cannot extend beyond a single cycle of enable/disable of the corresponding transition. On

the contrary, if a transition is assigned an age memory policy, the age variable accounts for all the periods of time in which the transition has been enabled, independent of the number of enable/disable cycles. Hence, the age memory is the only policy that allows a transition to be associated with a positive age variable also in markings in which it is not enabled.

Notice that all the above definitions apply also to the *SPN*s considered in Chapter 9. In particular, since exponential transitions do not have memory, the residual life time distribution is independent of the value of the age variable. Hence, the three policies have the same effect and we can conventionally assume that the age variables associated with exponential transitions are identically zero.

The marginal distributions $\phi_k(x|\boldsymbol{m}, \boldsymbol{Z})$ of equation (10.8) must be computed from the individual distributions $G_k(x|\boldsymbol{m})$ and from the knowledge of the memory policy associated with t_k.

10.2.3 Marking Dependence

The derivation of the marginal distributions $\phi_k(x|\boldsymbol{m}, \boldsymbol{Z})$ from the individual firing time distributions $G_k(x|\boldsymbol{m})$ associated with each transition of the net, must account for the way in which these last distributions depend on the marking of the net. In general, the specification of these distributions requires an a-priori knowledge of the reachability set of the net. Special cases exist in which the description of this marking dependence can be provided at the net level.

Three different cases of marking dependence that are considered useful for the specification of *GDSPN* models are the following:

1. *The individual distributions do not depend on the marking*:

$$G_k(x|\boldsymbol{m}) \;=\; G_k(x) \tag{10.12}$$

   The marginal distributions in (10.8) are the residual life distributions of t_k conditioned on a_k:

$$\phi_k(x|\boldsymbol{m}, \boldsymbol{Z}) \;=\; G_k^r(x|a_k) \;=\; \frac{G_k(x + a_k) \;-\; G_k(a_k)}{1 \;-\; G_k(a_k)} \tag{10.13}$$

2. *Marking dependence through a scaling factor*:

   In this case a_k depends both on the time interval spent in each marking in which t_k was enabled without firing and on the distribution $G_k(x|\boldsymbol{m})$ of t_k in that particular marking. We consider only the possibility that the dependence is reflected on the distribution by means of a scaling factor $\beta(\boldsymbol{m}) > 0$, so that

$$G_k(x|\boldsymbol{m}) \;=\; G_k(\beta(\boldsymbol{m})x) \tag{10.14}$$

Table 10.4: Specification of the transitions of the SPN of Fig. 10.4

transition	rate	distribution	semantics	firing policy
t_h	λ	Neg. Exp.	single-server	-
t_k	μ_1	Er.2	single-server	*R-R*

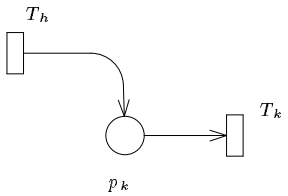


Figure 10.4: A case of marking dependence

As a possible instance of the process, let us suppose that transition t_k is first enabled in marking $\boldsymbol{m}_{i1}$, in which it spends x_1 time units; then, without firing, it enters marking $\boldsymbol{m}_{i2}$. The age a_k of t_k in $\boldsymbol{m}_{i2}$ given it has worked x_1 time units in $\boldsymbol{m}_{i1}$ is computed from the following equation:

$$a_k \;=\; x_1 \,\frac{\beta(\boldsymbol{m}_{i1})}{\beta(\boldsymbol{m}_{i2})} \tag{10.15}$$

where $x_1\beta(\boldsymbol{m}_{i1})$ can be interpreted as a measure of the quantity of work performed before the change of state in marking $\boldsymbol{m}_{i1}$.

The marginal distributions in (10.8) are thus formally expressed by an equation similar to (10.13) where the residual life distributions of each transition t_k at time x is obtained by introducing the transition age a_k calculated using (10.15):

$$\phi_k(x|\boldsymbol{m},\boldsymbol{Z}) \;=\; \phi_k(x|x_1,\boldsymbol{m}_{i1},x_2,\boldsymbol{m}_{i2},...) \;=\; G_k^r(x|a_k) \tag{10.16}$$

3. *More general kind of marking dependence*:

The marking dependence discussed above is well suited to represent the case of a change of speed in the execution of an activity consequent to a change of the net marking. More general cases are not difficult to conceive, but are difficult to include in a single general framework. This does not mean that they are intractable when the underlying behaviour of the net is well understood. For example, consider a transition t_k with a single input place p_i which models the execution of some activity (see Fig. 10.4).

The net is such that if several tokens reside in p_i each one of them represents an activity that can proceed in parallel with the others (t_k is an "infinite server" transition that models random delays without congestion); the execution policy be R-A, and the distribution associated with the activity be an Erlang of order 2. Assume now that one token is in p_k for some time while the net is in marking $\boldsymbol{m}_i$, and that some other transition fires changing the marking to $\boldsymbol{m}_j$ such that another token enters p_k. The firing distribution of t_k in the newly entered marking must be calculated as the distribution of the minimum between the residual firing time associated with the activity that was already in progress, and the whole activity duration. Thus, while in $\boldsymbol{m}_i$ the firing distribution is Erlang of order 2, in $\boldsymbol{m}_j$ it has a different form, that can be calculated from the knowledge of the model semantics.

10.3 Computational Restrictions of the GDSPN

The age variables a_k defined in the Section 10.1 make the stochastic process Markovian with partly discrete (states in $R(\boldsymbol{m}_0)$) and partly continuous (the Cartesian product of the age variables domains) state space.

Despites this observation, it must be emphasized that the numerical calculation of the measures pertinent to the process is quite intractable as long as the individual distributions are of generic form.

For these reasons, various restrictions of the general model have been discussed in the literature [16, 10] in order to insure that the underlying marking process $\mathcal{M}(\tau)$ belongs to a known class of analytically tractable stochastic processes.

10.3.1 Semi-Markov SPN

When all the *PN* transitions are assigned a resampling policy, the marking process becomes a semi-Markov process. This restriction has been studied in [33, 6], but, as we already observed at the beginning of this chapter, it is of little interest in applications where it is difficult to imagine situations in which the firing of each transition of the *PN* has the effect of forcing a resampling to all the other transitions.

A more interesting semi-Markov *SPN* model has been discussed in [22]. In this definition, the transitions are partitioned into three classes: exclusive, competitive and concurrent. Provided that the firing time of all concurrent transitions is exponentially distributed and that non-exponential competitive transitions are resampled at the time the transition is enabled, the associated marking process becomes a semi-Markov process.

10.3.2 Deterministic SPN

The *Deterministic and Stochastic PN* model has been introduced by Ajmone-Marsan and Chiola in [4], with the aim of providing a technique for considering stochastic systems in which some time variables assume a constant value. In [4] only the steady-state solution has been addressed. An improved algorithm for the evaluation of the steady-state probabilities has been successively presented in [31], and some structural extensions have been proposed in [17].

Definition 3 - *A DSPN is a GDSPN in which:*

- *The set T of transition is partitioned into a subset T_e of exponential transitions (EXP) and a subset T_d of deterministic transitions (DET), such that $T = T_e \cup T_d$* [4].
- *An exponentially distributed random variable θ_j. is associated with any EXP transition $t_j \in T_e$.*
- *A deterministic firing time d_k is associated with any DET transition $t_k \in T_d$.*
- *At most, a single DET transition is allowed to be enabled in each marking.*
- *The only allowed execution policy for the DET transition is the* race *policy with* enabling memory.

According to Definition 3, during the firing of a DET transition, the marking process can undergo EXP transitions only, thus describing a *CTMC* called the subordinated process. The steady-state solution technique, originally proposed in [4], is based on the evaluation of the subordinated *CTMC* at a time corresponding to the duration of the DET transition.

The idea is that of partitioning the reachability set of a *DSPN* into disjoint sets of markings according the deterministic transitions that they enable. Focussing on an arbitrary deterministic transition t_k, we can identify with $RSD_k(\boldsymbol{m}_0)$ [5] the set of markings in which deterministic transition t_k is enabled:

$$RSD_k \;=\; \{\boldsymbol{m}_i \;:\; t_k \in \boldsymbol{e}(\boldsymbol{m}_i)\} \quad . \tag{10.17}$$

The set of markings in which deterministic transitions are enabled is thus:

$$\begin{array}{lcl} RSD & = & \bigcup_{t_k \in T_d} RSD_k \\ & & with \quad RSD_k \bigcap RSD_l \;=\; \emptyset,\; t_k \neq t_l,\; t_k, t_l \in T_d \end{array} \tag{10.18}$$

RSE denotes instead, the set of markings in which only exponential transitions are enabled. Assuming that the markings are ordered in such a way that those

[4] In the original model $T_e \cup T_d$ represents the set of timed transitions. As we already pointed out, we neglect the existance of immediate transitions to keep the discussion simpler.

[5] From now on, we will omit the indication of the initial marking $\boldsymbol{m}_0$ in all the notation that we are using in this chapter, unless when really needed.

of RSD come first, we can write the transition rate matrix of the model in the following way:

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{D} & \boldsymbol{P} \\ \boldsymbol{A} & \boldsymbol{B} \end{bmatrix} \tag{10.19}$$

where $\boldsymbol{D}$ contains the firing rates of the exponential transitions that fire concurrently with the deterministic ones without disabling them; $\boldsymbol{P}$ contains instead the transition rates coming from the exponential transitions whose firings disable a deterministic transition (i.e., the deterministic transition is *preempted*). $\boldsymbol{A}$ and $\boldsymbol{B}$ contain the transition rates coming from the transitions that are enabled in markings of RSE. Moreover, if we assume that an ordering exists among the markings of RSD so that those of RSD_k are first, we can identify within the transition rate matrix $\boldsymbol{Q}$ a component matrix $\boldsymbol{Q}_k$

$$\boldsymbol{Q}_k = \begin{bmatrix} \boldsymbol{D}_k & \boldsymbol{P}_k \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \tag{10.20}$$

that is the infinitesimal generator of the $CTMC$ subordinated to transition t_k [6].

Using the $CTMC$ results recalled in Chapter 9, we may observe that $e^{\boldsymbol{Q}_k d_k}$ represents the probability matrix of moving among the states of the subordinated $CTMC$ in time d_k. Using standard results, it is easy to notice that, starting from an arbitrary marking $\boldsymbol{m}_i$ in which the deterministic transition t_k is enabled ($\boldsymbol{m}_i \in RSD_k$), we have:

$$Pr\{t_k \; is \; preempted \mid \boldsymbol{m}_i\} = \boldsymbol{u}_i \; e^{Q_k d_k} \; \boldsymbol{u}_e^T \tag{10.21}$$

where $\boldsymbol{u}_i$ is a unity row vector of suitable size, whose components are all zero, but the i-th one that is equal to one and $\boldsymbol{u}_e$ is a row vector such that:

$$u_{ej} = \begin{cases} 1 & if \; j \notin RSD_k \\ 0 & \text{otherwise} \end{cases} \tag{10.22}$$

Since the disabling of transition t_k can also occurr because of its own firing, we must account for this event by defining a matrix $\boldsymbol{\Delta}_k$ that collects the transition probabilities due to such an event:

$$\boldsymbol{\Delta}_k = \begin{bmatrix} \boldsymbol{\Delta}_{DD} & \boldsymbol{\Delta}_{DP} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \tag{10.23}$$

A generic component of $\boldsymbol{\Delta}_{DD}$ represents the probability that the firing of the deterministic transition t_k yields a marking in which t_k is still enabled.

[6] Notice that different markings of RSD_k in which transition t_k is newly enabled, may yield different CTMCs that are all subordinated to (the firing of) transition t_k. Due to the restrictions imposed on this model (Definition 3), these subordianated CTMC are all independent and the computation that follows can be carried on as if there was only one CTMC subordinated to transition t_k.

Conversely, $\mathbf{\Delta}_{DP}$ contains the probabilities that, upon firing of t_k, the *DSPN* moves into a marking in which t_k is no more enabled. With this new notation we have:

$$Pr\{t_k \ fires \mid \boldsymbol{m}_i\} \ = \ \boldsymbol{u}_i \ e^{\boldsymbol{Q}_k d_k} \ \mathbf{\Delta}_k \ \boldsymbol{u}_e^T \tag{10.24}$$

The derivation of these preliminary results suggests that an Embedded Markov Chain (EMC) can be identified within the marking process of a *DSPN* by sampling its behaviour at the firing instants of
i) any exponential transition that is neither concurrent nor competitive with a deterministic transition,
ii) of exclusively enabled deterministic transitions,
iii) and of competitive exponential transitions that preempt (i.e., whose firings disable) a deterministic transition.

The state space (SS) of the EMC is thus made of all the markings of the RS of the original *DSPN* model, in which either only exponential transitions are enabled (RSE) or a detrministic transition become newly enabled. Note that a change of marking in the *DSPN* model, due to the firing of an exponential transition concurrently enabled with a deterministic one is not represented in the EMC so that certain states of the EMC correspond to several markings of the *DSPN*. Identifying with N_m and N_s the cardinalities of the RS of the *DSPN* and of the SS of the corresponding EMC, we can observe that $N_s \leq N_m$, that a transition probability matrix $\boldsymbol{\mathcal{P}}$ of dimensions $N_s x N_s$ can be derived for the EMC. Moreover, due to the difference between the state spaces of the original and embedded models, in order to derive the steady-state probability distribution $\boldsymbol{\pi}$ of the *DSPN* from the steady-state probability vector $\boldsymbol{\psi}$ of the EMC, a matrix $\boldsymbol{C}$ (of dimensions $N_s x N_m$) of *conversion factors* must be computed

The computation of $\boldsymbol{\mathcal{P}}$ and $\boldsymbol{C}$ requires that the initial states of the subordianted Markov chains be identified. These markings belong to the intersection between RSD and the state space SS of the EMC. Focusing our attention on the deterministic transitions t_k, the initial states of the CTMC subordinated to t_k are the marking that belong both to RSD_k and to SS (i.e., are those markings of RSD_k in which transition t_k is newly enabled). Identifying with $\boldsymbol{s}_i$ one of these states, the i-th row of the transition probability matrix $\boldsymbol{\mathcal{P}}$ can be computed using the following expression:

$$\mathcal{P}_i \ = \ \boldsymbol{u}_i \ e^{\boldsymbol{Q}_k d_k} \ \mathbf{\Delta}_k \tag{10.25}$$

For all the states $\boldsymbol{s}_j$ of the EMC that correspond to markings of the *DSPN* belonging to RSE, the components of $\boldsymbol{\mathcal{P}}$ can be computed using the following expression:

$$\mathcal{P}_{jl} \ = \ q_{kh}/q_k \tag{10.26}$$

where q_{kh} is the component of the transition rate matrix $\boldsymbol{Q}$ of the *DSPN* (see Eq. 10.19), $\boldsymbol{m}_k$ and $\boldsymbol{m}_h$ are the markings of the *DSPN* that correspond to states

$\boldsymbol{s}_j$ and $\boldsymbol{s}_l$ of the EMC, respectively, and q_k is the total rate of departure from marking $\boldsymbol{m}_k$.

From the definition of $\boldsymbol{Q}$ and of the infinitesimal generator of a Markov chain (Chapter 9, Section 1.1), we have:

$$q_k \;=\; -q_{kk} \;=\; \boldsymbol{u}_k \; \boldsymbol{Q} \; \mathbf{1}_k^{\mathrm{T}} \tag{10.27}$$

where $\boldsymbol{u}_k$ and $\mathbf{1}_k$ are vectors of suitable size that identify the k-th row of $\boldsymbol{Q}$ and that allow to sum all its components, but the k-th one (the diagonal element).

Addressing now the computation of the correcting factor matrix, we can observe that the sojourn times in markings that belong to RSE can easily be computed from the transition rate matrix $\boldsymbol{Q}$:

$$SJ_j \;=\; \frac{1}{q_k} \tag{10.28}$$

when marking $\boldsymbol{m}_k$ of the *DSPN* corresponds to the state $\boldsymbol{s}_j$ of the EMC. For the other markings of the *DSPN* we must observe that if $\boldsymbol{m}_h$ enables an exclusive deterministic transition t_k, the mean sojourn time of the corresponding state $\boldsymbol{s}_l$ of the EMC assumes the value $SJ_l = d_k$; if instead, a marking $\boldsymbol{m}_j$ belongs to the set $RSE\&D$ (i.e., it enables some exponential transitions that are either concurrent or competitive with a deterministic transition t_k and is thus part of the state space of the corresponding subordinated CTMC), its sojourn time must be computed from the solution of such a chain. Using standard results from the theory of CTMC with absorbing states [30], we have that the average sojourn time spent in the marking $\boldsymbol{m}_j$ of the CTMC subordinated to transition t_k, given that its initial state is $\boldsymbol{s}_i$ is (i.e., $\boldsymbol{m}_i \in SS(i)$):

$$SJ_j(i) \;=\; \int_0^{d_k} \boldsymbol{u}_i \; e^{\boldsymbol{Q}_k x} \; \boldsymbol{u}_j^T \; dx \;=\; \boldsymbol{u}_i \; e^{\boldsymbol{Q}_k d_k} \; \boldsymbol{u}_j^T \tag{10.29}$$

With similar arguments, we have that the average sojourn time in state $\boldsymbol{s}_i$ is:

$$SJ(i) \;=\; \boldsymbol{u}_i \; e^{\boldsymbol{Q}_k d_k} \; \boldsymbol{u}_d^T \tag{10.30}$$

where $\boldsymbol{u}_d$ is a row vector defined as:

$$u_{dj} \;=\; \begin{cases} 1 & if \;\; j \;\in\; RSD_k \\ 0 & \text{otherwise} \end{cases} \tag{10.31}$$

Putting all these results together, we can compute the correcting factor matrix using the following definitions.

For any state $\boldsymbol{s}_i \notin RSE\&D$

$$c_{ij} = \begin{cases} 1 & if\ i = j \\ 0 & \text{otherwise} \end{cases} \tag{10.32}$$

For any state $\boldsymbol{s}_i \in RSE\&D$

$$c_{ij} = \frac{SJ_j(i)}{SJ(i)} \tag{10.33}$$

Having completed the construction of the EMC, we can compute its steady-state probability distribution solving the following system of linear equations:

$$\begin{cases} \boldsymbol{\psi} = \boldsymbol{\psi}\ \boldsymbol{\mathcal{P}} \\ \\ \boldsymbol{\psi}\ \mathbf{1}^{\mathrm{T}} = 1 \end{cases} \tag{10.34}$$

From the solution of the embedded model, the steady-state probability distribution of the original *DSPN* model can thus be computed deriving first the following set of un-normalized quantities:

$$\pi'_j = \begin{cases} \psi_j\ SJ_j & \boldsymbol{m}_j \notin RSE\&D \\ \sum_{i \in SS(i)} \psi_i\ SJ(i)\ c_{ij} & \text{otherwise} \end{cases} \tag{10.35}$$

and computing then the normalized distribution using the following expression:

$$\pi_j = \frac{\pi'_j}{\sum_{l \in RS}\ \pi'_l} \tag{10.36}$$

A Simple Example

This solution method that we have just presented can be better understood by carrying on the analysis of a simple queueing model (the M/D/1/2/2 queue) that can be represented with the *DSPN* of Fig. 10.5 where transition $t_{arrival}$ represents the arrival process by assuming that its firing rate is λ and that it has an infinite-server firing policy; transition $t_{service}$ instead represents the deterministic server characterized by a fixed service time d and by a single-server firing policy [7].

The reachability set of this model consists of only three markings and the corresponding reachability graph is that depicted in Fig. 10.6.

For this model we have:

$$RSD_2 = \{\boldsymbol{m}_1,\ \boldsymbol{m}_2\}, \qquad RSE = \{\boldsymbol{m}_3\} \tag{10.37}$$

[7] In the figures that follow, EXP transitions are drawn as empty rectangles and DET transitions as filled rectangles.

Table 10.5: Specification of the transitions of the SPN of Fig. 10.5

transition	rate	distribution	semantics	firing policy
$t_{arrival}$	λ	Neg. Exp.	infinite-server	-
$t_{service}$	$1/d$	Deterministic	single-server	R-R

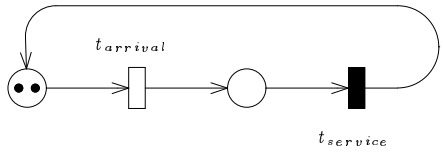


Figure 10.5: *DSPN* representation of a M/D/1/2/2 queueing system.

moreover, transition t_2 cannot be preempted, but its activity can be concurrent with that of transition t_1.

Using the definitions introduced in the previous section, we have:

$$Q_2 = \begin{bmatrix} -\lambda & \lambda & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \Delta_2 = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{10.38}$$

where $x = e^{\lambda d}$.

In this case the computation of e^{Qd} is quite simple, so that the transition probability matrix of the EMC becomes:

$$\mathcal{P} = \begin{bmatrix} 1-x & 0 & x \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{10.39}$$

As shown by this probability transition matrix, the EMC of this model has only two recurrent states $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ which are characterized by the firings of transitions t_2 and t_1, respectively. In particular, the state $\boldsymbol{s}_1$ of the EMC corresponds to the markings $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ of the original model. The steady-state probability distribution of the EMC is easily computed yielding:

$$\psi = \left[\frac{1}{1+x},\ 0,\ \frac{x}{1+x} \right] \tag{10.40}$$

The average time spent by the net in state $\boldsymbol{s}_1$ of the EMC is equal to d, while the EMC spends in average $1/2\lambda$ units of time in state $\boldsymbol{s}_3$. Since the

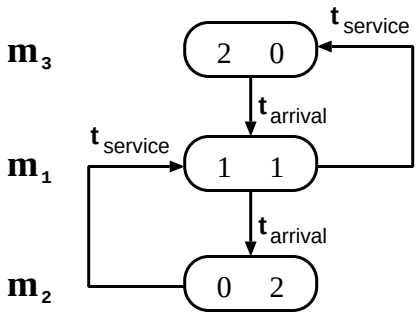


Figure 10.6: The reachability graph of the *DSPN* of Fig. 10.5.

EMC state $\boldsymbol{s}_1$ corresponds to markings $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ of the original model, the "redistribution" of the time spent in $\boldsymbol{s}_1$ in markings $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ requires the computation of the following correction factor

$$c \;=\; \frac{1}{d}\int_0^d e^{-\lambda x}\,dx \;=\; \frac{1 \;-\; e^{-\lambda d}}{\lambda d} \tag{10.41}$$

from which we can finally derive the steady-state distribution of the *DSPN* model:

$$\begin{array}{l} \pi_1 \;=\; \alpha\psi_1 dc, \qquad \pi_2 \;=\; \alpha\psi_1 d(1-c), \qquad \pi_3 \;=\; \alpha\psi_3\frac{1}{2\lambda} \\ where\ \alpha \;=\; [\psi_1 d \;+\; \psi_3/2\lambda]^{-1} \end{array} \tag{10.42}$$

Generalizations

Various improvements and extensions of the original algorithm proposed for the *DSPN* model are in [31, 17]. Choi et al. [14] have shown that the marking process associated with a *DSPN* is a Markov regenerative process (*MRP*), for which steady-state and transient solution equations are available [18]. In order to prove their assertion, Choi et al. have introduced the following modified execution sequence:

$$\mathcal{T}_E \;=\; \{\,(\tau_0^*, \boldsymbol{m}_{(0)})\,;\,(\tau_1^*, \boldsymbol{m}_{(1)})\,;\,\ldots\,;\,(\tau_i^*, \boldsymbol{m}_{(i)})\,;\,\ldots\} \tag{10.43}$$

Epoch τ_{i+1}^* is derived from τ_i^* as follows:

1. If no DET transition is enabled in marking $\boldsymbol{m}_{(i)}$, define τ_{i+1}^* to be the first time after τ_i^* that a state change occurs.

2. If a DET transition is enabled in marking $\boldsymbol{m}_{(i)}$, define τ^*_{i+1} to be the time when the DET transition fires or is disabled as a consequence of the firing of a competitive EXP transition.

Given the restrictions (Definition 3) of the class of models we are dealing with in this section, no other cases have to be considered. According to case 2) of the above definition, during $[\tau^*_i,\ \tau^*_{i+1})$, the *PN* can evolve in the subset of $RS(\boldsymbol{m}_0)$ reachable from $\boldsymbol{m}_{(i)}$, through EXP transitions concurrent with the given DET transition. The marking process during this time interval is the *CTMC* subordinated to marking $\boldsymbol{m}_{(i)}$. Therefore, if a DET transition is enabled in $\boldsymbol{m}_{(i)}$, the sojourn time is given by the minimum between the first passage time out of the subordinated *CTMC* and the constant firing time associated with the DET transition.

Choi et al. show that the τ^*_i form a sequence of regenerative time points, so that the marking process $\mathcal{M}(\tau)$ is a Markov regenerative process *MRP*. Distances between subsequent regenerative points (i.e., $[\tau^*_i,\ \tau^*_{i+1})$) are called *regeneration intervals*. The proof of this property corresponds to showing in a quite straightforward manner that, embedded in the marking process of these models, we can identify a Markov renewal sequence such that:

$$\begin{aligned} Pr\{\boldsymbol{m}_{(n+1)} = j, (\tau^*_{n+1} - \tau^*_n) \le \tau \mid \boldsymbol{m}_{(n)} = i, \tau^*_n;\ \boldsymbol{m}_{(n-1)}, \tau^*_{n-1};\ \ldots \boldsymbol{m}_{(0)}, \tau^*_0\} \\ = \ Pr\{\boldsymbol{m}_{(1)} = j, \tau^*_1 \le \tau \mid \boldsymbol{m}_{(0)} = i\} \end{aligned} \tag{10.44}$$

Moreover, the evolution of the marking process can be expressed in the following way:

$$\begin{aligned} \{\boldsymbol{\mathcal{M}}(\tau^*_n + \tau),\ \tau \ge 0 \mid \boldsymbol{\mathcal{M}}(u),\ 0 \le u \le \tau,\ \boldsymbol{\mathcal{M}}(\tau^*_n) = i\} \\ = \{\boldsymbol{\mathcal{M}}(\tau),\ \tau \ge 0 \mid \boldsymbol{\mathcal{M}}(0) = i\} \end{aligned} \tag{10.45}$$

where the equality means that the two marking processes have the same distributions.

From these expressions appears quite clearly that a central role is played by the following conditional probabilities:

$$\mathbf{K}(\tau) = [k_{ij}(\tau)] \quad \text{such that} \quad k_{ij}(\tau) = Pr\{\boldsymbol{m}_{(1)} = j\ ,\ \tau^*_1 \le \tau | \mathcal{M}(0) = i\} \tag{10.46}$$

The matrix $\mathbf{K}(\tau)$ is the *global kernel* of the *MRP* and provides the probability that the regeneration interval ends in marking j at time τ, given that it started in marking i at $\tau = 0$.

The distribution function of the first regeneration interval τ^*_1, starting from marking i is defined:

$$\begin{aligned} \mathbf{H}(\tau) = [h_{ij}(\tau)] \quad \text{such that} \quad h_{ij}(\tau) \ &= \ Pr\{\tau^*_1 \le \tau | \mathcal{M}(0) = i\} \\ &= \ \textstyle\sum_{j \in RS} k_{ij}(\tau) \end{aligned} \tag{10.47}$$

The sequence $\{\boldsymbol{m}_{(n)},\ n \geq 0\}$ is the embedded Markov chain of the *DSPN* model with one-step transition probability matrix

$$\boldsymbol{\mathcal{P}} \ = \ \lim_{\tau\to\infty} \boldsymbol{k}(\tau) \tag{10.48}$$

In [15, 18], a complete discussion of this problem can be found where for the transient analysis of the *DSPN* model two additional matrix valued functions are defined:

$$\begin{aligned} &\mathbf{V}(\tau) = [V_{ij}(\tau)] \quad \text{such that} \quad V_{ij}(\tau) = Pr\{\mathcal{M}(\tau) = j \,|\, \mathcal{M}(0) = i\} \\ &\mathbf{E}(\tau) = [E_{ij}(\tau)] \qquad " \qquad E_{ij}(\tau) = Pr\{\mathcal{M}(\tau) = j\ ,\ \tau_1^* > \tau | \,\mathcal{M}(0) = i\} \end{aligned} \tag{10.49}$$

Matrix $\mathbf{V}(\tau)$ is the transition probability matrix and provides the probability that the marking process $\mathcal{M}(\tau)$ is in marking j at time τ given it was in i at $\tau = 0$. Matrix $\mathbf{E}(\tau)$ is instead the *local kernel* and describes the behaviour of the marking process inside two consecutive regeneration time points. These quantities derive from the observation that, given that the marking process is Markov regenerative and that the execution sequence $\mathcal{T}_E$ is Markov renewal, the following relation holds:

$$\begin{aligned} &Pr\{\boldsymbol{\mathcal{M}}(\tau) = j \mid \boldsymbol{m}_{(0)} = i, \boldsymbol{m}_{(1)} = k, \tau_1^* = x\} \\ &\qquad = \begin{cases} Pr\{\boldsymbol{\mathcal{M}}(\tau) = j \mid \boldsymbol{m}_{(0)} = i, \tau_1^* = x\} & if\ \tau \ \leq \ x \\ v_{kj}(\tau - x) & if\ \tau \ > \ x \end{cases} \end{aligned} \tag{10.50}$$

The definition of $v_{ij}(\tau)$ derives from the unconditioning of this result:

$$\begin{aligned} v_{ij}(\tau) \ &= \ \textstyle\sum_{k\in RS} \int_0^\infty Pr\{\boldsymbol{\mathcal{M}}(\tau) = j \mid \boldsymbol{m}_{(0)} = i, \boldsymbol{m}_{(1)} = k, \tau_1^* = x\} dk_{ik}(x) \\ &= \ \textstyle\sum_{k\in RS} \int_t^\infty Pr\{\boldsymbol{\mathcal{M}}(\tau) = j \mid \boldsymbol{m}_{(0)} = i, \tau_1^* = x\} dk_{ik}(x) \\ &\quad + \qquad\qquad\qquad\qquad \textstyle\sum_{k\in RS} \int_0^t v_{kj}(\tau - x) dk_{ik}(x) \\ &= \ Pr\{\boldsymbol{\mathcal{M}}(\tau) = j,\ \tau_1^* > \tau \mid \boldsymbol{m}_{(0)} = i\},\ + \textstyle\sum_{k\in RS} k_{ik} * v_{kj}(\tau) \\ &= \ e_{ij}(\tau) \ + \ [k * v(\tau)]_{ij} \end{aligned} \tag{10.51}$$

This last result, expressed in matrix form, becomes:

$$\mathbf{V}(\tau) \ = \ \mathbf{E}(\tau) \ + \ \mathbf{K} * \mathbf{V}(\tau) \tag{10.52}$$

Equation (10.52) can be solved numerically in the time domain, once the explicit expressions of all the components of $\boldsymbol{K}$, $\boldsymbol{V}$, and $\boldsymbol{E}$ are computed following the detailed indications contained in [15].

An alternative approach, suggested in [14], consists in transforming the convolution equation (10.52) in the Laplace domain. By denoting the Laplace Stieltjes transform *(LST)* of a function $F(x)$ by $F^{\sim}(s) = \int_0^{\infty} e^{-sx}\, dF(x)$, Equation (10.52) becomes:

$$\mathbf{V}^{\sim}(s) \;=\; \mathbf{E}^{\sim}(s) \;+\; \mathbf{K}^{\sim}(s)\; \mathbf{V}^{\sim}(s) \tag{10.53}$$

whose solution is:

$$\mathbf{V}^{\sim}(s) \;=\; [\mathbf{I} \;-\; \mathbf{K}^{\sim}(s)]^{-1}\; \mathbf{E}^{\sim}(s) \tag{10.54}$$

The time domain solution of (10.54) can be obtained by numerical inversion [29].

The steady-state analysis of the model is quite straightforward as it follows the procedure originally proposed in [4] and recalled at the beginning of this section. Having recognized that the markings at the regeneration points are the states of an EMC, the probability distribution of the EMC is obtained by solving the usual system of linear equations:

$$\begin{cases} \boldsymbol{\psi} \;=\; \boldsymbol{\psi}\; \boldsymbol{\mathcal{P}} \\ \\ \boldsymbol{\psi}\; \mathbf{1}^{\mathrm{T}} \;=\; 1 \end{cases} \tag{10.55}$$

From the probability distribution of the regeneration intervals, we compute their average length as:

$$\gamma(i) \;=\; E\,[\tau_1^* \mid \mathcal{M}(0) = i] \tag{10.56}$$

Considering that also in this case we may have for each state of the EMC a subordinated CTMC, we compute the sojourn time of the marking process in the states of the subordinate CTMC using the following definition:

$$SJ_j(i) \;=\; E\,\big[\;\text{sojourn time in marking}\quad \boldsymbol{m} = j\quad \text{during}\quad [0,\ \tau_1^*) \mid \mathcal{M}_{(0)} = i\big] \tag{10.57}$$

and compute the correcting factors

$$c_{ij} \;=\; \frac{SJ_j(i)}{\gamma(i)} \tag{10.58}$$

As we observed before, the values of $\gamma(i)$ are easy to compute when in the initial marking either no deterministic transition is enabled or only en exclusive deterministic transition is enabled (notice that in these cases no correction factors are needed). In the case instead that the marking process starts from a marking that is also the starting point for the evolution of a subordinated CTMC (with state space $SS(i)$, the quantities of interest are evaluated using the following expressions:

$$\gamma(i) \;=\; \sum\nolimits_{h \in SS(i)} \Big[\boldsymbol{u}_i\; e^{\boldsymbol{Q}(i)\, d_k}\Big]_h \tag{10.59}$$

where d_k is the firing time of the deterministic transition enabled in state i and $\boldsymbol{Q}(i)$ is the infinitesimal generator of the CTMC subordinated to transition t_k, starting from state i.

$$SJ_j(i) = \int_0^\infty e_{ij}(\tau)\, d\tau \tag{10.60}$$

Having computed these quantities, the steady-state solution of the original model is computed using again the results of Eq.s 10.35 and 10.36.

10.3.3 Markov Regenerative SPN (MRSPN)

A natural extension of the *DSPN* has been proposed in [15], where the DET transition in Definition 3 is replaced by a GEN transition. This model is referred to by the authors as *MRSPN**.

Definition 4. A *MRSPN* is a* GDSPN *in which:*

- *The set T of transitions is partitioned into a subset T_e of exponential transitions (EXP) and a subset T_g of generally distributed transitions (GEN), such that $T = T_e \cup T_g$.*
- *An exponentially distributed random variable θ_k is associated with any EXP transition $t_k \in T_e$.*
- *A generally distributed random variable θ_j is associated with any GEN transition $t_j \in T_d$.*
- *At most, a single GEN is allowed to be enabled in each marking.*
- *The only allowed execution policy for the GEN transition is the* race *policy with* enabling memory.

By Definition 4, during the firing of a GEN transition only EXP transitions can concurrently fire: the process subordinated to a GEN transition is a *CTMC*. The matrices $\mathbf{K}(\tau)$ and $\mathbf{E}(\tau)$ in (10.49) depend on the specific probability distributions considered in the model. In [15], closed form expressions are derived when the GEN transitions of the *MRSPN** have uniform distributions.

With the aim of extending the modeling power of *MRSPN*s by including GEN transitions with age memory policy, Bobbio and Telek [12, 11] have investigated a class of models characterized by the fact that the subordinated process between two consecutive regeneration time points is a Semi-Markov Reward Process [35]. A detailed discussion of the analysis of these type of models is beyond the scope of this book and we refer to the specialized literature for a deeper study of these methods.

10.3.4 Phase-Type SPN (PHSPN)

A numerically tractable realization of the *GDSPN*, is obtained by restricting the random firing times θ_k to have a Phase Type distribution (*PH*) [34]. *PH*

distributions are the distributions of the time till absorption of continuous-time homogeneous Markov chains with at least one absorbing state. Their most interesting feature, from our point of view, is the representation as finite-state Markov chains, that allows a natural discretization of the age variable introduced in Section 10.1.

The simplest subclasses of PH distributions, like Erlang, Hyperexponential (and trivially Exponential), are commonly encountered in various areas of applied stochastic modeling. The main characteristics of PH distributions are summarized in the appendix that is included at the end of this chapter.

Phase-Type *SPN*s thus satisfy the following:

Definition 5. *A PHSPN is a GDSPN in which:*

- *A PH random variable θ_k is associated with any timed transition $t_k \in T$. The PH model assigned to θ_k has ν_k stages with a single initial stage numbered stage 1 and a single final stage numbered stage ν_k.*

- *A memory policy of resampling, enabling or age memory type, is assigned to any timed transition $t_k \in T$.*

When the transition firing distributions are of *Phase-type*, the reachability graph $RG(\boldsymbol{m}_0)$ of the PN can be expanded into a discrete-state transition graph over which a continuous-time homogeneous Markov chain, equivalent to the original non-Markov process, can be defined. The measures pertinent to the original process can then be evaluated by solving the expanded Markov chain.

PH distributions have already been mentioned, in connection with SPN models, by Natkin [33] and Molloy [32]. The suggestion of these authors was to incorporate the PH model into the original SPN, by replacing a timed, PH distributed, transition with a proper SPN subnetwork whose reachability graph yields the Markov graph of the given PH distribution. Suggestions for the construction of sub-*PN*s able to account for all the three memory policies, but restricted to *PH* distributions with equal diagonal elements, are discussed in [13, 3] This approach suffers, however, from the drawback that the complexity of the SPN is artificially increased by fictitious nodes (both places and transitions) which do not refer to the operation of the system, but only to the distributions of the firing times. In the resulting reachability set the markings representing physical conditions of the system are dispersed among markings representing the passage of a transition through the succession of exponential stages. Moreover, it seems difficult to generalize the approach of [13] to devise a procedure for the automatic generation of the *PHSPN* model expanding the basic PN and taking into account general *PH* distributions and interactions among different memory policies.

10.3.5 Expansion of PH Distributions at the State Space Level

When needed, the stage expansion is conveniently performed on the SPN reachability graph RG, knowing for each transition its PH model which is itself a labeled directed graph.

The only limitation on the PH distributions $G_k(x|\boldsymbol{m})$ is that they should be given in a canonical form (always existing, as shown in [20] in which there is a single initial state (a single state with initial probability equal to one) numbered state 1, and a single final absorbing state numbered ν_{k+1}, where ν_k is the order of the PH distribution associated with t_k. A transition t_k is said to fire when it reaches the absorbing state ν_{k+1} in its PH model.

Each marking of the original reachability graph RG is expanded into a group of submarkings to account for the possible stages of firing reached by all the transitions of the net. Each submarking of $\boldsymbol{m}$ is defined as a pair $(\boldsymbol{m}; \boldsymbol{w})$ where $\boldsymbol{w}$ is a vector of n_t integers whose k-th component $1 \leq w_k \leq \nu_k$ represents the stage of firing of transition t_k. The pairs $(\boldsymbol{m}; \boldsymbol{w})$ are the states of the expanded Markov chain equivalent to the original process. With this new notation, the initial state of the process is assumed to be $(\boldsymbol{m}_0; 1, ..., 1)$.

The connections among the states resulting from the expansion are made according to the firing policies used in the net. The expansion of the reachability graph can be performed automatically by an enumeration algorithm in which the various firing policies can be accommodated [21]. The equivalence between the Markov chain resulting from the expansion algorithm and the original process defined over the reachability graph RG was shown in [8].

The distinguishing feature of this model, is that it is possible to devise an algorithm (and thus to design a completely automated tool) that responds to the requirements stated in [27], and that, at the same time, includes all the issues listed in Definition 2. The non-Markovian process generated by the *GDSPN* over the reachability set RS is converted into a *CTMC* defined over an expanded state space. The measures pertinent to the original process are defined at the *PN* level and can be evaluated by solving the expanded *CTMC*.

The expanded *CTMC* is represented by a directed graph $H = (N_H, A_H)$ where N_H is the set of nodes (states of the expanded *CTMC*) and A_H is the set of directed arcs (transitions of the expanded *CTMC*). The nodes are the pairs $(\boldsymbol{m}, \boldsymbol{w})$ introduced before.

Arcs in A_H are represented by 5-tuples $(N, N'; k, i, j)$, where N is the source node, N' the destination node, and (i, j) is an arc in the PH model of transition t_k. Therefore, $(N, N'; k, i, j) \in A_H$ means that in the expanded graph the process goes from node N to node N' when the stage of firing of t_k goes from stage i to stage j.

The expanded graph H is generated by an iterative algorithm [21]. Let H be initially empty; the algorithm starts by putting in N_H the initial node

$N_H^{(1)} = (\boldsymbol{m}_0, [1, 1, \ldots, 1])$ (the *PN* is in its initial marking $\boldsymbol{m}_0$ and all the n_t random variables θ_k are in stage 1). $N_H^{(1)}$ is marked as a non-expanded node. An expansion step is then performed on each non-expanded node $N_H^{(\ell)} = (\boldsymbol{m}^{(\ell)}, \boldsymbol{w}^{(\ell)})$. For each transition $t_k^{(\ell)}$ enabled in $\boldsymbol{m}^{(\ell)}$, the algorithm searches for all the possible successors of stage numbered $w_k^{(\ell)}$ in the *PH* model of $t_k^{(\ell)}$ (where $w_k^{(\ell)}$ is the kth component of $\boldsymbol{w}^{(\ell)}$, i.e. it is the stage of θ_k in state $N_H^{(\ell)}$). Let j ($j \leq \nu_k$) be one of such possible successors. Two cases may arise depending on whether $j \neq \nu_k$ or $j = \nu_k$; i.e. transition $t_k^{(\ell)}$ is either still in the process of firing or it has reached its terminal stage.

CASE 1 - $j \neq \nu_k$

Transition $t_k^{(\ell)}$ has made a jump in its *PH* model without firing. Then a new node $N_H'^{(\ell)} = (\boldsymbol{m}^{(\ell)}, \boldsymbol{w}'^{(\ell)})$ is generated, with $w_k'^{(\ell)} = j$ and $w_l'^{(\ell)} = w_l^{(\ell)}$ $(\forall\, t_l \in T\,;\, t_l \neq t_k)$.

CASE 2 - $j = \nu_k$

Transition $t_k^{(\ell)}$ has reached the final node of its *PH* model and thus has fired. In this case, the new node $N_H'^{(\ell)} = (\boldsymbol{m}'^{(\ell)}, \boldsymbol{w}'^{(\ell)})$ is generated according to the following rule: $\boldsymbol{m}'^{(\ell)}$ is the marking immediately reached from $\boldsymbol{m}^{(\ell)}$ by firing $t_k^{(\ell)}$ $(\boldsymbol{m}^{(\ell)} - t_k^{(\ell)} \rightarrow \boldsymbol{m}'^{(\ell)})$, and $w_k'^{(\ell)} = 1$ since firing of $t_k^{(\ell)}$ resets its stage count to 1. The values of the other entries of vector $\boldsymbol{w}'^{(\ell)}$, corresponding to the transitions enabled in $\boldsymbol{m}^{(\ell)}$ are set according to the memory policy attached to the corresponding transition:

- *always set equal to 1 in the resampling case;*
- *not modified in the age memory case;*
- *conditionally reset in the enabling memory case (i.e. if the transition is still enabled in the new marking $\boldsymbol{m}'^{(\ell)}$ the corresponding stage count is not modified otherwise is set to 1).*

In both cases, the new node $N_H'^{(\ell)} = (\boldsymbol{m}'^{(\ell)}, \boldsymbol{w}'^{(\ell)})$ is entered in N_H (if not already there) and a new arc $A_H'^{(\ell)} = (N_H^{(\ell)}, N_H'^{(\ell)}; k, w_k^{(\ell)}, w_k'^{(\ell)})$ is added to A_H. The above expansion step is iterated for all the transitions enabled in the current marking $\boldsymbol{m}^{(\ell)}$, and until all the corresponding *PH* distributions have reached their terminal stage. At this point the expansion of the node $N_H^{(\ell)}$ is terminated and the node itself is marked as expanded. The algorithm then searches for the subsequent non-expanded node until all the nodes have been searched for.

The cardinality n_H of the expanded state space is upper bounded by the product of the cardinality of the reachability set of the basic *PN* times the cardinality of the *PH* distributions of the n_t random variables θ_k [2]. The actual value of n_H is difficult to evaluate a priori. In practical cases this number can be very much lower that the upper bound outlined before, and is, in any case,

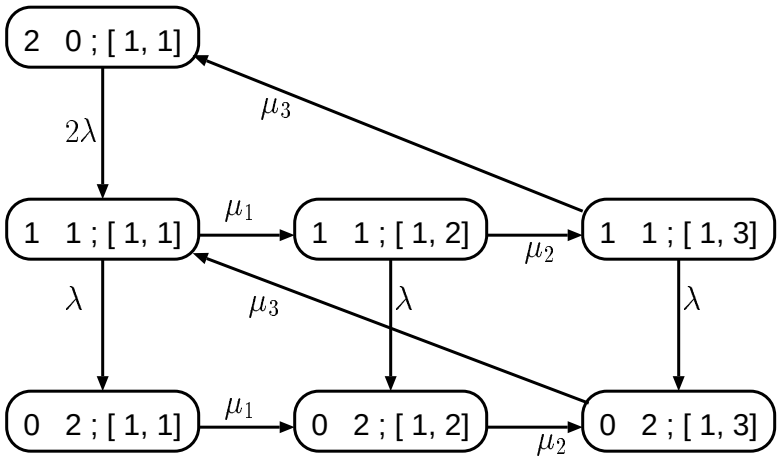


Figure 10.7: The expanded state space of the M/Er2/1/2/2 *PHSPN* model.

increasing with the complexity of the assigned memory policies. The resampling policy is the one that generates the expanded *CTMC* with the lower number of states, while the age policy generates the expanded *CTMC* with the larger number of states.

The marking $\boldsymbol{m}^{(\ell)}$ of the original reachability set, is mapped into a macro state formed by the union of all the nodes $N_H(\boldsymbol{m}, \boldsymbol{w})$ of the expanded graph such that $\boldsymbol{m} = \boldsymbol{m}^{(\ell)}$. This mapping allows to redefine the measures calculated as solution of the Markov equation over the expanded graph in terms of the markings of the original *PN*.

Figure 10.7 depicts the expanded state space of the model of Fig. 10.5 in which the service time has an Erlang-3 probability distribution. By introducing this modification the original *DSPN* model becomes a *PHSPN* whose analysis can be performed using the technique illustrated in this section. The marking process of the underlying PN, that stiil accounts for the three markings of Fig. 10.6 is now augmented with the submorking deriving from the different stages that compose the service. The rates of these individual stages are denoted by μ_1, μ_2, and μ_3 to explicitly identify them in the picture, even though the characteristics of the Erlang distribution are such that $\mu_1 = \mu_2 = \mu_3$.

10.4 Comparison of the Solution Methods

A discussion of the different features of the solution methods presented in this chapter is performed using the basis of the M/G/1/2/2 model that becomes either a *GDSPN* or a *PHSPN* depending on the characteristics of the service time distribution that is considered. A comparison between the computational

Table 10.6: Steady state probabilities for the non preemptive M/G/1/2/2 queue.

State	*DSPN*	*PHSPN*			
	(LTM)	*Erl(5)*	*Erl(10)*	*Erl(100)*	*Erl(1000)*
s_1	0.37754	0.38307	0.38039	0.37783	0.37757
s_2	0.48984	0.46773	0.47845	0.48867	0.48972
s_3	0.13262	0.14920	0.14116	0.13350	0.13271

characteristics of the Laplace Transform Method *(LTM)* applied to *MRSPN* (or *DSPN* in particular cases) and the expansion method typical of *PHSPN* models is made possible by observing that the Erlang distribution is the *PH* distribution with the minimum coefficient of variation [5] and it is thus appropriate to approximate the *DSPN*. By assigning to the transition that represents the service ($t_{service}$) an Erlang distribution of increasing order, it is possible to evaluate the improvements of the approximation, taking, however into accounts the progressively higher computational costs that are involved.

Table 10.6 shows the values of the steady-state probabilities calculated using the *LTM* for the *DSPN* and solving with the expansion method the *PHSPN* model. The results are obtained assuming that $t_{arrival}$ is EXP with firing rate $m_1 \cdot \lambda$ (being m_1 the number of tokens in its input place) and $\lambda = 0.5$ *job/hour*; $t_{service}$ is a DET transition modeling a constant service time of duration $d = 1.0\ hour$. In the *PHSPN* model, $t_{service}$ is assumed to be Erlang(5), Erlang(10), Erlang(100) and Erlang(1000), all with the same average service time that matches with the value $d = 1.0\ hours$ of the *DSPN* model, being all the other parameters unchanged. It should be stressed that the present case can be considered as a worst case example since a DET type variable can be closely approximated by a *PH* only as the number of stages grows to ∞ [5].

10.4.1 Computational complexity

Let us briefly summarize the elementary computational steps for the evaluation of the solution in the two considered methodologies (*MRSPN* and *PHSPN*). The *PHSPN* solution is fully supported by a tool [21], while the Laplace transform method for the transient analysis of *MRSPN* requires both manual and auto-

matic manipulation. The method of supplementary variables for *DSPN*, implemented in TimeNET [25], is also fully supported by a tool, but is presently restricted to the steady-state solution only.

- **Laplace Transform Method for MRSPN**
  Let us first suppose that all the GEN transitions are DET. The solution is obtained with a procedure summarized by the following steps [14]:

  1. generation of the reachability tree;
  2. manual derivation of the symbolical entries of the $\mathbf{K}^{\sim}(s)$ and $\mathbf{E}^{\sim}(s)$ matrices in the Laplace transform domain;
  3. symbolical matrix inversion and matrix multiplication by using a standard package (e.g. MATHEMATICA) in order to obtain the $\mathbf{V}^{\sim}(s)$ matrix (Equation 10.54) in the LT domain;
  4. time domain solution obtained by a numerical inversion of the entries of the $\mathbf{V}^{\sim}(s)$, resorting to the Jagerman's method [29]. For the sake of uniformity, this step has been implemented in MATHEMATICA language.

  In the GEN case, step 2. of the procedure must be replaced by

  2′. manual derivation of the symbolical entries of the $\mathbf{K}^{\sim}(s)$ and $\mathbf{E}^{\sim}(s)$ matrices in Laplace transform domain as in the DET case;

  2″. unconditioning of the entries of the $\mathbf{K}^{\sim}(s)$ and $\mathbf{E}^{\sim}(s)$ matrices, according to the form of the GEN distributions.

  Step 1) can be performed with any *PN* package. Step 2) is done manually, and its difficulty depends on the non-zero entries of the involved matrices, and on the complexity of the process subordinated to the dominant GEN transitions.

  The computational complexity of step 3) depends on the dimension of the matrices (i.e. the number of tangible markings) and the complexity of the elements of the kernels (the difficulty of step 3 is related to the difficulty of step 2). The complexity of the numerical inversion at step 4) also depends on two factors; the complexity of the function to invert, and the prescribed accuracy.

- Expansion Method for *PHSPN* Models
  For the evaluation of this model we must consider that the solution method consists of the following steps:

  1. generation of the reachability tree;
  2. generation of the expanded *CTMC*;
  3. solution of the resulting *CTMC*.

Step 1) is standard. The computational complexity of steps 2) and 3) depends on the number of tangible states and on the order of the *PH* distribution associated with each transition. As mentioned in Section 10.3.5, the cardinality of the expanded state space n_H is strongly influenced by the memory policies.

Even if the deterministic distribution is typically non *PH*, an approximation error for the steady-state probabilities of the order of 10^{-2} is reached by replacing the DET transition with an Erlang(5) and an error of the order of 10^{-4} by replacing the DET transition with an Erlang(1000). The use of *PH* distributions and of the *PHSPN* model offers the modeler a flexible tool for prescribing various interactions among the timed activities. Moreover, if the random variables of the system to be modeled are really of *PH* type, the *PHSPN* provides exact results. Otherwise, a preliminary step is needed in which the random times of the system are approximated by *PH* random variables resorting to a suitable estimation technique [9]. The expansion of the state space is, of course, a cause of non-negligible difficulties, since it worsens the problem of the exponential growth of the state space both with the model complexity, and with the order of the *PH* distribution assigned to each transition.

The *MRSPN* model, combining GEN (or DET) firing times with exponential firing times, offers an innovative approach in many practical applications. At the present state of the art, no automatized tools are available for the generation of the matrices $\mathbf{K}(x)$ and $\mathbf{E}(x)$ and for the solution of the convolution equation versus time. The Laplace transform technique, used in the examples, does not seem suited for a complete numerical automatization. An alternative numerical approach could be based on the direct solution of the convolution equation (10.52) in time domain or in the solution of a system of partial differential equations arising from the inclusion of supplementary variables [24].

Appendix A: Phase-type distribution

PH-distributions are an important class of distributions that can be seen as generalisations of the exponential distribution. In Figure 10.8 we depict the exponential distribution as an absorbing CTMC with two states. The initial probability distribution $\underline{p}(0) = (1, 0)$, and the time until absorption (in state 1) has an exponential distribution with rate λ. When we now generalise the single

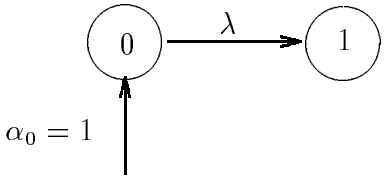


Figure 10.8: The exponential distribution as a phase-type distribution

state (state 0) of this CTMC into a number of states, but maintain the property that the CTMC is absorbing in a single state, we obtain a PH-distribution. The time until absorption consists of a number of phases, each of exponentially distributed length. The initial probability distribution is denoted $\underline{\alpha}$ (as far as the non-absorbing states are concerned).
To generalise this, consider a CTMC on the state space $\mathcal{I} = \{1, \cdots, m, m+1\}$, with generator matrix

$$\mathbf{Q} = \begin{pmatrix} \mathbf{T} & \underline{T}^0 \\ \underline{0} & 0 \end{pmatrix}, \tag{10.61}$$

where $\mathbf{T}$ is an $m \times m$ matrix with $t_{i,i} < 0$ $(i = 1, \cdots, m)$, $t_{i,j} \geq 0$ $(i \neq j)$ and $\underline{T}^0$ is a column vector with nonnegative elements. The row sums of $\mathbf{Q}$ equal zero, i.e., $\mathbf{T}\underline{1} + \underline{T}^0 = \underline{0}$. Notice that $\mathbf{T}$ by itself is not a proper generator matrix. The initial probability vector is given as $(\underline{\alpha}, \alpha_{m+1})$ with $\underline{\alpha}\,\underline{1} + \alpha_{m+1} = 1$. Furthermore, the states $1, \cdots, m$ are transient, and consequently, state $m+1$ is the one and only absorbing state, regardless of the initial probability vector. In Figure 10.9 we visualise this.

The probability distribution $F(x)$ of the time until absorption in state $m+1$ is then given as [34]:

$$F(x) = 1 - \underline{\alpha} e^{\mathbf{T}x} \underline{1}, \quad x \geq 0. \tag{10.62}$$

We now can define a distribution $F(x)$ on $[0, \infty)$ to be of phase type, if and only if it is the distribution of the time to absorption in a CTMC as defined above. The pair $(\underline{\alpha}, \mathbf{T})$ is called *a representation of* $F(x)$. Note that since $\mathbf{Q}$ is a generator matrix of which the row sums equal 0, the elements of $\underline{T}^0$ can be computed from $\mathbf{T}$. Then, the following properties hold:

- the distribution $F(x)$ has a jump of α_{m+1} at $x = 0$ and its density on $(0, \infty)$ equals

$$f(x) = F'(x) = \underline{\alpha} e^{\mathbf{T}x} \underline{T}^0.$$

- the moments $E[X^i]$ of $F(x)$ are finite and given by

$$E[X^i] = (-1)^i i! (\underline{\alpha} \mathbf{T}^{-i} \underline{1}), \quad i = 0, 1, \cdots. \tag{10.63}$$

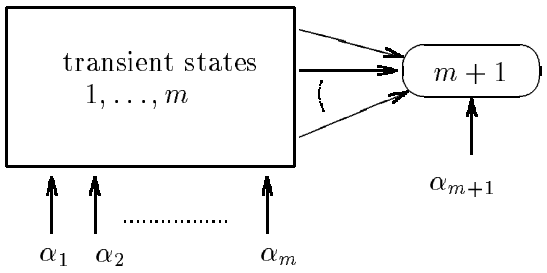


Figure 10.9: Schematic view of a PH distribution

Examples of PH-distributions are the earlier mentioned exponential distribution, the Erlang-k distribution, and the hyper- and hypoexponential distribution. Note that these well-known PH distributions are all represented by acyclic CTMCs. The definition above, however, also allows for the use of non-acyclic CTMCs.

Important to note is the fact that the first moment of a PH-distribution exactly expresses the *mean time to absorption* in an absorbing CTMC. Using (10.63) for that specific case, we obtain:

$$E[X] = -\underline{\alpha}\mathbf{T}^{-1}\underline{1}. \tag{10.64}$$

Note that the last multiplication (with $\underline{1}$) is just included to sum the elements in the row-vector $-\underline{\alpha}\mathbf{T}^{-1}$. We thus can also write:

$$E[X] = \sum_{i=1}^{m} x_i = \underline{x} \cdot \underline{1}, \quad \text{with} \quad \underline{x} = -\underline{\alpha}\mathbf{T}^{-1}, \tag{10.65}$$

where $\underline{x} = (x_1, \cdots, x_m)$ is a vector in which x_i denotes the mean time spent in state i before absorption. Instead of explicitly computing $\mathbf{T}^{-1}$ and performing a vector-matrix multiplication, it is often smarter to solve the linear system

$$\underline{x}\mathbf{T} = -\underline{\alpha}, \tag{10.66}$$

to obtain $\underline{x}$ and compute $E[X] = \sum_i x_i$. In a similar way we can compute the j-th moment $E[X^j] = \sum_i x_i^{(j)}$ where the vector $\underline{x}^{(j)}$ follows from

$$\underline{x}^{(j)}\mathbf{T}^j = (-1)^j j!\underline{\alpha}. \tag{10.67}$$

Suppose we have already computed $\underline{x}^{(j)}$ and want to compute $\underline{x}^{(j+1)}$, we then proceed as follows:

$$\underline{x}^{(j+1)}\mathbf{T}^{j+1} = (-1)^{j+1}(j+1)!\underline{\alpha} \tag{10.68}$$

Multiplying both sides of this equation with $\mathbf{T}^{-j}$ we obtain

$$\begin{aligned} \underline{x}^{(j+1)}\mathbf{T} &= (-1)^{j+1}(j+1)!\underline{\alpha}\mathbf{T}^{-j} \\ &= -(j+1)\underbrace{(-1)^j(j)!\underline{\alpha}\mathbf{T}^{-j}}_{\underline{x}^{(j)}} = -(j+1)\underline{x}^{(j)}, \end{aligned} \tag{10.69}$$

that is, we have obtained a linear system of equations that expresses $\underline{x}^{(j+1)}$ in terms of $\underline{x}^{(j)}$. If we solve this linear system of equations with a direct method such as LU-decomposition (see [36]) we only have to decompose $\mathbf{T}$ once and can compute successive vectors $\underline{x}^{(j)}$ using back-substitutions only. However, if $\mathbf{T}$ is large and we only require the first few moments, then an iterative solution might be the fastest way to proceed.

The Erlang-k distribution

The Erlang-k distribution is as a phase-type distribution consisting of a series of k exponential phases, followed by a single absorbing state. An Erlang-k distribution results when the sum of k independent and identically distributed exponential random variables is taken. The squared coefficient of variation of the Erlang-k distribution is $1/k$. Consequently, for large k, it approaches a deterministic arrival pattern. Erlang-k interarrival time distributions are often used to approximate true deterministic arrivals patterns (which cannot be incorporated in Markov models).

Instead of Erlang-k interarrival times, one can also define hypo-exponentially distributed interarrival times; they are the sum of a number of independent exponentially distributed random variables which need not be identically distributed. Hypo-exponentially distributed interarrival times also have a squared coefficient of variation at most equal to 1. It can be shown though, that no hypo-exponential distribution with k phases yields a coefficient of variation smaller than $1/k$, i.e., the Erlang-k distribution is the PH-distribution with the smallest coefficient of variation, for given k.

The hyperexponential distribution

Where the Erlang-k distribution is a series of exponential phases, the hyper-exponential distribution can be interpreted as a probabilistic choice between k exponential distributions (see Figure 10.10). As this probabilistic choice introduces extra randomness, one can imagine that for the H_k-renewal process the squared coefficient of variation is at least equal to 1. When modelling "bursty" traffic sources, this therefore seems to be a good choice.

Often, a two-phase hyperexponential distribution is used. The H_2-distribution has three free parameters, α, λ_1 and λ_2, so that

$$F_{H_2}(t) = 1 - \alpha e^{-\lambda_1 t} - (1-\alpha)e^{-\lambda_2 t}, \quad t \geq 0. \tag{10.70}$$

Fitting such a distribution on the first and second moment (or coefficient of variation) derived via measurements, leaves one free parameter. Therefore, often the H_2 distribution with balanced means is taken. The balanced means property can be expressed mathematically as $\alpha/\lambda_1 = (1-\alpha)/\lambda_2$. This extra equation can then be used to fit the interarrival time distribution.

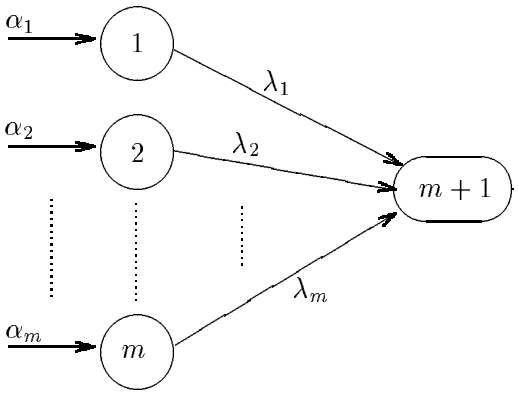


Figure 10.10: The hyperexponential distribution as a phase-type distribution

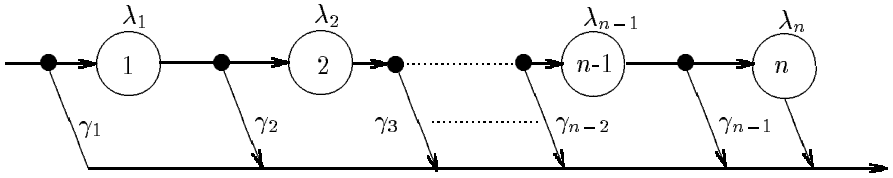


Figure 10.11: The Cox distributions as a mixed DTMC/CTMC state transition diagram

Suppose that measurements have revealed that the first moment of the interarrival times in a job stream can be estimated as $\hat{X}$, and that the squared coefficient of variation can be estimated as $\hat{C}_X^2$, the parameters of the H_2 distribution can be expressed as follows:

$$\alpha = \frac{1}{2} + \frac{1}{2}\sqrt{\frac{\hat{C}_X^2 - 1}{\hat{C}_X^2 + 1}}, \quad \lambda_1 = \frac{2\alpha}{\hat{X}}, \quad \lambda_2 = \frac{2(1-\alpha)}{\hat{X}}. \tag{10.71}$$

The Cox distribution

A well-known class of PH-distributions is formed by the Cox distributions. They can be regarded as a generalisation of the Erlang distributions. A Cox distribution consists of n exponential phases, with possibly different rates λ_1 through λ_n. Before each phase, it is decided whether the next phase is entered (with probability $1 - \gamma_i$) or not (with probability γ_i). The Cox distribution is visualised in Figure 10.11 where the small black diamonds indicate the probabilistic choices to be made before each exponential phase. With a proper choice of parameters, coefficients of variation both smaller and larger than 1 can be dealt with.

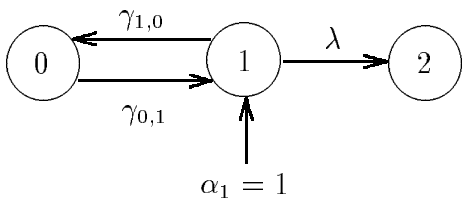


Figure 10.12: The interrenewal distribution of the IPP as a phase-type distribution

The interrupted Poisson process

In many telecommunication systems, the arrival stream of packets can be viewed as a Poisson stream, however, switched on and off randomly. The on-period then corresponds to an active period of a source, the off-period to a passive period of a source. The on- and off-periods are determined by the characteristics of the source. As an example of this, one can think of the stream of digitised voice packets that is originating from a telephone user: periods of speaking are interchanged with periods of silence. These two period durations determine the on- and off-time distributions. When on, the rate of digitised voice packets determines the rate of a Poisson process. When off, no arrivals are generated. The on/off-process is also called an interrupted Poisson process (IPP).

In Figure 10.12 we depict the interrenewal distribution of the IPP as a PH-distribution. Starting state is, deterministically, state 1, representing the on-state. In that state, a renewal period can end, i.e., an arrival can take place, via the transition with rate λ to the absorbing state. However, instead of an arrival, also the active period can end, according to the transition with rate $\gamma_{1,0}$ to state 0, the off-state. After an exponentially distributed off-period, with rate $\gamma_{0,1}$ the on-state is entered again. The generator matrix of this absorbing CTMC is given as

$$\mathbf{Q}_{\mathrm{IPP}} = \begin{pmatrix} -\gamma_{0,1} & \gamma_{0,1} & 0 \\ \gamma_{1,0} & -(\gamma_{1,0}+\lambda) & \lambda \\ 0 & 0 & 0 \end{pmatrix}, \tag{10.72}$$

and the initial probability vector $\underline{\alpha} = (0, 1)$. Note that due to the choice of $\underline{\alpha}$, we have established that after a packet arrival (an absorption) the PH distribution starts anew in state 1 (a renewal) so that the arrival process remains in a burst. From $\mathbf{Q}_{\mathrm{IPP}}$, we have

$$\mathbf{T} = \begin{pmatrix} -\gamma_{01} & \gamma_{01} \\ \gamma_{10} & -(\gamma_{10}+\lambda) \end{pmatrix}, \tag{10.73}$$

so that we can easily derive moments, using (10.63):

$$\begin{aligned} E[X] &= (-1)^1 1!(0,1) \cdot \mathbf{T}^{-1} \cdot \underline{1} \\ &= \frac{-1}{\gamma_{0,1}\lambda}(0,1) \cdot \begin{pmatrix} -(\gamma_{1,0}+\lambda) & -\gamma_{0,1} \\ -\gamma_{1,0} & -\gamma_{0,1} \end{pmatrix} \cdot \underline{1} \\ &= \frac{1}{\lambda}\left(1 + \frac{\gamma_{1,0}}{\gamma_{0,1}}\right). \end{aligned} \tag{10.74}$$

Note that the inverse of a 2×2 matrix can be expressed directly:

$$\mathbf{A} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad \Longrightarrow \quad \mathbf{A}^{-1} = \frac{1}{\det(\mathbf{A})} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix},$$

where $\det(\mathbf{A}) = ad - cb$ is the determinant of matrix $\mathbf{A}$.

The first moment $E[X]$ of an IPP can also be obtained using an alternative calculation as follows. When one enters the initial state of an IPP, i.e., state 1 in Figure 10.12, the average time until absorption $E[X]$ consists of a number of phases. With probability 1, there is residence in state 1, with average length $1/(\lambda + \gamma_{1,0})$. Then, with probability $\gamma_{1,0}/(\lambda + \gamma_{1,0})$, a state change to state 0 takes place, adding to the average time to absorption $1/\gamma_{0,1}$, the average state residence time in state 0, plus, after returning to state 1 again, another $E[X]$. In other words, when entering state 1 for the second time, it is as if one enters there for the first time. In equational form, this can be written as

$$E[X] = \frac{1}{\lambda + \gamma_{1,0}} + \frac{\gamma_{1,0}}{\lambda + \gamma_{1,0}}\left(\frac{1}{\gamma_{0,1}} + E[X]\right). \tag{10.75}$$

Collecting terms, this can be written as

$$E[X] = \frac{1}{\lambda}\left(1 + \frac{\gamma_{1,0}}{\gamma_{0,1}}\right), \tag{10.76}$$

which is the result we have seen before.

Bibliography

[1] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. On Petri nets with stochastic timing. Internal Report 314, IENGF, Torino, Italy, April 1985.

[2] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Transactions on Software Engineering*, 15(7):832–846, July 1989.

[3] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. J. Wiley, 1995.

[4] M. Ajmone Marsan and G. Chiola. *On Petri Nets with Deterministic and Exponentially Distributed Firing Times*, volume 266 of *LNCS*, pages 132–145. Springer Verlag, 1987.

[5] D. Aldous and L. Shepp. The least variable phase type distribution is Erlang. *Stochastic Models*, 3:467–473, 1987.

[6] A. Bertoni and M. Torelli. Probabilistic Petri nets and semi-Markov processes. In *Proc. 2^{nd} European workshop on Petri nets*, Bad Honnef, West Germany, September 1981.

[7] A. Bobbio. Stochastic reward models in performance/reliability analysis. *Journal on Communications*, XLIII:27–35, January 1992.

[8] A. Bobbio and A. Cumani. Discrete state stochastic systems with phase type distributed transition times. In *Proc. AMSE Intern. Conf. on Modelling and Simulation*, Athens, 1984.

[9] A. Bobbio and M. Telek. A benchmark for PH estimation algorithms: results for Acyclic-PH. *Stochastic Models*, 10:661–677, 1994.

[10] A. Bobbio and M. Telek. *Computational restrictions for SPN with generally distributed transition times*, pages 131–148. 1994.

[11] A. Bobbio and M. Telek. Markov regenerative SPN with non-overlapping activity cycles. Technical report, Department of Telecommunications - Technical University of Budapest, Budapest, Hungary, October 1994.

[12] A. Bobbio and M. Telek. Transient analysis of a preemptive resume M/D/1/2/2 through Petri nets. Technical report, Department of Telecommunications - Technical University of Budapest, Budapest, Hungary, April 1994.

[13] P. Chen, S.C. Bruell, and G. Balbo. Alternative methods for incorporating non-exponential distributions into stochastic Petri nets. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, pages 187–197, Kyoto, Japan, December 1989. IEEE-CS Press.

[14] H. Choi, G. Kulkarni, and K.S. Trivedi. Transient analysis of deterministic and stochastic petri nets. In *Proc.* 14^{th} *Intern. Conference on Application and Theory of Petri Nets*, Chicago, Illinois, June 1993. Springer Verlag.

[15] H. Choi, V.S. Kulkarni, and K.S. Trivedi. Markov regenerative stochastic Petri nets. In *Proc. of Performance 93*, Rome, Italy, September 1993.

[16] G. Ciardo, R. German, and C. Lindemann. A characterization of the stochastic process underlying a stochastic Petri net. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 170–179, Toulouse, France, October 1993. IEEE-CS Press.

[17] G. Ciardo and C. Lindemann. Analysis of deterministic and stochastic Petri nets. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 160–169, Toulouse, France, October 1993. IEEE-CS Press.

[18] E. Cinlar. *Introduction to Stochastic Processes.* Prentice-Hall, Englewood Cliffs, NJ, 1975.

[19] D.R. Cox. *The Analysis of Non-Markovian Stochastic Processes by the Inclusion of Supplementary Variables*, volume 51, pages 433–440. 1955.

[20] A. Cumani. On the canonical representation of homogeneousMarkov processes modelling failure time distributions. *Microelectron Reliability*, 22:583–602, 1982.

[21] A. Cumani. Esp - a package for the evaluation of stochastic Petri nets with phase-type distributed transition times. In *Proc. Intern. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.

[22] J.B. Dugan, K.S. Trivedi, R.M. Geist, and V.F. Nicola. Extended stochastic Petri nets: Applications and analysis. In *Proc. PERFORMANCE '84*, Paris, France, December 1984.

[23] G. Florin and S. Natkin. Les reseaux de Petri stochastiques. *Technique et Science Informatiques*, 4(1), February 1985.

[24] R. German. Transient analysis of deterministic and stochastic Petri nets by the method of supplementary variables. In *Proc. of MASCOT'95*, 1995.

[25] R. German, C. Kelling, A. Zimmermann, and G. Hommel. TimeNET - a toolkit for evaluating non-Markovian stochastic Petri nets. Technical Report 19, Technische Universitat Berlina, Berlin, Germany, 1994.

[26] R. German and C. Lindemann. Analysis of stochastic Petri nets by the method of supplementary variables. In *Proc. of Performance 93*, Rome, Italy, September 1993.

[27] B.R. Haverkort and K. Trivedi. Specification techniques for Markov Reward Models. *Discrete Event Dynamic Systems: Theory and Applications*, 3:219–247, 1993.

[28] R.A. Howard. *Dynamic Probabilistic Systems*. John Wiley, New York, NY, 1971.

[29] D.L. Jagerman. An inversion technique for the Laplace transform. *The Bell System Technical Journal*, 61:1995–2002, October 1982.

[30] V.G. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, London, UK, 1995.

[31] C. Lindemann. An improved numerical algorithm for calculating steady-state solutions of deterministic and stochastic Petri net models. *Performance Evaluation*, 18:75–95, 1993.

[32] M.K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models*. PhD thesis, UCLA, Los Angeles, CA, 1981. Ph.D. Thesis.

[33] S. Natkin. *Les Reseaux de Petri Stochastiques et leur Application a l'Evaluation des Systemes Informatiques*. PhD thesis, CNAM, Paris, France, 1980. These de Docteur Ingegneur.

[34] M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models*. Johns Hopkins University Press, Baltimore, MD, 1981.

[35] A. Reibman, R. Smith, and K.S. Trivedi. Markov and Markov revard model transient analysis: an overview of numerical approaches. *European Journal of Operation Research*, 40:257–267, 1989.

[36] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton, New Jersey, USA, 1994.

Chapter 11

Infinite-state stochastic Petri nets

11.1 Introduction

The general approach in solving SPNs is to translate them to an underlying finite CTMC which can be solved numerically. However, a problem that often arises when following this approach is the rapid growth of the state space. Various solutions have been proposed to this problem, e.g., the use of state space truncation techniques [23] or lumping techniques [4, 5, 9, 46]. For a restricted class of SPNs, product-form results apply, so that an efficient mean-value analysis style of solution becomes feasible [27, 11]. In all these cases, the "trick" lies in circumventing the generation of the large overall state space.

When studying queueing models, one observes that the analysis of models with an unbounded state space is often simpler than analysing similar models on a finite state space. This suggests the idea of studying SPNs that have an unbounded state space. Instead of generating and solving a very large but finite SPN model we then have to solve infinitely large CTMCs derived from special SPN models. In this chapter we focus on a special class of stochastic Petri nets with unbounded state space, known as one-place unbounded SPNs or *infinite-state SPNs* (abbreviated as iSPNs). In particular, we focus on a class of SPNs of which the underlying CTMC has a so-called quasi-birth-death (QBD) structure, for which efficient solution methods exist.

The properties an SPN has to fulfill to belong to this class can be verified at the SPN level, without having to construct the reachability graph. The main advantage of iSPNs is that efficient matrix-geometric techniques for the solution are combined with the powerful description facilities of (general) SPNs. This not only allows non-specialists to use these methods, it also avoids the state-space explosion problem that is so common in traditional SPN analysis based on the complete (finite) underlying Markov chain.

This chapter is further organised as follows. In Section 11.2 we readdress

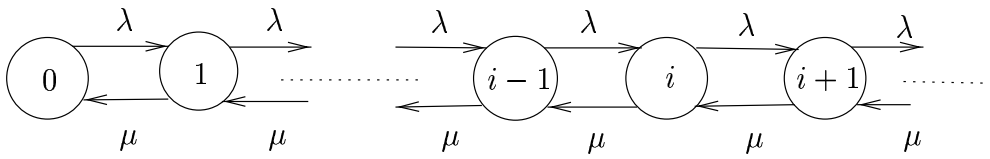


Figure 11.1: The state-transition diagram for the M/M/1 queue

the analysis of the M/M/1 queue in a "matrix-geometric way". This is used as a start-up for the matrix-geometric analysis of the PH/PH/1 queue in Section 11.3. We then formally characterise the class of iSPNs by defining a number of constraints that have to be fulfilled in Section 11.4, and elaborate on the QBD structure of the CTMCs underlying iSPNs. Numerical algorithms that play an important role in the matrix-geometric technique are discussed in Section 11.5. In Section 11.6 we then comment on algorithms to detect the special iSPN structure and to compute reward-based measures efficiently. We finally discuss a number of application examples in Section 11.7.

11.2 The M/M/1 queue

Consider an M/M/1 queueing model with arrival rate λ and service rate μ. The Markov chain underlying this queueing model is a simple birth-death process where the state variable denotes the total number of packets in the queueing station. Since the state variable is a scalar, such a process is sometimes called a *scalar state process*. The generator matrix $\mathbf{Q}$ of the CTMC underlying the M/M/1 queue has the following form:

$$\mathbf{Q} = \left(\begin{array}{c|ccccc} -\lambda & \lambda & 0 & \cdots & \cdots & \cdots \\ \mu & -(\lambda+\mu) & \lambda & 0 & \cdots & \cdots \\ 0 & \mu & -(\lambda+\mu) & \lambda & 0 & \cdots \\ \vdots & 0 & \mu & -(\lambda+\mu) & \lambda & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}\right). \tag{11.1}$$

Observe that, apart from the first column, all the columns are basically the same, except that "they are shifted down one row". We call the first column a *boundary column* and the other ones *repeating columns*. Notice that the repeating structure can also be observed nicely in the state-transition diagram of the CTMC underlying the M/M/1 queue, as given in Figure 11.1. The balance equations for states $i = 1, 2, \cdots$, have the following form:

$$\pi_i(\lambda + \mu) = \pi_{i-1}\lambda + \pi_{i+1}\mu. \tag{11.2}$$

Now, let us assume that the steady-state probability for state i only depends on the probability π_{i-1} and the rates between states i and $i-1$. As those

rates are constants, we *guess* that there is a constant ρ, of yet unknown value, which defines that $\pi_i = \rho\pi_{i-1}$, for $i = 1, 2, \cdots$, or equivalently, $\pi_i = \rho^i\pi_0$, for $i = 1, 2, \cdots$. Substituting this in the global balance equation for the repeating portion of the CTMC ($i = 1, 2, \cdots$):

$$\pi_0\rho^i(\lambda + \mu) = \pi_0\rho^{i-1}\lambda + \pi_0\rho^{i+1}\mu. \tag{11.3}$$

Since all steady-state probabilities depend via the multiplicative factor ρ on π_0, we have to assume that $\rho > 0$, otherwise all π_i would be equal to 0. In doing so, we may divide the above equation by ρ^{i-1}, so that we obtain

$$\pi_0\rho(\lambda + \mu) = \pi_0\lambda + \pi_0\rho^2\mu \;\Rightarrow\; \pi_0(\mu\rho^2 - (\lambda + \mu)\rho + \lambda) = 0. \tag{11.4}$$

The latter is true when either $\pi_0 = 0$ which does not make sense because in that case all the $\pi_i = 0$, or when the quadratic equation in ρ evaluates to 0. This is true for two values of ρ: $\rho = \lambda/\mu$ or $\rho = 1$. The latter solution cannot be accepted due to the fact that a proper normalisation of all the probabilities requires $\rho < 1$, otherwise $\sum_i \pi_i \neq 1$. We thus conclude that $\rho = \lambda/\mu$. Using this result, we can solve the "boundary" of the global balance equations as follows. We have

$$\pi_0\lambda = \pi_1\mu \;\Rightarrow\; \pi_0\lambda = \rho\mu\pi_0 = \pi_0\lambda. \tag{11.5}$$

This equality is always true and does not yield us the value of π_0. We therefore use the normalisation equation:

$$\sum_{i=0}^{\infty} \pi_i = 1 \Rightarrow \pi_0 \sum_{i=0}^{\infty} \rho^i = \frac{\pi_0}{1-\rho} = 1, \tag{11.6}$$

provided that $\rho < 1$. We therefore conclude that $\pi_0 = 1 - \rho$, and, consequently, $\pi_i = (1-\rho)\rho^i$, $i \in \mathbb{N}$. This result is of course well-known, however, the method by which we derived it here differs from standard methods [21, 28].

We need the fact that $\rho < 1$ in order for the geometric sum to be finite. The requirement $\rho < 1$ also exactly expresses a stability requirement for the queue: on average, the rate of incoming jobs must be smaller than the rate at which jobs can be served. In the state-transition diagram, this property can be interpreted as follows: the rate to the next higher state must be smaller than the rate to the next lower state.

Once the steady-state probabilities π_i ($i = 0, 1, \cdots$) are known, we can derive various performance metrics, such as:

- the average number of jobs in the queue: $E[N] = \sum_{i=0}^{\infty} i\pi_i = \frac{\rho}{1-\rho}$;
- the average response time (via Little's law): $E[R] = \frac{E[N]}{\lambda} = \frac{E[S]}{1-\rho}$, with the average service time $E[S] = 1/\mu$;
- the probability of at least k jobs in the queue: $B(k) = \sum_{i=k}^{\infty} \pi_i = \rho^k$.

In summary, to evaluate the M/M/1 queue, we have gone through 4 steps:

1. Guessing a solution based on the repetitive CTMC structure;
2. Substituting this solution in the repeating part of the global balance equations to derive the multiplicative constant ρ;
3. Computing the solution of the "boundary probabilities" by solving the corresponding part of the global balance equations, thereby using the normalisation equation and the result from Step 2;
4. Computing performance measures of interest.

In the next section we will use this 4-step approach to evaluate more complex queueing systems.

11.3 The PH/PH/1 queue

In Section 11.3.1 we present a structured description of the CTMC underlying a PH/PH/1 queue. In Section 11.3.2 we then proceed with the matrix-geometric solution. Stability issues are discussed in Section 11.3.3 and explicit expressions for performance measures of interest are discussed in Section 11.3.4.

11.3.1 A structured description of the CTMC

Now, consider the PH/PH/1 queue, the generalisation of the M/M/1 queue in which both the service and the interarrival times have a phase-type distribution. In this case, the state of the underlying CTMC is not totally described by the number of jobs in the queueing system. Part of the state description now is the phase of the arrival process, and the phase of the job in service, if any. Consequently, the state is a vector of three elements. The underlying Markov chain is therefore referred to as a *vector state process.*

Suppose that the representation of the arrival process is given by $(\boldsymbol{\alpha}, \mathbf{T})$, where $\mathbf{T}$ is a $m_a \times m_a$ matrix, and that the representation of the service process is given by $(\boldsymbol{\beta}, \mathbf{S})$, where $\mathbf{S}$ is a $m_s \times m_s$ matrix. Ordered lexicographically, the states of the Markov chain underlying this PH/PH/1 queueing system are of the form (n, a, s) where n is the number of jobs in the queue, $a \in \{1, \cdots, m_a\}$ is the phase of the arrival process, and $s \in \{1, \cdots, m_s\}$ is the phase of the service process:

$$\mathcal{I} = \{(0,1,1), (0,2,1), \cdots, (0,m_a,1), (1,1,1), \cdots, (1,1,m_s), (1,2,1), \cdots, \\ (1,2,m_s), \cdots, (1,m_a,1), \cdots, (1,m_a,m_s), \cdots, (2,m_a,m_s), \cdots\}.$$

All the states that have the same number of jobs in the queue are said to belong to one *level.* For example, level i, $i = 1, 2, \cdots$, corresponds to the set of states

$$\{(i,1,1), \cdots, (i,1,m_s), (i,2,1), \cdots, (i,2,m_s), \cdots, (i,m_a,1), \cdots, (i,m_a,m_s)\}.$$

Level 0 consists of the states $\{(0,1,1), (0,2,1), \cdots, (0,m_a,1)\}$. Apart from some irregularities at the boundary, the matrix $\mathbf{Q}$ shows great similarity with the generator matrix we have seen for the M/M/1 queue. In particular, we observe that

$\mathbf{Q}$ is block-tridiagonal. The upper-diagonal blocks describe the transitions from a certain level to the next higher level: they correspond to arrivals. The lower-diagonal blocks describe the transitions from a certain level to the next lower level: they correspond to service completions. The diagonal blocks describe transitions internal to a level causing arrivals nor departures, that is, they correspond to phase changes in the arrival or service process. Viewing the levels as "super states" the matrix $\mathbf{Q}$ in fact describes a birth-death process on them. For this reason vector state processes are most often called *quasi-birth-death models* (QBDs).

Returning to the specific case at hand, the generator matrix $\mathbf{Q}$ of the underlying Markov chain has the following structure:

$$\mathbf{Q} = \left(\begin{array}{cc|ccc} \mathbf{B}_{00} & \mathbf{B}_{01} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{B}_{10} & \mathbf{A}_1 & \mathbf{A}_0 & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_0 & \cdots \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_2 & \mathbf{A}_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{array}\right). \tag{11.7}$$

The square matrices $\mathbf{A}_i$ are of size $m_a m_s \times m_a m_s$ and have the following form:

$$\begin{aligned} \mathbf{A}_0 &= \boldsymbol{T}^0 \boldsymbol{\alpha} \otimes \mathbf{I}, \\ \mathbf{A}_1 &= (\mathbf{T} \otimes \mathbf{I}) + (\mathbf{I} \otimes \mathbf{S}), \\ \mathbf{A}_2 &= \mathbf{I} \otimes \boldsymbol{S}^0 \boldsymbol{\beta}, \end{aligned} \tag{11.8}$$

where the binary operator $\otimes$ is used to represents the tensor or Kronecker product of two matrices (note that $\boldsymbol{T}^0$ and $\boldsymbol{S}^0$ are column vectors). The matrix $\mathbf{A}_0$ describes the transitions to the next higher level and includes the component $\boldsymbol{T}^0$ which indicates the rate at which the arrival process completes. The factor $\boldsymbol{\alpha}$ accounts for the possible change in phase in the next arrival interval directly starting afterwards. Similarly, $\mathbf{A}_2$ describes the rate at which services complete (factor $\boldsymbol{S}^0$) multiplied with the vector $\boldsymbol{\beta}$ in order to account for the starting phase of the next service epoch. The matrix $\mathbf{A}_1$ describes changes in arrival or service process phase within a single level. The matrices $\mathbf{B}_{ij}$ are all differently dimensioned and have the following form:

$$\begin{aligned} \mathbf{B}_{00} &= \mathbf{T} \text{ (size } m_a \times m_a), \\ \mathbf{B}_{01} &= \boldsymbol{T}^0 \boldsymbol{\alpha} \otimes \boldsymbol{\beta} \text{ (size } m_a \times m_a m_s), \\ \mathbf{B}_{10} &= \mathbf{I} \otimes \boldsymbol{S}^0 \text{ (size } m_a m_s \times m_a). \end{aligned} \tag{11.9}$$

Matrix $\mathbf{B}_{00}$ describes state changes internal to level 0; because there is no job in the queue at level 0, only changes in the arrival process need to be kept track of, which explains its equality to $\mathbf{T}$. Matrix $\mathbf{B}_{01}$ has a similar structure as $\mathbf{A}_0$, however, as it represents the arrival of the first job after the system has been empty for some time, it needs an extra factor taking into account the phase at which the next service is started, which explains the factor $\boldsymbol{\beta}$. Similarly, $\mathbf{B}_{10}$

much resembles $\mathbf{A}_2$, however, as this matrix represents transitions to the empty system, no new service epochs can start, so that a factor $\boldsymbol{\beta}$ is missing.

Example 11.1 The $\mathbf{E}_2/\mathbf{H}_2/1$ queue (I) Consider a queueing model with a single server and with Erlang-2 interarrival times and hyper-exponentially distributed service times. Notice that this model is a special case of the G/G/1 queue. We use the following PH-type representation of a 2-stage Erlang distribution:

$$\mathbf{T} = \begin{pmatrix} -2\lambda & 2\lambda \\ 0 & -2\lambda \end{pmatrix}, \quad \boldsymbol{T}^0 = \begin{pmatrix} 0 \\ 2\lambda \end{pmatrix} \text{ and } \boldsymbol{\alpha} = (1,0), \tag{11.10}$$

that is, we have two exponential phases which are visited one after another. If both phases have been passed, an arrival takes place and the next interarrival period begins (a renewal process). We take the following representation of a 2-stage hyper-exponential service time distribution:

$$\mathbf{S} = \begin{pmatrix} -\mu_1 & 0 \\ 0 & -\mu_2 \end{pmatrix}, \quad \boldsymbol{S}^0 = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \text{ and } \boldsymbol{\beta} = \left(\frac{1}{4}, \frac{3}{4}\right), \tag{11.11}$$

that is, with probability $\frac{1}{4}$ phase 1 is taken, with mean length $\frac{1}{\mu_1}$ and with probability $\frac{3}{4}$ phase 2 is taken, with mean length $\frac{1}{\mu_2}$.

The CTMC describing this queueing model is a QBD process where the number of states per level is 4 (2×2; two phase possibilities for the service as well as for the arrival process). In case the system is empty, only the arrival process can be in one of its two states, hence, the first level (level 0) only has two states. The state transition diagram is depicted in Figure 11.2. As can be observed, transitions related to services always point to states one level lower. Notice that at the end of a service period, the next service period is started; the probabilities $\boldsymbol{\beta}$ are used for that purpose by multiplying them with the service rates μ_i. The first phase in the arrival process always leads to a state transition within a level; the completion of the second phase in the arrival process leads to a next higher level. Notice that only at the arrival of the first customer (transition from level 0 to level 1) the service time distribution is chosen (with probability β_i take rate μ_i). For arrivals at a non-empty queue the choice of the service time distribution is postponed until the next customer has completed its service.

From Figure 11.2 the generator matrix for this queueing system can readily be obtained. Instead, one could also apply the above definitions of the block matrices to arrive at:

$$\mathbf{B}_{00} = \mathbf{T} = \begin{pmatrix} -2\lambda & 2\lambda \\ 0 & -2\lambda \end{pmatrix}, \quad \mathbf{B}_{01} = \boldsymbol{T}^0\boldsymbol{\alpha} \otimes \boldsymbol{\beta} = \tfrac{1}{4}\begin{pmatrix} 0 & 0 & 0 & 0 \\ 2\lambda & 6\lambda & 0 & 0 \end{pmatrix},$$

$$\mathbf{B}_{10} = \mathbf{I} \otimes \boldsymbol{S}^0 = \begin{pmatrix} \mu_1 & 0 \\ \mu_2 & 0 \\ 0 & \mu_1 \\ 0 & \mu_2 \end{pmatrix}, \quad \mathbf{A}_0 = \boldsymbol{T}^0\boldsymbol{\alpha} \otimes \mathbf{I} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2\lambda & 0 & 0 & 0 \\ 0 & 2\lambda & 0 & 0 \end{pmatrix},$$

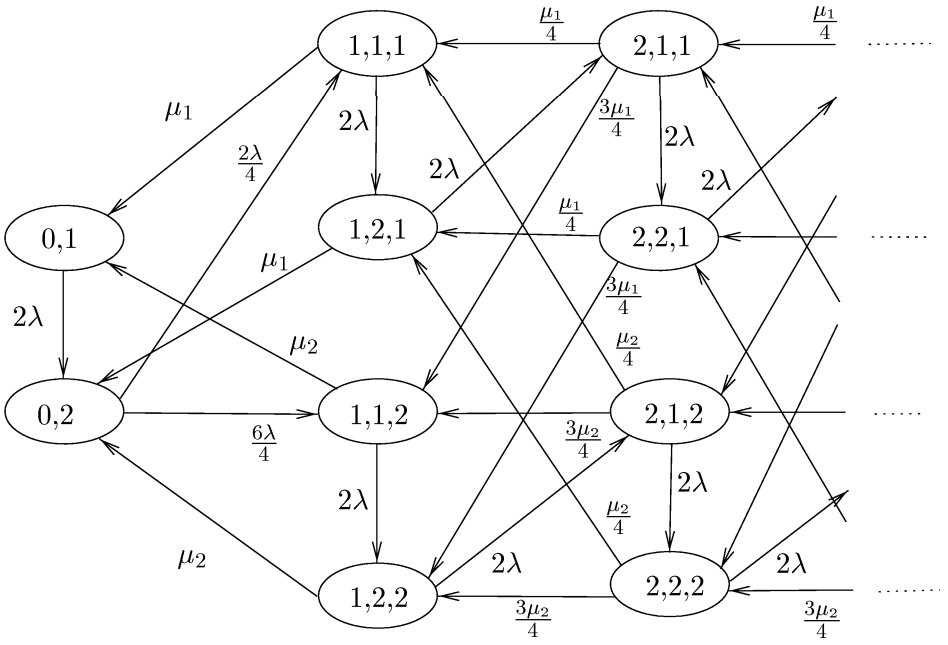


Figure 11.2: State transition diagram for the QBD underlying the $E_2/H_2/1$ queue

$$\mathbf{A}_1 = \begin{pmatrix} -(\mu_1+2\lambda) & 0 & 2\lambda & 0 \\ 0 & -(\mu_2+2\lambda) & 0 & 2\lambda \\ 0 & 0 & -(\mu_1+2\lambda) & 0 \\ 0 & 0 & 0 & -(\mu_2+2\lambda) \end{pmatrix},$$

and

$$\mathbf{A}_2 = \mathbf{I} \otimes \boldsymbol{S}^0\boldsymbol{\beta} = \tfrac{1}{4}\begin{pmatrix} \mu_1 & 3\mu_1 & 0 & 0 \\ \mu_2 & 3\mu_2 & 0 & 0 \\ 0 & 0 & \mu_1 & 3\mu_1 \\ 0 & 0 & \mu_2 & 3\mu_2 \end{pmatrix}. \tag{11.12}$$

11.3.2 Matrix-geometric solution

We now proceed to solve the CTMC for its steady-state probabilities in a way that closely resembles the approach followed for the M/M/1 queue. First, let $\boldsymbol{z}_i$ denote the vector of lexicographically ordered steady-state probabilities of states in level i. Then, for $i = 1, 2, \cdots$, we have

$$\boldsymbol{z}_i = (\pi_{(i,1,1)}, \cdots, \pi_{(i,1,m_s)}, \pi_{(i,2,1)}, \cdots, \pi_{(i,2,m_s)}, \cdots, \pi_{(i,m_a,1)}, \cdots, \pi_{(i,m_a,m_s)}). \tag{11.13}$$

The arguments that led to the simple geometric form for the steady-state probabilities in an M/M/1 queue apply again here, however, we now have to take

levels of states as neighbours between which probability flows. That is, the probability of residence in level i ($\boldsymbol{z}_i$) depends only on the probability of residence in level $i-1$ ($\boldsymbol{z}_{i-1}$), for those levels i which are already in the repeating portion of the CTMC, i.e., for $i = 2, 3, \cdots$. Since the rates between neighbouring levels of states are constants, we hope that the relation we are aiming at can be expressed as a multiplicative constant between $\boldsymbol{z}_i$ and $\boldsymbol{z}_{i-1}$. Note, however, that since we are dealing with vectors now, this constant needs to be a square matrix $\mathbf{R}$ of size $m_a m_s \times m_a m_s$. If we *assume* that

$$\boldsymbol{z}_i = \boldsymbol{z}_{i-1}\mathbf{R}, \tag{11.14}$$

it follows that the steady-state probability vectors $\boldsymbol{z}_i$ must have the form

$$\boldsymbol{z}_i = \boldsymbol{z}_1\mathbf{R}^{i-1}, \ , \ i = 1, 2, \cdots. \tag{11.15}$$

Notice that we "go back in the recursion" to level 1 here, and not to level 0 as in the M/M/1 case. This difference is due to the fact that in the M/M/1 case we had only one boundary level, whereas we have two here. The length of the vectors $\boldsymbol{z}_i$, $i = 1, 2, \cdots$, is $m_a m_s$. As in the M/M/1 case for ρ, the square matrix $\mathbf{R}$ of size $m_a m_s \times m_a m_s$ follows from a quadratic equation obtained by substituting the assumed geometric form in the repeating portion of the global balance equation:

$$[\cdots, \boldsymbol{z}_i, \boldsymbol{z}_{i+1}, \boldsymbol{z}_{i+2}, \cdots]\mathbf{Q} = \mathbf{0} \Rightarrow \ \boldsymbol{z}_i\mathbf{A}_0 + \boldsymbol{z}_{i+1}\mathbf{A}_1 + \boldsymbol{z}_{i+2}\mathbf{A}_2 = \mathbf{0}, \tag{11.16}$$

which, after rewriting the vectors $\boldsymbol{z}_i$ in terms of $\boldsymbol{z}_1$ yields

$$\boldsymbol{z}_1(\mathbf{R}^2\mathbf{A}_2 + \mathbf{R}^1\mathbf{A}_1 + \mathbf{R}^0\mathbf{A}_0) = \mathbf{0}. \tag{11.17}$$

This equation can only be true when either $\boldsymbol{z_1} = \mathbf{0}$, or when the quadratic equation within parentheses equals $\mathbf{0}$. For the same reasons as mentioned when discussing the M/M/1 queue, the latter must be the case, and the matrix $\mathbf{R}$ thus follows from the following matrix quadratic equation:

$$\mathbf{R}^2\mathbf{A}_2 + \mathbf{R}^1\mathbf{A}_1 + \mathbf{R}^0\mathbf{A}_0 = \mathbf{0}. \tag{11.18}$$

We discuss the actual computation of $\mathbf{R}$ from this quadratic equation in Section 11.5; for the time being we assume that we can do so.

To start the recursive relation (11.15) the following boundary equations must be solved to obtain $\boldsymbol{z}_0$ and $\boldsymbol{z}_1$:

$$(\boldsymbol{z}_0, \boldsymbol{z}_1) \cdot \begin{pmatrix} \mathbf{B}_{00} \\ \mathbf{B}_{10} \end{pmatrix} = \mathbf{0}, \ \text{ and } \ (\boldsymbol{z}_0, \boldsymbol{z}_1, \boldsymbol{z}_2) \cdot \begin{pmatrix} \mathbf{B}_{01} \\ \mathbf{A}_1 \\ \mathbf{A}_2 \end{pmatrix} = \mathbf{0}. \tag{11.19}$$

By the fact that $\boldsymbol{z}_2\mathbf{A}_2 = (\boldsymbol{z}_1\mathbf{R})\mathbf{A}_2$ we can rewrite the right equation as

$$(\boldsymbol{z}_0, \boldsymbol{z}_1) \cdot \begin{pmatrix} \mathbf{B}_{01} \\ \mathbf{A}_1 + \mathbf{R}\mathbf{A}_2 \end{pmatrix} = \mathbf{0}, \tag{11.20}$$

so that we obtain the following boundary equations:

$$(\boldsymbol{z}_0, \boldsymbol{z}_1) \cdot \begin{pmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \mathbf{B}_{10} & \mathbf{A}_1 + \mathbf{R}\mathbf{A}_2 \end{pmatrix} = (\mathbf{0}, \mathbf{0}). \tag{11.21}$$

As can be observed, the length of $\boldsymbol{z}_0 = (\pi_{(0,1,0)}, \cdots, \pi_{(0,m_a,0)})$ is m_a. Also, since (11.21) is not of full rank, the normalisation equation has to be used to arrive at a unique solution:

$$\sum_{i=0}^{\infty} \boldsymbol{z}_i \mathbf{1}^T = \boldsymbol{z}_0 \mathbf{1}^T + \sum_{i=1}^{\infty} \boldsymbol{z}_i \mathbf{1}^T = \boldsymbol{z}_0 \mathbf{1}^T + \boldsymbol{z}_1 (\mathbf{I} - \mathbf{R})^{-1} \mathbf{1}^T = 1. \tag{11.22}$$

This equation might be integrated in (11.21) to yield one system of linear equations.

11.3.3 Stability issues

Let us now return to the question under which conditions (11.18) can indeed be solved and leads to a unique matrix $\mathbf{R}$.

First recall from the M/M/1 queue that such a queue is stable as long as $\lambda < \mu$. Translated to the state-transition diagram, this inequality can be interpreted as follows. As long as the "drift" to higher level states is smaller than the "drift" to lower level states, the system is stable: $\lambda < \mu \Rightarrow \rho = \lambda/\mu < 1$ and a steady-state solution exists.

A similar reasoning can be followed in case we deal with QBDs. Let us denote $\mathbf{A} = \mathbf{A}_0 + \mathbf{A}_1 + \mathbf{A}_2$; this matrix can be interpreted as a Markov generator matrix which described the transition behaviour within a level. Transitions between levels, normally described by matrices $\mathbf{A}_0$ and $\mathbf{A}_2$, are simply "looped back" to their originating level. Now, define the vector $\boldsymbol{\nu}$ to be the steady-state probability vector belonging to the CTMC with generator matrix $\mathbf{A}$. $\boldsymbol{\nu}$ follows from $\boldsymbol{\nu}\mathbf{A} = \mathbf{0}$ under the normalising condition $\sum_i \nu_i = 1$. Thus, given presence in some repeating level j, the probability of being in the i-th state of that level, i.e., in state (j, i), equals ν_i. Given this probability, the total "drift" to a next higher level can be expressed as $\sum_i \nu_i \sum_l a^{(0)}_{(i,l)}$, where $a^{(0)}_{(i,l)}$ is the (i,l)-th element of matrix $\mathbf{A}_0$. Similarly, the drift to the next lower level can be expressed as $\sum_i \nu_i \sum_l a^{(2)}_{(i,l)}$. Notice that in both these expressions, the rates to the next higher (lower) level are weighted by their relative occurrence. Now, as long as the drift to next higher levels, i.e., $\sum_i \nu_i \sum_l a^{(0)}_{(i,l)}$, is smaller than the drift to a next lower level, i.e., $\sum_i \nu_i \sum_l a^{(2)}_{(i,l)}$, we have a stable system. In matrix notation, the required inequality for stability then becomes:

$$\boldsymbol{\nu}\mathbf{A}_0\mathbf{1}^T < \boldsymbol{\nu}\mathbf{A}_2\mathbf{1}^T. \tag{11.23}$$

Once the matrices $\mathbf{A}_i$ have been derived, this inequality can easily be verified.

11.3.4 Performance measures

Once the matrix $\mathbf{R}$ and the boundary vectors $\boldsymbol{z}_0$ and $\boldsymbol{z}_1$ have been obtained, various interesting performance metrics can be derived:

- Arrival and service process characteristics:
  - the average interarrival time $E[A] = -\boldsymbol{\alpha}\mathbf{T}^{-1}\mathbf{1}^T$; we define $\lambda = 1/E[A]$;
  - the average service time $E[S] = -\boldsymbol{\beta}\mathbf{S}^{-1}\mathbf{1}^T$; we define $\mu = 1/E[S]$.
- Average performance measures:
  - the utilisation $E[N_s] = \rho = \lambda/\mu$;
  - the average number of jobs in the queue and server:

$$\begin{aligned} E[N] &= \sum_{i=1}^{\infty} i\boldsymbol{z}_i\mathbf{1}^T = \boldsymbol{z}_1\left(\sum_{i=1}^{\infty} i\mathbf{R}^{i-1}\right)\mathbf{1}^T = \boldsymbol{z}_1\left(\sum_{i=1}^{\infty}\frac{d}{d\mathbf{R}}\mathbf{R}^i\right)\mathbf{1}^T \\ &= \boldsymbol{z}_1\frac{d}{d\mathbf{R}}\left(\sum_{i=1}^{\infty}\mathbf{R}^i\right)\mathbf{1}^T = \boldsymbol{z}_1\frac{d}{d\mathbf{R}}\left((\mathbf{I}-\mathbf{R})^{-1}-\mathbf{I}\right)\mathbf{1}^T \\ &= \boldsymbol{z}_1(\mathbf{I}-\mathbf{R})^{-2}\mathbf{1}^T; \end{aligned} \tag{11.24}$$

  - the average number of jobs in the queue $E[N_q] = E[N] - \rho$;
  - the average response time $E[R] = E[N]/\lambda$;
  - the average waiting time $E[W] = E[N_q]/\lambda$.
- Detailed performance measures:
  - the probability z_i of having i jobs in the queue: $z_i = \boldsymbol{z}_i\mathbf{1}^T$;
  - the probability B_k of having at least k jobs in the queue, which for $k \geq 1$ can be written as:

$$\begin{aligned} B(k) &= \sum_{i=k}^{\infty}\boldsymbol{z}_i\cdot\mathbf{1}^T = \boldsymbol{z}_1\left(\sum_{i=k}^{\infty}\mathbf{R}^{i-1}\right)\mathbf{1}^T \\ &= \boldsymbol{z}_1\mathbf{R}^{k-1}(\mathbf{I}-\mathbf{R})^{-1}\mathbf{1}^T. \end{aligned} \tag{11.25}$$

For all these measures, the similarity with the corresponding measures for the M/M/1 queue is striking.

11.4 Infinite-state SPNs

We discuss some preliminary notation and terminology in Section 11.4.1. The requirements for iSPNs are formally stated in Section 11.4.2 and discussed in Section 11.4.3. Finally, in Section 11.4.4 we present the matrix-geometric solution.

11.4.1 Preliminaries

The class of iSPNs is similar to the class of SPNs defined in Chapter 9. Without loss of generality we assume that the iSPN under study, denoted *iSPN*, has a set $P = \{P_0, P_1, \cdots, P_{n_p}\}$ of places of which P_0 may contain an infinitely large number of tokens. A distribution of tokens over the places is called a marking and denoted $\mathbf{m} = (m_0, \boldsymbol{m}) = (m_0, m_1, \cdots, m_n)$. With $m_0 \in \mathbb{N}$ and $\boldsymbol{m} \in \mathcal{R}'$, the set of all possible markings is denoted $\mathrm{RS} = \mathbb{N} \times \mathcal{R}'$. Clearly, $|\mathbb{N}| = \infty$ and $|\mathcal{R}'| < \infty$. The set of transitions is denoted T.

We now define *level* $\mathcal{R}(k)$ to be the set of markings such that place P_0 contains k tokens, i.e., $\mathcal{R}(k) = \{\mathbf{m} = (m_0, \boldsymbol{m}) \in \mathrm{RS} | m_0 = k\}$. The levels $\mathcal{R}(k)$, $k \in \mathbb{N}$ constitute a partition of the overall state space: $\mathcal{R} = \bigcup_{k=0}^{\infty} \mathcal{R}(k)$ and $\mathcal{R}(k) \cap \mathcal{R}(l) = \emptyset, k \neq l$. For ease in notation, we also introduce $\mathcal{R}'(k) = \{\boldsymbol{m} | (k, \boldsymbol{m}) \in \mathcal{R}(k)\}$.

We furthermore define the following two *leads to* relations. We denote $\mathbf{m} \xrightarrow{t} \mathbf{m}'$ if transition t is enabled in $\mathbf{m}$ and, upon firing, leads to marking $\mathbf{m}'$. The firing rate of t is not important. We denote $\mathbf{m} \xrightarrow{t,\lambda} \mathbf{m}'$ if transition $t \in T$ is enabled in $\mathbf{m}$ and, upon firing, with rate λ, leads to marking $\mathbf{m}'$.

11.4.2 Formal requirements

We now can define the class of iSPNs by imposing a number of requirements on the SPN structure and transition firing behaviour. It should be noted that these requirements are sufficient, rather than necessary.

Requirement 1. Given *iSPN*, there exists a $\kappa \in \mathbb{N}$ such that for all $k, l \geq \kappa$: $\mathcal{R}'(k) = \mathcal{R}'(l)$. We denote $L = |\mathcal{R}'(\kappa)|$.

Requirement 2. Given *iSPN* and κ as defined above, the following requirements should hold for the so-called repeating portion of the state space:

1. intra-level equivalence:
   $\forall k, l \geq \kappa$, $t \in T$, $\lambda \in \mathbb{R}^+$: if $(k, \boldsymbol{m}) \xrightarrow{t,\lambda} (k, \boldsymbol{m}')$ then $(l, \boldsymbol{m}) \xrightarrow{t,\lambda} (l, \boldsymbol{m}')$;

2. inter-level one-step increases only:
   $\forall k \geq \kappa$, $\exists t \in T$: $(k, \boldsymbol{m}) \xrightarrow{t} (k+1, \boldsymbol{m}')$;
   $\forall k \geq \kappa$, $t \in T$, $\lambda \in \mathbb{R}^+$: if $(k, \boldsymbol{m}) \xrightarrow{t,\lambda} (k+1, \boldsymbol{m}')$ then $(k+1, \boldsymbol{m}) \xrightarrow{t,\lambda} (k+2, \boldsymbol{m}')$;
   $\forall k \geq \kappa$, $\forall i \in \mathbb{N}, i \geq 2$, $\nexists t \in T$: $(k, \boldsymbol{m}) \xrightarrow{t} (k+i, \boldsymbol{m}')$;

3. inter-level one-step decreases only:
   $\forall k > \kappa$, $\exists t \in T$: $(k+1, \boldsymbol{m}) \xrightarrow{t} (k, \boldsymbol{m}')$;
   $\forall k > \kappa$, $t \in T$, $\lambda \in \mathbb{R}^+$: if $(k+2, \boldsymbol{m}) \xrightarrow{t,\lambda} (k+1, \boldsymbol{m}')$ then $(k+1, \boldsymbol{m}) \xrightarrow{t,\lambda} (k, \boldsymbol{m}')$;
   $\forall k > \kappa$, $\forall i \in \mathbb{N}, i \geq 2$, $\nexists t \in T$: $(k+i, \boldsymbol{m}) \xrightarrow{t} (k, \boldsymbol{m}')$;

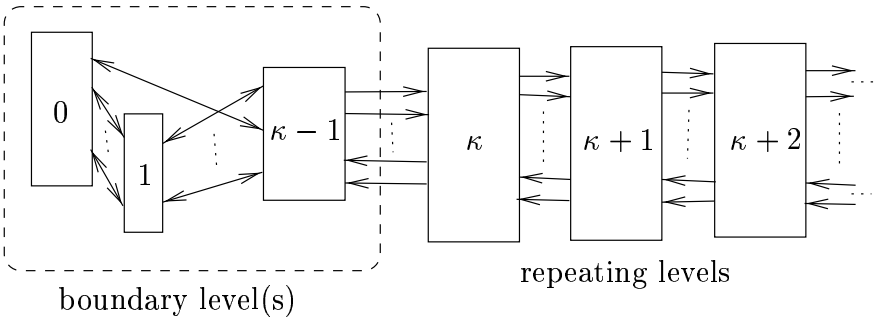


Figure 11.3: State space partitioning in levels

Requirement 3. Given *iSPN* and κ as defined above, for the so-called boundary portion of the state space, the following requirements should hold:

1. no boundary jumping:
   $\forall k < \kappa - 1, \forall l > \kappa,\ \nexists t_1 \in T\colon\ (k, \boldsymbol{m}) \xrightarrow{t_1} (l, \boldsymbol{m}')$;
   $\forall k < \kappa - 1, \forall l > \kappa,\ \nexists t_2 \in T\colon\ (l, \boldsymbol{m}) \xrightarrow{t_2} (k, \boldsymbol{m}')$;

2. only boundary crossing:
   $\exists t_1, t_2 \in T\colon\ (\kappa - 1, \boldsymbol{m}_1) \xrightarrow{t_1} (\kappa, \boldsymbol{m}_1'),\ (\kappa, \boldsymbol{m}_2) \xrightarrow{t_2} (\kappa - 1, \boldsymbol{m}_2')$;

11.4.3 Discussion of the requirements

To ease the understanding, let us now discuss the formal requirements in a more informal way. We first address the state space and its partitioning in levels. The first requirements states that, starting from a certain level κ upwards, all levels are the same as far as the non-infinite places (places P_1 through P_n) are concerned; they only differ in the number of tokens in place P_0. It is for this reason that the levels $k \geq \kappa$ are called the repeating portion (levels) of the state space. The levels $k < \kappa$ are denoted the boundary portion (levels) of the state space. In Figure 11.3 we depict the overall state space and its partitioning in levels. We have tried to visualise the fact that starting from level κ upwards, the levels repeat one another. Levels 0 through $\kappa - 1$ can be totally different from one another. Between states from levels 0 through $\kappa - 1$ all kinds of transitions may occur. That is why we can also see these boundary levels as one aggregated boundary level (Requirement 1).

Transitions can occur within a level, and between levels. Since the repeating levels are always the same (apart from the level number itself) all internal transitions in one level must have similar equivalents in other repeating levels. There are no transitions possible between non-neighbouring levels. There have to exist up- and down-going transitions between neighbouring levels. Also, for the repeating levels, their interaction with neighbouring levels is always the same (Requirement 2).

The transitions between the boundary levels and the repeating levels only take place in levels $\kappa-1$ and κ, however, they may have any form (Requirement 3).

As a conclusion, due to the three requirements the CTMC underlying iSPNs obeys a quasi-birth-death structure. We will exploit this fact in the solution of this class of SPNs.

A final remarks should be made regarding the necessity of the requirements. Indeed, the requirements are sufficient, however, not always necessary. One can imagine CTMCs which have a slightly different structure, especially in the boundary part of the state space, that still have a matrix-geometric solution. Stating necessary requirements, however, would make the requirements more cumbersome to validate.

11.4.4 Matrix-geometric solution

Referring to Figure 11.3, it is easy to see that the generator matrix $\mathbf{Q}$ of the QBD has the following form:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B}_{0,0} & \cdots & \mathbf{B}_{0,\kappa-1} & 0 & 0 & \cdots & \cdots \\ \mathbf{B}_{1,0} & \cdots & \mathbf{B}_{1,\kappa-1} & 0 & 0 & \cdots & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{B}_{\kappa-1,0} & \cdots & \mathbf{B}_{\kappa-1,\kappa-1} & \mathbf{B}_{\kappa-1,\kappa} & 0 & \cdots & \cdots \\ 0 & \cdots & \mathbf{B}_{\kappa,\kappa-1} & \mathbf{B}_{\kappa,\kappa} & \mathbf{B}_{\kappa,\kappa+1} & 0 & \cdots \\ 0 & \cdots & 0 & \mathbf{B}_{\kappa+1,\kappa} & \mathbf{B}_{\kappa+1,\kappa+1} & \mathbf{B}_{\kappa+1,\kappa+2} & \cdots \\ 0 & \cdots & 0 & 0 & \mathbf{B}_{\kappa+2,\kappa+1} & \mathbf{B}_{\kappa+2,\kappa+2} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{11.26}$$

Now, by the requirements posed on the intra- and inter-level transitions, we have

$$\begin{cases} \mathbf{B}_{k,k+1} = \mathbf{A}_0, & k = \kappa, \kappa+1 \cdots, \\ \mathbf{B}_{k,k} = \mathbf{A}_1, & k = \kappa+1, \kappa+2 \cdots, \\ \mathbf{B}_{k,k-1} = \mathbf{A}_2, & k = \kappa+1, \kappa+2 \cdots. \end{cases} \tag{11.27}$$

Using this notation, we may write $\mathbf{Q}$ as follows:

$$\mathbf{Q} = \begin{pmatrix} \mathbf{B}_{0,0} & \cdots & \mathbf{B}_{0,\kappa-1} & 0 & \cdots & \cdots & \cdots \\ \vdots & \cdots & \vdots & 0 & \cdots & \cdots & \cdots \\ \mathbf{B}_{\kappa-1,0} & \cdots & \mathbf{B}_{\kappa-1,\kappa-1} & \mathbf{B}_{\kappa-1,\kappa} & 0 & \cdots & \cdots \\ 0 & 0 & \mathbf{B}_{\kappa,\kappa-1} & \mathbf{B}_{\kappa,\kappa} & \mathbf{A}_0 & 0 & \cdots \\ 0 & 0 & 0 & \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_0 & \cdots \\ 0 & 0 & 0 & 0 & \mathbf{A}_2 & \mathbf{A}_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}. \tag{11.28}$$

From a CTMC with the above generator matrix, we can compute the steady-state probabilities $\boldsymbol{\pi}$ by solving the usual global balance equations (GBEs):

$$\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}, \text{ and } \sum_i \pi_i = 1, \tag{11.29}$$

where the right part is a normalisation to make sure that all the probabilities sum to 1. First, we partition $\boldsymbol{\pi}$ according to the levels, i.e., $\boldsymbol{\pi} = (\boldsymbol{z}_0, \boldsymbol{z}_1, \cdots, \boldsymbol{z}_{\kappa-1}, \boldsymbol{z}_\kappa, \boldsymbol{z}_{\kappa+1}, \cdots)$. Substituting this in (11.29) we obtain the following system of linear equations:

$$\begin{cases} i = 0, \cdots, \kappa-2: & \sum_{j=0}^{\kappa-1} \boldsymbol{z}_j \mathbf{B}_{j,i} = \mathbf{0}, & \text{(a)} \\ i = \kappa-1: & \sum_{j=0}^{\kappa} \boldsymbol{z}_j \mathbf{B}_{j,i} = \mathbf{0}, & \text{(b)} \\ i = \kappa: & \sum_{j=\kappa-1}^{\kappa+1} \boldsymbol{z}_j \mathbf{B}_{j,i} = \mathbf{0}, & \text{(c)} \\ i = \kappa+1, \cdots: & \sum_{j=0}^{2} \boldsymbol{z}_{i+j-1} \mathbf{A}_j = \mathbf{0}, & \text{(d)} \\ \text{normalisation}: & \sum_{i=0}^{\infty} \boldsymbol{z}_i \cdot \mathbf{1}^T = 1. & \text{(e)} \end{cases} \tag{11.30}$$

We now exploit the regular structure of the state space in the solution process in a similar way as we have done before. In particular, looking at (11.30(d)), it seems reasonable to *assume* that for the state probabilities $\boldsymbol{z}_i$,$i = \kappa, \kappa+1, \cdots$, only the neighbouring levels are of importance, so that they can be expressed as:

$$\boldsymbol{z}_{\kappa+1} = \boldsymbol{z}_\kappa \mathbf{R},\ \boldsymbol{z}_{\kappa+2} = \boldsymbol{z}_{\kappa+1}\mathbf{R} = \boldsymbol{z}_\kappa \mathbf{R}^2, \cdots, \tag{11.31}$$

or, equivalently,

$$\boldsymbol{z}_{\kappa+i} = \boldsymbol{z}_\kappa \mathbf{R}^i, i \in \mathbb{N}, \tag{11.32}$$

where $\mathbf{R}$ is a square $L \times L$ matrix relating the steady-state probability vector at level $\kappa+i$ to the steady-state probability vector at level $\kappa+i-1$ ($i = 1, 2, \cdots$). We know that this is true when the matrix $\mathbf{R}$ satisfies the matrix polynomial:

$$\mathbf{A}_0 + \mathbf{R}\mathbf{A}_1 + \mathbf{R}^2\mathbf{A}_2 = \mathbf{0}. \tag{11.33}$$

We will discuss means to solve this matrix polynomial in Section 11.5.

In case $i = \kappa$, (11.30(c)) can be rewritten to incorporate the above assumption, because $\boldsymbol{z}_{\kappa+1}$ can be written in terms of $\boldsymbol{z}_\kappa$ and $\mathbf{B}_{\kappa+1,\kappa} = \mathbf{A}_2$:

$$\begin{aligned} \sum_{j=\kappa-1}^{\kappa+1} \boldsymbol{z}_j \mathbf{B}_{j,\kappa} &= \boldsymbol{z}_{\kappa-1}\mathbf{B}_{\kappa-1,\kappa} + \boldsymbol{z}_\kappa \mathbf{B}_{\kappa,\kappa} + \boldsymbol{z}_{\kappa+1}\mathbf{B}_{\kappa+1,\kappa}, \\ &= \boldsymbol{z}_{\kappa-1}\mathbf{B}_{\kappa-1,\kappa} + \boldsymbol{z}_\kappa(\mathbf{B}_{\kappa,\kappa} + \mathbf{R}\mathbf{A}_2) = \mathbf{0}. \end{aligned} \tag{11.34}$$

With this substitution, (11.30(a–c)) comprises a system of $\kappa+1$ linear vector equations with as many unknown vectors. However, as these vectors are still dependent, the normalisation (11.30(e)) has to be integrated in it, to yield a unique solution. This normalisation can be written as follows:

$$\begin{aligned} \sum_{i=0}^{\infty} \boldsymbol{z}_i \cdot \mathbf{1}^T &= \sum_{i=0}^{\kappa-1} \boldsymbol{z}_i \cdot \mathbf{1}^T + \sum_{i=\kappa}^{\infty} \boldsymbol{z}_i \cdot \mathbf{1}^T = \sum_{i=0}^{\kappa-1} \boldsymbol{z}_i \cdot \mathbf{1}^T + \sum_{i=\kappa}^{\infty} (\boldsymbol{z}_\kappa \mathbf{R}^i) \cdot \mathbf{1}^T \\ &= \sum_{i=0}^{\kappa-1} \boldsymbol{z}_i \cdot \mathbf{1}^T + \boldsymbol{z}_\kappa \mathbf{R}^\kappa (\mathbf{I} - \mathbf{R})^{-1} \cdot \mathbf{1}^T = 1. \end{aligned} \tag{11.35}$$

Regarding the stability of the modelled system, similar remarks apply as given in Section 11.3.3 for PH/PH/1 queueing systems; condition (11.23) can be validated once the matrices $\mathbf{A}_i$ have been computed.

11.5 Numerical algorithms

In order to really compute steady-state probabilities we have to solve for the matrix $\mathbf{R}$ as well as for the (normalised) boundary equations. We discuss the numerical solution of the boundary equations in Section 11.5.1. Only in special cases, which we do not consider here, can $\mathbf{R}$ be computed explicitly. Normally, $\mathbf{R}$ has to be computed iteratively. We present a simple and straightforward substitution algorithm in Section 11.5.2, after which we present the recently developed logarithmic reduction algorithm in Section 11.5.3.

11.5.1 Solving the boundary equations

The calculation of the boundary probability vectors from the (normalised) system of linear equations can be done using standard techniques, such as e.g., Gaussian elimination or Gauss-Seidel iterations, as can be found in many textbooks [47]. As the number of boundary equations is normally not too large, direct approaches such as Gaussian elimination are in general well-suited. Notice that we need to compute $\mathbf{R}$ before we can solve the linear system of boundary equations (11.35).

11.5.2 A successive substitution algorithm

From the quadratic matrix equation

$$\mathbf{F}(\mathbf{R}) = \mathbf{R}^2\mathbf{A}_2 + \mathbf{R}^1\mathbf{A}_1 + \mathbf{R}^0\mathbf{A}_0 = \mathbf{0}, \tag{11.36}$$

we can derive

$$\mathbf{R} = -(\mathbf{A}_0 + \mathbf{R}^2\mathbf{A}_2)\mathbf{A}_1^{-1}. \tag{11.37}$$

Now, taking as a first guess $\mathbf{R}(0) = \mathbf{0}$, we obtain, the next guess $\mathbf{R}(1) = -\mathbf{A}_0\mathbf{A}_1^{-1}$. We obtain successive better approximations of $\mathbf{R}$ as follows:

$$\mathbf{R}(k+1) = -(\mathbf{A}_0 + \mathbf{R}^2(k)\mathbf{A}_2)\mathbf{A}_1^{-1}, \quad k = 1, 2, \cdots. \tag{11.38}$$

The iteration is stopped when $||\mathbf{F}(\mathbf{R})|| < \epsilon$ (we use $||\mathbf{A}|| = \max_j\{\sum_k |a_{j,k}|\}$). It has been shown that the sequence $\{\mathbf{R}(k),\ k = 0, 1, \cdots\}$ is entry-wise nondecreasing, and that it converges monotonically to the matrix $\mathbf{R}$.

The following remark is of importance when implementing this method in an efficient way. First notice that $\mathbf{R}(1)$ is defined as $-\mathbf{A}_0\mathbf{A}_1^{-1}$. In many cases we will see only a few non-zero entries in the vectors $\boldsymbol{\alpha}$ and $\boldsymbol{T}^0$. Since $\mathbf{A}_0 = (\boldsymbol{T}^0 \cdot \boldsymbol{\alpha}) \otimes \mathbf{I}$ (it describes arrivals to the queue) we see that rows containing just 0's in $\mathbf{A}_0$ will yield similar zero-rows in $\mathbf{R}$. Further iteration steps do not change this situation for the matrix $\mathbf{R}$. Therefore, the matrix $\mathbf{R}$ will often have

2λ	ρ	$E[N_q]$	SS	LR
2	0.0875	0.0023	6	3
4	0.1750	0.0160	11	4
6	0.2625	0.0492	15	4
8	0.3500	0.1122	21	4
10	0.4375	0.2200	28	5
12	0.5250	0.3985	38	5
14	0.6125	0.6963	52	5
16	0.7000	1.2186	72	6
18	0.7875	2.2432	*100*	6
20	0.8750	4.8242	*100*	7
22	0.9581	17.5829	*100*	9

Table 11.1: Analysis results for the $E_2/H_2/1$ queue

many rows completely zero. This fact can be exploited in the multiplications by simply skipping rows which contain only zero entries.

Per iteration step, 3 matrix-multiplications have to be made. If $\mathbf{R}$ is of size $N \times N$, then we require $O(3N^3)$ operations per iteration step. The number of iteration steps heavily depends on the utilisation of the queue under study.

Example 11.2 The $E_2/H_2/1$ queue (II) Consider again the $E_2/H_2/1$ queue in which we assume the following numerical values: $\lambda = 8$, $\mu_1 = 20$ and $\mu_2 = 10$. Due to this choice, $\rho = 0.7$. The matrix $\mathbf{R}$ is calculated with the above simple algorithm, with precision 10^{-8} in 72 steps and equals:

$$\mathbf{R} = \begin{pmatrix} 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.0000 & 0.0000 & 0.0000 & 0.0000 \\ 0.4678 & 0.0970 & 0.2229 & 0.1217 \\ 0.0415 & 0.7879 & 0.0521 & 0.6247 \end{pmatrix}. \tag{11.39}$$

First notice that the matrix $\mathbf{R}$ has two non-zero rows; these correspond to those situations in which arrivals can take place (at each level). There are 4 states per level, but in only two of them arrivals can take place, namely in those states where the Erlang-2 arrival process in its second phase.

To compute the boundary probabilities, we have to solve (11.35) under the normalising condition (11.35). We find $\boldsymbol{z}_0 = (0.1089, 0.1911)$ from which we quickly verify that $\rho = 1 - \boldsymbol{z}_0 \mathbf{1}^T = 0.70$.

In Table 11.1 we show some more analysis results. We vary λ from 1 through 11. Note the increase in the average queue length with increasing utilisation. We tabulated the required number of steps in the successive substitution procedure to compute $\mathbf{R}$ (column SS). Note that when ρ increases the iterative solution of $\mathbf{R}$ slows down tremendously; since we stopped the iterative procedure after 100 steps, the last three rows do not represent accurate results.

We finally note that there have been developed a number of more efficient methods based on successive substitutions than the one presented here. These methods rely on the computation of the intermediate matrices $\mathbf{G}$ and $\mathbf{U}$, which will be introduced and related to $\mathbf{R}$ in the next section. However, since also these algorithms are outperformed by the recently developed logarithmic reduction algorithm, we do not discuss them any further here; for further details, refer to [31, 34].

11.5.3 The logarithmic reduction algorithm

Recently, Latouche and Ramaswami developed the logarithmic reduction (LR) algorithm to compute $\mathbf{R}$ [32]. It is regarded as the most efficient to date. The algorithm is based on the following three matrix-quadratic equations (here given in case of a discrete-time QBD):

$$\begin{aligned} \mathbf{G} &= \mathbf{A}_2 + \mathbf{A}_1\mathbf{G} + \mathbf{A}_0\mathbf{G}^2, \\ \mathbf{R} &= \mathbf{A}_0 + \mathbf{R}\mathbf{A}_1 + \mathbf{R}^2\mathbf{A}_2, \\ \mathbf{U} &= \mathbf{A}_1 + \mathbf{A}_0(\mathbf{I}-\mathbf{U})^{-1}\mathbf{A}_2. \end{aligned} \tag{11.40}$$

The three unknown matrices (all of size $N \times N$) have the following interpretation:

- the element $g_{i,j}$ of the matrix $\mathbf{G}$ is the probability that, starting from state $(1,i)$, the QBD process eventually visits level 0 in state $(0,j)$;
- the element $u_{i,j}$ of the matrix $\mathbf{U}$ is the *taboo probability* that, starting from state $(1,i)$, the QBD eventually returns to level 1 by visiting state $(1,j)$, under taboo of level 0, that is, without visiting level 0 in between;
- the element $r_{i,j}$ of the matrix $\mathbf{R}$ can be interpreted as the expected number of visits into state $(1,j)$, starting from state $(0,i)$, until the first return to level 0.

Once one of these matrices is known, the other ones can be readily computed. For instance, having computed $\mathbf{G}$, we can derive $\mathbf{U} = \mathbf{A}_1 + \mathbf{A}_0\mathbf{G}$, and $\mathbf{R} = -\mathbf{A}_0\mathbf{U}^{-1}$.

When the QBD is recurrent, the matrix $\mathbf{G}$ is stochastic. We can therefore iteratively compute successive estimates for $\mathbf{G}$, denoted as $\mathbf{G}(k)$, $k = 1, 2, \cdots$, until the row sums equal (almost) 1, that is, until $||\mathbf{1}^T - \mathbf{G}\mathbf{1}^T|| < \epsilon$, where ϵ is a prespecified accuracy criterion. We have: $\lim_{k\to\infty} \mathbf{G}(k) = \mathbf{G}$.

The (i,j)-th element of $\mathbf{G}(k)$, denoted as $g_{i,j}(k)$, now has the following interpretation [32]: $g_{i,j}(k)$ is the probability that, starting from state $(1,i)$, the QBD visits level 0 in state $(0,j)$, under taboo of levels $k+1$ and beyond. Clearly, to compute $\mathbf{G}$ accurately, we should make k large, so that even very long queue lengths are allowed for (so that effectively the taboo is not there). The algorithms developed in the past, all compute successive matrices $\mathbf{G}(k)$ by increasing k one-at-a-time. Especially for queueing models with high utilisation, large queue lengths occur quite often, thus requiring many iteration steps (as

```
1.   B_0 := -A_1^{-1} A_0 ;  B_2 := -A_1^{-1} A_2
2.   G := B_2 ;  T := B_0
3.   while ||1^T - G 1^T|| > ε do
4.       D := B_0 B_2 + B_2 B_0
5.       B_0 := (I - D)^{-1} B_0^2
6.       B_2 := (I - D)^{-1} B_2^2
7.       G := G + T B_2
8.       T := T B_0
9.   od
10.  U := A_1 + A_0 G
11.  R := -A_0 U^{-1}
```

Figure 11.4: The logarithmic-reduction algorithm for continuous-time QBDs

witnessed by Table 11.1). In contrast, the new LR algorithm doubles the value of k in every step, thus reaching a far smaller effective taboo, given a fixed number of iteration steps. In practice, 20 iteration steps suffice most of the time, thus allowing the QBD to have upsurges to levels as high as 2^{20} (about 1 million).

Without going into the detailed derivations, we now present the basic equations to compute $\mathbf{G}$ (note that we treat the case of a continuous-time QBD here; in [31, 32, 47], the discrete-time variant is presented):

$$\mathbf{G} = -\sum_{k=0}^{\infty} \underbrace{\left(\prod_{i=0}^{k-1} \mathbf{B}_0(i) \right)}_{\mathbf{T}(k)} \mathbf{B}_2(k), \tag{11.41}$$

with

$$\mathbf{B}_0(0) = -\mathbf{A}_1\mathbf{A}_0, \quad \text{and} \quad \mathbf{B}_2(0) = -\mathbf{A}_1\mathbf{A}_2, \tag{11.42}$$

and

$$\mathbf{B}_i(k+1) = (\mathbf{I} - \underbrace{(\mathbf{B}_0(k)\mathbf{B}_2(k) + \mathbf{B}_2(k)\mathbf{B}_0(k))}_{\mathbf{D}(k)})^{-1} \cdot \mathbf{B}_i^2(k), \quad i = 0, 2. \tag{11.43}$$

The corresponding algorithm is given in Figure 11.4. Line 1 represent the initialisation according to (11.42), and in line 2 $\mathbf{G}$ is set equal to the first term in (11.41); note that when $k = 0$, $\mathbf{T}(0) = \mathbf{I}$ so that $\mathbf{G} = \mathbf{B}_2$. Then, until $\mathbf{G}$ truly is a stochastic matrix, successive terms for $\mathbf{D}(k)$ and the matrices $\mathbf{B}_i(k)$ are computed in lines 4–6, according to (11.43). In line 7 the current term is added to $\mathbf{G}$ and in line 8 the new value for $\mathbf{T}(k)$ is computed. When the iteration ends, $\mathbf{U}$ and $\mathbf{R}$ are finally computed.

Regarding the complexity of the LR algorithm, it can be shown that the number of operations per iteration step equals $O(\frac{25}{3}N^3)$. This is about 8 times

more than in case of the successive substitution method. However, the strength of the LR algorithm lies in the fact that it does need far less iterations; roughly speaking, when the successive substitution method requires k_ϵ iteration steps to reach an accuracy ϵ, then the LR-algorithm requires only $O(\log_2 k_\epsilon)$ steps. Recently, a slightly faster iteration step has been proposed by Wagner *et al.* [48]; their algorithm requires $O(\frac{19}{3}n^3)$ per iteration steps and the same number of steps as the LR algorithm.

Example 11.3 The $\mathbf{E}_2/\mathbf{H}_2/1$ queue (III) In the last column of Table 11.1 we show the number of steps required by the LR algorithm to reach the same accuracy as the SS algorithm. The increase in performance is indeed tremendous. The number of steps still increases for increasing ρ, however, only very slowly. Needing more then 20 iterations to compute $\mathbf{R}$ has been an exception in the case studies we have performed so far.

Example 11.4 The IPP/$\mathbf{E}_k$/1 queue (I) To model the bursty nature of many arrival processes we can use an IPP which has two states: "off" (0) and "on" (1) (see also Appendix 11.A). The transition rate from state i to state j equals $\gamma_{i,j}$. In state i, jobs are generated as a Poisson process with rate λ_i. Such an IPP is completely described by the matrix $\mathbf{T}$:

$$\mathbf{T} = \begin{pmatrix} -\gamma_{0,1} & \gamma_{0,1} \\ \gamma_{1,0} & -(\gamma_{1,0}+\lambda) \end{pmatrix}, \tag{11.44}$$

and the vector $(\boldsymbol{\alpha}, \alpha_A) = (0, 1, 0)$. Note that due to the choice of $\boldsymbol{\alpha}$, we have established that after an arrival (an absorption) the PH distribution stays in state 1 so that the arrival process remains in a burst. An important parameter for an IPP is the *burstiness* b. It is defined as the ratio between the arrival rate in a burst and the overall average arrival rate:

$$b = \frac{\text{arrival rate in a burst}}{\text{overall average arrival rate}} = \frac{\lambda}{\lambda\gamma_{0,1}/(\gamma_{0,1}+\gamma_{1,0})} = \frac{\gamma_{0,1}+\gamma_{1,0}}{\gamma_{0,1}}. \tag{11.45}$$

We will first study the influence of the burstiness of the IPP on the average number of customers queued while keeping the utilisation constant. This allows us to investigate quantitatively the influence of b on the performance. At the same time, we vary the number of Erlang phases k in the service time distribution from 1 to 10. We address four different burstiness levels: $b = 1$, 2.75, 5.5 and 11; we kept $\gamma_{0,1} = 10$ and varied $\gamma_{1,0}$ from 17.5 via 45 to 100 in the latter three cases. We want to keep the utilisation equal to 45.45% in all cases and we therefore adjusted λ to 45.45, 125, 250 and 500 respectively; the service rate $\mu = 100$. Notice that the case $b = 1$ corresponds to the case where the IPP has been replaced by a normal Poisson arrival process (or an IPP with $\gamma_{1,0} = 0$).

The results are depicted in Table 11.2. We observe that for all burstiness levels the average number of queued customers decreases as the service times become more deterministic, i.e., as k increases. Notice, however, that the relative decrease becomes less pronounced for higher values of b. This implies that for

	$E[N_q]$ for given b			
k	$b = 1$	$b = 2.75$	$b = 5.5$	$b = 11$
1	0.375	1.759	2.859	3.496
2	0.284	1.595	2.741	3.394
3	0.253	1.537	2.702	3.361
4	0.237	1.508	2.682	3.344
5	0.227	1.489	2.670	3.334
6	0.221	1.477	2.662	3.328
7	0.216	1.468	2.657	3.323
8	0.213	1.462	2.652	3.320
9	0.210	1.457	2.649	3.317
10	0.208	1.452	2.646	3.315

Table 11.2: The average queue length $E[N_q]$ in the IPP/E$_k$/1 queue, for increasing number of phases k and different burstiness factors ($\rho = 0.4545$)

(very) bursty sources, the service time distribution (or its second moment) plays a less important role, as far as the average performance variables are concerned. Looking at the arrival rates, we see that for the larger burstiness values, the queue is overloaded during the periods the arrival process is active; this causes the enormous increase in average number of customers queued for larger values of b (and so, via Little's law, in the average waiting time).

In Figure 11.5 we show the effect of increasing the number of Erlang stages in the service time distribution from 1 to 50. The figure shows the expected queue length in case the arrival process has $b = 11$ and utilisation 72.73%. Notice that altering k to a value around 5 already makes the services very deterministic. Adding more phases does not change the performance measure of interest very much anymore; it does change the number of states per level. The number of rows and columns in $\mathbf{R}$ equals $2k$.

11.6 iSPN specification and measure computation

In Section 11.6.1 we reflect on an algorithm to translate an iSPN to the underlying QBD process. In Section 11.6.2 we then present a number of techniques to evaluate reward-based measures for iSPNs in an efficient way.

11.6.1 From iSPN to the underlying QBD

The translation of an iSPNs to the underlying QBD can in principle be performed with well-known algorithms for state-space generation, as discussed in Chapter 9. It should be noted, however, that iSPNs have an unbounded number of states so that "standard" algorithm will not terminate. Instead, a proper

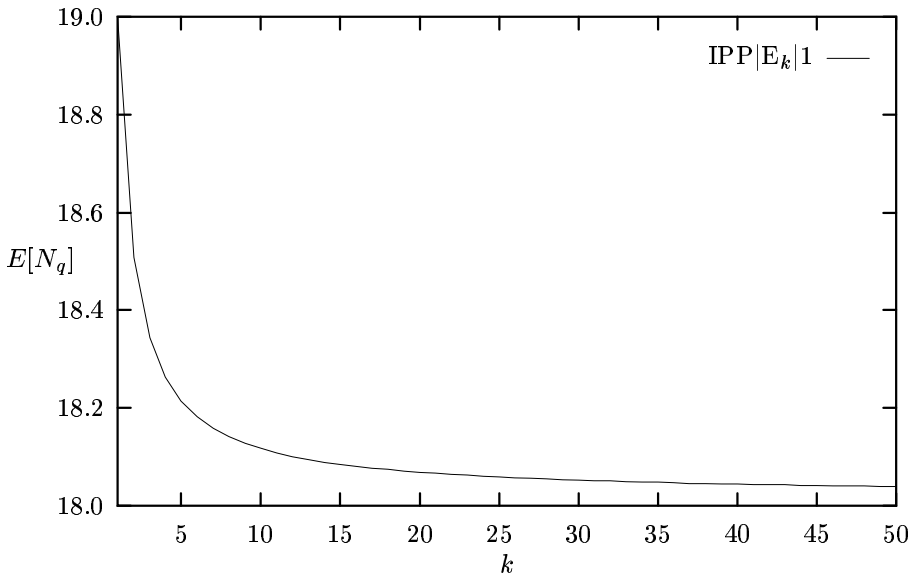


Figure 11.5: $E[N_q]$ for the IPP/E$_k$/1 queue as a function of the number of phases k in the Erlang service time distribution

translation algorithm should end when it recognises that it is exploring the state space in the repetitive part. One practical problem that comes along here is that the stated requirements are not easily verified in general. When inhibitor arcs and enabling functions are allowed in their most general setting, the verification problem is even undecidable (see also [1]). Therefore, the given requirements should be interpreted as being sufficient to allow for the matrix-geometric solution. However, we are free to pose extra requirements, in order to ease the decision task, albeit possibly at the cost of less modelling flexibility. Taking these considerations into account, we decided on the following practical restrictions on iSPNs:

- input and output arcs connected to place P_0 may only have multiplicity one;
- marking dependent rates and weights on the contents of place P_0 and enabling functions using the marking of place P_0 are not allowed.

Up till now, these restrictions did not bother us in performance evaluation studies.

In stating the above restrictions we assumed that the identity of the place that may contain an unbounded number of tokens is known in advance. Indeed, when using iSPNs it is normally easy for a modeller to indicate its identity; the whole model is normally built around this place. On the other hand, place

P_0 can also be recognised automatically. Since only for P_0 the marking is unbounded, place P_0 will not appear in any place invariant. Thus, given an iSPN, an invariant analysis as discussed in Chapter 2 will reveal the identity of P_0.

A final problem in the translation algorithm is the determination of the level number κ where the repetition starts, in order to stop the state-space generation process timely. Although this is easy for a human being, doing this for instance "by inspection" of the upper left part of the partially generated matrix $\mathbf{Q}$, it is less easy to grasp in an algorithm. However, given the above restrictions, we have been able to prove that once two successive levels are the same, all levels beyond these will also we the same. The 3-page proof, based on finite induction, goes beyond the scope of this chapter [17].

11.6.2 Efficient computation of reward-based measures

Once the steady-state probabilities are known, reward-based performance measures can be computed easily. Let $r : \mathrm{RS} \to \mathbb{R}$ denote a real-valued reward function defined on the state space of the model. The steady-state expected reward is then computed as

$$E[X] = \sum_{i=0}^{\infty} \sum_{\boldsymbol{m} \in \mathcal{R}'(i)} z_{i,\boldsymbol{m}} r(i, \boldsymbol{m}). \tag{11.46}$$

Without any further restrictions on the form of $r(i, \boldsymbol{m})$ we cannot further reduce the above expression. Thus, to compute $E[X]$ in such cases, we start the infinite summation and continue to add more terms until the additional terms are smaller than a certain threshold. Assuming that all rewards are positive, we thus compute a lower bound on the actual expected reward.

There are, however, quite a number of reward-based measures that are of general interest and that can be computed more efficiently, without involving infinite summations. we discuss a number of these special cases below.

Reward function only depending on the level

If the reward function only depends on the level number and not on the inter-level state, that is, if $r(i, \boldsymbol{m}) = r(i, \boldsymbol{m}')$, for all $i \in \mathbb{N}$ and for all $\boldsymbol{m}, \boldsymbol{m}' \in \mathcal{R}'(i)$, we may write $r(i, \boldsymbol{m}) = r(i)$. Note that this type of reward-based measure typically concerns the unbounded place P_0. When computing the

- probability that the number of tokens in P_0 is above a certain threshold l, we set $r(i) = \mathbf{1}\{P_0 > l\}$;
- probability that P_0 contains exactly l tokens, we set $r(i) = \mathbf{1}\{P_0 = l\}$.
- expected number of tokens in P_0, we set $r(i) = i$.

For these cases, we can write:

$$\begin{aligned} E[X] &= \sum_{i=0}^{\infty} r(i)(\boldsymbol{z}_i \cdot \mathbf{1}^T) \\ &= \sum_{i=0}^{\kappa-1} r(i)(\boldsymbol{z}_i \cdot \mathbf{1}^T) + \sum_{i=\kappa}^{\infty} r(i)(\boldsymbol{z}_i \cdot \mathbf{1}^T) \\ &= \underbrace{\sum_{i=0}^{\kappa-1} r(i)(\boldsymbol{z}_i \cdot \mathbf{1}^T)}_{LT} + \sum_{j=0}^{\infty} r(j)(\boldsymbol{z}_\kappa \mathbf{R}^{\kappa+j} \cdot \mathbf{1}^T). \end{aligned} \tag{11.47}$$

The first, finite, sum does not comprise a problem. We therefore concentrate on the second sum now, for each of the measures identified above:

- In case $r(i) = \mathbf{1}\{P_0 = l\}$ the infinite summation contains at most one non-zero term, so that we have:

$$E[X] = LT + \boldsymbol{z}_l \cdot \mathbf{1}^T. \tag{11.48}$$

- If we want to compute the probability that P_0 contains more than l tokens, this can be rewritten as 1 minus the probability that P_0 contains at most $l-1$ tokens. Thus, the infinite summation also reduces to a finite one.

- A more complex case arise when $r(i) = i$, however, also here a closed-form expression can be derived:

$$\begin{aligned} E[X] &= \sum_{i=0}^{\kappa-1} i(\boldsymbol{z}_i \cdot \mathbf{1}^T) + \sum_{j=0}^{\infty} (\kappa + j)(\boldsymbol{z}_\kappa \mathbf{R}^j \cdot \mathbf{1}^T) \\ &= \sum_{i=0}^{\kappa-1} i(\boldsymbol{z}_i \cdot \mathbf{1}^T) + \sum_{j=0}^{\infty} \kappa(\boldsymbol{z}_\kappa \mathbf{R}^j \cdot \mathbf{1}^T) + \sum_{j=0}^{\infty} j(\boldsymbol{z}_\kappa \mathbf{R}^j \cdot \mathbf{1}^T) \\ &= \sum_{i=0}^{\kappa-1} i(\boldsymbol{z}_i \cdot \mathbf{1}^T) + \kappa \boldsymbol{z}_\kappa \left(\sum_{j=0}^{\infty} \mathbf{R}^j \right) \mathbf{1}^T + \boldsymbol{z}_\kappa \mathbf{R} \left(\sum_{j=0}^{\infty} j \mathbf{R}^{j-1} \right) \mathbf{1}^T \\ &= \sum_{i=0}^{\kappa-1} i(\boldsymbol{z}_i \cdot \mathbf{1}^T) + \kappa \boldsymbol{z}_\kappa \left(\sum_{j=0}^{\infty} \mathbf{R}^j \right) \mathbf{1}^T + \boldsymbol{z}_\kappa \mathbf{R} \frac{d}{d\mathbf{R}} \left((\mathbf{I} - \mathbf{R})^{-1} \right) \mathbf{1}^T \\ &= \sum_{i=0}^{\kappa-1} i(\boldsymbol{z}_i \cdot \mathbf{1}^T) + \left(\boldsymbol{z}_\kappa (\mathbf{I} - \mathbf{R})^{-1} (\kappa \mathbf{I} + \mathbf{R}(\mathbf{I} - \mathbf{R})^{-1}) \right) \mathbf{1}^T. \end{aligned} \tag{11.49}$$

Reward function independent of level and P_0

When computing mean place occupancies for places other than P_0, the rewards depend on $\boldsymbol{m}$ rather than on i: $r(i, \boldsymbol{m}) = r(\boldsymbol{m})$, irrespective of i. We start with

the general reward-based expression:

$$\begin{aligned} E[X] &= \sum_{i=0}^{\infty} \sum_{\boldsymbol{m} \in \mathcal{R}'(i)} z_{i,\boldsymbol{m}} r(i,\boldsymbol{m}) \\ &= \underbrace{\sum_{i=0}^{\kappa-1} \sum_{\boldsymbol{m} \in \mathcal{R}'(i)} z_{i,\boldsymbol{m}} r(i,\boldsymbol{m})}_{LT} + \sum_{i=\kappa}^{\infty} \sum_{\boldsymbol{m} \in \mathcal{R}'(i)} z_{i,\boldsymbol{m}} r(i,\boldsymbol{m}). \quad (11.50) \end{aligned}$$

The left additive term LT again does not comprise problems. The right term can be reduced considerably, when changing the summation order and using the fact that the rewards are level-independent as follows:

$$\begin{aligned} E[X] &= LT + \sum_{\boldsymbol{m} \in \mathcal{R}'(\kappa)} r(\boldsymbol{m}) \sum_{i=\kappa}^{\infty} z_{i,\boldsymbol{m}} \\ &= LT + \sum_{\boldsymbol{m} \in \mathcal{R}'(\kappa)} r(\boldsymbol{m}) \sum_{i=0}^{\infty} (\boldsymbol{z}_\kappa \mathbf{R}^i) \cdot \boldsymbol{e}_{\boldsymbol{m}}, \quad (11.51) \end{aligned}$$

where $\boldsymbol{e}_{\boldsymbol{m}}$ is a vector with a single one at its $\boldsymbol{m}$-th position. The right-most sum can be reduced, yielding:

$$E[X] = LT + \sum_{\boldsymbol{m} \in \mathcal{R}'(\kappa)} r(\boldsymbol{m}) \boldsymbol{z}_\kappa (\mathbf{I} - \mathbf{R})^{-1} \boldsymbol{e}_{\boldsymbol{m}}. \quad (11.52)$$

Similar expressions are obtained when computing the probability that the number of tokens in a certain place (unequal P_0) is larger or smaller than a threshold l.

11.7 Application studies

To demonstrate the usability of iSPNs we present three small application examples. We present a queueing model with delayed service in Section 11.7.1; this model is so simple that we can solve it explicitly. We then continue with two more realistic models: a model of a connection management mechanism in Section 11.7.2 and a model of a queueing system with checkpointing and recovery in Section 11.7.3.

11.7.1 A queueing model with delayed service

Considers a single server queueing system (see Figure 11.6) at which customers arrive as a Poisson process with rate λ via transition `arr` and are served with rate μ via transition `serve`. Before service, arriving customers are stored in a `buffer`. Service is not immediately granted to an arriving customer, even not is the server is idle at that time (a token in place `sleep`). Only after there are

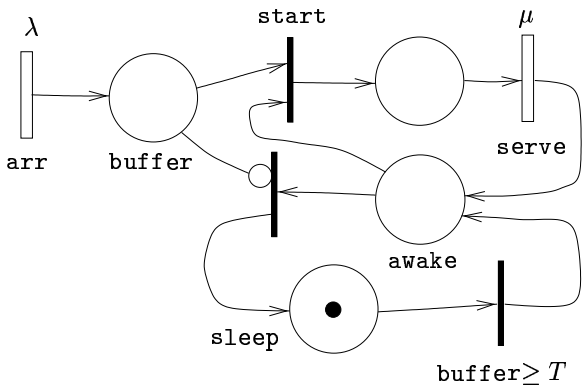


Figure 11.6: iSPN of a single server queueing system with delayed service

at least T (for threshold) customers queued, the server awakes and starts its duties. This is enforced by an enabling function associated with the immediate transition `wake-up`: `#buffer` $\geq T$. The server subsequently remains awake until the buffer becomes empty, after which it resumes sleeping.

For a threshold $T = 3$, the corresponding CTMC is given in Figure 11.7. From the models, it can easily be seen that they fulfill Requirements 1–3 with $\kappa = T = 3$. The generator matrix has the following form:

$$\mathbf{Q} = \left(\begin{array}{c|c|cc|ccc|c} -\Sigma & \lambda & 0 & 0 & 0 & 0 & 0 & \cdots \\ \hline 0 & -\Sigma & 0 & \lambda & 0 & 0 & 0 & \cdots \\ \mu & 0 & -\Sigma & 0 & \lambda & 0 & 0 & \cdots \\ \hline 0 & 0 & 0 & -\Sigma & 0 & \lambda & 0 & \cdots \\ 0 & 0 & \mu & 0 & -\Sigma & \lambda & 0 & \cdots \\ \hline 0 & 0 & 0 & 0 & \mu & -\Sigma & \lambda & \cdots \\ \hline 0 & 0 & 0 & 0 & 0 & \mu & -\Sigma & \cdots \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & \mu & \cdots \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}\right),$$

where Σ is chosen such that the row sums equal 0. From this matrix we observe that the $\mathbf{A}$-matrices are in fact scalars or 1×1 matrices: $\mathbf{A}_0 = (\lambda)$, $\mathbf{A}_1 = (-(\lambda+\mu))$ and $\mathbf{A}_2 = (\mu)$. From these matrices we derive $\mu\mathbf{R}^2 - (\lambda+\mu)\mathbf{R} + \lambda = 0$ which has as only valid solution $\mathbf{R} = (\lambda/\mu)$. From this, it once again becomes clear that $\mathbf{R}$ takes over the role of ρ in simpler queueing analysis, such as in the M/M/1 queue.

Denoting $\boldsymbol{z}_i = z_i$, for $i = 0$ or $i = 3, 4, \cdots$, and $\boldsymbol{z}_i = (z_{i,S}, z_{i,A})$, for $i = 2, 3,$

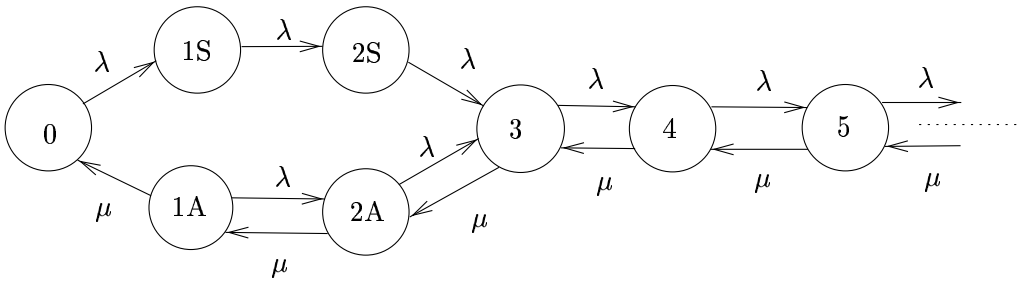


Figure 11.7: QBD of the single server queueing system with delayed service in case $T = 3$

the boundary equations become:

$$\begin{cases} -\lambda z_0 + \mu z_{1A} = 0, \\ \lambda z_0 - \lambda z_{1S} = 0, \\ -(\lambda+\mu) z_{1A} + \mu z_{2A} = 0, \\ \lambda z_{1S} - \lambda z_{2S} = 0, \\ \lambda z_{1A} - (\lambda+\mu) z_{2A} + \mu z_3 = 0, \\ \lambda z_{2S} + \lambda z_{2A} - (\lambda+\mu) z_3 + \mu z_4 = 0, \\ z_0 + z_{1S} + z_{1A} + z_{2S} + z_{2A} + z_3(1-\rho)^{-1} = 1. \end{cases} \tag{11.53}$$

As a numerical example, consider the case where $\lambda = 2$, $\mu = 3$, $T = 3$ and, consequently, $\rho = 2/3$. The matrix-geometric solution results in the following boundary steady-state probabilities:

$$\begin{cases} z_0 = 0.11111, & z_{1A} = 0.07407, & z_{1S} = 0.11111, \\ z_{2A} = 0.12346, & z_{2S} = 0.11111, & z_3 = 0.15638. \end{cases}$$

Using these probabilities, and $z_i = z_3 \rho^{i-3}$, $i = 4, 5, \cdots$, we obtain for the average number of customers in the system $E[N] = 3.00$.

11.7.2 Connection management in communication systems

In a communication infrastructure based on the asynchronous transfer mode (ATM), a connection-oriented service is normally offered. Via the ATM adaptation layers 3/4 and 5 (AAL 3/4 and 5), also connectionless services can be provided [42, 45]. Packets arriving at the AAL service boundary to make use of such a service, suffer a possible delay from the connection establishment at the ATM service boundary, unless there already exists a connection when the packet arrives. Once the connection has been established, all buffered packets can be transmitted and the connection can be released. This can be done immediately, or with some delay. The former has as disadvantage that a connection is being maintained when it is not needed, however, it has as advantage that some packets might profit from the fact that there is still a connection when they arrive.

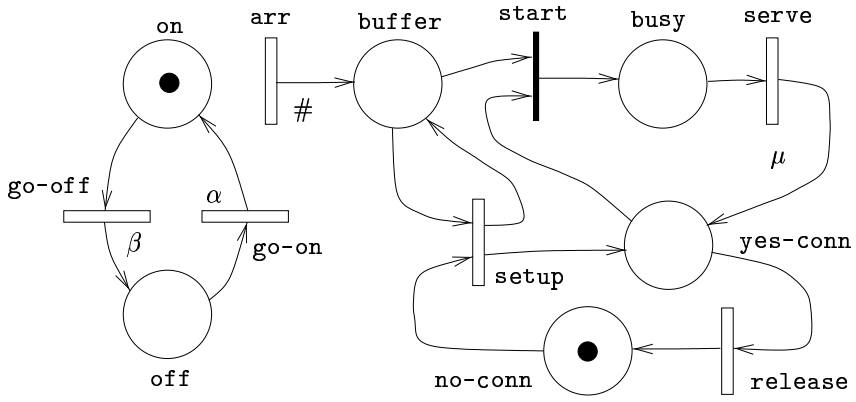


Figure 11.8: iSPN model of the OCDR mechanism

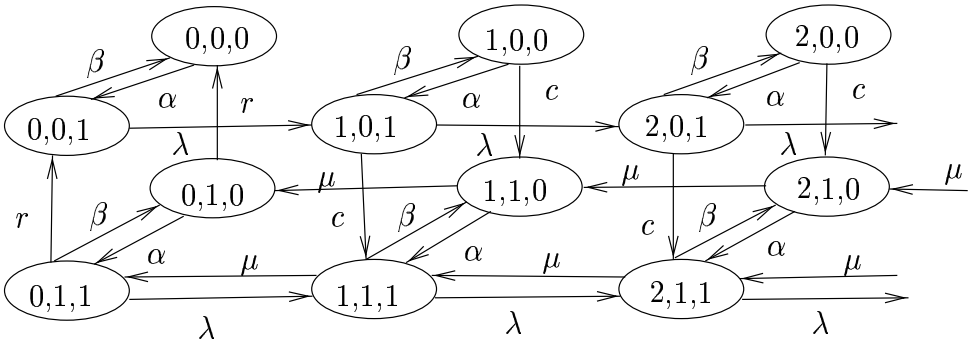


Figure 11.9: CTMC of the OCDR mechanism

Clearly, there is a trade-off between the release delay, the costs and of maintaining an unused connection and the perceived performance (average delay). The above way of implementing connectionless services, has been proposed by Heijenk *et al.* [25] under the name "on-demand connection with delayed release" (OCDR).

An iSPN model for such a system is given in Figure 11.8. Packets arrive via transition `arr` and are placed in the `buffer`. The rate of transition `arr` is modulated by an independent on/off-model. A token in place `on` or `off` models the fact that the source is in a burst or not, respectively. When in a burst, packets are generated according to a Poisson process with rate λ packets/second. When not in a burst, no packets are generated. The transitions `go-on` and `go-off`, with rates α and β respectively, model the time durations the source remains in the off and on state. The service rate is μ Mbps and the average packet length is denoted l.

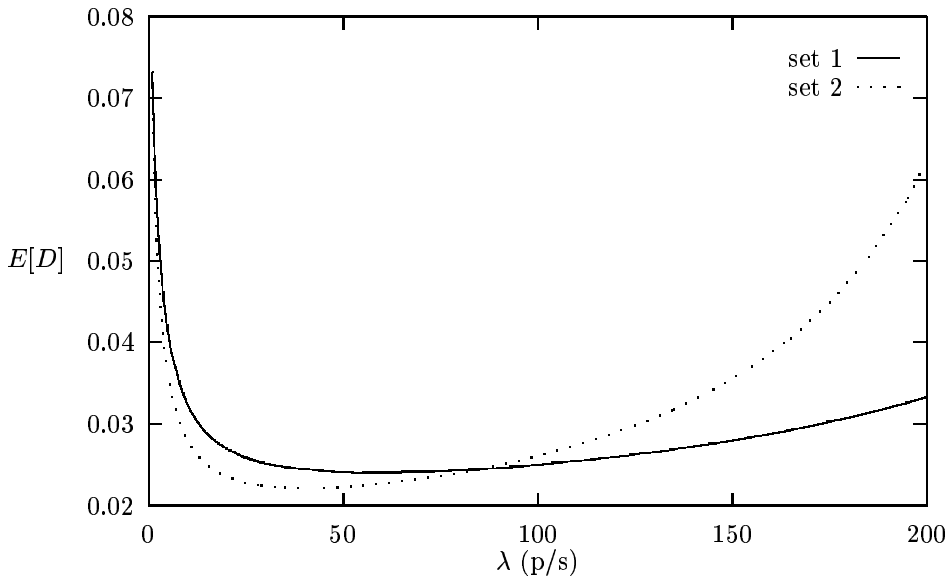


Figure 11.10: The expected delay $E[D]$ (in seconds) as a function of the arrival rate λ in a burst

If the server is busy, there will be a token in place `busy` and arriving packets have to wait on their turn. If the server is idle, but there is no connection available, signified by a token in place `no-conn`, a connection will be established, causing a negative exponential delay with average length $1/c$ (transition `set-up`). Once there is a connection, normal packet transmissions can take place. Once the buffer is empty, the connection is released with a negative exponential delay with average length $1/r$ (transition `release`).

The corresponding CTMC is given in Figure 11.9. In this model, the states space RS $= \{(i,j,k)|i \in \mathbb{N}, j,k = 0,1\}$. Parameter i denotes the number of packets in the system, j denotes whether there is a connection ($j = 1$) or not ($j = 0$), and k denotes whether the arrival process is in a burst ($k = 1$) or not ($k = 0$). As can be seen from the CTMC, but as can also be verified using Requirements 1–3, this model has a structure that allows for a matrix-geometric solution. Every level consists of the $L = 4$ states with i packets present, i.e., $\mathcal{R}(i) = \{(i,0,0),(i,0,1),(i,1,0),(i,1,1)\}$.

Clearly, $\kappa = 1$ and the boundary equations are given by the global balance equations for the first two levels, plus the normalisation equation, in total yielding a system of 8 linear equations. The $L \times L$ matrix $\mathbf{R}$ with $L = 4$ now has to be solved numerically.

As measures of interest we could address: (i) the average node delay $E[D]$ (in seconds); (ii) the average reserved bandwidth $E[B_W]$ (in Mbps); and (ii) the expected number of connection establishments per second $E[C]$ (in per-second).

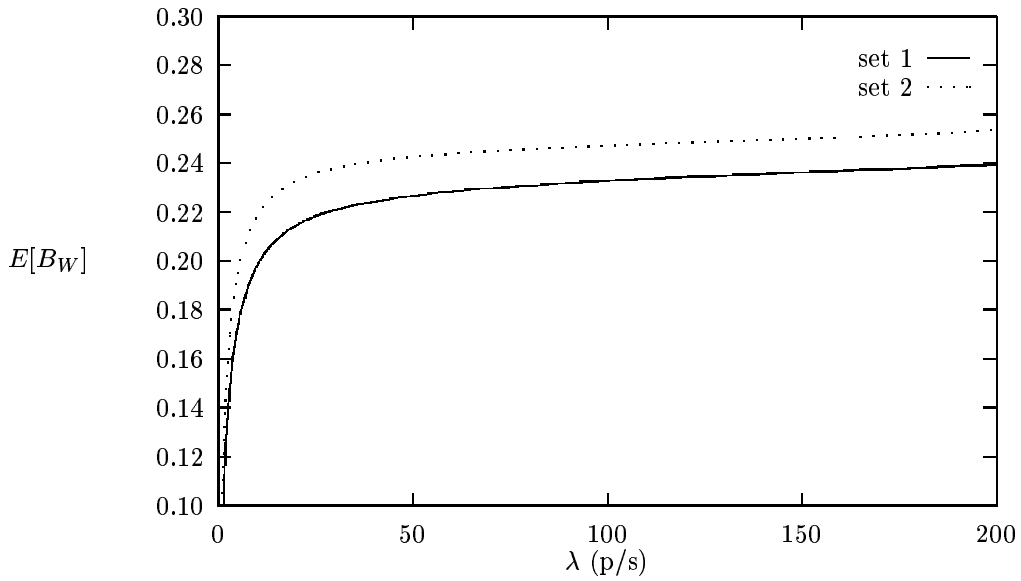


Figure 11.11: The expected bandwidth $E[B_W]$ (in Mbps) as a function of the arrival rate λ in a burst

All these quantities can be expressed in closed-form using $\mathbf{R}$ and the boundary vector $\boldsymbol{z}_0$ (for details, see [25]). Under the assumption that communication capacity can be claimed in various amounts, the service rate μ can be chosen freely. Given a certain workload, a higher requested transmission speed μ will yield smaller connection times, however, at higher costs per time unit. The parameter μ together with the connection release rate r are therefore interesting quantities to control the system performance and cost.

Let us now turn to some numerical results. We assume that $\alpha = 1.0$, $\beta = 0.04$, $c = 10.0$, and $l = 10$ kbit. We address the following two combinations of transmission and release rates: $(\mu, r)_1 = (336.0, 1.0)$ and $(\mu, r)_2 = (236.0, 0.5)$. In the first case, the transmission speed is relatively high, but connections are rapidly released after usage. In the second case, a lower transmission speed is used, but a connection is maintained longer. Therefore, arriving packets have a smaller probability to perceive an extra connection setup delay.

In Figure 11.10 we depict the expected delay $E[D]$ (in seconds) as a function of the arrival rate λ in a burst. Although for $\lambda \approx 85$ the average delay values coincide ($E[D] \approx 24.5$), for changing λ this is certainly not the case. For the first parameter set, the average delay is less sensitive to changes in λ, especially towards higher values. For smaller values of λ, the average delay is smaller for parameter set 2. Surprisingly, the less sensitive solution requires a smaller average bandwidth as well, as illustrated in Figure 11.11. The number of connection establishments, however, is higher, as illustrated by Figure 11.12. Since

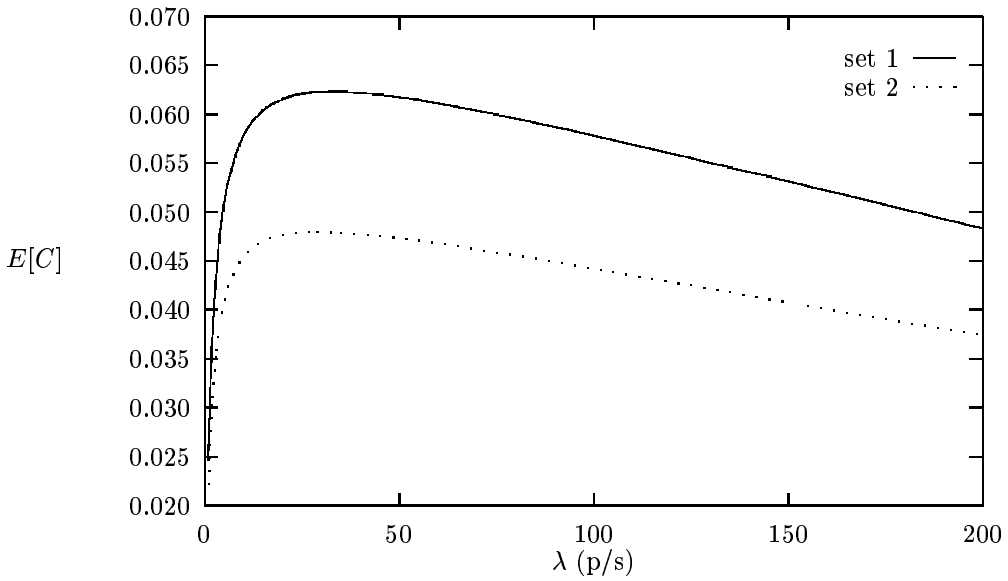


Figure 11.12: The expected connection setup rate $E[C]$ (in s^{-1}) as a function of the arrival rate λ in a burst

the latter can also be associated with costs in a B-ISDN context, the price for the less sensitive delay behaviour and for the smaller bandwidth consumption is paid here. Also observe, that for higher traffic, the number of connection establishments decreases, i.e., a connection that is once established, is used for a long time since the probability of having a connection and no packets present decreases with larger λ.

11.7.3 A system with checkpointing and recovery

In this section we present an analysis of the time to complete tasks on systems that change their structure in the course of time (multi-mode systems). The task-completion time of jobs on multi-mode systems has been studied by several authors. Bobbio describes the completion-time problem as the time-to-absorption problem in a CTMC with an absorbing state [2]. Chimento and Trivedi consider different types of failures that can occur as well as different ways in which failed services might be resumed [8]. The above studies aim at the completion time distribution of single jobs; effects of queueing, due to congestion, are not taken into account. In [40] the performance of a queueing system in which the server is subject ot breakdowns and repairs is studied, so that queueing *is* taken into account. Typically, variants of queues of M/G/1 type are studied, where, depending on the status of the server, jobs experience a delay in the queueing system. Related models occur when studying the effect

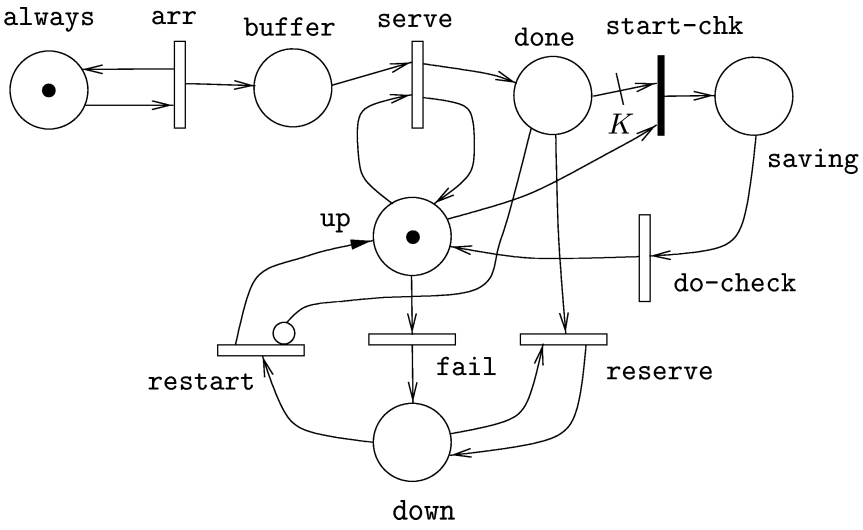


Figure 11.13: iSPN of the transaction processing system model with checkpointing and recovery

of checkpointing strategies on the performance of computer systems that are subject to failures and repairs [7, 41, 44]. When increasing the checkpointing frequency, the amount of overhead increases, however, the amount of work to be done after a failure and subsequent rollback, i.e., the actual recovery time, decreases. This interesting trade-off has lead researchers to study the optimality of checkpointing strategies in various situations.

To illustrate the use of iSPNs we address a job completion-time problem taking into account queueing. We consider a simple model of a transaction processing system where jobs (transactions) arrive in a buffer. A single server is normally available to process these jobs. After K jobs have been processed, a checkpoint is made. We refer to K as the checkpointing interval. When making a checkpoint, the server is unable to serve other jobs. When the server is idle or processing ordinary jobs, it might fail. Once the server has failed, it requires a repair after which it needs to re-serve all the jobs that were processed since the last checkpoint. After that, the server becomes available for ordinary job processing. For simplicity, it is assumed that the server cannot fail when it is checkpointing or recovering. This assumption can be changed without altering the fundamental solution approach proposed here. Also, it is assumed that the service times, the checkpointing time and the repair time are exponentially distributed and that the arrivals form a Poisson process. The corresponding iSPN is depicted in Figure 11.13. In Table 11.3 we summarise the meaning of the places and transitions.

Interesting reward-based performance measures are, among others, the average number of jobs queued, the average number of jobs that has not yet been

place	meaning	init
`buffer`	input queue of jobs	0
`up`	server available	1
`down`	server down and recovering	0
`done`	number of jobs processed since last checkpoint	0
`saving`	server making a checkpoint	0
`always`	infinite source of jobs	1
transistion	**meaning**	**rate**
`arr`	arrivals	λ
`serve`	services	μ
`reserve`	recovery	μ
`fail`	failures	ϕ
`do-check`	checkpointing	σ
`restart`	put into normal operation	β
`start-chk`	start checkpointing	1

Table 11.3: Places and transitions in the iSPN describing the job completion time problem including queueing aspects

checkpointed, the percentage of time that the processor is available for normal processing and the probability that the buffer is empty (or not), all as a function of the checkpointing interval K.

We have evaluated the above described model for three different scenarios (see Table 11.4) for varying checkpointing intervals K. First notice that the number of (tangible) states in the repeating levels equals $2K + 1$. This can be explained as follows. As long as the system is up, i.e., there is a token in place `up`, there can be 0 through $K - 1$ tokens in place `done`. As the the K-th token arrives in place `done`, transition `start-chk` fires, yielding a single token in place `saving`, and none in `up`. This makes $K + 1$ different states already. Then, when the server is down, i.e., when there is a token in place `down`, there are up to $K-1$ possible tokens in place `done`, thus making up another K different situations. In total, this yields $2K + 1$ states per level. As a consequence of this, the matrices $\mathbf{A}_i$ and $\mathbf{R}$ have dimension $(2K+1) \times (2K+1)$. Since these matrices in principle remain transparent to the modeller, this is not a problem at all, however, the construction of these matrices "by hand" would have been very impractical.

Studying the model reveals that the level number κ at which the repeating structure begins is 1. This implies that the boundary equations comprise a system of $(4K + 2)$ linear equations. Consequently, the state probabilities for levels 0 and 1 are computed directly, including their normalisation, after which state probabilities for higher levels can be computed recursively.

In Figure 11.14 we present the expected buffer occupancy $E[\texttt{buffer}]$ as a function of the checkpointing interval K for two values of the job arrival rate λ

	1	2	2
λ	5.0	6.5	8.0
μ	10	10	10
ϕ	0.13	0.13	0.25
σ	21.0	21.0	21.0
β	11.0	11.0	11.0

Table 11.4: Parameters used in the three scenarios

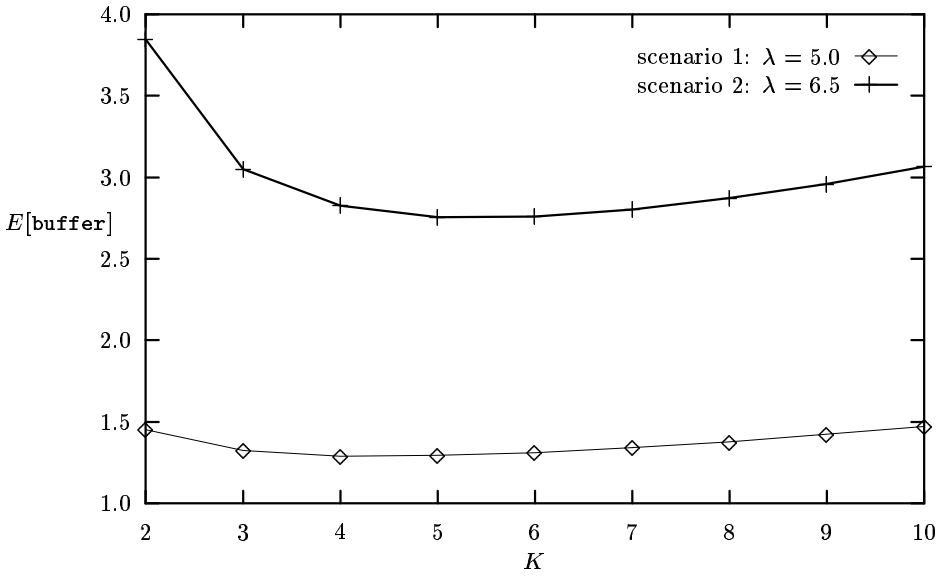


Figure 11.14: The expected buffer occupancy as a function of the checkpointing interval K

(we compare scenarios 1 and 2 here). As can be observed, the buffer occupancy is higher for a more heavily loaded system, for all K. Interestingly, the curves show a pronounced minimum for $K = 4$ (scenario 1) or $K = 5$ (scenario 2); a choice of K too small causes the system to make checkpoints too often (checkpoints take an amount of time that is independent of K), thus losing processing capacity. On the other hand, if only very few checkpoints are made, the recovery process, which requires the reprocessing of not-yet-checkpointed jobs plus the restart delay, will be longer. As such, this figure clearly illustrates the trade-off that exists in designing rollback/recovery schemes.

Then, in Figure 11.15, we present the percentage of time the server is making checkpoints ($E[\texttt{saving}]$), as well as the expected time the server is reserving unchecked jobs after a failure has occurred plus the expected restart time ($E[\texttt{down}]$), again for scenarios 1 and 2. The latter quantity is almost the same for both scenarios and therefore drawn as a single line; since failures occur almost randomly, the amount of unchecked jobs is almost the same in both cases, so that the percentage of time the server is actively recovering is also almost the same. The fact that the computed values for this probability are not exactly alike lies in the fact that the server can only fail when it is serving regular jobs or when it is idle. Since in scenario 2 the job arrival rate is larger than in scenario 1, the server is slightly more busy making checkpoints and therefore is slightly less failure prone. Therefore, in scenario 2 this probability is slightly smaller but the difference is at most 10^{-3}. The other two curves show the decreasing percentage of time the server is busy making checkpoints when K increases. As expected, in scenario 1 there is less load, so there is less checkpointing required.

We finally increased the job arrival rate to 8.0, as indicated in scenario 3. For this scenario we show two probabilities in Figure 11.16, again as a function of K. First notice that for $K = 2$ the system is not stable anymore. This can be understood as follows. The percentage of time needed for normal processing equals $\lambda/\mu = 8/10 = 80\%$. On top of that comes, for $K = 2$, half a checkpointing time per job, requiring another $4/21 \approx 19.04\%$ capacity of the server. Taking into account the non-zero probability of failure, and the extra work required when failures occur, it becomes clear that the server cannot do all the work when $K = 2$.

Let us now address the cases where the system is stable. The upper curve showes the probability that the buffer is non-empty. As can be seen, this probability ranges around 95%. Here we see similar behaviour as we have seen for the expected buffer occupancy. There is a pronounced minimum for $K = 6$. For $K < 6$, the server is making too many checkpoints. This extra work is not earned back by the shorter recovery time that results. On the other hand, for $K > 6$, the recovery times become larger so that the gain of less checkpointing overhead is lost. The lower curve shows the probability that the server is available for regular processing or for being idle, i.e., the probability that the server is not making checkpoints nor is recovering. Also here a choice of K too small or too large yields a loss in performance.

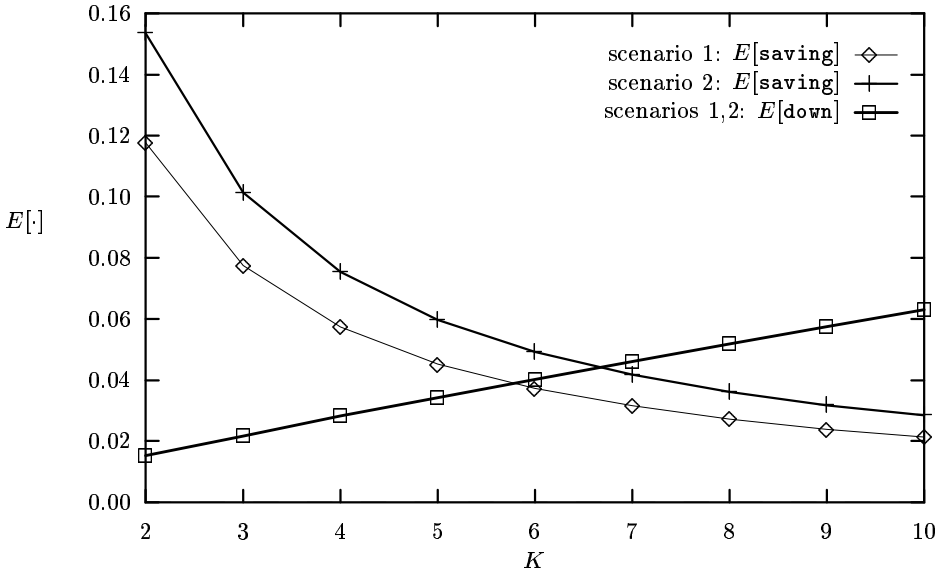


Figure 11.15: The percentage of time spent making checkpoints and the percentage of time recovering as a function of the checkpointing interval K

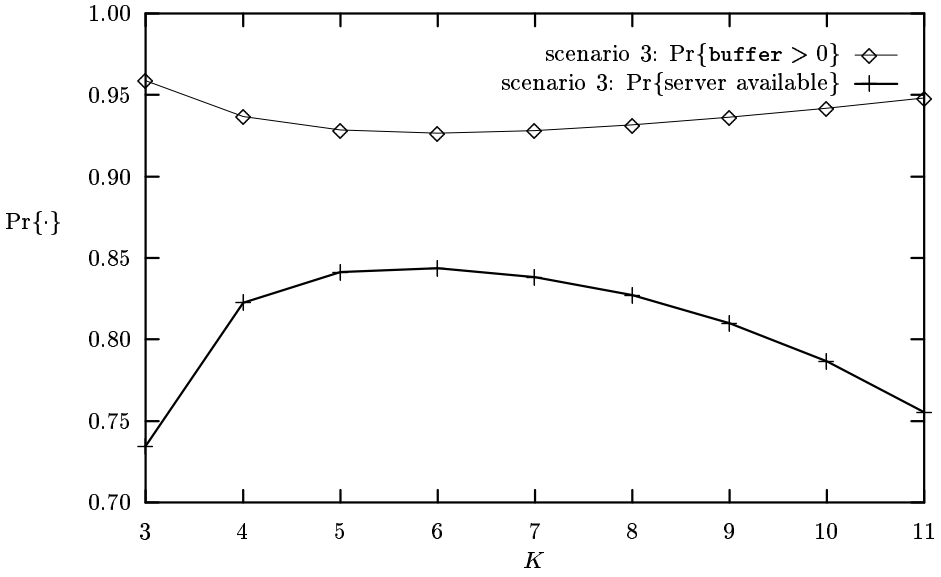


Figure 11.16: The probability of actual server availability and the probability of the buffer being non-empty as a function of the checkpointing interval K

11.8 Further reading

One-place unbounded SPNs have been described in the mid-1980's by Florin and Natkin [14, 15, 16]. Haverkort proposed the more general class of iSPNs (not yet named as such) in [20], after which a software tool, named SPN2MGM, supporting the construction and evaluation of iSPNs was reported in [24]. The M.Sc. students Klein [30] and Frank [17] contributed to the development of SPN2MGM. The OCDR mechanism was introduced by Heijenk [25] and evaluated by Heijenk and Haverkort [26]. Recently, Ost *et al.* extended the OCDR model to include non-exponential connection-setup and -release times [43].

Despite their great applicability and their numerically attractiveness, only a few books on model-based performance evaluation of computer and communication systems actually address matrix-geometric methods; most notably are [19, 28, 29, 37]. The tutorial paper by Nelson also provides a good introduction [36]. Haverkort discusses both matrix-geometric methods and iSPNs in [21]. Background information can be found in the books by Neuts [38, 39] and in many mathematical journals; a seminal paper has been written by Evans [12]. Matrix-geometric analyses of queues subject to complex arrival processes, such as Markov-modulated arrival processes, can be found in [13, 33]. Surveys on algorithms to solve for the matrix $\mathbf{R}$ can be found in [31, 34]. The logarithmic-reduction algorithm has been published by Latouche and Ramaswami [32]. The spectral expansion method has been advocated by Daigle and Lucantoni [10] and Mitrani *et al.* [35]; a comparison with the logarithmic reduction method has been performed by Haverkort and Ost [22]. Finite QBD models have been discussed by various authors as well. Gün and Makowski [18], Bocharov and Naoumov [3] and Wagner *et al.* [48] present a matrix-geometric solution. Chakka and Mitrani use the spectral expansion method also for these models [6] whereas Ye and Li recently proposed a new (and fast) folding algorithm [49].

Bibliography

[1] S.S. Berson and R.R. Muntz. *Detecting Block GI/M/1 and Block M/G/1 Matrices from Model Specifications.* Technical report, University of California, Los Angeles, 1994.

[2] A. Bobbio and L. Roberti. Distribution of the minimal completion time of parallel tasks in multi-reward semi-Markov models. *Performance Evaluation*, 14:239–256, 1992.

[3] P.P. Bocharov and V. Nauomov. Matrix-geometric stationary distribution for the PH/PH/1/r queue. *Elektronische Informationsverarbeitung und Kybernetik*, 22(4):179–186, 1986.

[4] P. Buchholz. Hierarchical Markovian models: Symmetries and reduction. In R. Pooley and J. Hillston, editors, *Computer Performance Evaluation '92: Modelling Techniques and Tools*, pages 305–319. Edinburgh University Press, 1992.

[5] P. Buchholz. Aggregation and reduction techniques for hierarchical GC-SPNs. In *Proceedings of the 5th International Workshop on Petri Nets and Performance Models*, pages 216–225. IEEE Computer Society Press, 1993.

[6] R. Chakka and I. Mitrani. Spectral expansion solution for a finite capacity multiserver system in a Markovian environment. In D.D. Kouvatsos, editor, *Proceedings of the 3rd International Workshop on Queueing Networks with Finite Capacity*, pages 6.1–6.9, 1995.

[7] K.M. Chandy. A survey of analytic models of rollback and recovery strategies. *IEEE Computer*, 8(5):40–47, 1975.

[8] P. Chimento and K.S. Trivedi. The completion time of programs on processors subject to failure and repair. *IEEE Transactions on Computers*, 42(10):1184–1194, 1993.

[9] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets and symmetric modelling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, 1993.

[10] J.N. Daigle and D.M. Lucantoni. Queueing systems having phase-dependent arrival and service rates. In W.J. Stewart, editor, *Numerical Solution of Markov Chains*, pages 161–202. Marcel Dekker Inc., 1991.

[11] S. Donatelli and M. Sereno. On the product-form solution for stochastic Petri nets. In K. Jensen, editor, *Application and Theory of Petri Nets 1992*, pages 154–172. Springer-Verlag, 1992.

[12] R.V. Evans. Geometric distribution in some two-dimensional queueing systems. *Operations Research*, 15:830–846, 1967.

[13] W. Fischer and K.S. Meier-Hellstern. The Markov-modulated Poisson process (MMPP) cookbook. *Performance Evaluation*, 18:149–171, 1992.

[14] G. Florin and S. Natkin. On open synchronized queueing networks. In *Proceedings of the 1st International Workshop on Timed Petri Nets*, pages 226–223. IEEE Computer Society Press, 1985.

[15] G. Florin and S. Natkin. One place unbounded stochastic Petri nets: Ergodicity criteria and steady-state solution. *Journal of Systems and Software*, 1(2):103–115, 1986.

[16] G. Florin and S. Natkin. A necessary and sufficient saturation condition for open synchronized queueing networks. In *Proceedings of the 2nd International Workshop on Petri Nets and Performance Models*, pages 4–13. IEEE Computer Society Press, 1987.

[17] C. Frank. *Bewertung von stochastischen Petrinetzen mit Hilfe der Matrix-geometrischen Methode.* Master's thesis, RWTH Aachen, 1997.

[18] L. Gün and A.M. Makowski. Matrix geometric solutions for finite capacity queues with phase-type distributions. In P.J. Courtois and G. Latouche, editors, *Proceedings Performance '87*, pages 269–282, North-Holland, 1987.

[19] P.G. Harrison and N.M. Patel. *Performance Modelling of Communication Networks and Computer Architectures.* Addison-Wesley, 1992.

[20] B. R. Haverkort. Matrix-geometric solution of infinite stochastic Petri nets. In *Proceedings of the 1st International Computer Performance and Dependability Symposium*, pages 72–81. IEEE Computer Society Press, 1995.

[21] B. R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach.* John Wiley & Sons, 1998.

[22] B. R. Haverkort and A. Ost. Steady-state analysis of infinite stochastic Petri nets: A comparison between the spectral expansion and the matrix-geometric method. In *Proceedings of the 7th International Workshop on Petri Nets and Performance Models*, pages 36–45. IEEE Computer Society Press, 1997.

[23] B.R. Haverkort. Approximate performability and dependability modelling using generalized stochastic Petri nets. *Performance Evaluation*, 18:61–78, 1993.

[24] B.R. Haverkort and I.G. Niemegeers. Performability modelling tools and techniques. *Performance Evaluation*, 25:17–40, 1996.

[25] G.J. Heijenk. *Connectionless Communications using the Asynchronous Transfer Mode.* PhD thesis, University of Twente, 1995.

[26] G.J. Heijenk and B.R. Haverkort. Design and evaluation of a connection management mechanism for an ATM-based connectionless service. *Distributed System Engineering Journal*, 3(1):53–67, 1996.

[27] W. Henderson and P.G. Taylor. Embedded processes in stochastic Petri nets. *IEEE Transactions on Software Engineering*, 17(2):108–116, 1991.

[28] K. Kant. *Introduction to Computer System Performance Evaluation.* McGraw-Hill, 1992.

[29] P.J.B. King. *Computer and Communication Systems Performance Modelling.* Prentice-Hall, 1990.

[30] E.F.J. Klein. *SPN2MGM: a tool for solving a class of infinite GSPN models.* Master's thesis, University of Twente, 1995.

[31] G. Latouche. Algorithms for infinite Markov chains with repeating columns. In C.D. Meyer, editor, *Linear algebra, Markov chains, and queueing models*, pages 231–265. Springer-Verlag, 1993.

[32] G. Latouche and V. Ramaswami. A logarithmic reduction algorithm for quasi birth and death processes. *Journal of Applied Probability*, 30:650–674, 1993.

[33] D.M. Lucantoni, K.S. Meier-Hellstern, and M.F. Neuts. A single-server queue with server vacations and a class of non-renewal arrival processes. *Advances in Applied Probability*, 22:676–705, 1990.

[34] D.M. Lucantoni and V. Ramaswami. Efficient algorithms for solving the non-linear matrix equations arising in phase-type queues. *Stochastic Models*, 1(1):29–51, 1985.

[35] I. Mitrani and R. Chakka. Spectral expansion solution of a class of Markov models: Application and comparison with the matrix-geometric method. *Performance Evaluation*, 23:241–260, 1995.

[36] R. Nelson. *Matrix Geometric Solutions in Markov Models: A Mathematical Tutorial.* Technical report, IBM Research Report RC 16777, 1991.

[37] R. Nelson. *Probability, stochastic processes, and queueing theory.* Springer-Verlag, 1995.

[38] M.F. Neuts. *Matrix Geometric Solutions in Stochastic Models: An Algorithmic Approach.* Johns Hopkins University Press, 1981.

[39] M.F. Neuts. *Structured Stochastic Matrices of M/G/1 Type and Their Applications.* Marcel Dekker, 1989.

[40] V.F. Nicola, V.G. Kulkarni, and K.S. Trivedi. Queueing analysis of fault-tolerant computer systems. *IEEE Transactions on Software Engineering*, 13(3):363–375, 1987.

[41] V.F. Nicola and J.M. van Spanje. Comparative analysis of different models of checkpointing and recovery. *IEEE Transactions on Software Engineering*, 16(8):807–821, 1990.

[42] R.O. Onvural. *Asynchronous Transfer Mode Networks: Performance Issues.* Artech House, 1994.

[43] A. Ost, C. Frank, and B.R. Haverkort. Untersuchungen zum Verbindungsmanagement bei Videoverkehr mit Matrix-geometrischen stochastischen Petrinetzen. In K. Irmscher, editor, *Proceedings of the 9th ITG/GI Workshop on Measurement, Modeling and Evaluation of Computer System Performance*, pages 71–85. VDE Verlag, 1997.

[44] A. Ranganathan and S.J. Upadhyaya. Performance evaluation of rollback-recovery techniques in computer programs. *IEEE Transactions on Reliability*, 42(2):220–226, 1993.

[45] H. Saito. *Teletraffic Technologies in ATM Networks.* Artech House, 1994.

[46] W.H. Sanders and J.F. Meyer. Reduced-base model construction for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.

[47] W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, 1994.

[48] D. Wagner, V. Nauomov, and U. Krieger. Analysis of a finite capacity multi-server delay-loss system with a general Markovian arrival process. In S.S. Alfa and S. Chakravarthy, editors, *Matrix-Analytic Methods in Stochastic Models.* Marcel Dekker, 1995.

[49] J. Ye and S. Li. Folding algorithm: A computational method for finite QBD processes with level-dependent transitions. *IEEE Transactions on Communications*, 42(2):625–639, 1994.

Part IV

Net-Driven Markov Chain Generation

Chapter 12

Tensor based methods in GSPN

In this chapter we describe an additional way to attack the state space explosion problem using the structure of the model: the method presented here enlarges of about one order of magnitude the size of the models that can be solved (in terms of the cardinality of the state space), by lowering the complexity of the analysis in terms of space as well as complexity.

In the context of exact solution of stochastic models, the *tensor algebra* approach allows to express the infinitesimal generator $\mathbf{Q}$ of an SPN in terms of $\mathbf{Q}_i$ matrices coming from some *components* (subnets of smaller state space) and to implement the solution of the characteristic steady-state solution equation $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$ without computing and storing $\mathbf{Q}$ [4, 15, 16, 6, 21]: this technique allows to move the storage bottleneck from the infinitesimal generator matrix to the probability vector. All the works for SPN cited above were inspired by the pioneering work of Plateau [28] on Stochastic Automata Networks. Since the tensor algebra method is based on components of the net, and therefore on the *structure* of the net, we shall often refer to it as a "structured solution method."

The approach is based on a construction of the state space in which a certain Cartesian product of reachable states of components is performed, leading eventually to a *product space* PS that includes the actual *reachability set* RS: in general, the product space can be much bigger than the actual state space. In other words, the PS may contain many non-reachable states (that we shall call *spurious*). According to this structured view of the state space the $\mathbf{Q}_i$ matrices can be derived. The probability vector is sized according to PS, but, even if there are spurious states, the tensor algebra approach leads to exact solution [28, 16], but the storage and computational complexity may be increased in practice to the point that the advantages of the technique are lost.

A way to overcome the creation of spurious states is to generate a *high level representation* of the full model that acts like a "filter" with respect to the spurious solutions. An abstract or high level model —called here *basic*

skeleton— constraints the product space PS, leading to a *restricted product space* RPS expressed as the union of the Cartesian product of subsets of the reachability sets of the components. The $\mathbf{Q}$ matrix has a block structure that reflects the high level view, and a tensor expression for each block of $\mathbf{Q}$ in terms of blocks of $\mathbf{Q}_i$ can be derived.

We refer to the product space method as *flat*, or *single level*, and to the restricted product space one as *abstract*, or *two levels*.

Two level techniques proved very effective in a number of cases, and, for certain net subclasses, like MG, no spurious states are generated, so that equality between RS and RPS is obtained [6]. In [9], the MG case is generalized to DSSP, where RPS may strictly include RS, but, in many of the cases the authors have experimented with, they coincide.

In summary, two beneficial effects are produced: the space complexity of the solution is lowered, moving the bottleneck from the infinitesimal generator to the probability vector, with possibly a gain also in time complexity (this depends from the structure of the model), moreover the existence of an expression for the infinitesimal generator provides insight on the relationships between the structure of a GSPN and that of its underlying CTMC.

Two classes of GSPN nets will be used in this chapter to illustrate different aspects of tensor based techniques: the plain method is presented using Superposed Generalized Petri Nets (SGSPN) [16], while the modification of the technique to include abstract view informations is exemplified by means of a particular structured view of the system here called *SAM view*, where SAM stands for System of Asynchronously communicating Modules.

An SGSPN is basically a set of GSPNs that interact through transition synchronization. A SAM view identifies instead a subset of places as *buffers*, and a number of disjoint subnets called *modules*; the only exchange of information among modules is through buffers, so that SAM are a natural model for asynchronously communicating systems. A SAM view of the net can be provided by the model construction process, or it can be defined only for solution purposes.

We therefore consider two modular construction patterns: *synchronous* composition for SGSPN and *asynchronous* for SAM. Although we use a synchronous composition class for the flat method, and asynchronous for the abstract one, following the literature, the modularity principle and the type of technique are completely orthogonal.

We use "modular" GSPN to explain the tensor based methods, and this modularity can be as fine as places (each place is a module) as it has been recently shown in [8] and [10], although the improvements in space and time may be lost in some cases: the method can therefore be applied to a general GSPN, but its efficacy in reducing the solution cost may be very low, or even result in a worsening of the problem.

The chapter is organized into three sections. Section 12.1 describes the main ideas behind the application of tensor algebra to GSPN solution. Moreover the notation and definitions specific to this chapter are introduced. Section 12.2 introduces instead the "flat" and the "abstract" approaches, the former being exemplified in Section 12.3 using SGSPN, while the latter is explained on the

SAM view case in Section 12.4. In both sections the tensor algebra expression for the reachability graph is derived first, followed by its modification to account for rates. Section 12.5 provides pointer to additional literature.

12.1 Basic principles and definitions

Tensor algebra is an algebra defined on matrices, with a product operator denoted by $\otimes$, and a sum operator denoted by $\oplus$. In the following definitions we consider matrices on real values.

Definition 12.1 *Let* $\mathbf{A}$ *be a* $n \times m$ *matrix (*$\mathbf{A} \in \mathbb{R}^{n \times m}$*), and* $\mathbf{B}$ *be a* $p \times q$ *one (*$\mathbf{B} \in \mathbb{R}^{p \times q}$*);* $\mathbf{C}$ *is the tensor (Kronecker) product of* $\mathbf{A}$ *and* $\mathbf{B}$ *and we write* $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ *iff* $\mathbf{C}$ *is a* $n \cdot p \times m \cdot q$ *matrix (*$\mathbf{C} \in \mathbb{R}^{n \cdot p \times m \cdot q}$*) defined by:*

$$\mathbf{C}_{i,j} = \mathbf{C}_{\bar{i}\bar{j}} = a_{i_1 j_1} b_{i_2 j_2}$$

with $\bar{i} = (i_1, i_2), \bar{j} = (j_1, j_2)$*, and* $i = (i_1) \cdot p \ + \ i_2$ *(similarly,* $j = (j_1) \cdot q \ + \ j_2$*).*

As a simple example consider the tensor product of a 2×2 matrix, with a 2×3. We have

$$\mathbf{A} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \qquad \mathbf{B} = \begin{pmatrix} b_{00} & b_{01} & b_{02} \\ b_{10} & b_{11} & b_{12} \end{pmatrix}$$

$$\mathbf{C} = \mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{00}b_{00} & a_{00}b_{01} & a_{00}b_{02} & a_{01}b_{00} & a_{01}b_{01} & a_{01}b_{02} \\ a_{00}b_{10} & a_{00}b_{11} & a_{00}b_{12} & a_{01}b_{10} & a_{01}b_{11} & a_{01}b_{12} \\ a_{10}b_{00} & a_{10}b_{01} & a_{10}b_{02} & a_{11}b_{00} & a_{11}b_{01} & a_{11}b_{02} \\ a_{10}b_{10} & a_{10}b_{11} & a_{10}b_{12} & a_{11}b_{10} & a_{11}b_{11} & a_{11}b_{12} \end{pmatrix}$$

The generalization to K matrices is immediate: Let $\mathbf{A} = \bigotimes_{k=1}^{K} \mathbf{A}^k$ denote the tensor product of K matrices $\mathbf{A}^k \in \mathbb{R}^{n_k \times m_k}$. Let $n_l^u = \prod_{k=l}^{u} n_k$, $n = n_1^K$, and $\bar{n}_k = n/n_k$. If we assume a $(n_1, \ldots, n_K)$ mixed-base numbering scheme, the tuple $\bar{i} = (i_1, \ldots, i_K)$ corresponds to the number $(\ldots((i_1)n_2 + i_2)n_3 \cdots)n_K + i_K$ A $(m_1, \ldots, m_K)$ mixed-base numbering scheme can be similarly be associated to $\bar{j}$. If we assume that $\bar{i}$ $(\bar{j})$ is the mixed-based representation of i and j, respectively, the generic element of $\mathbf{A} \in \mathbb{R}^{n \times n}$ is

$$a_{i,j} = a_{\bar{i},\bar{j}} = a^1_{i_1,j_1} \cdot a^2_{i_2,j_2} \cdots a^K_{i_K,j_K} \tag{12.1}$$

The tensor sum is defined for square matrices, in terms of the tensor product operator:

Definition 12.2 *Let* $\mathbf{A}$ *be a* $n \times n$ *matrix, (*$\mathbf{A} \in \mathbb{R}^{n \times n}$*) and* $\mathbf{B}$ *be a* $p \times p$ *one (*$\mathbf{B} \in \mathbb{R}^{p \times p}$*);* $\mathbf{D}$ *is the tensor (Kronecker) sum of* $\mathbf{A}$ *and* $\mathbf{B}$ *and we write* $\mathbf{D} = \mathbf{A} \oplus \mathbf{B}$ *iff* $\mathbf{D}$ *is a* $n \cdot p \times n \cdot p$ *matrix (*$\mathbf{D} \in \mathbb{R}^{n \cdot p \times n \cdot p}$*) defined by:*

$$\mathbf{D} = \mathbf{A} \oplus \mathbf{B} = \mathbf{A} \otimes \mathbf{I}_p + \mathbf{I}_n \otimes \mathbf{B}$$

where $\mathbf{I}_x$ *is the* $x \times x$ *identity matrix.*

Let's consider again the two matrices $\mathbf{A}$ and $\mathbf{B}$, where $\mathbf{A}$ is the same as before, and in $\mathbf{B}$ the last column has been deleted. The computation of their tensor sum is:

$$\mathbf{A} \otimes \mathbf{Id}_2 = \begin{pmatrix} a_{00} & 0 & a_{01} & 0 \\ 0 & a_{00} & 0 & a_{01} \\ a_{10} & 0 & a_{11} & 0 \\ 0 & a_{10} & 0 & a_{11} \end{pmatrix}$$

$$\mathbf{Id}_1 \otimes \mathbf{B} = \begin{pmatrix} b_{00} & b_{01} & 0 & 0 \\ b_{10} & b_{11} & 0 & 0 \\ 0 & 0 & b_{00} & b_{01} \\ 0 & 0 & b_{10} & b_{11} \end{pmatrix}$$

$$\mathbf{D} = \begin{pmatrix} a_{00}+b_{00} & b_{01} & a_{01} & 0 \\ b_{10} & a_{00}+b_{11} & 0 & a_{01} \\ a_{10} & 0 & a_{11}+b_{00} & b_{01} \\ 0 & a_{10} & b_{10} & a_{11}+b_{11} \end{pmatrix}$$

The generalization to K matrices is straightforward. Let $\mathbf{A} = \bigoplus k = 1^K \mathbf{A}^i$ denote the tensor sum of K matrices $\mathbf{A}^k \in \mathbb{R}^{n_k \times m_k}$:

$$\bigoplus_{k=1}^{K} \mathbf{A}^k = \sum_{k=1}^{K} \mathbf{I}_{n_1} \otimes \cdots \otimes \mathbf{I}_{n_{k-1}} \otimes \mathbf{A}^k \otimes \mathbf{I}_{n_{k+1}} \otimes \cdots \otimes \mathbf{I}_{n_K} = \sum_{k=1}^{K} \mathbf{I}_{n_1^{k-1}} \otimes \mathbf{A}^k \otimes \mathbf{I}_{n_{k+1}^K}.$$

12.1.1 Tensor operators and independent composition of Markov chains

Let us consider two separated models $\mathcal{M}_1$ and $\mathcal{M}_2$, specified as either Markov chains or GSPNs and assume that we know the set of reachable states of the two, $\mathrm{RS}(\mathcal{M}_1)$ and $\mathrm{RS}(\mathcal{M}_2)$. Assuming that the two are finite, of cardinality n_1 and n_2 respectively, it is a straightforward task to associate a number from 1 to n_1 (n_2) to each state of the first (second) model.

If we now consider the global model $\mathcal{M}$ obtained by the parallel and independent composition of the two models, then a global state is a pair $\bar{\imath} = (i_1, i_2)$, and, if n_1 and n_2 are finite, it is straightforward to associate to $\bar{\imath}$ the number $i = i_1 \cdot n_2 + i_2$.

In general, if we have K models $\mathcal{M}_i$, each of state space $\mathrm{RS}(\mathcal{M}_i)$, of n_i states each, then the state space of the model $\mathcal{M}$, obtained by the independent parallel composition of the K models, is the Cartesian product of the state spaces:

$$\mathrm{RS}(\mathcal{M}) = \mathrm{RS}(\mathcal{M}_1) \times \ldots \mathrm{RS}(\mathcal{M}_K)$$

and the number of states is:

$$n = |\mathrm{RS}(\mathcal{M})| = \prod_{k=0}^{K-1} n_k$$

states. Each state $\bar{\imath} = (i_1, \ldots, i_K)$ of the global model can be identified by the number $i = (\ldots((i_1)n_2 + i_2)n_3 \cdots)n_K + i_K$, so that we shall indicate a state either as the number i or as the vector $\bar{\imath}$ of size K.

In case of square matrices, if $\mathbf{A}$ and $\mathbf{B}$ are interpreted as state transition matrices of two discrete time Markov chains, it is immediate to recognize (see Davio in [14]) that $\mathbf{C}$ is the transition probabilities matrix of the process obtained as independent composition of the two original processes. Indeed, for the global system to move from $\bar{\imath} = (i_1, \ldots, i_K)$ to $\bar{\jmath} = (j_1, \ldots, j_K)$ each component k has to move from i_k to j_k at the same time step, and this indeed happens with a probability that is the product of the probabilities of the local moves.

Again if we consider $\mathbf{A}$ and $\mathbf{B}$ as the infinitesimal generator of two time continuous Markovian processes, then $\mathbf{D}$ is the infinitesimal generator of the process obtained by independent composition of the two original one. Indeed the behaviour of the independent composition of 2 CTMC is the sum of two terms: the first CTMC moves while, at the same time, the second one does not move $(\mathbf{A} \otimes \mathbf{I}_p)$ or viceversa $(\mathbf{I}_n \otimes \mathbf{B})$.

Observe that a system obtained by the independent composition of K CTMCs, can move from a state $\bar{\imath} = (i_1, \ldots, i_K)$ to a state $\bar{\jmath} = (j_1, \ldots, j_K)$ only if it exists one and only one index k such that $i_k \neq j_k$, since we are in a continuous time environment, and indeed the tensor sum produces a matrix in which all transition rates among states that differ by more than a single component are set equal to zero.

An aspect that deserves attention is that even if the infinitesimal generator of the global system is rewritten using only the infinitesimal generators of the component systems, nevertheless an evaluation of the expression leads to a matrix of size equal to the product of the size of the components. The interesting point is that the expression need not to be evaluated explicitly: indeed each time that an element a_{ij} of $\mathbf{A}$ is used, we only need to compute its value according to the tensor product definition, which requires $K - 1$ additional multiplications.

A more efficient method has been explained in [14]. Since the tensor sum is defined in terms of tensor product, we explain how we can avoid the construction of the complete matrix in the tensor product case, for the general situation of K matrices $\mathbf{A}^k$, of size $n_k \times n_k$. Following the method proposed by Plateau, the $\mathbf{A}^k$ matrices are considered sequentially, one at a time, exploiting the equality [14]:

$$\mathbf{D} = \bigotimes_{k=1}^{K} \mathbf{A}^k = \prod_{k=1}^{K} \mathbf{S}_{(n_1 \cdots n_k, n_{k+1} \cdots n_K)} \cdot (\mathbf{I}_{\bar{n}_k} \otimes \mathbf{A}^k) \cdot \mathbf{S}^T_{(n_1 \cdots n_k, n_{k+1} \cdots n_K)} \qquad (12.2)$$

where $\bar{n}_k = n_1 \cdot \ldots n_{k-1} \cdot n_{k+1} \cdot \ldots \cdot n_K$, and $\mathbf{S}_{(a,b)} \in \{0,1\}^{a \cdot b \times a \cdot b}$ is the matrix describing an (a, b) perfect shuffle permutation:

$$(\mathbf{S}_{(a,b)})_{i,j} = \begin{cases} 1 & \text{if } j = (i \bmod a) \cdot b + (i \operatorname{div} a) \\ 0 & \text{otherwise} \end{cases} .$$

Therefore, a vector-matrix multiplication can be performed using K vector permutations and K multiplications of the type $\mathbf{x} \cdot (\mathbf{I}_{\bar{n}_k} \otimes \mathbf{A}^k)$. Matrix $\mathbf{I}_{\bar{n}_k} \otimes \mathbf{A}^k$

has a peculiar structure: it is simply matrix $\mathbf{A}^k$ repeated $\bar{n}_k$ times over the diagonal, hence it is not necessary to store it to compute the vector-matrix multiplication. The cost of the k-th multiplication is $O(\bar{n}_k \cdot \eta[\mathbf{A}^k])$, where $\eta[\mathbf{A}^k]$ is the number of non-zero entries of $\mathbf{A}^k$, so that the total cost of $\mathbf{x} \cdot \mathbf{A}$ is

$$O\left(n \cdot \sum_{k=1}^{K} \frac{\eta[\mathbf{A}^k]}{n_k}\right)$$

instead of $O(\eta[A])$, that is the cost of the multiplication when $\mathbf{A}$ is stored in sparse form. If $\mathbf{A} = \bigotimes_{k=1}^{K} \mathbf{A}^k$, then $\eta[A] = \prod_{k=1}^{K} \eta[\mathbf{A}^k]$, and the multiplication based on shuffle is advantageous if the matrix is not too sparse.

12.1.2 Tensor operators and dependent composition

The above observations are not very interesting from an applicative point of view since usually the stochastic processes involved in an application are far from being independent.

Let us introduce a very simple example to explain how the technique can be applied also in case of dependencies. We consider two CTMS $\mathcal{S}_1$ and $\mathcal{S}_2$ defined by the following matrices:

$$\mathbf{Q}_1 = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & -\mu & \mu \\ 1 & \nu & -\nu \end{array} \qquad \mathbf{Q}_2 = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & -\mu & \mu \\ 1 & \lambda & -\lambda \end{array}$$

Let us consider the CTMC $\mathcal{S}$ resulting from the composition of $\mathcal{S}_1$ and $\mathcal{S}_2$ when we impose that a transition from state 0 to state 1 in $\mathcal{S}_1$ can happen only if, simultaneously, $\mathcal{S}_2$ moves from state 0 to state 1, and viceversa. $\mathcal{S}$ has four states, numbered from 0 to 3, but that we shall indicate below as pairs of states local to $\mathcal{S}_1$ and $\mathcal{S}_2$. Matrix $\mathbf{Q}$ of the CTMC $\mathcal{S}$ is given by

$$\mathbf{Q} = \begin{array}{c|cccc} & 0,0 & 0,1 & 1,0 & 1,1 \\ \hline 0,0 & -\mu & 0 & 0 & \mu \\ 0,1 & \lambda & -\lambda & 0 & 0 \\ 1,0 & \nu & 0 & -\nu & 0 \\ 1,1 & 0 & \nu & \lambda & -\lambda-\nu \end{array}$$

If we delete from $\mathbf{Q}_1$ and $\mathbf{Q}_2$ the elements that concern the non independent behavior of the two CTMCs, that is to say, if we delete from $\mathbf{Q}_1$ and $\mathbf{Q}_2$ the elements that represent the firing of the simultaneous event t_2, we obtain the two following matrices $\mathbf{Q}_1'$ and $\mathbf{Q}_2'$.

$$\mathbf{Q}_1' = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & \nu & -\nu \end{array} \qquad \mathbf{Q}_2' = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 0 \\ 1 & \lambda & -\lambda \end{array}$$

The composition of the independent behavior of $\mathcal{S}_1$ and $\mathcal{S}_2$ leads to

$$\mathbf{Q}' = \mathbf{Q}'_1 \oplus \mathbf{Q}'_2 = \begin{array}{c|cccc} & 0,1 & 0,1 & 1,0 & 1,1 \\ \hline 0,0 & 0 & 0 & 0 & 0 \\ 0,1 & \lambda & -\lambda & 0 & 0 \\ 1,0 & \nu & 0 & -\nu & 0 \\ 1,1 & 0 & \nu & \lambda & (-\lambda-\nu) \end{array}$$

The resulting matrix $\mathbf{Q}'$ differs from $\mathbf{Q}$ only in the first row, because the first row corresponds to the state in which there is a simultaneous change of state in the two components. We can rewrite $\mathbf{Q}$ as

$$\mathbf{Q} = \mathbf{Q}' + \mathbf{K}$$

where $\mathbf{K}$ is an appropriate "correcting factor" (correcting matrices). If we want to express also the correcting factor as Kronecker expression of matrices which are related to the single components, then we can observe that

$$\mathbf{K} = \mu \cdot \widehat{\mathbf{K}} = \begin{array}{c|cccc} & 0,0 & 0,1 & 1,0 & 1,1 \\ \hline 0,0 & -\mu & 0 & 0 & \mu \\ 0,1 & 0 & 0 & 0 & 0 \\ 1,0 & 0 & 0 & 0 & 0 \\ 1,1 & 0 & 0 & 0 & 0 \end{array}$$

where $\widehat{\mathbf{K}}$ can be rewritten as $-\mathbf{K}_1 \otimes \mathbf{K}_2 + \mathbf{K}_3 \otimes \mathbf{K}_4$. Please note the use of the tensor product, an operator which is usually associated to discrete processes. This is because the simultaneous event causes a change of state in all the CTMCs involved in the synchronization (called "simultaneous jump" in [29]).

Finally, we can express $\mathbf{K}_1, \mathbf{K}_2, \mathbf{K}_3, \mathbf{K}_4$ as

$$\mathbf{K}_1 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \qquad \mathbf{K}_2 = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\mathbf{K}_3 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \qquad \mathbf{K}_4 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$$

The product $\mathbf{K}_1 \otimes \mathbf{K}_2$ is the correcting factor for the diagonal elements, while $\mathbf{K}_3 \otimes \mathbf{K}_4$ inserts the effects of the synchronized transition.

To avoid to have an expression also for the diagonal, that can be precomputed and stored in memory, certain authors prefer to write a tensor expression only for the rate matrix $\mathbf{R}$, obtained from the infinitesimal generator by deleting the diagonal elements:

$$\mathbf{Q} = \mathbf{R} - \mathbf{\Delta} \tag{12.3}$$

where $\mathbf{\Delta}$ is a diagonal matrix and $\mathbf{\Delta}[i,i] = \sum_{k \neq i} \mathbf{Q}[i,k]$. Since the expression for $\mathbf{R}$ is simpler to read and write that the one for $\mathbf{Q}$, we shall only concentrate on $\mathbf{R}$ from now on.

In the case of independent composition, the state space of the composed system is obviously the Cartesian product of the state spaces of the component systems. Indeed also in this example we have that:

$$\mathrm{RS}(\mathcal{S}) \;=\; \mathrm{RS}(\mathcal{S}_1) \times \mathrm{RS}(\mathcal{S}_2)$$

but this is not always the case, for example if also the state transition from 1 to 0 in $\mathcal{S}_1$ must happen at the same time as the state transition from 1 to 0 in $\mathcal{S}_2$, then the resulting CTMC has only 2 states against the 4 of the Cartesian product. In general we can say that, if there are dependencies, the Cartesian product of the state spaces of the components is a superset of the state space of the composed system. This has "annoying" consequences from a storage point of view as will be discussed in the following section.

12.2 Tensor operators and GSPNs through an example

In general the distance between RS and the PS can be such that all the storage saving due to the use of a tensor expression instead than the explicit storage of $\mathbf{R}$ may be lost due to the size of the probability vector. We shall discuss this problem, and possible solutions, first with an example.

Flat approach: Figure 12.1 shows a GSPN $\mathcal{S}$ that can be considered as the composition of two GSPNs $\mathcal{S}_1$ and $\mathcal{S}_2$ over three common transitions $T1, T2$ and $T3$. Places whose names start with letter a define component $\mathcal{S}_1$, and those starting with b define $\mathcal{S}_2$. We assume that there is a sequence of n places and transitions between $b21$ and $b2n$, and of m places and transitions between $a31$ and $a3m$. $\mathcal{S}_1$ has therefore $m+2$ states, $\mathcal{S}_2$ has $n+2$. For brevity, we shall write RS for $\mathrm{RS}(\mathcal{S})$, RS_1 for $\mathrm{RS}(\mathcal{S}_1)$, and RS_2 for $\mathrm{RS}(\mathcal{S}_2)$. A product state space PS can then be defined as

$$\mathrm{PS} = \mathrm{RS}_1 \times \mathrm{RS}_2$$

and it is straightforward to observe that $\mathrm{RS} \subseteq \mathrm{PS}$. Note that PS has $(m+2) \cdot (n+2)$ states, but the reachability set of $\mathcal{S}$ has only $m+n+1$.

According to the techniques presented in [16], and that shall be discussed more extensively in the next section, the following matrix $\mathbf{F}$ of size $|\mathrm{PS}| \times |\mathrm{PS}|$ can be constructed:

$$\mathbf{F} \;=\; \hat{\mathbf{B}}_1 \oplus \hat{\mathbf{B}}_2 \;+\; \sum_{t \in \{T1,T2,T3\}} [\mathbf{B}_1(t) \otimes \mathbf{B}_2(t)]$$

where the $\hat{\mathbf{B}}_i$ and $\mathbf{B}_i(t)$ are $|\mathrm{RS}_i| \times |\mathrm{RS}_i|$ matrices (for $i \in \{1,2\}$), that can all be derived from matrix $\mathbf{B}_i$, that describes the reachability graph $\mathrm{TRG}(\mathbf{m_{0}}_i)$ ($\mathbf{B}_i(t)$ being the contribution of transition t to matrix $\mathbf{B}_i$, and $\hat{\mathbf{B}}_i$ being the contribution of all transitions but $\{T1, T2, T3\}$).

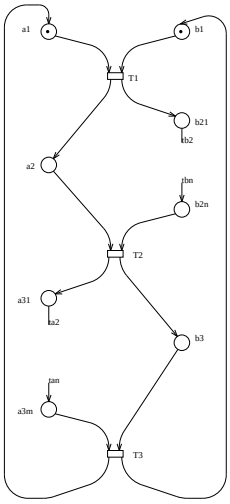


Figure 12.1: Explaining abstract views.

It is possible to show that $\mathbf{F}$ is a supermatrix of $\mathbf{B}$ ($\mathbf{B}$ can be derived from F by cancelling rows and columns corresponding to spurious states).

A similar technique can be applied for the definition of the rate matrix $\mathbf{R}$, by building a matrix $\mathbf{G}$ of size $|\mathrm{PS}| \times |\mathrm{PS}|$:

$$\mathbf{G} \;=\; \hat{\mathbf{R}}_1 \oplus \hat{\mathbf{R}}_2 \;+\; \sum_{t \in \{T1,T2,T3\}} w(t)[\mathbf{B}_1(t) \otimes \mathbf{B}_2(t)]$$

where the $\hat{\mathbf{R}}_i$ (for $i \in \{1,2\}$) are $|\mathrm{RS}_i| \times |\mathrm{RS}_i|$ matrices, that can all be derived from the infinitesimal generator $\mathbf{R}_i$ of $\mathcal{S}_i$, by considering contributions of all transitions but $\{T1, T2, T3\}$. It is possible to show that $\mathbf{G}$ is a supermatrix of $\mathbf{R}$ ($\mathbf{R}$ can be derived from $\mathbf{G}$ by cancelling rows and columns corresponding to spurious states), and that can be used for the correct computation of the steady state probability vector.

Two-levels approach. To limit the number of spurious states in the considered example, an abstract description of the system can be used to appropriately pre-select the subsets of the states that should enter in the Cartesian product.

For example, we can consider the net $\mathcal{S}_a$ of Figure 12.2 as an abstract representation of the one in Figure 12.1, with place $B2$ "summarizing" the structure of places $b21, \ldots, b2n$ and $A3$ "summarizing" the structure of places $a31, \ldots, a3m$. $\mathcal{S}_a$ has three reachable states: $\mathbf{z}_1 = (\mathsf{a1}, \mathsf{b1})$, $\mathbf{z}_2 = (\mathsf{a2}, \mathsf{B2})$, and $\mathbf{z}_3 = (\mathsf{A3}, \mathsf{b3})$. The states of $\mathcal{S}_1$ and $\mathcal{S}_2$ can be partitioned according to the states of $\mathcal{S}_a$: the $m+2$ states of $\mathcal{S}_1$ are partitioned in three equivalence classes: $\mathrm{RS}_{\mathbf{z}_1}(\mathcal{S}_1) = \{\mathsf{a1}\}$, $\mathrm{RS}_{\mathbf{z}_2}(\mathcal{S}_1) = \{\mathsf{a2}\}$, and $\mathrm{RS}_{\mathbf{z}_3}(\mathcal{S}_1) = \{\mathsf{a31}, \ldots, \mathsf{a3n}\}$. Similarly, for $\mathcal{S}_2$ we get: $\mathrm{RS}_{\mathbf{z}_1}(\mathcal{S}_2) = \{\mathsf{b1}\}$, $\mathrm{RS}_{\mathbf{z}_2}(\mathcal{S}_2) = \{\mathsf{b21}, \ldots, \mathsf{b2n}\}$, and $\mathrm{RS}_{\mathbf{z}_3}(\mathcal{S}_2) = \{\mathsf{b3}\}$.

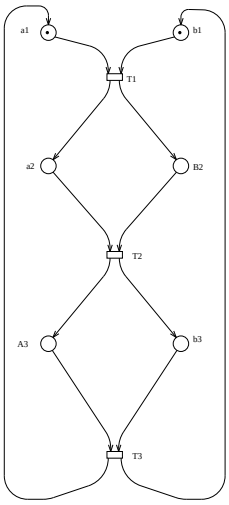


Figure 12.2: Explaining abstract views.

The restricted product state space RPS can then be built as:

$$\mathrm{RPS} = \biguplus_{\mathbf{z}\in \mathrm{RS}(\mathcal{S}_a)} \mathrm{RS}_{\mathbf{z}}(\mathcal{S}_1) \times \mathrm{RS}_{\mathbf{z}}(\mathcal{S}_2) =$$

$$\{\mathtt{a1}\} \times \{\mathtt{b1}\} \quad \cup \quad \{\mathtt{a2}\} \times \{\mathtt{b21}, \ldots, \mathtt{b2n}\} \quad \cup \quad \{\mathtt{a31}, \ldots, \mathtt{a3n}\} \times \{\mathtt{b3}\}$$

where $\biguplus$ is the *disjoint* set union. Note that in this case we obtain a precise characterization of the state space, but the union of Cartesian products can in general produce a superset of the reachable state space, depending on how precise is the abstract representation. We refer to methods based on a construction of a restricted state space as *abstract*, or *two levels*.

Since the state space is no longer the Cartesian product of sets of local state spaces, but the union of Cartesian products, we cannot expect to have for the infinitesimal generator a tensor expression as simple as before: we can build a matrix $\mathbf{G}$, supermatrix of the rate matrix $\mathbf{R}$, of size $|\mathrm{RPS}| \times |\mathrm{RPS}|$, and then consider it as block structured according to the states of $\mathcal{S}_a$. Each block will thus refer to a set of states obtained by a Cartesian product, and a tensor expression for each block can be derived. A similar method works for the tangible reachability graph matrix.

12.3 The flat approach: the SGSPN case

Informally a SGSPN is a set of GSPNs that synchronize on a common subset of timed transitions of equal rate.

Given a GSPN $\mathcal{S}$, let $\mathrm{TE}_{\mathcal{S}}$ denotes timed transitions and $\mathrm{TI}_{\mathcal{S}}$ denotes immediate ones (the index being omitted if no ambiguity may arise).

An SGSPN is a GSPN in which we identify a partition of the set of places P, and a set of timed transitions TS $\subseteq$ TE.

Definition 12.3 *A GSPN system,* $\mathcal{S} = \langle P_1 \cup \ldots \cup P_K, T_1 \cup \ldots \cup T_K \cup \text{TS}, \mathbf{pri}, \mathbf{Pre}, \mathbf{Post}, \mathbf{w}, \mathbf{m_0}, \Pi \rangle$ *is a* Superposed Generalized Stochastic Petri Net, *or, simply, an SGSPN, if:*

1. TS $\subseteq$ TE
2. $P_i \cap P_j = \emptyset$, $\forall i, j \in \{1, \ldots, K\}, i \neq j$;
3. $T_i \cap T_j = \emptyset$, $\forall i, j \in \{1, \ldots, K\}, i \neq j$;
4. $T_i \cap \text{TS} = \emptyset$, $\forall i \in \{1, \ldots, K\}$;
5. $\mathbf{Pre}[P_i, T_j] = \mathbf{Post}[P_i, T_j] = \mathbf{0}$, $\forall i, j \in \{1, \ldots, K\}, i \neq j$.

$\mathcal{S}_i = \langle \mathcal{N}_i, \mathbf{m_{0}}_i \rangle = \langle P_i, {}^\bullet P_i \cup P_i{}^\bullet, \mathbf{pri}_i, \mathbf{Pre}_i, \mathbf{Post}_i, \mathbf{w}_i, \mathbf{m_{0}}_i \rangle$ *are the* components *of* $\mathcal{S}$ *(where* $\mathbf{Pre}_i$, $\mathbf{Post}_i$, $\mathbf{w}_i$, *and* $\mathbf{m_{0}}_i$ *are the restrictions of* $\mathbf{Pre}$, $\mathbf{Post}$, $\mathbf{w}$, *and* $\mathbf{m_0}$ *to* P_i *and* $T_i \cup$ TS*). Transitions* TS *are called* synchronizing *transitions; transitions not in* TS *are called* internal, *or* local.

A natural way to obtain an SGSPN is the parallel composition of a set of GSPNs over transitions of equal label and equal rate, the set of new transitions obtained by the composition being TS.

Fig. 12.3 with TS $= \{T_1, T_2\}$, and with the partition represented by the dotted lines, shows an example of SGSPN that is the composition of three GSPNs. Each GSPN can be considered as a model of a process: the upper left is a while statement nested into an infinite loop, the body of the inner loop is an if statement that leads to synchronization with either the upper right GSPN or with the bottom one. The other two processes are replicas of the same GSPN: each can be considered as a model of a process that executes in an infinite loop a for statement controlled by a variable initially set to the value 10. The body of the for activates four processes in parallel with a co-begin/co-end scheme, and one of the four synchronize with the first process.

12.3.1 Solution method

Let us assume that each component has a finite state space, that is known, and that we indicate with RS_i. Let $\mathbf{B}$ be the matrix that describes the TRG($\mathcal{S}$) ($\mathbf{B}_{i,j} = 1$ iff $\exists t \in T : i[t\rangle j$), and $\mathbf{B}_i$ be the matrix that describes the TRG($\mathcal{S}_i$), Each $\mathbf{B}_i$ that be rewritten in terms of the contribution of each single transition:

$$\mathbf{B}_i = \sum_{t \in T_i \cup \text{TS}} \mathbf{B}_i(t)$$

that can be rewritten, if we define $\hat{\mathbf{B}}_i = \sum_{t \in T_i} \mathbf{B}_i(t)$, as

$$\mathbf{B}_i = \hat{\mathbf{B}}_i + \sum_{t \in \text{TS}} \mathbf{B}_i(t).$$

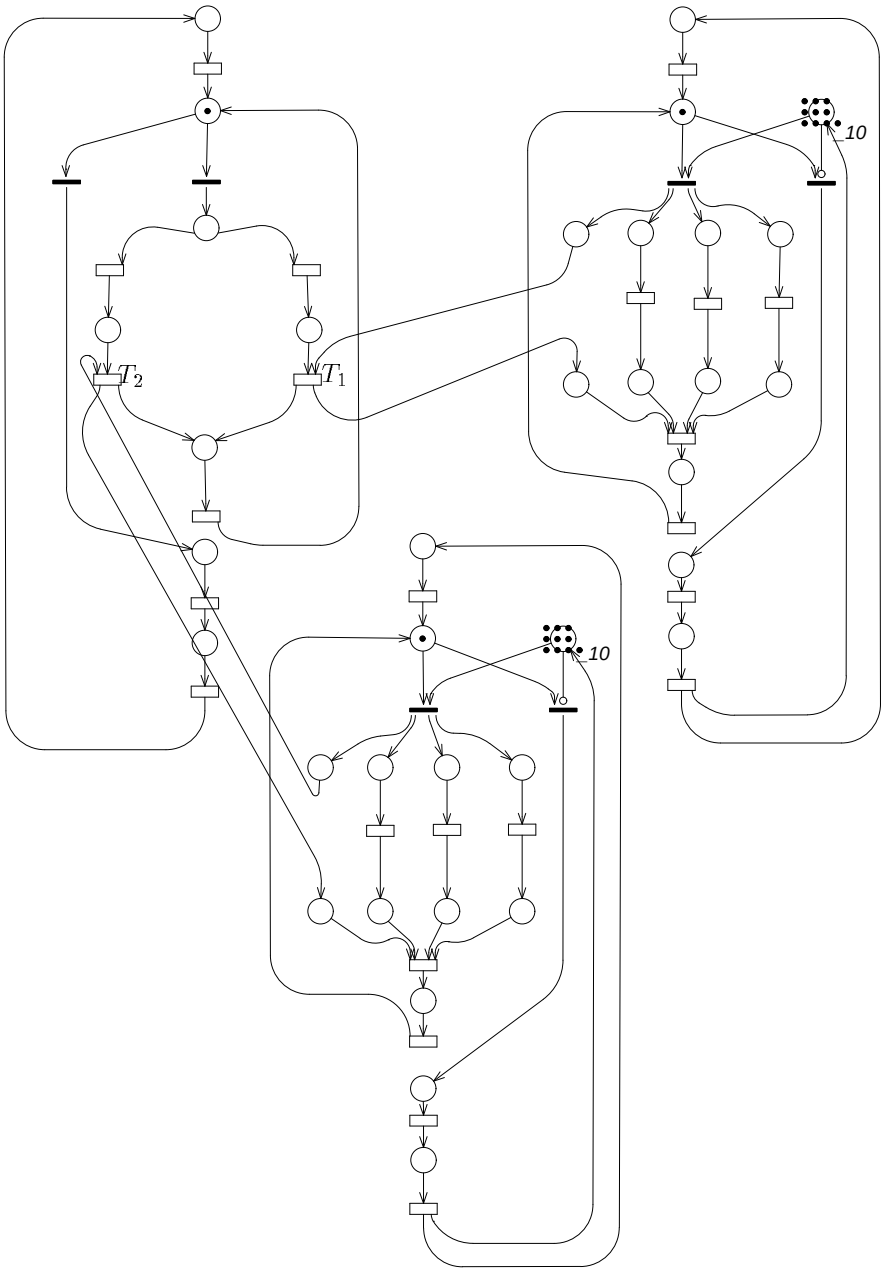


Figure 12.3: A example of SGSPN

Theorem 12.4 *The following formula expresses a supermatrix* $\mathbf{F}$ *of the reachability graph matrix* $\mathbf{B}$ *of a* K *component SGSPN* $\mathcal{S}$*:*

$$\mathbf{F} = \bigoplus_{i=1}^{K} \hat{\mathbf{B}}_i + \sum_{t \in \mathrm{TS}} [\bigotimes_{i=1}^{K} \mathbf{B}_i(t)]$$

where we assume that $\mathbf{B}_i(t) = \mathbf{I}$ *for all* $t : t \notin \mathrm{TS} \cap \{P_i^\bullet \cup {}^\bullet P_i\}$ *(if* t *is not a transition of* $\mathcal{S}_i$*.*
Moreover

$$\forall\ \bar{i} \in \mathrm{RS}(\mathcal{S}) \text{ and } \forall\ \bar{j} \in \mathrm{RPS}(\mathcal{S}) \setminus \mathrm{RS}(\mathcal{S}) : \quad \mathbf{F}_{\bar{i},\bar{j}} = 0.$$

Proof:
We have to prove that

1. $\forall\ \bar{i},\ \bar{j} \in \mathrm{RS}(\mathcal{S}),\ \mathbf{B}_{\bar{i},\bar{j}} = 1$ iff $\mathbf{F}_{\bar{i},\bar{j}} = 1$
2. $\forall\ \bar{i} \in \mathrm{RS}(\mathcal{S})$ and $\forall\ \bar{j} \in \mathrm{RPS}(\mathcal{S}) \setminus \mathrm{RS}(\mathcal{S}) : \quad \mathbf{F}_{\bar{i},\bar{j}} = 0$

Let us start with the first point, that is to say, we assume that $\bar{i}$ and $\bar{j}$ are reachable. We first prove that $\mathbf{B}_{\bar{i},\bar{j}} = 1 \longrightarrow \mathbf{F}_{\bar{i},\bar{j}} = 1$. Let t be the transition such that $\bar{i}[t\rangle\bar{j}$. We distinguish two cases, depending on whether t is a synchronizing transition or not.

If $t \in \mathrm{TS}$, then $\mathbf{B}_{\bar{i},\bar{j}} = 1$ implies that, in all components, $i_k[t\rangle j_k$, and therefore $(\mathbf{B}_k(t))_{i_k,j_k} = 1$, which implies that $(\bigotimes_{i=1}^{K} \mathbf{B}_i(t))_{\bar{i},\bar{j}} = 1$, by definition of tensor product, and, consequently, that $\mathbf{F}_{\bar{i},\bar{j}} = 1$.

If $t \notin \mathrm{TS}$, and $\mathbf{B}_{\bar{i},\bar{j}} = 1$ then there exists exactly one index k such that $i_k \neq j_k$, while, $\forall\ h \neq k,\ i_h = j_h$. As a consequence $(\hat{\mathbf{B}}_k)_{i_k,j_k} = 1$ and this is enough to ensure that $(\bigoplus_{i=1}^{K} \hat{\mathbf{B}}_i)_{\bar{i},\bar{j}} = 1$, and, consequently, that $\mathbf{F}_{\bar{i},\bar{j}} = 1$

We now prove the reverse implication: $\mathbf{F}_{\bar{i},\bar{j}} = 1 \longrightarrow \mathbf{B}_{\bar{i},\bar{j}} = 1$. Again, we distinguish two cases, depending on whether t is a synchronizing transition or not.

If $t \in \mathrm{TS}$, then $\mathbf{F}_{\bar{i},\bar{j}} = 1$ implies that $(\bigotimes_{i=1}^{K} \mathbf{B}_i(t))_{\bar{i},\bar{j}} = 1)$, which implies that, in each component k, $(\mathbf{B}_k(t))_{i_k,j_k} = 1$. By definition of synchronizing transition $\bar{i}[t\rangle\bar{j}$, and therefore $\mathbf{B}_{\bar{i},\bar{j}} = 1$.

If $t \notin \mathrm{TS}$, then $\mathbf{F}_{\bar{i},\bar{j}} = 1$ implies that $(\bigoplus_{i=1}^{K} \hat{\mathbf{B}}_i)_{\bar{i},\bar{j}} = 1$, but, by definition of tensor sum, this implies that there exists exactly one index k such that $i_k \neq j_k$, and that $(\hat{\mathbf{B}}_k)_{i_k,j_k} = 1$, but this is indeed the notion of $\bar{j}$ being reachable from $\bar{i}$ by firing t, and therefore $\mathbf{B}_{\bar{i},\bar{j}} = 1$.

Let us consider the second point: $\bar{i}$ is reachable and $\bar{j}$ is not reachable. The proof is by contradiction, assuming that $\mathbf{F}_{\bar{i},\bar{j}} = 1$. But for $\mathbf{F}_{\bar{i},\bar{j}} = 1$, then either $(\bigotimes_{i=1}^{K} \mathbf{B}_i(t))_{\bar{i},\bar{j}} = 1)$ or $(\bigoplus_{i=1}^{K} \hat{\mathbf{B}}_i)_{\bar{i},\bar{j}} = 1$, and we have already shown that each one of the two above conditions implies that $\bar{j}$ is reachable from $\bar{i}$, which contradicts the hypothesis of $\bar{j} \notin \mathrm{RS}(\mathcal{S})$. ♢

A similar result can be derived for the rate matrix, if we define $\hat{\mathbf{R}}_i = \sum_{t \in T_i} \mathbf{w}(t)\mathbf{B}_i(t)$.

Theorem 12.5 *The following formula expresses a supermatrix* $\mathbf{G}$ *of the rate matrix* $\mathbf{R}$ *of a* K *component SGSPN* $\mathcal{S}$*:*

$$\mathbf{G} = \bigoplus_{i=1}^{K} \hat{\mathbf{R}}_i + \sum_{t \in \mathrm{TS}} \mathbf{w}(t)[\bigotimes_{i=1}^{K} \mathbf{B}_i(t)]$$

Moreover

$$\forall \bar{i} \in \mathrm{RS}(\mathcal{S}) \text{ and } \forall \bar{j} \in \mathrm{RPS}(\mathcal{S}) \setminus \mathrm{RS}(\mathcal{S}) : \quad \mathbf{F}_{\bar{i},\bar{j}} = 0.$$

Proof:
It follows directly from theorem 12.4. ♢

As a consequence the steady state distribution of a stochastic SAM can be computed using the $\mathbf{G}$ matrices given in equations (12.5). Indeed, as in the *Superposed GSPN* case [16], if we apply an iterative solution method for $\boldsymbol{\pi} \cdot \mathbf{G} = \mathbf{0}$, and if the initial probability vector assigns a non-null probability only to reachable states, for example by assigning a value of 1 to the initial marking, then by the second item of the above theorem we never assign a non-null probability to a non reachable state.

The distance between RS and PS can limit the applicability of the technique, but if a bit vector of size |PS| can be allocated in memory, then a state space exploration can be performed [21], which, with an additional tree-like data structure of size $O(|\mathrm{RS}| \cdot \log |\mathrm{RS}_i|)$ (i is the index of the component whose state space is represented in the tree leaves), allows the definition of multiplication algorithms that consider only reachable states [7]. The computational overhead is at most logarithmic in |RS| [5].

In those cases in which PS is too large for the bit vector to fit in memory it is critical to be able to reduce the distance between RS and PS. The next section presents the use of abstract view to reach this goal.

12.4 Using an abstract view: the SAM case

Consider a general P/T system that (eventually by construction) admits a structured view as a set of K *modules* (disjoint GSPN systems) that asynchronously communicate by means of a set of places called *buffers*.

Definition 12.6 *A PN system,* $\mathcal{S} = \langle P_1 \cup \ldots \cup P_K \cup B, T_1 \cup \ldots \cup T_K, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$*, is a* System of Asynchronously Communicating Modules *view, or simply a SAM, if:*

1. $P_i \cap P_j = \emptyset$, $\forall i, j \in \{1, \ldots, K\}, i \neq j$;
2. $T_i \cap T_j = \emptyset$, $\forall i, j \in \{1, \ldots, K\}, i \neq j$;
3. $P_i \cap B = \emptyset$, $\forall i \in \{1, \ldots, K\}$;

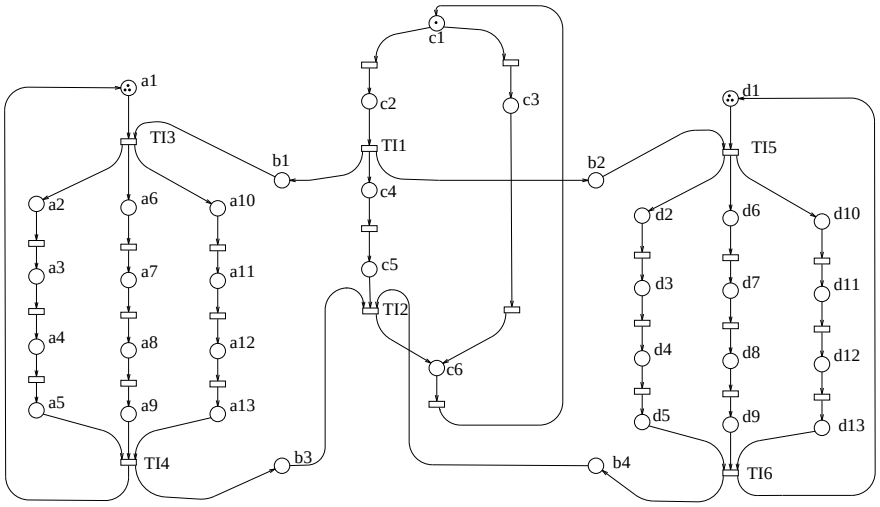


Figure 12.4: SAM view of an SPN.

4. $\mathbf{Pre}[P_i, T_j] = \mathbf{Post}[P_i, T_j] = \mathbf{0}$, $\forall i, j \in \{1, \ldots, K\}, i \neq j$.

$\langle \mathcal{N}_i, \mathbf{m_{0}}_i \rangle = \langle P_i, T_i, \mathbf{Pre}_i, \mathbf{Post}_i, \mathbf{m_{0}}_i \rangle$, $i \in \{1, \ldots, K\}$, *are called* modules *of* $\mathcal{S}$ *(where* $\mathbf{Pre}_i$,$\mathbf{Post}_i$, *and* $\mathbf{m_{0}}_i$ *are the restrictions of* $\mathbf{Pre}$, $\mathbf{Post}$, *and* $\mathbf{m_0}$ *to* P_i *and* T_i*).*

The places B *are called buffers.*

Transitions belonging to the set $\mathrm{TI} = {}^\bullet B \cup B^\bullet$ *are called* interface *transitions. The remaining ones* $((T_1 \cup \ldots \cup T_K) \setminus \mathrm{TI})$ *are called* internal *transitions.*

A PN system may have several SAM views varying between the following two extreme positions. Any arbitrary PN system might be considered as a SAM with:

- a single module, thus with an empty set of buffers; or, in the other extreme, with
- as many modules as transitions, and in this case all the places would be considered as buffers.

Therefore, the effect of the above definition is to assume that a *partitioned view* of the system into modules connected through buffers is given (or known a priori).

An example of SAM view of a GSPN is depicted in Figure 12.5, where we can consider the system as a couple of GSPN with two buffers (places) that connect them. The first GSPN, $\mathcal{S}_1$, is generated by places labeled with a while the second one, $\mathcal{S}_2$, is generated by places labeled with c. Buffers are labelled $b1$ and $b2$. Transitions $I1$ and $I2$ are *interface transitions* between $\mathcal{S}_1$ and the buffers while $I3, I4, I5, I6$ are the interface between the buffers and $\mathcal{S}_2$. The

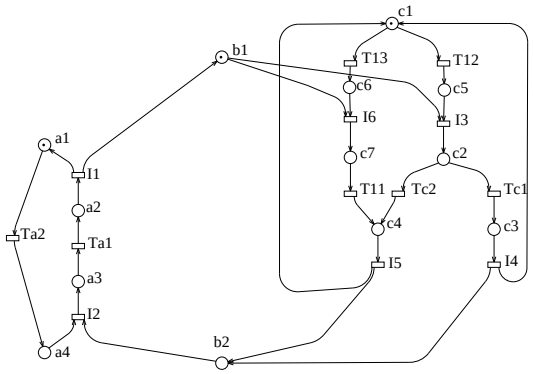


Figure 12.5: The example SAM.

rest of transitions are called *internal transitions* (since they model actions that change only the internal state of the corresponding GSPNs).

Other views for the same net are possible: apart from the two extreme SAM views mentioned above (a single module or as many modules as transitions), two alternative SAM views for the system could be easily considered. The first would be to consider as a single module the subnet generated by places $b1$, $b3$, and those labeled with a and c, the second module would be the subnet generated by places labeled with d, the buffers being $b2$ and $b4$. The second additional SAM view would consider as the first module the subnet generated by places labeled with a; the second module would be the subnet generated by places $b2$, $b4$, and those labeled with c and d. In this case, places $b1$ and $b3$ would be the buffers. The existence of many SAM views of a net system opens the question of which one is the better (e.g., the more efficient) for the computation of the solution.

12.4.1 Decomposition according to the SAM view: basic ideas

We now show the use of abstract view for limiting the distance between RS and PS. The abstract view is called *basic skeleton*, and is obtained applying a a *reduction* of the internal behaviour of the modules of the SAM (*internal behaviour* means firing of internal transitions). To partition the states of the K components according to the states of the basic skeleton, K additional systems are built called *low level systems*. In each low level system, only one of the modules of the original system is kept in full details while the internal structure of the others is reduced. In the basic skeleton, the internal structure of all modules is reduced.

In the next subsection, formal definitions and technical details of the reduc-

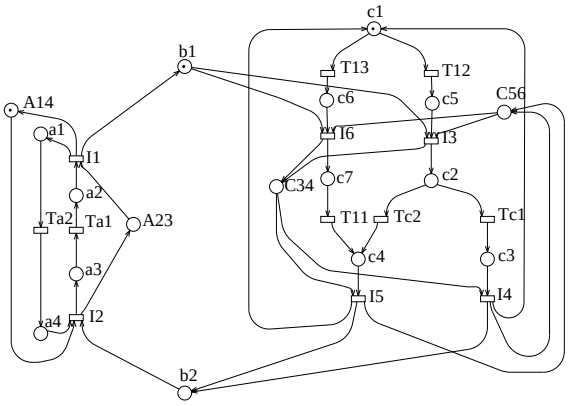


Figure 12.6: The SAM example model with additional implicit places.

tion process will be introduced. Later, we shall use the low level systems and basic skeleton introduced here for a structured construction of the reachability set of the original model and also for a structured computation of its steady-state probabilities. The complete reduction rule is presented in [10], while in the following we discuss a simplified version, that applies to modules that are state machines.

Let us consider the example in Figure 12.5 with a SAM view that distinguishes two modules, $\mathcal{N}_1$ and $\mathcal{N}_2$, (subnets generated by places labeled with a and c, respectively) and two buffers, $b1$ and $b2$. The first step is to derive in an efficient way an *extended system*, $\mathcal{ES}$, like that depicted in Figure 12.6. It consists of the original system plus the addition of some *implicit places* ($A14$, $A23$, $C34$, and $C56$) that summarize information of the structure of $\mathcal{N}_1$ and $\mathcal{N}_2$. An implicit place [13] is one whose removal does not affect the behaviour of the system (therefore, behaviour of the original and of the extended systems is the same). Here, by behaviour we understand the *interleaving semantics*, i.e., the sequential observations or language [13], although the notion of implicit place can be directly extended to cope with a *step semantics* [12]. Since we are considering a Markovian interpretation of PN's and single-server semantics of transitions, the embedded CTMC of a system is preserved if implicit places are added or removed.

Implicit places to be added are computed as follows. First, an *equivalence relation* R is introduced in the set of places P_i of each module, partitioning P_i into equivalence classes P_i^j. Two places of a module are related by R iff there exists a non-directed path including only nodes from that module that does not include interface transitions. In the example, places in module $\mathcal{N}_1$ are partitioned into two equivalence classes, $P_1^1 = \{a1, a4\}$ and $P_1^2 = \{a2, a3\}$, while places in module $\mathcal{N}_2$ are partitioned into $P_2^1 = \{c2, c3, c4, c7\}$ and $P_2^2 = \{c1, c5, c6\}$. Then, a set H_i^j (standing for "High-level places") of implicit places

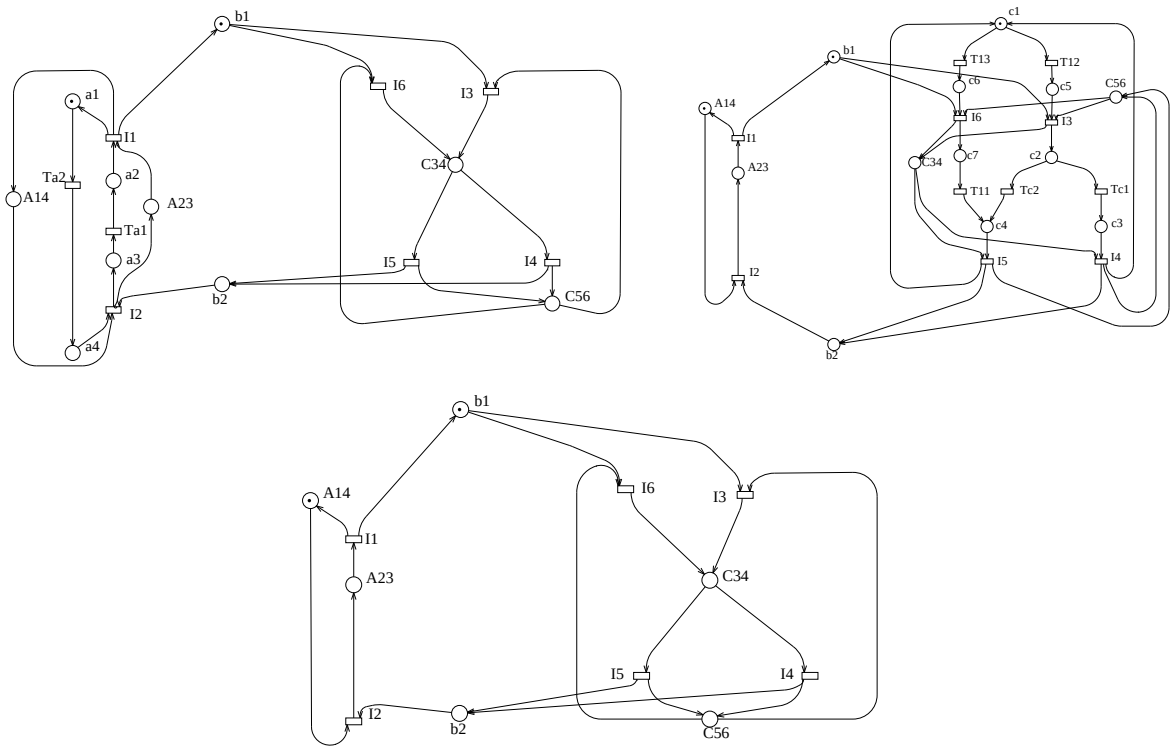


Figure 12.7: The $\mathcal{LS}_1$, $\mathcal{LS}_2$, and the $\mathcal{BS}$ systems (upper left, upper right, and central bottom, respectively) corresponding to the SAM of Fig. 12.5.

that include information of the behaviour on $\mathcal{N}_i$ is computed for each equivalence class P_i^j. For instance, in the example, $H_2^1 = \{C34\}$ is the set (in this case only one place) of implicit places corresponding to the equivalence class $P_2^1 = \{c2, c3, c4, c7\}$.

The next step is to derive the *low level system*, $\mathcal{LS}_i$ $(i = 1, \ldots, K)$, by reducing all the modules $\mathcal{N}_j$, $j \neq i$, to their interface transitions and to the implicit places that were added in the extended system, while $\mathcal{N}_i$ is fully preserved. Then, the *basic skeleton*, $\mathcal{BS}$, is derived by reducing *all* the modules in the same way as for $\mathcal{LS}_i$. The basic skeleton defines the more abstract view of the original system that we are going to consider. Figure 12.7 shows the low level systems, $\mathcal{LS}_1$ and $\mathcal{LS}_2$, and the basic skeleton, $\mathcal{BS}$ for the running example. Notice that if $\mathcal{LS}_1$ and $\mathcal{LS}_2$ were synchronized by merging common transitions (interface transitions) and identifying common places (buffers and implicit places), the extended system (with equivalent behaviour to the original system) would be obtained. Notice also that the basic skeleton is the common abstraction between $\mathcal{LS}_1$ and $\mathcal{LS}_2$.

Definition 12.7 *Let* $\mathcal{S} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$ *be a SAM with* $P = P_1 \cup \ldots \cup$

$P_K \cup B$ *and* $T = T_1 \cup \ldots \cup T_K$. *Let* R *be the equivalence relation defined on all the places in* $P \setminus B$ *by:* $\langle p_1^i, p_2^i \rangle \in \mathrm{R}$ *for* $p_1^i, p_2^i \in P_i$ *iff there exists a non-directed path* np *in* $\mathcal{SM}_i$ *from* p_1^i *to* p_2^i *such that* $\mathrm{np} \cap \mathrm{TI} = \emptyset$ *(i.e., containing only internal transitions). Let* $[p_1^i], \ldots, [p_{r(i)}^i]$ *be the different equivalence classes defined in* P_i *by the relation* R, $i = 1, \ldots, K$. *The extended system of* $\mathcal{S}$ *is* $\mathcal{ES} = \langle P_{\mathcal{ES}}, T_{\mathcal{ES}}, \mathbf{Pre}_{\mathcal{ES}}, \mathbf{Post}_{\mathcal{ES}}, \mathbf{m_0}^{\mathcal{ES}} \rangle$, *defined as:*
i) $P_{\mathcal{ES}} = P \cup H_1 \cup \ldots \cup H_K$, *with* $H_i = \{h_1^i, \ldots, h_{r(i)}^i\}$, $i = 1, \ldots, K$, *such that* $H_i \cap P = \emptyset$ *and* $H_i \cap H_j = \emptyset$, *for* $j \neq i$, $i = 1, \ldots, K$;
ii) $T_{\mathcal{ES}} = T$;
iii) $\mathbf{Pre}_{\mathcal{ES}}|_{P \times T} = \mathbf{Pre}$; $\mathbf{Post}_{\mathcal{ES}}|_{P \times T} = \mathbf{Post}$;
iv) $\mathbf{Pre}[h_j^i, t] = \sum_{p \in [p_j^i]} \mathbf{Pre}[p, t]$ *and* $\mathbf{Post}[h_j^i, t] = \sum_{p \in [p_j^i]} \mathbf{Post}[p, t]$, *for* $t \in \mathrm{TI} \cap T_i$; $\mathbf{Pre}[h_j^i, t] = 0$ *and* $\mathbf{Post}[h_j^i, t] = 0$ *for* $t \notin \mathrm{TI} \cap T_i$; *for* $j = 1, \ldots, r(i)$, *for* $i = 1, \ldots, K$;
v) $\mathbf{m_0}^{\mathcal{ES}}[p] = \mathbf{m_0}[p]$, *for all* $p \in P$;
vi) $\mathbf{m_0}^{\mathcal{ES}}[h_j^i] = \sum_{p \in [p_j^i]} \mathbf{m_0}[p]$, *for* $j = 1, \ldots, r(i)$, *for* $i = 1, \ldots, K$.

As an example, the extended system of the GSPN of Figure 12.5 is depicted in Figure 12.6 where place C_{34} summarizes places c_2, c_3, c_4, and c_7, place C_{56} summarizes places c_1, c_5, and c_6, place A_{14} summarizes places a_1 and a_4, and place A_{23} summarizes places a_2 and a_3. In the example, we use the convention of naming implicit places with upper case letters, with an index that is a composition of two of the indices of the implicated places. In the example we have $H_1 = \{A_{14}, A_{23}\}$ and $H_2 = \{C_{34}, C_{56}\}$.

From the extended system, K different *low level systems* $\mathcal{LS}_i$ can be obtained deleting all places in P_j, $j \neq i$, and transitions in $T_j \setminus \mathrm{TI}$, $j \neq i$ $(i = 1, \ldots, K)$.

Definition 12.8 *Let* $\mathcal{S} = \langle P_1 \cup \ldots \cup P_K \cup B, T_1 \cup \ldots \cup T_K, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$ *be a SAM, and* $\mathcal{ES}$ *its corresponding extended system.*
i) *The low level system* $\mathcal{LS}_i$ $(i = 1, \ldots, K)$ *of* $\mathcal{S}$ *is the system obtained from* $\mathcal{ES}$ *deleting all the nodes in* $\bigcup_{j \neq i}(P_j \cup (T_j \setminus \mathrm{TI}))$ *and their adjacent arcs.*
ii) *The basic skeleton* $\mathcal{BS}$ *of* $\mathcal{S}$ *is the system obtained from* $\mathcal{ES}$ *deleting all the nodes in* $\bigcup_j (P_j \cup (T_j \setminus \mathrm{TI}))$ *and their adjacent arcs.*

In other words, in each $\mathcal{LS}_i$ all modules j, with $j \neq i$, are reduced to their interface transitions and to the implicit places that were added in the extended system, while $\mathcal{SM}_i$ is fully preserved. In the basic skeleton all the SM's are reduced in the same way. Figure 12.7, upper left, shows the low level system $\mathcal{LS}_1$ for the SAM in Figure 12.5, and Figure 12.7, upper right, shows the low level system $\mathcal{LS}_2$. Figure 12.7, lower part, shows instead the corresponding $\mathcal{BS}$ system (please note that we preserve the names that transitions and places have in $\mathcal{S}$ when they are also present in other systems like $\mathcal{ES}$ and $\mathcal{LS}_i$ $(i = 1, \ldots, K)$).

In general, the reduction technique presented here does not remove but eventually adds new paths between interface transitions (see, for instance, the path from $I6$ to $I4$ that is present in $\mathcal{BS}$ (lower portion of Figure 12.7), while it does not exist in the original model of Figure 12.5); from this fact, the next property can be proved.

Property 12.9 *define a SAM system* $\mathcal{S} = \langle P_1 \cup \ldots \cup P_K \cup B, T_1 \cup \ldots \cup T_K, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$, $\mathcal{LS}_i$ *its low level systems* ($i = 1, \ldots, K$), $\mathcal{BS}$ *its basic skeleton, and* $\mathrm{L}(\mathcal{S})$ *the language of* $\mathcal{S}$. *Then:*
1. $\mathrm{L}(\mathcal{S})|_{T_i \cup \mathrm{TI}} \subseteq \mathrm{L}(\mathcal{LS}_i)$, *for* $i = 1, \ldots, K$.
2. $\mathrm{L}(\mathcal{S})|_{\mathrm{TI}} \subseteq \mathrm{L}(\mathcal{BS})$.

(for a proof see [9])

12.4.2 Construction of the reachability set

Let us define a partition on the reachability sets of original ($\mathrm{RS}(\mathcal{S})$), extended ($\mathrm{RS}(\mathcal{ES})$), and low level systems ($\mathrm{RS}(\mathcal{LS}_i)$) according to the projection of the marking on the places of the basic skeleton.

Definition 12.10 *Let* $\mathcal{S} = \langle P_1 \cup \ldots \cup P_K \cup B, T_1 \cup \ldots \cup T_K, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$ *be a SAM,* $\mathcal{ES}$ *its extended system,* $\mathcal{LS}_i$ *its low level systems* ($i = 1, \ldots, K$), *and* $\mathcal{BS}$ *its basic skeleton. Then, the following subsets of the reachability sets* $\mathrm{RS}(\mathcal{S})$, $\mathrm{RS}(\mathcal{ES})$, *and* $\mathrm{RS}(\mathcal{LS}_i)$ *are defined for each* $\mathbf{z} \in \mathrm{RS}(\mathcal{BS})$:

$$\mathrm{RS}_{\mathbf{z}}(\mathcal{ES}) = \{\mathbf{m} \in \mathrm{RS}(\mathcal{ES}) \mid \mathbf{m}|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z}\}$$
$$\mathrm{RS}_{\mathbf{z}}(\mathcal{S}) = \{\mathbf{m} \in \mathrm{RS}(\mathcal{S}) \mid \exists \mathbf{m}' \in \mathrm{RS}_{\mathbf{z}}(\mathcal{ES}) : \mathbf{m}'|_{P_1 \cup \ldots \cup P_K \cup B} = \mathbf{m}\}$$
$$\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_i) = \{\mathbf{m}_i \in \mathrm{RS}(\mathcal{LS}_i) \mid \mathbf{m}_i|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z}\}$$

From the definition, the following obvious partitions are obtained: $\mathrm{RS}(\mathcal{S}) = \biguplus_{\mathbf{z} \in \mathrm{RS}(\mathcal{BS})} \mathrm{RS}_{\mathbf{z}}(\mathcal{S})$, $\mathrm{RS}(\mathcal{ES}) = \biguplus_{\mathbf{z} \in \mathrm{RS}(\mathcal{BS})} \mathrm{RS}_{\mathbf{z}}(\mathcal{ES})$, and $\mathrm{RS}(\mathcal{LS}_i) = \biguplus_{\mathbf{z} \in \mathrm{RS}(\mathcal{BS})} \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_i)$ ($i = 1, \ldots, K$), where symbol $\uplus$ denotes disjoint union of sets. Remember that reachability sets of low level systems and of the basic skeleton are finite.

The following result essentially states that each reachable marking of a SAM can be expressed as a composition of conveniently selected markings of all the low level systems built from the original model.

Theorem 12.11 *Let* $\mathcal{S} = \langle P_1 \cup \ldots \cup P_K \cup B, T_1 \cup \ldots \cup T_K, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$ *be a SAM,* $\mathcal{LS}_i$ *its low level systems* ($i = 1, \ldots, K$), *and* $\mathcal{BS}$ *its basic skeleton. Denoting, for each* $\mathbf{z} \in \mathrm{RS}(\mathcal{BS})$,

$$\mathrm{RPS}_{\mathbf{z}}(\mathcal{S}) = \{\mathbf{z}|_B\} \times \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_1)|_{P_1} \times \cdots \times \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_K)|_{P_K}$$

and

$$\mathrm{RPS}(\mathcal{S}) = \bigcup_{\mathbf{z} \in \mathrm{RS}(\mathcal{BS})} \mathrm{RPS}_{\mathbf{z}}(\mathcal{S})$$

then:

$$\mathrm{RS}(\mathcal{S}) \subseteq \mathrm{RPS}(\mathcal{S}) = \biguplus_{\mathbf{z} \in \mathrm{RS}(\mathcal{BS})} \mathrm{RPS}_{\mathbf{z}}(\mathcal{S}) \tag{12.4}$$

Moreover:

$$\mathrm{RS}_{\mathbf{z}}(\mathcal{S}) \subseteq \mathrm{RPS}_{\mathbf{z}}(\mathcal{S})$$

Proof:
First, we prove the inclusion.

Let $\mathbf{m} \in \mathrm{RS}_{\mathbf{z}}(\mathcal{S})$. Then there exists a sequence σ such that $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$. By Property 12.9, there exist sequences σ_i with $\sigma_i = \sigma|_{T_i \cup \mathrm{TI}}$ $(i = 1, \ldots, K)$ and $\sigma_{\mathcal{BS}}$ with $\sigma_{\mathcal{BS}} = \sigma|_{\mathrm{TI}}$ that may be fired in $\mathcal{LS}_i$ $(i = 1, \ldots, K)$ and in $\mathcal{BS}$, respectively: $\mathbf{m}_0^i \xrightarrow{\sigma_i} \mathbf{m}_i$ $(i = 1, \ldots, K)$ and $\mathbf{z_0} \xrightarrow{\sigma_{\mathcal{BS}}} \mathbf{z}$, where $\mathbf{m}_0^i$ $(i = 1, \ldots, K)$ and $\mathbf{z_0}$ denote the initial markings of $\mathcal{LS}_i$ $(i = 1, \ldots, K)$ and $\mathcal{BS}$, respectively.

Then: $\mathbf{m}_i|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z}$, thus $\mathbf{m}_i \in \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_i)$, $i = 1, \ldots, K$ (this is because $\mathbf{m}_0^i|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z_0}$, $\sigma_i|_{\mathrm{TI}} = \sigma_{\mathcal{BS}}$, and $H_1 \cup \ldots \cup H_K \cup B \subset {}^\bullet\mathrm{TI} \cup \mathrm{TI}^\bullet$).

From analogous arguments, $\mathbf{z}|_B = \mathbf{m}|_B$ (because $\mathbf{z_0}|_B = \mathbf{m_0}|_B$, $\sigma|_{\mathrm{TI}} = \sigma_{\mathcal{BS}}$, and $B \subset {}^\bullet\mathrm{TI} \cup \mathrm{TI}^\bullet$).

Then, $\mathbf{m} = (\mathbf{z}|_B, \mathbf{m}_1|_{P_1}, \ldots, \mathbf{m}_K|_{P_K})$, because $\mathbf{m_0}|_{P_i} = \mathbf{m}_0^i|_{P_i}$, $\sigma|_{T_i} = \sigma_i|_{T_i}$, and $P_i \subset {}^\bullet T_i \cup T_i^\bullet$, $i = 1, \ldots, K$.

Now, we prove by contradiction that the union is disjoint.

Assume that there exist two different states $\mathbf{z}, \mathbf{z}' \in \mathrm{RS}(\mathcal{BS})$ such that: $(\mathbf{z}|_B, \mathbf{m}_1|_{P_1}, \ldots, \mathbf{m}_K|_{P_K}) = (\mathbf{z}'|_B, \mathbf{m}'_1|_{P_1}, \ldots, \mathbf{m}'_K|_{P_K})$, with $\mathbf{m}_i \in \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_i)$, $\mathbf{m}'_i \in \mathrm{RS}_{\mathbf{z}'}(\mathcal{LS}_i)$ $(i = 1, \ldots, K)$.

Then: $\mathbf{z}|_B = \mathbf{z}'|_B$ (obvious), $\mathbf{m}_i|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z}$ (by definition of $\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_i)$) and $\mathbf{m}'_i|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z}'$ (by definition of $\mathrm{RS}_{\mathbf{z}'}(\mathcal{LS}_i)$).

Since $\mathbf{m}_i|_{P_i} = \mathbf{m}'_i|_{P_i}$ (obvious) and places H_i are implicit in $\mathcal{LS}_i$, $\mathbf{m}_i|_{H_i} = \mathbf{m}'_i|_{H_i}$. Therefore, $\mathbf{z}|_{H_i} = \mathbf{z}'|_{H_i}$, $i = 1, \ldots, K$.

Then, $\mathbf{z} = \mathbf{z}'$ and the result follows. ♢

Observe that, by definition, the RPS of a SAM with K modules is the Cartesian product of $K + 1$ terms, since the buffers contribution is an isolated term $(\mathbf{z}|_B)$. Actually this term can be removed from the formula by simply taking the buffer contribution out of any $\mathcal{LS}_i$ system, by writing $\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_i)|_{P_i \cup B}$. Concerning the example, Table 12.1 lists the reachability set of the SAM of Figure 12.5, that consists of 26 states, $\mathbf{v}_i$. Table 12.2 lists instead the reachability sets of $\mathcal{BS}$ $(\mathbf{z}_i)$, $\mathcal{LS}_1$ $(\mathbf{x}_j)$, and $\mathcal{LS}_2$ $(\mathbf{y}_k)$, respectively. For each state of $\mathcal{LS}_1$ ($\mathcal{LS}_2$) we have indicated the partition $\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_1)$ ($\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_2)$) to which the state belongs (third column). The elements of the partition are identified by the corresponding high level marking in $\mathcal{BS}$. As proved above

$$\mathrm{RS}(\mathcal{S}) \subseteq \mathrm{RPS}(\mathcal{S}) = \biguplus_{\mathbf{z} \in \mathrm{RS}(\mathcal{BS})} \{\mathbf{z}|_B\} \times \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_1)|_{P_1} \times \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_2)|_{P_2}$$

and in this case we actually have an equality. As an example consider the case of $\mathbf{z} = \mathbf{z}_1$: then the cross product of $\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_1) = \{\mathbf{x}_1, \mathbf{x}_2\}$ (markings of $\mathcal{LS}_1$) and $\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_2) = \{\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3\}$ (markings of $\mathcal{LS}_2$) produces the states of RPS: $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_6$, and $\mathbf{v}_7$.

It is important to remember that in general $\mathrm{RPS}(\mathcal{S}) \neq \mathrm{RS}(\mathcal{S})$. The live and bounded PN system of Figure 12.8 is a case in which the inclusion is strict. Consider its SAM view where *bi* $(i = 1, \ldots, 4)$ are buffers, interface transitions are labeled with TI, and there are three modules: the first one is just $TI5$, the second is the subnet generated by places labeled with a, and the third is generated by places labeled with c. Places $z1, z2$ are the implicit places added

RS of $\mathcal{S}$	
$\mathbf{v}_1$	a1, b1, c1
$\mathbf{v}_2$	a1, b1, c6
$\mathbf{v}_3$	a1, b1, c5
$\mathbf{v}_4$	a4, b1, c1
$\mathbf{v}_5$	a1, c7
$\mathbf{v}_6$	a4, b1, c6
$\mathbf{v}_7$	a4, b1, c5
$\mathbf{v}_8$	a1, c2
$\mathbf{v}_9$	a1, c4
$\mathbf{v}_{10}$	a4, c7
$\mathbf{v}_{11}$	a4, c2
$\mathbf{v}_{12}$	a1, c3
$\mathbf{v}_{13}$	a4, c4

RS of $\mathcal{S}$	
$\mathbf{v}_{14}$	a1, c1, b2
$\mathbf{v}_{15}$	a4, c3
$\mathbf{v}_{16}$	a4, c1, b2
$\mathbf{v}_{17}$	a1, b2, c6
$\mathbf{v}_{18}$	a1, b2, c5
$\mathbf{v}_{19}$	a4, b2, c6
$\mathbf{v}_{20}$	a4, b2, c5
$\mathbf{v}_{21}$	a3, c1
$\mathbf{v}_{22}$	a3, c6
$\mathbf{v}_{23}$	a3, c5
$\mathbf{v}_{24}$	a2, c1
$\mathbf{v}_{25}$	a2, c6
$\mathbf{v}_{26}$	a2, c5

Table 12.1: RS of the SAM of Figure 12.5.

RS of $\mathcal{BS}$	
$\mathbf{z}_1$	A14, C56, b1
$\mathbf{z}_2$	A14, C34
$\mathbf{z}_3$	A14, C56, b2
$\mathbf{z}_4$	A23, C56

RS of $\mathcal{LS}_1$		
$\mathbf{x}_1$	a1, b1, C56, A14	$\mathbf{z}_1$
$\mathbf{x}_2$	a4, b1, C56, A14	$\mathbf{z}_1$
$\mathbf{x}_3$	a1, C34, A14	$\mathbf{z}_2$
$\mathbf{x}_4$	a4, C34, A14	$\mathbf{z}_2$
$\mathbf{x}_5$	a1, b2, C56, A14	$\mathbf{z}_3$
$\mathbf{x}_6$	a4, b2, C56, A14	$\mathbf{z}_3$
$\mathbf{x}_7$	a3, C56, A23	$\mathbf{z}_4$
$\mathbf{x}_8$	a2, C56, A23	$\mathbf{z}_4$

RS of $\mathcal{LS}_2$		
$\mathbf{y}_1$	A14, b1, c1, C56	$\mathbf{z}_1$
$\mathbf{y}_2$	A14, b1, c6, C56	$\mathbf{z}_1$
$\mathbf{y}_3$	A14, b1, c5, C56	$\mathbf{z}_1$
$\mathbf{y}_4$	A14, c7, C34	$\mathbf{z}_2$
$\mathbf{y}_5$	A14, c2, C34	$\mathbf{z}_2$
$\mathbf{y}_6$	A14, c4, C34	$\mathbf{z}_2$
$\mathbf{y}_7$	A14, c3, C34	$\mathbf{z}_2$
$\mathbf{y}_8$	A14, b2, c1, C56	$\mathbf{z}_3$
$\mathbf{y}_9$	A14, b2, c6, C56	$\mathbf{z}_3$
$\mathbf{y}_{10}$	A14, b2, c5, C56	$\mathbf{z}_3$
$\mathbf{y}_{11}$	A23, c1, C56	$\mathbf{z}_4$
$\mathbf{y}_{12}$	A23, c6, C56	$\mathbf{z}_4$
$\mathbf{y}_{13}$	A23, c5, C56	$\mathbf{z}_4$

Table 12.2: RS's of the SAM of Figures 12.7.

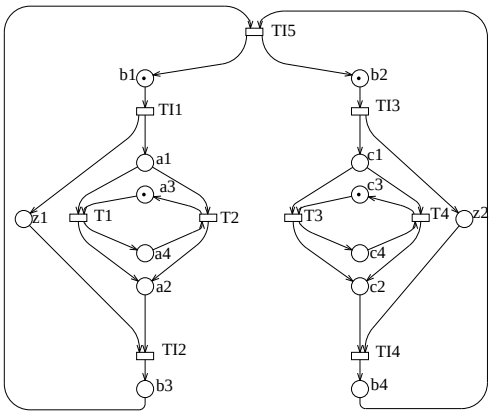


Figure 12.8: A SAM with RPS $\neq$ RS.

to define the extended system. Reachable states corresponding to the high level state [z1, z2] are: [z1, z2, a1, a3, c1, c3], [z1, z2, a1, a3, c2, c4], [z1, z2, a2, a4, c1, c3], [z1, z2, a2, a4, c2, c4], [z1, z2, a1, a4, c1, c4], [z1, z2, a1, a4, c2, c3], [z1, z2, a2, a3, c1, c4], and [z1, z2, a2, a3, c2, c3]. But the cross product of [z1, z2, a1, a3] (reachable in $\mathcal{LS}_1$) and [z1, z2, c1, c4] (reachable in $\mathcal{LS}_2$) generates [z1, z2, a1, a3, c1, c4], that belongs to RPS but is non-reachable in the system of Figure 12.8.

12.4.3 The structured solution of SAM

If we arrange the order of states according to the high level state $\mathbf{z}$, then we can describe matrix $\mathbf{R}$ (respectively, $\mathbf{R}_i$) in terms of blocks $(\mathbf{z}, \mathbf{z}')$, of size $|\mathrm{RS}_{\mathbf{z}}(\mathcal{S})| \cdot |\mathrm{RS}_{\mathbf{z}'}(\mathcal{S})|$ (respectively, $|\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_i)| \cdot |\mathrm{RS}_{\mathbf{z}'}(\mathcal{LS}_i)|$). We shall indicate with $\mathbf{R}(\mathbf{z}, \mathbf{z}')$ ($\mathbf{R}_i(\mathbf{z}, \mathbf{z}')$, respectively) such blocks.

Blocks $\mathbf{R}_i(\mathbf{z}, \mathbf{z}')$ with $\mathbf{z} = \mathbf{z}'$ have non null entries that are due *only* to the firing of transitions in $T_i \setminus \mathrm{TI}$ (internal behaviour), while blocks $\mathbf{R}_i(\mathbf{z}, \mathbf{z}')$ with $\mathbf{z} \neq \mathbf{z}'$ have non null entries due *only* to the firing of transitions in TI.

Let $\mathrm{TI}_{\mathbf{z},\mathbf{z}'}$ with $\mathbf{z} \neq \mathbf{z}'$, be the set of transitions $t \in \mathrm{TI}$ such that $\mathbf{z} \xrightarrow{t} \mathbf{z}'$ in the basic skeleton $\mathcal{BS}$. From a matrix $\mathbf{R}_i(\mathbf{z}, \mathbf{z}')$, with $\mathbf{z} \neq \mathbf{z}'$ we can build additional matrices $\mathbf{B}_i(t)(\mathbf{z}, \mathbf{z}')$, for each $t \in \mathrm{TI}_{\mathbf{z},\mathbf{z}'}$, according to the following definition:

$$\mathbf{B}_i(t)(\mathbf{z}, \mathbf{z}')[\mathbf{m}, \mathbf{m}'] = \begin{cases} 1 & \text{if } \mathbf{m} \xrightarrow{t} \mathbf{m}' \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{m}$ and $\mathbf{m}'$ are two of the states with a high level view equal to $\mathbf{z}$ and $\mathbf{z}'$ respectively: $\mathbf{m}|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z}$, and $\mathbf{m}'|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z}'$.

We can now build the following matrices $\mathbf{G}(\mathbf{z}, \mathbf{z}')$ of size $|\mathrm{RPS}_{\mathbf{z}}(\mathcal{S})| \cdot$

$|\mathrm{RPS}_{\mathbf{z}'}(\mathcal{S})|$:

$$\begin{aligned} \mathbf{G}(\mathbf{z},\mathbf{z}) &= \bigoplus_{i=1}^{K} \mathbf{R}_i(\mathbf{z},\mathbf{z}) \\ \mathbf{G}(\mathbf{z},\mathbf{z}') &= \sum_{t\in \mathrm{TI}_{\mathbf{z},\mathbf{z}'}} w(t) \bigotimes_{i=1}^{K} \mathbf{B}_i(t)(\mathbf{z},\mathbf{z}') \end{aligned} \tag{12.5}$$

The following theorem states that a stochastic SAM can be solved for the steady-state distribution using the $\mathbf{G}$ matrix defined by the $\mathbf{G}(\mathbf{z},\mathbf{z})$ and $\mathbf{G}(\mathbf{z},\mathbf{z}')$ blocks of equation (12.5).

Theorem 12.12 *Let $\mathcal{S} = \langle P_1 \cup \ldots \cup P_K \cup B, T_1 \cup \ldots \cup T_K, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0}, w\rangle$ be a stochastic SAM, $\mathbf{R}$ its rate matrix, $\mathcal{LS}_i$ its low level systems ($i = 1,\ldots,K$), and $\mathcal{BS}$ its basic skeleton. Let $\mathbf{G}$ be the matrix defined by equations (12.5). Then:*

1. *$\forall \mathbf{z}$ and $\mathbf{z}' \in \mathrm{RS}(\mathcal{BS})$: $\mathbf{R}(\mathbf{z},\mathbf{z}')$ is a submatrix of $\mathbf{G}(\mathbf{z},\mathbf{z}')$.*
2. *$\forall \mathbf{m} \in \mathrm{RS}(\mathcal{S})$ and $\forall \mathbf{m}' \in \mathrm{RPS}(\mathcal{S}) \setminus \mathrm{RS}(\mathcal{S})$: $\mathbf{G}[\mathbf{m},\mathbf{m}'] = 0$.*

Proof:
Considering that $\mathcal{S}$ and $\mathcal{ES}$ have the same behaviour, and the same set of states, for notational convenience we use $\mathcal{ES}$ instead of $\mathcal{S}$. If $\mathbf{m}$ is a state of $\mathrm{RS}(\mathcal{ES})$ then, by Theorem 12.11, $\mathbf{m}$ can be rewritten in terms of the buffers state and of the $\mathcal{LS}_i$ component states $\mathbf{m}_i$ as

$$\mathbf{m} = \mathbf{z}|_B, \mathbf{m}_1|_{P_1\cup H_1}, \ldots, \mathbf{m}_K|_{P_K\cup H_K}$$

For the rest of the proof we shall make the buffers and implicit places component by writing a generic marking as

$$\mathbf{m} = \mathbf{l}_1, \ldots, \mathbf{l}_K, \mathbf{b}, \mathbf{H}$$

where $\mathbf{l}_i = \mathbf{m}|_{P_i}$, $\mathbf{b} = \mathbf{z}|_B$, and $\mathbf{H} = \mathbf{z}|_{H_1\cup\ldots\cup H_K}$.

We first prove that:

$$\forall\, \mathbf{m},\mathbf{m}' \in \mathrm{RS}(\mathcal{ES}) : \quad \mathbf{R}[\mathbf{m},\mathbf{m}'] = \mu \implies \mathbf{G}[\mathbf{m},\mathbf{m}'] = \mu$$

(since from $\mathbf{m}$ and $\mathbf{m}'$ the high level states $\mathbf{z}$ and $\mathbf{z}'$ are uniquely determined, we usually omit the specification of $\mathbf{z}$ and $\mathbf{z}'$).

Case $\mathbf{z} = \mathbf{z}'$: if the high level view is the same, then the change of state from $\mathbf{m}$ to $\mathbf{m}'$ can only be due to internal transitions (i.e., not belonging to TI), thus belonging to a single module $\mathcal{N}_i$. But, by definition of $\bigoplus$, $\mathbf{G}(\mathbf{z},\mathbf{z})$ expresses the independent composition of the stochastic processes represented by the $\mathbf{R}_i(\mathbf{z},\mathbf{z})$, which is exactly the behaviour of $\mathcal{LS}_i$ system due to transitions local to $\mathcal{N}_i$ (transitions belonging to $T_i \setminus \mathrm{TI}$).

Case $\mathbf{z} \neq \mathbf{z}'$: if the high level view is different, the change of state from $\mathbf{m}$ to $\mathbf{m}'$ can only be due to transitions in TI. Let t be such a transition and assume, for the time being, that there is only one, so that $\mu = w(t)$. If t is a transition

in the interface, then its enabling depends on the marking of the buffer places, and on the places of a *single* module.

If t is enabled in $\mathbf{m}$, then t is enabled in *each* $(\mathbf{l}_i, \mathbf{b}, \mathbf{H})$ state of the K $\mathcal{LS}_i$ systems, and in each $\mathcal{LS}_i$ system produces a change of state $\mathbf{l}_i, \mathbf{b}, \mathbf{H} \xrightarrow{t} \mathbf{l}'_i, \mathbf{b}', \mathbf{H}$. By definition of $\mathbf{B}_i(t)(\mathbf{z}, \mathbf{z}')$ we have that $\mathbf{B}_i(t)(\mathbf{z}, \mathbf{z}')[\mathbf{m}_i, \mathbf{b}, \mathbf{H}] = 1, \ \forall i$, and by definition of $\bigotimes$ there is a 1 in the corresponding entry of $\bigotimes_{i=1}^{K} \mathbf{B}_i(t)(\mathbf{z}, \mathbf{z}')$, and therefore a value of $w(t)$ in the $\mathbf{G}(\mathbf{z}, \mathbf{z}')$ matrix.

In the expression of $\mathbf{G}(\mathbf{z}, \mathbf{z}')$ the situation in which more than one transition realizes the same change of state is accounted for by the summation over all transitions in $\mathrm{TI}_{\mathbf{z},\mathbf{z}'}$.

We now prove the second part of the theorem, that can be rewritten, in terms of $\mathcal{ES}$, as:

$$\forall \mathbf{m} \in \mathrm{RS}(\mathcal{ES}), \forall \mathbf{m}' \in \mathrm{RPS}(\mathcal{S}) \setminus \mathrm{RS}(\mathcal{ES}) : \mathbf{G}[\mathbf{m}, \mathbf{m}'] = 0$$

As in the previous case, from $\mathbf{m}$ and $\mathbf{m}'$ the high level states $\mathbf{z}$ and $\mathbf{z}'$ are uniquely determined, and we consider for $\mathbf{m}$ and $\mathbf{m}'$ the same decomposition in terms of modules, buffers, and implicit places.

We prove this part by contradiction, assuming that $\mathbf{G}[\mathbf{m}, \mathbf{m}'] \neq 0$, which can be rewritten as: $\mathbf{G}[(\mathbf{l}_1, \ldots, \mathbf{l}_K, \mathbf{b}, \mathbf{H}), (\mathbf{l}'_1, \ldots, \mathbf{l}'_K, \mathbf{b}', \mathbf{H}')] \neq 0$.
Case $\mathbf{z} = \mathbf{z}'$: by definition of $\bigoplus$, $\mathbf{G}[\mathbf{m}, \mathbf{m}'] \neq 0$ implies that there exists exactly one index i such that $\mathbf{R}_i(\mathbf{z}, \mathbf{z})[(\mathbf{l}_i, \mathbf{b}, \mathbf{H}), (\mathbf{l}'_i, \mathbf{b}, \mathbf{H})] \neq 0$, and this change of state can be due only to the $\mathbf{l}_i$ portion of the state (since the high level view $\mathbf{z}$ does not change). Therefore there must be a transition $t \in T_i \setminus \mathrm{TI}$ which is enabled in state $(\mathbf{l}_i, \mathbf{b}, \mathbf{H})$ of $\mathcal{LS}_i$, but then t is enabled also in state $\mathbf{m} = (\mathbf{l}_1, \ldots, \mathbf{l}_K, \mathbf{b}, \mathbf{H})$, and its firing produces $\mathbf{m}' = (\mathbf{l}_1, \ldots, \mathbf{l}'_i, \ldots, \mathbf{l}_K, \mathbf{b}, \mathbf{H})$, thus making $\mathbf{m}'$ is reachable in $\mathcal{ES}$, which contradicts the hypothesis.

Case $\mathbf{z} \neq \mathbf{z}'$: by definition of $\bigotimes$, $\mathbf{G}[\mathbf{m}, \mathbf{m}'] \neq 0$ implies that there exists a $t \in \mathrm{TI}$ such that, for all indices i, $\mathbf{B}_i(t)[(\mathbf{l}_i, \mathbf{b}, \mathbf{H}), (\mathbf{l}'_i, \mathbf{b}', \mathbf{H}')] = 1$, but then t is enabled in state $\mathbf{m}$ of $\mathcal{ES}$, and its firing will exactly produce the state $(\mathbf{l}'_1, \ldots, \mathbf{l}'_K, \mathbf{b}', \mathbf{H}')$, thus making $\mathbf{m}'$ is reachable in $\mathcal{ES}$, and therefore in $\mathcal{S}$, which contradicts the hypothesis. ♢

As a consequence the steady state distribution of a stochastic SAM can be computed using the $\mathbf{G}$ matrices given in equations (12.5). Indeed, as in the *Superposed GSPN* case, if we apply an iterative solution method for $\boldsymbol{\pi} \cdot \mathbf{G} = \mathbf{0}$, and if the initial probability vector assigns a non-null probability only to reachable states, for example by assigning a value of 1 to the initial marking, then by the second item of the above theorem we never assign a non-null probability to a non reachable state.

Coming back to the example, we can order states in $\mathrm{RS}(\mathcal{S})$ according to

their projection over $\mathcal{BS}$, so that $\mathbf{R}$ can be written in block structured form as:

$$\mathbf{R} = \left(\begin{array}{c|c|c|c} \mathbf{R}(\mathbf{z}_1,\mathbf{z}_1) & \mathbf{R}(\mathbf{z}_1,\mathbf{z}_2) & \mathbf{R}(\mathbf{z}_1,\mathbf{z}_3) & \mathbf{R}(\mathbf{z}_1,\mathbf{z}_4) \\ \hline \mathbf{R}(\mathbf{z}_2,\mathbf{z}_1) & \mathbf{R}(\mathbf{z}_2,\mathbf{z}_2) & \mathbf{R}(\mathbf{z}_2,\mathbf{z}_3) & \mathbf{R}(\mathbf{z}_2,\mathbf{z}_4) \\ \hline \mathbf{R}(\mathbf{z}_3,\mathbf{z}_1) & \mathbf{R}(\mathbf{z}_3,\mathbf{z}_2) & \mathbf{R}(\mathbf{z}_3,\mathbf{z}_3) & \mathbf{R}(\mathbf{z}_3,\mathbf{z}_4) \\ \hline \mathbf{R}(\mathbf{z}_4,\mathbf{z}_1) & \mathbf{R}(\mathbf{z}_4,\mathbf{z}_2) & \mathbf{R}(\mathbf{z}_4,\mathbf{z}_3) & \mathbf{R}(\mathbf{z}_4,\mathbf{z}_4) \end{array}\right)$$

By definition of SAM only interface transitions contribute to $\mathbf{R}(\mathbf{z}_i,\mathbf{z}_j)$, when $i \neq j$, while only internal transitions contribute to $\mathbf{R}(\mathbf{z}_i,\mathbf{z}_i)$. Let us consider in more detail blocks $\mathbf{R}(\mathbf{z}_1,\mathbf{z}_1)$ and $\mathbf{R}(\mathbf{z}_1,\mathbf{z}_2)$: we explicitly write in the matrix the state identifier for rows and columns as pairs of states of $\mathcal{LS}_1$ and $\mathcal{LS}_2$. For example the second row corresponds to state $\mathbf{v}_2$ of RS($\mathcal{S}$), which is obtained as composition of state $\mathbf{x}_1$ of $\mathcal{LS}_1$ and state $\mathbf{y}_2$ of $\mathcal{LS}_2$.

$\mathbf{R}(\mathbf{z}_1,\mathbf{z}_1) =$

	$\mathbf{x}_1,\mathbf{y}_1$	$\mathbf{x}_1,\mathbf{y}_2$	$\mathbf{x}_1,\mathbf{y}_3$	$\mathbf{x}_2,\mathbf{y}_1$	$\mathbf{x}_2,\mathbf{y}_2$	$\mathbf{x}_2,\mathbf{y}_3$
$\mathbf{x}_1,\mathbf{y}_1$		$w(T_{13})$	$w(T_{12})$	$w(Ta_2)$		
$\mathbf{x}_1,\mathbf{y}_2$					$w(Ta_2)$	
$\mathbf{x}_1,\mathbf{y}_3$						$w(Ta_2)$
$\mathbf{x}_2,\mathbf{y}_1$					$w(T_{13})$	$w(T_{12})$
$\mathbf{x}_2,\mathbf{y}_2$						
$\mathbf{x}_2,\mathbf{y}_3$						

$\mathbf{R}(\mathbf{z}_1,\mathbf{z}_2) =$

	$\mathbf{x}_3,\mathbf{y}_4$	$\mathbf{x}_3,\mathbf{y}_5$	$\mathbf{x}_3,\mathbf{y}_6$	$\mathbf{x}_3,\mathbf{y}_7$	$\mathbf{x}_4,\mathbf{y}_4$	$\mathbf{x}_4,\mathbf{y}_5$	$\mathbf{x}_4,\mathbf{y}_6$	$\mathbf{x}_4,\mathbf{y}_7$
$\mathbf{x}_1,\mathbf{y}_1$								
$\mathbf{x}_1,\mathbf{y}_2$	$w(I_6)$							
$\mathbf{x}_1,\mathbf{y}_3$		$w(I_3)$						
$\mathbf{x}_2,\mathbf{y}_1$								
$\mathbf{x}_2,\mathbf{y}_2$					$w(I_6)$			
$\mathbf{x}_2,\mathbf{y}_3$						$w(I_3)$		

The only non null element of $\mathbf{R}_1(\mathbf{z}_1,\mathbf{z}_1)$ is $\mathbf{R}_1(\mathbf{z}_1,\mathbf{z}_1)[\mathbf{x}_1,\mathbf{x}_2] = w(Ta_2)$, while the only non null elements of $\mathbf{R}_2(\mathbf{z}_1,\mathbf{z}_1)$ are: $\mathbf{R}_2(\mathbf{z}_1,\mathbf{z}_1)[\mathbf{y}_1,\mathbf{y}_2] = w(T_{13})$ and $\mathbf{R}_2(\mathbf{z}_1,\mathbf{z}_1)[\mathbf{y}_1,\mathbf{y}_3] = w(T_{12})$.

The change of state from $\mathbf{z}_1$ to $\mathbf{z}_2$ can be due only to transition I_3 and I_6, we therefore only build matrices $\mathbf{K}_1(I_3)(\mathbf{z}_1,\mathbf{z}_2)$, $\mathbf{K}_2(I_3)(\mathbf{z}_1,\mathbf{z}_2)$, $\mathbf{K}_1(I_6)(\mathbf{z}_1,\mathbf{z}_2)$, and $\mathbf{K}_2(I_6)(\mathbf{z}_1,\mathbf{z}_2)$. $\mathbf{K}_1(I_3)(\mathbf{z}_1,\mathbf{z}_2)$ and $\mathbf{K}_1(I_6)(\mathbf{z}_1,\mathbf{z}_2)$ are identity matrices. $\mathbf{K}_2(I_3)(\mathbf{z}_1,\mathbf{z}_2)[\mathbf{y}_3,\mathbf{y}_5] = 1$ and the other elements of $\mathbf{K}_2(I_3)(\mathbf{z}_1,\mathbf{z}_2)$ are null. $\mathbf{K}_2(I_6)(\mathbf{z}_1,\mathbf{z}_2)[\mathbf{y}_2,\mathbf{y}_4] = 1$ and the other elements of $\mathbf{K}_2(I_6)(\mathbf{z}_1,\mathbf{z}_2)$ are null.

According to equations (12.5) we get:

$$\begin{aligned}\mathbf{G}(\mathbf{z}_1,\mathbf{z}_1) &= \mathbf{R}_1(\mathbf{z}_1,\mathbf{z}_1) \oplus \mathbf{R}_2(\mathbf{z}_1,\mathbf{z}_1)\\ \mathbf{G}(\mathbf{z}_1,\mathbf{z}_2) &= w(I_3)(\mathbf{K}_1(I_3)(\mathbf{z}_1,\mathbf{z}_2) \otimes \mathbf{K}_2(I_3)(\mathbf{z}_1,\mathbf{z}_2)) +\\ &\quad w(I_6)(\mathbf{K}_1(I_6)(\mathbf{z}_1,\mathbf{z}_2) \otimes \mathbf{K}_2(I_6)(\mathbf{z}_1,\mathbf{z}_2))\end{aligned}$$

Since $\mathrm{RPS}(\mathcal{S}) = \mathrm{RS}(\mathcal{S})$, then $\mathbf{G}(\mathbf{z}_1,\mathbf{z}_1) = \mathbf{R}(\mathbf{z}_1,\mathbf{z}_1)$ and $\mathbf{G}(\mathbf{z}_1,\mathbf{z}_2) = \mathbf{R}(\mathbf{z}_1,\mathbf{z}_2)$.

12.4.4 Hints about complexity

What is the complexity of the two-level approach with respect to the explicit generation and storage of the infinitesimal generator? There are clear limit cases: for example if all transitions are interface transitions (the system is tightly coupled), then $\mathcal{S} = \mathcal{BS} = \mathcal{LS}_i$, and it makes no sense to apply this method.

The computational cost to solve a SAM in a "traditional" manner is the sum of the cost to build the RG, the cost to build the expression of the infinitesimal generator of the associated CTMC, and the cost of solving the characteristic equation $\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0}$. The two-level method has instead a cost that is due to: the construction of the $K+1$ component models, the construction of the RG_i of each component model, the construction of the $\mathbf{R}_i(\mathbf{z},\mathbf{z}')$ and $\mathbf{K}_i(t)(\mathbf{z},\mathbf{z}')$ matrices (that may include a re-ordering of the states in the reachability sets), the cost of solving the characteristic equation $\boldsymbol{\pi} \cdot \mathbf{G} = \mathbf{0}$, when $\mathbf{G}$ is expressed as in equation (12.5). It is clear that the advantages/disadvantages of the method depend on the relative size of the reachability graphs of $\mathcal{S}$, $\mathcal{BS}$, and $\mathcal{LS}_i$.

The storage cost of the classical, direct, solution method is due to the storage of vector $\boldsymbol{\pi}$ of size $|\mathrm{RS}(\mathcal{S})|$, and of matrix $\mathbf{Q}$. Usually $\mathbf{Q}$ is stored in sparse form, so that, disregarding the diagonal, its occupation is of the same order as the number of arcs in the $\mathrm{RG}(\mathcal{S})$. The storage cost of the proposed approach is instead that of a vector $\boldsymbol{\pi}$ of size $|\mathrm{RPS}(\mathcal{S})|$, and of a number of matrices, all stored in sparse form: the total number of non-null elements, disregarding the diagonal, is of the same order as the sum of the number of arcs in the K reachability graphs $\mathrm{RG}_i(\mathcal{LS}_i)$.

In summary, the difference between the number of arcs in $\mathrm{RG}(\mathcal{S})$ and the sum of the number of arcs in the K $\mathrm{RG}_i(\mathcal{LS}_i)$ is what makes the method applicable in cases in which a direct solution is not possible, due to the lack of memory to store $\mathbf{Q}$.

12.5 Annotated bibliography

A reader interested in knowing the basis of the tensor algebra should read [14], while a basis for tensor algebra and (in)dependent composition of Markov chain can be found in [28]¿

Complexity plays a relevant role in tensor algebra based method: while the storage advantages are obvious, the time is less intuitive. A discussion of the time complexity of tensor based algorithms for the solution of the steady state

equations is contained in [5], while a discussion of the complexity of the shuffle based algorithms, for the full storage case, can be found in [30]. References [18] and [19] define a generalization of tensor product to deal with "functional rates" (a rate in a component may depend on the state of some of the other components). In the same papers the complexity of this extension is discussed in details.

Despite the fact that almost all papers deal with steady state solution, this is an issue that is somewhat orthogonal: as shown in [23] the method can be successfully applied to transient analysis as well.

Single level solution are the basic method for Stochastic Automata Network (SAN) presented in [27]. The ideas presented in this work have been applied first to a simple subset of SPN called Superposed Stochastic Automata [15], and later on to the larger SGSPN class [16, 21, 22]. All the above references pose the limitation that modules should synchronize on timed transitions, a restriction that has been removed in a recent paper by Ciardo and Tilgner [8].

Two-levels solution has appeared in different context: Buchholz has applied to stochastic process algebra, queueing networks and hierarchical Petri nets [4]. The idea of building an abstract view as a basis for a two levels method works very well for the MG subclass (no spurious states) [6], a method that has been generalized to DSSP in [9], and to the general GSPN case in [10]. But two levels method may work even if there is no partition given: the work in [3] shows how to build automatically a two levels hierarchy starting from any state space, while a more recent improvement [2] shows how this can be done at the net level.

The flat approach has been applied to the solution of GSPN where some transitions can have a phase type distribution of the delay in [20]. The application of the two level method to the same problem proved more advantageous [17], since it allows a precise characterization of the state space (no spurious states).

But new techniques should have supporting tools: Plateau implemented a first version of the solution of SAN in [29], while, very recently, Kemper has implemented in a tool the solution of SGSPN models [24]. The SGSPN solution module is integrated into the environment Toolbox [25, 26].

Bibliography

[1] G. Balbo, S. Donatelli, and G. Franceschinis. Understanding parallel programs behaviour through Petri net models. *Journal of Parallel and Distributed Computing*, 15, October 1992.

[2] P. Buchholz, and P. Kemper. Hierarchical reachability graph generation for Petri nets, To appear in "Performance Evaluation Review".

[3] P. Buchholz. Hierarchical structuring of Superposed GSPN. In *Proc. of the* 7^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 11–90, Saint Malo, France, June 1997. IEEE-CS Press.

[4] P. Buchholz. A hierarchical view of GCSPN's and its impact on qualitative and quantitative analysis. *Journal of Parallel and Distributed Computing*, 15(3):207–224, July 1992.

[5] P. Buchholz, G. Ciardo, S Donatelli, and P. Kemper. Complexity of Kronecker operations on sparse matrices with applications to the solution of Markov models. Icase report 97-66, Institute for Computer Applications in Science and Engeneering, Hampton, VA, 1997.

[6] P. Buchholz and P. Kemper. Numerical analysis of stochastic marked graphs. In *Proc.* 6^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 32–41, Durham, NC, USA, October 1995. IEEE-CS Press.

[7] G. Ciardo and A. S. Miner. Storage alternatives for large structured state spaces. In R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, editors, *Proc. 9th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, LNCS 1245, pages 44–57, Saint Malo, France, June 1997. Springer-Verlag.

[8] G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. ICASE Report 96-35, Institute for Computer Applications in Science and Engineering, Hampton, VA, May 1996.

[9] J. Campos, S. Donatelli, and M. Silva. Structured solution of stochastic DSSP systems. In *Proc. of the* 7^{th} *Intern. Workshop on Petri Nets and*

Performance Models, pages 91–100, Saint Malo, France, June 1997. IEEE-CS Press.

[10] J. Campos, S. Donatelli, and M. Silva. Structured solution of Asynchronously Communicating Stochastic Modules. submitted for pubblication, 1998.

[11] G. Chiola, S. Donatelli, and G. Franceschinis. GSPN versus SPN: what is the actual role of immediate transitions? In *Proc. 4^{th} Intern. Workshop on Petri Nets and Performance Models*, pages 20–31, Melbourne, Australia, December 1991. IEEE-CS Press.

[12] J. M. Colom. *Análisis Estructural de Redes de Petri, Programación Lineal y Geometría Convexa.* PhD thesis, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, Spain, June 1989. Research Report. GISI-RR-89-11. In Spanish.

[13] J. M. Colom and M. Silva. Improving the linearly based characterization of P/T nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 113–145. Springer-Verlag, Berlin, 1991.

[14] M. Davio. Kronecker products and shuffle algebra. *IEEE Transactions on Computers*, 30(2):116–125, 1981.

[15] S. Donatelli. Superposed stochastic automata: A class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–36, 1993.

[16] S. Donatelli. Superposed generalized stochastic Petri nets: Definition and efficient solution. In R. Valette, editor, *Proc. of the 15^{th} Intern. Conference on Applications and Theory of Petri Nets*, volume 815 of *Lecture Notes in Computer Science*, pages 258–277. Springer-Verlag, Berlin Heidelberg, 1994.

[17] S. Donatelli, S. Haddad, and P. Moreaux. Structured characterization of the Markov Chain of phase-type SPN. To appear in *Proc. 10th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Palma de Mallorca, September 98.

[18] P. Fernandes, B. Plateau, and W. J. Stewart. Efficient descriptor-vector multiplication in stochastic automata networks. J.ACM (to appear).

[19] P. Fernandes, B. Plateau, and W. J. Stewart. Numerical issues for stochastic automata networks. INRIA research report no 2938, July 1996 (available by ftp from ftp.inria.fr).

[20] S. Haddad, P. Moreaux, and G. Chiola. Efficient handling of phase-type distributions in generalized stochastic Petri nets. In *Proc. of the 18th International Conference on Application and Theory of Petri Nets*, number 1248

in LNCS, pages 175–194, Toulouse, France, June 23–27 1997. Springer–Verlag.

[21] P. Kemper. Numerical analysis of superposed GSPN. *IEEE Transactions on Software Engineering*, 22(4):615–628, September 1996.

[22] P. Kemper. *Superposition of generalized stochastic Petri nets and its impact on performance analysis.* PhD thesis, Universität Dortmund, 1996.

[23] P. Kemper. Transient analysis of superposed GSPNs. In *7-th International Conference on Petri Nets and Performance Models - PNPM97*, pages 101–110. IEEE Computer Society, 1997.

[24] P. Kemper. SupGSPN Version 1.0 - an analysis engine for superposed GSPNs. Technical report, Universität Dortmund, 1997.

[25] P. Kemper F. Bause, P. Buchholz. A toolbox for functional and quantitative analysis of deds. Technical Report 680, Universität Dortmund, 1998.

[26] P. Kemper F. Bause, P. Buchholz. A toolbox for functional and quantitative analysis of deds. In *Short paper at the PERFORMANCE TOOLS'98 - 10th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation Palma de Mallorca, Spain*, 1998.

[27] B. Plateau and K. Atif. Stochastic automata network for modeling parallel systems. *IEEE Transactions on Software Engineering*, 17(10):1093–1108, 1991.

[28] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proc. 1985 SIGMETRICS Conference*, pages 147–154, Austin, TX, USA, August 1985. ACM.

[29] B. Plateau. PEPS: A package for solving complex Markov models of parallel systems. In R. Puigjaner and D. Poiter, editors, *Modeling techniques and tools for computer performance evaluation*, pages 291–306. Plenum Press, New York and London, 1990.

[30] W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, 1994.

Chapter 13

Symmetries and exact lumping in SWN

In this chapter the SWN formalism is defined as a timed extension of the WN formalism. The type of timing allowed corresponds to that already defined for GSPNs, i.e., the unfolding of a SWN model is a GSPN model, however some constraints are imposed to preserve the behavioural symmetry properties of WNs also in the time domain. The symmetric timed behaviour can then be exploited at the CTMC solution level by applying the lumping technique.

The chapter is organized as follows: Section 13.1 contains the definition of SWNs and a discussion about the assignment of rates and weights to SWN transitions which involve some subtle problems not present in the context of GSPNs; in Section 13.2 it will be shown that the CTMC corresponding to the RG of a SWN satisfies the strong lumpability conditions with respect to the aggregation induced by the SRG algorithm and that a lumped CTMC can be directly computed from the information contained in the SRG, moreover any performance index that can be computed by solving the complete CTMC can be obtained from the aggregated CTMC as well; some cases in which it is possible to apply steady state analysis despite the sufficient condition for RG ergodicity is not satisfied are also discussed; finally in Section 13.3 it will be discussed the relation between SWNs decolourization and lumpability.

13.1 Formal definition of SWN

Definition 13.1 (Stochastic Well Formed Nets) *a* Stochastic Well-Formed Net *(SWN) is a nine-tuple*

$$\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{Inh}, \mathbf{pri}, \mathcal{C}, cd, \mathbf{w} \rangle$$

where P, T, **Pre**, **Post**, **Inh**, **pri**, $\mathcal{C}$ *and* cd *are define as in WNs (see Chap-*

ter 2), while $\mathbf{w}$ *is a* T *indexed vector of functions:*

$$\mathbf{w}[t] : cd(t) \times (\bigotimes_{p \in P} Bag(cd(p))) \rightarrow \mathbb{R}^+$$

defining a rate *for each (exponential) timed transition instance and a* weight *for each immediate transition instance. The transition rate/weight function may depend on the transition instance and it may be marking dependent, however, the type of dependence on the instance or on the marking must be such that it allows to maintain in the time domain the symmetry properties holding at the qualitative behaviour level. More precisely, the definition of function* $\mathbf{w}[t]$ *can be expressed as:*

case
$cond_1$ *:* r_1
$cond_2$ *:* r_2
...
$cond_n$ *:* r_n
default *:* r_{def}

Where $cond_i$ *is a boolean expression comprising* standard predicates *on the transition colour instance and predicates on the marking: the predicates on the marking must be such that* the specific colours of tokens in each place are never used*. Allowed predicates on markings include those testing the presence of tokens whose colour tuple can be obtained by composition of the transition instance parameters values, or those testing the presence of tokens whose colour tuple comprises elements belonging to specific static subclasses.*

For example, conditions $cond_i$ cannot contain predicates of the type $\mathbf{m}[p] \supseteq \{3\langle c_1, c_2\rangle\}$, where $\langle c_1, c_2\rangle \in cd(p)$ while it is allowed a predicate like $\mathbf{m}[p] \supseteq \{3\langle x, y\rangle\}$ where $x, y \in var(t)$, or like $\mathbf{m}[p] \supseteq \{3\langle x, y\rangle : (d(x) = C_{i,j}) and (d(y) = C_{h,k})\}$ where $x, y \notin var(t)$.

Observe that a marking dependent transition rate function satisfying the above constraints is well defined for both *ordinary* and *symbolic* markings.

A more constrained definition for transition rates, that satisfies the above symmetry requirements, has been defined[5] for practical reasons: it can be easily implemented in a tool and is powerful enough to model most interesting cases of dependency on the transition instance and on the marking. To this purpose we need to introduce the following notation: for any $c =< c_1, \ldots, c_k > \in cd(t)$ we define $\tilde{c} =< \tilde{c}_1, \ldots, \tilde{c}_k >$ such that $\tilde{c}_j = C_{i,q}$ iff $c_j \in C_{i,q}$. Finally we define the *static partition of a marking* $\mathbf{m}$ denoted $\tilde{\mathbf{m}}[p] \in Bag(\tilde{cd(p)})$ as follows:

$$\tilde{\mathbf{m}}[p](\tilde{c}) \;=\; \sum_{c' \,:\, \tilde{c'} = \tilde{c}} \mathbf{m}[p](c')$$

The static partition of a marking represents, for each place and for each Cartesian product of static subclasses, the number of tuples in the place that belong to the same Cartesian product of static subclasses.

Property 13.2 *Static partition of symbolic markings.*

$$\forall \mathbf{m},\ \mathbf{m}' \in \widehat{\mathbf{m}},\quad \forall p \in P,\quad \tilde{\mathbf{m}}[p] = \tilde{\mathbf{m}}'[p]$$

Hence we can define the static partition of a symbolic marking as well and denote it $\widehat{\widetilde{\mathbf{m}}}$.

We can now define $\mathbf{w}[t]$ as a function:

$$\mathbf{w}[t] \quad : \quad \tilde{cd}(t) \times \ Bag(\tilde{cd}(p_1)) \times Bag(\tilde{cd}(p_2)) \times \ldots Bag(\tilde{cd}(p_{|P|})) \ \longrightarrow \ \mathbb{R}^+$$

Single, multiple and infinite server semantics in SWNs Let us discuss some new options that the SWN formalism may allow in the (marking dependent) timing semantics of the transitions with respect to the GSPN formalism, and highlight some frequent errors in the timing specification of coloured models (often due to the fact that the modeller doesn't realize the actual structure of the unfolding and has not a correct perception of the instances of a given transition that can be concurrently enabled).

Before starting we need to extend the definition of *enabling degree* to transition instances of SWNs, that is the number of simultaneous enablings of a transition instance $\langle t, c\rangle$ at a given marking $\mathbf{m}$:

$$\mathbf{e}(\mathbf{m})[\langle t, c\rangle] = \max\{k \in \mathbb{N}_+ \mid \mathbf{m} \supseteq k \cdot \mathbf{Pre}[P, t](c)\}.$$

In GSPNs, it is possible to assign a number of servers to a given transition so that single, multiple or infinite server semantics can be defined for each timed transition. As explained in Chapter 2 if all transitions are implicitly defined as self-concurrent (i.e., if by default infinite servers are assigned to each transition) it is possible to use self-loops to model transitions with a limited number k of servers.

Extending these concepts to SWNs is straightforward, since all the above considerations have now to be applied to the single transition instances. Of course, because of the limitations imposed on the marking dependent definition of transition rates, the type of semantics associated with the different instances of a given transition will be naturally consistent with the symmetry properties of the model. As for GSPNs, it is still possible to assign infinite server semantics by default, and to use self loops to enforce a single or multiple server semantics for some or all instances of a given transition. There exists a further possibility in SWNs that does not apply to GSPNs, namely limiting the concurrency of a SWN transition rather than limiting the self-concurrency of each instance of that transition, that is assigning a limited number of servers to the whole set of enabled instances of that transition[1].

Let us consider the multicomputer PLC example once more: starting from its functional definition through a WN model (see Chapter 2) we can derive a SWN model by assigning weights to transitions: weights of timed transitions are interpreted as durations, weights of immediate transitions are interpreted as probabilities.

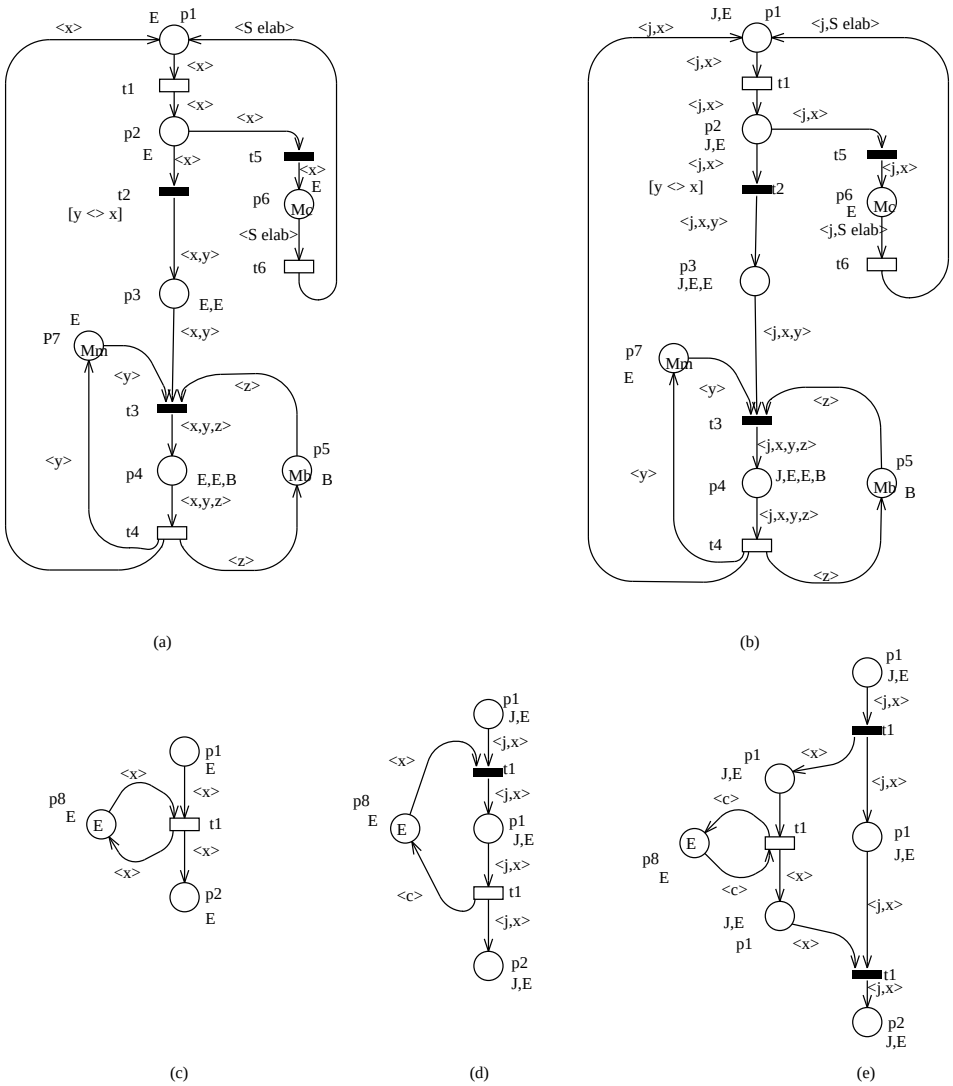


Figure 13.1: Assigning time to the multicomputer PLC model.

transition	rate/weight	semantics/priority
Timed Transitions		
t_1	α_1	single-server per colour
t_4	α_4	single-server per colour
t_6	α_6	infinite server
Immediate Transitions		
t_2	β_2	1
t_3	β_3	1
t_5	β_5	1

Table 13.1: Timing of the multicomputer PLC SWN model.

The timed transitions of the model in Fig. 13.1(a) are t_1, t_4, and t_6 representing respectively a local computation, a remote memory access, and a synchronization among all completed tasks (one on each computer) to start a new cycle. The immediate transitions of the model are t_2, t_5 and t_3. Transitions t_2 and t_5 represent a probabilistic choice used to establish when a task has completed. Transition t_3 represents the acquisition of a (remote) memory and a bus to start the service of a remote memory access request.

Assuming that the time required to perform a given operation does not depend on the specific computer/memory/bus involved (i.e., if the system is completely symmetric) then all instances of a given transition have the same associated rate/weight; if this is not the case, it is necessary to partition the basic colour classes into (static) subclasses, each representing a set of objects with homogeneous performance characteristics (e.g., we may need to partition the set of computers into two subclasses: the slow and the fast ones) and then associate different rates/weights with different transition instances based on which static subclass their parameters belong to.

Let us concentrate on the timing specification of transition t_1 in the completely symmetric case. A *rate* (say α_1) is assigned to any instance of t_1, moreover, since this transition represents a local computation it makes sense to define it as single server, meaning that just one such operation can take place on each computer at a time. In other words, the number of tokens of a given colour e_i simultaneously present into place p_1 (representing several tasks running on the same computer) does not influence the firing rate of transition instance $\langle t_1, e_i \rangle$ since the self concurrent instances are served "sequentially", one at a time. This can be enforced by explicitly adding a self loop limiting the enabling degree of transition t_1 as shown in Fig. 13.1(c). Observe that due to the initial marking defined for this model this concurrency limiting place is redundant (because p_1 is colour safe), however in this way we make the specification more complete and ensure a correct behaviour independently of the initial marking.

Now, let us assume that for some reason we need to refine the model allowing

[1] This possibility could be further generalized by defining a partition of the possible transition instances into subsets, and assigning to each subset a limited number of servers

multiple tasks (of different type) running on each computer, with the further constraint that transition t_6 can only synchronize tasks of the same type. This can be implemented by adding a new colour class $J = \{j_1, \ldots, j_{nt}\}$, redefining the colour domain of all places but p_5 and p_7 in order to add a task identifier to each token (e.g, $cd(p_1) = J \times E$), and modify the arc expressions accordingly. The refined model is shown in Fig. 13.1(b).

Observe that with this new specification, the timing semantics of transition t_1 has to be adjusted, indeed since the single server semantics (implemented through the self loop place) refers to *each instance* of a transition, the two instances $\langle t_1, \langle j_2, e_1 \rangle \rangle$ and $\langle t_1, \langle j_4, e_1 \rangle \rangle$, if enabled together proceed in parallel (due to the *race* policy) producing the effect that the speed of a computer (e_1 in the example) increases as the number of tasks running on it grows larger.

A possible solution to this problem is to enforce a *preselection* of the task to be run next for each computer, by introducing explicit acquisition of a computer through the introduction of an immediate transition as shown in Fig. 13.1(d): this implements a random order scheduling of tasks on each computer. If a processor sharing service policy is to be modelled instead, it would be useful to allow the possibility of assigning a single server semantics among all instances of t_1 with same value of parameter c. In Fig. 13.1(e) a model is shown that implements this type of timing semantics for transition t_1.

Tab. 13.1 summarizes the timing definition of the multicomputer PLC SWN model that will be used later in this chapter.

About the ECS notion and related issues in SWNs As discussed in Chapter 9, GSPN immediate transitions are partitioned into equivalence classes, called Extended Conflict Sets (ECS), which correspond to conflict situations that may be (probabilistically) solved independently from each other. Transitions belonging to different ECS cannot be in conflict in any marking. Let us here recall that ECSs are the equivalence classes of the equivalence relation obtained by the transitive closure of the Symmetric Structural Conflict relation (SSC), which is in turn defined as the (symmetric) union of the Structural Conflict (SC) and Indirect Structural Conflict (ISC) relations. For the sake of simplicity in the following we consider only nets where Indirect Structural Conflicts never occur. The SSC relation gives necessary conditions for two transitions to be in conflict in some marking.

The relevance of the ECS notion is twofold. On one hand, it provides a full knowledge of potential conflicts among immediate transitions, driving the modeller to a correct assignment of the weights used to probabilistically solve such conflicts. On the other hand, it allows an efficient construction of the RG, in terms of number of vanishing markings, making it possible to fix and arbitrary firing order among ECS that is used to establish a firing order among concurrently enabled immediate transitions belonging to different ECSs. This avoids to consider all possible firing interleavings when building the model reachability graph; it is important to remark that this technique is applicable only under the assumption of absence of asymmetric confusion.

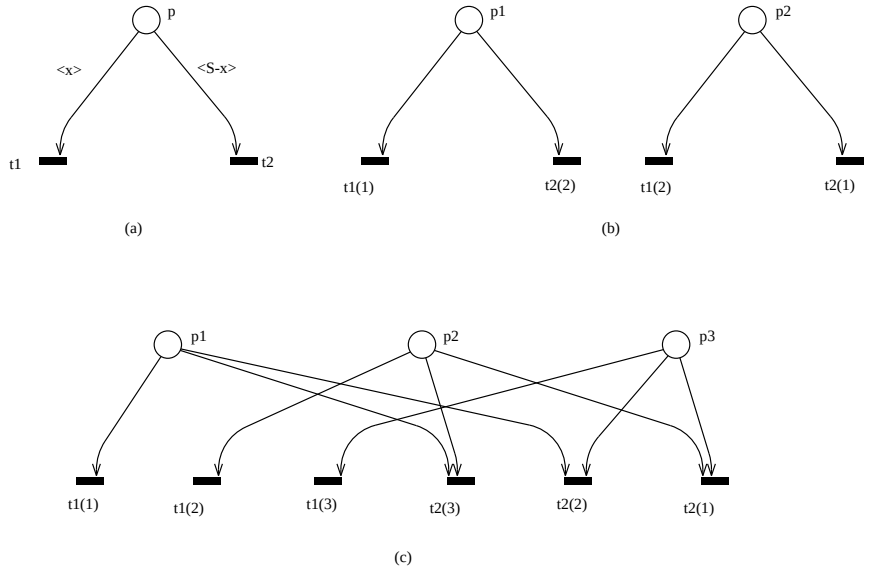


Figure 13.2: An example of structural conflict.

The possibility of extending the ECS notion, as well as those of ME, SC, etc., on which it relies, to SWNs is an important issue, strongly related with the ability of correctly specifying the probabilistic conflict resolution of immediate transitions. In SWNs, and more generally in CPNs, all structural relation notions refer to transition instances. As already observed, it might be difficult for the modeller to perceive the actual structure of the unfolding, i.e., the actual potential conflicts of the underlying unfolded model.

Considering the model in Fig. 13.1(a) we can observe that actually only instances $\langle t_2, e_i \rangle$ and $\langle t_5, e_i \rangle$ are in conflict with each other, while instances $\langle t_2, e_i \rangle$ and $\langle t_5, e_j \rangle$, $i \neq j$ are not. Moreover, different instances of t_2 are not in conflict, the same applies to different instances of t_5. Different instances of t_3 might be in conflict with each other: for example instances $\langle t_3, \langle e_i, e_j, b_1 \rangle \rangle$, $\langle t_3, \langle e_i, e_j, b_2 \rangle \rangle$ are in conflict because they both try to withdraw a token $\langle e_i, e_j \rangle$ in p_3; moreover instances $\langle t_3, \langle e_i, e_j, b_k \rangle \rangle$, $\langle t_3, \langle e_l, e_m, b_k \rangle \rangle$ are in conflict because they both try to withdraw a token $\langle b_k \rangle$ in p_5.

Let us now consider the simple net in Fig. 13.2(a), whose unfoldings are depicted in Fig. 13.2(b) and Fig. 13.2(c), for the case $|cd(p)| = 2$ and $|cd(p)| = 3$, respectively. We can observe that actually only instances $\langle t_1, c_i \rangle$, $\langle t_2, c_j \rangle$, $i \neq j$ are in structural conflict in both cases (b) and (c). In case (c) different instances of t_2 might be also in conflict with each other, because they try to withdraw a common token in p. An important consequence is that in case (b) there are actually two independent probabilistic choices, i.e. two different ECSs, while

in case (c), there is a unique ECS involving all transition instances, since they are obtained as transitive closure of the SC relation (according with the ECS definition for GSPNs). Supposing we want to assign colour dependent weights to transition t_1 (for the moment disregarding model symmetries), in case (b) the weight associated with $\langle t_1, c_1 \rangle$ and $\langle t_1, c_2 \rangle$ are reciprocally independent from the probabilistic conflict resolution point of view, while in case (c) they are not.

The simplest solution that can be adopted when defining weights in SWN models is that of considering only one ECS containing all transition instances, i.e. in each vanishing marking the firing probabilities are computed by normalizing the weights of all the enabled transition instances. A minor refinement is that of considering the ECSs of the GSPN obtained by decolouring a SWN (arc and marking expressions are substituted by the corresponding cardinalities). Notice that adopting either one of these solutions does not make any difference in the example of Fig. 13.2 (in both cases (b) and (c), a single conflict set is obtained). In the example of Fig. 13.1 instead the second approach would give two ECS, one containing all instances of t_2 and t_5, and another one containing all instances of t_3.

The previous simple solutions provide the modeller with a limited support in the weight assignment task, often leading to erroneous or doubtful probabilistic resolutions of conflicts in SWN models. A more refined ECS indication may substantially contribute to solve uncertain situations, allowing the modeller to recognize that some transition instances are only apparently in conflict or, on the opposite, that apparently independent instances are actually in conflict, and making it easier to assign proper weights.

The natural way to obtain precise indication on the ECS of a model is to pass through the underlying unfolded model, but this approach reveals to be very inefficient, and often unfeasible in practice. More efficient ways of detecting structural conflicts (and more generally, structural relations) between transition instances, based e.g. on partial unfoldings or on high-level structural definitions need to be investigated.

The other way of attacking this problem is to define a symbolic representation of ECS and an algorithm to derive such symbolic ECS directly from the high-level model definition. Symbolic definitions of structural relations were proposed in [7, 8]. The definitions proposed in [8] are quite general, while in [7] they were proposed for a subclass of WNs, namely Unary Regular Nets. These definitions are based on the linear colour functions inscribing the net arcs, and they describe in a compact way the structural relations between transitions of the unfolding.

In [7] a symbolic relation between transitions t_1 and t_2 is defined by a function $f : \mathcal{P}(cd(t_1)) \rightarrow \mathcal{P}(cd(t_2))$ (where $\mathcal{P}(E)$ is the power-set associated with the domain E), which gives for each color instance $c_1 \in cd(t_1)$ all the color instances of t_2 wich are in structural relation (R_f) with c_1 (i.e. $\langle t_2, c_2 \rangle \; \mathrm{R}_f \; \langle t_1, c_1 \rangle$ iff $c_2 \in f(c_1)$). If stability conditions hold on colour functions (the most important is the closure property w.r.t. a set of function operators such as union, intersection, difference, composition), such symbolic relations may be computed in an efficient way.

Let us consider again the net in Fig. 13.2(a). The structural conflict between t_1 and t_2 is expressed by the symbolic relation $\mathrm{SC}(t_2, t_1)$ defined by $\mathbf{Pre}[p, t_2]_{\mathcal{P}}{}^t \circ \mathbf{Pre}[p, t_1]_{\mathcal{P}}$ (where $f_{\mathcal{P}} : \mathcal{P}(E) \to \mathcal{P}(F)$ is the power-set extension of the function $f : \mathrm{Bag}(E) \to \mathrm{Bag}(F)$, ${f_{\mathcal{P}}}^t : \mathcal{P}(F) \to \mathcal{P}(E)$ is the transposed of $f_{\mathcal{P}}$, i.e $x \in {f_{\mathcal{P}}}^t(y)$ iff $y \in f_{\mathcal{P}}(x)$, and $\circ$ denotes the composition of functions). In other words the expression $\mathrm{SC}(t_2, t_1)(c_1)$ represents the color instances of t_2 whose firing withdraws some colour present in the expression obtained by evaluating the input function from p to t_1 on the instance c_1.

Denoting hereafter colour functions with the corresponding arc expressions, and recalling that S represents the diffusion function on a (sub)class and that any variable symbol represents the identity function on the associated colour class, we obtain: $\mathrm{SC}(t_2, t_1) = \langle S - x \rangle_{\mathcal{P}}{}^t \circ \langle x \rangle_{\mathcal{P}} = \langle S - x \rangle_{\mathcal{P}}$, i.e. a color instance of t_1 is in conflict with all the color instances of t_2 but itself (and vice-versa). If we consider the symmetric, reflexive and transitive closure of SC, defined in [7] and based on the usual function operators, we obtain a symbolic expression for the ECS relation, which turns out to be different in the cases (b) and (c). In fact, concerning the conflict between instances of transition t_2 (which have to be considered for correctly computing the transitive closure), we obtain $\mathrm{SC}(t_2, t_2) = \langle S - x \rangle_{\mathcal{P}}{}^t \circ \langle S - x \rangle_{\mathcal{P}}$, which is equal to $\langle S \rangle_{\mathcal{P}}$ if $|cd(p)| \geq 2$, while it is equal to $\langle x \rangle_{\mathcal{P}}$ if $|cd(p)| = 2$. Observe that the computation of symbolic ECSs may depend on the cardinality of color domains.

Extending this kind of computation of symbolic structural relations to SWNs (or at least, to a significant SWN subclass) is still an open question. The fulfillment of stability condition must be checked for a larger class of functions. The possibility of defining symbolic ECSs could also have a deep impact on the efficiency of the SRG generation. Let us consider the polling system model described in Fig. 7.12, and in particular the transitions ts and tw. With simple arguments we can derive that $\mathrm{SC}(ts, tw) = \langle x, z \rangle_{\mathcal{P}}$ (where $\langle x, z \rangle$ is the identity function on the domain $Q \times R$), meaning that a color instance of tw may be in conflict only with the same instance of ts. Analogously, considering only mutual exclusion due to inhibitor arcs we obtain $\mathrm{ME}(ts, tw) = \langle x, S \rangle_{\mathcal{P}}$, i.e. a color instance (c_1, c_2) of tw is in mutual exclusion with all the color instances of ts whose first component is c_1. Since two instances may be in conflict only if they are not mutually exclusive, we derive that $\mathrm{ECS}(ts, tw) = \emptyset$, i.e. the instances of ts and tw belong to different ECSs, and the symbolic firings of these transitions may occur in an arbitrary order. Instead, instances of ts may be in conflict with each other: denoting with $Proj_{(Q)}(\langle x, z \rangle) : Q \times R \to Q$ the projection on the domain Q of the function $\langle x, z \rangle$, we obtain (considering the input place pq), $\mathrm{SC}(ts, ts) = Proj_{(Q)}(\langle x, z \rangle)_{\mathcal{P}}{}^t \circ Proj_{(Q)}(\langle x, z \rangle)_{\mathcal{P}} = \langle x, S \rangle_{\mathcal{P}}$, i.e. different instances of ts having the same first component may be in conflict.

In the discussion above we have disregarded the symmetry constraints imposed on the definition of SWN rates and weights, however symbolic ECS computation is still useful when this constraints are introduced. In this case, the weight must be defined as a function of the static partition of the colour classes into subclasses (for simplicity, we here do not consider the case of marking dependent weights). Once again, let us consider the example of Fig. 13.2, and

suppose that $cd(p)$ is partitioned into two static subclasses, C_1 and C_2. From the symbolic ECS expressions previously computed, we can easily derive that in case (c) ($|cd(p)| = 3$) the symbolic instances $\langle t_1, Z_{C_1} \rangle$ and $\langle t_1, Z_{C_2} \rangle$ are in conflict (i.e. the corresponding ordinary instances belong to the same ECS), while in case (b) ($|cd(p)| = 2$) they are not. Consequently, the weights assigned to these symbolic instances may have a reciprocal influence (from the probabilistic point of view) in case (c), while they are independent in case (b). In order to provide the modeller with an effective support, symbolic ECS should be expressed in a structured form which outlines, for a given static partition of immediate transition instances, all possible conflicting symbolic instances.

13.2 SRG aggregation and lumpability

In this section we shall show that the aggregation induced on the CTMC describing the behaviour in time of a given SWN model satisfies the *strong lumpability condition*[12] defined hereafter:

Definition 13.3 (Strong lumpability of CTMC) *Let $S = \{s_1, \ldots, s_n\}$ be the set of states of a CTMC, $\mathbf{Q}$ be the corresponding infinitesimal generator and $A = \{A_1, \ldots, A_k\}$ be a partition of S into* aggregates. *The strong lumpability condition is defined as:*

$$\forall A_i, A_j \in A, \quad \forall s_{i1}, s_{i2} \in A_i, \sum_{s_k \in A_j} q_{i1,k} = \sum_{s_k \in A_j} q_{i2,k}$$

Intuitively if a CTMC satisfies the strong lumpability condition, the probability of moving from one aggregate A_i into an aggregate A_j does not depend on the specific state of A_i, as illustrated in Fig. 13.3. As a consequence, if the modeller is just interested in the high-level behaviour (evolution through the *macro states* corresponding to the aggregates) of the system, a smaller CTMC can be used to get the solution, with as many states as the number of aggregates. The infinitesimal generator $\mathbf{Q}'$ of the reduced CTMC is defined as follows:

$$q'[i,j] = \sum_{s_k \in A_j} q_{h,k}$$

where $s_h \in A_i$ is any state of aggregate A_i.

Let us show that the state aggregation induced by the SRG algorithm on the CTMC corresponding to the RG of a SWN model satisfies the strong lumpability condition.

Proposition 13.4 *Let $\mathbf{m_{i,k}}$ be one of the ordinary markings in the equivalence class represented by the symbolic marking $\widehat{\mathbf{m_i}}$. Due to the symmetry constraints imposed on the transition rates, the probability of going from $\mathbf{m_{i,k}}$ to $\mathbf{m_{j,q}}$ is the same as the probability of going from $\mathbf{m_{i,s(k)}}$ to $\mathbf{m_{j,s(q)}}$ (see Fig. 13.4), i.e.:*

$$q_{(i,k)(j,q)} = q_{(i,s(k))(j,s(q))}$$

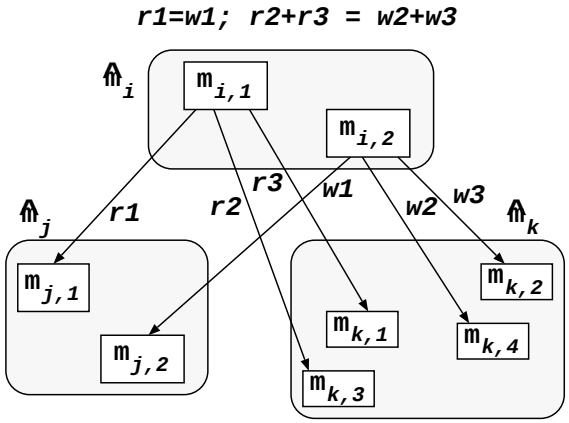


Figure 13.3: Strong lumpability condition for CTMC

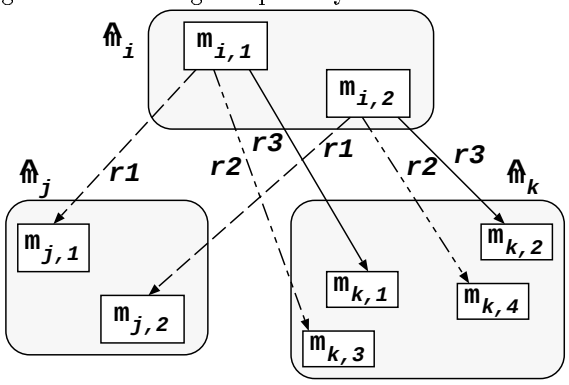


Figure 13.4: Condition satisfied by the CTMC of a SWN

Hence, it is easy to show that the lumping condition is satisfied.

Proposition 13.5 *The embedded Markov chain verifies the lumping condition, i.e.:*

$$\forall \widehat{\mathbf{m_i}}, \widehat{\mathbf{m_j}}, \quad \forall \mathbf{m_{i,k}} \in \widehat{\mathbf{m_i}}, \quad \forall s \in \xi, \qquad \sum_{\mathbf{m_{j,q}} \in \widehat{\mathbf{m_j}}} q_{(i,k)(j,q)} = \sum_{\mathbf{m_{j,q}} \in \widehat{\mathbf{m_j}}} q_{(i,s(k))(j,q)}$$

As a consequence, a reduced system can be computed whose variables are the symbolic marking probabilities and whose coefficients $q'_{i,j}$ are defined as: $q'_{i,j} = \sum_{\mathbf{m_{j,q}} \in \widehat{\mathbf{m_j}}} q_{(i,k)(j,q)}$. We show now that these coefficients can be computed using the information contained in the SRG alone.

The definition of the elements of the infinitesimal generator of the embedded Markov chain corresponding to the complete RG is reported hereafter for the reader convenience, a more detailed explanation on how it is derived can be found in Chapter 9:

$$q_{(i,k)(j,q)} = \frac{\sum_{\langle t,c\rangle \,:\, \mathbf{m_{i,k}} \xrightarrow{\langle t,c\rangle} \mathbf{m_{j,q}}} \mathbf{w}[t](c, \mathbf{m_{i,k}})}{\sum_{\langle t,c\rangle \,:\, \mathbf{m_{i,k}} \xrightarrow{\langle t,c\rangle}} \mathbf{w}[t](c, \mathbf{m_{i,k}})}$$

It is now necessary to extend the definition of the functions $\mathbf{w}[t], t \in T$ to the symbolic firing instances. Let us denote such extension $\widehat{\mathbf{w}}$:

$$\widehat{\mathbf{w}}[t](\lambda, \mu, \widehat{\mathbf{m}}) = \mathbf{w}[t](c, \mathbf{m}), \forall c \in cd(t), \mathbf{m} \in \widehat{\mathbf{m}} \; s.t. \; \lambda, \mu = \eta(c) \; and \; \widehat{\mathbf{m}} = \eta(\mathbf{m})$$

where $\eta(c)$ is a *valid assignment* of subsets of colours to dynamic subclasses[2] This equality is guaranteed to hold for all $c \in cd(t) \;\; s.t. \; \lambda, \mu = \eta(c)$ due to the symmetry property of functions $\mathbf{w}[t]$ (otherwise $\widehat{\mathbf{w}}[t](\lambda, \mu, \widehat{\mathbf{m}})$ would not be well defined).

It is now possible to define the transition rate between symbolic markings:

$$\widehat{q}_{i,j} = \sum_{\mathbf{m_{j,q}} \in \widehat{\mathbf{m_j}}} q_{(i,k)(j,q)} = \frac{\sum_{\langle t,\lambda,\mu\rangle \,:\, \widehat{\mathbf{m_i}} \xrightarrow{\langle t,\lambda,\mu\rangle} \widehat{\mathbf{m_j}}} \widehat{\mathbf{w}}[t](\lambda, \mu, \widehat{\mathbf{m_i}}) |\widehat{\mathbf{m_i}} \xrightarrow{\langle t,\lambda,\mu\rangle}|}{\sum_{\langle t,\lambda,\mu\rangle \,:\, \widehat{\mathbf{m_i}} \xrightarrow{\langle t,\lambda,\mu\rangle}} \widehat{\mathbf{w}}[t](\lambda, \mu, \widehat{\mathbf{m_i}}) |\widehat{\mathbf{m_i}} \xrightarrow{\langle t,\lambda,\mu\rangle}|} \quad (13.1)$$

where $|\widehat{\mathbf{m_i}} \xrightarrow{\langle t,\lambda,\mu\rangle}|$ (defined in Chapter 7) is the number of arcs that are *grouped* into the symbolic firing instance $\langle t, \lambda, \mu\rangle$ enabled in $\widehat{\mathbf{m_i}}$[3].

[2] Observe that actually η is a set $\{\eta_1, \ldots, \eta_n\}$ of valid assignments η_i, one for each colour class C_i in the model, and that it is possible to define η for application on tuples of colours as follows: $\eta(\langle c_{(1)}, \ldots, c_{(k)}\rangle) = \langle \eta_{(1)}(c_{(1)}), \ldots, \eta_{(k)}(c_{(k)})\rangle$, moreover it is possible to extend its definition to apply it also to colour tuples representing transition instances as follows: $\lambda(i) = \eta_{(i)}(c_{(i)}), \mu(i) = \#$of distinguished $c_{(j)}$ s.t.$1 \leq j \leq i, C_{(j)} = C_{(i)}, \eta_{(j)}(c_{(j)}) = \lambda(i)$.

[3] Note that this is not the overall number of ordinary transition instances represented by the symbolic instance λ, μ but rather the number of ordinary instances in λ, μ firable in *each* single ordinary marking $\mathbf{m_i} \in \widehat{\mathbf{m_i}}$

A lumped CTMC can thus be derived *directly* from the SRG, so that the steady state probability of symbolic markings can be computed (provided that the lumped CTMC is ergodic).

Actually from the symbolic markings steady state probability it is possible to compute the steady state probability of each ordinary marking: this is due to the fact that *all ordinary markings in a symbolic marking have equal probability.* This is a consequence of Proposition 13.4 and the following proposition:

Proposition 13.6

$$\forall \widehat{\mathbf{m_i}}, \quad \forall \mathbf{m_{j,q}} \in \widehat{\mathbf{m_j}}, \quad \forall s \in \xi, \qquad \sum_{\mathbf{m_{i,k}} \in \widehat{\mathbf{m_i}}} q_{(i,k)(j,q)} = \sum_{\mathbf{m_{i,k}} \in \widehat{\mathbf{m_i}}} q_{(i,k)(j,s_q)}$$

Intuitively the above proposition means that the overall rate from any symbolic marking $\widehat{\mathbf{m_i}}$ to each ordinary marking $\mathbf{m_{j,k}} \in \widehat{\mathbf{m_j}}$ is the same.

Example 13.1 Let us discuss the effect of the SRG aggregation on the size of the multicomputer PLC state space, and explain how the lumped CTMC is derived from the SRG and the performance results obtained from it.

The aggregation achieved in the multicomputer PLC SWN model with three computers and two busses is illustrated in the Tabs. 13.2 and 13.3 of tangible and vanishing symbolic states: 17 tangible and 22 vanishings symbolic markings against 80 tangible and 123 vanishing ordinary markings. The reduction is even more relevant if the busses were not decoloured: same number of symbolic markings as above against 152 tangible and 210 vanishing ordinary markings (although the size of the SRG is the same for the two models, the computation time of the SRG on the latter model is twice as much the computation time of the SRG on the former model).

The SRG is shown in Fig. 13.5; by elimination of the vanishing markings, it is possible to get the TSRG, shown in Fig. 13.6, which is isomorphic to the lumped CTMC representing the stochastic behaviour of the SWN.

Tab. 13.4 shows a number of state transitions in the TSRG with the corresponding symbolic transition instances sequence[4] and the associated transition rate in the CTMC. In Fig. 13.7, an example is shown of the effect of lumping on transitions rates: the transition rate from the aggregate marking $\widehat{\mathbf{m_1}}$ to the aggregate marking $\widehat{\mathbf{m_2}}$ is equal to the sum of rates from (any)one marking in $\widehat{\mathbf{m_1}}$ to all states in $\widehat{\mathbf{m_2}}$. Since from marking $\mathbf{m_{1,1}}$ (the single ordinary marking belonging to $\widehat{\mathbf{m_1}}$) there are six transitions towards (states in) $\widehat{\mathbf{m_2}}$, each with transition rate $\alpha_1 \frac{\beta_2}{2\beta_2+\beta_5}$, the transition rate from $\widehat{\mathbf{m_1}}$ to $\widehat{\mathbf{m_2}}$ is $6\alpha_1 \frac{\beta_2}{2\beta_2+\beta_5}$. Observe that this is the same rate obtained from the SRG using Equation 13.1: the rate associated with symbolic firing (see Tab. 13.4) $\widehat{\mathbf{m_1}} \xrightarrow{\langle t_1, \lambda, \mu \rangle}$, where λ and μ are simple defined as $\lambda[1] = 0,\ \mu[1] = 1$, is

$$|\widehat{\mathbf{m_1}} \xrightarrow{\langle t_1, \lambda, \mu \rangle}|\ \widehat{\mathbf{w}}[t_1](\lambda, \mu, \widehat{\mathbf{m_1}}) = \frac{\widehat{\mathbf{m_1}}.card(Z_E^0)!}{(\widehat{\mathbf{m_1}}.card(Z_E^0)-1)!}\alpha_1 = 3\alpha_1.$$

[4] In this table the transition instances are characterized simply through a tuple of dynamic subclasses, i.e. simply through λ, because in this example the μ is always a vector of 1s (see Chapter 7).

Tangible Markings: 17 symbolic, 80 ordinary		
	Symbolic Marking	$\lvert\widehat{\mathbf{m_i}}\rvert$
$\widehat{\mathbf{m_1}}$	$p7(1 < Z_E^0 >)p5(2)p1(1 < Z_E^0 >)$ $\lvert Z_E^0\rvert = 3$	1
$\widehat{\mathbf{m_2}}$	$p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^1, Z_E^0 >)p1(1 < Z_E^0 > 1 < Z_E^2 >)$ $\lvert Z_E^2\rvert = 1\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{m_3}}$	$p6(1 < Z_E^1 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(2)p1(1 < Z_E^0 >)$ $\lvert Z_E^0\rvert = 2\lvert Z_E^1\rvert = 1$	3
$\widehat{\mathbf{m_4}}$	$p7(1 < Z_E^2 >)p4(1 < Z_E^0, Z_E^1 > 1 < Z_E^2, Z_E^0 >)p1(1 < Z_E^1 >)$ $\lvert Z_E^0\rvert = 1\lvert Z_E^2\rvert = 1\lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{m_5}}$	$p7(1 < Z_E^2 >)p4(1 < Z_E^0, Z_E^1 > 1 < Z_E^1, Z_E^0 >)p1(1 < Z_E^2 >)$ $\lvert Z_E^2\rvert = 1\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1$	3
$\widehat{\mathbf{m_6}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(1)p4(1 < Z_E^0, Z_E^2 >)p1(1 < Z_E^1 >)$ $\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1\lvert Z_E^2\rvert = 1$	6
$\widehat{\mathbf{m_7}}$	$p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^2, Z_E^0 >)p3(1 < Z_E^1, Z_E^0 >)p1(1 < Z_E^0 >)$ $\lvert Z_E^0\rvert = 1\lvert Z_E^2\rvert = 1\lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{m_8}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^1, Z_E^0 >)p1(1 < Z_E^0 >)$ $\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1\lvert Z_E^2\rvert = 1$	6
$\widehat{\mathbf{m_9}}$	$p6(1 < Z_E^1 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(2)p1(1 < Z_E^0 >)$ $\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 2$	3
$\widehat{\mathbf{m_{10}}}$	$p7(1 < Z_E^2 >)p4(1 < Z_E^1, Z_E^0 > 1 < Z_E^2, Z_E^1 >)p3(1 < Z_E^0, Z_E^2 >)$ $\lvert Z_E^2\rvert = 1\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{m_{11}}}$	$p7(1 < Z_E^2 >)p4(1 < Z_E^1, Z_E^0 > 1 < Z_E^2, Z_E^1 >)p3(1 < Z_E^0, Z_E^1 >)$ $\lvert Z_E^2\rvert = 1\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{m_{12}}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^1 >)p4(1 < Z_E^0, Z_E^2 > 1 < Z_E^1, Z_E^0 >)$ $\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1\lvert Z_E^2\rvert = 1$	6
$\widehat{\mathbf{m_{13}}}$	$p7(1 < Z_E^2 >)p4(1 < Z_E^0, Z_E^1 > 1 < Z_E^1, Z_E^0 >)p3(1 < Z_E^2, Z_E^1 >)$ $\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1\lvert Z_E^2\rvert = 1$	6
$\widehat{\mathbf{m_{14}}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^2 >)p4(1 < Z_E^0, Z_E^1 > 1 < Z_E^1, Z_E^0 >)$ $\lvert Z_E^1\rvert = 1\lvert Z_E^0\rvert = 1\lvert Z_E^2\rvert = 1$	3
$\widehat{\mathbf{m_{15}}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(1)p4(1 < Z_E^1, Z_E^2 >)p3(1 < Z_E^0, Z_E^2 >)$ $\lvert Z_E^2\rvert = 1\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{m_{16}}}$	$p6(1 < Z_E^1 > 1 < Z_E^2 >)p7(1 < Z_E^0 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^0, Z_E^1 >)$ $\lvert Z_E^0\rvert = 1\lvert Z_E^1\rvert = 1\lvert Z_E^2\rvert = 1$	6
$\widehat{\mathbf{m_{17}}}$	$p6(1 < Z_E^0 >)p7(1 < Z_E^0 >)p5(2)$ $\lvert Z_E^0\rvert = 3$	1

Table 13.2: Table of tangible symbolic states of the multicomputer PLC SWN model (without bus colours).

Vanishing Markings: 22 symbolic, 123 ordinary		
	Symbolic Marking	$\lvert\widehat{\mathbf{mv_i}}\rvert$
$\widehat{\mathbf{mv_1}}$	$p2(1 < Z_E^1 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(2)p1(1 < Z_E^0 >)$ $\lvert Z_E^0\rvert = 2 \lvert Z_E^1\rvert = 1$	3
$\widehat{\mathbf{mv_2}}$	$p7(1 < Z_E^0 > 1 < Z_E^1 > 1 < Z_E^2 >)p5(2)p3(1 < Z_E^0, Z_E^2 >)p1(1 < Z_E^1 > 1 < Z_E^2 >)$ $\lvert Z_E^1\rvert = 1 \lvert Z_E^0\rvert = 1 \lvert Z_E^2\rvert = 1$	6
$\widehat{\mathbf{mv_3}}$	$p2(1 < Z_E^2 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(1)p4(1 < Z_E^0, Z_E^2 >)p1(1 < Z_E^1 >)$ $\lvert Z_E^1\rvert = 1 \lvert Z_E^2\rvert = 1 \lvert Z_E^0\rvert = 1$	6
$\widehat{\mathbf{mv_4}}$	$p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^1, Z_E^0 >)p3(1 < Z_E^0, Z_E^2 >)p1(1 < Z_E^2 >)$ $\lvert Z_E^2\rvert = 1 \lvert Z_E^0\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_5}}$	$p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^2, Z_E^0 >)p3(1 < Z_E^0, Z_E^2 >)p1(1 < Z_E^1 >)$ $\lvert Z_E^2\rvert = 1 \lvert Z_E^1\rvert = 1 \lvert Z_E^0\rvert = 1$	6
$\widehat{\mathbf{mv_6}}$	$p2(1 < Z_E^2 >)p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^1, Z_E^0 >)p1(1 < Z_E^0 >)$ $\lvert Z_E^0\rvert = 1 \lvert Z_E^1\rvert = 1 \lvert Z_E^2\rvert = 1$	6
$\widehat{\mathbf{mv_7}}$	$p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^2, Z_E^0 >)p3(1 < Z_E^1, Z_E^2 >)p1(1 < Z_E^0 >)$ $\lvert Z_E^0\rvert = 1 \lvert Z_E^2\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_8}}$	$p2(1 < Z_E^2 >)p6(1 < Z_E^1 >)p7(1 < Z_E^0 > 1 < Z_E^1 > 1 < Z_E^2 >)p5(2)p1(1 < Z_E^0 >)$ $\lvert Z_E^0\rvert = 1 \lvert Z_E^1\rvert = 1 \lvert Z_E^2\rvert = 1$	6
$\widehat{\mathbf{mv_9}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^0 > 1 < Z_E^1 > 1 < Z_E^2 >)p5(2)p3(1 < Z_E^0, Z_E^2 >)p1(1 < Z_E^1 >)$ $\lvert Z_E^2\rvert = 1 \lvert Z_E^0\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_{10}}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^0 > 1 < Z_E^1 > 1 < Z_E^2 >)p5(2)p3(1 < Z_E^0, Z_E^1 >)p1(1 < Z_E^1 >)$ $\lvert Z_E^2\rvert = 1 \lvert Z_E^0\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_{11}}}$	$p2(1 < Z_E^2 >)p7(1 < Z_E^1 >)p4(1 < Z_E^0, Z_E^2 > 1 < Z_E^1, Z_E^0 >)$ $\lvert Z_E^0\rvert = 1 \lvert Z_E^2\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_{12}}}$	$p2(1 < Z_E^2 >)p7(1 < Z_E^2 >)p4(1 < Z_E^0, Z_E^1 > 1 < Z_E^1, Z_E^0 >)$ $\lvert Z_E^1\rvert = 1 \lvert Z_E^0\rvert = 1 \lvert Z_E^2\rvert = 1$	3
$\widehat{\mathbf{mv_{13}}}$	$p2(1 < Z_E^2 >)p6(1 < Z_E^1 >)p7(1 < Z_E^0 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^0, Z_E^1 >)$ $\lvert Z_E^0\rvert = 1 \lvert Z_E^2\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_{14}}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(1)p4(1 < Z_E^0, Z_E^2 >)p3(1 < Z_E^1, Z_E^0 >)$ $\lvert Z_E^0\rvert = 1 \lvert Z_E^2\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_{15}}}$	$p2(1 < Z_E^2 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(1)p4(1 < Z_E^1, Z_E^2 >)p3(1 < Z_E^0, Z_E^2 >)$ $\lvert Z_E^1\rvert = 1 \lvert Z_E^2\rvert = 1 \lvert Z_E^0\rvert = 1$	6
$\widehat{\mathbf{mv_{16}}}$	$p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^2, Z_E^0 >)p3(1 < Z_E^0, Z_E^2 > 1 < Z_E^1, Z_E^0 >)$ $\lvert Z_E^2\rvert = 1 \lvert Z_E^0\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_{17}}}$	$p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^1, Z_E^0 >)p3(1 < Z_E^0, Z_E^2 > 1 < Z_E^2, Z_E^0 >)$ $\lvert Z_E^2\rvert = 1 \lvert Z_E^1\rvert = 1 \lvert Z_E^0\rvert = 1$	6
$\widehat{\mathbf{mv_{18}}}$	$p2(1 < Z_E^2 >)p6(1 < Z_E^1 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(1)p4(1 < Z_E^0, Z_E^2 >)$ $\lvert Z_E^1\rvert = 1 \lvert Z_E^2\rvert = 1 \lvert Z_E^0\rvert = 1$	6
$\widehat{\mathbf{mv_{19}}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^1, Z_E^0 >)p3(1 < Z_E^0, Z_E^2 >)$ $\lvert Z_E^2\rvert = 1 \lvert Z_E^0\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_{20}}}$	$p6(1 < Z_E^2 >)p7(1 < Z_E^1 > 1 < Z_E^2 >)p5(1)p4(1 < Z_E^1, Z_E^0 >)p3(1 < Z_E^0, Z_E^1 >)$ $\lvert Z_E^2\rvert = 1 \lvert Z_E^0\rvert = 1 \lvert Z_E^1\rvert = 1$	6
$\widehat{\mathbf{mv_{21}}}$	$p2(1 < Z_E^1 >)p6(1 < Z_E^0 >)p7(1 < Z_E^0 > 1 < Z_E^1 >)p5(2)$ $\lvert Z_E^1\rvert = 1 \lvert Z_E^0\rvert = 2$	3
$\widehat{\mathbf{mv_{22}}}$	$p6(1 < Z_E^1 > 1 < Z_E^2 >)p7(1 < Z_E^0 > 1 < Z_E^1 > 1 < Z_E^2 >)p5(2)p3(1 < Z_E^0, Z_E^2 >)$ $\lvert Z_E^1\rvert = 1 \lvert Z_E^0\rvert = 1 \lvert Z_E^2\rvert = 1$	6

Table 13.3: Table of vanishing symbolic states of the multicomputer PLC SWN model (without bus colours).

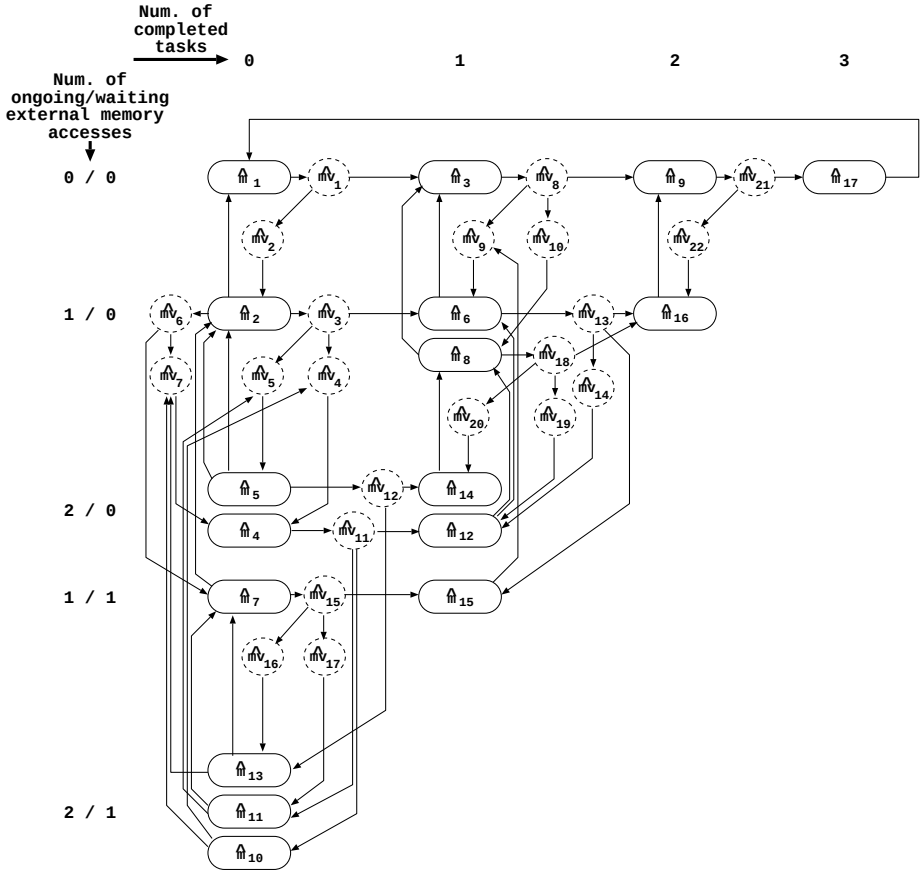


Figure 13.5: Symbolic Reachability Graph of the multicomputer PLC SWN model (without bus colour).

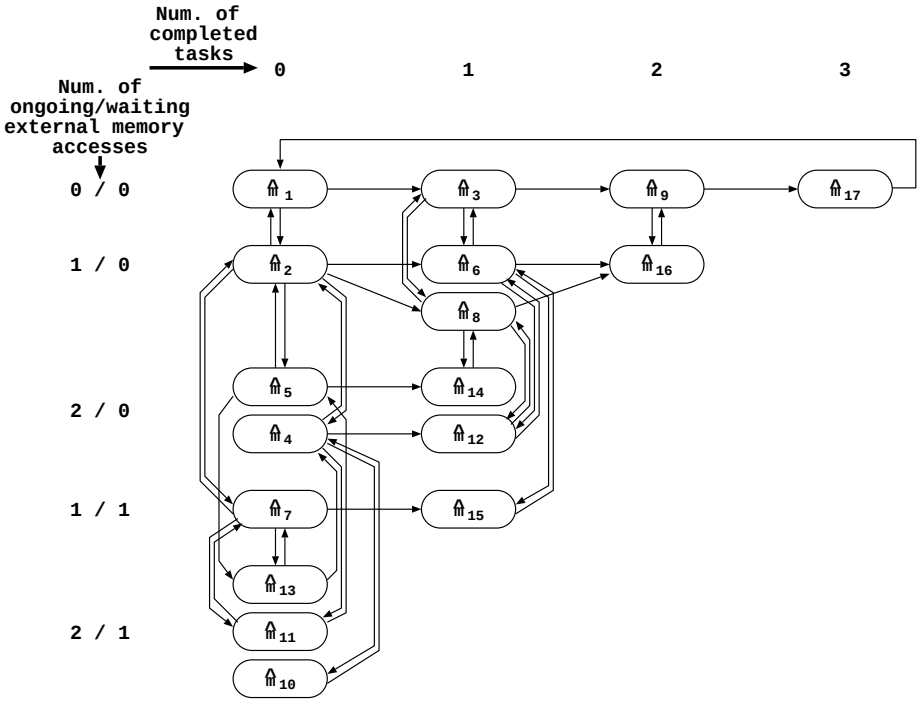


Figure 13.6: Tangible Symbolic Reachability Graph of the multicomputer PLC SWN model (without bus colour).

State Transition	Transition firing sequence and corresponding rate
$\widehat{\mathbf{m_1}} \rightarrow \widehat{\mathbf{m_2}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_2, \langle Z_E^1, Z_E^0 \rangle \rangle$, $\langle t_3, \langle Z_E^1, Z_E^0 \rangle \rangle$ $3\alpha_1 \ \frac{2\beta_2}{\beta_5+2\beta_2} \ \frac{\beta_3}{\beta_3}$
$\widehat{\mathbf{m_1}} \rightarrow \widehat{\mathbf{m_3}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_5, \langle Z_E^1 \rangle \rangle$ $3\alpha_1 \frac{\beta_5}{\beta_5+2\beta_2}$
$\widehat{\mathbf{m_2}} \rightarrow \widehat{\mathbf{m_1}}$	$\langle t_4, \langle Z_E^1, Z_E^0 \rangle \rangle$ α_4
$\widehat{\mathbf{m_2}} \rightarrow \widehat{\mathbf{m_4}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_2, \langle Z_E^2, Z_E^1 \rangle \rangle$, $\langle t_3, \langle Z_E^2, Z_E^0 \rangle \rangle$ $\alpha_1 \frac{\beta_2}{\beta_5+2\beta_2} \ \frac{\beta_3}{\beta_3}$
	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_2, \langle Z_E^2, Z_E^1 \rangle \rangle$, $\langle t_3, \langle Z_E^2, Z_E^1 \rangle \rangle$ $\alpha_1 \frac{\beta_2}{\beta_5+2\beta_2} \ \frac{\beta_3}{\beta_3}$
$\widehat{\mathbf{m_2}} \rightarrow \widehat{\mathbf{m_5}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_2, \langle Z_E^2, Z_E^0 \rangle \rangle$, $\langle t_3, \langle Z_E^2, Z_E^0 \rangle \rangle$ $\alpha_1 \frac{\beta_2}{\beta_5+2\beta_2} \ \frac{\beta_3}{\beta_3}$
$\widehat{\mathbf{m_2}} \rightarrow \widehat{\mathbf{m_6}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_5, \langle Z_E^2 \rangle \rangle$ $\alpha_1 \frac{\beta_5}{\beta_5+2\beta_2}$
$\widehat{\mathbf{m_2}} \rightarrow \widehat{\mathbf{m_7}}$	$\langle t_1, \langle Z_E^2 \rangle \rangle$, $\langle t_2, \langle Z_E^2, Z_E^0 \rangle \rangle$ $\alpha_1 \frac{\beta_2}{\beta_5+2\beta_2}$
$\widehat{\mathbf{m_3}} \rightarrow \widehat{\mathbf{m_6}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_2, \langle Z_E^2, Z_E^1 \rangle \rangle$, $\langle t_3, \langle Z_E^2, Z_E^0 \rangle \rangle$ $\alpha_1 \frac{\beta_2}{\beta_5+2\beta_2} \ \frac{\beta_3}{\beta_3}$
$\widehat{\mathbf{m_3}} \rightarrow \widehat{\mathbf{m_8}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_2, \langle Z_E^2, Z_E^0 \rangle \rangle$, $\langle t_3, \langle Z_E^1, Z_E^0 \rangle \rangle$ $\alpha_1 \frac{\beta_2}{\beta_5+2\beta_2} \ \frac{\beta_3}{\beta_3}$
$\widehat{\mathbf{m_3}} \rightarrow \widehat{\mathbf{m_9}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_5, \langle Z_E^2 \rangle \rangle$ $\alpha_1 \frac{\beta_5}{\beta_5+2\beta_2}$
$\widehat{\mathbf{m_7}} \rightarrow \widehat{\mathbf{m_2}}$	$\langle t_4, \langle Z_E^2, Z_E^0 \rangle \rangle$, $\langle t_3, \langle Z_E^2, Z_E^0 \rangle \rangle$ $\alpha_4 \frac{\beta_3}{\beta_3}$
$\widehat{\mathbf{m_7}} \rightarrow \widehat{\mathbf{m_{15}}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_5, \langle Z_E^2 \rangle \rangle$ $\alpha_1 \frac{\beta_5}{\beta_5+2\beta_2}$
$\widehat{\mathbf{m_7}} \rightarrow \widehat{\mathbf{m_{11}}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_2, \langle Z_E^2, Z_E^0 \rangle \rangle$, $\langle t_3, \langle Z_E^2, Z_E^0 \rangle \rangle$ $\alpha_1 \ \frac{\beta_2}{\beta_5+2\beta_2} \ \frac{\beta_3}{\beta_3}$
$\widehat{\mathbf{m_7}} \rightarrow \widehat{\mathbf{m_{13}}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_2, \langle Z_E^2, Z_E^1 \rangle \rangle$, $\langle t_3, \langle Z_E^2, Z_E^0 \rangle \rangle$ $\alpha_1 \ \frac{\beta_2}{\beta_5+2\beta_2} \ \frac{\beta_3}{\beta_3}$
$\widehat{\mathbf{m_9}} \rightarrow \widehat{\mathbf{m_{16}}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_2, \langle Z_E^1, Z_E^0 \rangle \rangle$, $\langle t_3, \langle Z_E^2, Z_E^0 \rangle \rangle$ $\alpha_1 \ \frac{2\beta_2}{\beta_5+2\beta_2} \ \frac{\beta_3}{\beta_3}$
$\widehat{\mathbf{m_9}} \rightarrow \widehat{\mathbf{m_{17}}}$	$\langle t_1, \langle Z_E^0 \rangle \rangle$, $\langle t_5, \langle Z_E^1 \rangle \rangle$ $\alpha_1 \frac{\beta_5}{\beta_5+2\beta_2}$
$\widehat{\mathbf{m_{17}}} \rightarrow \widehat{\mathbf{m_1}}$	$\langle t_6, \cdot \rangle$ α_6

Table 13.4: Portion of the Tangible SRG for the multicomputer PLC model.

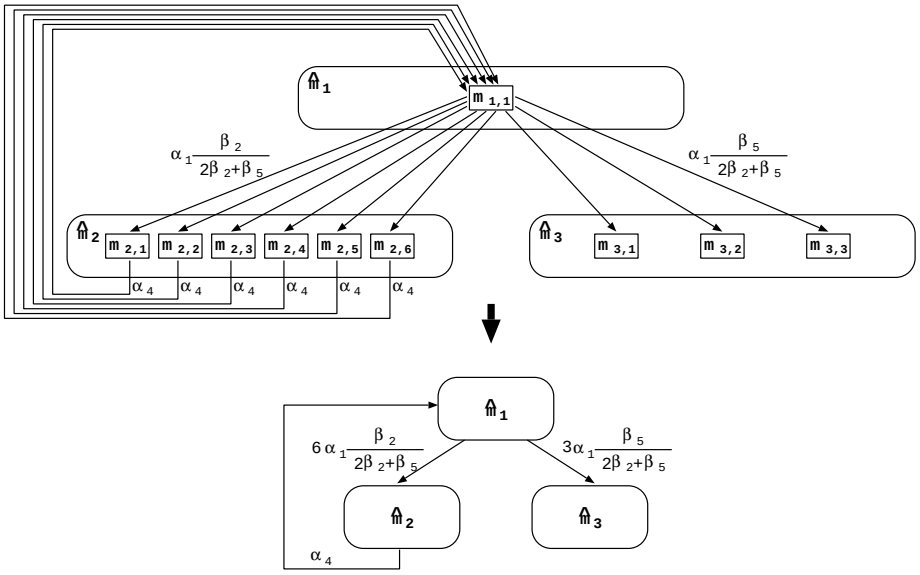


Figure 13.7: Example of lumping of states in the CTMC of the multicomputer PLC model according to the aggregation induced by the SRG.

Similarly, the weight of the symbolic firing of immediate transition t_2 following the t_1 firing considered above is obtained as product of the multiplicity 2 of the symbolic firing and weight β_2 associated with any instance of t_2; the transition probability is then obtained by normalization with respect to all immediate transition instances enabled in the same vanishing marking ($\widehat{\mathbf{mv_1}}$). Finally, after firing t_2 only one instance of t_3 is enabled, whose firing leads to $\widehat{\mathbf{m_2}}$, hence its firing probability is set to 1. By multiplying the rate/probability of the symbolic transition firing sequence just described one gets as expected:

$$3\alpha_1 \frac{2\beta_2}{2\beta_2+\beta_5} = 6\alpha_1 \frac{\beta_2}{2\beta_2+\beta_5}$$

The rate from $\widehat{\mathbf{m_2}}$ back to $\widehat{\mathbf{m_1}}$ is instead α_4, i.e. the rate of the single arc from any marking $\mathbf{m_{2,j}}$ to $\mathbf{m_{1,1}}$. Indeed, the symbolic firing of t_4 from symbolic marking $\widehat{\mathbf{m_2}}$ has cardinality 1, moreover

$$\widehat{\mathbf{w}}[t_4](\lambda, \mu, \widehat{\mathbf{m_1}}) = \mathbf{w}[t_4](\langle e_i, e_j \rangle) = \alpha_4, \forall e_i, e_j \in E$$

By solving the CTMC that can be derived from the SRG as explained above, it is possible to get the steady state probability of the aggregate states. Let us discuss the definition of the model performance indices, and their computation starting from the symbolic markings steady state probability distribution.

In Chapter 9 it has been shown how performance indices can be characterized through reward functions defined on the state space, and computed using

the steady state probability distribution of the markings of a GSPN. The same applies to SWNs, however since the states of the CTMC in this case correspond to the symbolic markings, it is necessary to make a distinction between reward functions that can be defined at the level of the symbolic markings, and reward functions that instead refer to ordinary markings. The performance indices corresponding to the former type of reward functions can be computed in a direct way from the solution of the lumped CTMC. The performance indices defined by means of the latter type of reward function can also be computed from the symbolic markings probability distribution, but requires an additional elaboration, based on the knowledge of the number of ordinary markings represented by each symbolic marking.

Examples of reward functions that can be applied to symbolic markings, are functions referring to the static partition of a marking. For example, assume that in the random polling system model we had the two basic classes Q and R partitioned into two static subclasses each, i.e., $Q = Q_1 \cup Q_2$ and $R = R_1 \cup R_2$, used to distinguish between queues with high and low arrival rates and between fast and slow servers. Under this hypothesis one could be willing to measure the average number of fast servers working for clients queued up in a queue of the second subclass; the reward function in this case should be defined as:

$$r(\mathbf{m}) = n \text{ iff } \tilde{\mathbf{m}}[p_s](\langle Q_2, R_2 \rangle) = n.$$

Since the static partition of every ordinary marking belonging to a given symbolic marking is the same, and it is equal to the static partition of the symbolic marking itself, it is possible to redefine the above reward function substituting $\mathbf{m}$ with $\widehat{\mathbf{m}}$, so that it is defined directly on the SRS.

Similarly, if we want to compute the number of services completed in the time unit by fast servers, the corresponding reward function should select those symbolic markings $\widehat{\mathbf{m}}$ in which one (or more) symbolic instances of T_s are enabled such that $Z_R^{\lambda(2)} \subseteq R_2$, and for these markings define

$$r(\widehat{\mathbf{m}}) = \sum_{\lambda,\mu : \widehat{\mathbf{m}} \xrightarrow{\langle T_s, \lambda, \mu \rangle} \wedge Z_R^{\lambda(2)} \subseteq R_2} |\widehat{\mathbf{m}} \xrightarrow{\langle T_s, \lambda, \mu \rangle}| \widehat{\mathbf{w}}[T_s](\lambda, \mu).$$

If we wish to compute the number of services completed in the time unit by a specific server $r_i \in R_k$, it is possible to just divide by $|R_k|$ the reward function for the computation of the number of services completed in the time unit by any server in class R_k; indeed since all servers in this class behave homogeneously, they must all have the same throughput.

As an example of reward function that requires to explicitly consider the ordinary markings belonging to a given symbolic marking, let us consider the computation of the probability of a specific server $r_i \in R_k$ being in the "working" state (represented by place p_s), then we need to select from the set of symbolic markings the subset having at least one server of subclass R_k in place p_s, then evaluating for each such symbolic marking $\widehat{\mathbf{m}_\mathbf{j}}$ the number l_j of ordinary markings included in it such that r_1 is in p_s. Hence, due to the equiprobability

of ordinary markings belonging to the same symbolic marking, the definition of the reward function would be:

$$r(\widehat{\mathbf{m_j}}) = \frac{l_j}{|\widehat{\mathbf{m_j}}|}.$$

Some remarks about ergodicity of SWNs As already noted, the steady-state solution of a CTMC can be computed only if it is ergodic. Ergodicity is ensured if the reachability graph associated with a CTMC contains one maximal strongly-connected component. As stated in Chapter 7, strong connectedness of the SRG of a SWN is a necessary, but not sufficient condition for the strong connectedness of the ordinary RG. Therefore the SRG generated by a SWN is hereafter assumed to be strongly connected.

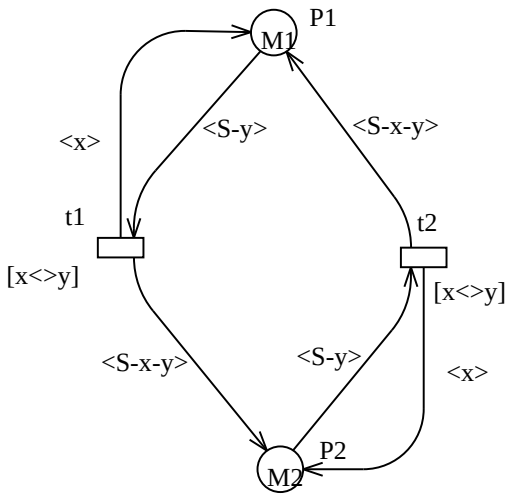
In Chapter 7 a *sufficient condition* for the RG ergodicity has been defined: informally, and considering only non ordered classes, it states that for each class it must exist a symbolic marking where there is a single dynamic subclass for each static subclass (i.e. all the objects of a static subclass are grouped).

Let us consider and briefly discuss some cases which may occur, concerning the fulfillment of the sufficient condition and the RG ergodicity. They represent all the situations we dealt with in our experience. In particular, an example is presented and discussed showing that computing the steady-state solution of the CTMC obtained from the SRG makes sense even if the corresponding RG is not strongly connected.

Example 13.2 (sufficient condition satisfied).
Let us consider the model of the PLC system (Fig. 13.1), whose SRG (without bus colour) is described in Fig. 13.5 and in Tabs. 13.2 and 13.3. We can observe that in the symbolic initial marking $\widehat{\mathbf{m_1}}$ there is only one dynamic subclass ($< Z_E^0 >$) associated with the unique colour class E (there is a trivial partition of E into static subclasses).

It is worth noting that in general the sufficient condition (s.c.) is trivially satisfied if the symbolic initial marking $\widehat{\mathbf{m_1}}$ is symmetric (i.e. it corresponds to only one ordinary marking $\mathbf{m_1}$). Hence in the following examples we restrict our attention to SWN models with asymmetric initial markings. Notice that in this case the RG is the one generated from the set of ordinary markings contained in the equivalence class $\widehat{\mathbf{m_1}}$.

Example 13.3 (s.c. not satisfied and RG strongly connected).
Let us consider the simple SWN in Fig. 13.8, where we suppose that $cd(p1) = cd(p2) = C$, $|C| \geq 3$. The corresponding SRG, whose size is independent of $|C|$, is described in Tabs. 13.5 and 13.6 ($\widehat{\mathbf{m_1}}$ is the symbolic initial marking). The sufficient condition for ergodicity is neither verified in $\widehat{\mathbf{m_1}}$ nor in $\widehat{\mathbf{m_2}}$. However, it may be easily checked that the ordinary RG, composed by six ordinary markings (in the case $C = \{a, b, c\}$), and described in Tabs. 13.7 and 13.8, is strongly connected. The initial markings contained in $\widehat{\mathbf{m_1}}$ are in fact reachable from each other (each of them is a home state of the RG).



$$M1 = \langle Z_C^0 \rangle, M2 = \langle Z_C^1 \rangle, |Z_C^0| = |C| - 1, |Z_C^1| = 1$$

Figure 13.8: Ergodic SWN, for which the sufficient condition does not hold.

Obviously, the modeller should demonstrate that the RG is strongly connected without generating the RG itself. Intuitively, demonstrating the RG strong connectedness (in case the ergodicity sufficient condition does not hold) consists of showing that the color elements may "move" from one dynamic subclass to another, by a finite number of symbolic firings. For the considered example, it may be easily verified that this is true for all elements of colour class C in both $\widehat{\mathbf{m_1}}$ and $\widehat{\mathbf{m_2}}$: in fact they can change dynamic subclass by firing a sequence of t_1 and t_2 symbolic instances. Translating the above intuitive form into conditions on the SRG structure, or better, on the SWN structure, which ensure (if satisfied) the RG ergodicity, is an important question which must be addressed. Notice that the original s.c. for ergodicity may be seen as a particular case where the above intuitive condition holds.

Example 13.4 (s.c. not satisfied and RG formed by disjoint strongly connected components).
Let us consider again the SWN model of the PLC system (Fig. 13.1). We here introduce some slight modifications/additions to the original model, which are illustrated in Fig. 13.9. With respect to the original model, we now assume that one task, identified by the initial marking of the (new) place p_8, plays the role of

Symbolic Reachability Set		
	Symbolic Marking	$\lvert\widehat{\mathbf{m_i}}\rvert$
$\widehat{\mathbf{m_1}}$	$p1(1 < Z_C^0 >)p2(1 < Z_C^1 >)$ $\lvert Z_C^0\rvert = \lvert C\rvert - 1 \lvert Z_C^1\rvert = 1$	$\lvert C\rvert$
$\widehat{\mathbf{m_2}}$	$p1(1 < Z_C^0 >)p2(1 < Z_C^1 >)$ $\lvert Z_C^0\rvert = 1 \lvert Z_C^1\rvert = \lvert C\rvert - 1$	$\lvert C\rvert$

Table 13.5: Symbolic markings of the SWN in Fig. 13.8

State Transition	symbolic firing $(\langle t, \langle x, y\rangle\rangle)$
$\widehat{\mathbf{m_1}} \rightarrow \widehat{\mathbf{m_2}}$	$\langle t_1, \langle Z_C^0, Z_C^1\rangle\rangle$
$\widehat{\mathbf{m_2}} \rightarrow \widehat{\mathbf{m_1}}$	$\langle t_2, \langle Z_C^1, Z_C^0\rangle\rangle$

Table 13.6: Symbolic state transitions of the SWN in Fig. 13.8

Ordinary Reachability Set		
	Ordinary Marking	Corresp. SM
$\mathbf{m_1}$	$p1(1\langle a\rangle 1\langle b\rangle)\, p2(1\langle c\rangle)$	$\widehat{\mathbf{m_1}}$
$\mathbf{m_2}$	$p1(\langle b\rangle)\, p2(1\langle a\rangle 1\langle c\rangle)$	$\widehat{\mathbf{m_2}}$
$\mathbf{m_3}$	$p1(1\langle a\rangle 1\langle c\rangle)\, p2(1\langle b\rangle)$	$\widehat{\mathbf{m_1}}$
$\mathbf{m_4}$	$p1(1\langle a\rangle)\, p2(1\langle b\rangle 1\langle c\rangle)$	$\widehat{\mathbf{m_2}}$
$\mathbf{m_5}$	$p1(1\langle c\rangle 1\langle b\rangle)\, p2(1\langle a\rangle)$	$\widehat{\mathbf{m_1}}$
$\mathbf{m_6}$	$p1(1\langle c\rangle)\, p2(1\langle a\rangle 1\langle b\rangle)$	$\widehat{\mathbf{m_2}}$

Table 13.7: Ordinary markings of the SWN in Fig. 13.8 ($C = \{a, b, c\}$)

State Transition	ordinary firing $(\langle t, \langle x, y\rangle\rangle)$
$\mathbf{m_1} \rightarrow \mathbf{m_2}$	$\langle t_1, \langle a, c\rangle\rangle$
$\mathbf{m_1} \rightarrow \mathbf{m_4}$	$\langle t_1, \langle b, c\rangle\rangle$
$\mathbf{m_2} \rightarrow \mathbf{m_1}$	$\langle t_2, \langle a, b\rangle\rangle$
$\mathbf{m_2} \rightarrow \mathbf{m_5}$	$\langle t_2, \langle c, b\rangle\rangle$
$\mathbf{m_3} \rightarrow \mathbf{m_4}$	$\langle t_1, \langle c, b\rangle\rangle$
$\mathbf{m_3} \rightarrow \mathbf{m_6}$	$\langle t_1, \langle a, b\rangle\rangle$
$\mathbf{m_4} \rightarrow \mathbf{m_1}$	$\langle t_2, \langle b, a\rangle\rangle$
$\mathbf{m_4} \rightarrow \mathbf{m_3}$	$\langle t_2, \langle c, a\rangle\rangle$
$\mathbf{m_5} \rightarrow \mathbf{m_6}$	$\langle t_1, \langle b, a\rangle\rangle$
$\mathbf{m_5} \rightarrow \mathbf{m_2}$	$\langle t_1, \langle c, a\rangle\rangle$
$\mathbf{m_6} \rightarrow \mathbf{m_5}$	$\langle t_2, \langle b, c\rangle\rangle$
$\mathbf{m_6} \rightarrow \mathbf{m_3}$	$\langle t_2, \langle a, c\rangle\rangle$

Table 13.8: Ordinary state transitions of the SWN in Fig. 13.8 ($C = \{a, b, c\}$)

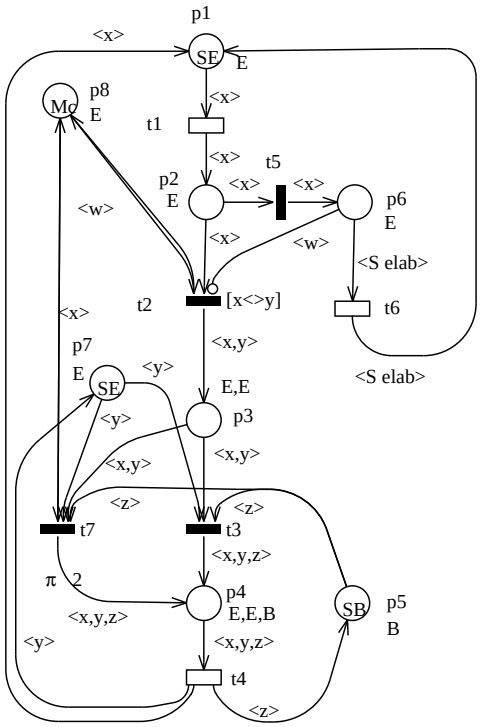


Figure 13.9: Not ergodic version of the PLC model, whose RG is formed by disjoint strongly connected components.

coordinator of the task pool. The behaviour of the coordinator task differs from that of the remaining ones for two aspects. Firstly, its external memory requests are scheduled with priority with respect to the other tasks: this is achieved by introducing the (new) immediate transition t_7, which inherits the input/output connections from t_3, and which in addition is connected by a test arc to the place p_8. To solve the (structural) conflict between t_7 and t_3 in favour of t_7, an higher priority is assigned to the transition t_7. Secondly, when the coordinator task becomes ready for a synchronization, no other task can issue a new memory request. This is realized by adding a test connection between the place p_8 and the transition t_2, and an inhibition connection between the place p_6 and the transition t_2.

We also assume that the role of coordinator is maintained by a task during the system evolution, and that this role is arbitrarily assigned to any task of the pool at starting time. This is obtained by a dynamic partition of the task

pool. For this purpose, the symbolic initial marking is defined as follows:

$$\widehat{\mathbf{m_1}} = p_1(1 < Z_E^0 > +1 < Z_E^1 >) + p_8(1 < Z_E^1 >) + p_7(1 < Z_E^0 > +1 < Z_E^1 >)$$

with $|Z_E^0| = |E|-1$, $|Z_E^1| = 1$. The dynamic subclass Z_E^1 represents the (generic) task chosen as coordinator.

The SRG generated by the modified PLC model (not depicted due to its dimensions) has a structure similar to that obtained from the original model (shown in Fig. 13.5). Instead, the ordinary RG is not strongly connected, being formed by a number ($|E|$) of disjoint strongly connected components, isomorphic with each other (each ordinary initial marking contained in $\widehat{\mathbf{m_1}}$ is a home-state of the single RG component to which it belongs, but it is not a home-state if we consider the whole RG). It may be checked that the intuitive ergodicity condition stated in the previous example does not hold, simply observing that the marking of the place p_8 (denoting the coordinator class) never changes (there are only test or inhibition connections from/to this place).

It is worth noting that the SRG of the above model is isomorphic to that obtained using a static partition of the colour class E to identify the coordinator task, i.e., by partitioning E into two static subclasses E_1 and E_2, with $|E_1| = 1$, $|E_2| = |E| - 1$, and defining a *symmetric* initial marking:

$$\widehat{\mathbf{m_1}} = p_1(1 < S_{E_1} > +1 < S_{E_2} >) + p_8(1 < S_{E_1} >) + p_7(1 < S_{E_1} > +1 < S_{E_2} >)$$

Moreover the transition probabilities between pairs of corresponding symbolic markings of the SRG_1 obtained from the first model (where a *dynamic* partition is used to identify the coordinator task) and the SRG_2 obtained from the second model (where a *static* partition is used to identify the coordinator task) are identical. Consequently, also the corresponding CTMCs are identical. Moreover, the single strongly connected components of the RG associated with SRG_1 are isomorphic to the RG associated with SRG_2. Hence, we may interpret the SRG_1 as the lumped reachability graph of *any* of its strongly connected RGs.

What is interesting is that, on the light of the previous considerations, performance indices computed by solving at steady-state the CTMC associated with a SRG whose associated RG is formed by disjoint strongly connected components are still meaningful, even if they require a different interpretation with respect to the case of an ergodic RG. The "high level" indices directly derivable from the symbolic marking probability vector (e.g., in our example, $Pr\{\#p_6 = 1\}$) may be computed in the usual way. If instead we have to consider indices defined as functions of ordinary marking probabilities, e.g. $Pr\{\#(p6(\langle e_3 \rangle)) > 0 \text{ AND } \#(p8(\langle e_1 \rangle)) = 1\}$ (where the marking expression corresponds to a set of ordinary markings), we have to define a new relation between the states of the CTMC generated by the SRG, and subsets of ordinary markings (which is no more the same relation assigning ordinary markings to symbolic markings).

We know that in case of ergodic RG the steady-state probability of an ordinary marking $\mathbf{m_i}$ can be derived from the probability of the corresponding symbolic marking, due to the equiprobability property: if $\mathbf{m_i} \in \widehat{\mathbf{m_i}}$, it holds

$Pr\{\mathbf{m_i}\} = \frac{Pr\{\widehat{\mathbf{m_i}}\}}{|\widehat{\mathbf{m_i}}|}$. In case of non ergodic RG, this definition is not correct, since we have to consider the number k_i of ordinary markings of a single strongly connected component belonging to the same symbolic marking ($k_i < |\widehat{\mathbf{m_i}}|$). In fact we just observed that a single RG component is the unfolding of SRG_2, which is in turn isomorphic to SRG_1. Therefore, in case of non ergodic RG, it holds $Pr\{\mathbf{m_i}\} = \frac{Pr\{\widehat{\mathbf{m_i}}\}}{|k_i|}$. With simple arguments it can be shown that $k_i = \frac{|\widehat{\mathbf{m_i}}|}{n}$, where n is the number of disjoint components (in our example $n = |E|$).

Notice that performance indices on ordinary markings must be defined in a consistent way, i.e. the marking expression used in index definition must correspond to a set of markings all belonging to the same strongly connected component. Concerning the PLC model, the marking of place p_8 must be included in the marking expression (as in the above definition). An example of inconsistent definition is $Pr\{\#(p6(\langle e_3 \rangle) > 0\}$, which involves ordinary markings belonging to disjoint RG components.

As a final remark, it may be observed that if in the revisited PLC model the coordinator task were re-elected after any synchronization, we would come back to the situation described in the Example 13.3. This might be simply obtained by connecting the place p_8 to the transition t_6 of Fig. 13.9 with input and output arcs labeled by the colour functions $\langle x \rangle$ and $\langle y \rangle$, respectively. With this addition, several ($|E|$) conflicting instances of the timed transition t_6 would substitute the original transition t_6. In such a case, due to the adopted race firing policy, the firing rate of t_6 should be normalized by factor $|E|$, in order to obtain a correct stochastic behaviour.

At the moment, there are not precise definitions/conditions characterizing the situations of "apparent" non-ergodicity, described in the two previous examples. This is an important open question, since the modeller, if the ergodicity s.c. is not satisfied, should demonstrate to fall into either one of the previous cases (described by the Examples 13.3 and 13.4), in order to correctly interpret the performance results obtained by the SRG steady-state analysis.

13.3 Decolourization and lumpability

In this section we shall reconsider the idea of decolourization presented in Chapter 7 for WN models, and discuss the possibility of extending it to SWN models. Structurally redundant colour flows (see Chapter 7) of a WN can be reduced without changing the *qualitative* behaviour of the model. It is moreover possible to show that the aggregation induced by decolourization satisfies the lumpability condition for Markov chains, however it is not easy to exploit this property at a structural reduction level because in general it is difficult to derive the rates to be associated with the transitions in the decoloured model in such a way that the CTMC computed from the reduced model is equivalent to the CTMC obtained by applying lumping to the original CTMC: let us illustrate this problem on the simple example shown in Fig. 13.10. The problem is due to the fact that as colours are reduced, the number of concurrently enabled colour instances of a

transition may decrease. Let us consider the decolourization of the structurally redundant colour flow corresponding to class C_2, assuming for simplicity that all instances of transition t_1 have the same associated rate α, all instances of transition t_2 have the same associated rate β and both transitions have infinite server semantics. In the CTMC corresponding to the RG of the original model, there is a rate of 4 α out of marking $\mathbf{m_0}$ while in the CTMC corresponding to the reduced RG obtained after the decolourization of C_2, there is a rate of 2 α out of $\mathbf{m_0}'$, unless a marking dependent rate is defined for the decolourized transition t_1. The situation is different for the state transitions corresponding to the firing of transition t_2, due to the fact that the instances of t_2 are not mutually conflicting and since the transition has an associated infinite server semantics.

From the above very simple example it is clear that in general it is difficult to define the correct rate functions for the decolourized model: all the experiences in this direction up to now have dealt with this problem on a case by case basis (see for example [1]). Hence the procedure for decolouring a SWN model consists of first checking the structural redundancy with respect to the qualitative behaviour, then analyzing each decoloured transition, observing what kind of transition aggregation is induced on the unfolded model by the decolourization operation, and possibly adjust the rates/weights of transitions of the decolourized model in order to maintain the same stochastic behaviour as in the original model.

Some *rules of thumb* can help in this delicate rates adjustment operation: timed transitions with single server semantics may reduce the degree of concurrency (among different instances), and in general it is not possible to define the correct rate for the decoloured transition except for some particular cases, for example if the transition has just one input place which is colour safe, and the function on the input arc is the identity, its decoloured counterpart can be simply assigned an infinite server semantics.

Timed transitions with infinite server semantics are easier to deal with, however, as we have shown on the example of Fig. 13.10, when there are several input places and there is a *combinatorial* number of different colour instances which are in conflict with each other in the original model the correct rate of the decolourized transition might require the definition of a complex marking dependent rate.

Concerning the partial decolourization operations defined in Chapter 7 it is possible to define the following rules to maintain the stochastic behaviour. In case of the *elimination of a colour not immediately used* we assume that the rate/weight of t_1 does not depend on the specific value assigned to variable x to be eliminated. By observing the effect of the reduction on the unfolding, one can realize that the instances of t_1 that are merged as a consequence of the decolourization, were in free choice conflict in the original net. If C_1 is the type of x, then the number of concurrently enabled instances of t_1 (in any marking) is reduced by a factor $|C_1|$ by the application of the rule. To compensate the loss of concurrency it is therefore necessary to multiply the weight/rate of (each instance of) t_1 by $|C_1|$. On the contrary, by merging the instances of p, we

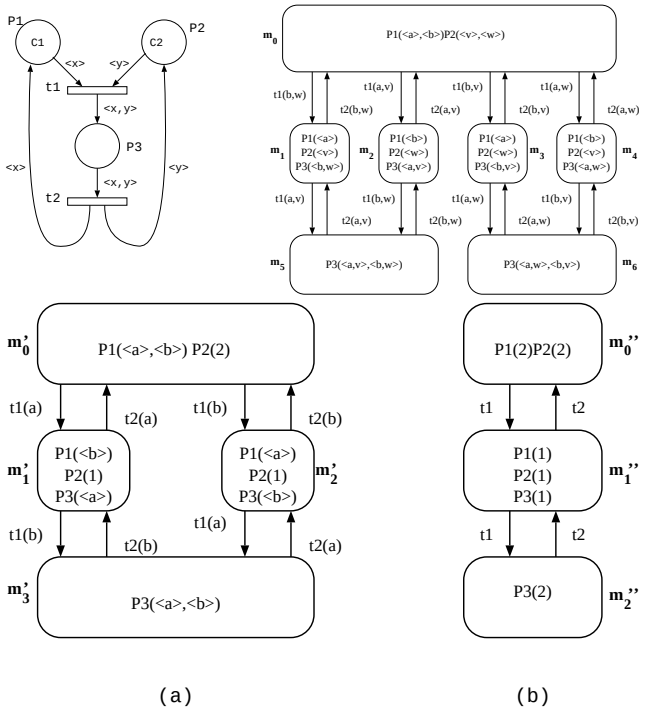


Figure 13.10: An example of decolourization that does not satisfy the strong lumpability condition for CTMC

transition	rate/weight	semantics/priority
Timed Transitions		
T_a	α	single-server per colour
T_s	β	single-server per colour
T_w	ω	single-server per colour
Immediate Transitions		
t_s	1	1
t_w	1	1
t_r	1	1

Table 13.9: Characteristics of the transitions in the first SWN model of a multiserver random polling system

may increase the number of (possibly) enabled instances of t_2 (x is no longer constrained by the marking of p). This is a problem only if t_2 is timed. In this case we require an infinite-server semantics for t_2, and divide its rate by the cardinality of the colour class C_1 of x. The reason for dividing the rate by $|C_1|$ is that, after the reduction, t_2 makes a (free) choice of a value for x among the objects in C_1 and we assume a uniform probability distribution over C_1.

In case of the *elimination of a colour "swallowed" by a transition* we assume that the rate/weight function associated with t_2 does not depend on the specific colour assigned to variable x; the effect of this reduction is to merge all the instances of p with equal first component, and all the instances of t_2 with the same value of x. If t_2 is timed with infinite-server semantics, then the timed behavior can be preserved by keeping the infinite-server semantics after the reduction. If t_2 is timed with single-server semantics, then the reduction is possible only if p is colour-safe for any colour in its colour domain (a place p is *colour safe* for colour q if there can be at most one token of colour q in p, in any reachable state): in this case the semantics of t_2 is changed to infinite-server in the reduced net. If t_2 is immediate, no problem arises, because the merged instances of t_2 are non-conflicting in the original net (conflicts exist neither among the different instances, nor with respect to other transitions). The transitions $t_{11}, \ldots, t_{1h}$ are not affected by the reduction (their colour domains and enabling conditions remain unchanged).

Example 13.5 In Example 7.1 we have shown how the WN of a random polling system can be simplified using the decolourization technique. In this case, the decolourization preserves also the stochastic behaviour of the corresponding SWN model, provided that the rates of the decoloured transitions are properly adjusted. The initial definition of the rates/weights associated with the transitions in the SWN model is given in Tab. 13.9. Let us recall that we are considering a system with capacity 1 queues.

We first consider the effect of the complete decolourization of the servers colour class. Transition T_w represents a server walk from queue x to queue y, its original timing definition is rate ω for all instances, and single-server per colour

semantics: observe that for the given initial marking it is irrelevant whether the semantics of this transition is single-server or infinite-server because place p_w is colour safe. However, after decolouring the servers component, we need to define it as infinite server, since there might be several servers walking between the same two queues identified by several tokens of identical colour in p_w. The effect of decolouring on transition transition T_s is similar, however, since we are assuming capacity 1 queues, place p_s remains colour safe after decolourization and T_s can maintain the single-server per colour semantics, as well as the same rate β. The effect of decolouring t_r with respect to the servers component is that of superposing $|R|$ subnets that were not connected before decolouring. Each subnet is a free choice over the set of queues. The decolourization does not change the conflicts to be resolved when transition t_r fires, so that its weight definition can be left unchanged. The decolourization of t_s causes several (equiprobable) conflicting instances (representing several servers trying to start a service at the same queue) to be merged. Let us consider a state in which servers r_1 and r_2 have reached queue q_1 (i.e., place p_p contains the two tokens $\langle r_1, q_1\rangle$ and $\langle r_2, q_1\rangle$) while a customer was queued in q_1 (i.e., p_q contains a token $\langle q_1\rangle$). From this state, with probability 1/2 a new state is reached in which r_1 is serving q_1 (token $\langle r_1, q_1\rangle$ in place p_s), and with probability 1/2 a new state is reached in which r_2 is serving q_1. After decolourization of the servers, these two states are merged into a single state, and so are the arcs: hence in the CTMC associated with the decoloured RG, the probability of the new "merged" arc should be the sum of the probability of the k originating arcs (in the example above $k = 2$); observe that if the originating arcs had all equal probability p, the new arc will have probability kp. Since in SWNs the probability of immediate transition firing is computed by normalization of the weights of the conflicting transitions, merging k conflicting transitions (with equal weight) automatically causes the probability of the decoloured transition replacing them to be multiplied by a factor k. Hence t_s can be decolourized without changes in the associated weight. Finally, the instances of t_w that are merged, are not in conflict in the original model, so that the decolourization does not affect their firing probability.

Let us now consider the partial decolourization steps that allow to simplify the colour flow of class Q. The first rule applied was a reduction of a colour component x swallowed by transition T_w. As already discussed earlier in this section, it is required that the rate of T_w instances does not depend on x, moreover, since several instances of T_w are merged as effect of the decolourization, the rate out of the markings in which T_w is enabled can be preserved if its semantics is infinite server before decolourization. Both conditions above are satisfied by transition T_w in the SWN model in which the servers component has been already decoloured.

The second reduction rule is again a reduction of colour component x swallowed by transition t_r: in this case, it is just required that the weight of the instances of t_r does not depend on x, requirement indeed satisfied by our model.

Finally the reduction of colour component y created by t_r but not immediately used, requires that the rate of t_r does not depend on y, which is true,

transition	rate/weight	semantics/priority
Timed Transitions		
T_a	α	single-server per colour
T_s	β	single-server per colour
T_w	$\omega/\|Q\|$	infinite-server
Immediate Transitions		
t_s	1	1
t_w	1	1

Table 13.10: Characteristics of the transitions in the decoloured SWN model of a multiserver random polling system

moreover the weight of each decoloured instance t_r must be multiplied by a factor $|Q|$ while the rate of the instances of T_w must be divided by a factor $|Q|$ to compensate the fact that y is no more constrained by the marking of place p_w. In the last reduction step t_r is recognized to be redundant and eliminated from the net: this reduction does not affect the stochastic behaviour of the model because t_r fires in 0 time and it always fires with probability 1 after it gets enabled (i.e., it doesn't model any probabilistic conflict resolution). Tab. 13.10 summarizes the new timing definition of the decoloured model of Fig. 7.27.

13.4 Concluding Remarks

In this chapter the SWN formalism has been defined as a timed extension of WNs; although the extension is quite similar to the one leading from PNs to GSPNs there are subtle issues in the assignment of rates and weights to the transitions of a SWN model that are completely new. Basically, the problem is due to the fact that sometimes it is difficult for the modeller to realize at a first glance which is the unfolding of a given high-level model. When assigning timed transition rates for example, it is important to take into account the fact that several instances of the same transition may be concurrently enabled: due to the race policy this may result in undesired increased activity rates. When assigning immediate transition weights, it might be difficult to precisely identify the ECS of the model without unfolding it: therefore it is important to support the modeller weight definition task providing a list of ECS in a symbolic form.

When the transition rates/weights are defined in such a way to ensure symmetric timed behaviour, it is possible to take advantage of the SRG technique developed for WN models to reduce the size of the CTMC associated with a SWN by applying lumpability; as a consequence performance indices of SWN models can be computed efficiently by automatically exploiting the system symmetries. The most remarkable aspect of this approach, if compared to other approaches applying lumpability, is that the lumped CTMC is obtained directly from the (SRG of the) SWN model without need to ever generate the complete

stochastic process.

Sometimes, it may happen that the SRG of a SWN model is ergodic (i.e., strongly connected) but the sufficient condition that ensures that the corresponding RG is ergodic is not satisfied. Since the steady state solution makes sense only if the underlying MC is ergodic, in principle when the sufficient condition does not hold we do not know whether we can obtain meaningful steady state performance measures or not. We have discussed some cases in which it is possible to apply steady state analysis despite the sufficient condition for RG ergodicity is not satisfied.

Finally it has been discussed the relation between the aggregation achievable through decolourization and lumpability: the application of the decolourization technique to SWNs is not automatizable in general; the problem in this case is the definition of the rates/weights of the decoloured transitions. We have shown a set of cases in which the decolourization rules defined for WN models can be extended to work on SWNs.

13.5 Bibliographical Remarks

The first works dealing with efficient performance analysis algorithms for stochastic coloured Petri nets are [17, 13, 2].

In [6] it has been shown that the aggregation induced by the SRG algorithm can be exploited for performance evaluation purposes: this paper was based on the formalism of Regular Nets, the "ancestor" of Well-formed Nets.

The algorithms for the computation of the SRG and the automatic derivation of the lumped CTMC of a *SWN* model have been first presented in [3, 4].

Another approach to the efficient performance analysis of HLPN has been defined for the Stochastic Activity Network (SAN) formalism in [16, 15]. This approach is very similar to the one described in this chapter, in fact it derives a lumped stochastic process, based on the symmetries that are automatically detected in the model (symmetries are present when the "replicate" operator is used in the composition of the complete net starting from simpler subnets).

The relation between aggregation in stochastic high-level Petri nets and in stochastic process algebras (see Chapter 16) has been investigated in [14, 11, 9, 10].

Bibliography

[1] M. Ajmone Marsan, S. Donatelli, G. Franceschinis, and F. Neri. Reductions in Generalized Stochastic Petri Nets and Stochastic Well-formed Nets: An Overview and an Example of Application. In *Network Performance Modeling and Simulation, J. Walrand, K. Bagchi and G. Zobrist (editors).* Gordon and Breach Publishers INC, 1997.

[2] G. Chiola, G. Bruno, and T. Demaria. Introducing a color formalism into generalized stochastic Petri nets. In *Proc.* 9^{th} *Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[3] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic Well-Formed coloured nets and multiprocessor modelling applications. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets. Theory and Application.* Springer Verlag, 1991.

[4] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11), November 1993.

[5] G. Chiola, G Franceschinis, R. Gaeta, and M. Ribaudo. Symmetry exploitation in stochastic well-formed colored nets. Technical Report DW2.T2.13.v1, EEC ESPRIT-BRA QMIPS (Quantitative Modeling In Parallel Systems) Project, 1994.

[6] C. Dutheillet and S. Haddad. Regular stochastic Petri nets. In *Proc. 10th Intern. Conf. Application and Theory of Petri Nets*, Bonn, Germany, June 1989.

[7] C. Dutheillet and S. Haddad. Structural analysis of coloured nets. Application to the detection of confusion. Technical report, IBP-Laboratoire MASI, Universite P. et M. Curie, IFCPAR Project 302-1, 1992.

[8] C. Dutheillet and S. Haddad. Conflict sets in colored Petri nets. In *Proc.* 5^{rd} *Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

[9] G. Franceschinis and M. Ribaudo. Efficient performance analysis techniques for stochastic well-formed nets and stochastic process algebras. To appear in LNCS.

[10] G. Franceschinis and M. Ribaudo. Symmetric and behavioural aggregation in a simple protocol example. In *Proc. of the* 6^{th} *International Workshop on Process Algebra and Performance Modelling, PAPM98*, Nice, France, September 1998.

[11] S. Gilmore, J. Hillston, and M. Ribaudo. An Efficient Algorithm for Aggregating PEPA Models. Technical report, University of Edinburgh, 1998.

[12] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Van Nostrand, Princeton, NJ, 1960.

[13] C. Lin and D.C. Marinescu. On stochastic high level Petri nets. In *Proc. Intern. Workshop on Petri Nets and Performance Models*, Madison, WI, USA, August 1987. IEEE-CS Press.

[14] M. Ribaudo. On the Aggregation Techniques in Stochastic Petri Nets and Stochastic Process Algebras. *The Computer Journal*, 38(6), 1995. Special Issue: Proc. of 3rd Process Algebra and Performance Modelling Workshop.

[15] W. H. Sanders and J. F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, January 1991. Special Issue on Computer-Aided Modeling, Analysis and Design of Communication Networks.

[16] W.H. Sanders and J.F. Meyer. Reduced base model construction methods for stochastic activity networks. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.

[17] A. Zenie. Colored stochastic Petri nets. In *Proc. Intern. Workshop on Timed Petri Nets*, pages 262–271, Torino, Italy, July 1985. IEEE-CS Press.

Chapter 14

Bounds for quasi-lumpable SWNs

The work presented in this chapter is related to the lumpability approach to state space reduction presented in Chapter 13, in fact it focuses on Markov chains that incorporate some structural regularity similar to that required by lumpability methods. With respect to the MC lumping based on the SRG definition, this method allows a further contraction of the state space by relaxing the lumpability conditions, and computes approximate performance indices on the (smaller) MC. More precisely the results obtained are upper and lower bounds on the performance indices of interest.

The chapter is organized as follows: in Section 14.1 the concept of a *quasi-lumpable* Markov chain is defined, in Section 14.2 we show that given a *quasi-lumpable* Markov chain, it can be altered in such a way that the new resulting MC is lumpable and is such that Courtois and Semal's steady state probability bounding methods [3, 2] can be applied to the new lumped MC to obtain bounds on the performance measures of the original model. In Section 14.3 the extent of applicability of the method is discussed. The most important issue addressed in this section is how to check if a given system has a quasi-lumpable underlying MC. We claim that it is not reasonable to check whether the quasi-lumpability property holds by inspecting the MC itself, but rather by exploiting some high level property of the abstract model from which the MC is derived, such as in the case of SWNs.

14.1 Quasi-Lumpable Markov Chains

Definition 14.1 *A CTMC is said to be* ϵ-quasi-lumpable *with respect to a given state space partition A if its infinitesimal generator* $\mathbf{Q}$ *can be rewritten as* $\mathbf{Q} = \mathbf{Q}^{-} + \mathbf{Q}^{\epsilon}$*, where* $\mathbf{Q}^{-}$ *is the largest lower bound (componentwise) for* $\mathbf{Q}$ *that satisfies the lumpability conditions with respect to partition A.*

The intention is that $\mathbf{Q}^\epsilon$ is a matrix with many more zero elements than $\mathbf{Q}^-$ and with relatively small non-zero elements. Henceforth we use the term *quasi-lumpable* CTMC for a CTMC that is ϵ-quasi-lumpable for some ϵ.

An example[1] of $\mathbf{Q}$, $\mathbf{Q}^-$ and $\mathbf{Q}^\epsilon$ is given in Fig. 14.1 (we assume that $\lambda_2 = \lambda_1 + \epsilon$). The state partition is the following: A = { {1}, {2,3}, {4,5}, {6}}.

$\mathbf{Q} =$

	1	2	3	4	5	6
1	•	$2\lambda_1$	λ_2			
2		•		λ_1	λ_2	
3			•		$2\lambda_1$	
4				•		λ_2
5					•	λ_1
6	μ					•

$= \mathbf{Q}^- + \mathbf{Q}^\epsilon =$

	1	2	3	4	5	6
1	•	$2\lambda_1$	λ_2			
2		•		λ_1	λ_1	
3			•		$2\lambda_1$	
4				•		λ_1
5					•	λ_1
6	μ					•

$+$

	1	2	3	4	5	6
1						
2					ϵ	
3						
4						ϵ
5						
6						

Figure 14.1: An example of $\mathbf{Q}$, $\mathbf{Q}^-$, $\mathbf{Q}^\epsilon$.

Quasi Lumpable MC and High Level Modeling Formalisms In Chapter 13 it has been shown how it is possible to take advantage of a a SWN description of a system, in which the structural symmetries are evident, to automatically generate a lumped state space [1]. Sometimes however the lumping is prevented by some *small* perturbation in the transition rates. In this chapter we will explain how the approach presented in Chapter 13 can be extended to cases in which the state space has only a quasi-lumpable structure. Let us introduce a simple example that will be used throughout the chapter to illustrate the concepts.

Example 14.1 Consider the system depicted in Fig. 14.2.(a) and assume that there are 5 customers in the system. Customers cycle between two service centers. The first one is a single server whose service time is an exponentially distributed random variable with parameter μ. The second is a multiple server machine with four servers. A customer chooses randomly one of the four servers on each visit to the multiserver node. Let's assume that the choice is equiprobable for all queues. We also assume that the service time at each server $S_i, i = 1, \ldots, 4$ is exponential and does not depend on the customer identity. In Fig. 14.2.(b) a SWN representation of the system is depicted. There is only one color class (*Lines*) representing the servers in the multiserver, comprising one element for each of the four servers in the second stage. The clients flowing in the system are represented by tokens in the places. The indistinguishable tokens in place *Line*0 represent customers in the first server. The tokens in place

[1] The diagonal elements of $\mathbf{Q}$ are equal to the negative sum of all off diagonal elements in the row, as usual in infinitesimal generators.

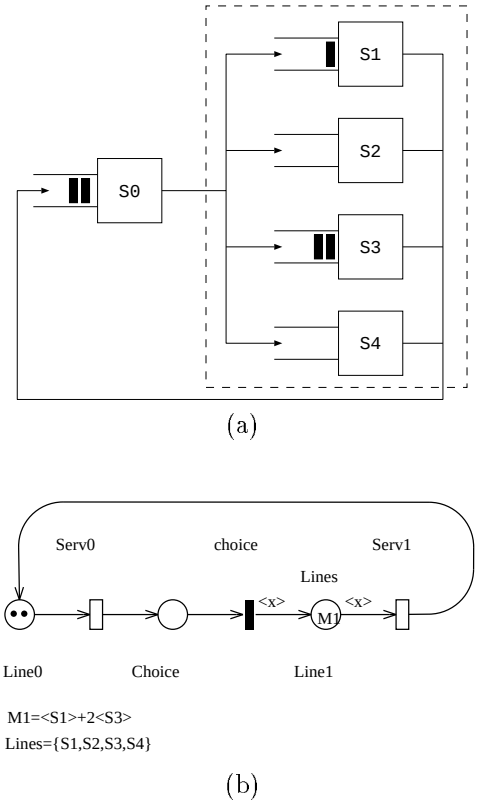


Figure 14.2: A Multiple Server System Model.

$Line1$ represent customers in the four parallel servers and take the color that corresponds to the queue that they join. The event associated with $\langle choice, S_i \rangle$ is the choice of joining the queue of server $Si,\ i = 1, \ldots, 4$ after exiting S_0 (this event is supposed to take no time). The event associated with $\langle Serv1, S_i \rangle$ is the completion of a service by S_i.

Observe that as long as the four servers in the multiserver have the same service rate and the four queues are chosen using a uniform distribution, class $Lines$ can be specified as a unique static subclass; if instead the servers in the multiservers had different service rates or were chosen by the customers with non uniform probability, then colour class $Lines$ must be partitioned in several static subclasses. For example if servers S_1 and S_2 had firing rate λ_1 while S_3 and S_4 had firing rate $\lambda_2 \neq \lambda_1$ then we should partition the class into two static subclasses: $Lines = Lines_1 \cup Lines_2$ and the rate of transition $Serv1$ should be defined accordingly.

The RG of the SWN in Fig. 14.2(b) has 126 tangible states, hence the corresponding CTMC has 126 states. The SRG of the same model, when $Lines$ is composed of a unique static subclass has 18 tangible states, hence the corre-

sponding lumped CTMC has 18 *aggregate* states. Finally, when *Lines* is composed of two static subclasses of cardinality two, the SRG contains 50 tangible states. In this case the corresponding CTMC is quasi-lumpable with respect to the aggregation induced by the SRG algorithm when *Lines* contains a single static subclass.

So quasi-lumpability is a property that can be observed in systems where a regularity in the state space structure is only partially reflected in the transition rates. Very often in this situation the modeler might decide to introduce an approximation and force the transition rates to become equal, thus forcing the model to become lumpable, using some kind of weighted average for the modified transition rates. This approach however may produce misleading results. The method proposed in the next section is safer from this point of view since it gives bounds rather than an approximation of uncertain error.

Before proceeding in the presentation of the method for the computation of bounds, let us comment on the possible interpretation of the different types of aggregation obtained partitioning class *Lines* as needed to properly define the timing of actions. The most intuitive representation of the state of the system depicted in Fig. 14.2 (suggested by the queueing network model) is a five element vector whose i^{th} element represents the number of clients in the i^{th} queue. Let us call this state representation, R_1. However, as long as the service rates of the four servers that work in parallel ($S_1 - S_4$) are the same (e.g., λ), a more abstract representation R_2 for the state can be used, namely $\langle n_0, nq_0, nq_1, \ldots, nq_5 \rangle$ where n_0 is the number of customers in the first queue, and $nq_j,\ j = 0, \ldots, 5,$ is the number of queues of length j in the second stage of four servers. Hence, the state $\langle 1, 1, 2, 1, 0, 0, 0 \rangle$ of type R_2 comprises several (more detailed) states of type $R1$, for example $\langle 1, 2, 1, 1, 0 \rangle$ and $\langle 1, 1, 0, 1, 2 \rangle$.

In case the four servers in parallel did not have the same speed, for example two of them had a given speed λ_1 while the other two had speed $\lambda_2 > \lambda_1$, then the first description would still be too detailed while the second would be too abstract. The appropriate state representation (called R_3) in the latter case is $\langle n_0, nq_0^1, \ldots, nq_5^1, nq_0^2, \ldots, nq_5^2 \rangle$ where nq_j^i is the number of queues of length j among the servers with speed λ_i.

The SRG technique automatically choses the more appropriate state representation for the SWN model of Fig. 14.2(b) based on the way colour class *Lines* is partitioned into static subclasses. Tab. 14.1 lists the 18 symbolic markings of the system corresponding to the 18 states obtained when using representation R_1.

When a reward process is associated with the CTMC, then the reward function definition should also be taken into account when checking lumpability (see [9]). A sufficient (though strong) condition is that the reward function gives the same value for every state in an aggregate[2]. We'll see how this condition can be relaxed in our framework of bounding.

[2]Observe that when applying the lumping technique presented in Chapter 13 this constraint can be dropped because of the equiprobability of the ordinary markings belonging to the same symbolic marking.

State	Symbolic Marking	
1	$Line0(5)$ $Line1()$,	$\|Z_L^0\|$ =4
	multiserver: 4 empty queues	
2	$Line0(4)$ $Line1(\langle Z_L^0\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 3$
	multiserver: 1 queue of length 1	
3	$Line0(3)$ $Line1(\langle Z_L^0\rangle)$,	$\|Z_L^0\| = 2, \|Z_L^1\| = 2$
	multiserver:2 queues of length 1	
4	$Line0(3)$ $Line1(2\langle Z_L^0\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 3$
	multiserver: 1 queue of length 2	
5	$Line0(2)$ $Line1(\langle Z_L^0\rangle)$,	$\|Z_L^0\| = 3, \|Z_L^1\| = 2$
	multiserver: 3 queues of length 1	
6	$Line0(2)$ $Line1(\langle Z_L^0\rangle, 2\langle Z_L^1\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 1, \|Z_L^2\| = 2$
	multiserver: 1 queue of length 1,	1 queue of length 2
7	$Line0(2)$ $Line1(3\langle Z_L^0\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 3$
	multiserver: 1 queue of length 3	
8	$Line0(1)$ $Line1(\langle Z_L^0\rangle)$,	$\|Z_L^0\| = 4$
	multiserver: 4 queues of length 1	
9	$Line0(1)$ $Line1(\langle Z_L^0\rangle, 2\langle Z_L^1\rangle)$,	$\|Z_L^0\| = 2, \|Z_L^1\| = 1, \|Z_L^2\| = 1$
	multiserver: 2 queues of length 1,	1 queue of length 2
10	$Line0(1)$ $Line1(2\langle Z_L^0\rangle)$,	$\|Z_L^0\| = 2, \|Z_L^1\| = 2$
	multiserver: 2 queues of length 2	
11	$Line0(1)$ $Line1(\langle Z_L^0\rangle, 3\langle Z_L^1\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 1, \|Z_L^2\| = 2$
	multiserver: 1 queue of length 1,	1 queue of length 3
12	$Line0(1)$ $Line1(4\langle Z_L^0\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 3$
	multiserver: 1 queue of length 4	
13	$Line0(0)$ $Line1(\langle Z_L^0\rangle, 2\langle Z_L^1\rangle)$,	$\|Z_L^0\| = 3, \|Z_L^1\| = 1$
	multiserver: 3 queues of length 1,	1 queue of length 2
14	$Line0(0)$ $Line1(\langle Z_L^0\rangle, 2\langle Z_L^1\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 2, \|Z_L^2\| = 1$
	multiserver: 1 queue of length 1,	2 queues of length 2
15	$Line0(0)$ $Line1(\langle Z_L^0\rangle, 3\langle Z_L^1\rangle)$,	$\|Z_L^0\| = 2, \|Z_L^1\| = 1, \|Z_L^2\| = 1$
	multiserver: 2 queues of length 1,	1 queue of length 3
16	$Line0(0)$ $Line1(2\langle Z_L^0\rangle, 3\langle Z_L^1\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 1, \|Z_L^2\| = 2$
	multiserver: 1 queue of length 2,	1 queue of length 3
17	$Line0(0)$ $Line1(\langle Z_L^0\rangle, 4\langle Z_L^1\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 1, \|Z_L^2\| = 2$
	multiserver: 1 queue of length 1,	1 queue of length 4
18	$Line0(0)$ $Line1(5\langle Z_L^0\rangle)$,	$\|Z_L^0\| = 1, \|Z_L^1\| = 3$
	multiserver: 1 queue of length 5	

Table 14.1: Symbolic Reachablity Set of the model of Fig. 14.2(a).

14.2 Bounds for quasi-lumpable markov chains

In this section we first summarize the Courtois and Semal's results that we are going to use; then we show how a quasi-lumpable MC can be transformed into a lumpable one and how Courtois and Semal's methods can be applied to it.

14.2.1 The Courtois and Semal bounding methods

In this section we summarize Courtois and Semal's results [2, 3] that are relevant to our discussion. The results reported here are specific to stochastic matrices, and are thus less general than those presented in [2, 3]; the reader can refer to the original papers for the more general theorems and their proofs (which are for non-negative matrices). The theorems presented in this section are stated in terms of transition probability matrices as in the original papers, rather than in terms of infinitesimal generators, hence they apply to Deterministic Time Markov Chains (DTMC), however they can be easily adapted to work on the CTMC associated with an SWN model, as we shall see in the next sections.

Let $\mathbf{P}$ be a stochastic matrix, and let $\mathbf{P}^-$ and $\mathbf{P}^+$ be a lower and upper bound (componentwise) for $\mathbf{P}$ respectively. We assume that $\mathbf{P}$, $\mathbf{P}^-$ and $\mathbf{P}^+$ are all irreducible.

Lemma 14.2 *[2]*

- $(I - \mathbf{P}^-)$ *is non-singular*
- $(I - \mathbf{P}^-)^{-1} \geq 0$
- $(I - \mathbf{P}^-)^{-1}\mathbf{e}^T > 0$

where $\mathbf{e}^T$ *denotes a properly dimensioned column of ones.*

Theorem 14.3 *[2] Let* $\mathbf{P}$ *be a stochastic matrix, and let* $\mathbf{P}^-$ *be a lower bound for* $\mathbf{P}$*. We assume that both* $\mathbf{P}$ *and* $\mathbf{P}^-$ *are irreducible. The steady state probability* $\boldsymbol{\pi}$ *of* $\mathbf{P}$ *belongs to the polyhedron whose vertices are the rows* $\{z_j\}$ *of the matrix* Z *with indices in the set* $J = \{j \in 1, \ldots, n, \ s.t. \ \exists i : P_{i,j} > \mathbf{P}^-{}_{i,j}\}$ *where*

$$Z = \Omega^{-1}(I - \mathbf{P}^-)^{-1}$$

and Ω *is a diagonal normalization matrix defined as follows:*

$$\Omega = diag((I - \mathbf{P}^-)^{-1}\mathbf{e}^T)$$

Hence there exists a vector β *s.t.*

- $\beta \ \mathbf{e}^T = 1$
- $\beta_j = 0, \ j \notin J$
- $\boldsymbol{\pi} = \beta \ Z$

and

$$\forall i,\ \pi_i^- \leq \pi_i \leq \pi_i^+$$

where

$$\pi_i^- = max\{min_{k\in J}(Z)_{k,i}; 1 - \sum_{j\neq i} max_{k\in J}(Z)_{k,j}\}$$

and

$$\pi_i^+ = min\{max_{k\in J}(Z)_{k,i}; 1 - \sum_{j\neq i} min_{k\in J}(Z)_{k,j}\}$$

The following theorem gives an alternative way for obtaining bounds without computing inverses.

Theorem 14.4 *[2] Given a lower bound* $\mathbf{P^-}$ *for* $\mathbf{P}$*, the* k^{th} *row of matrix* Z *is the steady state probability vector of the matrix obtained by incrementing the* k^{th} *column of* $\mathbf{P^-}$ *in order to make it stochastic.*

Thus if we have only a lower bound $\mathbf{P^-}$ of the transition probability matrix of a given CTMC (let's assume that the chain has n states), we can still compute bounds for its steady state probability vector $\boldsymbol{\pi}$. This is done by computing the solution of n CTMCs; the transition probability matrix P_i of the i^{th} CTMC is obtained from the substochastic matrix $\mathbf{P^-}$ by increasing the elements of column i enough to make P_i stochastic. Let $\boldsymbol{\pi}^i$ be the steady state probability vector solution of the i^{th} CTMC. The lower (upper) bound on the steady state probability of state k is computed as $min_i\ \pi_k^i$ ($max_i\ \pi_k^i$).

The next theorem is analogous to Theorem 14.3 when an upper bound $\mathbf{P^+}$ of $\mathbf{P}$ is available instead of a lower bound. However this result is slightly weaker because the inverse is not guaranteed to exist; furthermore, even when it does exist some additional constraints have to be tested in order to derive bounds from it (i.e., there is no result similar to Lemma 14.2 in this case).

Theorem 14.5 *[2] Let* $\mathbf{P}$ *be a stochastic matrix, and let* $\mathbf{P^+}$ *be an upper bound for* $\mathbf{P}$*. We assume that both* $\mathbf{P}$ *and* $\mathbf{P^+}$ *are irreducible. If the following inverse matrix*

$$(I - \mathbf{P^+})^{-1}$$

exists and satisfies

$$(I - \mathbf{P^+})^{-1}\mathbf{e}^T < \mathbf{0}^T \qquad \mathbf{e}\ (I - \mathbf{P^+})^{-1} < \mathbf{0}$$

then there exists a vector β *s.t.*

- $\beta\mathbf{e}^T = 1$
- $\boldsymbol{\pi} = \beta\Omega^{-1}(I - \mathbf{P^+})^{-1}$

where Ω^{-1} *is a diagonal normalization matrix.*

In [2, 3] two applications of the above theorems are presented.

1. Computation of bounds on conditional steady state probability of a subset of states $\mathcal{S}$ in a CTMC when the transition probability among the states in the subset is known while only partial or no information is available about the transition probability between states in $\bar{\mathcal{S}}$ and states in $\mathcal{S}$. Of course the amount of information available affects the accuracy of bounds.

   The less accurate and at the same time less computationally expensive way of obtaining these bounds is to applying Theorem 14.3 using the submatrix $\mathbf{P}_{\mathcal{S},\mathcal{S}}$ of transition probabilities among states in $\mathcal{S}$ as $\mathbf{P}^-$.

2. Computation of bounds on the steady state probability vector of a large system by decomposition into smaller subsystems: this is called the *bounded aggregation method*. This method consists of two steps:

   - computation of bounds on the conditional state probabilities within each aggregate using the method just explained;
   - computation of bounds on the probability of being in each aggregate. This part is divided into two steps: (1) derivation of a lower bound $\mathbf{P}_a^-$ for the inter-aggregate probability matrix $\mathbf{P}_a$ using the results of previous step; (2) application of Theorem 14.3 to $\mathbf{P}_a^-$ to compute bounds on the aggregates probability vector.

   Observe that if we knew the vector of conditional probabilities $\boldsymbol{\pi}^{\mathcal{S}}$ of the states in aggregate $\mathcal{S}$, then we could easily compute the transition probability $\mathbf{P}_a[\mathcal{S},\mathcal{S}']$ between aggregates $\mathcal{S}$ and $\mathcal{S}'$ as $\sum_{s,s':s\in\mathcal{S},s'\in\mathcal{S}'} \boldsymbol{\pi}_s^{\mathcal{S}} \mathbf{P}_{s,s'}$. Since we know only bounds on $\boldsymbol{\pi}^{\mathcal{S}}, \forall\mathcal{S}$, we are able to compute only a bound $\mathbf{P}_a^-$ for $\mathbf{P}_a$.

These techniques can be used to compute bounds not only on the steady state probability vector, but also on any kind of a performance metric defined in terms of a reward function r from the set of states to $\mathbb{R}$. Let r_j denote the reward associated with state j, and Let $\boldsymbol{\pi}^i$ denote the steady state probability vector obtained by solving the CTMC characterized by the matrix $\mathbf{P}_i$ obtained from $\mathbf{P}^-$ by increasing its i^{th} column such as to make it stochastic. The mean accumulated reward $\bar{R}$ is defined as

$$\bar{R} = \boldsymbol{\pi} r^T = \sum_i \pi_i r_i$$

and can be rewritten as

$$\bar{R} = \sum_i (\sum_j \beta_j \pi_i^j) r_i = \sum_j \beta_j \sum_i \pi_i^j r_i.$$

The term $\sum_i \pi_i^j r_i$ is equal to the mean accumulated reward $\bar{R}_j = \boldsymbol{\pi}^j r^T$ computed from $\mathbf{P}_j^-$ and

$$\bar{R} = \sum_i \beta_i \bar{R}_i$$

so that $min_i \bar{R}_i$ and $max_i \bar{R}_i$ are a lower and an upper bound for $\bar{R}$ respectively.

14.2.2 The proposed method

Transforming a Quasi-Lumpable MC into a lumpable MC The definition of a quasi-lumpable MC contains the first hint on how to proceed in transforming the quasi-lumpable chain into a lumpable one: with reference to matrix $\mathbf{Q}$ in Fig. 14.1, if the values of ϵ were small enough we could be tempted to just use $\mathbf{Q}^- + diag(\mathbf{Q}^\epsilon \mathbf{e}^T)$ instead of $\mathbf{Q}$ and obtain an approximate result from the lumped version of this new matrix.

Even if this method leads to a good approximation in some cases, in general it is difficult to forecast the actual influence of the approximation on the result both from a quantitative point of view (how far is the approximation from the actual result?) and a qualitative point of view (does the approximation lay above or below the correct result?).

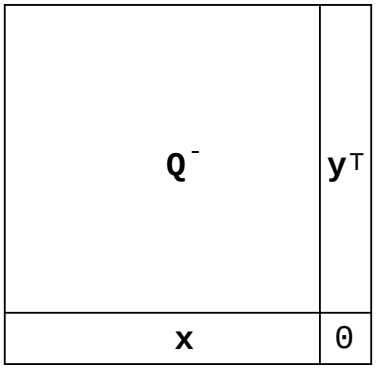


Figure 14.3: Modified matrix $\mathbf{Q}^s$.

Instead of just "forgetting" about the elements in $\mathbf{Q}^\epsilon$, we could imagine redirecting these "*disturbing*" transition rates (or probabilities if we refer to transition probability matrices instead of infinitesimal generators) to a new extra state s. By doing this we end up with a new stochastic matrix $\mathbf{Q}^s$ that has one more state than $\mathbf{Q}^-$. The new matrix is depicted in Fig. 14.3; vectors $\mathbf{x}$ and $\mathbf{y}^T$ are defined as follows:

- $\mathbf{y}^T = \mathbf{Q}^\epsilon \mathbf{e}^T = (\mathbf{Q} - \mathbf{Q}^-)\mathbf{e}^T$
  where $\mathbf{e}^T$ is a column vector of 1s, of appropriate dimension; $\mathbf{y}^T$ is a column vector whose elements are the row sums of $\mathbf{Q}^\epsilon$ (hence the interpretation of element i of vector $\mathbf{y}^T$ as the global rate from state i to state s due to a redirection of rates);

- $\mathbf{x}$ is a row vector of the same dimension as a row of $\mathbf{Q}^-$; right now we do not define its contents but we do impose the constraint $\mathbf{x}\mathbf{e}^T = 1$.

We claim that there exists an assignment of values to the elements of $\mathbf{x}$ such that the steady state probability of MC $\mathbf{Q}$ is equal to the conditional probability

of the first n states[3] of chain $\mathbf{Q}^s$. In other words, if we redistribute "properly" the transition probabilities from the extra state to the states that have had a decrement in the incoming transition probabilities, we end up with the same conditional probability distribution as in the original model.

Theorem 14.6 *Let* $\mathbf{Q}$ *be the transition probability matrix of an ergodic CTMC with n states. Let* $\mathbf{Q^-}$ *be a lower bound (componentwise) for* $\mathbf{Q}$*, i.e.,* $\mathbf{Q^-} \leq \mathbf{Q}$*. Let* $\mathbf{y}^T = (\mathbf{Q} - \mathbf{Q^-})\mathbf{e}^T$ *and finally let* $\mathbf{Q}^s$ *be the stochastic matrix of Fig. 14.3.*

There exists a vector $\mathbf{x}$ *such that the conditional steady state probabilities for the first n states of* $\mathbf{Q}^s$ *is equal to the steady state probabilities of CTMC* $\mathbf{Q}$*.*

Proof: The theorem is proven by showing that the vector x defined as:

$$\mathbf{x} = \frac{\boldsymbol{\pi}(\mathbf{Q} - \mathbf{Q^-})}{\boldsymbol{\pi}(\mathbf{Q} - \mathbf{Q^-})\mathbf{e}^T} \tag{14.1}$$

where $\boldsymbol{\pi}$ is the steady state probability vector for $\mathbf{Q}$, satisfies the requirements stated in the theorem (see [4]). □

At this point we can use Courtois and Semal's technique to compute bounds on the conditional probabilities of a subset of states in a system when only the transition probabilities among the states in the subset are known [3]. In this case the subset of states in the system represented by $\mathbf{Q}^s$ is the set of original states, and the rest of the system is the extra state. The bounds can be computed without knowledge of the transition probabilities from the rest of the system (state s) to the selected subset of states (original states), hence there is no problem in not knowing the values of the elements of $\mathbf{x}$. However if something is known about $\mathbf{x}$ then better bounds can be computed. Some information about $\mathbf{x}$ can be easily derived from $\mathbf{Q}^\epsilon$, namely any column j of $\mathbf{Q}^\epsilon$ with all zero elements, corresponds to a zero element in $\mathbf{x}$. Theorem 14.3 takes into account this factor by defining the set J of Z rows that contribute to the computation of bounds.

So far, the lumpability property of $\mathbf{Q^-}$ has not been exploited. Observe that the MC $\mathbf{Q}^s$ in general does not satisfy the lumpability conditions with respect to a given state space partition A.

For example matrix $\mathbf{Q}^s$ depicted below and derived from matrix $\mathbf{Q}$ of Fig. 14.1 is not lumpable.

$\mathbf{Q}^s =$

	1	2	3	4	5	6	s
1	♦	$2\lambda_1$	λ_2				
2		♦		λ_1	λ_1		ϵ
3			♦		$2\lambda_1$		
4				♦		λ_1	ϵ
5					♦	λ_1	
6	μ					♦	
s	x						0

But now let's modify the column $\mathbf{y}^T$ so that the lumpability conditions are satisfied. This can be done by increasing some of the elements of $\mathbf{y}^T$ and decreasing the matrix diagonal elements by the same quantity. The modified matrix $\tilde{\mathbf{Q}}^s$ obtained in this way is lumpable.

[3] n is the number of states in the original chain $\mathbf{Q}$.

$\tilde{\mathbf{Q}}^s =$

	1	2	3	4	5	6	s
1	♦	$2\lambda_1$	λ_2				
2		♦		λ_1	λ_1		ϵ
3			♦ − ϵ		$2\lambda_1$		ϵ
4				♦		λ_1	ϵ
5					♦ − ϵ	λ_1	ϵ
6	μ					♦	
s	x						0

It is then possible to derive the lumped matrix $\mathbf{Q}_a^s$ shown below.

$\mathbf{Q}_a^s =$

	a1	a2	a3	a4	s
a1	♦	$2\lambda_1 + \lambda_2$			
a2		♦	$2\lambda_1$		ϵ
a3			♦	λ_1	ϵ
a4	μ			♦	
s	x				0

Moreover the conditional probabilities of the aggregate states are still bounds for the sum of the probabilities in the aggregates computed on the original model. This statement is true because in the above proof we didn't make any assumption about $\mathbf{Q}^-$ and $\mathbf{Q}^\epsilon$ other than $\mathbf{Q}^- \leq \mathbf{Q}$. In the sequel, n' is used to denote the number of aggregate states (i.e., $n' = |A|$) and $\mathcal{S}$ is used to denote the first n' states of $\mathbf{Q}_a^s$. Courtois and Semal's method can now be applied to the new lumped model.

Bounds on steady state probabilities The procedure to be followed once the lumped matrix $\mathbf{Q}_a^s$ and rates redirection vector $\mathbf{y}_a^T$ have been obtained, is to compute the steady state probability vector of (at most) as many Markov chains as the number n' of aggregates (macro states or symbolic markings) in the lumped Markov chain. The i^{th} chain to be solved is obtained by adding the rates redirection vector $\mathbf{y}_a^T$ to the i^{th} column of the $n' \times n'$ upper left submatrix of $\mathbf{Q}_a^s$ (see Theorem 14.4). This is equivalent to setting the i^{th} entry of vector x to 1 in $\mathbf{Q}_a^s$ and then computing the conditional steady state probability distribution of the first n' states. Intuitively, this corresponds to redirecting all the *disturbing* rates to the i^{th} (aggregate) state. Let $\boldsymbol{\pi}^i$ be the steady state probability vector computed from this modified model. Theorem 14.3 states that the correct steady state probability vector $\boldsymbol{\pi}$ can be computed as a weighted sum of the n' solutions $\boldsymbol{\pi}^i, i = 1, \ldots, n'$:

$$\pi_i = \sum_{k=1}^{n'} \beta_k \pi_i^k$$

where $\sum_{k=1}^{n'} \beta_k = 1$.

The problem is that we do not know the appropriate weights β_j to compute the exact solution $\boldsymbol{\pi}$, however from the $\boldsymbol{\pi}^i$ we can obtain upper and lower bounds for the probability of each state using the following formulas:

$$\pi_i^- = max\{min_{k \in 1,\ldots,n'} \pi_i^k \; 1 - \sum_{j \neq i} max_{k \in 1,\ldots,n'} \pi_j^k\}$$

$$\pi_i^+ = min\{max_{k \in 1,\ldots,n'} \pi_i^k; 1 - \sum_{j \neq i} min_{k \in 1,\ldots,n'} \pi_j^k\}$$

An improvement in the efficiency and accuracy of the method can be obtained by observing that if all the columns belonging to aggregate k in $\mathbf{Q}^s$ are equal to the corresponding columns in $\mathbf{Q}$, then $\beta_k = 0$, and hence the k^{th} modified model can be ignored (and hence not solved) for the computation of the bounds.

Bounds on accumulated reward Often the performance indices of interest are not the steady state probabilities, but rather the accumulated reward given a reward function r from the set of states to $\mathbb{R}$. When the bounding method proposed has to be applied, it may be the case that aggregated states do not have the same associated reward. Of course if every state in aggregate A_i had the same reward r_{A_i}, then the aggregate reward would be r_{A_i}, otherwise the computation of the exact reward associated with the aggregate would require the knowledge of the conditional probability distribution $\boldsymbol{\pi}^{A_i}$ of the states in the aggregate:

$$r_{A_i} = \sum_{s \in A_i} \pi_s^{A_i} r_s.$$

In this case the method described in Section 14.2.1 for the computation of bounds on the mean accumulated reward needs some minor adjustment. For each aggregate A_i we have to compute the minimum and maximum reward rate:

$$r_{A_i}^- = min_{s \in A_i} r_s$$

$$r_{A_i}^+ = max_{s \in A_i} r_s$$

then instead of computing $\bar{R}_i$ we compute $\bar{R}_i^+$ using for each aggregate the maximum reward and $\bar{R}_i^-$ using for each aggregate the minimum reward. The lower and upper bounds for $\bar{R}$ are then derived as $min_i \bar{R}_i^-$ and $max_i \bar{R}_i^+$ respectively. Notice that this is the best we can do if we don't have any information about the conditional probability of the states in each aggregate, but this approach could result in loose bounds.

Interpretation in terms of Bounded Aggregation The previous method can be interpreted in a different way, as a simplified version of the bounded aggregation technique [2]. However, the former interpretation is useful for devising bound improvement methods as we'll see in the next section.

The bounded aggregation method consists of two steps:

1. computation of bounds on the conditional state probabilities within each aggregate.

2. computation of bounds on the probability of being in each aggregate.

The bounds obtained in the two steps are then combined to obtain bounds on the steady state probability vector over the original set of states.

We are actually interested in computing only the probability vector of the aggregates (second step) since our aim is to deal directly with the lumped process. The $n' \times n'$ upper-leftmost submatrix $\mathbf{Q}_a^-$ of the lumped matrix $\mathbf{Q}_a^s$ derived

in the previous paragraph is a lower bound for the correct inter-aggregates transition probability matrix $\mathbf{Q}_a$ associated with $\mathbf{Q}$ given state partition A; indeed the correct transition probability between aggregate I and aggregate J is a convex combination of the rowsums in the submatrix $P_{I,J}$ (where the weight used for each row in the submatrix is the conditional probability of the corresponding state in aggregate I). In our approach we are giving a weight of one to the minimum rowsum, thus obtaining a lower bound with respect to the correct convex combination of the rowsums. In our example, matrix $\mathbf{Q}_{A2,A3}$ is

	4	5
2	λ_1	λ_2
3		$2\lambda_1$

and the transition probability between $A2$ and $A3$ in $\mathbf{Q}_a^-$ is $min(2\lambda_1, \lambda_1 + \lambda_2)$ while the corresponding value in $\mathbf{Q}_a$ would be $2\alpha\lambda_1 + (1-\alpha)(\lambda_1 + \lambda_2)$ for some $0 \leq \alpha \leq 1$. Observe that an upper bound, $\mathbf{Q}_a^+$, for the probability between two aggregates could be computed as well by choosing the maximum among the rowsums in the submatrix (in the example above $max(2\lambda_1, \lambda_1 + \lambda_2)$ is an upper bound for the transition probability between A_2 and A_3).

Example 14.2 The model of Fig. 14.2 is a good candidate for the application of the method. In fact this is a typical case in which the partition of the servers in two subclasses with equal speed within each subclass leads to a quasi-lumpable MC structure. We assume that the rates λ_1 and λ_2 associated with the two classes of servers are $\lambda_1 = 1.00$ and $\lambda_2 = 1.01$. The performance indices of interest for us are the throughput out of[4] S_0 and the mean queue length at the entrance of the multiple server. The reward rates for all aggregates used to compute the performance indices are shown in Fig. 14.4. Observe that in this example we were able to compute an exact reward for each aggregate because all the aggregated states have the same associated reward.

State #	1	2	3-4	5-7	8-12	13-18
MQL	0	1	2	3	4	5
Throughput	μ	μ	μ	μ	μ	0

Figure 14.4: Reward rates.

In Fig. 14.5 the aggregated matrix (split into two parts due to space constraints) and the vector $\mathbf{y}^T$ are shown . All we know about $\mathbf{x}$ is that its 13^{th} element must be zero (because the corresponding column is not modified to derive $\mathbf{Q}_a^-$ from $\mathbf{Q}$). The description of the states for the aggregated model is given in Tab. 14.1. The state description for the complete model (corresponding to the intermediate detail level representation R_2 presented in Section 14.1) is not reported here for space reasons.

[4] μ is the service rate of S_0.

	1	2	3	4	5	6	7	8	9
1	-1.00	1.00							
2	1.00	-2.00	0.75	0.25					
3		2.00	-3.00		0.50	0.50			
4		1.00		-2.00		0.75	0.25		
5			3.00		-4.00			0.25	0.75
6			1.00	1.00		-3.00			0.50
7				1.00			-2.00		
8					4.02			-5.02	
9					1.00	2.00			-4.00
10						2.00			
11						1.00	1.00		
12							1.00		
13								1.00	3.00
14									2.00
15									1.00
16									
17									
18									

	10	11	12	13	14	15	16	17	18
1									
2									
3									
4									
5									
6	0.25	0.25							
7		0.75	0.25						
8				1.00					
9				0.25	0.50	0.25			
10	-3.00				0.50		0.50		
11		-3.00				0.50	0.25	0.25	
12			-2.00					0.75	0.25
13				-4.00					
14	1.00				-3.00				
15		2.00				-3.00			
16	1.00	1.00					-2.00		
17		1.00	1.00					-2.00	
18			1.00						-1.00

	1	2	3	4	5	6	7	8	9
y[1-9]	0.00	0.01	0.02	0.01	0.02	0.02	0.02	0.00	0.02
	10	11	12	13	14	15	16	17	18
y[10-18]	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.01

Figure 14.5: The aggregated matrix $\mathbf{Q}_a$ and the corresponding vector $\mathbf{y}^T$.

	Lower bound	Exact value	Upper bound
Prob(State 1)	0.307430	0.321631	0.329239
Prob(State 2)	0.307430	0.320039	0.323022
Prob(State 3)	0.115106	0.119420	0.122238
Prob(State 4)	0.078126	0.079616	0.084354
Prob(State 5)	0.019179	0.019804	0.022648
Prob(State 6)	0.057889	0.059415	0.063442
Prob(State 7)	0.019131	0.019806	0.026669
Prob(State 8)	0.001198	0.001232	0.003351
Prob(State 9)	0.014373	0.014780	0.018140
Prob(State 10)	0.007139	0.007389	0.011374
Prob(State 11)	0.014219	0.014782	0.019938
Prob(State 12)	0.004703	0.004928	0.012848
Prob(State 13)	0.001194	0.001226	0.003879
Prob(State 14)	0.003562	0.003677	0.007472
Prob(State 15)	0.003544	0.003677	0.007506
Prob(State 16)	0.003527	0.003677	0.009229
Prob(State 17)	0.003506	0.003677	0.009698
Prob(State 18)	0.001164	0.001226	0.012440
Mean Queue Length	1.250705	1.273430	1.385868
Throughput	0.967612	0.98284	0.983502

Table 14.2: Results of the bounds computation for the multiserver example.

Solving the 17 (aggregated) models (each with only one element of $\mathbf{x}$ equal to 1 and the others equal to 0) we obtain the steady state probabilities, mean queue length, and throughput bounds shown in Table 14.2. The table also contains the correct values computed on the complete model (that has 50 states). The spread of the bounds for the mean queue length and throughput is 11% and 1.6% respectively.

The example model satisfies the constraints for the applicability of Theorem 14.5 that exploits the upper bound $\mathbf{Q}_a^+$, so that we could use that theorem to obtain improved bounds. In Section 14.4 we show a technique for bounds improvement that doesn't require the computation of the inverse and the verification of constraints to be applied.

The reader may object that good bounds for the throughput and mean queue length of this specific example could be found by just observing that the (actually lumpable) model obtained by setting the service time of all the servers in the multiple server to the minimum among the service times in the original model, gives an upper bound for the throughput and a lower bound for the queue length. If the maximum among the service times in the original model is assigned to the servers, then we get a lower bound for the throughput and an upper bound for the queue length. Note however that it is not possible to compute bounds on the state probability distribution in the same simple way. Another possible objection is that the model is a product form queueing network, and efficient algorithms exist for the computation of exact results, however if we changed

the policy for choosing the queue to join in the multiserver to a load dependent one (e.g., shortest-queue), product form solutions would not be applicable any more, while our method still applies. This example was chosen on purpose for its simplicity, in general it is not easy to prove that by lowering/increasing a rate in the system we get a bound on a given performance index. In Section 14.5 a more complex example is presented in which the presence of synchronization among the system components makes it difficult to find any intuitive argument for simple bounds computation.

14.3 Discussion on the advantages of the method

In this section we'll discuss the proposed method from the point of view of its convenience in terms of computational complexity. The main advantage of the method is that it allows us to deal with a smaller system than the one we start with, however there is a price we have to pay to be able to work with the reduced model. In fact there are at least two kinds of overhead we have to take into account in evaluating the method:

- derivation of the reduced matrix;
- computation of bounds.

Concerning the derivation of the reduced matrix, the ideal situation is to derive the lumped matrices $\mathbf{Q}_a^-$ and $\mathbf{Q}_a^+$ directly from the high level model without having to compute (or at least to store) the large complete matrix. We'll show how this can be done when the high level formalism used to describe the system is that of SWNs.

The computational complexity issue instead, is related to the fact that the computation of bounds requires the solution of n' matrices of dimension $n' \times n'$ in the worst case. Later in this section we'll discuss how much smaller the lumped matrix must be for the method to be more efficient from the time complexity point of view, considering also the possibility of solution parallelization. The space complexity issue is discussed as well.

The issue of automatic construction of the lumped matrix is closely related to that of the detection of the symmetries on the state space that induce a partition into state aggregates. The same problem has to be faced when exact lumping methods are used, indeed the bounding method presented in this chapter can be considered as an extension of the existing exact lumping methods presented in Chapter 13.

Automatic construction of a lumped matrix from a SWN model In SWNs, color classes are used to define sets of "similar" objects and the partition of a class into subclasses is used to identify subsets of objects in a class that share the same behaviour. As already pointed out, it is possible to distinguish between two possible situations: (1) the objects in different subclasses have

different *qualitative* behaviour, i.e., they cannot play the same "role" in the system because they have different possible evolutions, (2) the objects in different subclasses have the same *qualitative* behaviour, but the event sequences happen with different rates depending on which subclass the object belongs to.
For example, in the model of Fig. 14.2 the colored tokens representing customers in the "multiserver" station follow the same route through transition $Serv1$, independently of their color, however, the firing rate of the transition depends on the token color.

Hence, given a SWN model of the system under study, a first analysis is needed to detect all the subclasses in each class with similar "qualitative" behaviour. This permits automatic determination of the candidate quasi-lumpable state aggregates. Then two approaches are possible, (1) apply the Symbolic Reachability Graph generation algorithm to the system as it is specified and use the information about aggregate states to compute the lumped matrices $\mathbf{Q}_a^-$ and $\mathbf{Q}_a^+$ in a second step; (2) apply the SRG generation algorithm to a modified system with some of the homogeneously behaving subclasses merged in such a way that $\mathbf{Q}_a^-$ and $\mathbf{Q}_a^+$ are directly derived.

Follows the description of the subclasses merge algorithm and the modified model construction algorithm[5], where the modified model is the one from which the aggregate matrices $\mathbf{Q}_a^-$ and $\mathbf{Q}_a^+$ can be directly derived.

The subclasses merge algorithm starts from the set of static subclasses of each colour class, and generates a (possibly) coarser partition into new static subclasses by merging old static subclasses sharing the same qualitative behaviour. Basically two static subclasses $C_{i,sc}$ and $C_{i,sc'}$ can be merged if for all transitions that have C_i in their color domain definition, no predicates of type $d(x) = sc$, $d(x) = sc'$, $d(x) = d(y)$ or their negation are used either in the transition predicate or in the function predicates labeling the arcs connected to the transition (i.e., neither the transition enabling conditions nor the transition state change can depend on the fact that some element in the transition color instance belong or not to sc or sc').

Aggregate model computation algorithm The aggregate matrices $\mathbf{Q}_a^-$ and $\mathbf{Q}_a^+$ are computed by applying the usual SRG generation algorithm to a modified model. The modified model has the same structure as the original one, while it differs from it in the static subclasses definition and, as a consequence, in the transition firing rates.

In a SWN, the transition firing rate of a generic transition t is defined as a function $\mathbf{w}[t]$ from tuples of static color classes to reals.

For example transition *compute* in Fig. 14.6 takes a token with its two component color from its input place: the first component represents a processor and the second represents a job. If the jobs class J is divided into two subclasses of short jobs J_s and long jobs J_l and the processors are divided into two subclasses of fast processors P_f and slow processors P_s, then a possible definition for the transition rate could be: $\mathbf{w}[t](\langle < P_s, J_s > \rangle) = 0.3$, $\mathbf{w}[t](\langle < P_f, J_s > \rangle) = 1$, $\mathbf{w}[t](\langle < P_s, J_l > \rangle) = 0.15$, $\mathbf{w}[t](\langle < P_f, J_l > \rangle) = 0.5$.

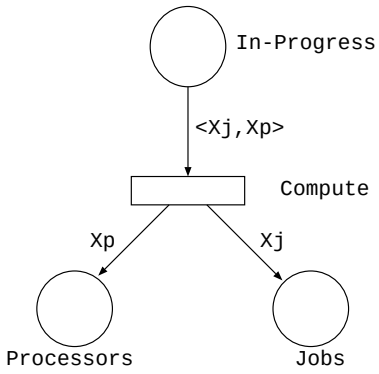


Figure 14.6: A simple SWN example.

When a symbolic transition instance $\langle t, z\rangle$ is fired, its rate λ is computed as follows: (1) derive from z the tuple d of corresponding static subclasses (each dynamic subclass is associated with exactly one static subclass, see [1]; (2) $\lambda = m\mathbf{w}[t](d)$ where m is a factor that depends on the cardinality of both the subclasses in z and in d. The reason for the multiplicative factor is that a symbolic firing instance is an aggregation of m regular firing instances all with the same rate $\mathbf{w}[t](d)$. In our example a possible symbolic instance for transition $Compute$ could be $\langle Compute, \langle Z_P^1, Z_J^2\rangle\rangle$ with $|Z_P^1| = 2, Z_P^1 \in P_s$, and $|Z_J^2| = 1, Z_J^2 \in J_l$ meaning that there are 2 ($= |Z_P^1|$) slow processors each of which is processing 1 ($= |Z_J^2|$) long job. This symbolic instance stands for 2 "ordinary" instances that are grouped in the lumping process.

In the following we show how to derive the modified model to compute the aggregate matrices given a specific aggregation of the static subclasses. We denote $C_{i,j}$ the new static subclasses and $\{C_{i,j}^l\}$ the set of the original subclasses that are aggregated into $C_{i,j}$.

The rates of the resulting new transitions, in general, will depend on the cardinality of the dynamic subclasses in the instantiation.

Let $\langle t, z\rangle$ be the transition instance for which a rate has to be computed. Let d be the associated static subclasses tuple. Let d' be the subtuple of d composed of aggregate static subclasses and z' the corresponding subtuple of z.

For each element $d_k = C_{i,j}$ in d', compute the set $\mathcal{Z}_k$ of possible partitions of dynamic subclass z_k into one or more new dynamic subclasses, each associated with a different static subclass $C_{i,j}^l$ of $C_{i,j}$ (see Fig. 14.7).
The Cartesian product of the $\mathcal{Z}_k$s leads to a set of sums of original transition instances. From each sum a rate can be computed that corresponds to a value for a rowsum in the aggregate transition represented by $\langle t, z\rangle$. The minimum and maximum in this set of rates gives the value for the corresponding transition rate in the aggregate matrices.

In the example of Fig. 14.6 assume that $|P_s| = |P_f| = 2$ and $|J_s| = 2, |J_l| = 1$, and suppose we want to merge slow and fast processors into a unique class $\mathbf{Q}$

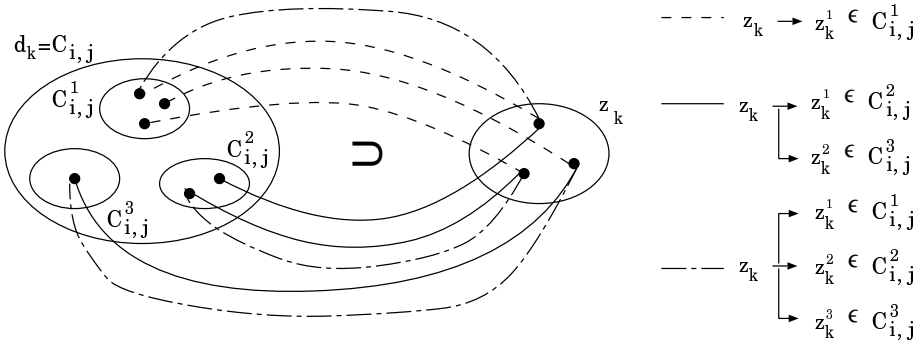


Figure 14.7: Possible partitions of z_k with respect to the "merged" static subclass $C_{i,j}$.

and short and long jobs into a unique class J. Hence the symbolic transition instance $\langle Compute, \langle Z^1_P, Z^2_J \rangle\rangle$ with $|Z^1_P| = 2, Z^1_P \in P$ and $|Z^2_J| = 2, Z^2_J \in J$ incorporates the following 6 possible instances combinations with respect to the old classes partition.

- $\langle Compute, \langle Z^{1a}_P, Z^{1a}_J \rangle\rangle + \langle Compute, \langle Z^{1a}_P, Z^{1b}_J \rangle\rangle$ with $|Z^{1a}_P| = 2$ and $Z^{1a}_P \in P_s$, $|Z^{1a}_J| = 1$ and $Z^{1a}_J \in J_s$, $|Z^{1b}_J| = 1$ and $Z^{1b}_J \in J_l$; corresponding rate 0.9

- $\langle Compute, \langle Z^{1a}_P, Z^{1a}_J \rangle\rangle + \langle Compute, \langle Z^{1a}_P, Z^{1b}_J \rangle\rangle$ with $|Z^{1a}_P| = 2$ and $Z^{1a}_P \in P_f$, $|Z^{1a}_J| = 1$ and $Z^{1a}_J \in J_s$, $|Z^{1b}_J| = 1$ and $Z^{1b}_J \in J_l$; corresponding rate 3

- $\langle Compute, \langle Z^{1a}_P, Z^{1a}_J \rangle\rangle$ with $|Z^{1a}_P| = 2$ and $Z^{1a}_P \in P_s$, $|Z^{1a}_J| = 2$ and $Z^{1a}_J \in J_s$; corresponding rate 1.2

- $\langle Compute, \langle Z^{1a}_P, Z^{1a}_J \rangle\rangle$ with $|Z^{1a}_P| = 2$ and $Z^{1a}_P \in P_f$, $|Z^{1a}_J| = 1$ and $Z^{1a}_J \in J_s$; corresponding rate 4

- $\langle Compute, \langle Z^{1a}_P, Z^{1a}_J \rangle\rangle + \langle Compute, \langle Z^{1a}_P, Z^{1b}_J \rangle\rangle + \langle Compute, \langle Z^{1b}_P, Z^{1a}_J \rangle\rangle + \langle Compute, \langle Z^{1b}_P, Z^{1b}_J \rangle\rangle$ with $|Z^{1a}_P| = 1$ and $Z^{1a}_P \in P_s$, $|Z^{1b}_P| = 1$ and $Z^{1a}_P \in P_f$, $|Z^{1a}_J| = 1$ and $Z^{1a}_J \in J_s$, $|Z^{1b}_J| = 1$ and $Z^{1b}_J \in J_l$; corresponding rate 3.3

- $\langle Compute, \langle Z^{1a}_P, Z^{1a}_J \rangle\rangle + \langle Compute, \langle Z^{1b}_P, Z^{1a}_J \rangle\rangle$ with $|Z^{1a}_P| = 1$ and $Z^{1a}_P \in P_s$, $|Z^{1b}_P| = 1$ and $Z^{1a}_P \in P_f$, $|Z^{1a}_J| = 2$ and $Z^{1b}_J \in J_s$; corresponding rate 2.6

There is a computationally less expensive but not always accurate method for computing lower/upper bounds for the elements of $\mathbf{Q}^-_a$ and $\mathbf{Q}^+_a$. Given a transition t and a tuple d of aggregate static subclasses, compute the Cartesian product D of the sets of original subclasses in the component aggregate subclasses. Associate with d the $min(max)_{d' \in D} \mathbf{w}[t](d')$. In this way the computed

$\mathbf{Q}_a^-$ and $\mathbf{Q}_a^+$ are bounds for $\mathbf{Q}_a$, but they can be very inaccurate in some cases. In the previous example this simplified method would have predicted correctly the maximum rate (= 4), while the minimum (= 0.6) would have been less than the the correct one (0.9).

Observe that even if the computation of the aggregate transition rates has a certain cost, it could be performed once and for all in a way that the result is easily reusable for models differing only in the actual parameter values.

Complexity of bounds computation The main goal of the proposed method is to cope with the problem of state space explosion by allowing some aggregation of the state space. However since the aggregate chain is only a bound for the actual aggregate chain and since the computation of bounds of a chain given its lower (and/or upper) bound transition probability matrix is more expensive than just computing the steady state solution of a chain of the same dimension, in some cases the method proposed could be computationally more expensive than the solution of the whole model we started with.

Let's consider the queueing network example we have been using throughout the paper. The complete model has 50 states while the aggregate model has 18 states. If we assume that the computation of the steady state solution of a MC of dimension n is $O(n^2)$ (we are assuming that sparse matrix algorithms can be applied, if this is not the case, the worst case complexity would be $O(n^3)$) then the time required for solving the complete model is proportional to $50^2 = 2500$, while the time required for getting bounds from the aggregate model is proportional to $17 * (18^2) = 5508$. Apparently the bounding method for this case is not computationally convenient, however the situation changes completely when the dimension of the studied system grows. The size of this model was chosen small so that the reader could try to build both the complete and aggregate transition matrices. Similar models with different number of customers, servers in the multiple server station and static subclasses in the class of servers, show how convenient the method can be as the size of the problem increases. This is shown in Table 14.3 where the number of states in the lumped (column a) and complete (column b) Markov chain and the corresponding estimated computational time cost as function of the number of servers and customers are listed. Notice that the number of states in the complete model refers to the case in which each queue in the multiserver has a different speed.

In general the complexity for computing bounds is $k(n')^2$ where k is the number of aggregate columns modified in the original MC to make it lumpable, hence in the worst case $k = n'$. In order for the method to be computationally convenient the ratio n/n' must be such that $k(n')^2 < n^2$.

Finally observe that even when the ratio between the time complexity of exact solution computation versus that of bounds computation is less than one, the computation of bounds can still be convenient. In fact the computation of bounds on the lumped model can be easily parallelized because it requires the solution of several smaller and completely independent (lumped) models, thus

Serv.	Aggr. States	Non Aggr. States	(a) Bounds Comp.Time	(b) Exact Sol. Comp.Time	(b)/(a)
5 customers					
4	18	126	5832	15876	2.72
5	19	252	6859	63504	9.26
6	19	462	6859	213444	31.12
8 customers					
4	53	495	148877	245025	1.64
5	60	1287	216000	1656369	7.6
6	64	1716	262144	2944656	11.23
10 customers					
4	94	1001	830584	1002001	1.21
5	113	3003	1442897	9018009	6.25
6	125	8008	1953125	64128064	32.83

Table 14.3: Comparing time complexity for the computation of bounds vs. exact solution.

resulting in a gain in time complexity. Breaking up the problem into smaller ones may be also crucial from the space complexity point of view since the whole model could not fit into the available main memory, preventing the solution if no virtual memory is provided or causing a loss of performance due to frequent paging if virtual memory is available.

14.4 Improving bounds

The bounds found with the method described above may have a different degree of tightness depending on the sensitivity of the solution to the rates replaced by lower bounds. In this section a technique is presented that can be used to improve the bounds when those obtained by directly applying the method presented in Section 14.2 are too loose. In the previous section we mentioned that it is possible to derive an upper bound $\mathbf{Q}_a^+$ for the rates between aggregates; once we have this upper bound it is possible to use Theorem 14.5 to compute another set of bounds that, combined with those obtained from the lower bound $\mathbf{Q}_a^-$, may lead to tighter bounds. The problem is that in this case the inverse $(I - \mathbf{Q}^+)^{-1}$ is not guaranteed to exist, and even if some further condition must be verified in order to be sure that the results obtained are indeed bounds. In the next paragraph we will see how the upper bound on the lumped infinitesimal generator can be used in conjunction with the the steady state probability bounds $\boldsymbol{\pi}^-, \boldsymbol{\pi}^+$ previously computed to get better bounds on any Markov reward function (and hence also on the state probabilities).

Exploiting information about the "rates redistribution" vector x The following method is aimed at improving the bounds on the performance measures we are interested in; we assume that each performance metric is defined

State #	1	2	3	4	5	6
x^+	0.350781	0.840613	0.609799	0.258352	0.07127	0.434729
State #	7	8	9	10	11	12
x^+	0.0862158	0.0042123	0.100793	0.0733268	0.160317	0.048524
State #	13	14	15	16	17	18
x^+	0.000000	0.0162282	0.0163021	0.0400884	0.0421257	0.0135091

Figure 14.8: Upper bound for vector x.

	Lower bound	Exact value	Upper bound	Spread %
Mean Queue Length	1.250705	1.273430	1.295075	3.5%
Throughput	0.980303	0.98284	0.983502	0.3%

Table 14.4: Improved bounds.

by associating a reward function r on states with the model. In particular a possible performance metric could be the steady state probability of one specific state s_j; in this case all the states are assigned a null reward except s_j which is assigned a reward $r_j = 1$.

If the exact vector $\mathbf{x}$ of *rates redistribution* was known and substituted into matrix $\mathbf{Q}_a^s$, then the mean cumulative reward (conditioned on being in the first n' states) computed from $\mathbf{Q}_a^s$ would be the correct value we are looking for.

The behaviour of the Markov chain described by $\mathbf{Q}_a^s$ is the following: the evolution starts from one state in $\mathcal{S}$ and goes on until it reaches the extra state s, then the evolution starts again from a state in $\mathcal{S}$ probabilistically chosen according to the probability distribution defined by $\mathbf{x}$ until it reaches s again. The accumulated reward between two successive visits to s depends on which state is chosen to start with when exiting s. Assume we have the list $s_{j1}, s_{j2}, \ldots, s_{jn'}$ of states ordered in such a way that the accumulated reward before exiting to s, given that we started in $s_{jm}, m = 1, \ldots, n'$, is greater than or equal to the accumulated reward before exiting to s, given that we started in $s_{jk}, k > m$.

An upper bound on the mean cumulative reward can be computed from model $\mathbf{Q}_a^s$ by substituting for $\mathbf{x}$ a modified vector $\mathbf{x}'$ such that $\mathbf{x}'_{jm} \geq \mathbf{x}_{jm}$ $(1 \leq m \leq k \leq n')$ and $\mathbf{x}'_{jl} \leq \mathbf{x}_{jl}$ $(k < l \leq n')$.

A lower bound on the mean cumulative reward can be computed if in model $\mathbf{Q}_a^s$ we substitute for $\mathbf{x}$ a modified vector $\mathbf{x}'$ such that $\mathbf{x}'_{jm} \leq \mathbf{x}_{jm}$ $(1 \leq m \leq k \leq n')$ and $\mathbf{x}'_{jl} \geq \mathbf{x}_{jl}$ $(k < l \leq n')$.

If we were able to estimate an upper bound for $\mathbf{x}$ this could thus be used to compute better bounds on the desired performance figures. See [4] for the details on how to derive an upper bound x^+ for x using $\mathbf{Q}_a^+$, $\mathbf{Q}_a^-$, $\boldsymbol{\pi}^+$ and $\boldsymbol{\pi}^-$.

Applying the above method to the queueing network example we obtain for the vector $\mathbf{x}$ the upper bound estimate in Fig. 14.8. The new (much improved) bounds computed for the mean queue length and throughput using this method are shown in Table 14.4.

14.5 A more complex example

In this section we present a more complex example in which the presence of synchronization constraints does not allow us to easily obtain bounds on the performance figures of interest using some other method.

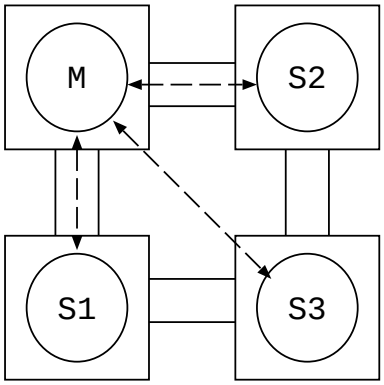


Figure 14.9: A parallel program model.

The modeled system consists of a parallel program organized according to a master-slave computation paradigm. The program is mapped on a parallel architecture: we assume there is a processor for each process, that the processors are homogeneous, and that due to the type of the interconnection network the communication time between the master and the slaves is not homogeneous (some slaves are "closer" to the master than others). This is the case for the example of a master-slave program mapped onto a mesh architecture depicted in Fig. 14.9. The processors are represented by squares, while the processes are represented by circles. In the figure both the physical channels (connecting the processors) and the logical channels (connecting the processes) are depicted.

The master divides the problem to be solved into several subproblems, and as long as there are free slaves it delivers the tasks to be executed to them. When no more free slaves are available, the master waits for some slave to complete its current task. The master also performs some computation in parallel with communicating, to process the results received and to prepare new data to be distributed.

We have modeled the master-slave system using the SWN formalism; in Fig. 14.10 the resulting model is depicted. In this model the presence of a token in one place indicates the program counter position for the corresponding process, so that the next statement to be executed is represented by the transition pointed by the arc exiting the marked place.

The model is composed of two subnets, the first (leftmost) one represents the master's behaviour while the second (rightmost) one represent the common behaviour of the slaves. The distinction among the three slaves is obtained by using different basic colors. The behavioural model has been simplified as much

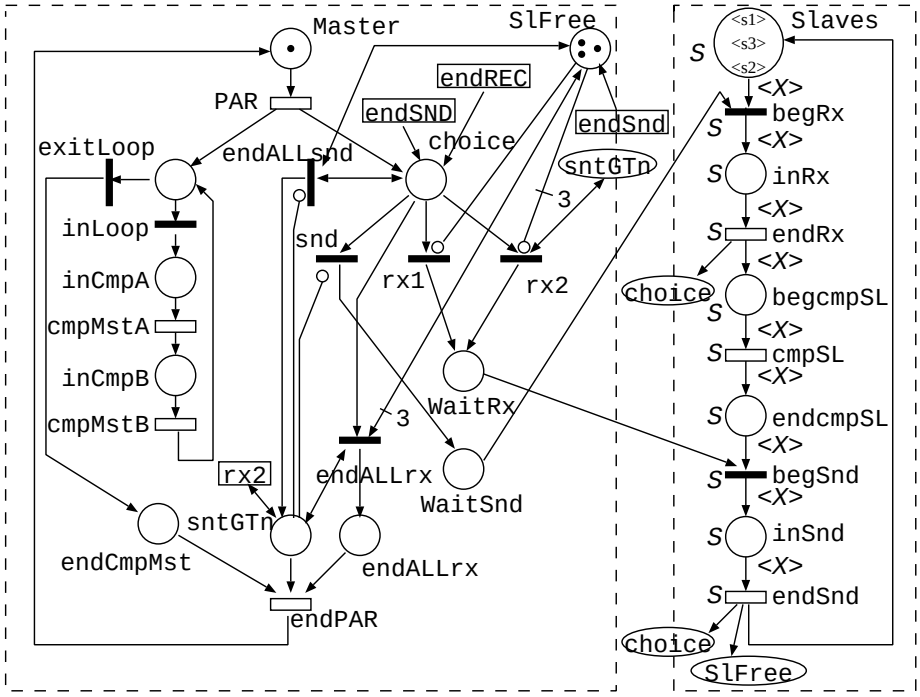


Figure 14.10: The SWN model of the Master-Slave Program.

as possible to make the picture readable. The master is composed of two parts that work in parallel: the computation part, consisting of a certain number of iterations of two procedures (transitions *inLoop*, *cmpMstA*, *cmpMstB*), and the synchronous communication part consisting of a certain number of iterations of send/receive operations (transitions *snd*, *rx1*, *rx2* plus the transitions representing actual communication, shared with the slaves net: *begRx* , *endRx* , *begSnd*, *endSnd*). The slaves behaviour can be described as repeated iteration of three operations: receive (transitions *begRx* and *endRx*), compute (transition *compSL*), send (transitions *begSnd* and *endSnd*). Only one color class is needed: the slaves class S. It has cardinality three and is divided into two static subclasses $S1$ of cardinality 2 and $S2$ of cardinality 1. Subclass $S1$ represents the slaves that are closer to the master while subclass $S2$ represents the farther slave. As a consequence the transitions representing communication between the master and the slaves have a rate that depends on the slave identity.

The detailed state representation for this model is a list of program counter values, one for each process in the program. The MC representing this model behaviour is quasi-lumpable with respect to the aggregation of states induced by considering the slaves as undistinguishable. In other words we aggregate all the states that are equal up to a permutation of slaves identities. For example we aggregate the state M_1 with slave s_1 waiting to receive some data from the master, s_2 sending a result to the master and s_3 performing a computation, and state M_2 with slave s_2 waiting to receive some data from the master, s_3 sending a result to the master and s_1 performing a computation. Observe that although this aggregation reflects a symmetry in the state space structure due to the homogeneous behaviour of the slaves, there is a perturbation in the transition rates, due to the fact that the communication between s_3 and the master takes longer than the communication between the other two slaves and the master. The quasi-lumpable model has 109 states and the number of aggregate states is 49.

We have considered three performance measures: *throughput of the system*, i.e. the number of completed computations for time unit which corresponds to the throughput of transition *PAR* representing the beginning of a new computation; the acronym used in the table is THRUPUT; *mean number of slaves waiting to receive a task from the master*, i.e., mean number of tokens in place *Slaves*; the acronym used in the table is MNSWRx; *mean number of slaves waiting to send the result of their computation to the master*, i.e., mean number of tokens in place *endcmpSL*; the acronym used in the table is MNSWSnd.

We have performed three experiments keeping the rates $\mathbf{w}[endRx](S_3)$ and $\mathbf{w}[endSnd](S_3)$ fixed while varying the rates $\mathbf{w}[endRx](S_i), i = 1, 2$ and $\mathbf{w}[endSnd](S_i), i = 1, 2$, as shown in Table 14.5, to study the effect of varying the difference in the communication rates between the master and the slaves in the two static subclasses. The results of each experiment are reported in Table 14.6.

These results are obtained from the direct method. We have also applied the bounds improvement method: it resulted in tightened bounds in the first experiment but it did not give any substantial improvement in experiments 2

Exp. num.	$\mathbf{w}[endRx]$		$\mathbf{w}[endSnd]$	
	$S_{1,2}$	S_3	$S_{1,2}$	S_3
1	0.51	0.5	3.57	3.5
2	0.525	0.5	3.675	3.5
3	0.55	0.5	3.85	3.5

Table 14.5: Rates definition for the three bounds computation experiments.

Exp.num.	Perf. Measure	Lower bound	Upper bound	Spread %
1	THRUPUT	0.012396	0.014595	16.29%
	MNSWSnd	0.079165	0.137257	53.68%
	MNSWRx	2.547911	2.726692	6.78%
2	THRUPUT	0.011936	0.016718	33.38%
	MNSWSnd	0.056813	0.180464	104.23%
	MNSWRx	2.436226	2.802309	13.98%
3	THRUPUT	0.011333	0.019553	53.23%
	MNSWSnd	0.039295	0.257986	147.13%
	MNSWRx	2.249367	2.861513	23.95%

Table 14.6: Bounds for the parallel program model.

and 3. As a matter of fact, the first case is also the only one that is eligible for the application of Theorem 14.5 of Section 14.2.

Clearly there is a strict correlation between the difference $\mathbf{Q}^+ - \mathbf{Q}^-$ and the spread in the bounds. It might be worthwhile to assess a priori the sensitivity of the system on variations on the transition probabilities that are positive in $\mathbf{Q}^+ - \mathbf{Q}^-$.

In this example we have also observed that some further bounds refinement could be obtained by exploiting some further property of the high level model. If no contention for resources were present in the slaves subnet, then we would know for sure that the throughput of slave s_3 is lower than the throughput of slaves s_1 and s_2. Since the slaves compete to communicate with the master, the throughput of both kind of slaves decreases with respect to the no contention case; moreover the difference between the throughput of the slave s_3 and the other two slaves becomes smaller as the probability of contention increases. Our conjecture is that in the system we are studying, the throughput of slave s_3 is less than or equal to the throughput of the other two slaves. Assuming that the conjecture is true (and it is indeed true for our experiment cases), we show how this information can be used to improve the estimate of the lower bound matrix $\mathbf{Q}_a^-$. Of course, the kind of property to be proved about the high level model in order to obtain improved estimates of $\mathbf{Q}_a^-$ and/or $\mathbf{Q}_a^+$ and the technique used to prove it depend heavily on the kind of high level formalism adopted and in general cannot be automated.

Notice that in our example, every aggregate transition submatrix $\mathbf{Q}_{I,J}$ with different row-sums, refers to a communication transition (let's call this transition

t_{comm}). Since only one slave at a time can communicate, each quasi-lumpable communication transition aggregate contains two kinds of ordinary transitions: one for the slave s_3 and the other one for the slaves s_1 and s_2. Thus a lower bound for $\mathbf{Q}_{a_{I,J}}$ is given by:
Pr^+(comm. by the slave in $S2$) $\mathbf{w}[t_{comm}](S2)$ + Pr^-(comm. by a slave in $S1$) $\mathbf{w}[t_{comm}(S1)$
with the constraint Pr^+(comm. by the slave in $S2$) + Pr^-(comm. by a slave in $S1$) = 1.

The probability p that place $inREC$ contains a token representing slave $S3$ given that $inREC$ is not empty can be formulated as

$$p = \frac{\frac{throughput(endRx,S2)}{\mathbf{w}[endRx](S2)}}{\frac{throughput(endRx,S1)}{\mathbf{w}[endRx](S1)} + \frac{throughput(endRx,S2)}{\mathbf{w}[endRx](S2)}}$$

An upper bound for this probability can be found by considering the throughput of the slave in $S2$ to be equal to the throughput of both slaves in $S1$ (let's call the common throughput value t):

$$p^+ = \frac{\frac{t}{\mathbf{w}[endRx](S2)}}{\frac{2\ t}{\mathbf{w}[endRx](S1)} + \frac{t}{\mathbf{w}[endRx](S2)}}$$

and after a simplification

$$p^+ = \frac{\mathbf{w}[endRx](S1)}{2\ \mathbf{w}[endRx](S2)\ +\ \mathbf{w}[endRx](S1)}$$

A similar argument applies to place $inSnd$.

Thus a lower bound for the rate of an aggregate transition representing a communication is obtained by weighting by p^+ the row-sum corresponding to a communication transition involving the slave in $S2$, and weighting by $1 - p^+$ the remaining communication transitions associated with the slaves in $S1$.

Perf. Measure	Lower bound	Upper bound	Spread %
THRUPUT	0.012154	0.015649	25.14%
MNSWSnd	0.069087	0.161423	80.11%
MNSWRx	2.507557	2.773172	10.06%

Table 14.7: Improved bounds for the parallel program model.

.

The results obtained by applying this technique to the model of experiment 3 are reported in Table 14.7. The bounds are definitely tighter than those obtained in experiment three ignoring the model semantics. This indicates that whenever some additional information about the system behaviour can be derived from the high level model, it should be exploited to get improved bounds.

14.6 Conclusions and Bibliographical Remarks

In this chapter it has been presented a method for the computation of bounds of quasi-lumpable SWN models that can be considered as an evolution of the lumping method presented in Chapter 13. The difference with respect to other chapters on approximations and bounds that appear later in this book (Part 5 and Part 6), is that in our method the net structure is used to generate a smaller Markov chain from which bounds can be computed, whereas in later chapters, methods will be presented that show how to compute bounds working at the net level without passing through the underlying (complete or aggregate) Markov chain.

The results presented in this chapter have been first published in [4, 5] and are based on the work of Courtois and Semal reported in [2, 3]. Other examples of application of the Courtois and Semal's method can be found in [8, 7, 6].

Bibliography

[1] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11), November 1993.

[2] P.J. Courtois and P. Semal. Bounds for positive eigenvectors of nonnegative matrices and their approximations. *Journal of the ACM*, 31(4):804–825, Oct. 1984.

[3] P.J. Courtois and P. Semal. Computable bounds on conditional steady-state probabilities in large markov chains and queueing models. *IEEE Journal on Selected Areas in Communication*, SAC-4(6):926–937, Sept. 1986.

[4] G. Franceschinis and R.R. Muntz. Bounds for quasi-lumpable markov chains. *Performance Evaluation*, 20(1), May 1994. (Performance '93. 16th IFIP Working Group 7.3 International Symposium on Computer Performance Modeling, Measurement and Evaluation, Rome, Italy, 27 Sept.-1 Oct. 1993).

[5] G. Franceschinis and R.R. Muntz. Computing bounds for the performance indices of quasi-lumpable stochastic well-formed nets. *IEEE Transactions on Software Engineering*, 20(7), July 1994.

[6] L. Golubchik and J. C.-S. Lui. Bounding of performance measures for a threshold-based queueing system with histeresis. In *Proc. of 1997 ACM Sigmetrics Conf.*, Seattle, WA, June 1997.

[7] R.R. Muntz and E. de Souza e Silva. *Computational Solution Methods for Markov Chains: Applications to Computer and Communication Systems.* 1992. Lectures notes.

[8] R.R. Muntz, E. de Souza e Silva, and A. Goyal. Bounding availability of repairable computer systems. *IEEE Trans. on Computers*, 38(12):1714–1723, 1989.

[9] V. F. Nicola. Lumping in markov reward processes. In *Proc.* 1^{st} *International Conference on Numerical Solution of Markov Chains.* Marcel Dekker, Inc., January 1990.

Chapter 15

Combining decomposition and symmetry

15.1 Introduction

In this chapter we present how to combine the aggregation (or lumping) method with the tensor decomposition methods to compute performance indices of systems modelled with SPNs.

The tensor decomposition methods exploit the net structure to deduce a tensor expression of the generator of the underlying Markov chain of the Stochastic Petri Net (SPN) system, involving matrices computed at the subnets level. The net is expressed either as a synchronous composition of subnets sharing common transitions in Superposed Generalized Stochastic Petri Nets (SGSPN) introduced by S. Donatelli (see chapter 12) or as an asynchronous composition of subnets sharing common places, with transitions firing tokens from one subnet into other subnets in the works of P. Buchholz [4, 3]. The main interest of such methods is to deal with much smaller matrices (as a general rule), to compute steady state probabilities exploiting tensor properties, hence reducing memory requirements and computation time.

On the other hand, the aggregation methods are based on a partition of the state space allowing the definition of a Markov chain on the subsets of states of this partition. These subsets may be viewed as "macro-states" of the system behaviour. The clear benefits of these methods is the reduction of the size of the state space of the Markov chain to be solved. The Stochastic Well formed Petri Net model (SWN) is a high level net formalism (see chapters 7 and 13) which derives automatically such an aggregation from the net definition without computing the reachability graph (RG) of the net system. Moreover, macro-states have an easy interpretation in terms of system behaviour.

As a further step, to cope with the state space explosion problem, one may hope to take advantage of both kinds of methods by combining decomposition

and aggregation. Clearly, such an approach may only be devised for systems with entities which can be grouped in classes with identical (stochastic) behaviour, since, otherwise, no aggregation may take place. In the same way, these classes need to exhibit some kind of synchronization between them to introduce a decomposition in sub-models. These two requirements must be checked during the specification of the system.

So, let us assume that we have identified such entities in the system to be modelled. A straightforward merging of the decomposition and aggregation methods would be to extend results about SPN tensor decomposition, "replacing" standard sub-GSPNs with sub-SWN to introduce an aggregation deduced from the net definition. As a general rule however, this approach cannot be used: the synchronized activities introduce *dependencies between entities which must be memorized* and a simple (coloured) transition firing does not allow such a memorization. We can say that *decomposition and aggregation are not "orthogonal" methods*. To memorize these dependencies it is necessary to include in the model of each subsystem a part of the subsystems with which it is synchronized. Under appropriate conditions, it is shown that tuples of aggregates (standard symbolic markings) of these extended submodels are aggregates of the states of the whole model.

The combined approach is summarized in Fig. 15.1 together with its context. From a net $\mathcal{N}$, (synchronous or asynchronous) composition of subnets $\mathcal{N}_k$, the basic method (left branch of the diagram) computes the RG of the net system ("G" -graph building- arcs) and deduces the generator $\mathbf{Q}$ of the CTMC ("M" -Markov chain- arcs) from it.

The aggregation method (SRG branch) used by the SWN model defines an exact lumping of the original CTMC with generator $\widehat{\mathbf{Q}}$.

The tensor decomposition method (right hand side, RG_k branch) provides a tensor expression $\mathbf{Q}'$ which is a "super-matrix" of $\mathbf{Q}$ (notice that for the asynchronous case, it is first necessary to define the environment of a subnet to be able to talk about composition of subnets).

The validity ("C" -consistency- arcs) of the expression of these new generators has been shown.

For the combined aggregation/decomposition method (central branch of Fig. 15.1), three points must be addressed:

1. how to build *enlarged* SWN (denoted by $\overline{\mathcal{N}_k}$) of the subsystems to keep track of colour synchronization,
2. deduce an expression of the generator $\overline{\mathbf{Q}}'$ of the CTMC underlying the composition of the $(\overline{SRG}_k)_{k=1,\ldots,K}$ of the $(\overline{\mathcal{S}}_k)$ similar to tensor expressions for GSPNs,
3. prove that the given tensor expression is a supermatrix of a lumping of $\mathbf{Q}$.

In this context, we define classes of asynchronous and synchronous composition of models for which we are able to give explicit methods for extended subnets building (1).

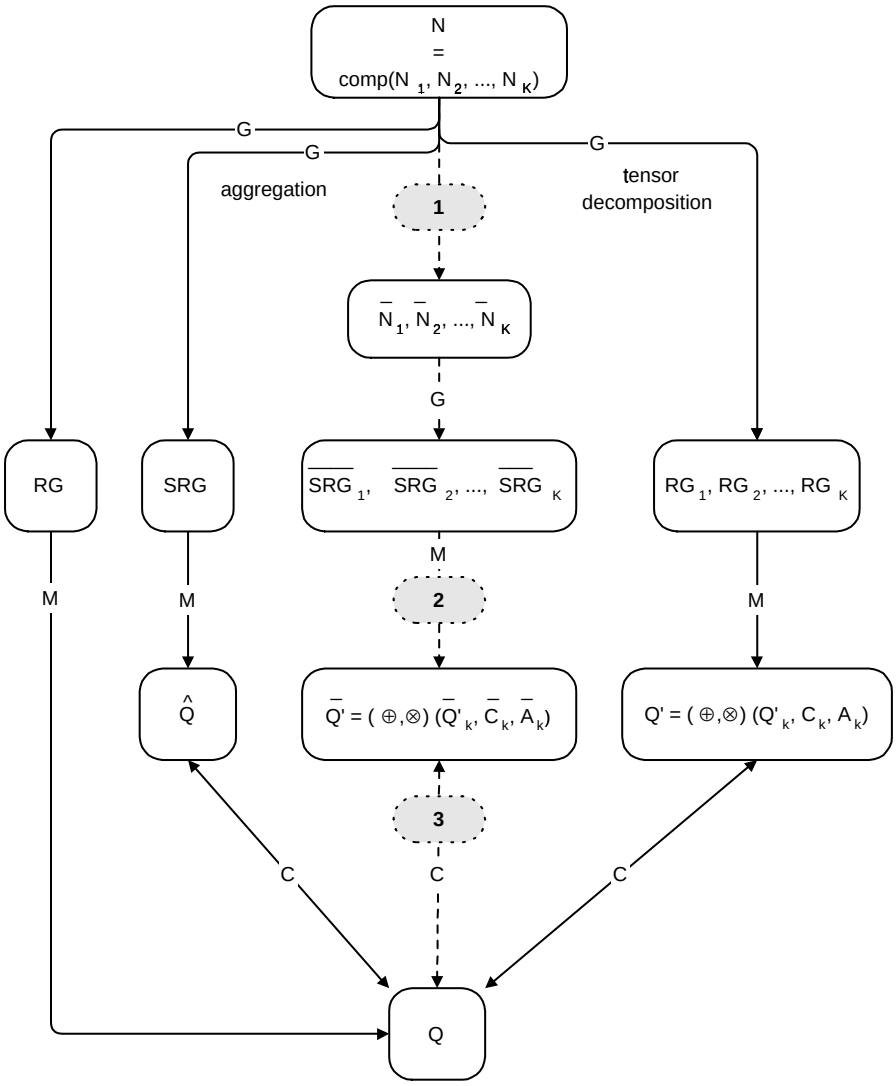


Figure 15.1: Proposed approach and its context

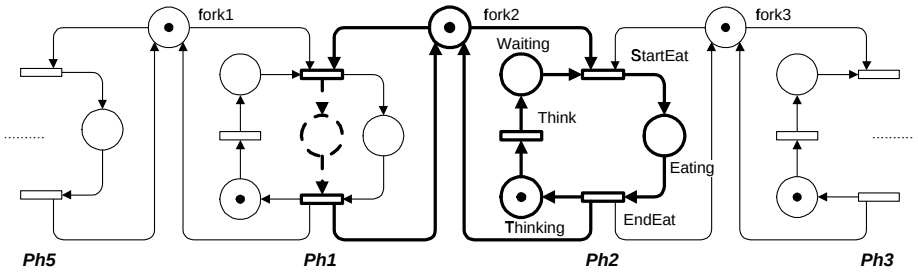


Figure 15.2: GSPN of the Philosopher's Dinner

We also deduce an expression of the generator $\overline{\mathbf{Q}}'$ in each case (2) and we prove the consistency of the given tensor expressions (3).

The rest of the chapter is organized as follows: in Sect. 15.2 we highlight the main problems to be solved and we present, in Sect. 15.3, the general principles of the method for both asynchronous and synchronous composition of SWN. In Sect. 15.4 we detail the synchronous composition case, and the asynchronous case in Sect. 15.5. For the sake of simplicity, all transitions of the SWN are timed (with exponential firing time) in the above sections. In the concluding section, we discuss the applicability of the method and present extensions to more general SWN.

15.2 Main problems

As mentioned above, unless for very particular synchronous cases, no direct merging of the aggregation and tensor decomposition methods may be devised. Two basic problems must be taken into account: a *specification problem*, that is to recognize the *kind of entities* which are synchronized (different/same type), and a *resolution problem* that is to be able to deduce the global CTMC from the smaller ones computed from the synchronized subnets.

15.2.1 External/Internal synchronization

The first problem with our approach relates to the kind of synchronization found at the system specification level: if several objects of the same "type" (processes for instance) are synchronized together, then we may build a model with a product structure , each of the terms of this synchronized product being a model of one object behaviour, leading to a product of synchronized CTMCs. We refer to this situation as *internal synchronization*.

Alternatively, we may also build SWN models of such systems, and we get an aggregated CTMC of the whole model.

A very simple example of such a system (for synchronous composition) is the well-known philosopher's dinner problem, the GSPN of which (with 5 philo-

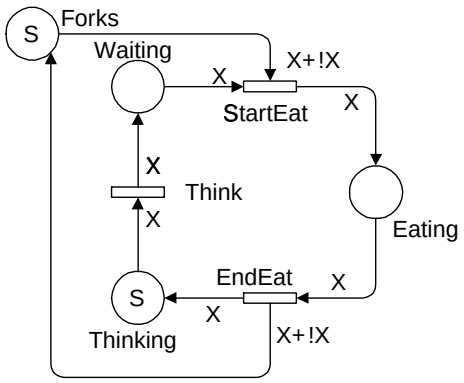


Figure 15.3: SWN of the Philosopher's Dinner

sophers) is given in Fig. 15.2 and the SWN in Fig. 15.3. As we see, the SWN is the "folding" of the GSPN and the synchronization between philosophers is embedded in it; on the other side, we could try to decompose the GSPN as a two by two synchronization between five identical GSPNs (one copy of which is drawn in bold in Fig. 15.2).

A more effective example of internal synchronization is the study of the so called Lamport's fast mutual exclusion algorithm, using the Stochastic Automata Network (SAN) model in [13] and the SWN model (although not fully exploiting the SWN power for performance evaluation) in [1].

So, in such situations, we have to choose between two strategies for the specification of the system: factorization by classes of entities with same behaviour leading to one SWN of the whole system, or composition of entity behaviours, using a GSPN or SAN decomposition.

In systems with synchronization of objects of different classes, we say that the system exhibits *external synchronization*. Following the tensor decomposition method presented in chapter 12, we want to build a synchronized product of subnets such that each subnet models the behaviour of one or more object classes. In this way, we could use aggregation at the subnets level using the SWN model and at the same time, a composition of the CTMCs underlying these SWN at the global level.

15.2.2 Synchronization memory

Given a system with external synchronization, we can model it as a SWN $\mathcal{N}$, *synchronous or asynchronous composition* of SWN $(\mathcal{N}_k)_{k\in K}$[1], each $\mathcal{N}_k$ being a SWN model. Unfortunately, as we are going to show now, a direct extension

[1] from now on, we shall use for ease of writing, K as set of k indexes or as maximal k index when no confusion can arise.

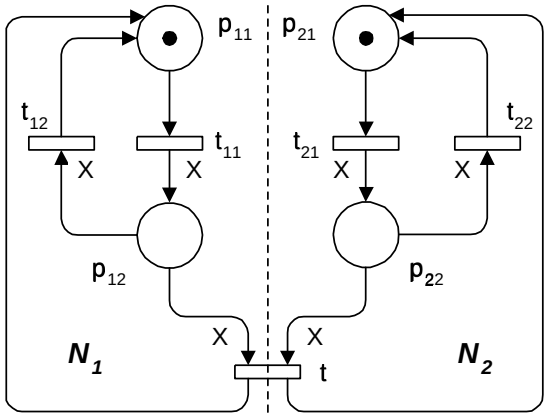


Figure 15.4: Synchronized SWN with memory

of the GSPNs composition cannot be used *to solve* the initial CTMC because composition (i.e. sum of graphs[2]) of aggregates provided by the SRGs of the $\mathcal{S}_k$ *does not provide an aggregation of the whole CTMC of the model* verifying the strong lumpability condition (see chapter 13).

Let us give an elementary example of this problem with a system formed of two subsystems: in each subsystem, the activity begins by the choice of the kind of job to perform; then the job may by autonomously finished, or, for the same kind of jobs in the two subsystems, by the two subsystems together. The SWN of Fig. 15.4 is a model of the system. $\mathcal{S}$ is the composition of two SWN $\mathcal{S}_1$ and $\mathcal{S}_2$ through t: jobs are finished either autonomously in $\mathcal{S}_1$ via t_{12} (resp. $\mathcal{S}_2$ via t_{22}) or, for the same kind of jobs (translated by the same input function: $\mathbf{Pre}(p_{12}, t) = \mathbf{Pre}(p_{22}, t) = X$) in both nets via t.

The colour domain of p_{12} and p_{22} is a given class C and the colour domain of p_{11} and p_{21} is the neutral colour; all transitions fire for a colour c from C. We give (without formal notation) in Fig. 15.5 the SRG of $\mathcal{S}$ and in Fig. 15.6 the SRGs of $\mathcal{S}_1$, $\mathcal{S}_2$ and their "synchronized product".

The independent works in the subsystems are translated in Fig. 15.5 by firing sequences (t_{11}, t_{12}) and (t_{21}, t_{22}). We also observe that the common termination of jobs (firing of t) is enabled in *only one symbolic marking (4)* which represents states with *same* kind of jobs in the subsystems (places p_{12} and p_{22}), whereas marking (5) represents states with *different* kind of jobs in each subsystem.

On the contrary, in Fig. 15.6, we have only one "symbolic marking" (4') with tokens in p_{12} and p_{22}: the relative identity of the jobs (defined by the firings of t_{11} and t_{21}) is lost and (4') is the merging of (4) and (5) which is an incorrect

[2]the sum of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is the graph $G = G_1 + G_2 = (V_1 \times V_2, E)$ with $((u_1, u_2), (v_1, v_2)) \in E$ iff $(((u_1, v_1) \in E_1)$ and $((u_2 = v_2)$ or $((u_2, v_2) \in E_2))$ or $(((u_2, v_2) \in E_2)$ and $((u_1 = v_1)$ or $((u_1, v_1) \in E_1))$

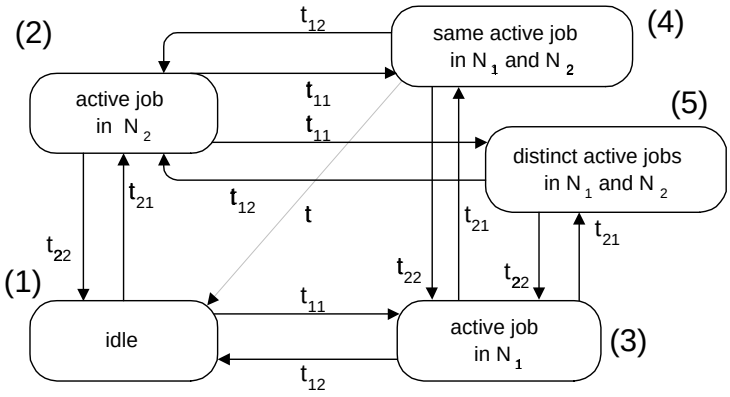


Figure 15.5: Synchronized SWN with memory: SRG of $\mathcal{S}$

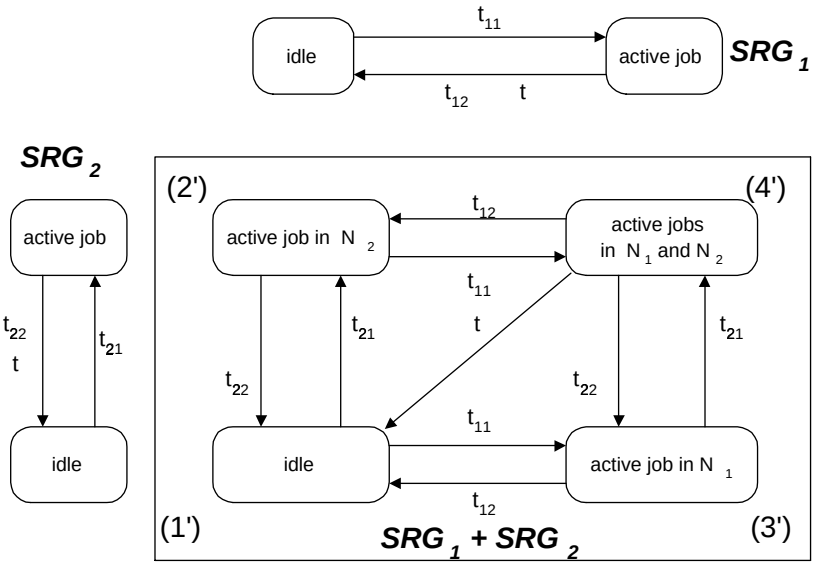


Figure 15.6: Synchronized SWN with memory: SRGs of $\mathcal{S}_1$, $\mathcal{S}_2$ and their "synchronized product"

lumping since (4) enables t whereas (5) does not.

This example points out the main reason for which the direct composition of sub-SWN is disallowed: the firing of a synchronization transition may introduce changes in the markings forbidding some admissible permutations. In short, we can say that the (stochastic) transition system must "memorize" these firings, and direct composition of sub-SWN does not allow this memorization. We call such synchronization transitions *memory synchronization transitions*.

Let us emphasize that this memory problem in synchronization is a general one encountered in any model (SWN, SAN, GSPN, ...): in Plateau's work [12] for instance, auxiliary automata are introduced specifically to memorize specific states of the system. However we aim to find methods which do not compel us to modify the original model.

In the present method, the synchronization memory is embedded in a syntactic way by *enlarging* each subnet $\mathcal{N}_k$ with sufficient information, leading to new subnets $\overline{\mathcal{N}}_k$.

15.3 General features of the method

In this section, we give an overview of the combined aggregation/decomposition method. The framework is a SWN $\mathcal{N}$ composed of subnets $(\mathcal{N}_k)_{k\in K}$, in a synchronous manner through common transitions, or in an asynchronous manner through "output" transitions firing tokens between subnets. In both cases, we denote by TX the set of these synchronization transitions. Although the synchronous and asynchronous compositions need specific treatments, they share a common approach.

We provide efficient criteria to check the applicability of the method by the way of *syntactic conditions of aggregation* on the net, that is to say either syntactic constraint (for instance arc functions) or syntactic properties (for instance symbolic flows). These sufficient conditions allow to verify, without computing the RGs of the whole net or of its subnets, if a combined aggregation/decomposition may be used.

15.3.1 Enlarging the subnets

For the asynchronous case, we have firstly to define the environment of each subnet to be composed. We give below a formal definition of this environment based upon the *abstract views* of the other subnets. The enlargement of a subnet is then built adding these abstract views to it. In the synchronous case, only pairwise synchronization will be allowed and the memorization of the colours synchronization is obtained through an enlargement of one of the subnets. These extensions are formally defined in each case (see below). In both cases, we obtain a collection of extended subnets $(\overline{\mathcal{N}}_k)_{k\in K}$.

15.3.2 The general algorithm

The first step is to build the SRGs (resp. the SRSs) of the extended subnets $\overline{\mathcal{S}}_k$, denoted by $\overline{SRG}_k$ (resp. $\overline{SRS}_k$). A symbolic marking of $\overline{SRS}_k$ is denoted by $\widehat{\overline{\mathbf{m}}}_k$. The cartesian product of the $(\overline{SRS}_k)$ will be the new aggregated state space. It is denoted by $\overline{XSRS}$ and its elements are denoted by $\overline{\mathcal{M}}$ (hence $\overline{\mathcal{M}} = (\widehat{\overline{\mathbf{m}}}_k)_{k=1,\ldots,K}$).

The basis of the algorithm is a combination of the tensor expression of the generator $\mathbf{Q}$ of the CTMC of the synchronized product of the $\overline{SRG}_k$ and of the regular computation for SWN. We show that the matrix $\mathbf{Q}$ of the CTMC can be written as sub-matrix of

$$\mathbf{Q}' = \bigoplus_{k=1}^{K} \mathbf{Q}'_k + \sum_{t \in TX} \sum_{d} \theta(t,d) \left[\bigotimes_{k=1}^{K} \mathbf{C}_k(t,d) - \bigotimes_{k=1}^{K} \mathbf{A}_k(t,d) \right] \tag{15.1}$$

where $\theta(t,d)$ is the rate of the transition t and d is a choice of static subclasses for the symbolic firings of t. The $\mathbf{Q}'_k$ matrices come from the generator of the CTMC of the $\overline{\mathcal{S}}_k$ nets using only local transitions: they can be built from classical SWN technics, discarding any synchronization transition effect. The $\bigoplus$ operator means that these CTMCs are independent stochastic processes. The $\mathbf{A}_k$ and $\mathbf{C}_k$ matrices are obtained as consequence of synchronization transition firings: any such firing produces a state change in each component $\overline{\mathcal{S}}_k$ involved in the marking change.

By sub-matrix we mean that non zero terms (for same pair of states) of $\mathbf{Q}$ equal those of $\mathbf{Q}'$ and that if $\overline{\mathcal{M}}$ is reachable and $\overline{\mathcal{M}}'$ is unreachable then $q'_{\overline{\mathcal{M}},\overline{\mathcal{M}}'} = 0$.

The sketch of the computation of a performance measure is given by the algorithm 15.1.

Let us emphasize that such computations never use $\mathbf{Q}'$ directly but instead the $\mathbf{Q}'_k$, $\mathbf{C}_k$ and $\mathbf{A}_k$ matrices.

As for GSPNs (see chapter 12), to use the tensor expression of $\mathbf{Q}'$, the numerical method computing a given measure has to verify the conditions:

- only linear functions of products $\mathbf{V}.\mathbf{Q}^m$ are used, with $\mathbf{V}$ a vector
- no unreachable state is involved in the computation

15.3.3 Proof of the algorithm

In this chapter we sketch the description of the proof method. More information about the proofs may be found in [7] (resp. [9]) for the synchronous (resp. asynchronous) case and detailed proofs are given in [8] and [10].

We first show that the reachability set RS of the original net is a subset of $\{(\mathbf{m}_k)_{k\in K} \mid \exists \overline{\mathcal{M}} \in \overline{XSRS}$ s.t. $\forall k \in K,\ \overline{\mathbf{m}}_k \in \widehat{\overline{\mathbf{m}}}_k\}$ which may be seen a "disaggregated" state space of $\overline{XSRS}$.

Algorithm 15.1 Computation of a performance measure

begin
  for each $k = 1, \ldots, K$ **do** compute $\overline{SRG}_k$ (hence $\overline{SRS}_k$) **od**
  for each $k = 1, \ldots, K$ **do**
    compute the $\mathbf{Q}'_k$ matrices from $\overline{SRG}_k$, using only local transitions
  od
  for each $t \in TX$ **do**
    for each $h = 1, \ldots K$ **do**
      (* compute $\mathbf{C}_h(t,d)$ and $\mathbf{A}_h(t,d)$ from $\overline{SRG}_h$: *)
      for all markings $(\widehat{\overline{\mathbf{m}}}_h, \widehat{\overline{\mathbf{m}}}'_h)$ **do** compute $c_h(t,d)_{\widehat{\overline{\mathbf{m}}}_h, \widehat{\overline{\mathbf{m}}}'_h}$ **od**
      for all markings $(\widehat{\overline{\mathbf{m}}}_h, \widehat{\overline{\mathbf{m}}}'_h)$ **do**
        compute $a_h(t,d)_{\widehat{\overline{\mathbf{m}}}_h, \widehat{\overline{\mathbf{m}}}'_h}$ as the diagonal compensation
        of the $(c_h(t,d)_{\widehat{\overline{\mathbf{m}}}_h, \widehat{\overline{\mathbf{m}}}''_h})$
      od
    od
  od
  compute the performance measure using the tensor expression of $\mathbf{Q}'$.
end

Secondly we prove that $\mathcal{A}(\mathbf{m}) = (\widehat{\overline{\mathbf{m}}}_k)_{k \in K} \in \overline{XSRS}$, (which is the K-tuple of the *symbolic markings of the extended sub-markings* $\overline{\mathbf{m}}_k$ of $\mathbf{m}$) is an aggregation function over RS verifying the strong lumpability condition.

Finally we show that the transition rate from any marking $\overline{\mathcal{M}}$ of $\overline{XSRS}$ is effectively given in the previous algorithm.

The proof is established in several steps:

- definition of a set of *semantic conditions*, that is to say at the marking level,
- proof that these conditions imply the strong lumpability condition (see chapter 13) (this is the core of the proof),
- proof that the syntactic conditions of aggregation imply those semantic conditions,
- with a careful examination of firing colours of synchronization transitions in each subnet, proof that the generator of the aggregated CTMC is a submatrix of $\mathbf{Q}'$ given in the algorithm.

The introduction of a semantic intermediate level is motivated by two reasons:

- the semantic conditions provide a guideline for the definition of syntactic conditions: for each of the first one, we try to find a syntactic translation.

- given the same set of semantic conditions, we shall be able to find other sets of syntactic conditions, for special classes of nets, reducing the new proof to the fact that syntactic conditions imply semantic conditions.

15.4 Synchronous composition of SWN

15.4.1 An introducing example

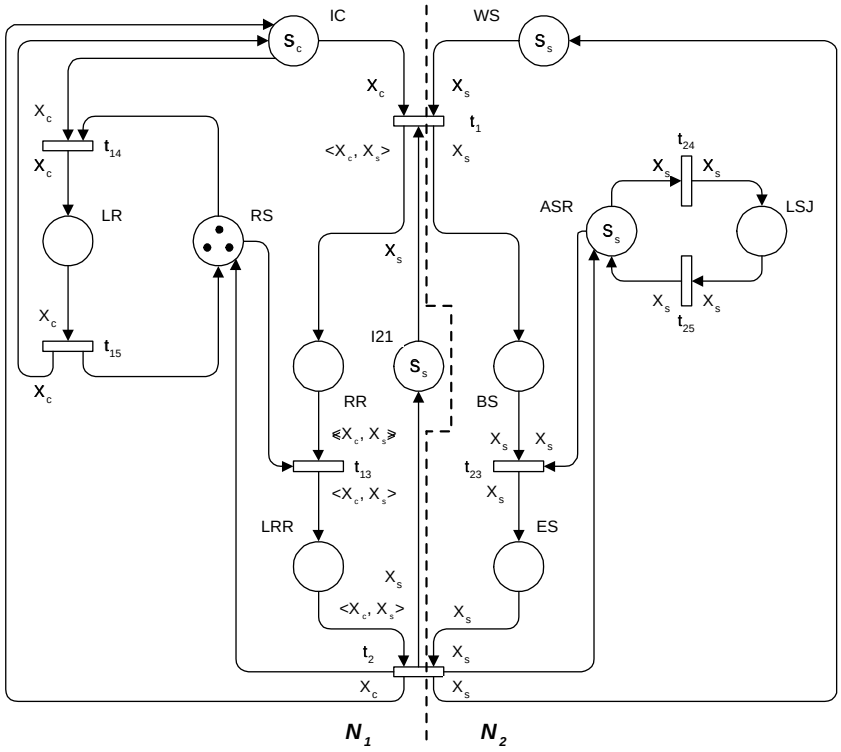


Figure 15.7: Example of synchronized SWN

We shall illustrate definitions and results of this section through the example SWN of Fig. 15.7. The net $\mathcal{N}$ is composed of two nets modeling a simple client ($\mathcal{N}_1$) – server ($\mathcal{N}_2$) system. We have two colour classes (C_c for the clients, C_s for the servers) and a neutral colour $E = \{e\}$ (tokens "without colour" in RS for instance). The clients can make local computations (place LR) or request a server (transition t_1). These two jobs require local resources (place RS) which are limited (3 tokens initially in RS). An important feature of this system is that during request servicing, a common client-server job (transition t_{13}) is running concurrently with the server job (transition t_{23}). The servers wait for a request

(place WS) and may also run local jobs (repair for example, places ASR and LSJ). When a server runs a request it cannot run simultaneously local jobs (transition t_{23}).

The colour domains of the places and the transitions are[3]:

- C_c for IC, LR, t_{14} and t_{15},
- $C_c \times C_s$ for RR, LRR, t_1 and t_{13}, t_2 (the neutral colour is not used to select a firing instance of these transitions),
- E for RS,
- C_s for I21, WS, BS, ES, ASR, t_{24}, LSJ and t_{25},
- $C_s \times C_s$ for t_{23}.

The role of the implicit place I21 will be explained in Sect. 15.4.3.

We give below the formal definition of synchronous composition of SWN corresponding to fusion (or superposition, see chapter 12) of transitions and related notations. The net is a collection of subnets with common transitions various attributes of which must be coherent.

Definition 15.1 *The synchronous composition of the SWN* $(\mathcal{S}_k = \langle P_k, T_k, \mathcal{C}, cd_k, \mathbf{Pre}_k, \mathbf{Post}_k, \mathbf{guard}_k, \mathbf{pri}_k, \mathbf{m_0}_k, \theta_k\rangle)_{k\in K}$ *is the SWN* $\mathcal{S} = \langle P, T, \mathcal{C}, cd, \mathbf{Pre}, \mathbf{Post}, \mathbf{guard}, \mathbf{pri}, \mathbf{m_0}, \theta\rangle$ *with:*

1. $\mathcal{C} = \{C_i, i \in I\}$ *(the (common) set of basic colour classes of every* $\mathcal{S}_k$*),*
2. $P = \biguplus_{k\in K} P_k$ *(*$(P_k)_{k\in K}$ *is a partition of the set of places).* $\mathbf{m_0}[p] = \mathbf{m_0}_k[p]$ *if* $p \in P_k$ *(the initial marking),*
3. $T = \bigcup_{k\in K} T_k$ *with,* $\forall t \in T_k \bigcap T_{k'}$, $cd_k(t) = cd_{k'}(t)$, $\mathbf{pri}_k(t) = \mathbf{pri}_{k'}(t)$, $\mathbf{guard}_k(t) = \mathbf{guard}_{k'}(t)$ *et* $\theta_k(t) = \theta_{k'}(t)$ *(the set of transitions);*
   (a) $cd(t) = cd_k(t)$ *if* $t \in T_k$ *and* $cd(p) = cd_k(p)$ *if* $p \in P_k$ *(cd defines the colour domain of a node x:* $cd(x) = \prod_{i\in I} C_i^{e_i}$ *with* $(e_i)_{i\in I} \in \mathrm{Bag}(I)$*),*
   (b) $\mathbf{guard}(t) = \mathbf{guard}_k(t)$ *if* $t \in T_k$ *(the guard function),*
   (c) $\mathbf{pri}(t) = \mathbf{pri}_k(t)$ *if* $t \in N_k$ *(the priority function),*
   (d) $\theta(t) = \theta_k(t)$ *if* $t \in T_k$ *(the transition rate or weight function).*
4. $\mathbf{Pre}(p,t) = \mathbf{Pre}_k(p,t)$ *and* $\mathbf{Post}(p,t) = \mathbf{Post}_k(p,t)$ *if* $p \in P_k$ *(the incidence functions).*

The following definition will be used to state the conditions on the synchronization transitions.

Definition 15.2 *Let* t *a transition of* $\mathcal{N}$*:*

[3] for ease of reading, we replace standard class indexed notation i, with alphabetical notation c, s.

- $Var(t) = \{X \mid X$ *occurs in some* $\mathbf{Pre}(p,t)$ *or in some* $\mathbf{Post}(p,t)\}$ *(input or output variables of* t*);*
- $Range(t,X) = \{k \in K \mid \exists p \in P_k$ *such that* X *occurs in some* $\mathbf{Pre}(p,t)$ *or in some* $\mathbf{Post}(p,t)\}$ *(indexes of the subnets which have input or output place(s) for* X *and* t*).*

In the formal definitions bellow, we also need some notations:

- $TS = \bigcup_{k \neq k'} T_k \bigcap T_{k'}$ is the set of synchronization transitions, corresponding to the set TX of Sect. 15.3.
- $\mathbf{m} = (\mathbf{m}_k)_{k \in K}$ is a marking of $\mathcal{S}$ with $\mathbf{m}_k$ its restriction to $\mathcal{S}_k$.
- RS is the RS of $\mathcal{S}$.
- S is the group of admissible permutations of $\mathcal{N}$.
- SRS_k (resp. SRG_k) is the SRS (resp. the SRG) of $\mathcal{S}_k$.

15.4.2 Two-phases synchronous composition of SWN

As we have shown in previous sections, any composition does not allow, in general, a combined aggregation/decomposition. So, we introduce conditions which imply the "orthogonality" of the two methods: we restrict ourselves to systems with two by two synchronization between subnets (hence the pairwise indexes), but a given subnet may be synchronized with several other subnets; moreover each synchronization is an asymmetrical client-server type relation, which is one of the most used relationship in practical applications.

Definition 15.3 *$\mathcal{S}$ is a two-phases synchronous composition of the SWN $(\mathcal{S}_k)_{k \in K}$ iff:*

1. *$\mathcal{S}$ is the synchronous composition of the $(\mathcal{S}_k)_{k \in K}$,*
2. *$\forall k \in K$ there are $(P_{k,k'})_{k' \in K}$ such that $P_k = \biguplus_{k' \in K} P_{k,k'}$,*
3. *$\forall k \in K$ there are $(T_{k,k'})_{k' \in K}$, $T^-_{k,k'}$ and $T^+_{k,k'}$ s.t. $T_k = \biguplus_{k' \in K} T_{k,k'}$ and $T_{k,k'} = T^-_{k,k'} \biguplus T^+_{k,k'}$, with the following properties:*
4. *$\forall k,k',l,l' \in K$ if $T_{k,k'} \bigcap T_{l,l'} \neq \emptyset$ then $(k,k') = (l,l')$ or $k \neq k'$ and $k' = l = l'$ or $l \neq l'$ and $l' = k = k'$,*
5. *$\forall k,k' \in K \; k \neq k' \; \left(T^\bullet_{k,k'} \bigcup {}^\bullet T_{k,k'}\right) \bigcap P_k \subseteq P_{k,k'} \biguplus P_{k,k}$ ($T^\bullet$ and ${}^\bullet T$ are the output and input places of the transitions of T),*
6. *$\forall k \in K \; \left(T^\bullet_{k,k} \bigcup {}^\bullet T_{k,k}\right) \bigcap P_k \subseteq P_{k,k}$.*

This definition can be intuitively explained as follows:

- One can identify two phases in a synchronization (described here with $\mathcal{N}_k$ the client part and $\mathcal{N}_{k'}$ the server part): a beginning, modelled by a firing of a transition of $T^-_{k,k'} \bigcap T_{k',k'}$, which does not involve yet synchronized objects of each subnet, and a continuation and an end, both using firings of transitions of $T^+_{k,k'}$ which may be also in $T_{k',k'}$; during this second phase, there is no new synchronized objects between the two subnets, and the end of synchronization means that these objects go back to their respective subnets. This temporal order in the behaviour of the synchronized objects explains the name we gave to this kind of synchronization.

- $P_{k,k'}$ is the set of input and output places of $\mathcal{N}_k$ which are involved only in the synchronization client $\mathcal{N}_k$ - server $\mathcal{N}_{k'}$.

- The different sets of transitions are the following:

  - $T^-_{k,k'}$ is the set of beginning synchronization transitions when $\mathcal{N}_k$ plays the role of the client and $\mathcal{N}_{k'}$ the role of the server,
  - $T^+_{k,k'}$ is the set of continuing and ending synchronization transitions when $\mathcal{N}_k$ plays the role of the client and $\mathcal{N}_{k'}$ the role of the server,
  - $T_{k,k} \setminus \biguplus_{k' \neq k} T_{k',k}$ is the set of transitions of $\mathcal{N}_k$ modelling local activities in $\mathcal{N}_k$, not involved in synchronization with other subnets,

In our example, $\mathcal{N}_1$ and $\mathcal{N}_2$ exhibit one two-phases synchronization where $\mathcal{N}_2$ plays the role of the server; it starts with t_1 which is the unique element of $T^-_{1,2}$. A t_1 firing is followed by t_{13} and t_{23} firings, and the synchronization ends with a firing of t_2. Hence we have $T^+_{1,2} = \{t_{13}, t_2\}$. The places corresponding to this synchronization are $P_{1,2} = \{\text{RR, LRR, I21}\}$, $P_{2,2} = P_2$. For $\mathcal{N}_1$ we have $T_{2,1} = \emptyset$ and $P_{2,1} = \emptyset$, $P_{1,1} = \{\text{IC, LR, RS}\}$.

Let us remark that the set T of transitions of $\mathcal{N}$ can be decomposed as

$$T = \left(\biguplus_{k \in K} \left(T_{k,k} \setminus \biguplus_{k' \neq k} T_{k',k} \right) \right) \biguplus \left(\biguplus_{k \neq k'} T^-_{k,k'} \right) \biguplus \left(\biguplus_{k \neq k'} T^+_{k,k'} \right)$$

so we introduce the following definitions:

Definition 15.4

- $t \in \biguplus_{k \in K} (T_{k,k} \setminus \biguplus_{k' \neq k} T_{k',k})$ *is called a local transition,*
- $t \in \biguplus_{k \neq k'} T^-_{k,k'}$ *is called an input synchronization transition,*
- $t \in \biguplus_{k \neq k'} T^+_{k,k'}$ *is called an output synchronization transition.*

In our example, the local transitions are t_{14}, t_{15} (in $\mathcal{N}_1$) and t_{23}, t_{24}, t_{25} (in $\mathcal{N}_2$), t_1 is the unique input synchronization transition, t_{13} (in $\mathcal{N}_1$), t_{23} (in $\mathcal{N}_2$) and t_2 (in $\mathcal{N}_1$ and $\mathcal{N}_2$) are the output synchronization transitions and $TS = \{t_1, t_{13}, t_2\}$.

15.4.3 Syntactic conditions

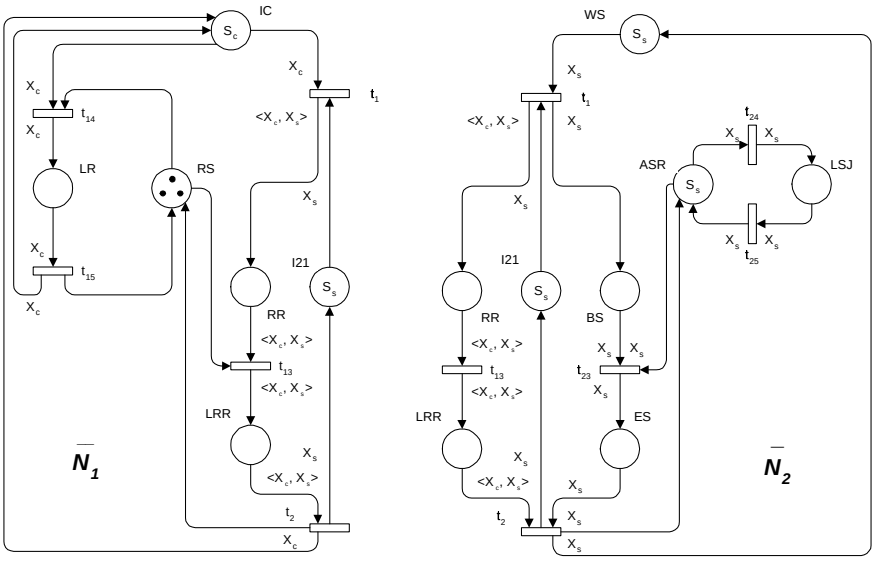


Figure 15.8: $\overline{\mathcal{N}}_1$ and $\overline{\mathcal{N}}_2$ SWN for the example SWN

Synchronous composition is not, by itself, sufficient to assure an independent application of aggregation and decomposition: we need to strengthen these conditions by new theoretical hypotheses, which in fact, are also natural ones. We first define the modified subnets allowing decomposition under appropriate conditions, set additional notations and define the aggregation function we shall use. Then we give the set of syntactic conditions allowing a combined aggregation/decomposition.

Definition 15.5 *Let $\mathcal{S}$ be a two-phases synchronous composition of the $(\mathcal{S}_k)_{k\in K}$. The extension of $\mathcal{N}_k$ is the SWN $\overline{\mathcal{N}}_k = \mathcal{N}_k \bigcup(\bigcup_{k'\neq k} \mathcal{N}_{k',k})$ where $\mathcal{N}_{k',k}$ is the subnet of $\mathcal{N}_{k'}$ restricted to places of $P_{k',k}$, transitions of $T_{k',k}$ and arcs between these nodes (and we denote by $\overline{P}_k$ the set of places of $\overline{\mathcal{N}}_k$).*

The extension of $\mathcal{N}_k$ includes all the client parts of the $\mathcal{N}_{k'}$ $(k' \neq k)$ which are synchronized with $\mathcal{N}_k$ and for which $\mathcal{N}_k$ is the server side.

In our example we have: $\overline{\mathcal{N}}_1$ is exactly $\mathcal{N}_1$ and $\overline{\mathcal{N}}_2 = \mathcal{N}_2 \biguplus \mathcal{N}_{12}$ so that only $\mathcal{N}_2$ is extended (see in Fig. 15.8 the SWN $\overline{\mathcal{N}}_1$ and $\overline{\mathcal{N}}_2$).

Notations

- $\overline{\mathbf{m}}_k$ is the restriction of $\mathbf{m}$ to $\overline{P}_k$,
- $\overline{SRS}_k$ (resp. $\overline{SRG}_k$) is the SRS (resp. the SRG) of $\overline{\mathcal{S}}_k$,

- $\overline{XSRS} = \prod_{k \in K} \overline{SRS}_k$,
- $\overline{\mathcal{M}} = (\widehat{\overline{\mathbf{m}}}_k)_{k \in K}$ is an element of $\overline{XSRS}$,
- $\mathcal{D}(\overline{XSRS}) = \{\mathbf{m} \mid \exists\, \overline{\mathcal{M}} \in \overline{XSRS} \text{ s.t. } \forall k \in K,\ \overline{\mathbf{m}}_k \in \widehat{\overline{\mathbf{m}}}_k\}$

Definition 15.6 *Let $\mathcal{S}$ be a two-phases synchronous composition of the $(\mathcal{S}_k)_{k \in K}$. The aggregation function $\mathcal{A}$ is:*

$$\mathcal{A}(\mathbf{m}) = (\widehat{\overline{\mathbf{m}}}_k)_{k \in K} \in \overline{XSRS}$$

for any marking $\mathbf{m}$ of $\mathcal{S}$, with $\forall k \in K$, $\widehat{\overline{\mathbf{m}}}_k$ the symbolic marking of $\overline{\mathbf{m}}_k$ in $\overline{\mathcal{S}}_k$.

We now give a first set of conditions for which the algorithm outlined in the preceding section may be used to compute performance measures of $\mathcal{S}$. As usual with (stochastic) Petri nets, such conditions must be expressed at the *net definition level* that is to say relative to either syntactic properties (like colour domains, incidence function expressions, ...) or to properties which may be checked only using these syntactic properties, like semiflows, ..., to avoid checking of the RG of $\mathcal{S}$. The reader will find definitions about semiflows in chapter 7.

Definition 15.7 *We say that $\mathcal{S}$ fulfills the syntactic conditions of aggregation iff $\forall k, k' \in K,\ k \neq k'$ we have the following properties:*

1. *If a colour class C_i is in the colour domain of $\mathcal{N}_{k,k'}$ and $\overline{\mathcal{N}}_k \backslash \mathcal{N}_{k,k'}$: for every $p \in \overline{P}_k \backslash P_{k,k'}$ with $cd(p) = \prod_{j \in I} C_j^{e_j}$, and every $p' \in P_{k,k'}$ with $cd(p') = \prod_{j \in I} C_j^{e'_j}$:*
   $\forall 1 \leq j \leq e_i,\ 1 \leq j' \leq e'_i$ there is a semiflow in $\overline{\mathcal{S}}_k$, $f = (f_q)_{q \in \overline{P}_k}$ on C_i s.t. $\forall \mathbf{m}$, $f(\mathbf{m}) = S_i$ with:
   $$f_q = \begin{cases} 0 \text{ or a projection} & \text{if} \quad q \neq p, q \neq p' \\ j\text{th projection} & \text{if} \quad q = p \\ j'\text{th projection} & \text{if} \quad q = p' \end{cases}$$
2. *If a colour class C_i is in the colour domain of $\mathcal{N}_{k,k'}$ but not in the colour domain of $\overline{\mathcal{N}}_k \backslash \mathcal{N}_{k,k'}$ then C_i is in the colour domain of at least two places and one of them, denoted by $p_{i,k,k'}$, verifies $cd(p_{i,k,k'}) = C_i$; moreover for every $p \in P_{k,k'}$, $p \neq p_{i,k,k'}$, with $cd(p) = \prod_{j \in I} C_j^{e_j}$:*
   $\forall 1 \leq j \leq e_i$ there is a semiflow (f_q) in $\mathcal{S}_{k,k'}$ on C_i s.t. $\forall \mathbf{m}$, $f(\mathbf{m}) = S_i$ with:
   $$f_q = \begin{cases} 0 \text{ or a projection} & \text{if} \quad q \neq p, q \neq p_{i,k,k'} \\ j\text{th projection} & \text{if} \quad q = p \\ \text{Identity (on } C_i) & \text{if} \quad q = p_{i,k,k'} \end{cases}$$
3. *$\forall t \in T^+_{k,k'}$, $\forall X \in Var(t)$*
   (a) *$\exists p \in P_{k,k'}$, $p \neq p_{i,k,k'}$, s.t. $\mathbf{Pre}(p,t)$ contains a positive term $\langle \ldots, X, \ldots \rangle$,*

(b) $\forall p \in P_k \backslash P_{k,k'}$ X *does not occur in* $\mathbf{Pre}(p,t)$.

4. $\forall t \in T^-_{k,k'}$ $Var(t) = Var_k(t) \biguplus Var_{k'}(t)$ *(hence* $cd(t) = C^k(t) \times C^{k'}(t)$*) with:*

   (a) $\forall X \in Var_k(t)$ $\forall p \in P_{k'}$, X *is neither in* $\mathbf{Pre}(p,t)$ *nor in* $\mathbf{Post}(p,t)$,

   (b) $\forall X \in Var_{k'}(t)$ *with domain* C_i *with* $p_{i,k,k'}$, $\mathbf{Pre}(p_{i,k,k'},t) = X$ *and* $\forall p \in P_k$, $p \neq p_{i,k,k'}$, X *does not occur in* $\mathbf{Pre}(p,t)$,

   (c) $\forall X \in Var_{k'}(t)$ *with domain* C_i *without* $p_{i,k,k'}$, $\forall p \in P_k$, X *does not occur in* $\mathbf{Pre}(p,t)$ *nor in* $\mathbf{Post}(p,t)$,

   (d) Let $C^{k'}(t) = \prod_{i \in I} C_i^{e_i}$. *If* $e_i > 0$ *then* C_i *occurs neither in the colour domain of any place of* $\overline{P}_k \backslash P_{k,k'}$ *nor in* $C^k(t)$.

Let us explain the role of these conditions (application to our example are indicated between parentheses).

Conditions 1 and 2 ensure one aspect of the two-phases synchronization: for a given basic colour class used in the synchronization, a colour used in the synchronization phase must leave it before being reused in it. Colours from the client side are constrained by condition 1 (C_c in $\overline{\mathcal{N}}_1 \backslash \mathcal{N}_{12}$ and $\mathcal{N}_{12}$). Condition 2 applies to colours from the server side (C_s in $\mathcal{N}_{12}$): the $p_{i,k,k'}$ ($p_{s,1,2} = \text{I21}$) may be seen as a "condensed view" of $\mathcal{S}_{k'}$ in the client's subnet with respect to C_i and the synchronization.

Conditions 3a and 3b mean that $T^+_{k,k'}$ (t_{13}, t_2) is the set of transitions of the "internal" or "output" phase of the synchronization: their firing colours must be controlled by colours of $P_{k,k'}$ (3a) and only those ones (3b).

Conditions 4, 4a, 4b, 4c and 4d refer to the beginning of the synchronization: the colour classes involved in this phase may be partitioned in two disjoints subsets: those from $C^{k'}(t)$ correspond to server colour classes (C_s) while those from $C^k(t)$ correspond to client colour classes (C_c): condition 4a ensures that clients beginning the synchronization are not related to other entities from $\mathcal{S}_{k'}$. Conditions 4b and 4c supplements conditions 1 and 2: the server colour classes are allowed to control the beginning of the synchronization in the client subnet only by means of the control place $p_{i,k,k'}$ (I21) if these classes exist in the synchronization part of the client, and only by places in the server subnet otherwise. Condition 4d forbids server colours to appear in the local part of the client.

It is easy to show that our example SWN fulfills these conditions. Please note however that we introduced an additional place I21 which ensures the syntactic condition 2 (place $p_{s,1,2}$ for the colour C_s). Addition of such places is often used with decomposition methods and does not present any technical difficulty; usually -as it is the case here- these control places are implicit ones (see chapter 7 for implicit place definition and [2, 6] for implicit places and reduction methods in ordinary and coloured nets).

Let us observe that although these conditions seem strong, they are fulfilled for many nets which are models of systems with several components each having some kind of autonomy, and that they constitute a first set of syntactic conditions.

15.4.4 Algorithmic solution

The specific points of the general algorithm 15.1 are the expressions of the c_k and consequently a_k terms. As the expression of the a_k entirely follows from the ones of the c_k we only need the c_k expressions. Because of memory synchronization, the expression of $\mathbf{C}_k(t,d)$ and $\mathbf{A}_k(t,d)$ in (15.1) must be adapted: the cardinality of firings set of a symbolic firing of $t \in TS$ in $\overline{\mathcal{N}}_k$ depends upon type of t (input or output transition). If t is in $T^-_{k,k'}$, we have to count firings using colour classes of $C^k(t)$ when we are in $\overline{\mathcal{S}}_k$ and colour classes of $C^{k'}(t)$ when we are in $\overline{\mathcal{S}}_{k'}$. On the other hand, if t is in $T^+_{k,k'}$, we do not use colour classes of $cd(t)$ when we are in $\overline{\mathcal{S}}_k$.
So we need the following notations.

- $d = (d_1^1, \ldots, d_1^{e_1}, \ldots, d_i^1, \ldots, d_i^{e_i}, \ldots, d_n^1, \ldots, d_n^{e_n}) = (d_i^j)_n^{e_i(t)}$ (with $1 \leq d_i^j \leq n_i$ and n_i the number of static subclasses of C_i) is a choice of static subclasses relative to $cd(t)$,
- $\forall\ \langle \lambda, \mu \rangle$ (instantiation functions for the restriction t_k of t to $\overline{\mathcal{N}}_k$), $\widehat{\overline{\mathbf{m}}}_k$ and $\widehat{\overline{\mathbf{m}}}'_k$ in $\overline{SRS}_k$:
  - $(d(Z_i^{\lambda_i(j)}))_n^{e_i(t)}$ are the static subclasses of the dynamic subclasses (of the canonical representation of $\widehat{\overline{\mathbf{m}}}_k$) of $\langle \lambda, \mu \rangle$, $d^k = (d_i^j)_n^{e_i(t_k)}$ is the restriction of d relative to $cd(t_k)$ and

$$1_{(t,d,\langle\lambda,\mu\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)} = \begin{cases} 1 & \text{if} \quad d^k = (d(Z_i^{\lambda_i(j)}))_n^{e_i(t_k)} \\ & \text{and} \quad \widehat{\overline{\mathbf{m}}}_k[t_k(\langle\lambda,\mu\rangle)\rangle\widehat{\overline{\mathbf{m}}}'_k \\ 0 & \text{else} \end{cases}$$

$$1_{(t,d,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)} = \bigvee_{\langle\lambda,\mu\rangle} 1_{(t,d,\langle\lambda,\mu\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)}$$

    with $\bigvee$ denoting the Boolean addition (logical or),
  - for $t \in T^-_{k,k'}$ (we assume for ease of writing and without loss of generality, that $C^k(t) = \prod_{i=1}^{n(t)} C_i^{e_i(t)}$ and $C^{k'}(t) = \prod_{i=n(t)+1}^{n} C_i^{e_i(t)}$) we denote by $\langle \lambda, \mu \rangle^k$ the restriction of $\langle \lambda, \mu \rangle$ to $C^k(t)$, $\langle \gamma^k, \delta^k \rangle$ instantiation functions for $C^k(t)$ and

$$1_{(t,d,\langle\gamma^k,\delta^k\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)} = \bigvee_{\substack{\langle\lambda,\mu\rangle \\ \langle\lambda,\mu\rangle^k = \langle\gamma^k,\delta^k\rangle}} 1_{(t,d,\langle\lambda,\mu\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)}$$

$$F^-_{(\langle\gamma^k,\delta^k\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)} = \prod_{i=1}^{h(t)} \prod_{j=1}^{m_i} \frac{\mathrm{card}(Z_i^j)!}{(\mathrm{card}(Z_i^j) - \delta_i^{k,j})!}$$

$$F'^-_{(\langle\gamma^{k'},\delta^{k'}\rangle,\widehat{\overline{\mathbf{m}}}_{k'},\widehat{\overline{\mathbf{m}}}'_{k'})} = \prod_{i=n(t)+1}^{h'(t)} \prod_{j=1}^{m_i} \frac{\mathrm{card}(Z_i^j)!}{(\mathrm{card}(Z_i^j) - \delta_i^{k',j})!}$$

with $h(t)$ (resp. $h'(t)$) the highest index of non ordered basic colour classes of $C^k(t)$ (resp. $C^{k'}(t)$), m_i the number of dynamic subclasses of C_i in the canonical representation of $\widehat{\mathbf{m}}_k$ or $\widehat{\mathbf{m}}'_k$ and $\delta_i^{k,j} = \sup\{\delta_i^k(x) \mid \gamma_i^k(x) = j\}$ the number of different instantiations in the dynamic subclass Z_i^j for $\langle \gamma^k, \delta^k \rangle$ ($\delta_i^{k,j} = 0$ if Z_i^j is not instantiated).

– for $t \in T^+_{k,k'}$

$$F'^+_{(\langle \lambda,\mu \rangle, \widehat{\mathbf{m}}_{k'}, \widehat{\mathbf{m}}'_{k'})} = \prod_{i=1}^{h} \prod_{j=1}^{m_i} \frac{\mathrm{card}(Z_i^j)!}{(\mathrm{card}(Z_i^j) - \mu_i^j)!}$$

with h the highest index of non ordered basic colour classes of $cd(t)$ and m_i , μ_i^j as above.

Then, for a given $t \in TS$ (say $t \in T_{k,k'}$), we have the following expressions:

- if $h \neq k$ and $h \neq k'$, then $\mathbf{C}_h(t,d) = \mathbf{A}_h(t,d) = \mathbf{I}_{r_h}$
- if $h = k$ or $h = k'$ then $c_h(t,d)_{\overline{\mathcal{M}_h}, \overline{\mathcal{M}'_h}}$ is:

$$\begin{array}{ll}
\sum_{\langle \gamma^k, \delta^k \rangle} 1_{(t,d,\langle \gamma^k, \delta^k \rangle, \widehat{\mathbf{m}}_k, \widehat{\mathbf{m}}'_k)} \cdot F^-_{(\langle \gamma^k, \delta^k \rangle, \widehat{\mathbf{m}}_k, \widehat{\mathbf{m}}'_k)} & (h = k,\ \ t \in T^-_{k,k'}) \\
1_{(t,d,\widehat{\mathbf{m}}_k, \widehat{\mathbf{m}}'_k)} & (h = k,\ \ t \in T^+_{k,k'}) \\
\sum_{\langle \gamma^{k'}, \delta^{k'} \rangle} 1_{(t,d,\langle \gamma^{k'}, \delta^{k'} \rangle, \widehat{\mathbf{m}}_{k'}, \widehat{\mathbf{m}}'_{k'})} \cdot F'^-_{(\langle \gamma^{k'}, \delta^{k'} \rangle, \widehat{\mathbf{m}}_{k'}, \widehat{\mathbf{m}}'_{k'})} & (h = k',\ \ t \in T^-_{k,k'}) \\
\sum_{\langle \lambda, \mu \rangle} 1_{(t,d,\langle \lambda, \mu \rangle, \widehat{\mathbf{m}}_{k'}, \widehat{\mathbf{m}}'_{k'})} \cdot F'^+_{(\langle \lambda, \mu \rangle, \widehat{\mathbf{m}}_{k'}, \widehat{\mathbf{m}}'_{k'})} & (h = k',\ \ t \in T^+_{k,k'})
\end{array}$$

15.5 Asynchronous composition of SWN

15.5.1 An example

Asynchronous composition of Petri nets models communicating subsystems (subnets), with entities (tokens) moving from one subsystem to another one. The communication links between subnets are (output) transitions from the source subnet with at least one output place in the destination subnet.

Here also, we shall explain definitions and results through an example SWN. The net $\mathcal{N}$ (Fig. 15.9) is composed of three subnets modeling a simple client (local work, $\mathcal{N}_1$) – server (remote work, $\mathcal{N}_2$) with a repair (server repair, $\mathcal{N}_3$) system. Clients are initially located in the local area (place p_{13}) and servers in the remote site (place p_{25}). Clients emit a server request by pairs of neighbours (variables X and $!X$, the client class C_c is ordered). The requests are treated in the remote site where servers (class C_s) execute the requests (transitions t_{21}, t_{22} and t_{23}). A server may fail: in this case it must be repaired with two jobs located in the repair station (transitions t_{31}, t_{32} and t_{33}).

The basic colour classes are hence C_c and C_s and the colour domains of places and transitions (not shown in the figure for ease of reading) are:

- C_c for p_{11}, p_{12}, p_{13}, p_{21}, p_{22}, t_{11} ,t_{12} and t_{13},

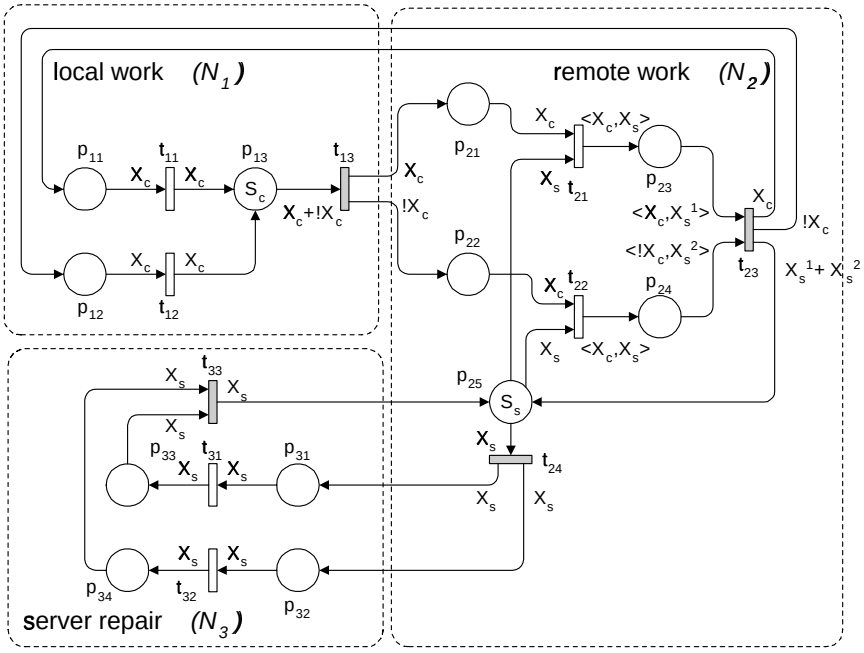


Figure 15.9: Example of asynchronous composition of SWN

- C_s for p_{25}, p_{31}, p_{32}, p_{33}, p_{34}, t_{24}, t_{31} , t_{32} and t_{33},
- $C_c \times C_s$ for p_{23}, p_{24}, t_{21} and t_{22},
- $C_c \times C_s^2$ for t_{23}.

The formal expression of asynchronous composition for a given class of Petri nets (GSPN, SWN, ...) is summarized in the following definition.

Definition 15.8 *Let $((P_k, T_k))_{k\in K}$ be families of places and transitions of the net system $\mathcal{S} = \langle P, T, \ldots\rangle$. $\mathcal{S}$ is the asynchronous composition of* its subsystems[4] *$(\mathcal{S}_k = \langle P_k, T_k, \ldots\rangle)_{k\in K}$ iff:*

1. $T = \biguplus_{k\in K} T_k$ *(partition of the transitions);*
2. $P = \biguplus_{k\in K} P_k$ *(partition of the places);*
3. $\forall k \in K$,
$$\forall t \in T_k,\ {}^\bullet t \bigcup {}^\circ t \subseteq P_k$$

In the rest of this section, for any marking $\mathbf{m}$ of $\mathcal{S}$, we denote by $\mathbf{m}_k = \mathbf{m}[P_k]$, so that $\mathbf{m} = (\mathbf{m}_k)_{k\in K}$. $TO_k = \{t \in T_k \mid t^\bullet \bigcap (P \backslash P_k) \neq \emptyset\}$ is the set of output

[4] restrictions of $\mathcal{S}$ to places of P_k, transitions of T_k and edges between them

transitions of $\mathcal{S}_k$ and $TO = \bigcup_{k \in K} TO_k$ is the set of all output transitions of $\mathcal{N}$ (the TX set of the Sect. 15.3).

15.5.2 Abstract view

In the asynchronous case, the first problem is to be able to identify subnets to compose since the $\mathcal{N}_k$ subnets are not autonomous. This means that we need to extend a subnet with some "view" of its environment to get an isolated subnet. As this enlargement must be done for every subnet, a convenient way is to define an abstract view of each $\mathcal{S}_k$ which will be used in the description of the environment of other subnets. In fact, we show that we can enlarge a subnet, both to include its environment and to allow the combination of aggregation and decomposition (following our general approach), if the abstract view of a subnet is adapted.

An abstract view has to enforce a set of constraints:

- it allows to hide details of behaviour of the subnet.
- it is consistent, that is to say: if $\mathbf{m}[\delta\rangle\mathbf{m}'$ with a sequence of firings $\delta = \tau_1^* \sigma \tau_2^*$ composed of local firings τ_1^* and τ_2^* to be hidden, then we must have: $a(\mathbf{m})[\sigma\rangle a(\mathbf{m}')$ with $a(\mathbf{m})$ the abstraction of the marking $\mathbf{m}$.
- it must let "visible" interactions between global entities.
- it has to be formally defined from the net description.
- it is compatible with a combined aggregation/decomposition method.

To take into account these constraints, the definition of this abstraction should be *guided by qualitative considerations*, especially by observing interactions between global entities (colour classes) inside each subnet keeping these interactions "visible" in the abstraction. In particular, it does not seem possible to abstract a subnet with *only one place*: we need at least one place to model entities of each basic colour class moving from one subnet to another one.

We propose to define each place through a partial[5] semiflow to ensure consistency of the abstraction and deal with formal definitions deduced from the net description. Since the important basic colour classes with regard to the synchronization are those of tokens moving from one subnet to another one (*global* classes), the abstract view will be based on these classes only. On the contrary, all colour of tokens staying in one subnet will be called *local*. Hence, we set the following definitions.

Definition 15.9 *A colour class C_i of $\mathcal{N}_k$ is* local *iff*

$$\forall t \in TO_k,\ \left(X_i^j \text{ is in } \mathbf{Post}(p,t) \Rightarrow p \in P_k\right)$$

[5] f is a partial semiflow on $\mathcal{N} = (P, T, \ldots)$ w.r.t a set $T' \subseteq T$ of transitions iff f is a semiflow on the net $\mathcal{N}' = (P, T', \ldots)$

A non local colour class is said global.

A colour of a local (resp. global) colour class is said local (resp. global).

Definition 15.10 *f is an* abstraction semiflow *w.r.t. to a global colour class C_i of $\mathcal{S}_k$ iff f is a partial semiflow of $\mathcal{S}_k$ with respect to $T_k \backslash TO_k$ s.t.:*

- *$cd(f) = C_i$.*
- *$\forall p \in P_k$, f_p is 0 or b times a projection (with b a positive constant).*

We can now formally define an abstract view: to ensure consistency of the abstract view of a subnet, every global colour class must be represented in this abstraction.

Definition 15.11 *A subnet system $\mathcal{S}_k$* has an abstract view *iff there is an abstraction semiflow w.r.t. to each global class of $\mathcal{S}_k$.*

Let $F_k = \{f_{C_i} \mid C_i$ global class of $\mathcal{N}_k\}$ the set of abstraction semiflows of a subnet system having an abstract view.

- *The* abstract view *of $\mathcal{S}_k$ is the set of places $PA_k = \{p_f \mid f \in F_k\}$ with:*
  - *$cd(p_f) = cd(f)$ (the colour domain of p_f);*
  - *$\mathbf{m_0}[p_f] = \sum_{p \in P_k} f(\mathbf{m_0}[p])$ (the initial marking of p_f).*
- *For every marking $\mathbf{m} = (\mathbf{m}_k)_{k \in K}$ of $\mathcal{S}$, the* abstract marking *of $\mathbf{m}_k$ is*

$$am(\mathbf{m}_k) = \left(\sum_{p \in P_k} f(\mathbf{m}[p]) \right)_{f \in F_k}$$

  also denoted by $\mathbf{m}[PA_k]$ by extension of $\mathbf{m}$ (the values of the semiflows of F_k in the marking $\mathbf{m}_k$).

Notice that a global class for $\mathcal{N}_k$ may be a non global one for another $\mathcal{N}_{k'}$. Such classes may be renamed with different names in each $\mathcal{N}_k$ where it is local. From now on, *we assume that such a renaming has been done.*

In our example SWN, we have two partial semiflows in $\mathcal{N}_2$ ($f_{2c} = X_c.p_{21} + X_c.p_{23} + X_c.p_{22} + X_c.p_{24}$ and $f_{2s} = X_s.p_{25} + X_s.p_{23} + X_s.p_{24}$) and one partial semiflow in $\mathcal{N}_3$ ($f_{3s} = X_s.p_{31} + X_s.p_{33}$).

15.5.3 Syntactic conditions of aggregation

As presented in the general overview of our method, we now define the enlargement of the subnets allowing decomposition under appropriate conditions (the notations and the aggregation function are the same as in the synchronous case, see Sect. 15.4): the extension of a subnet is built with the abstract views of all others subnets and the output transitions of the whole net, adapted to the abstract views.

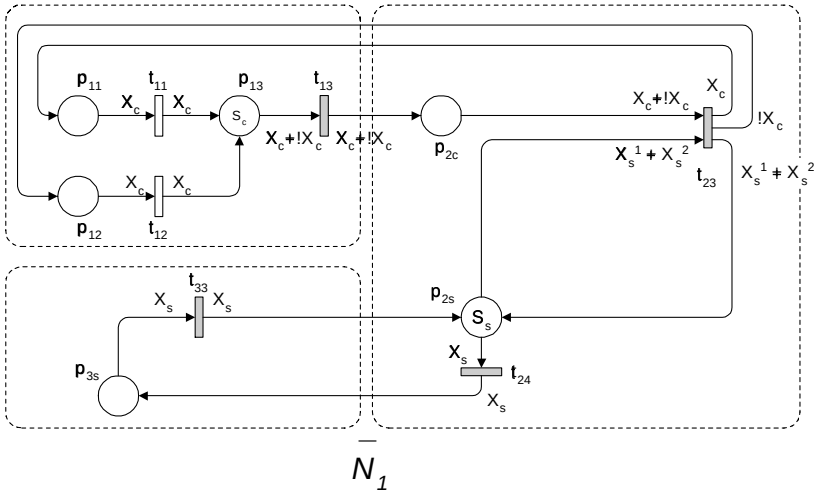


Figure 15.10: Subnet $\overline{\mathcal{S}}_1$ for the example SWN

Definition 15.12 *Let $\mathcal{S}$ be an asynchronous composition of the $(\mathcal{S}_k)_{k\in K}$. The* extension $\overline{\mathcal{S}}_k$ *of $\mathcal{S}_k$ is the SWN $\langle \overline{P}_k, \overline{T}_k, \mathcal{C}, \overline{cd}, \overline{\mathbf{Pre}}_k, \overline{\mathbf{Post}}_k, \overline{\mathbf{pri}}, \overline{\mathbf{m_0}}_k, \theta_k\rangle$ with:*

- $\overline{P}_k = P_k \bigcup_{k'\neq k} PA_{k'}$*: for each marking* $\mathbf{m}$ *of* $\mathcal{S}$*, the corresponding marking of* $\overline{\mathcal{S}}_k$ *is* $\overline{\mathbf{m}}_k = (\mathbf{m}_k, (\mathbf{m}[PA_{k'}])_{k'\neq k})$
- $\overline{T}_k = T_k \bigcup_{k'\neq k} TO_{k'}$
- $\forall p \in P_k,\ \forall t \in \overline{T}_k,\ \ \overline{\mathbf{Pre}}_k(p,t) = \mathbf{Pre}_k(p,t)$ *and* $\overline{\mathbf{Post}}_k(p,t) = \mathbf{Post}_k(p,t)$
- $\forall p_f \in PA_{k'}, \forall t \in \overline{T}_k, \overline{\mathbf{Pre}}(p_f,t) = \sum_{p'\in {}^\bullet t} f_{p'} \circ \mathbf{Pre}(p',t)$ *and* $\overline{\mathbf{Post}}(p_f,t) = \sum_{p'\in t^\bullet} f_{p'} \circ \mathbf{Post}(p',t)$

$\overline{\mathbf{m}}_k[PA_{k'}]$ *is still named the* abstract marking *of $\mathcal{S}_{k'}$ (in $\overline{\mathbf{m}}_k$).*

Figure 15.10 shows the extension $\overline{\mathcal{S}}_1$ of $\mathcal{S}_1$. We see that in PA_2 we have only two places p_{2x} and p_{2y} for the colours C_c and C_s, and in PA_3 one place for C_s. The transitions t_{13}, t_{23}, t_{24} and t_{33} are also modified accordingly with our definition.

We could also define a full abstract view $\overline{\mathcal{S}}$ of $\mathcal{S}$ with abstract views only of all subnets, useful in a hierarchical design process, however our method does not use $\overline{\mathcal{S}}$.

In order to be able to apply a combined aggregation/decomposition method of resolution, we have, as in the synchronous case, to add syntactic conditions

to the model: roughly speaking, they mean that for each global colour class and every marking, the subsets of colours in the abstract views are a partition of this class. Here also, these conditions are fulfilled for many nets which are models of systems with several components each having some kind of autonomy.

Definition 15.13 *We say that $\mathcal{S}$ fulfills the syntactic conditions of aggregation iff $\forall k \in K$ we have the following properties:*

1. *$\forall p \in {}^\bullet TO_k$ with $cd(p) = \prod_{i\in I} C_i^{e_i}$, $\forall i \in I$ s.t. C_i is a global colour of $\mathcal{N}_k$ and $e_i > 0$ we have: $\forall 1 \leq j \leq e_i$, X_i^j (the jth projection on C_i) is in the abstraction semiflow $f_{C_i} \in F_k$.*

2. *$\forall t \in TO_k$, $\forall X \in Var(t)$ corresponding to a global colour class of $\mathcal{N}_k$, X is in a positive term of some $\mathbf{Pre}(p,t)$.*

3. *$\forall p \in PA_k$ with $cd(p) = C_l$, $\forall k' \neq k$, $\forall p' \in P_{k'}$ with $cd(p') = \prod_{i\in I} C_i^{e'_i}$: $\forall\ 1 \leq j' \leq e'_l$ there is a semiflow $g = (g_q)_{q\in\overline{P}_{k'}}$ on C_l in $\overline{\mathcal{S}}_{k'}$, s.t. $\forall \mathbf{m}$, $g(\overline{\mathbf{m}}_{k'}) = S_l$ with:*
$$g_q = \begin{cases} 0 \text{ or a projection} & \text{if} \quad q \neq p, q \neq p' \\ Identity & \text{if} \quad q = p \\ j'\text{th projection} & \text{if} \quad q = p' \end{cases}$$

Informally, these conditions mean first that all activities must keep their identities when they move from one subnet to another one (condition 1). Secondly, only *transfer* of activities may occur between subnets, no creation neither destruction (condition 2). And finally, the activities from a given global colour are located in only one subnet, at a given time (condition 3).

It is easy to verify that our example SWN fulfills these conditions.

15.5.4 Details of the algorithm

In order to give the expression of the c_k terms of the generator, let us denote:

- $\forall\ \langle\lambda,\mu\rangle$ (instantiation functions in $\overline{\mathcal{S}}_k$ for t), $\widehat{\overline{\mathbf{m}}}_k$ and $\widehat{\overline{\mathbf{m}}}'_k$ in $\overline{SRS}_k$:
$$1_{(t,d,\langle\lambda,\mu\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)} = \begin{cases} 1 & \text{if} \quad d = (d(Z_i^{\lambda_i(j)}))_n^{e_i(t)} \\ & \text{and} \quad \widehat{\overline{\mathbf{m}}}_k[t(\langle\lambda,\mu\rangle)\rangle\widehat{\overline{\mathbf{m}}}'_k \\ 0 & \text{else} \end{cases}$$
$$1_{(t,d,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)} = \bigvee_{\langle\lambda,\mu\rangle} 1_{(t,d,\langle\lambda,\mu\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)}$$
with $\bigvee$ denoting the Boolean addition (logical or).

- for $t \in TO_k$
$$F_{(\langle\lambda,\mu\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)} = \prod_{i=1}^{h}\prod_{j=1}^{m_i} \frac{\mathrm{card}(Z_i^j)!}{(\mathrm{card}(Z_i^j) - \mu_i^j)!}$$
with h the highest index of non ordered basic colour classes of $cd(t)$.

Then, for a given $t \in TO_k$, we have the following expressions of the algorithm 15.1:

- if $h \neq k$ and $t^\bullet \bigcap P_h = \emptyset$ then $\mathbf{C}_h(t,d) = \mathbf{A}_h(t,d) = \mathbf{I}_{r_h}$
- if $h = k$ or $h = k'$ then $c_h(t,d)_{\overline{\mathcal{M}}_h, \overline{\mathcal{M}}'_h}$ is:

$$\begin{array}{ll} \sum_{\langle\lambda,\mu\rangle} 1_{(t,d,\langle\lambda,\mu\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)} \cdot F_{(\langle\lambda,\mu\rangle,\widehat{\overline{\mathbf{m}}}_k,\widehat{\overline{\mathbf{m}}}'_k)} & (h = k) \\ 1_{(t,d,\widehat{\overline{\mathbf{m}}}_h,\widehat{\overline{\mathbf{m}}}'_h)} & (h \neq k \text{ and } t^\bullet \bigcap P_h \neq \emptyset) \end{array}$$

15.6 Discussion and extensions

In this last section we first discuss the applicability of the presented methods and the possible complexity savings that they provide. Then we indicate some conceivable extensions and we conclude with some methodological remarks.

Modern systems are more and more designed as sets of components (sometimes already modelled) which exhibit some kind of synchronization. For systems with symmetrical behaviour of entities, the question is to recognize among these synchronization, those for which we may apply the synchronous or the asynchronous composition. The synchronous composition of SWN seems well suited for the classical paradigm of synchronous interaction like "rendez-vous". The client-server model, for its part, must be modelled either as an asynchronous or a synchronous composition of SWN, depending on the behaviour of clients during service: if clients continue activities related to the server identities we need a synchronous composition. In contrast, if clients continue jobs which do not involve the server identities, then an asynchronous composition can model the interaction.

We may notice however, that the asynchronous syntactic conditions appear stronger than the synchronous ones: this follows from the stronger nature of the asynchronous composition, in particular, the preservation of the client entities during the service may be a too strong constraint to model Remote Procedure Calls or device driver system calls, with a simple asynchronous composition.

Concerning the complexity reduction of the computation of the performance measures, we have to distinguish the asynchronous composition from the asynchronous composition. In the synchronous case, the savings depend on the relative complexity of the three parts: local work of the clients (1), synchronized work of the clients (2) and server work without client (3). For a given part (2), the more parts (1) and/or (3) are complex, the more the composition method reduces the resolution complexity. Nevertheless, a set of case-studies will have to clarify the effective reduction, with a SWN tool like GreatSPN [5]. The reduction provided by the asynchronous composition appears easier to estimate: the more a subnet includes entities (colour classes) without interaction with global entities, the smaller will be its abstract view, and consequently, the larger will be the complexity reduction.

About further extensions of the present work, a first aim will be to incorporate immediate transitions (with priorities) and the dependency of the rate (or weight) of transitions with respect to the static partitions of the markings, like in the general SWN formalism. A preliminary study has shown that it is necessary to restrict immediate transitions to local parts of the subnets in the synchronous composition and to non output transitions in the asynchronous composition. Similarly, restrictions on the dependency of transition rates have shown to be necessary.

Among middle term perspectives, we shall try to relax the conditions of the asynchronous composition and, at the same time, to reinforce the constraints on the local behaviour of the global entities. This new sharing out of the applicability conditions could allow us to model a larger range of systems.

In a broader perspective, it would be opportune to develop methodologies for the composition/decomposition of models taking into account the synchronous and asynchronous composition of SWN, preferably combined with a reduction/synthesis approach. A modular design leads to composition of subsystems and is probably the easiest way to apply the results of this chapter.

Moreover, starting from a large net, it is more difficult to decompose it so as to be able to apply directly the method of composition of SWN. Therefore, we need to define methodologies and heurisms to identify parts of the net allowing the decomposition of the net into smaller subnets.

Bibliography

[1] G. Balbo, G. Chiola, S.C. Bruell, and P. Chen. An example of modelling and evaluation of a concurrent program using coloured stochastic Petri nets: Lamport's fast mutual exclusion algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):221–240, March 1992. Reprinted in *High-Level Petri Nets. Theory and Application*, K. Jensen and G. Rozenberg (editors), Springer Verlag, 1991.

[2] G. Berthelot. *Transformations et Analyse de Reseaux de Petri. Application aux Protocoles.* PhD thesis, Université P. et M. Curie, Paris, France, 1983. Thèse d'Etat, (in French).

[3] P. Buchholz. Aggregation and reduction techniques for hierarchical GC-SPNs. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, pages 216–225, Toulouse, France, October 1993. IEEE-CS Press.

[4] P. Buchholz. Hierarchies in colored GSPNs. In *Proc.* 14^{th} *Intern. Conference on Application and Theory of Petri Nets*, volume 691 of *LNCS*, Chicago, Illinois, June 1993. Springer Verlag.

[5] G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. To appear in Performance Evaluation: special issue on Performance Modeling Tools.

[6] S. Haddad. *A reduction theory for coloured nets*, volume 424 of *LNCS*, pages 209–235. Springer–Verlag, 1989. reprinted in [11] pp. 399–425.

[7] S. Haddad and P. Moreaux. Evaluation of high level Petri nets by means of aggregation and decomposition. In *Proc. of the 6th International Workshop on Petri Nets and Performance Models*, pages 11–20, Durham, NC, USA, October 3–6 1995. IEEE Computer Society Press.

[8] S. Haddad and P. Moreaux. Aggregation and decomposition for performance evaluation of synchronous product of high level Petri nets. Document du Lamsade 96, LAMSADE, Université Paris Dauphine, Paris, France, September 1996.

[9] S. Haddad and P. Moreaux. Asynchronous composition of high level Petri nets: a quantitative approach. In *Proc. of the 17th International Conference on Application and Theory of Petri Nets*, number 1091 in LNCS, pages 193–211, Osaka, Japan, June 24–28 1996. Springer–Verlag.

[10] S. Haddad and P. Moreaux. Aggregation and decomposition for performance evaluation of asynchronous product of high level Petri nets. Document du Lamsade 102, LAMSADE, Université Paris Dauphine, Paris, France, May 1997.

[11] K. Jensen and G. Rozenberg, editors. *High-Level Petri Nets. Theory and Application.* Springer Verlag, 1991.

[12] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proc. 1985 SIGMETRICS Conference*, pages 147–154, Austin, TX, USA, August 1985. ACM.

[13] B. Plateau and K. Atif. Stochastic automata network for modeling parallel systems. *IEEE Transactions on software engineering*, 17(10):1093–1108, 1991.

Chapter 16

Compositional Nets and Compositional Aggregation

16.1 Introduction

Designing accurate Petri net models of complex hardware and software systems is usually difficult and error prone. The nice visual representation typical of Petri Nets is not really helpful for complex nets. Apart from the size of the net itself, large numbers of synchronisations between distinct parts typically obstruct the design process.

In Chapter 5 we have introduced process algebra as a *compositional* specification formalism. In process algebra complex models are composed in a stepwise fashion out of smaller building blocks. In this chapter we introduce means to incorporate compositionality into SPN, exploiting results established in the framework of Stochastic Process Algebra (SPA). The question arises whether it is possible to exploit hierarchies of such SPN or SPA models to ease their solution, i.e. performance analysis. Indeed there is a positive answer to this question. A solid framework for exploiting hierarchies relies on *equivalence* relations that are *congruences* with respect to the composition operators, such as strong equivalence, introduced in Chapter 5. The congruence property, also called *substitutivity*, implies that the behaviour of nets that equivalently behaving nets are interchangeable when composed with other nets within a complex model. For instance, strong equivalence induces a strongly lumpable partition on the underlying Markov Chain, as we will highlight later in this chapter. We can therefore use this relation to aggregate the reachability graph. Moreover, since it is a congruence with respect to parallel composition, it can equally be applied to components of a hierarchical model in order to reduce their reachability graphs. This implies that the size of the state space of the hierarchical model is reduced as well, but with the additional gain that the original state space of the hierarchical model does not need to be constructed. In the context of process algebras, this general technique is known as *compositional aggregation* and has

successfully been applied to a variety of non-stochastic concurrent systems in order to perform verification of functional properties. The only requirement is that the equivalence is a congruence with respect to composition.

In this chapter, we address the issue of composition and compositional aggregation in the framework of SPA, and of (G)SPN. In particular we introduce two composition operators, hiding and parallel composition, that are borrowed from Chapter 5 as the basic ingredients of *compositional SPN*. Extending the concept of strong equivalence we develop the notion of weak Markovian bisimulation for theses nets. Since this relation is indeed a congruence we are able to show how it supports compositional construction and aggregation in the framework of Petri Nets. In particular, we address algorithmic issues related to compositional aggregation. As a result, it is possible to exploit hierarchies in the model definition on the level of performance analysis.

16.2 Compositionality and Abstraction

In order to recall the basic constructs of a process algebraic specification, as introduced in Chapter 5, we consider a simple queueing system. We start with an untimed system, in order to point out some crucial aspects of compositionality. Our example consists of the arrival process *Arrival*, a queue with finite capacity, and a *Server*. First, we model an arrival process as an infinite sequence of incoming arrivals (*arrive*), each followed by an enqueue action (*enq*).

$$Arrival \quad := \quad arrive.enq.Arrival$$

The behaviour of a finite queue can be described by a family of processes, one for each value of the current queue population. Depending on the population, the queue may permit to enqueue a job (*enq*), dequeue a job (*deq*) or both. The latter possibility is described by the *choice* operator + between two alternatives.

$$\begin{aligned} Queue_0 &:= enq.Queue_1 \\ Queue_i &:= enq.Queue_{i+1} + deq.Queue_{i-1} \qquad 1 \leq i < max \\ Queue_{max} &:= deq.Queue_{max-1} \end{aligned}$$

Next, we need to define a server process, as follows:

$$Server \quad := \quad deq.serve.Server$$

These separate processes can now be combined by the *parallel composition* operator $\|_{\{..\}}$ in order to describe the whole queueing system. This operator is parametrised with a set of actions on which the partners are required to synchronise:

$$System \quad := \quad Arrival \quad \|_{\{enq\}} \quad Queue_0 \quad \|_{\{deq\}} \quad Server$$

The formal semantics introduced in Chapter 5 associates each language expression with an unambiguous interpretation represented in terms of a variant

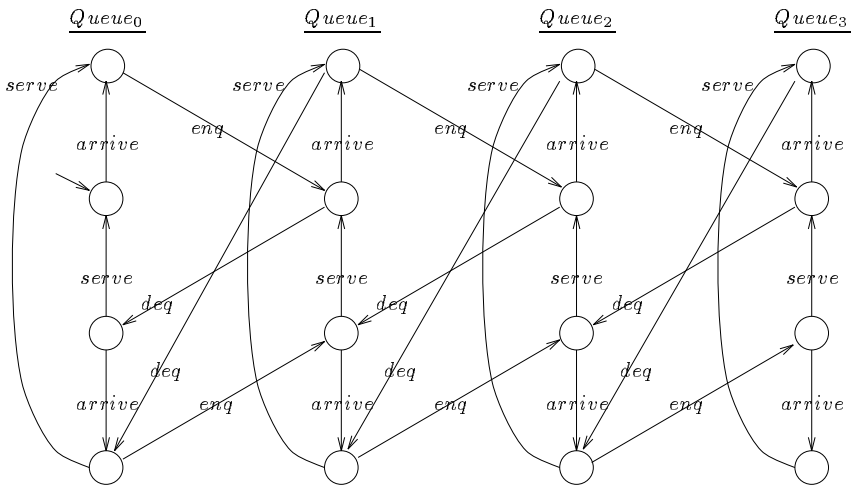


Figure 16.1: Transition diagram of the queueing system example

of the well known state transition diagrams. Fig. 16.1 shows the transition diagram for our example queueing system, under the assumption that the maximal population of the queue is three ($max = 3$). There are 16 states, the initial state being indicated by an arrow. A transition between two states is represented by an arrow and is labelled with an action which occurs when the system changes from one state to another. Note that the arrival of a customer and its enqueueing into the queue are separate actions, so that one more arrival is possible if the queue is already full. Further arrivals will be discarded, as long as the queue is full. Likewise, dequeueing and serving are modelled as separate actions, such that at the moment the queue becomes empty, the server is still serving the last customer.

In Chapter 5, the concept of hiding has been introduced, as a means to abstract from internal details that are not relevant for the chosen level of abstraction. To illustrate the usefulness of this idea, we assume that we are not interested in the internal details of interaction between *Arrival* and *Queue*, respectively *Queue* and *Server*, we may wish to only observe actions *arrive* and *serve*. This requires *abstraction* from internal details, and is achieved by employing the *hiding* operator /:

$$System/\{enq,\ deq\}$$

As a result, actions *enq* and *deq* are now internal actions, i.e. they are not visible from the environment. According to the operational semantics of hiding, actions hidden from the environment become the distinguished internal action τ. In other words, the transition diagram of the above expression is obtained by turning all *enq* or *deq* labels appearing in Fig. 16.1 into τ. The result is depicted Fig. 16.2.

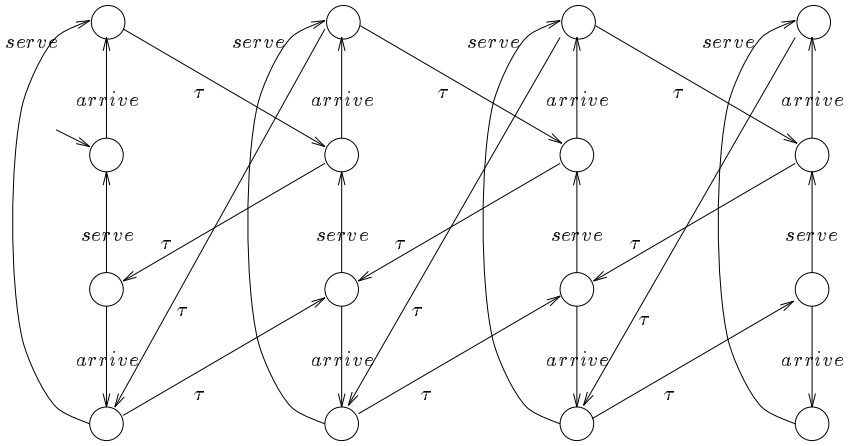


Figure 16.2: Transition diagram after hiding internal details

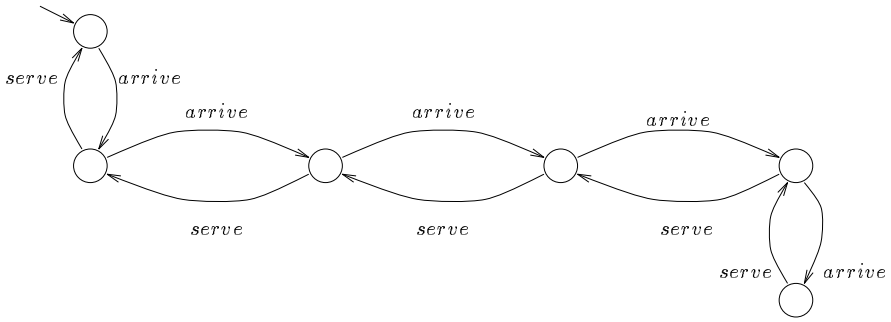


Figure 16.3: A weakly bisimilar transition diagram

Even though these details are now internalised, they still appear in the transition diagram. In Chapter 5, we have learned that equivalence notions are *the* process algebraic means to compare specifications, with respect to the behaviour that an external observer is able to witness. Clearly, an observer should not be able to witness internal details, such as τ-labelled moves between states. This raises the question whether we can find an equivalent transition diagram, where these details are completely absent instead of being (internally) present. For the above example, this is indeed possible: The transition diagram in Fig. 16.2 is equivalent to the diagram depicted in Fig. 16.3, under a specific notion of equivalence, *weak* bisimulation.

The notion of weak bisimulation is an important one in the context of this chapter. It is based on the notion of strong bisimulation, introduced in Chapter 5, Definition 5.1. We recall the definition here for convenience.

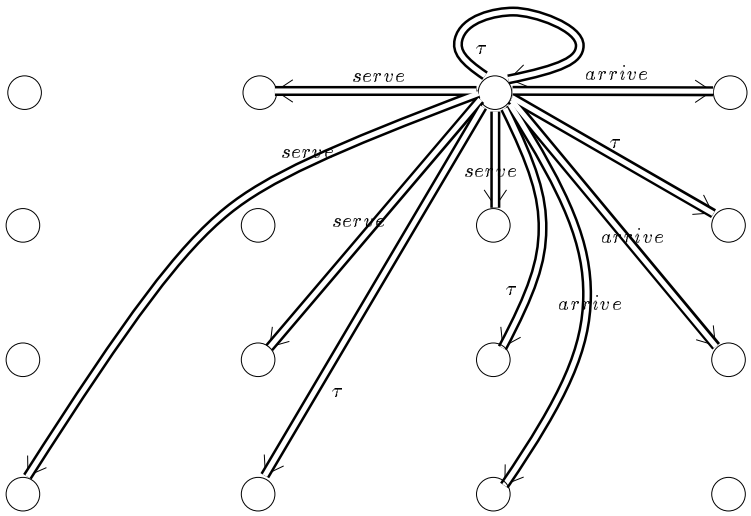


Figure 16.4: Weak transitions

Definition 16.1 *A binary relation $\mathcal{R}$ over agents is a* **strong bisimulation** *if $(P, Q) \in \mathcal{R}$ implies, for all $a \in \mathcal{A}ct \cup \tau$:*

1. *Whenever $P \xrightarrow{a} P'$, then for some Q', $Q \xrightarrow{a} Q'$ and $(P', Q') \in \mathcal{R}$;*
2. *Whenever $Q \xrightarrow{a} Q'$, then for some P', $P \xrightarrow{a} P'$ and $(P', Q') \in \mathcal{R}$.*

The definition of weak bisimulation requires just a slight variation of the above definition. In order to abstract from internal moves, it appears natural to neglect them as far as they do not influence the future behaviour of a process. For this purpose, one introduces a notion of observable steps of a process, that consist of a single *external* action preceded and followed by an arbitrary number (including zero) of internal steps [22]. Technically this is achieved by deriving a 'weak' transition relation $\Longrightarrow$ from the 'strong' transition relation $\longrightarrow$.

Definition 16.2 *For internal actions, $\overset{\tau}{\Longrightarrow}$ is defined as the reflexive and transitive closure $\overset{\tau}{\longrightarrow}^*$ of the relation $\overset{\tau}{\longrightarrow}$ of internal transitions. External weak transitions are then obtained by defining $\overset{a}{\Longrightarrow}$ to denote $\overset{\tau}{\Longrightarrow} \overset{a}{\longrightarrow} \overset{\tau}{\Longrightarrow}$.*

Note that a weak internal transition $\overset{\tau}{\Longrightarrow}$ is possible without actually performing an internal action, because $\overset{\tau}{\longrightarrow}^*$ contains the reflexive closure, i.e. the possibility of not moving at all. In contrast, a weak external transition $\overset{a}{\Longrightarrow}$ must contain exactly one transition $\overset{a}{\longrightarrow}$ preceded and followed by arbitrary (possibly empty) sequences of internal moves. To illustrate the definition we have depicted in Figure 16.4 all weak transitions that emanate from a specific state of the transition diagram underlying the queueing system example.

With the notion of a weak transition, weak bisimulation arises from the above definition of strong bisimulation by simply substituting $\Longrightarrow$ for $\longrightarrow$:

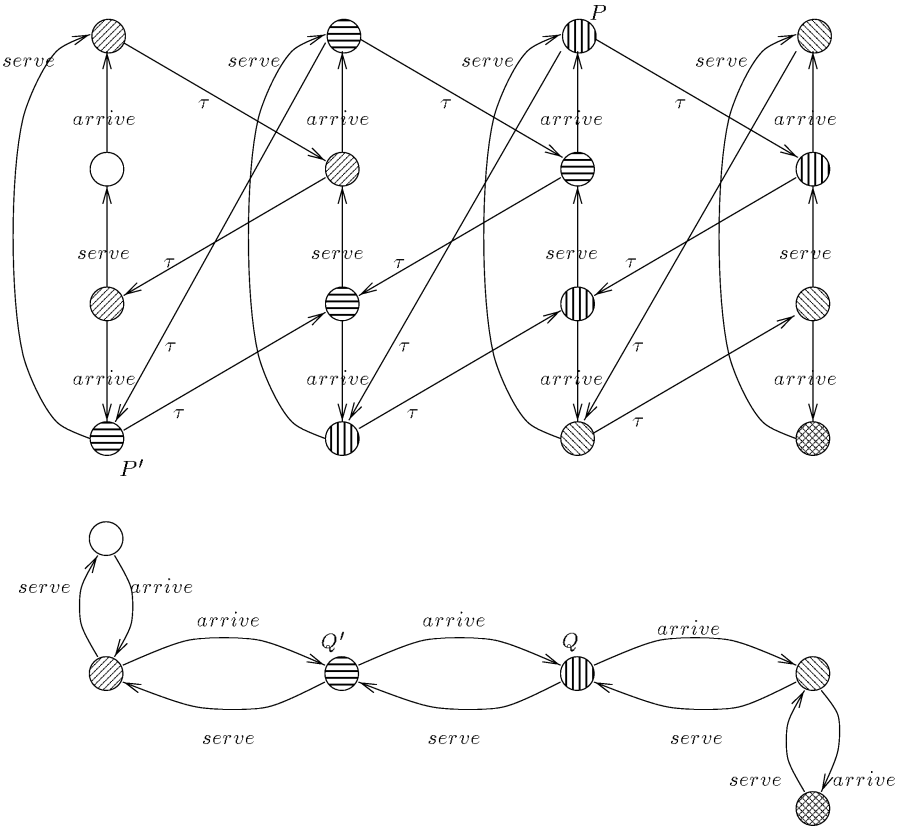


Figure 16.5: A weak bisimulation

Definition 16.3 *A binary relation $\mathcal{R}$ is over agents a* **weak bisimulation** *if $(P,Q) \in \mathcal{R}$ implies, for all $a \in \mathcal{A}ct \cup \tau$:*

1. *Whenever $P \stackrel{a}{\Longrightarrow} P'$, then for some Q', $Q \stackrel{a}{\Longrightarrow} Q'$ and $(P',Q') \in \mathcal{R}$;*

2. *Whenever $Q \stackrel{a}{\Longrightarrow} Q'$, then for some P', $P \stackrel{a}{\Longrightarrow} P'$ and $(P',Q') \in \mathcal{R}$.*

Any relation which satisfies Definition 16.3 is a weak bisimulation. The notion of *weak bisimilarity* ($\approx$) then arises as the largest weak bisimulation. In other words, $P \approx Q$ holds, if there is some weak bisimulation containing the pair (P,Q). As discussed in Chapter 5, strong and weak bisimilarity are congruences with respect to the language operators except the choice operator. A slight refinement of weak bisimilarity is needed to ensure congruence for this operator, as well.

In order to illustrate the definition, and to verify the above claim that the transition diagrams of Fig. 16.2 and Fig. 16.3 are indeed weakly bisimilar, we have depicted in Fig. 16.5 a relation $\mathcal{R}$, using a particular shading of states. Two states in this figure are contained in $\mathcal{R}$ whenever they are shaded with the same pattern. In fact, this determines that $\mathcal{R}$ is an equivalence relation on the states in Fig. 16.5. It is a tedious exercise to verify that each pair of $\mathcal{R}$ satisfies the above definition. As an example we have to verify that the weak transition $P \stackrel{serve}{\Longrightarrow} P'$ consisting of $P \stackrel{\tau}{\longrightarrow} \stackrel{serve}{\longrightarrow} \stackrel{\tau}{\longrightarrow} P'$ can indeed be matched by some $Q \stackrel{serve}{\Longrightarrow} Q'$, namely $Q \stackrel{serve}{\longrightarrow} Q'$.

This tedious case analysis can be checked mechanically in a rather efficient way. The general algorithmic strategy used for this purpose is of some importance for this chapter. Therefore we will focus on the algorithmic issues related to the computation of (strong and weak) bisimilarity. We exemplify a strategy that efficiently factorises a transition diagram into equivalence classes of bisimilar states. The fact that the algorithm returns equivalence classes of states has an interesting side effect. By essentially representing each class by a single state, it is possible to construct an *aggregated* transition diagram that is equivalent to the original one. The transition diagram depicted in Fig. 16.2 is indeed such an aggregated transition diagram. It may be surprising that this diagram has 6 and not 4 states (we assumed $max = 3$). This is due to the fact that the arrival of a customer and its enqueueing into the queue are separate actions, so that one more arrival is possible if the queue is already full. Likewise, dequeueing and serving are modelled as separate actions, such that at the moment the queue becomes empty, the server is still serving the last customer.

In practice, this 'equivalence preserving state space aggregation' is highly favourable for analysing models of real systems. Such models, consisting of a hierarchy of interacting components, tend to have very large state spaces that are often even too large to be stored in memory. For such systems, any kind of analysis already fails because the state space to be analysed remains unknown. However, the *congruence property* allows one to replace arbitrary interacting components by their aggregated representatives, since the above aggregation is bisimilarity preserving. This implies that the size of the state space of the hierarchical system is reduced as well, but with the additional gain that it avoids the construction of the original state space which would be prohibitively large. This general technique is known as *compositional aggregation* and has successfully been applied to a variety of very large non-stochastic systems in order to verify behavioural properties. An example of compositional aggregation has also appeared in Chapter 5, using the PLC example.

16.3 Algorithmic computation

The general algorithmic strategy to compute a bisimulation style relation is called *partition refinement*. We exemplify this strategy by means of strong bisimilarity; weak bisimilarity only requires some slight modifications. A partitioning of a transition diagram is a set of disjoint subsets (partitions) of the state

space, such that each state is contained in (exactly) one subset. The algorithm should obviously terminate once the partitions correspond to the equivalence classes of bisimilarity. This is the case indeed, and it is achieved by successive refinement of the partitions. Starting with a partitioning consisting of a single (rather large) partition, the partitions become finer and finer until no further refinement is needed, or, in algebraic terms, a fixed-point is reached. This fixed-point is the desired result.

Before we address the question how partitions are successively refined we characterise strong bisimilarity as a fixed-point of successively finer relations. For this purpose, we define a boolean function γ as follows:

$$\gamma(P \xrightarrow{a} C) := \begin{cases} \texttt{true} & \text{if there is } P' \in C \text{ such that } P \xrightarrow{a} P', \\ \texttt{false} & \text{otherwise.} \end{cases}$$

This function is similar to the probability measure $\nu(P \xrightarrow{a} C)$ (and also to the total conditional transition rate function $q[P, C, \alpha]$) introduced in Chapter 5, but instead of cumulating probabilities on transitions, it simply talks about the presence, or absence, of a transition from a state into set of states.

Proposition 16.4 *Let P be an agent with finite (reachable) state space S.*

Strong bisimilarity on S is the unique fixed-point of the following family of relations

- $\backsim_0 = S \times S$,
- $\backsim_{k+1} = \Big\{(P, Q) \in \backsim_k \mid$ *for all* $a \in \mathcal{A}ct \cup \{\tau\}$ *and all equivalence classes* C *induced by* $\backsim_k$, $\gamma(P \xrightarrow{a} C) = \gamma(Q, \xrightarrow{a} C)\Big\}$.

Milner [22] has shown that this property holds even for infinite state spaces. However, it has to be clarified whether each $\backsim_k$ is indeed an equivalence relation, because otherwise $\backsim_{k+1}$ would not be well-defined. A simple induction shows this: First, $\backsim_0$ is a trivial equivalence. Second, assuming that $\backsim_k$ is an equivalence, $\backsim_{k+1}$ also is, essentially because '=', equality on boolean values, is an equivalence relation.

The refinement step from $\backsim_k$ to $\backsim_{k+1}$ rules out those pairs (P', Q') from $\backsim_k$ for which γ produces (at least one) different truth value. In other words P' and Q' are split into different classes (of $\backsim_{k+1}$), if one of them is able to move into a class C (of $\backsim_k$) by performing an action a while the other one is not. So, (a, C) is a specific reason why (P', Q') has to be split. A pair (a, C) is therefore called a *splitter*.

In general, a splitter (a, C) is a means to refine all partitions for which $\gamma(\_ \xrightarrow{a} C)$ returns different truth values into two finer partitions such that all states with the same truth value are contained in the same finer partition. Formally the procedure of refining a partitioning *Part* by means of a splitter (a, C) is as follows:

Input: labelled transition system $(S, \mathcal{A}ct, \longrightarrow)$
Output: $S/\sim$
Method: $Part := \{S\}$;
$Spl := \mathcal{A}ct \times \{S\}$;
While Spl not empty **do**
  Choose (a, C) **in** Spl;
  $Old := Part$;
  $Part := Refine(Part, a, C)$;
  $New := Part - Old$;
  $Spl := (Spl - \{(a, C)\}) \cup (\mathcal{A}ct \times New)$;
od
Return $Part$.

Table 16.1: Algorithm for computing strong bisimulation equivalence classes.

$$Refine(Part, a, C) := \left(\bigcup_{X \in Part} \left(\bigcup_{v \in \{\texttt{true},\texttt{false}\}} \left\{ \{ P \in X \mid \gamma(P \xrightarrow{a} C) = v \} \right\} \right) \right) - \{\emptyset\}$$

Each partition X is refined into two partitions, one for each possible value of γ. Since some of these partitions may be empty, the empty set is eventually extracted. This refinement step is the core of our global algorithm depicted in Table 16.1. We initialise this algorithm with a partitioning $\{S\}$ (a single partition) and a set of possible splitters Spl, containing (a, S) for each action $a \in \mathcal{A}ct$. In the body of the algorithm each of the splitters is iteratively applied to refine the current partitioning. After each refinement, New contains all new (and thus finer) partitions. Each of them gives rise to new splitters, one for each action $a \in \mathcal{A}ct$. All these splitters are added to Spl while the currently processed splitter is removed.

We illustrate the basic algorithm by means of an example. We aim to compute the equivalence classes with respect to strong bisimilarity of the transition diagram depicted in Figure 16.1. Note that we are computing *strong* bisimilarity. Hence one should not expect that we obtain the result depicted in Figure 16.4 which corresponds to *weak* bisimilarity.

As before, we use shading of states to distinguish states with different properties, represented in Figure 16.6. In the beginning all states are assumed to be equivalent, and hence, all states are shaded with the same pattern. We use  to refer to the set of states shaded like  .

So, we initialise the algorithm with $Part := \{$  $\}$, a single partition containing all states, and with three splitters, $(arrive,$  $)$, $(serve,$  $)$, and $(\tau,$  $)$. This situation is represented in Figure 16.6.

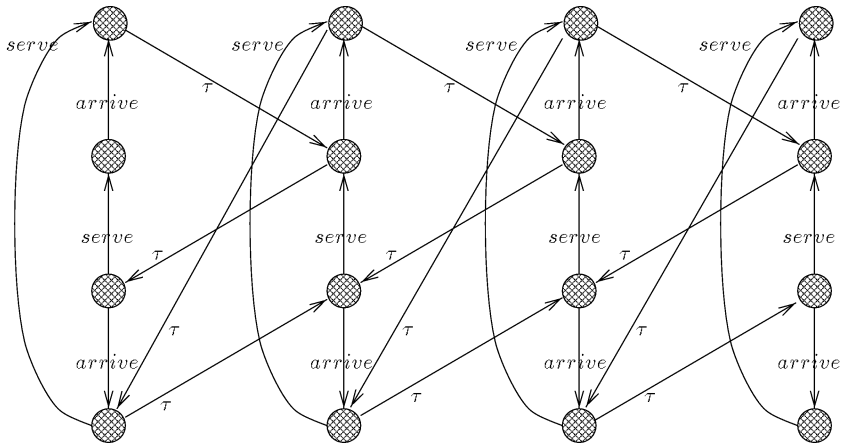


Figure 16.6: Initialisation of the algorithm

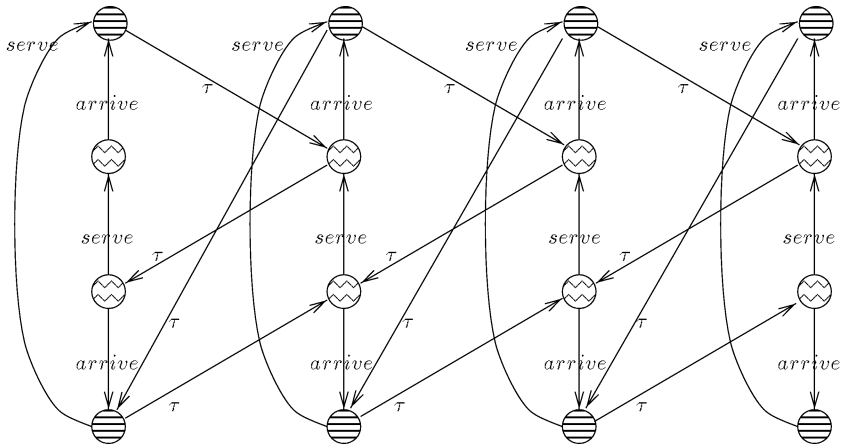


Figure 16.7: Result of the first refinement step

We start partition refinement by choosing a splitter, say $(arrive, \blacksquare)$ and computing the values $\gamma(\_, arrive, \blacksquare)$ for all the states in $\blacksquare$. One half of the states return `false` while the others return `true`. As a result, we have to refine partition $\blacksquare$ into two partitions, thus $Part := \{ \blacksquare , \blacksquare \}$. Adding new splitters to Spl (and removing splitter $(arrive, \blacksquare)$) leads to the situation depicted in Figure 16.7.

Continuing with a different splitter, say $(serve, \blacksquare)$ returns different truth values of γ in partition $\blacksquare$, and $\blacksquare$. Therefore, we split these partitions into $\blacksquare$ and $\blacksquare$, respectively $\blacksquare$ and $\blacksquare$. Both, $Part$ and New become

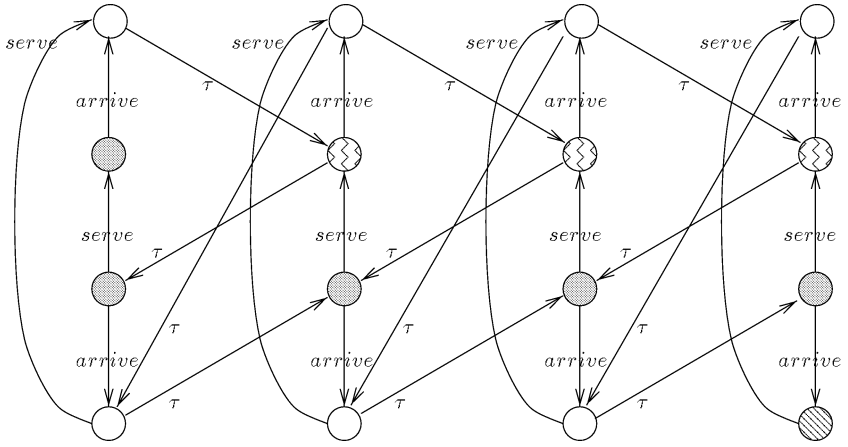


Figure 16.8: Result of the second refinement step

{}. We update *Spl* accordingly. This leads to the situation depicted in Figure 16.8.

The next refinement steps lead to finer and finer partitions, until eventually the set of splitters is empty. The algorithm terminates with a partitioning where every partition just consists of a single state.

As illustrated by this example, the algorithm of Table 16.1 works fine, but its complexity is relatively high. In particular, the administration of splitters can be improved a lot. For instance, when updating *Spl* it is plausible to remove unprocessed splitters (a, C) from *Spl*, if C has been refined during the last step (i.e. if $C \in (Old - Part)$) because finer splitters will be processed anyway. Further improvements are possible by essentially adopting the deliberate management of splitters in the style of [19, 23, 18, 5]. As a result, the algorithm can be shown to have a time complexity of order $\mathcal{O}(m \log n)$, where n is the number of states and m is the number of transitions [23].

The same algorithm can be used to compute weak bisimilarity. For this purpose, $\gamma(P \stackrel{a}{\Longrightarrow} C)$ replaces $\gamma(P \stackrel{a}{\longrightarrow} C)$ in function *Refine*. However, this requires the computation of $\stackrel{a}{\Longrightarrow}$ during the initialisation phase. As a matter of fact, the computation of $\Longrightarrow$ dominates the complexity of partition refinement, basically because the reflexive and transitive closure $\stackrel{\tau}{\longrightarrow}^*$ of internal moves has to be computed in order to build the weak transition relation. The usual way of computing a transitive closure has cubic complexity. Some improvements are known for this task, see, for instance, [8] for an algorithm with $\mathcal{O}(n^{2.376})$ time requirements. In any case, this is the computationally expensive part.

16.4 Compositional Nets

The preceding section has been devoted to the concepts of compositionality and abstraction, in a process algebraic setting. In this section we adopt these ideas to the setting of Stochastic Petri nets. We introduce *compositional stochastic Petri nets* (CSPN), and define composition operators on them. While introducing them formally, we illustrate the concept by developing a compositional queueing system (essentially the one from the previous sections) with CSPN. The components are depicted in Figure 16.9, an arrival stream, a buffer (with three places), and a server. To understand this example, and to isolate the difference to a usual (G)SPN, some peculiarities have to be addressed.

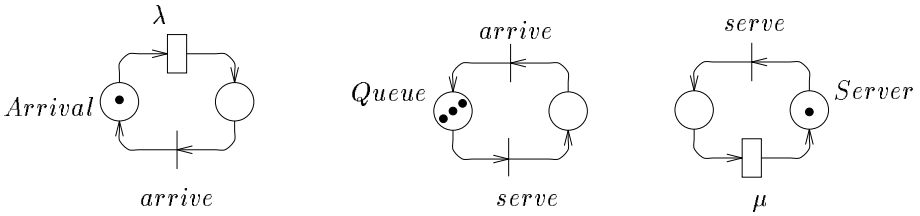


Figure 16.9: Three simple compositional nets

Similar to GSPNs, a CSPN may contain immediate transitions and timed transitions. Both types of transitions are labelled with additional information. Timed transitions are labelled with rates, drawn from the set of positive reals, and describing a negative exponentially distributed delay, as usual for SPN and GSPN. Immediate transitions, on the other hand, are labelled with *names*, taken from a set *Sync*. These names play the role of actions in the process algebraic sense. When composed in parallel, immediate transitions with the same name have to happen synchronously if their names appear in a list of (actually) synchronising names $s_1, \ldots, s_n$. Internal immediate transitions named τ will happen in total independence of the environment. Before discussing composition operators for CSPN, we give the formal definition of CSPN.

Definition 16.5 *A compositional SPN is a 7-tuple* $(P, M, I, F, \Lambda, L, q)$, *where*

- P *is a non empty set of places,*
- M *is a set of Markovian transitions,*
- I *is a set of immediate transitions,*
- $F \subseteq \big(P \times (M \cup I)\big) \cup \big((M \cup I) \times P\big)$ *is the flow relation.*
- $\Lambda : M \mapsto]0, \infty[$, *where* $\Lambda(m)$ *is the rate of the exponential probability distribution associated with the Markovian transition* m,
- $L : I \mapsto Sync \cup \{\tau\}$, *where* $L(i)$ *is the name of the immediate transition* i.

- $q : P \mapsto I\!N$ *is the initial marking.*

Parallel composition and abstraction of CSPN uses a slightly different notation than the one we have seen before (just for visualisation convenience). In particular, $\overline{\overline{\;s_1,\ldots,s_n\;}}$ denotes parallel composition via a list of names $s_1, \ldots, s_n$, and $\boxed{N \;\; s_1,\ldots,s_n}$ denotes hiding of the names $s_1, \ldots, s_n$ in the net N. For instance,

$$Arrival \quad \overline{\overline{enq}} \quad Queue_0 \quad \overline{\overline{deq}} \quad Server$$

corresponds to the composite specification of the queueing system, and

$$\boxed{Arrival \quad \overline{\overline{enq}} \quad Queue_0 \quad \overline{\overline{deq}} \quad Server \quad \text{enq,deq}}$$

denotes the same system, but now the detailed interaction on *enq* and *deq* is hidden. The components of the above expressions are the CSPN depicted in Figure 16.9. This raises the question about the formal meaning of such expressions that are built up from CSPNs and composition operators. Indeed each such expression defines a CSPN, a composite net that is determined by the CSPN components and by the semantics of the respective operator. As an example, Figure 16.10 contains the CSPN obtained for both of the above expressions.

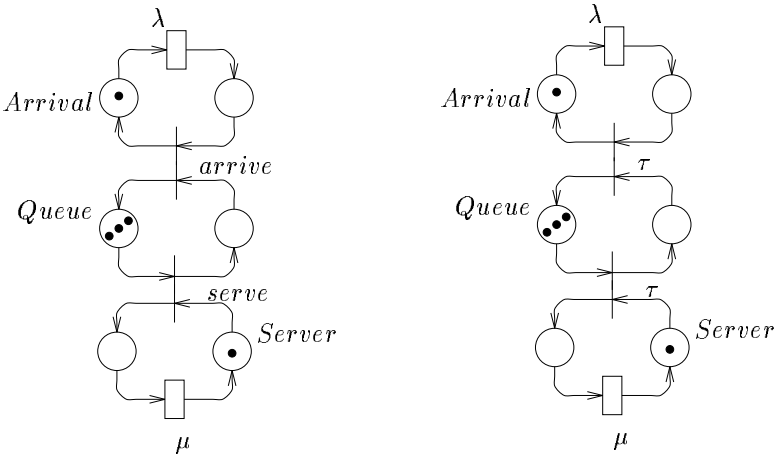


Figure 16.10: Composite SPN of the queueing system example with (right) and without (left) hiding.

As illustrated by the above example, parallel composition of CSPN is an operation that superposes immediate transitions with the same label, if the label occurs in the synchronisation set. Hiding changes names (those listed in the operator) into the distinguished label τ. The operators are formally defined as follows:

Definition 16.6 *Let* $N = (P, M, I, F, \Lambda, L, q)$ *be a CSPN. Hiding of names* $h_1, \ldots, h_n$, *denoted* $N \setminus \{h_1, \ldots, h_n\}$, *results in the composite SPN* $(P, M, I, F, \Lambda, L', q)$, *where* $L'(i) = L(i)$ *for all* $i \notin \{h_1, \ldots, h_n\}$, *and* $L'(i) = \tau$ *for all* $i \in \{h_1, \ldots, h_n\}$.

Definition 16.7 *Let* $N_1 = (P_1, M_1, I_1, F_1, \Lambda_1, L_1, q_1)$ *and* $N_2 = (P_2, M_2, I_2, F_2, \Lambda_2, L_2, q_2)$ *be two CSPN with* $P_1 \cap P_2 = \emptyset$, $M_1 \cap M_2 = \emptyset$ *and* $I_1 \cap I_2 = \emptyset$. *Let* $S = \{s_1, \ldots, s_n\} \subseteq Sync$ *be a set of names and* $\tilde{I}_1 = \{i_1 \in I_1 \mid L_1(i_1) \in S\}$, $\tilde{I}_2 = \{i_2 \in I_2 \mid L_2(i_2) \in S\}$ *and* $\tilde{I} = \{(i_1, i_2) \in (\tilde{I}_1 \times \tilde{I}_2) \mid L_1(i_1) = L_2(i_2)\}$.

The parallel composition of N_1 *and* N_2 *via a list of names* $s_1, \ldots, s_n$, *denoted* $N_1 \overline{\overline{\ s_1, \ldots, s_n\ }} N_2$, *is the composite SPN* $(P, M, I, F, \Lambda, L, q)$, *where*

- $P = P_1 \cup P_2 \quad and \quad M = M_1 \cup M_2$,
- $I = (I_1 - \tilde{I}_1) \cup (I_2 - \tilde{I}_2) \cup \tilde{I}$,
- $F = \begin{array}{l} F_1 \cap (P_1 \times M_1) \cup \\ F_2 \cap (P_2 \times M_2) \cup \\ F_1 \cap (P_1 \times (I_1 - \tilde{I}_1)) \cup \\ F_2 \cap (P_2 \times (I_2 - \tilde{I}_2)) \cup \\ \{(p, (i_1, i_2)) \in P \times \tilde{I} \mid p \in {}^\bullet i_1 \vee p \in {}^\bullet i_2\} \cup \\ \{((i_1, i_2), p) \in \tilde{I} \times P \mid p \in i_1^\bullet \vee p \in i_2^\bullet\} \end{array}$
- $\Lambda(m) = \begin{cases} \Lambda_1(m) & if \ m \in M_1, \\ \Lambda_2(m) & if \ m \in M_2, \end{cases}$
- $q(p) = \begin{cases} q_1(p) & if \ p \in P_1, \\ q_2(p) & if \ p \in P_2, \end{cases}$
- $L(i) = \begin{cases} L_1(i) & if \ i \in I_1 - \tilde{I}_1, \\ L_2(i) & if \ i \in I_2 - \tilde{I}_2, \\ L_1(i_1) & if \ i = (i_1, i_2) \in \tilde{I}. \end{cases}$

According to these definitions, CSPN are closed under parallel composition and hiding. It is worth remarking that the effect of parallel composition is, in general, more involved than the simple example in Figure 16.10 might suggest. If multiple immediate transitions of a component are labelled with the same name (also appearing in the synchronisation set), then Definition 16.7 requires the introduction of 'glued' transition for any possible combination of such transitions with (equally labelled) transitions of the partner component. This fact will become obvious in the example developed in the next section.

The reachability graph of a CSPN is defined by means of the token game in complete analogy to an ordinary Petri net. We omit the definition which can be found in Chapter 2. Note however, that we do not impose a priority structure on transitions, in contrast to the situation in GSPN. In particular, timed transitions may fire even if immediate transitions are concurrently enabled. The reason is

that an immediate transition may be subject to further synchronisation (if the labelling name appears in a composition operator applied to the CSPN), and this synchronisation may impose further timing constraints on an immediate transition. The most extreme case is that a synchronisation is forced, but no 'partner' transition exists which is labelled with the same name. This will postpone the firing of the immediate transition forever, and therefore the above semantics of parallel composition directly removes such immediate transitions.

So, the reachability graph describes the complete potential behaviour of a CSPN, taking into account that this CSPN may appear as a component of a larger CSPN. The reachability graphs of the two queueing system examples of Figure 16.10 have already appeared. They are isomorphic to the transition diagrams in Figure 16.1 and Figure 16.2, respectively.

16.5 The PLC multicomputer example

In order to illustrate how a nontrivial CSPN arises by construction out of smaller components, we will describe the simple multicomputer PLC example discussed already in various other chapters of this book. Indeed, our specification builds on the process algebraic one developed in Chapter 5.

As in the example discussed in Section 5.2.3, let us suppose the system is formed by two computers which synchronise to start a control cycle. This system is modelled by two instances of the entity *Comp* which synchronise on the action *start_cycle*, representing the beginning of the control cycle. Moreover, both computers need the bus in order to read external data. This behaviour is achieved by adding one instance of the *Bus* component with an appropriate

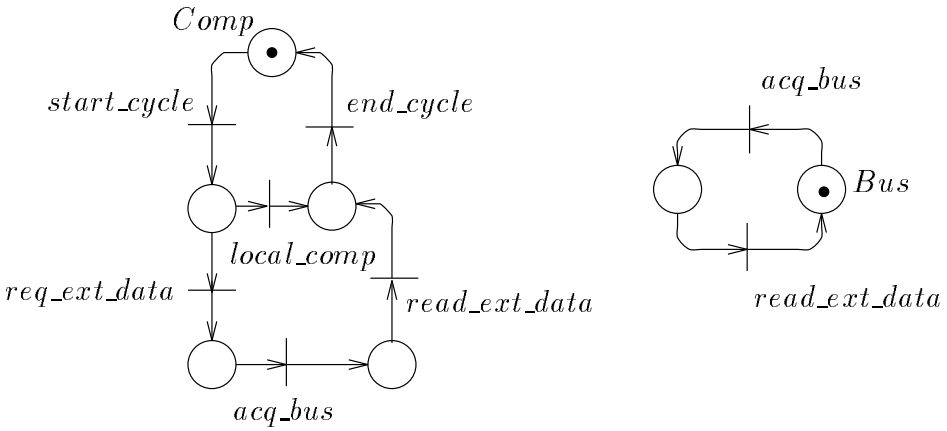


Figure 16.11: Component SPNs of the PLC example

synchronisation set. The overall specification is

$$\left(\left(Comp \underset{start\_cycle}{||} Comp \right) \Big/ _{start\_cycle} \underset{acq\_bus, read\_ext\_data}{||} Bus \right) \Big/ _{acq\_bus, read\_ext\_data}.$$

In this term, we have used hiding to internalise the start of a cycle. Similarly, the activities that take place on the bus are local to the system, and hence internalised. The component CSPNs of the computer, *Comp*, and of the bus, *Bus*, are depicted in Figure 16.11. Note that there is a strong similarity to the transition diagrams depicted in Figure 5.4..

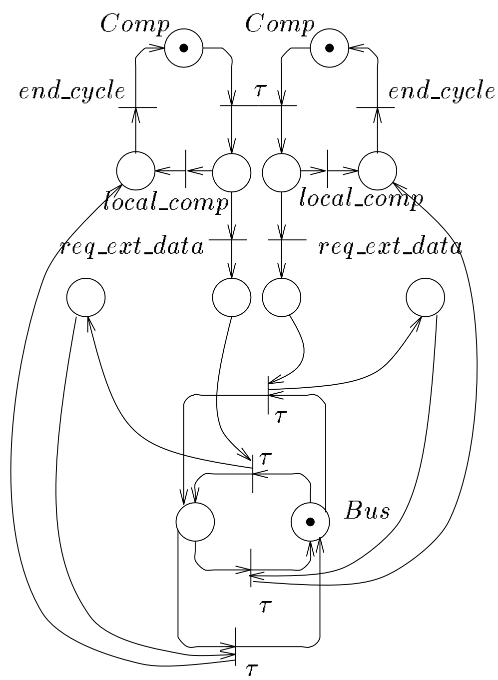


Figure 16.12: Composite SPN of the PLC example

Applying the definitions of hiding and parallel composition, we obtain the composite net depicted in Figure 16.12. If we now want to incorporate some Markovian delays into the model, we may, just as an example, decide to change the CSPN *Comp* such that *local_comp* describes an exponentially distributed duration given by rate μ, and that *end_cycle* is given by some rate λ. This new component is depicted in Figure 16.13.

Indeed, we prefer to work with this variation of the example in the sequel, for didactical reasons. If we plug in this CSPN instead of the original one into the complete PLC example, the shape of the composite SPN changes, of course. In particular, the immediate transitions labelled *local_comp* and *end_cycle* in the

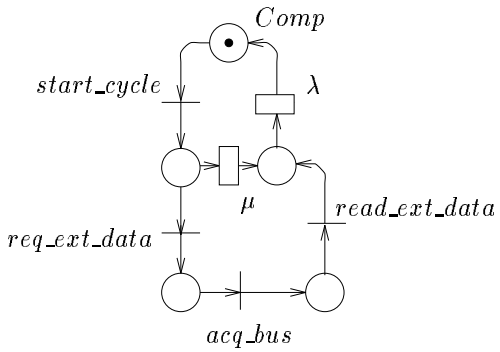


Figure 16.13: A component involving timed transitions

composite SPN of Figure 16.12 are replaced by timed transitions labelled with rate μ and λ, respectively (cf. Figure 16.14). The reachability graph underlying this net is depicted in Figure 16.15, using the same notation as in the previous sections of this chapter.

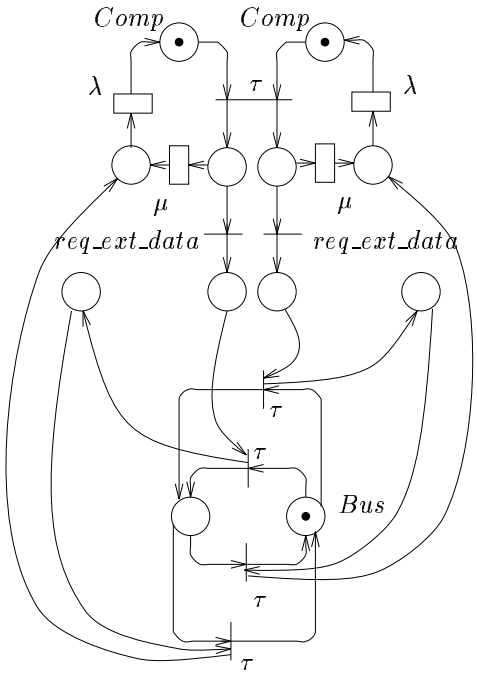


Figure 16.14: Composite SPN of the modified PLC example

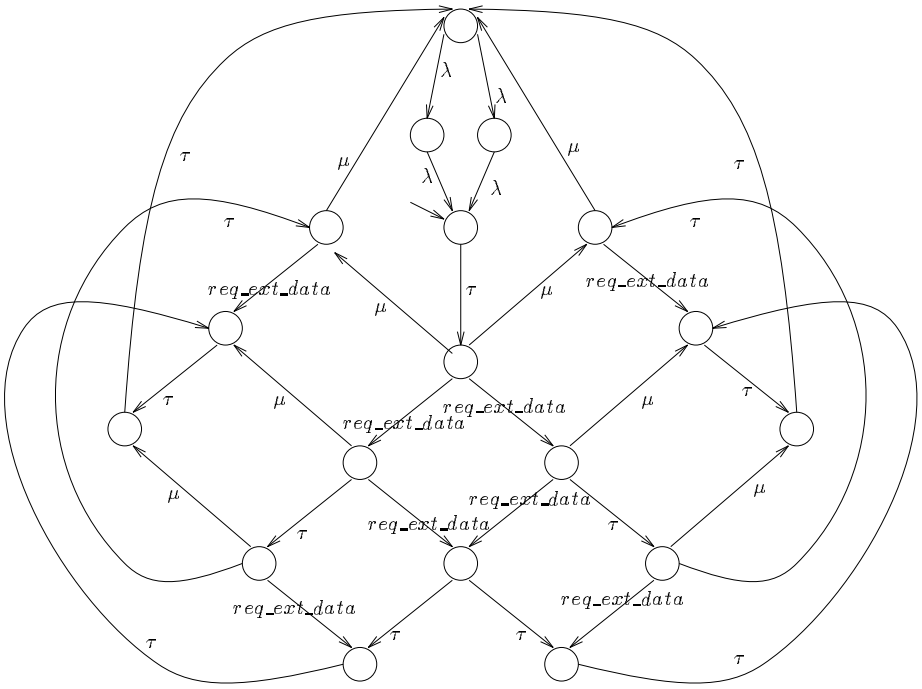


Figure 16.15: Reachability graph of the CSPN describing the (modified) PLC example

The graph contains various markings where both timed and immediate transitions firings are possible. One may now ask how the decision between these alternatives is taken. In fact, the decision is taken in favour of the fastest, i.e. the first alternative that is able to evolve further, either because some delay has elapsed, or because some immediate transition can occur. This is known as the race condition. If this fastest process is not uniquely determined, a *non-deterministic* selection among the fastest processes is made. Note that, since immediate transition possibly depend on further synchronisation, it is not the case that they in general may fire with zero delay (as their name might suggest). *Internal* immediate transitions form an exception, because they are internal to components, and can hence not be subject to further synchronisation. As a consequence, internal immediate transitions fire with zero delay, and hence we will not observe a concurrently enabled timed transition fire first. However, this fact is *not* incorporated into the reachability graph. In the next section we will introduce the notion of an *observable reachability graph* which, among other things, factors out such timed transitions.

16.6 Equivalence Notions

In this section we adopt the equivalences of Chapter 5 defined in the context of Stochastic Process Algebra to compositional SPN. We first define an equivalence notion for purely Stochastic Petri Nets, Markovian bisimilarity, along the lines of the notion of strong equivalence in the style of Chapter 5. This relation is the starting point for *weak Markovian bisimilarity*, a relation introduced to compare CSPN. We discuss the notion of an *observable reachability graph* that appears as the canonical representation of the reachability graph of an equivalence class of CSPNs.

16.6.1 Strong equivalence and lumpability

First, we define a function that computes the apparent rate, the sum of rates of all firings of timed transitions enabled in a marking q', and leading to any marking in a given set. It is a variant of the total conditional transition rate function $q[P, B, \alpha]$ that has appeared in the context of strong equivalence.

Definition 16.8 *For $q' \in P$ and $B \subseteq P$ we define*

$$\gamma(q', B) = \sum_{q' \xrightarrow{m} q'', q'' \in B} \Lambda(m).$$

Definition 16.9 *Let $N = (P, M, \emptyset, F, \Lambda, L, q)$ be a SPN. A* **Markovian bisimulation** *$\mathcal{R}$, is an equivalence relation over RS_N such that whenever $(q', q'') \in \mathcal{R}$ then for all $C \in RS_N/\mathcal{R}$*

$$\gamma(q', C) = \gamma(q'', C)$$

The notion of *Markovian bisimilarity* ($\sim$) then arises as the largest Markovian bisimulation. This notion of equivalence essentially coincides with the stochastic counterpart of Milner's strong bisimilarity for SPAs, *strong equivalence*. It has been adapted to SPN in [6]. The reader is invited to compare the definition with Definition 5.2 and Definition 5.3.

In Section 5.5.4 it has been illustrated how the notion of strong equivalence can be used to compare and replace components by simpler ones. Indeed we are now have the means to fully explain why this definition is also very favourable for model aggregation and efficient analysis of compositional models:

In Chapter 13 the notion of *strong lumpability* has been introduced as a means to aggregate the CTMC associated with a SWN, justifying the construction of a symbolic reachability graph (SRG). In fact, strong equivalence (in the SPA setting), or Markovian bisimulation (in the setting of SPN) closely corresponds to strong lumpability, formally introduced in Definition 13.3. We present this result formally in the context of SPN, though it has originally been established in the SPA setting [17].

Proposition 16.10 *Let N be a SPN and $\mathcal{R}$ be a Markovian bisimulation on N. Then, the equivalence classes induced by $\mathcal{R}$ partition RS_N into aggregates satisfying the strong lumpability condition.*

Hence, any Markovian bisimulation can be used to *lump* the underlying marking process, and Markovian bisimilarity, the largest Markovian bisimulation, induces the *best possible* lumping, since $\sim$ identifies as many markings as possible without violating the lumpability condition. In the SPA context, strong equivalence plays the same role: The reader is invited to re-investigate the PLC example given in Section 5.4.4 in order to verify that strong equivalence is applied there to form $Comp^{eq}$ in a way that preserves strong lumpability (on the associated CTMC, obtained by removing activity type information from transitions).

16.6.2 Weak equivalence

We now turn our attention from SPN towards CSPN where names of immediate transitions are used for synchronisation purposes. In order to keep track of the impact of immediate transitions on the behaviour of the system, we distinguish between observable and unobservable firings of transitions. In particular, firing of internal immediate transitions (named τ) is not observable. For this purpose, we adapt concepts of weak transitions introduced in Section 16.2 (Definition 16.2) to CSPN.

We introduce the following abbreviations. For $m \in M$ we shall indicate with $q' \stackrel{\lambda}{\longrightarrow} q''$ that $q'\ [m\rangle\ q'' \ \wedge\ \Lambda(m) = \lambda$. Similarly, if $i \in I$ then $q' \stackrel{a}{\longrightarrow} q''$ abbreviates that $q'\ [i\rangle\ q'' \ \wedge\ L(i) = a$. We extend this definition to words of $(Sync \ \cup \{\tau\} \cup\ ]0,\infty[)^*$ in the usual way. For example, $q' \stackrel{\mu\ a\ \tau\ \nu}{\longrightarrow} q''$ is a shorthand for $\exists q_1, q_2, q_3 : q' \stackrel{\mu}{\longrightarrow} q_1 \stackrel{a}{\longrightarrow} q_2 \stackrel{\tau}{\longrightarrow} q_3 \stackrel{\nu}{\longrightarrow} q''$. Furthermore, $q' \stackrel{\tau}{\Longrightarrow} q''$ abbreviates $q' \stackrel{\tau^*}{\longrightarrow} q''$ and $q' \stackrel{a}{\Longrightarrow} q''$ abbreviates $q' \stackrel{\tau^*\ a\ \tau^*}{\longrightarrow} q''$ if $a \neq \tau$. We shall say that a *weak firing sequence* of a transition labelled a leads from marking q' to marking q'' if $q' \stackrel{a}{\Longrightarrow} q''$. Note that, if a is observable, $q' \stackrel{a}{\Longrightarrow} q''$ denotes that there is a firing sequence with exactly one observable transition involved, whereas $q' \stackrel{\tau}{\Longrightarrow} q''$ includes the case that no firing has to take place at all.

With these definitions, we may attack the definition of a variant of weak bisimilarity (Definition 16.3) using weak firing sequences. A few questions have to be addressed in order to achieve a proper definition. While the treatment of immediate transitions can follow the lines of Definition 16.3, we cannot equate a sequence of timed transition firings with a single timed transition firing, since sequences of exponential distributions are not exponentially distributed. So, we are forced to demand that timed transitions have to be bisimulated in the *strong* sense, using function γ, even for a weak bisimulation. However, we allow them to be preceded (and followed) by arbitrary sequences of internal immediate transitions. These sequences are given by $\stackrel{\tau}{\Longrightarrow}$, the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$.

We have argued in the preceding section that internal immediate transitions preempt the firing of concurrently enabled timed transitions. Therefore, timed transition firings preceded by a sequence of internal immediate transitions are only relevant if they emanate a marking q', where no internal immediate transition may fire. Then, the cumulative rate $\gamma(q', C)$ should be taken into account.

We call such a marking q' *stable*, and use $q' \not\xrightarrow{\tau}$ to denote this situation.

Definition 16.11 *Let $N = (P, M, I, F, \Lambda, L, q)$ be a CSPN. A* **weak Markovian bisimulation** *$\mathcal{R}$, is an equivalence relation over RS_N such that whenever $(q', q'') \in \mathcal{R}$ then for all $a \in \mathcal{A}ct \cup \tau$ and for all $C \in RS_N/\mathcal{R}$*

1. *Whenever $q' \stackrel{a}{\Longrightarrow} \hat{q}'$, then for some $\hat{q}''$, $q'' \stackrel{a}{\Longrightarrow} \hat{q}''$ and $(\hat{q}', \hat{q}'') \in \mathcal{R}$;*
2. *Whenever $q'' \stackrel{a}{\Longrightarrow} \hat{q}''$, then for some $\hat{q}'$, $q' \stackrel{a}{\Longrightarrow} \hat{q}'$ and $(\hat{q}', \hat{q}'') \in \mathcal{R}$;*
3. *Whenever $q' \stackrel{\tau}{\Longrightarrow} \hat{q}'$ and $\hat{q}' \not\xrightarrow{\tau}$ then for some $\hat{q}'' \not\xrightarrow{\tau}$, $'q'' \stackrel{\tau}{\Longrightarrow} \hat{q}''$ and*

$$\gamma(\hat{q}', C) = \gamma(\hat{q}'', C).$$

Once again, weak Markovian bisimilarity (denoted $\approx$) arises as the largest weak Markovian bisimulation. Let us explain this definition step by step. For ordinary SPN, immediate transitions are absent by definition and all markings are stable. Hence, only condition *(3.)* must be satisfied. Taking into account that we can choose q'' for $\hat{q}''$, the above definition obviously coincides with Markovian bisimilarity (Definition 16.9), when applied to SPN. On the other hand, for nets without timed transitions merely condition *(2.)* and *(3.)* have to be checked. In this case, the above equivalence is exactly the same as the one given in Definition 16.3 of Section 16.2. So, weak Markovian bisimilarity merges weak bisimilarity for immediate transitions with Markovian bisimilarity for timed transitions. Because of the priority of *internal* immediate transitions, the latter need not be checked for unstable markings (condition *(3.)*). Furthermore, if q' is stable, it is sufficient that q'' can invisibly and immediately descend to a stable marking $\hat{q}''$ that is weakly Markovian bisimilar to q'.

To illustrate Definition 16.11, we consider the PLC example as depicted in Figure 16.14. In Figure 16.16 depicted the reachability graph using again a specific shading of states to indicate which markings are weakly Markovian bisimilar. Each pair of equally shaded states satisfies the above definition. To check the relevant properties manually is rather cumbersome, however we encourage the reader to check a few of the pairs, such as (q', q'') indicated in the figure. A mechanisation of this task can be based on the partition refinement algorithm described in Section 16.3. An implementation requires cubic time and quadratic space, measured in the number of reachable markings [15].

16.6.3 Observable reachability graph

The fact that $\approx$ extends both Markovian and weak bisimilarity has important implications. First, partitioning the reachability graph induces a *lumpable* partition, since $\approx$ inherits this property from $\sim$. In addition, immediate firing sequences between equivalent markings can be eliminated from the graph if they are unobservable.

We exploit these features by factorising the reachability graph with respect to weak Markovian bisimilarity. An observable reachability graph (ORG) is

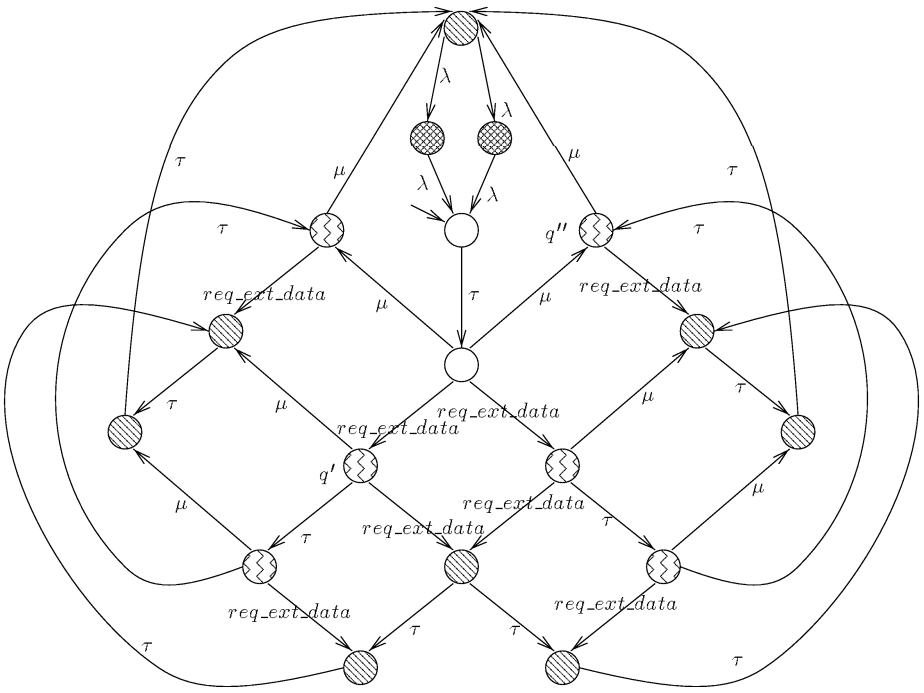


Figure 16.16: Weak Markovian bisimilarity on the PLC example

obtained by representing each equivalence class as a macro state, together with appropriate edges between macro states for edges in RG that cross the border of equivalence classes.

Definition 16.12 *The observable reachability graph ORG_N of a CSPN N is a labelled directed graph whose set of nodes is $ORS_N = RS_N/\approx$ (ranged over by Q, Q', ...) and whose set of arcs $A \subseteq ORS_N \times (Sync \cup \{\tau\} \cup]0,\infty[) \times ORS_N$ is*

$$\{(Q', \gamma(q', Q''), Q'') \mid q' \in Q' \wedge Q' \neq Q'' \wedge q' \stackrel{\tau}{\not\longrightarrow}\} \cup$$
$$\{(Q', a, Q'') \mid a \neq \tau \wedge \exists q' \in Q', q'' \in Q'' : q' \stackrel{a}{\longrightarrow} q''\} \cup$$
$$\{(Q', \tau, Q'') \mid Q' \neq Q'' \wedge \exists q' \in Q', q'' \in Q'' : q' \stackrel{\tau}{\longrightarrow} q''\}.$$

remark that existential and universal quantifiers are interchangeable in this definition, because all markings in a partition are equivalent. The observable reachability graph corresponding to the PLC example (Figure 16.16) is depicted in Figure 16.17.

Weak Markovian bisimilarity is defined on the reachable markings of a single CSPN. In general we want to compare nets. For this purpose, we lift this definition to pairs of CSPN. Two nets are equivalent iff their reachability graphs

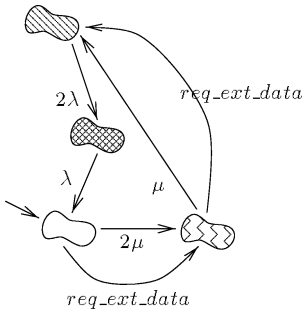


Figure 16.17: Observable RG of the PLC example

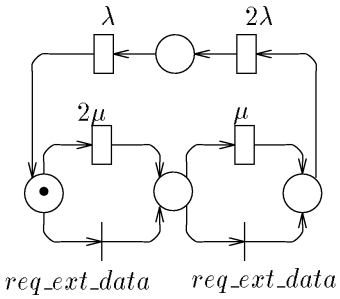


Figure 16.18: A net equivalent to the composite PLC example

are weakly Markovian bisimilar. This obviously holds iff their observable reachability graphs are isomorphic.

Definition 16.13 *Let N_1 and N_2 be two CSPNs with initial markings q_1 and q_2, respectively. N_1 and N_2 are Markovian observational equivalent ($N_1 \approx N_2$), if an isomorphism $\phi : ORS_{N_1} \mapsto ORS_{N_2}$ exists such that*

$$q_1 \in Q \iff q_2 \in \phi(Q) \quad and \quad \phi(ORG_{N_1}) = ORG_{N_2}$$

Following this definition, the net depicted in Figure 16.18 is weak Markovian bisimilar to the composite net of the PLC example (Figure 16.14).

Since we are aiming to use CSPN to estimate performance, we have to address the question of how to transform the (observable) reachability graph of a net into a Markov chain. Similar to the situation in GSPN, where *confusion* may hamper the unambiguous association of a Markov chain (cf. Chapter 3), the presence of *nondeterminism* in CSPN poses some intricate problems. The decisions between concurrently enabled transitions is taken in favour of the fastest, but it is very often not determined which of the transitions is actually faster. This nondeterminism stems from the fact that the decisions can be influenced by the environment, due to additional synchronisation on immediate

transitions. As a consequence, the first step to unambiguously determine the MC underlying some CSPN consists of *hiding* all names, turning the net into a closed system. This rules out any further influence on this net. The ORG of this net can then be interpreted as a *lumped* Markov chain, if it does not contain immediate arcs. This is expressed in the following proposition.

Proposition 16.14 *Let N_1 and N_2 be two CSPNs such that $I_1 \neq \emptyset$ and $I_2 = \emptyset$. If $N_1 \approx N_2$ then ORG_{N_2} is isomorphic to a lumped version of the MC associated with N_1.*

For instance, hiding of *req_ext_data* for the CSPN of Figure 16.14 leads to an ORG that does not contain any immediate transition and it can hence be interpreted as a Markov chain. It is depicted in Figure 16.19.

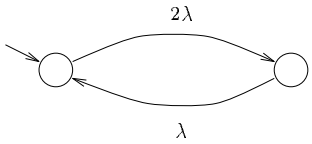


Figure 16.19: ORG obtained after hiding *red_ext_data*

There are, however, some implicit restrictions on the method. First, all names must be hidden before an ORG without immediate arcs is attainable. This requirement is necessary but by no means sufficient to guarantee the successful elimination of immediate firing sequences. The problem is related to the fact that we have excluded weights and priorities for immediate transitions. Consider a net containing a marking q, where two *internal* immediate transitions, say i_1 and i_2, are both enabled and in conflict. Then, it is not determined which one will fire first. This nondeterminism is problematic, if the two markings q_1 and q_2, reachable via $q\ [i_1\rangle\ q_1$ and $q\ [i_2\rangle\ q_2$ do not fall into the same equivalence class. This situation lacks information about which kind of behaviour will be observed with what probability. The stochastic process associated with the net is not fully determined. If otherwise $q_1 \approx q_2$, no problem arises, since q, q_1, and q_2 fall into the same equivalence class of $\approx$. Similarly, if at least one of the transitions is observable, the situation can potentially be improved due to synchronisation in a larger composite net. However, a lack of determinacy is a frequently observed phenomenon in the context of CSPN, reflecting that the modelled system behaves nondeterministically at the chosen level of abstraction.

16.7 Compositional Aggregation

Typical applications of compositionality result in large composite nets that are constructed using a hierarchy of instances of parallel composition and hiding.

If the reachability graph of such net models is constructed at once we usually observe state space explosion. In order to avoid this, we use the properties of CSPN to develop a *compositional* aggregation strategy. The key idea is to generate the reachability graph in a stepwise fashion along the parallel structure of a composite SPN. In every step, the composite net is reduced without affecting the behavioural properties of the net. The aggregation performs lumping and elimination of internal immediate transitions.

The nets in Figure 16.14 and 16.18 are equivalent, and the latter can be regarded as the aggregated representation of the former. However, without being assured that our aggregation preserves relevant properties in the context of composition, we are not allowed to use the latter instead of the former within a composite net.

The crucial property that ensures that the aggregation does not affect relevant (stochastic or logical) properties is the congruence property. The aggregation is based on $\approx$, and hence $\approx$ has to be a congruence with respect to the composition operators in order to perform this compositional aggregation technique. Indeed, the congruence property holds for $\approx$:

Proposition 16.15 *Let N_1, N_2 and N_3 be CSPN. Weak Markovian bisimilarity is substitutive with respect to parallel composition and hiding, i.e.*

$$N_1 \approx N_2 \text{ implies } N_1 \,\underline{\underline{\underline{\underline{\underline{\underline{\underline{\underline{s_1,\ldots,s_n}}}}}}}}\, N_3 \approx N_2 \,\underline{\underline{\underline{\underline{\underline{\underline{\underline{\underline{s_1,\ldots,s_n}}}}}}}}\, N_3,$$

$$N_1 \approx N_2 \text{ implies } N_3 \,\underline{\underline{\underline{\underline{\underline{\underline{\underline{\underline{s_1,\ldots,s_n}}}}}}}}\, N_1 \approx N_3 \,\underline{\underline{\underline{\underline{\underline{\underline{\underline{\underline{s_1,\ldots,s_n}}}}}}}}\, N_2,$$

$$N_1 \approx N_2 \text{ implies } \boxed{N_1 \;\blacksquare_{h_1,\ldots,h_n}} \approx \boxed{N_2 \;\blacksquare_{h_1,\ldots,h_n}}.$$

The validity of this proposition relies particularly on the fact that only internal immediate transitions preempt firing of timed transitions, while other immediate transitions do not have this preempting effect. Proposition 16.15 is the formal justification that we can apply compositional aggregation to composite nets. Assume that we want to use the CSPN describing the PLC example (Figure 16.14) in the context of further composition. Then, the above proposition makes it possible to use the net in Figure 16.18 instead. The state space underlying this net is drastically smaller than the one obtained when using the original net. We will illustrate this property by means of a workstation cluster example, given in Section 16.8.

As a whole, compositional aggregation with $\approx$ may be applied to larger composite nets recursively. If finally the observable reachability graph does not contain immediate arcs, then Proposition 16.14 ensures that we have obtained a lumped version of the MC of the original net. This lumped MC has been derived in a stepwise fashion from the component's behaviour without constructing the whole reachability graph and even without constructing the whole net.

16.8 Application example

In this section we apply CSPN to the modelling of a network of workstations. The model consists of a number of processors (e.g. a workstation cluster) ac-

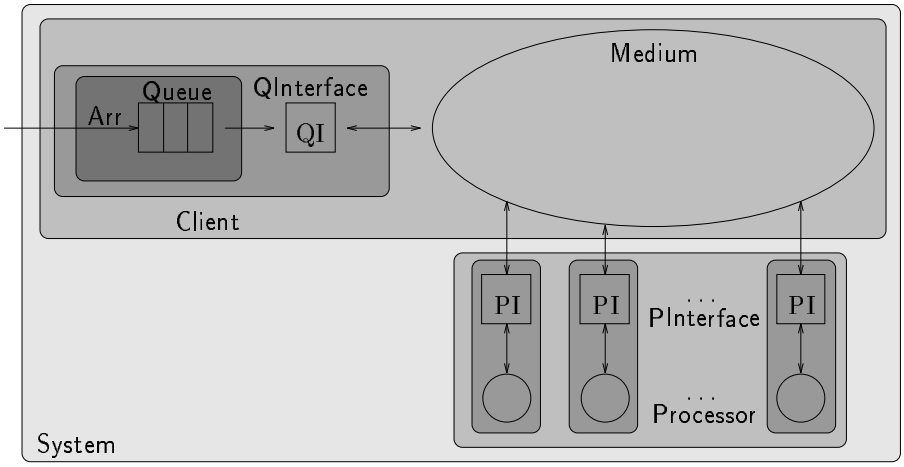


Figure 16.20: A cluster of workstations

cepting jobs from a common client. The stations are connected to each other via a token ring. In particular, communication times caused by the protocol and by the transmission of jobs are explicitly considered. The structure of the system is depicted in Figure 16.20.

The interface of the client puts a request on the medium (sendreq), if a job has appeared in its queue. It then waits for a response. If all servers are currently busy, it receives a fullconf-signal. Then the client waits an exponentially distributed phase with rate *wait* before repeating its request. Eventually it will receive an availconf-signal, indicating that at least one server is free and *exactly one* is ready to accept a job. Now the interface transmits the job to this server via the medium (jobtrans).

The medium transfers messages between stations. This transfer causes message delays. The length of the delay is supposed to be dependent on the type of the message. Short protocol messages are transmitted with the higher rate *msgdelay*, job transfers with the lower rate *jobdelay*.

The interface of each processor is closely connected to the respective processor. If the processor is free the interface awaits a query from the medium and returns availresp. The medium only synchronises with *one* interface, because the availresp signals of the interfaces happen independently of each other. If, on the other hand, all processors are full, they synchronise together with the medium on fullresp.

We choose this simplified model of a token ring because we do not want to blur the concept with technical details. A more detailed model where the rotation of the token is explicitly described can be developed using the composition operators starting from this composite model. It requires a refinement of the interfaces and the medium, while the other net components can remain

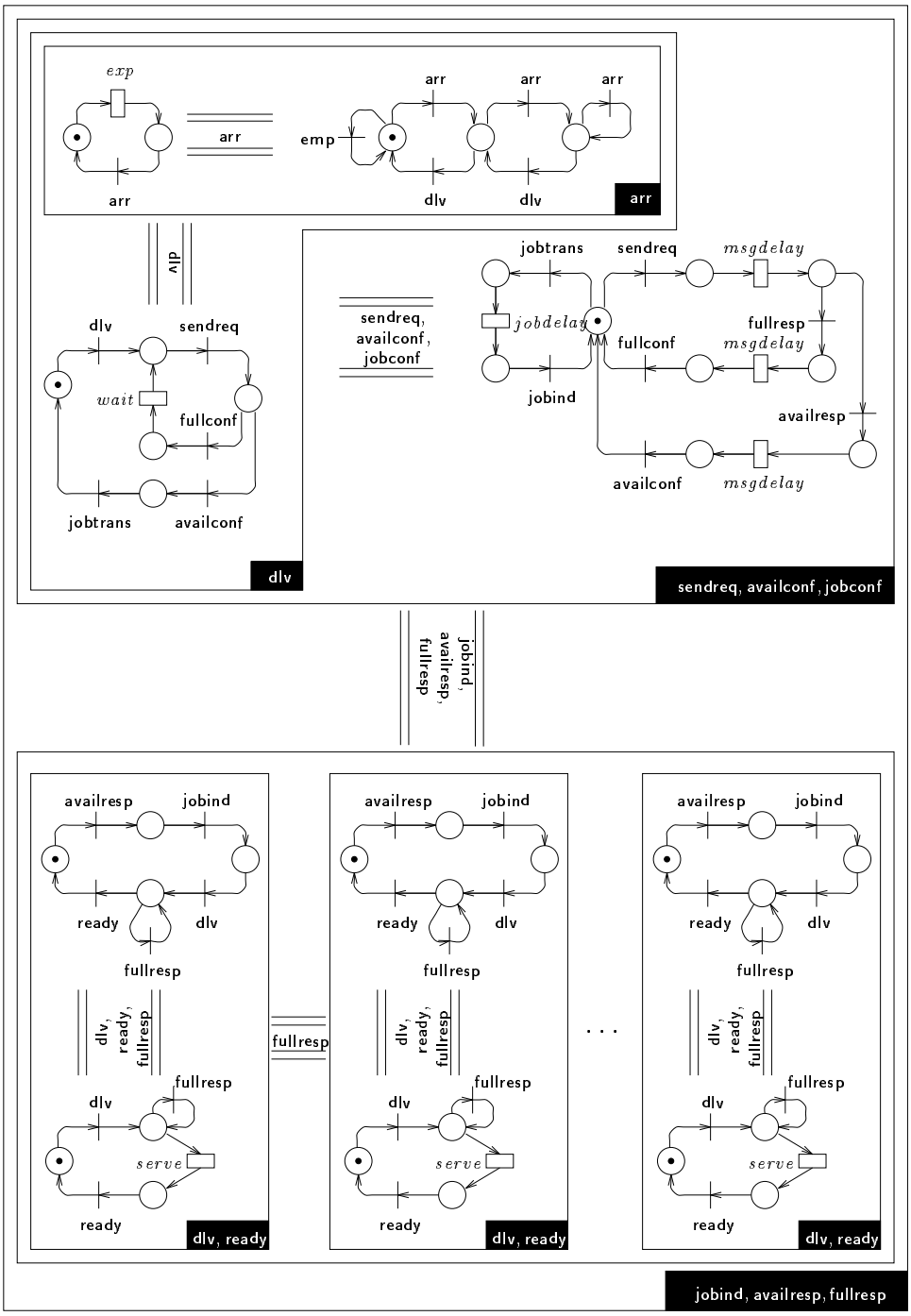


Figure 16.21: Composite SPN of the workstation cluster example

unchanged.

The composite SPN model of the system described above is depicted in Figure 16.21. The parallel structure clearly resembles the block structure of Figure 16.20. As motivated in the preceding sections, immediate transitions are used for synchronisation purposes. If this purpose is fulfilled their names are hidden in order to prevent further synchronisation with other net components of the composite net.

We investigated the system with maximal queue length 2 and with two servers. The reachability graph of this composite net consists of 882 markings and 2798 arcs. Applying the algorithm to compute weak Markovian bisimilarity we compute an observable reachability graph that contains 45 macro states and 113 arcs. This ORG contains no immediate state changes. Thus, Theorem 16.14 implies that this ORG is a representation of the lumped Markov chain associated with the whole composite net.

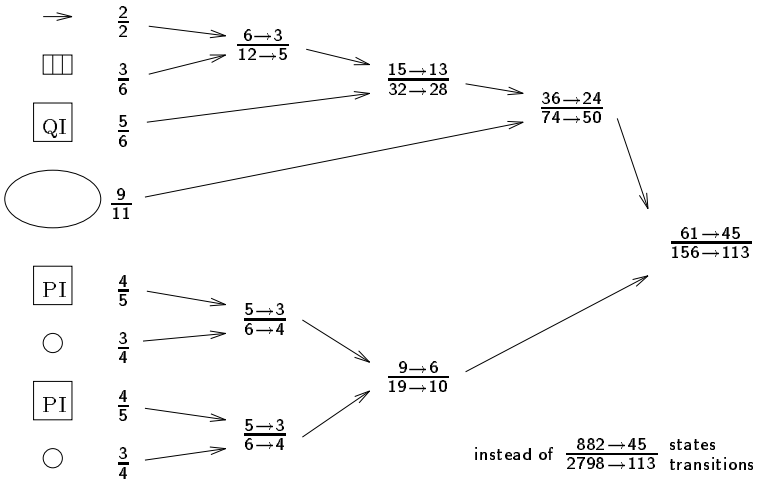


Figure 16.22: Stepwise compositional reduction

Alternatively, we can generate the state space along the compositional structure of the model, and exploit substitutivity of $\approx$ for compositional reduction in each step. This stepwise procedure is depicted in Figure 16.22. In each step we have noted the number of markings (first row) and arcs (second row) of RG (first column) and ORG (second column), i.e. before and after reduction. As an example, the RG of the arrival process has 2 markings and 2 arcs, the RG of the queue possesses 3 markings and 6 arcs. The composition of both components (shown in the upper left of Figure 16.21) has 6 reachable markings and 12 arcs. Applying the refinement algorithm developed in Section 16.3 leads to an ORG with 3 markings and 5 arcs. Due to our substitutivity result, a corresponding net with 3 places and 5 transitions can be used instead of the larger CSPN in subsequent construction and reduction steps.

The resulting ORG of this stepwise procedure is (by construction) isomorphic

to the lumped Markov chain of the whole composite net. Note that the largest intermediate state space that has to be stored (61 markings and 156 arcs) is more than one order of magnitude smaller than that obtained without compositional reduction. The reduction sketched in Figure 16.22 is partly based on lumping of symmetric components. This is especially true for the lower half of this reduction tree. The reduction obtained in the upper half does not exploit symmetries. It essentially relies on early elimination of (hidden) immediate transitions.

It is worth mentioning that the absence of immediate transitions (and hence of nondeterminism) in the resulting ORG is a consequence of the fact that all processors are identical. If they were different, for instance, with respect to their processing speed (*serve*), the specification would turn out to be too coarse to serve as a performance model. The stochastic process associated with the net would not be fully determined, technically because the ORG would contain multiple internal immediate transitions. These transitions reflect different possibilities of choosing a processor for processing a job. In reality, however, the ordering of processors in a ring topology will always uniquely determine the processor to be chosen. So, a more accurate compositional net would have to be developed where the token rotation is modelled explicitly, essentially by refining the medium and the interfaces of the servers. This highlights the fact that nondeterminism may arise in a CSPN reflecting a certain level of abstraction, while it can be resolved in a more concrete compositionally specified net. This is different to the situation in GSPN, where probabilities of different immediate alternatives can be defined. In this way nondeterminism is resolved without the need to change to a more concrete level of abstraction.

16.9 Conclusion

In this chapter we have introduced composition operators for a subset of GSPN together with a substitutive notion of equivalence. We have shown that the congruence property has important effects on the construction and solution of such models. This is the central benefit we obtain by exploiting established results from stochastic process algebras [17, 16] to Petri nets. Defining a net model by composition of smaller nets is not only a means to effectively construct complex nets. The compositional structure can be exploited during the generation of the state space by stepwise compositional aggregation. The congruence property of our equivalence ensures that behavioural properties (performance properties, in particular) remain unchanged whereas the size of the state space is reduced. We have also sketched an efficient algorithm to compute reduced state spaces by means of partition refinement.

The class of GSPN we considered is quite restrictive. We have excluded priorities as well as probability assignments to immediate transitions. Therefore, nondeterminism might hamper the association of a Markov chain to some CSPN. It is a demanding task for future work to define an extension of CSPN with probabilities.

16.10 Bibilographic remarks

The idea of composing large nets out of smaller building blocks is inspired by advances in the area of process algebra. For untimed Petri nets this compositional approach has cumulated in the definition of the *box calculus* [1]. Composition of Generalized Stochastic Petri Nets has been discussed in [3, 9] as main constituents of a methodology to construct complex GSPN models, see also Chapter 12. [24] investigates compositional construction of Stochastic well-formed nets. The composition operators used in CSPN are different from these alternative approaches, in fact, they are borrowed from LOTOS [2]. Their definition ensures that the substitutivity property holds, allowing compositional construction and aggregation techniques. From an algebraic point of view, this property is discussed in [16].

The algorithms that allow efficient compositional aggregation are based on a partition refinement strategy. Though originally developed in [19, 23], the probably best explanation of the key implementation details can be found in [11]. The general strategy of compositional aggregation has successfully been applied to a variety of non-stochastic concurrent systems in order to perform verification of functional properties, see e.g. [4, 10, 13, 14, 7, 21]. In the stochastic setting, the benefits of compositional aggregation have been discussed for instance in [17], where also the correspondence between lumpability and bisimulation is established.

One effect of our compositional aggregation method is mechanised detection and exploitation of symmetries in a hierarchical specification. Stochastic well-formed nets, as introduced in Chapter 13 allow explicit specifications of such symmetries. This eases their exploitation, because a lumped state space (the SRG) can be constructed directly. On the other hand, the power of aggregation based on bisimulation goes beyond symmetry exploitation.

Another effect of compositional aggregation is that immediate transitions can be eliminated as soon as they are irrelevant in the context of further composition. Elimination of immediate transitions at component level is also possible in the superposed GSPN approach (Chapter 12), because in this approach immediate transitions cannot be used for composition purposes. This is different from our approach, where immediate transitions are essential for composition. In a sense CSPN and SGSPN approach the composition of net models in two orthogonal ways. The former composes via immediate transitions only, while the latter composes via timed transitions only.

Bibliography

[1] E. Best, R. Devillers, and J. Hall. The Petri Box Calculus: A new causal algebra with multilabel communication. In G. Rozenberg, editor, *Advances in Petri Nets*, pages 21–69. Springer LNCS 609, 1992.

[2] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14, 1987.

[3] O. Botti and F. De Cindio. Process and Resource boxes: An Integrated PN Performance Model for Application and Architectures. In *IEEE proc. of the Int. Conference on Systems, Man and Cybernetics*, 1993.

[4] A. Bouajjani, J.C. Fernandez, N. Halbwachs, P. Raymond, and C. Rattel. Minimal state graph generation. *Science of Computer Programming*, 18(3):247–271, 1992.

[5] A. Bouali. Weak and branching bisimulation in FCTOOL. Rapports de Recherche 1575, INRIA Sophia Antipolis, Valbonne Cedex, France, 1992.

[6] P. Buchholz. An notion of equivalence for Stochastic Petri Nets. In *Application and Theory of Petri Nets*. Springer LNCS 935, 1995.

[7] G. Chehaivbar, H. Garavel, N. Tawbi, and F. Zulian. Specification and Verification of the Powerscale Bus Arbitration Protocol: An Industrial Experiment with LOTOS. In R. Gotzhein and J. Bredereke, editors, *Formal Description Techniques IX*. Chapmann Hall, 1996.

[8] D. Coppersmith and S. Winograd. Matrix Multiplication via Arithmetic Progressions. In *Proc. 19th ACM Symposium on Theory of Computing*, 1987.

[9] S. Donatelli and G. Franchesinis. The PSR Methodology: Integrating Hardware and Software Models. In *Application and Theory of Petri Nets*. Springer LNCS 1102, 1996.

[10] J.C. Fernandez. *Aldébaran, Un Système de Vérification par Réduction de Processus Communicants*. PhD thesis, Université de Grenoble, 1988.

[11] J.C. Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 13:219–236, 1989.

[12] S. Gilmore and J. Hillston, editors. *Proc. of the 3rd Workshop on Process Algebras and Performance Modelling.* Oxford University Press, Special Issue of "The Computer Journal", 38(7) 1995.

[13] S. Graf and P. Loiseaux. Programm verification using compositional abstraction. In *Proceedings of FASE/TAPSOFT'93*, 1993.

[14] S. Graf, B. Steffen, and G. Luettgen. Compositional Minimization of Finite State Systems. *Formal Aspects of Computing*, 8(5):607–616, 1996.

[15] H. Hermanns. *Interactive Markov Chains.* PhD thesis, Universität Erlangen-Nürnberg, 1998.

[16] H. Hermanns, M. Rettelbach, and T. Weiß. Formal Characterisation of Immediate Actions in SPA with Nondeterministic Branching. In Gilmore and Hillston [12].

[17] J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

[18] J.F.Groote and F.W.Vaandrager. An efficient algorithm for branching bisimulation and stuttering equivalence. In M. S. Paterson, editor, *Seventeenth Colloquium on Automata, Languages and Programming (ICALP) (Warwick, England)*, volume 443 of *Lecture Notes in Computer Science*, pages 626–638. Springer, 1990.

[19] P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86:43–68, 1990.

[20] J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Springer, 1976.

[21] J.-P. Krimm and L. Mounier. Compositional State Space Generation from Lotos Programs. In *Proc. of 3rd Int. Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 97)*, volume 1217 of *Lecture Notes in Computer Science*, Twente, April 1997. Springer.

[22] R. Milner. *Communication and Concurrency.* Prentice Hall, London, 1989.

[23] R. Paige and R. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.

[24] I. Rojas. Compositional construction of SWN models. In Gilmore and Hillston [12].

Part V

Net-Level Analysis Techniques

Chapter 17

Performance Bounds

A complementary approach to exact or approximation techniques for the analysis of queueing network models is the computation of *bounds* for their performance measures. Performance bounds are useful in the preliminary phases of the design of a system, in which many parameters are not known accurately. Several alternatives for those parameters should be quickly evaluated, and rejected those that are clearly bad. Exact (and even approximate) solutions would be computationally very expensive. Bounds become useful in these instances since they usually require much *less computation effort*.

In this chapter, we concentrate in *net-driven* techniques for the computation of bounds for the main performance indices of timed Petri net models. Previous works on bounds computation for classical queueing networks are not included here and the interested reader is referred to the bibliographic remarks in Section 17.5. The presented techniques are characterized by their interest in stressing the intimate relationship between qualitative and quantitative aspects of Petri nets. In particular, the intensive use of *structure theory* of net models allows to obtain very *efficient computation* techniques.

The organization of the chapter is the following. In Section 17.1, a general approach for the computation of upper and lower bounds for arbitrary linear functions of average marking of places and throughput of transitions of both timed Place/Transition nets and timed Well-Formed Coloured nets is presented. Section 17.2 includes a more intuitive approach for the computation of throughput upper bounds, even if it is valid only for restricted Petri net subclasses. The relation of this technique with the general approach and the attainability of the bound for a particular subclass of nets is included. In Section 17.3, two possible improvements of upper and lower throughput bounds are presented using implicit places and liveness bounds of transitions, respectively. All the bounds computed in Sections 17.1, 17.2 and 17.3 are *insensitive* to the timing probability distributions since they are based only on the knowledge of the average service times. Section 17.4 includes a brief overview of three additional techniques for the improvement of the bounds. Since some additional assumptions on the form of the probability distribution functions associated

to the service of transitions or on the conflict resolution policies are done, the obtained bounds are non-insensitive. Finally, some bibliographic remarks are included in Section 17.5.

17.1 Insensitive Performance Bounds

In this section, we present a general approach for the computation of bounds for performance indices of both *timed Place/Transition nets* and *timed Well-Formed Coloured nets*. The bounds are computed from the solution of proper linear programming problems, therefore they can be obtained in polynomial time on the size of the net model, and they depend only on the mean values of service time associated to the firing of transitions and the routing rates associated with transitions in conflict and not on the higher moments of the probability distribution functions of the random variables that describe the timing of the system. This notion of independence of the computed mean measures on the form of the probability distribution functions is known as the *insensitivity property* in queueing networks literature. The independence of the probability distribution can be viewed as a practical estimation of the performance results, since higher moments of the delays are usually unknown for real cases, and difficult to estimate and assess.

Unless otherwise explicitly stated, in this chapter we assume an *infinite-server semantics* for timed transitions, in other words, a transition t enabled K times in a marking $\mathbf{m}$ (i.e., $K = \max\{k \mid \mathbf{m} \geq k\mathbf{Pre}[\cdot, t]\}$) works at speed K times that it would work in the case it was enabled only once. Of course, an infinite-server transition can always be constrained to a "k–server" behaviour by adding one place that is both input and output (self-loop with multiplicity one) for that transition and marking it with k tokens. Other kinds of marking or time dependency of service times are forbidden.

17.1.1 General Statement Based on Linear Programming

In Chapter 8 (cfr. Section 8.3.2) several linear operational laws that relate the average marking of places and the throughput of transitions were derived from the definition of these performance measures. Those equations and inequalities can be used to compute upper and lower bounds for the throughput of transitions or for the average marking of places for general timed Petri nets using *linear programming* techniques. The idea is to compute vectors $\boldsymbol{\chi}$ and $\overline{\boldsymbol{\mu}}$ that maximize or minimize the throughput of a transition or the average marking of a place among those verifying the operational laws and other linear constraints that can be easily derived from the net structure.

A first set of linear equality constraints can be derived from the fact that the average marking vector $\overline{\boldsymbol{\mu}}$ is an average weight of reachable markings:

$$\overline{\boldsymbol{\mu}} = \sum_{\mathbf{m_r} \in RS(\mathbf{m_0})} \beta_r \mathbf{m_r}$$

Since for each reachable marking,

$$\mathbf{m_r} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma_r}$$

we obtain that also the average marking must satisfy the same linear equation:

$$\overline{\boldsymbol{\mu}} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$$

where

$$\boldsymbol{\sigma} = \sum_{\mathbf{m_r} \in \mathrm{RS}(\mathbf{m_0})} \beta_r \boldsymbol{\sigma_r}$$

The following set of linear inequalities imposes that for each place the token flow out is less than or equal to the token flow in:

$$\sum_{t \in {}^\bullet p} \boldsymbol{\chi}[t]\ \mathbf{Post}[p,t] \geq \sum_{t \in p^\bullet} \boldsymbol{\chi}[t]\ \mathbf{Pre}[p,t], \quad \forall p \in P$$

If place p is known to be bounded, then the above inequality becomes an equality which represents the classical *flow balance* equation:

$$\mathbf{C}[p,\cdot] \cdot \overline{\boldsymbol{\mu}} = 0$$

On the other hand, for each pair of transitions t_i, t_j in (behavioural) free conflict (i.e., such that they are always simultaneously enabled or disabled the following equation is verified:

$$\frac{\boldsymbol{\chi}[t_i]}{r_i} = \frac{\boldsymbol{\chi}[t_j]}{r_j}$$

where r_i, r_j are the routing rates that define the resolution of the conflict between t_i and t_j.

Additionally, most of the operational inequality laws that were derived in Chapter 8 linearly relate the average marking of places with the throughput of their output transitions. Hence they can be considered as constraints for the following linear programming problem:

maximize [or minimize] $f(\overline{\boldsymbol{\mu}}, \boldsymbol{\chi})$ with f a linear function of $\overline{\boldsymbol{\mu}}, \boldsymbol{\chi}$
subject to

$$(c_1)\quad \overline{\boldsymbol{\mu}} = \mathbf{m}_0 + \mathbf{C}\cdot\boldsymbol{\sigma};$$

$$(c_2)\quad \sum_{t\in {}^\bullet p} \boldsymbol{\chi}[t]\ \mathbf{Post}[p,t] \geq \sum_{t\in p^\bullet} \boldsymbol{\chi}[t]\ \mathbf{Pre}[p,t], \qquad \forall p\in P;$$

$$(c_2')\quad \sum_{t\in {}^\bullet p} \boldsymbol{\chi}[t]\ \mathbf{Post}[p,t] = \sum_{t\in p^\bullet} \boldsymbol{\chi}[t]\ \mathbf{Pre}[p,t], \qquad \forall p\in P \text{ bounded};$$

$$(c_3)\quad \frac{\boldsymbol{\chi}[t_i]}{r_i} = \frac{\boldsymbol{\chi}[t_j]}{r_j}, \qquad \forall t_i, t_j \in T : \text{behav. free choice (e.g. } \mathbf{Pre}[p,t_i] = \mathbf{Pre}[p,t_j], \forall p\in P);$$

$$(c_4)\quad \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t] \leq \frac{\overline{\boldsymbol{\mu}}[p]}{\mathbf{Pre}[p,t]}, \qquad \forall t\in T, \forall p\in {}^\bullet t;$$

$$(c_5)\quad \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t] \geq \frac{\overline{\boldsymbol{\mu}}[p] - \mathbf{Pre}[p,t] + 1}{\mathbf{Pre}[p,t]}, \qquad \forall t\in T \text{ persistent, age memory or immediate (p/a-m/i): } {}^\bullet t = \{p\};$$

$$(c_5')\quad \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t] \geq k\frac{\overline{\boldsymbol{\mu}}[p] - k\ \mathbf{Pre}[p,t] + 1}{\mathbf{b}[p] - k\ \mathbf{Pre}[p,t] + 1}, \qquad \forall t\in T \text{ p/a-m/i: } {}^\bullet t = \{p\} \wedge k\in \mathbb{N} : k\ \mathbf{Pre}[p,t] \leq \mathbf{b}[p] < (k+1)\mathbf{Pre}[p,t];$$

$$(c_6)\quad \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t]\ \mathbf{Pre}[p_1,t] \geq \overline{\boldsymbol{\mu}}[p_1] - \mathbf{Pre}[p_1,t] + 1 - \mathbf{b}[p_1]\cdot\left(1 - \frac{\overline{\boldsymbol{\mu}}[p_2] - \mathbf{Pre}[p_2,t] + 1}{\mathbf{b}[p_2] - \mathbf{Pre}[p_2,t] + 1}\right), \qquad \forall t\in T \text{ p/a-m/i: } {}^\bullet t = \{p_1,p_2\}, \mathbf{b}[p_1] \leq \mathbf{b}[p_2];$$

$$(c_7)\quad \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t]\ \mathbf{Pre}[p_1,t] \geq \overline{\boldsymbol{\mu}}[p_1] - \mathbf{Pre}[p_1,t] + 1 - \mathbf{b}[p_1]\max_{1<j\leq k}\left(1 - \frac{\overline{\boldsymbol{\mu}}[p_j] - \mathbf{Pre}[p_j,t] + 1}{\mathbf{b}[p_j] - \mathbf{Pre}[p_j,t] + 1}\right), \qquad \forall t\in T \text{ p/a-m/i: } {}^\bullet t = \{p_1,\cdots,p_k\}, \mathbf{b}[p_1] \leq \mathbf{b}[p_j];$$

$$(c_8)\quad \overline{\boldsymbol{\mu}}, \boldsymbol{\chi}, \boldsymbol{\sigma} \geq \mathbf{0}$$

(17.1)

As we remarked before, constraint (c_2) becomes an equality for bounded places (c_2'). Constraints (c_4) to (c_7) correspond to the operational inequality laws that were derived in Chapter 8. The equality sign also holds true in (c_4) if $\sum_{p\in {}^\bullet t} \mathbf{Pre}[p,t] = \mathbf{1}$ since in this case it may be combined with the opposite inequality (c_5). The constraint labelled with (c_5) can be improved if the input place to t is bounded, by introducing the additional constraint (c_5') (where $\mathbf{b}[\mathrm{p}]$ is the marking bound of p).

The above linear programming problem provides a general method to compute upper and lower bounds for arbitrary linear functions of average marking of places and throughput of transitions. For instance, if $f(\overline{\boldsymbol{\mu}}, \boldsymbol{\chi}) = \boldsymbol{\chi}[t]$, then the problem can be used to compute an upper or a lower bound (depending on the selection of "max" or "min" optimization for the objective function) for the throughput of transition t. In an analogous way, upper or lower bounds for the average marking of a given place p can be derived by solving the above linear programming problem for the objective function $f(\overline{\boldsymbol{\mu}}, \boldsymbol{\chi}) = \overline{\boldsymbol{\mu}}[p]$. The bounds are insensitive to the timing probability distributions since they are based only on the knowledge of the average service times.

The reader should notice also that most equalities and inequalities contain coefficients that depend only on the net structure and on the (known) average transition firing times (or probabilities in case of free choice immediate conflicts). The only coefficients that may be unknown at the time of the formulation of the model are the actual bounds for places $\mathbf{b}[p]$. If the modeller has no more *a priori* precise knowledge of these bounds, notice that an upper bound for them that can be used in the linear programming problem in (17.1) may be computed from a simplified linear programming problem that contains only constraint (c_1) (structural marking bound).

An improvement of the proposed bounds can be obtained if additional constraints that improve the linear characterization of the average marking in terms of the equation $\overline{\boldsymbol{\mu}} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$ are considered. For instance, if a *trap* Θ (i.e., $\Theta \subseteq P, \Theta^\bullet \subseteq {}^\bullet\Theta$) is not a P-semiflow, the net is live, and we are interested only in the steady state performance, then we can add the constraint:

$$\sum_{p \in \Theta} \overline{\boldsymbol{\mu}}[p] \geq 1$$

Similarly, if a *siphon* Σ (i.e., $\Sigma \subseteq P, {}^\bullet\Sigma \subseteq \Sigma^\bullet$) is not a P-semiflow and the net is live, then we can add the constraint:

$$\sum_{p \in \Sigma} \overline{\boldsymbol{\mu}}[p] \geq 1$$

A systematic method for the improvement of linear characterization of reachable markings [16] based on the addition of *implicit places* can be also applied and will be presented later in Section 17.3.1.

We remark that linear programming problems can be solved in *polynomial time* [26], therefore the above presented method for the computation of (upper and lower) bounds for the throughput and for the average marking of general timed nets has a polynomial complexity on the number of nodes of the net. Moreover, the well-known *simplex* method for the resolution of linear programming problems proceeds in linear time in most cases even if it has a theoretically exponential complexity.

17.1.2 Extension to Timed Well-Formed Coloured Nets

For Timed Well-Formed Coloured Nets (TWN's) it is possible to derive, directly from the inequalities developed in Chapter 8, operational relations allowing an efficient computation of performance bounds. Given a TWN, the basic idea is to consider the corresponding unfolded net and to apply the relations previously developed. The relations for the TWN are then obtained combining the partial results for the unfolded one.

A fundamental property that TWN's must have in order to be able to combine the results for the unfolded one is the *symmetry*, meaning that in the unfolded nets obtained from the Well-Formed ones all colour instances of a given place and of a given transition must be equivalent. To be more precise, if a

transition t has average service time $\overline{\mathbf{s}}[t]$, then all of its instances have the same average service time. Moreover if a place p is bounded, then we assume that the maximum number of tokens that each of its instances can contain is the same.

In the following paragraphs we show, as an example, the derivation of *Minimum Throughput Law for single input arc* for TWN's.

Firstly, we recall some basic notations used in the derivations of relations for TWN's. A *generic function* is $f = \sum_{j=1}^{k} F_j$, where F_j is the j^{th} tuple and its arity l is given by the number of colour classes composing the colour domain of the place. This definition of function is slightly different from the classical one, since here we allow linear combinations only outside the tuples (i.e., each tuple is composed only by elementary functions). For example the function $F = \langle S - x, y \rangle$ is written as $F' = \langle S, y \rangle - \langle x, y \rangle$.

The *cardinality of a function* is defined as $|f| = \sum_{j=1}^{k} |F_j|$, where $|F_j| = \alpha_j \times \prod_{i=1}^{l} |(F_j)_i|$ is the cardinality of the j^{th} tuple. The coefficient α_j denotes the product of the coefficients of the elementary functions composing the tuple and $(F_j)_i$ is the i^{th} function of the j^{th} tuple. For example if $F_j = \langle 3x, 2y \rangle$, then $\alpha_j = 6$.

Each tuple F_j of a function f identifies a set of arcs, or *family of arcs*, (with weight α_j), whose cardinality is $A(F_j) = \prod_{i=1}^{l} |(F_j)_h|$. The global number of arcs corresponding to function f is $A(f) = \sum_{j=1}^{k} A(F_j)$, where each $A(F_j)$ has the sign of the corresponding tuple F_j. When $A(f) = 1$, then we denote as α_f the weights associated to the unique family of arcs corresponding to f.

If t is an input transition of place p (with function f), then $IN(p,t) = \frac{|cd(t)|}{|cd(p)|} A(f)$ is the number of input instances of t for each instance of p. Similarly if t is an output transition of place p, then $OUT(p,t) = \frac{|cd(t)|}{|cd(p)|} A(f)$ is the number of output instances of t for each instance of p.

To apply the Minimum Throughput Law for single input arc to an unfolded net, the conditions for its applicability must be met for all transition instances. This means that each instance t_i of a coloured transition t must have only one input place. This condition is met if the function f labelling the arc contains only *projection* and *successor* elementary functions (that is $A(f) = 1$).

Property 17.1 (Minimum Throughput Law for single input arc) *For all transition $t \in T$ with ${}^\bullet t = \{p\}$, $\mathbf{Pre}[p,t] = f$, and $A(f) = 1$:*

$$\alpha_f \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t] \geq OUT(p,t)\overline{\boldsymbol{\mu}}[p] - |cd(t)|(\alpha_f - 1)$$

Proof:
Assume to have a portion of a TWN containing transition t and its input place p and that $|cd(t)| = n$ and $|cd(p)| = m$. Considering the n instances of t we can write the following set of inequalities

$$\forall i \in \{1, \ldots, n\}\ \ \alpha_f \boldsymbol{\chi}[t_i]\ \overline{\mathbf{s}}[t_i] \geq \overline{\boldsymbol{\mu}}[p_{t_i}] - \alpha_f + 1$$

where p_{t_i} is the unique input place of transition instance t_i. Summing the left-hand sides and the right-hand sides of the above inequalities we obtain:

(c_1)	$\overline{\boldsymbol{\mu}}[p] = \mathbf{m}_0[p] + \sum_{t_i \in {}^\bullet p} \lvert f_i \rvert \boldsymbol{\sigma}[t_i] - \sum_{k_j \in p^\bullet} \lvert g_j \rvert \boldsymbol{\sigma}[k_j]$,	$\forall p \in P : \mathbf{Post}[p,t_i] = f_i$, $\mathbf{Pre}[p,k_i] = g_j$;
(c_2)	$\sum_{t_i \in {}^\bullet p} \lvert f_i \rvert \boldsymbol{\chi}[t_i] \geq \sum_{k_j \in p^\bullet} \lvert g_j \rvert \boldsymbol{\chi}[k_j]$,	$\forall p \in P : \mathbf{Post}[p,t_i] = f_i$, $\mathbf{Pre}[p,k_i] = g_j$;
(c_2')	$\sum_{t_i \in {}^\bullet p} \lvert f_i \rvert \boldsymbol{\chi}[t_i] = \sum_{k_j \in p^\bullet} \lvert g_j \rvert \boldsymbol{\chi}[k_j]$,	$\forall p \in P$ bounded;
(c_3)	$\frac{\boldsymbol{\chi}[t_i]}{r_i} = \frac{\boldsymbol{\chi}[t_j]}{r_j}$,	$\forall t_i, t_j \in T$: behav. free choice;
(c_4)	$\lvert f \rvert \boldsymbol{\chi}[t]\ \bar{\mathbf{s}}[t] \leq OUT(p,t)\ \overline{\boldsymbol{\mu}}[p]$,	$\forall t \in T, \forall p \in {}^\bullet t : \mathbf{Pre}[p,t] = f$;
(c_5)	$\alpha_f\ \boldsymbol{\chi}[t]\ \bar{\mathbf{s}}[t] \geq OUT(p,t)\ \overline{\boldsymbol{\mu}}[p] - \lvert cd(t) \rvert (\alpha_f - 1)$,	$\forall t \in T$ persistent, age memory or immediate: ${}^\bullet t = \{p\}$, $\mathbf{Pre}[p,t] = f$, $A(f) = 1$;
(c_5')	$\boldsymbol{\chi}[t]\ \bar{\mathbf{s}}[t] \geq k \frac{OUT(p,t)\overline{\boldsymbol{\mu}}[p] + \lvert cd(t) \rvert (1 - k\alpha_f)}{OUT(p,t) + \lvert cd(t) \rvert (1 - k\alpha_f)}$,	$\forall t \in T$ persistent, age memory or immediate: ${}^\bullet t = \{p\}$, $\mathbf{Pre}[p,t] = f$, $A(f) = 1$, $\wedge\ k \in \mathbb{N} : k\alpha_f \leq \mathbf{b}[p] \leq (k+1)\alpha_f$;
(c_6)	$\alpha_f \boldsymbol{\chi}[t]\ \bar{\mathbf{s}}[t] \geq OUT(p,t)\overline{\boldsymbol{\mu}}[p] + \lvert cd(t) \rvert (1 - \alpha_f) - OUT(p,t)\mathbf{b}[p] f_q$, where $f_q = \lvert cd(t) \rvert - \frac{OUT(q,t)\overline{\boldsymbol{\mu}}[q] + \lvert cd(t) \rvert (1-\alpha_g)}{OUT(q,t)\mathbf{b}[q] + \lvert cd(t) \rvert (1-\alpha_g)}$,	$\forall t \in T$ persistent, age memory or immediate: ${}^\bullet t = \{p,q\}$, $\mathbf{b}[p] \leq \mathbf{b}[q]$, $\mathbf{Pre}[p,t] = f, \mathbf{Pre}[q,t] = g, A(f) = A(g) = 1$;
(c_7)	$\alpha_1 \boldsymbol{\chi}[t]\ \bar{\mathbf{s}}[t] \geq OUT(p1,t)\overline{\boldsymbol{\mu}}[p1] - \lvert cd(t) \rvert (-\alpha_1 + 1) - OUT(p_1,t)\mathbf{b}[p_1] \max_{1 \leq j \leq n} f_j$, where $f_j = 1 - \frac{OUT(p_j,t)\overline{\boldsymbol{\mu}}[p_j] + \lvert cd(p_j) \rvert (1-\alpha_j)}{\mathbf{b}[p_j]/\lvert cd(p_j) \rvert - \alpha_j + 1}$,	$\forall t \in T$ persistent, age memory or immediate: ${}^\bullet t = \{p_1, \ldots, p_n\}, \mathbf{b}[p_1] \leq \mathbf{b}[p_j], j \in \{2, \ldots, n\}$, $\mathbf{Pre}[p_i,t] = f_i$, $A(f_1) = 1$;
(c_8)	$\overline{\boldsymbol{\mu}}, \boldsymbol{\chi}, \boldsymbol{\sigma} \geq \mathbf{0}$	

Table 17.1: Constraints for a linear programming problem for TWN's.

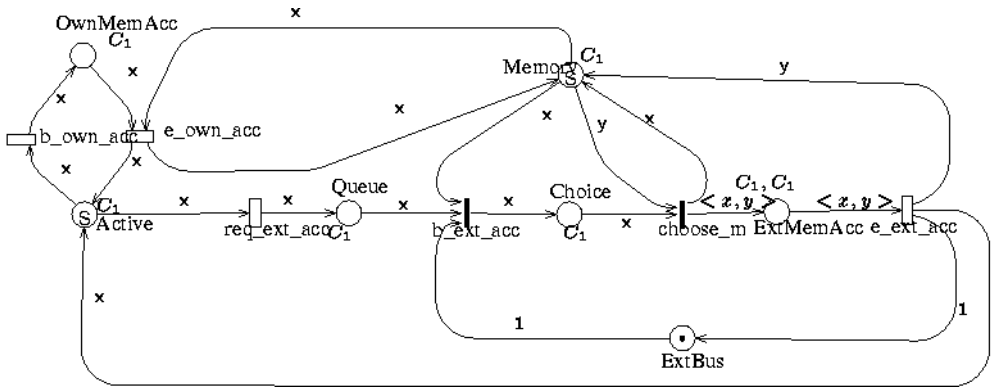


Figure 17.1: TWN model of a shared-memory multiprocessor.

$$\alpha_f \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t] \geq \left(\sum_{i=1}^{n} \overline{\boldsymbol{\mu}}[p_{t_i}] - |cd(t)|(\alpha_f - 1) \right) \tag{17.2}$$

Since each instance of p appears exactly $OUT(p,t)$ times in the summation of the above expression we can rewrite inequality (17.2) as

$$\alpha_f \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t] \geq \left(OUT(p,t) \sum_{i=1}^{m} \overline{\boldsymbol{\mu}}[p_i] - |cd(t)|(\alpha_f - 1) \right)$$

and the result follows. ◊

In a similar way it is possible to derive, for TWN's, the equivalent of relations devised for timed Petri nets. Therefore, performance bounds for TWN's can be computed solving linear programming problems with constraints included in Table 17.1 and any linear function of $\overline{\boldsymbol{\mu}}$ and $\boldsymbol{\chi}$ as objective function.

The average marking equation is written here in explicit form, but it could be written also in matricial form. Moreover relation (c_7) has been derived for TWN's under the hypothesis of strong symmetries. In particular we assumed that, for each input place of transition t in inequality (c_7), the weights of the arcs belonging to the families corresponding to the function labelling the arc are the same. Obviously the uncoloured version of (c_7) has no such restriction.

As we remarked in the case of timed Petri nets, also for TWN's constraint (c_2) becomes an equality for bounded places (c_2'). The equality sign also holds true in (c_4) if $\alpha_f = 1$ (i.e., the unique family of arcs corresponding to function f have weight 1) since in this case it may be combined with the opposite inequality (c_5). The constraint labelled with (c_5) can be improved if the input place to t_i is bounded, by introducing the additional constraint (c_5').

(c_1)	$\overline{\boldsymbol{\mu}}[Active] = 4 + \boldsymbol{\sigma}[e\_e\_a] + \boldsymbol{\sigma}[e\_o\_a] - \boldsymbol{\sigma}[r\_e\_a] - \boldsymbol{\sigma}[b\_o\_a];$
	$\overline{\boldsymbol{\mu}}[Memory] = 4 + \boldsymbol{\sigma}[e\_e\_a] - \boldsymbol{\sigma}[b\_e\_a];$
	$\overline{\boldsymbol{\mu}}[OwnMemAcc] = \boldsymbol{\sigma}[b\_o\_a] - \boldsymbol{\sigma}[e\_o\_a];$
	$\overline{\boldsymbol{\mu}}[Queue] = \boldsymbol{\sigma}[r\_e\_a] - \boldsymbol{\sigma}[b\_e\_a];$
	$\overline{\boldsymbol{\mu}}[Choice] = \boldsymbol{\sigma}[b\_e\_a] - \boldsymbol{\sigma}[c\_m];$
	$\overline{\boldsymbol{\mu}}[ExtMemAcc] = \boldsymbol{\sigma}[c\_m] - \boldsymbol{\sigma}[e\_e\_a];$
	$\overline{\boldsymbol{\mu}}[ExtBus] = 1 + \boldsymbol{\sigma}[e\_e\_a] - \boldsymbol{\sigma}[b\_e\_a];$
(c'_2)	$\boldsymbol{\chi}[e\_e\_a] + \boldsymbol{\chi}[e\_o\_a] = \boldsymbol{\chi}[r\_e\_a] + \boldsymbol{\chi}[b\_o\_a];$
	$\boldsymbol{\chi}[b\_e\_a] = \boldsymbol{\chi}[c\_m] = \boldsymbol{\chi}[e\_e\_a] = \boldsymbol{\chi}[r\_e\_a];$
(c_3)	$\boldsymbol{\chi}[b\_o\_a] = \boldsymbol{\chi}[r\_e\_a];$
$(c_4 \& c_5)$	$\boldsymbol{\chi}[b\_o\_a]\ \overline{\mathbf{s}}[b\_o\_a] = \overline{\boldsymbol{\mu}}[Active]/2;$
	$\boldsymbol{\chi}[r\_e\_a]\ \overline{\mathbf{s}}[r\_e\_a] = \overline{\boldsymbol{\mu}}[Active]/2;$
	$\boldsymbol{\chi}[e\_e\_a]\ \overline{\mathbf{s}}[e\_e\_a] = \overline{\boldsymbol{\mu}}[ExtMemAcc];$
(c_4)	$\boldsymbol{\chi}[e\_o\_a]\ \overline{\mathbf{s}}[e\_o\_a] \leq \overline{\boldsymbol{\mu}}[OwnMemAcc];$
	$\boldsymbol{\chi}[e\_o\_a]\ \overline{\mathbf{s}}[e\_o\_a] \leq \overline{\boldsymbol{\mu}}[Memory];$
(c_6)	$\boldsymbol{\chi}[e\_o\_a]\ \overline{\mathbf{s}}[e\_o\_a] \geq \overline{\boldsymbol{\mu}}[OwnMemAcc] + \frac{\mathbf{b}[OwnMemAcc]}{\mathbf{b}[Memory]}\overline{\boldsymbol{\mu}}[Memory] - \mathbf{b}[Memory];$
(c_7)	$4\left(\overline{\boldsymbol{\mu}}[ExtBus] - \mathbf{b}[ExtBus]\left(1 - \frac{\overline{\boldsymbol{\mu}}[Memory]}{\mathbf{b}[Memory]}\right)\right) \leq 0;$
	$4\left(\overline{\boldsymbol{\mu}}[ExtBus] - \mathbf{b}[ExtBus]\left(1 - \frac{\overline{\boldsymbol{\mu}}[Queue]}{\mathbf{b}[Queue]}\right)\right) \leq 0;$

Table 17.2: Constraints for the model in Figure 17.1.

17.1.3 An Example of Application

Let us present an example of application for the computation of bounds in the case of the TWN model of a shared-memory multiprocessor depicted in Figure 17.1. The architecture comprises a set of processing modules interconnected by a common bus called the "external bus". A processor can access its own memory module directly from its private bus through one port, or it can access non-local shared-memory modules by means of the external bus. In case of contention for the access to one shared-memory module, preemptive priority is given to external access through the external bus with respect to the accesses from the local processor. The experiments on the shared-memory model have been carried out assuming to have 4 processors and that the average service time of all the transitions are equal to 0.5.

According to the arguments presented in the previous sections, bounds can be computed solving linear programming problems with constraints included in Table 17.2, where the first letters of each transition name have been used for reasons of space. The solution for the linear programming problems leads to upper and lower bounds, for the throughput of transitions, given by

$$\frac{8}{11} \leq \boldsymbol{\chi}[e\_e\_a] \leq 2$$

while the "exact" solution with exponential distribution is

$$\boldsymbol{\chi}[e\_e\_a] = 1.71999$$

An improvement in the lower bound can be obtained observing that when a token arrives in place Choice transition choose_m is enabled at least for one transition instance. This implies that the average marking of place Choice is equal to 0 (transition choose_m is immediate), so

$$\overline{\boldsymbol{\mu}}[Choice] = 0; \quad \mathbf{b}[Choice] = 0$$

(only tangible markings are considered) can be added to the set of constraints. Moreover place Memory is implicit with respect to the enabling of transition b_ext_acc, so we can consider this transition as having only two input places, so constraint (c_6) can be applied instead of constraint (c_7). Finally,

$$\mathbf{b}[Queue] = 3$$

can be added since the output transition of place Queue is immediate, and from the behaviour of the model it is clear that at most 3 processors can be waiting in the queue. The relations (c_7) in the above linear programming problem can thus be replaced with the new constraint:

$$4\left(\overline{\boldsymbol{\mu}}[ExtBus] + \frac{\mathbf{b}[ExtBus]}{\mathbf{b}[Queue]}\overline{\boldsymbol{\mu}}[Queue] - \mathbf{b}[ExtBus]\right) \leq 0$$

where $\mathbf{b}[Queue] = 3$. Solving this reduced linear programming problem the values obtained for the upper and lower bounds are:

$$1 \leq \boldsymbol{\chi}[e\_e\_a] \leq 2$$

17.2 Reinterpretation of the Insensitive Bounds for Net Subclasses

This section includes a more intuitive approach for the computation of throughput upper bounds for particular net classes than the general technique presented above. It makes use of an *implicit decomposition* of the net model into P-semiflows. The details of the technique are presented in Section 17.2.1. Section 17.2.2 addresses the relationship between the general technique and this more intuitive approach. Section 17.2.3 shows that the bound is tight for marked graphs.

17.2.1 Little's Law and P-Semiflows

Three of the most significant performance measures for a closed region of a network in the analysis of queueing systems are related by Little's formula: the average number of customers, the output (or input) rate (throughput), and the average time spent by a customer within the region. For the case of timed Petri nets, if the involved *limit average performance indices* exist, then Little's formula can be applied, in particular, to each place of the system as follows (cfr. Chapter 8):

$$\overline{\boldsymbol{\mu}}[p] = (\mathbf{Pre}[p, \cdot] \cdot \boldsymbol{\chi})\overline{\mathbf{r}}[p]$$

where $\mathbf{Pre}[p, \cdot]$ is the row of the pre-incidence matrix of the underlying Petri net corresponding to place p, thus $\mathbf{Pre}[p, \cdot] \cdot \boldsymbol{\chi}$ is the output rate of place p, and $\overline{\mathbf{r}}[p]$ is the limit average token residence time at p.

In the study of computer systems, Little's law is frequently used when two of the related quantities are known and the third one is needed. This is not exactly the case here. In this case, $\overline{\mathbf{r}}[p]$ and $\overline{\boldsymbol{\mu}}[p]$ are unknown, while partial information about $\boldsymbol{\chi}$ can be easily computed only for some (interesting) net subclasses. Let us recall the definition of the *relative throughput vector* or *vector of visit ratios* to transitions (cfr. Chapter 8):

$$\mathbf{v}[t] = \frac{\boldsymbol{\chi}[t]}{\boldsymbol{\chi}[t_1]} = \Gamma[t_1]\ \boldsymbol{\chi}[t]$$

where $\Gamma[t_i] = 1/\boldsymbol{\chi}[t_i]$ represents the *average interfiring time* of transition t_i (i.e., the inverse of its throughput). Here we consider *live and bounded* timed net systems whose vector of visit ratios to transitions can be computed in polynomial time from the net structure $\mathcal{N}$ and from the relative frequency of conflict resolutions $\mathcal{R}$ (i.e., the routing rates associated with decisions). In Chapter 8, a class of net models for which such computation is possible was presented, as well as some interesting subclasses were identified. As an example, let us consider the net system depicted in Figure 17.2. For this net, the vector of visit ratios for transitions can be computed by solving the following linear system of equations:

$$\begin{array}{rcl} \mathbf{C} \cdot \mathbf{v} & = & \mathbf{0}; \\ r_1\mathbf{v}[t_2] & = & r_2\mathbf{v}[t_1]; \\ r_3\mathbf{v}[t_4] & = & r_4\mathbf{v}[t_3]; \\ \mathbf{v}[t_1] & = & \mathbf{1} \end{array} \tag{17.3}$$

where r_1 and r_2 (r_3 and r_4) are the routing rates used for the resolution of the conflict between t_1 and t_2 (respectively, t_3 and t_4). The first set of equations (implying that $\mathbf{v}$ is a T-semiflow) are the *flow balance* equations written for each place (input and output flows of tokens are equal, provided that the places are bounded). The second (third) equation is directly derived from the fact that conflict between t_1 and t_2 (respectively, t_3 and t_4) is free and rates r_1 and r_2 (respectively, r_3 and r_4) are fixed. The fourth equation is the normalization for transition t_1.

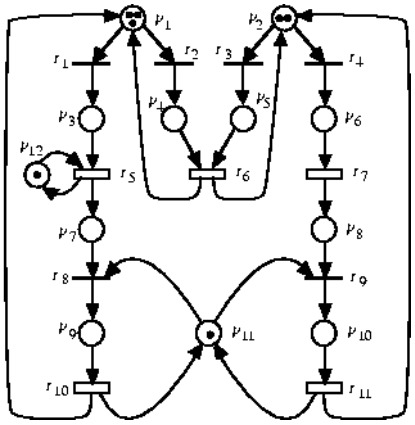


Figure 17.2: A live and bounded stochastic Petri net system.

In Chapter 8, equations like (17.3) have been shown to characterize the vector of visit ratios for important net subclasses such as, for instance, live and bounded *mono-T-semiflow systems* and live and bounded *free choice systems*. Unfortunately, for other subclasses like *simple net systems*, the relative throughput vector also depends on the initial marking and on the service times of transitions.

In the sequel of this chapter (unless otherwise explicitly stated), *we assume that timed transitions may never be in conflict.* For the modelling of conflicts we use immediate transitions with the addition of (marking and time independent) routing rates $\mathcal{R}$. In other words, for the subset of immediate transitions $\{t_1, \ldots, t_k\} \subset T$ being in conflict at each reachable marking, we suppose that the constants $r_1, \ldots, r_k \in \mathbb{R}^+$ are explicitly defined in the system interpretation in such a way that when $t_1, \ldots, t_k$ are simultaneously enabled, transition t_i $(i = 1, \ldots, k)$ fires with relative rate $r_i/(\sum_{j=1}^{k} r_j)$. In this way, routing is completely decoupled from duration of activities. The only restriction that this decoupling imposes to the system is that *preemption* cannot be modelled with two timed transitions (in conflict) competing for the tokens (i.e., *race policy* cannot be modelled; this constraint is equivalent to the use of a *preselection policy* for the resolution of conflicts among timed transitions).

If timed transitions are never in conflict, either all output transitions of a place p are immediate or p has a unique output transition, say t_1, and t_1 is timed. Then, in the later case:

$$\begin{aligned} \overline{\boldsymbol{\mu}}[p] &= (\mathbf{Pre}[p,\cdot] \cdot \boldsymbol{\chi})\overline{\mathbf{r}}[p] = \mathbf{Pre}[p,t_1]\boldsymbol{\chi}[t_1]\overline{\mathbf{r}}[p] \\ &\geq \mathbf{Pre}[p,t_1]\boldsymbol{\chi}[t_1]\overline{\mathbf{s}}[t_1] = \sum_{j=1}^{m} \mathbf{Pre}[p,t_j]\boldsymbol{\chi}[t_j]\overline{\mathbf{s}}[t_j] \end{aligned}$$

The inequality follows from the fact that the residence time $\overline{\mathbf{r}}[p]$ of a token

at place p with only one output transition is greater than or equal to the service time $\overline{\mathbf{s}}[t_1]$ of that transition. So that:

$$\Gamma[t_1]\overline{\boldsymbol{\mu}}[p] \geq \sum_{j=1}^{m} \mathbf{Pre}[p,t_j]\Gamma[t_1]\boldsymbol{\chi}[t_j]\overline{\mathbf{s}}[t_j] = \sum_{j=1}^{m} \mathbf{Pre}[p,t_j]\mathbf{v}[t_j]\overline{\mathbf{s}}[t_j]$$

hence:

$$\Gamma[t_1]\overline{\boldsymbol{\mu}} \geq \mathbf{Pre} \cdot \overline{\mathbf{D}} \tag{17.4}$$

where $\overline{\mathbf{D}}$ is the vector of *average service demands* of transitions, with components:

$$\overline{\mathbf{D}}[t] = \overline{\mathbf{s}}[t]\mathbf{v}[t] \tag{17.5}$$

If all output transitions of place p are immediate, then $\overline{\boldsymbol{\mu}}[p] = \mathbf{Pre}[p,\cdot] \cdot \overline{\mathbf{D}} = 0$, thus inequality (17.4) holds for all place p.

P-semiflows $\mathbf{y}$ are non-negative left annullers of the incidence matrix $\mathbf{C}$ (i.e., $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$), thus $\forall \mathbf{m_0} : \mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0}$ for all reachable marking $\mathbf{m}$. Therefore, $\mathbf{y} \cdot \overline{\boldsymbol{\mu}} = \mathbf{y} \cdot \mathbf{m_0}$. Now, premultiplying by $\mathbf{y}$ the relation (17.4), the following lower bound for the average interfiring time of transition t_1 can be derived:

$$\Gamma[t_1] \geq \max_{\mathbf{y} \in \{P-\text{semiflow}\}} \frac{\mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}}}{\mathbf{y} \cdot \mathbf{m_0}} \tag{17.6}$$

Of course, un upper bound for the throughput of t_1 can be computed taken the inverse. From that bound and from the knowledge of the vector of visit ratios, upper bounds for the throughput of the other transitions can be derived.

Let us formulate the previous lower bound for the average interfiring time of t_1 in terms of a particular class of optimization problems called *fractional programming problems* [26]:

$$\begin{array}{lll} \Gamma[t_1] \geq & \text{maximum} & \dfrac{\mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}}}{\mathbf{y} \cdot \mathbf{m_0}} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & & \mathbf{1} \cdot \mathbf{y} > 0 \\ & & \mathbf{y} \geq \mathbf{0} \end{array} \tag{17.7}$$

where $\mathbf{1}$ is a vector with all entries equal to one. The above problem can be rewritten as follows:

$$\begin{array}{lll} \Gamma[t_1] \geq & \text{maximum} & \dfrac{\mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}}}{q} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & & \mathbf{1} \cdot \mathbf{y} > 0 \\ & & q = \mathbf{y} \cdot \mathbf{m_0} \\ & & \mathbf{y} \geq \mathbf{0} \end{array} \tag{17.8}$$

Then, because $\mathbf{y} \cdot \mathbf{m_0} > 0$ (guaranteed for live systems), we can change $\mathbf{y}/q$ to $\mathbf{y}$ and obtain the linear programming formulation stated in the next theorem (in which $\mathbf{1} \cdot \mathbf{y} > 0$ is removed because $\mathbf{y} \cdot \mathbf{m_0} = 1 \Longrightarrow \mathbf{1} \cdot \mathbf{y} > 0$).

Theorem 17.2 *For any live and bounded system, a lower bound for the average interfiring time $\Gamma[t_1]$ of transition t_1 can be computed by the following linear programming problem:*

$$\begin{array}{lll} \Gamma[t_1] \geq & \textit{maximum} & \mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}} \\ & \textit{subject to} & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & & \mathbf{y} \cdot \mathbf{m_0} = \mathbf{1} \\ & & \mathbf{y} \geq \mathbf{0} \end{array} \tag{17.9}$$

If the solution of the above problem is unbounded and since it is a lower bound for the average interfiring time of transition t_1, the non-liveness can be assured (infinite interfiring time). If the visit ratios of all transitions are non-null (i.e., $\mathbf{v} > \mathbf{0}$), the unboundedness of the above problem implies that a total deadlock is reached by the net system. Anyhow, the unboundedness of the solution of (17.9) means that there exists an unmarked P-semiflow, and obviously the net system is non-live: if $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$ and $\mathbf{y} \cdot \mathbf{m_0} = 0$, then $\forall \mathbf{m}$ $\forall p \in \|\mathbf{y}\|$: $\mathbf{m}[p] = 0$, and the input and output transitions of p are never firable.

The basic advantage of Theorem 17.2 lies, again, in the fact that the *simplex method* for the solution of a linear programming problem has almost linear complexity in practice, even if it has exponential worst case complexity. In any case, algorithms of polynomial worst case complexity can be found in [26].

In order to interpret Theorem 17.2, let us consider again the net system of Figure 17.2. Assuming, for instance, that all routing rates associated with output transitions at conflicts in p_1 and p_2 are equal to one, then the system (17.3) gives $\mathbf{v} = \mathbf{1}$. Therefore, according to (17.5), the vector of average service demands for transitions (normalized for t_1) is $\overline{\mathbf{D}} = (0, 0, 0, 0, \overline{\mathrm{s}}[t_5], \overline{\mathrm{s}}[t_6], \overline{\mathrm{s}}[t_7], 0, 0, \overline{\mathrm{s}}[t_{10}], \overline{\mathrm{s}}[t_{11}])$, because transitions t_1, t_2, t_3, t_4, t_8, and t_9 are assumed to be immediate.

The *minimal* P-semiflows (minimal support solutions of $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}, \mathbf{y} \geq \mathbf{0}$) of this net are:

$$\begin{array}{l} \mathbf{y_1} = (1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0) \\ \mathbf{y_2} = (0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0) \\ \mathbf{y_3} = (0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0) \\ \mathbf{y_4} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1) \end{array} \tag{17.10}$$

and the application of (17.9) gives:

$$\begin{array}{lll} \Gamma[t_1] & \geq & \max\{\ (\overline{\mathrm{s}}[t_5] + \overline{\mathrm{s}}[t_6] + \overline{\mathrm{s}}[t_{10}])/3, \\ & & \quad (\overline{\mathrm{s}}[t_6] + \overline{\mathrm{s}}[t_7] + \overline{\mathrm{s}}[t_{11}])/2, \\ & & \quad \overline{\mathrm{s}}[t_{10}] + \overline{\mathrm{s}}[t_{11}], \\ & & \quad \overline{\mathrm{s}}[t_5]\ \} \end{array} \tag{17.11}$$

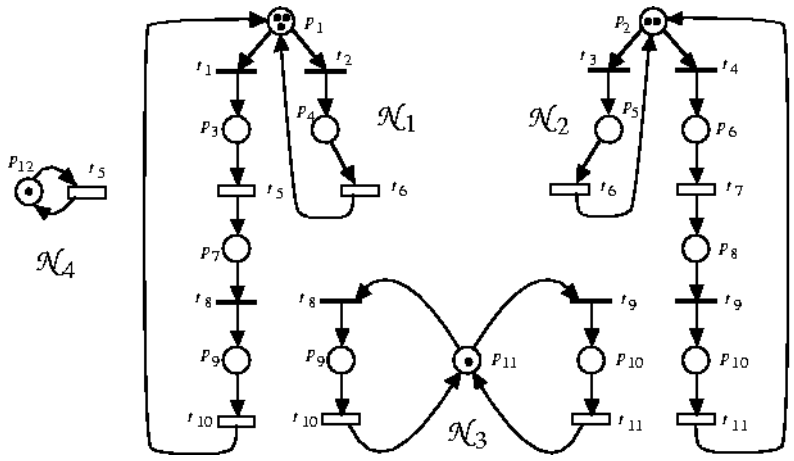


Figure 17.3: Embedded queueing networks of the net in Figure 17.2 generated by minimal P-semiflows.

Now, let us consider the P-semiflow decomposed view of the net: the four *subnets generated by* $\mathbf{y_1}$, $\mathbf{y_2}$, $\mathbf{y_3}$, and $\mathbf{y_4}$ are depicted in isolation in Figure 17.3. Formally speaking, if $\mathbf{y_i}$ is a minimal P-semiflow of a net $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$, the subnet generated by $\mathbf{y_i}$ is $\mathcal{N}_i = \langle P_i, T_i, \mathbf{Pre}_i, \mathbf{Post}_i \rangle$ where $P_i = \|\mathbf{y_i}\|$ (the support of the P-semiflow), $T_i = {}^\bullet P_i \cup P_i^\bullet$ (i.e., the subset of input or output transitions of places belonging to P_i), and $\mathbf{Pre}_i$, $\mathbf{Post}_i$ are the functions $\mathbf{Pre}$, $\mathbf{Post}$ restricted to $P_i \times T_i$.

The quantities under the max operator in (17.11) represent, for this particular case, the average interfiring time of a transition of each of the four subnets (embedded queueing networks) assuming that all the nodes are delay stations (infinite-server semantics). Therefore, the lower bound for the average interfiring time of t_1 in the original net system given by (17.11) is computed *looking at the "slowest subsystem" generated by the P-semiflows*, considered in *isolation* (with delay nodes).

We remark that in this case, since $\mathbf{v} = \mathbf{1}$, the throughput of all transitions is equal and it is not necessary to weight the average interfiring time of transitions computed in isolated subnets in order to get a bound for transition t_1.

17.2.2 Equivalence with the General Statement

In this section, we study the relation between the general technique for the computation of bounds presented in Section 17.1.1 and the particular technique presented in the previous section (cfr. Theorem 17.2). More precisely, we show that the bound given by Theorem 17.2 never improves the bound given by the general technique presented in Section 17.1.1.

Let us consider live systems built on structurally live and structurally bounded nets. With respect to the timing interpretation, we are still consider-

ing infinite-server semantics for transitions and, as in the previous section, we assume that timed transitions may never be in conflict (conflicts resolution is quantified with routing rates associated to immediate transitions).

If $\mathcal{N}$ is a structurally live and structurally bounded net and $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ is a live system, Theorem 17.2 gives the following lower bound for the average interfiring time $\Gamma[t_1]$ (i.e., the inverse of the throughput, $\Gamma[t_1] = 1/\boldsymbol{\chi}[t_1]$) of transition t_1:

$$\begin{array}{rll} \Gamma_1^{\min} = & \text{maximum} & \mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} = \mathbf{0};\ \mathbf{y} \cdot \mathbf{m_0} = \mathbf{1};\ \mathbf{y} \geq \mathbf{0} \end{array} \tag{17.12}$$

where $\overline{\mathbf{D}}$ is the vector of average service demands of transitions, with components $\overline{\mathbf{D}}[t] = \overline{\mathbf{s}}[t]\mathbf{v}[t]$, and $\mathbf{v}$ is the vector of visit ratios to transitions, with components $\mathbf{v}[t] = \frac{\boldsymbol{\chi}[t]}{\boldsymbol{\chi}[t_1]} = \Gamma[t_1]\boldsymbol{\chi}[t]$.

Since the net is structurally live, in particular, it is *structurally repetitive* (i.e., $\exists \mathbf{x} \geq \mathbf{0} :\ \mathbf{C} \cdot \mathbf{x} \geq \mathbf{0}$), thus by *Minkowski-Farkas lemma* [26]:

$$\nexists \mathbf{y} \geq \mathbf{0} :\ \mathbf{y} \cdot \mathbf{C} \leq \mathbf{0} \ \wedge\ \mathbf{y} \cdot \mathbf{C} \neq \mathbf{0}$$

Then, the solution of (17.12) is the same as that of:

$$\begin{array}{rll} \Gamma_1^{\min} = & \text{maximum} & \mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} \leq \mathbf{0};\ \mathbf{y} \cdot \mathbf{m_0} = \mathbf{1};\ \mathbf{y} \geq \mathbf{0} \end{array} \tag{17.13}$$

Since the system is live:

$$\mathbf{y} \neq \mathbf{0} \ \wedge\ \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \implies \mathbf{y} \cdot \mathbf{m_0} \geq \mathbf{1}$$

Then, the solution of (17.13) is the same as that of:

$$\begin{array}{rll} \Gamma_1^{\min} = & \text{maximum} & \mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} \leq \mathbf{0};\ \mathbf{y} \cdot \mathbf{m_0} \leq \mathbf{1};\ \mathbf{y} \geq \mathbf{0} \end{array} \tag{17.14}$$

Let us consider the *dual* linear programming problem[1] [26] of (17.14):

$$\begin{array}{rll} \gamma^* = & \text{minimum} & \gamma \\ & \text{subject to} & \gamma \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \geq \mathbf{Pre} \cdot \overline{\mathbf{D}};\ \gamma \geq 0, \boldsymbol{\sigma} \geq \mathbf{0} \end{array} \tag{17.15}$$

The *Strong Duality Theorem* of linear programming [26] states that if $\Gamma_1^{\min}$ or γ^* is finite, then both (17.14) and (17.15) have finite optimal value and $\Gamma_1^{\min} = \gamma^*$.

On the other hand, let us consider the general statement presented in Section 17.1.1 applied for the computation of a lower bound for the average interfiring time $\Gamma[t_1]$ of transition t_1. If we consider only constraints c_1, c_2', c_3, and c_4, the following system is obtained:

[1] The dual problem of $\max\{\mathbf{c} \cdot \mathbf{x} :\ \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\}$ is $\min\{\mathbf{y} \cdot \mathbf{b} :\ \mathbf{y} \cdot \mathbf{A} \geq \mathbf{c},\ \mathbf{y} \geq \mathbf{0}\}$.

$$
\begin{array}{lll}
\gamma^* = & \text{minimum} & \dfrac{\mathbf{1}}{\boldsymbol{\chi}[t_1]} \\
& \text{subject to} & \overline{\boldsymbol{\mu}} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \\
& & \displaystyle\sum_{t \in {}^\bullet p} \boldsymbol{\chi}[t]\ \mathbf{Post}[p,t] = \sum_{t \in p^\bullet} \boldsymbol{\chi}[t]\ \mathbf{Pre}[p,t],\ \forall p \in P \\
& & \dfrac{\boldsymbol{\chi}[t_i]}{r_i} = \dfrac{\boldsymbol{\chi}[t_j]}{r_j},\ \forall t_i, t_j \in T : \mathbf{Pre}[p,t_i] = \mathbf{Pre}[p,t_j], \forall p \in P \\
& & \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t] \leq \dfrac{\overline{\boldsymbol{\mu}}[p]}{\mathbf{Pre}[p,t]},\ \forall p \in {}^\bullet t,\ \forall t \in T \\
& & \overline{\boldsymbol{\mu}}, \boldsymbol{\chi}, \boldsymbol{\sigma} \geq \mathbf{0}
\end{array}
\tag{17.16}
$$

The first and the fourth sets of constraints in the above problem can be combined to eliminate variables $\overline{\boldsymbol{\mu}}$, leading to:

$$
\begin{array}{lll}
\gamma^* = & \text{minimum} & \dfrac{\mathbf{1}}{\boldsymbol{\chi}[t_1]} \\
& \text{subject to} & \displaystyle\sum_{t \in {}^\bullet p} \boldsymbol{\chi}[t]\ \mathbf{Post}[p,t] = \sum_{t \in p^\bullet} \boldsymbol{\chi}[t]\ \mathbf{Pre}[p,t],\ \forall p \in P \\
& & \dfrac{\boldsymbol{\chi}[t_i]}{r_i} = \dfrac{\boldsymbol{\chi}[t_j]}{r_j},\ \forall t_i, t_j \in T : \mathbf{Pre}[p,t_i] = \mathbf{Pre}[p,t_j], \forall p \in P \\
& & \mathbf{m_0}[p] + \mathbf{C}[p,T] \cdot \boldsymbol{\sigma} \geq \mathbf{Pre}[p,t]\ \boldsymbol{\chi}[t]\ \overline{\mathbf{s}}[t],\ \forall p \in {}^\bullet t,\ \forall t \in T \\
& & \boldsymbol{\chi}, \boldsymbol{\sigma} \geq \mathbf{0}
\end{array}
\tag{17.17}
$$

Now, the first and the second sets of constraints in the above problem state that vector $\boldsymbol{\chi}$ is proportional to the visit ratios vector, i.e., $\boldsymbol{\chi} = \mathbf{v}/\gamma$ (with $\mathbf{v}[t_1] = \mathbf{1}$), then the vector of variables $\boldsymbol{\chi}$ can be reduced to a single variable γ, and by definition of $\overline{\mathbf{D}}$, we get:

$$
\begin{array}{lll}
\gamma^* = & \text{minimum} & \gamma \\
& \text{subject to} & \gamma \mathbf{m_0}[p] + \mathbf{C}[p,T] \cdot \boldsymbol{\sigma} \geq \mathbf{Pre}[p,t]\ \overline{\mathbf{D}}[t],\ \forall p \in {}^\bullet t,\ \forall t \in T \\
& & \gamma \geq 0, \boldsymbol{\sigma} \geq \mathbf{0}
\end{array}
\tag{17.18}
$$

Since we are assuming that timed transitions may never be in conflict, the above problem is equivalent to (17.15). Therefore, the bound (17.12) derived from Theorem 17.2 can be obtained also from the general approach (17.16) presented in Section 17.1.1.

17.2.3 Reachability of the Throughput Upper Bound for Marked Graphs

A particular interesting case of nets whose vector of visit ratios is fixed by the structure is that of marked graphs (MG's) (cfr. Chapter 8). Since MG's are *consistent* nets and their unique minimal P-semiflow is $\mathbf{1}$, their vector of visit

ratios is also **1**; therefore, $\Gamma[t] = \Gamma, \forall t \in T$, Γ is called the *average cycle time* of the MG and it can be bounded (assuming the net is strongly connected thus structurally bounded) by solving the following linear programming problem:

$$\begin{array}{rll} \Gamma \geq & \text{maximum} & \mathbf{y} \cdot \mathbf{Pre} \cdot \bar{\mathbf{s}} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & & \mathbf{y} \cdot \mathbf{m_0} = \mathbf{1} \\ & & \mathbf{y} \geq \mathbf{0} \end{array} \tag{17.19}$$

For deterministically timed nets, the *attainability* of this bound was shown by C. Ramchandani in his Ph.D. dissertation, meaning that the value computed in (17.19) is in fact the actual average cycle time of the MG. Even more, the next result shows that the previous bound cannot be improved only on the basis of the knowledge of the coefficients of variation for transition service times.

Theorem 17.3 *For live strongly-connected MG's with arbitrary values of mean and variance for transition service times, the bound for the average cycle time obtained from (17.19) cannot be improved.*

Proof:
We know from Ramchandani's work [27] that for deterministic timing the bound is reached. Only "max" and sum operators are needed to compute the average cycle time in case of MG's. Therefore, let us construct a family of random variables with arbitrary means and variances behaving in the limit like deterministic timing for both operators (max and sum).

This is the case for the following family of random variables, for varying values of the parameter α $(0 \leq \alpha \leq 1)$:

$$X_{\mu,\sigma}(\alpha) = \begin{cases} \mu\alpha & \text{with probability } 1-\epsilon \\ \mu\left(\alpha + \frac{1-\alpha}{\epsilon}\right) & \text{with probability } \epsilon \end{cases} \tag{17.20}$$

where

$$\epsilon = \frac{\mu^2(1-\alpha)^2}{\mu^2(1-\alpha)^2 + \sigma^2} \tag{17.21}$$

These variables are such that

$$\mathrm{E}[X_{\mu,\sigma}(\alpha)] = \mu; \quad \mathrm{Var}[X_{\mu,\sigma}(\alpha)] = \sigma^2$$

and they satisfy:

$$\lim_{\alpha \to 1} \mathrm{E}[\max(X_{\mu,\sigma}(\alpha), X_{\mu',\sigma'}(\alpha))] = \max(\mu, \mu') \tag{17.22}$$

and, of course,

$$\forall\ 0 \leq \alpha < 1 : \mathrm{E}[X_{\mu,\sigma}(\alpha) + X_{\mu',\sigma'}(\alpha)] = \mu + \mu'$$

Then, if random variables $X_{\bar{\mathbf{s}}[t],\sigma_t}(\alpha)$ are associated with transitions $t \in T$, taking α closer to 1, the average cycle time tends to the bound given by (17.19).
$\diamondsuit$

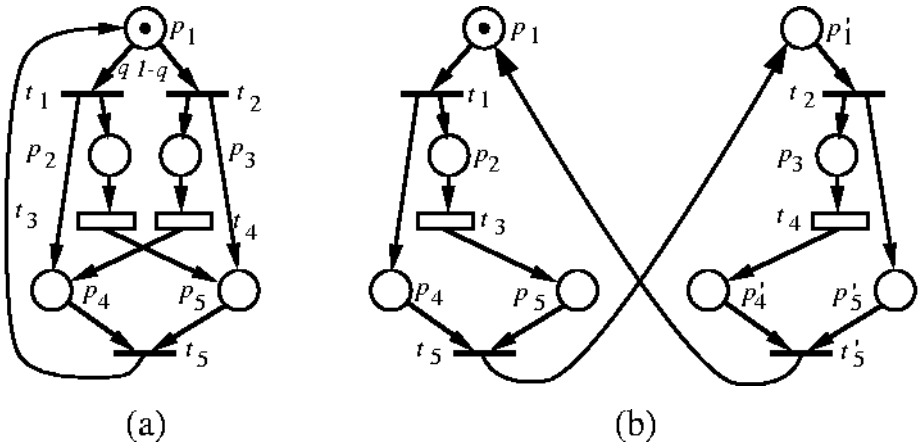


Figure 17.4: (a) A live and safe Free Choice net and (b) its behaviourally equivalent safe marked graph for deterministic resolution of conflict and $q = 1/2$.

We remark that the significance of the above theorem is the attainability of the bound for *any given means and variances* of involved random variables. In other words, even with the knowledge of second order moments, it is not possible to improve the bound given by (17.19), computed only with mean values.

17.3 Structural Improvements of the Insensitive Bounds

This section includes two approaches for improving the throughput bounds presented above. The first of them makes use of *implicit places* to improve the throughput upper bound presented in Section 17.2.1. The basic idea is that the addition of implicit places may increment the number of minimal P-semiflows of the net. Therefore, the computed bound (basically obtained by searching the slowest subsystem among those defined by the minimal P-semiflows) can be improved. The second technique allows to get a tight throughput lower bound for marked graphs, using the *structural liveness bound* of transitions.

17.3.1 The Role of Implicit Places

For strongly connected marked graphs, the bound derived in Theorem 17.2 has been shown to be reachable for arbitrary mean values and coefficients of variation associated with transition service times (cfr. Theorem 17.3). Unfortunately, this is not the case for more general net subclasses. Let us consider, for instance, the live and safe (1–bounded) Free Choice net in Figure 17.4.a. Let $\overline{s}[t_3]$ and $\overline{s}[t_4]$ be the average service times associated with t_3 and t_4, respectively. Let t_1, t_2, and t_5 be immediate transitions (i.e., they fire in zero time). Let

$q, 1-q \in (0,1)$ be the probabilities defining the resolution of conflict between t_1 and t_2. The relative (to t_5, i.e., such that $\mathbf{v}[t_5] = 1$) throughput vector is $\mathbf{v} = (q, 1-q, q, 1-q, 1)$. The elementary P-semiflows are $\mathbf{y_1} = (1,1,0,0,1)$ and $\mathbf{y_2} = (1,0,1,1,0)$. Then, applying the linear programming problem in (17.9) to this net, the following lower bound for the average interfiring time of transition t_5 is obtained:

$$\Gamma[t_5] \geq \max\{q\overline{\mathbf{s}}[t_3], (1-q)\overline{\mathbf{s}}[t_4]\}$$

while the actual interfiring time for this transition is

$$\Gamma[t_5] = q\overline{\mathbf{s}}[t_3] + (1-q)\overline{\mathbf{s}}[t_4]$$

independently of the higher moments of the probability distribution functions associated with transitions t_3 and t_4. Therefore the bound given by Theorem 17.2 is non-reachable for the net in Figure 17.4.a.

As was announced in Section 17.1.1, an improvement of the bound can be obtained if additional constraints that improve the linear characterization of the average marking are considered. Now, we exploit the linear information that can be obtained from *traps* of the net and later we reinterpretate the obtained improvement of the bound in terms of *implicit* places.

A trap in a Petri net $\mathcal{N}$ is a subset of places Θ such that $\Theta^\bullet \subseteq {}^\bullet\Theta$. A well-known property of these structural elements is recalled now.

Property 17.4 *Let $\mathcal{S}$ be a Petri net system and $\Theta \subseteq P$ a trap. If Θ is initially marked, then Θ is marked throughout the net's evolution.*

This property can be expressed in algebraic terms considering the vector $\mathbf{y_\Theta}$ associated with a given trap Θ, and defined as

$$\mathbf{y_\Theta}[p] = \begin{cases} 1, & \text{if } p \in \Theta \\ 0, & \text{otherwise} \end{cases}$$

(i.e., the *characteristic function* of the set Θ). The next inductive invariant is true: if $\mathbf{y_\Theta} \cdot \mathbf{m_0} \geq 1$ then $\mathbf{y_\Theta} \cdot \mathbf{m} \geq 1$ for all reachable marking $\mathbf{m}$.

Let us consider the vector $\mathbf{y_\Theta}$ associated with a trap Θ of a net, and a P-semiflow $\mathbf{y}$ such that $\mathbf{y} - \mathbf{y_\Theta} \geq 0$ (it always exists for conservative nets). The following linear relations can be derived for all reachable marking $\mathbf{m}$ and for the average marking vector $\overline{\boldsymbol{\mu}}$:

$$(\mathbf{y} - \mathbf{y_\Theta}) \cdot \mathbf{m} \leq \mathbf{y} \cdot \mathbf{m_0} - 1$$

$$(\mathbf{y} - \mathbf{y_\Theta}) \cdot \overline{\boldsymbol{\mu}} \leq \mathbf{y} \cdot \mathbf{m_0} - 1$$

Premultiplying inequality (17.4) by $\mathbf{y} - \mathbf{y_\Theta}$, a lower bound for $\Gamma[t_1]$ is derived:

Theorem 17.5 *For any net $\mathcal{N}$ and for any trap Θ of $\mathcal{N}$, a lower bound for the average interfiring time $\Gamma[t_1]$ of transition t_1 is given by:*

$$\Gamma[t_1] \geq \begin{array}{ll} \textit{maximum} & \dfrac{(\mathbf{y}-\mathbf{y_\Theta})\cdot \mathbf{Pre}\cdot \overline{\mathbf{D}}}{\mathbf{y}\cdot\mathbf{m_0}-1} \\ \textit{subject to} & \mathbf{y}\cdot\mathbf{C}=\mathbf{0} \\ & \mathbf{y}-\mathbf{y_\Theta}\geq \mathbf{0} \\ & \mathbf{y_\Theta}[p] = \texttt{ if } p\in\Theta \texttt{ then 1 else } 0 \end{array} \tag{17.23}$$

Going back to the net in Figure 17.4.a, the unique minimal trap different from the P-semiflows is $\Theta = \{p_1, p_4, p_5\}$. Considering the P-semiflow $\mathbf{y} = (2,1,1,1,1)$, we have $\mathbf{y} \geq \mathbf{y_\Theta} = (1,0,0,1,1)$, and Theorem 17.5 can be applied:

$$\Gamma[t_5] \geq q\bar{\mathbf{s}}[t_3] + (1-q)\bar{\mathbf{s}}[t_4] \tag{17.24}$$

Therefore the bound obtained in Section 17.2.1 using only P-semiflows has been improved (in fact the bound computed now coincides with the actual interfiring time for this particular example).

In order to explain in an intuitive way (with the example) the reason of the previous improvement, let us derive a behaviourally equivalent safe marked graph (Figure 17.4.b) for the safe Free Choice net of Figure 17.4.a, assuming for the sake of simplicity that the resolution of conflict at place p_1 is deterministic with $q = 1/2$ (i.e., transitions t_1 and t_2 fire once each one, alternatively). The lower bound for the average cycle time of this MG based on Theorem 17.2 (i.e., using the P-semiflows) is

$$\Gamma_{\mathrm{MG}} \geq \bar{\mathbf{s}}[t_3] + \bar{\mathbf{s}}[t_4]$$

(in fact it is reached) and corresponds to the circuit $\langle p_1, p_2, p_5, p_1', p_3, p_4'\rangle$. Since transition t_5 appears instantiated twice in the MG, the obtained bound for the interfiring time of this transition is

$$\Gamma[t_5] \geq (\bar{\mathbf{s}}[t_3] + \bar{\mathbf{s}}[t_4])/2$$

In the original Free Choice net there does not exist any minimal P-semiflow including both p_2 and p_3 in its support, thus the previous bound is not obtained.

Now, we reinterpretate the linear marking relations derived from traps using *implicit places* (cfr. Chapter 6). Let us consider again the net in Figure 17.4.a and its behaviourally equivalent (for $q = 1/2$) marked graph depicted in Figure 17.4.b. The elementary circuits (P-semiflows) of this MG are

$$\begin{aligned} c_1 &= \langle p_1, p_2, p_5, p_1', p_5'\rangle \\ c_2 &= \langle p_1, p_4, p_1', p_3, p_4'\rangle \\ c_3 &= \langle p_1, p_2, p_5, p_1', p_3, p_4'\rangle \\ c_4 &= \langle p_1, p_4, p_1', p_5'\rangle \end{aligned}$$

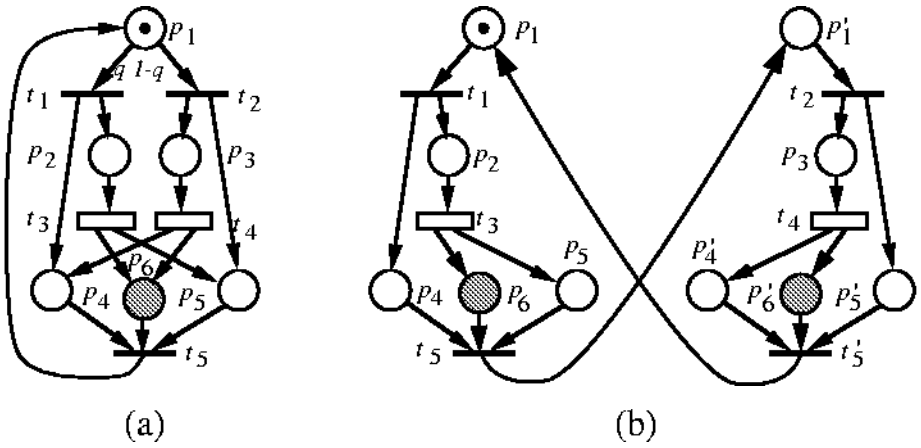


Figure 17.5: The same systems (a) and (b) of Figure 17.4 with the addition of implicit places (a) p_6 and (b) p_6 and p'_6, respectively.

The circuits c_1 and c_2 correspond to the elementary P-semiflows of the original net $\mathbf{y_1}$ and $\mathbf{y_2}$, respectively. Thus, these circuits cannot contribute to the improvement of the bound computed for the original net based on the P-semiflows. This is not the case for the circuits c_3 and c_4. These circuits add linear information which is not reflected by P-semiflows in the original net. The circuit c_4 does not include any timed transition and must not be considered. On the other hand, the circuit c_3 reflects the sequentialization of transitions t_3 and t_4, and it gives the actual cycle time of the net system.

A given elementary circuit of the derived MG does not correspond to any elementary P-semiflow of the original Free Choice net when it includes several instances of a unique transition and each instance has as input (or output) places which are instances of different original places. This is the case, for example, for the circuit c_3 of the MG of Figure 17.4.b. It includes instances t_5 and t'_5 of a unique transition, and the input places of these transitions in circuit c_3 are p_5 and p'_4, respectively, which are instances of different original places.

Now, let us increment the number of circuits of the MG of Figure 17.4.b, by adding the places p_6 and p'_6, as depicted in Figure 17.5.b. Places p_6 and p'_6 are replicas of places p_5 and p'_4, respectively (thus they are implicit), and can be supposed to be different instances of a new (implicit) place in the original net (place p_6 of the net in Figure 17.5.a). The addition of this place generates a new elementary P-semiflow $\mathbf{y_3} = (1,1,1,0,0,1)$. With this P-semiflow, the lower bound for the interfiring time computed with the linear programming problem in (17.9) is

$$\Gamma[t_5] \geq q\overline{\mathbf{s}}[t_3] + (1-q)\overline{\mathbf{s}}[t_4]$$

which is the same obtained in (17.24), using relations derived from trap structures stated in Theorem 17.5.

Let us remark that the relation between the implicit place p_6 of the net in Figure 17.5.a and the trap $\Theta = \{p_1, p_4, p_5\}$ considered previously is straightforward: $\mathbf{C}[p_6, \cdot] = \mathbf{y_\Theta} \cdot \mathbf{C}$, that is, the incidence vector of p_6 is the sum of those of places p_1, p_4, and p_5.

The following linear relation can be derived from the trap Θ (and the P-semiflow $\mathbf{y} = \mathbf{y_1} + \mathbf{y_2}$):

$$(\mathbf{y} - \mathbf{y_\Theta}) \cdot \mathbf{m} = \mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_3] \leq 1, \forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) \qquad (17.25)$$

While this one follows from the new P-semiflow $\mathbf{y_3}$ that includes the implicit place p_6:

$$\mathbf{y_3} \cdot \mathbf{m} = \mathbf{m}[p_1] + \mathbf{m}[p_2] + \mathbf{m}[p_3] + \mathbf{m}[p_6] = 1, \forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) \qquad (17.26)$$

It can be pointed out that the information given by relation (17.25) is included in that given by the new P-semiflow (equation (17.26)), because $\mathbf{m}[p_6] \geq 0$.

Now, technical details related with the addition of implicit places which improve the throughput upper bound computed by means of P-semiflows and traps are considered.

Let us consider an initially marked trap Θ of a given net $\mathcal{N}$, and its associated vector $\mathbf{y_\Theta}$ defined as in previous paragraphs. The following result, which follows from Property 6.1.2 in Chapter 6, assures that a structurally implicit place p_Θ associated with Θ, can be added to $\mathcal{N}$.

Property 17.6 *Let Θ be an initially marked trap of $\mathcal{N}$, $\mathbf{y_\Theta}[p] =$ `if` $p \in \Theta$ `then 1 else` 0, $\mathbf{y_\Theta} \cdot \mathbf{m_0} \geq 1$, and $\mathcal{N}^{p_\Theta}$ the net resulting from the addition of place p_Θ with incidence vector $\mathbf{C}[p_\Theta, \cdot] = \mathbf{y_\Theta} \cdot \mathbf{C}$ to $\mathcal{N}$. Then p_Θ is structurally implicit in $\mathcal{N}^{p_\Theta}$.*

The importance of the previous structural implicit place lies on the fact that, if a marking makes it implicit (e.g., the marking given by Property 6.1.2 in Chapter 6), then the lower bound for the average interfiring time of a transition computed using P-semiflows of the augmented net can improve the bounds based on P-semiflows of the original net (Theorem 17.2) and on the trap Θ (Theorem 17.5). Before to state this result we firstly present a technical lemma.

Lemma 17.7 *Let Θ be an initially marked trap of $\mathcal{N}$ and $\mathbf{y_\Theta}[p] =$ `if` $p \in \Theta$ `then 1 else` 0. Let p_Θ be a place defined as $\mathbf{C}[p_\Theta, \cdot] = \mathbf{y_\Theta} \cdot \mathbf{C}$. Then the pair composed by $\mathbf{y_\Theta}$ and $\mu_\Theta = -1$ is a feasible solution of the linear programming problem of Property 6.1.2 in Chapter 6, and $\mathbf{m_0}[p_\Theta] \leq \mathbf{y_\Theta} \cdot \mathbf{m_0} + \mu_\Theta$ (place p_Θ is assumed to be pure, i.e., selfloop-free).*

Proof:
$\mathbf{y_\Theta} \cdot \mathbf{C} = \mathbf{C}[p_\Theta, \cdot] \Rightarrow \forall t \in {}^\bullet p_\Theta : \mathbf{y_\Theta} \cdot \mathbf{Post}[\cdot, t] - \mathbf{y_\Theta} \cdot \mathbf{Pre}[\cdot, t] = -\mathbf{Pre}[p_\Theta, t]$.

Taking into account that $\mathbf{y_\Theta}$ is the characteristic function of a trap we have in the last equality that $\mathbf{y_\Theta} \cdot \mathbf{Pre}[\cdot, t] > 0$ if and only if $\mathbf{y_\Theta} \cdot \mathbf{Post}[\cdot, t] > 0$. Therefore, $\forall t \in {}^\bullet p_\Theta$: $\mathbf{y_\Theta} \cdot \mathbf{Pre}[\cdot, t] > \mathbf{Pre}[p_\Theta, t]$ and from the linear programming problem of Property 6.1.2 in Chapter 6 we conclude that $\mathbf{y_\Theta}$ and $\mu_\Theta = -1$ is a feasible solution. From the same linear programming problem we also conclude directly that $\mathbf{m_0}[p_\Theta] \leq \mathbf{y_\Theta} \cdot \mathbf{m_0} + \mu_\Theta$ because $\mathbf{y_\Theta} \cdot \mathbf{m_0} \geq 1$. ◊

Theorem 17.8 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system, Θ an initially marked trap of $\mathcal{N}$, $\mathbf{y_\Theta}[p] =$ `if` $p \in \Theta$ `then 1 else 0`, and $\langle \mathcal{N}^{p_\Theta}, \mathbf{m_0}^{p_\Theta} \rangle$ the net system resulting from the addition to the original net of the structural implicit place p_Θ with incidence vector $\mathbf{C}[p_\Theta, \cdot] = \mathbf{y_\Theta} \cdot \mathbf{C}$ and with $\mathbf{m_0}^{p_\Theta}[p_\Theta]$ given by Property 6.1.2 in Chapter 6.*

1. *A lower bound $\Gamma^{p_\Theta}[t_1]$ for the average interfiring time $\Gamma[t_1]$ of transition t_1 in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ can be computed applying Theorem 17.2 to the system $\langle \mathcal{N}^{p_\Theta}, \mathbf{m_0}^{p_\Theta} \rangle$.*
2. *If $\Gamma^{PS}[t_1]$ and $\Gamma^{\Theta}[t_1]$ are the lower bounds of $\Gamma[t_1]$ derived from the direct application of Theorems 17.2 and 17.5, respectively, to the original system, then $\Gamma^{p_\Theta}[t_1] \geq \Gamma^{PS}[t_1]$ and $\Gamma^{p_\Theta}[t_1] \geq \Gamma^{\Theta}[t_1]$.*

Proof:
$\Gamma^{p_\Theta}[t_1]$ is a lower bound for the average interfiring time of t_1 in $\langle \mathcal{N}^{p_\Theta}, \mathbf{m_0}^{p_\Theta} \rangle$ by Theorem 17.2. Since p_Θ is implicit, t_1 has the same average interfiring time in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ and in $\langle \mathcal{N}^{p_\Theta}, \mathbf{m_0}^{p_\Theta} \rangle$. Then, $\Gamma^{p_\Theta}[t_1]$ is a lower bound for the average interfiring time of t_1 in $\langle \mathcal{N}, \mathbf{m_0} \rangle$.

$\Gamma^{p_\Theta}[t_1] \geq \Gamma^{PS}[t_1]$ because if $\mathbf{y}$ is a P-semiflow of $\mathcal{N}$, then $\mathbf{z} = [\mathbf{y} \mid 0]$ is a P-semiflow of $\mathcal{N}^{p_\Theta}$.

Finally, we prove that $\Gamma^{p_\Theta}[t_1] \geq \Gamma^{\Theta}[t_1]$. Let $\mathbf{y}$ be a P-semiflow of $\mathcal{N}$ such that $\mathbf{y} - \mathbf{y_\Theta} \geq \mathbf{0}$. Then $\mathbf{z} = [(\mathbf{y} - \mathbf{y_\Theta}) \mid 1]$ is a P-semiflow of $\mathcal{N}^{p_\Theta}$. Now, applying equation (17.6) for $\Gamma^{p_\Theta}[t_1]$:

$$\begin{aligned} \Gamma^{p_\Theta}[t_1] &\geq \frac{[(\mathbf{y} - \mathbf{y_\Theta}) \mid 1] \cdot \mathbf{Pre}^{p_\Theta} \cdot \overline{\mathbf{D}}}{\mathbf{y} \cdot \mathbf{m_0} - \mathbf{y_\Theta} \cdot \mathbf{m_0} + \mathbf{m_0}^{p_\Theta}[p_\Theta]} = \\ &= \frac{(\mathbf{y} - \mathbf{y_\Theta}) \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}}}{\mathbf{y} \cdot \mathbf{m_0} - \mathbf{y_\Theta} \cdot \mathbf{m_0} + \mathbf{m_0}^{p_\Theta}[p_\Theta]} + \frac{\mathbf{Pre}[p_\Theta, \cdot] \cdot \overline{\mathbf{D}}}{\mathbf{y} \cdot \mathbf{m_0} - \mathbf{y_\Theta} \cdot \mathbf{m_0} + \mathbf{m_0}^{p_\Theta}[p_\Theta]} \end{aligned} \tag{17.27}$$

And this value is greater than or equal to that given by equation (17.23) in Theorem 17.5 because the second term of the above sum is non negative and the first term in the above sum is greater than or equal to that given by equation (17.23) in Theorem 17.5 (take into account that $\mathbf{m_0}^{p_\Theta}[p_\Theta] \leq \mathbf{y_\Theta} \cdot \mathbf{m_0} - 1$, by Lemma 17.7, and that the denominator is less than or equal to $\mathbf{y} \cdot \mathbf{m_0} - 1$). ◊

Theorem 17.8.2 tells that the addition of implicit places allows to obtain better bounds than those computed using traps or the P-semiflows of the original

net. The problems that remain are how to add implicit places (i.e., which ones allow to improve the bounds) and when no more improvements are possible with the technique.

Previously, the net system of Figure 17.4.a has been considered as an example in which the bound computed using the trap $\Theta = \{p_1, p_4, p_5\}$ is tight because it reaches the actual value of the average interfiring time. It is also shown that the same value can be derived, after the addition, in Figure 17.5.a, of the associated implicit place p_6, considering the new P-semiflow $(1, 1, 1, 0, 0, 1)$.

Let us consider the same net system of Figure 17.4.a, but assuming now that transition t_5 is not immediate but timed, with average service time equal to $\bar{\mathbf{s}}[t_5]$. If the linear programming problem in (17.9) is applied to the net, the following bound is obtained:

$$\Gamma^{PS}[t_5] = \max\{q\bar{\mathbf{s}}[t_3] + \bar{\mathbf{s}}[t_5], (1-q)\bar{\mathbf{s}}[t_4] + \bar{\mathbf{s}}[t_5]\}$$

If trap $\Theta = \{p_1, p_4, p_5\}$ and P-semiflow $\mathbf{y} = (2, 1, 1, 1, 1)$ are considered, inequality (17.23) gives the bound:

$$\Gamma^{\Theta}[t_5] = q\bar{\mathbf{s}}[t_3] + (1-q)\bar{\mathbf{s}}[t_4]$$

If the implicit place associated with Θ is added to the net, Theorem 17.8 gives the bound:

$$\Gamma^{p_\Theta}[t_5] = q\bar{\mathbf{s}}[t_3] + (1-q)\bar{\mathbf{s}}[t_4] + \bar{\mathbf{s}}[t_5]$$

(for the P-semiflow $(1, 1, 1, 0, 0, 1)$), which improves both $\Gamma^{PS}[t_5]$ and $\Gamma^{\Theta}[t_5]$, and, in fact, it gives the actual interfiring time of transition t_5 (i.e., it is tight). Note that, in this case, the improvement is due to the non-null second term of the expression (17.27).

17.3.2 The Role of Liveness Bounds of Transitions

In a classical product-form QN, the number of servers at each station is explicitly given as a modelling choice (e.g., it can be said that a certain station has two servers). Stations may vary between *single* server and *delay* node (infinite server). In the second case, the maximum number of servers that can be working at such delay node is exactly the number of customers in the whole net system.

Since in this chapter we assume infinite server semantics for transitions, several instances of a same transition can work in parallel at a given marking. How many of them? The answer is given by the *degree of enabling* of a transition, t, at a given marking, $\mathbf{m}$, defined in Chapter 8 as:

$$\mathbf{e}[t](\mathbf{m}) = \sup\{k \in \mathbb{N} : \forall p \in {}^\bullet t, \mathbf{m}[p] \geq k\, \mathbf{Pre}[p, t]\}$$

Therefore it can be said that at $\mathbf{m}$, in transition t, $\mathbf{e}[t](\mathbf{m})$ servers work in parallel. This value can be eventually reduced by a design choice adding a self-loop place around t with q tokens: it is obvious that in this case $\mathbf{e}[t](\mathbf{m}) \leq q$.

The maximum number of servers working in parallel clearly influences the performance of the system. This value, in net systems terms, has been called the *enabling bound* of a transition.

Definition 17.9 *Let $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system. The enabling bound of a given transition t of $\mathcal{S}$ is*

$$\mathbf{eb}[t] = \sup\{k \in \mathbb{N} : \exists\ \mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}, \forall p \in {}^\bullet t, \mathbf{m}[p] \geq k\ \mathbf{Pre}[p,t]\} \qquad (17.28)$$

The enabling bound is a quantitative generalization of the basic concept of enabling, and is closely related to the concept of *marking bound of a place* (see Chapter 6).

Since we are interested in the steady-state performance of a model, one can ask the following question: how many servers can be available in transitions in any possible steady-state condition? The answer is given by the definition of the *liveness bound* concept.

Definition 17.10 *Let $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system. The liveness bound of a given transition t of $\mathcal{S}$ is*

$$\mathbf{lb}[t] = \sup\{k \in \mathbb{N} : \forall \mathbf{m}', \mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}', \exists \mathbf{m}, \mathbf{m}' \xrightarrow{\sigma'} \mathbf{m} \wedge \forall p \in {}^\bullet t, \mathbf{m}[p] \geq k\ \mathbf{Pre}[p,t]\} \qquad (17.29)$$

The above definition generalizes the classical concept of liveness of a transition. In particular, a transition t is live if and only if $\mathbf{lb}[t] > 0$, i.e., if there is at least one working server associated with it in any steady-state condition. The following is also obvious from the definitions.

Property 17.11 *Let $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system. For any transition t in $\mathcal{S}$,* $\mathbf{eb}[t] \geq \mathbf{lb}[t]$.

The definition of enabling bound refers to a behavioural property. Since we are looking for computational techniques at the structural level, we define also the structural counterpart of the enabling bound concept. Essentially, the reachability condition is substituted by the (in general) weaker (linear) constraint that markings satisfy the net state equation: $\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$, with $\mathbf{m}, \boldsymbol{\sigma} \geq \mathbf{0}$.

Definition 17.12 *Let $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system. The structural enabling bound,* $\mathbf{seb}[t]$*, of a given transition t of $\mathcal{S}$ is*

$$\begin{array}{lll} \mathbf{seb}[t] = & \textit{maximum} & k \\ & \textit{subject to} & \mathbf{m_0}[p] + \mathbf{C}[p,T] \cdot \boldsymbol{\sigma} \geq k\ \mathbf{Pre}[p,t],\ \forall p \in P \\ & & \boldsymbol{\sigma} \geq \mathbf{0} \end{array} \qquad (17.30)$$

Note that the definition of structural enabling bound reduces to the formulation of a linear programming problem, that can be solved in polynomial time.

Now let us remark the relation between behavioural and structural enabling bound concepts that follows from the implication "$\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m} \Rightarrow \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \sigma \wedge \sigma \geq \mathbf{0}$".

Property 17.13 *Let $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system. For any transition t in $\mathcal{S}$,* $\mathbf{seb}[t] \geq \mathbf{eb}[t]$.

As we remarked before, the concept of enabling bound of transitions is closely related to the marking bound of places. In an analogous way, the structural enabling bound is closely related to the structural marking bound of places (see Chapter 6).

For the particular case of live and bounded free choice systems (thus, in particular, for live and bounded marked graphs), the following result holds.

Theorem 17.14 *Let $\mathcal{S} = \langle \mathcal{N}, \mathbf{m_0} \rangle$ be a live and bounded free choice system. For any transition t in $\mathcal{S}$,* $\mathbf{seb}[t] = \mathbf{eb}[t] = \mathbf{lb}[t]$.

A "trivial" lower bound in steady-state performance for a live net system with a given vector of visit ratios for transitions is of course given by the inverse of the sum of the services times of all the transitions weighted by the vector of visit ratios. Since the net system is live, all transitions must be firable, and the sum of all service times multiplied by the number of occurrences of each transition in the average cycle of the model corresponds to any *complete sequentialization* of all the transition firings.

Theorem 17.15 *For any live and bounded system, an upper bound for the average interfiring time $\Gamma[t_1]$ of transition t_1 can be computed as follows:*

$$\Gamma[t_1] \leq \sum_{t \in T} \mathbf{v}[t]\bar{\mathbf{s}}[t] = \sum_{t \in T} \overline{\mathbf{D}}[t] \tag{17.31}$$

This *pessimistic* behaviour is always reached in a marked graph consisting on a single loop of transitions and containing a single token in one of the places, independently of the higher moments of the probability distribution functions (this observation can be trivially confirmed by the computation of the lower bound given by (17.19), which in this case gives the same value).

Before trying to improve this trivial bound let us first consider the case of 1–live (i.e., all its transitions have a liveness bound equal to 1) strongly-connected MG's. If we specify only the mean values of the transition service times and not the higher moments, we may always find an stochastic model whose steady-state throughput is arbitrarily close to the trivial lower bound, independently of the topology of the MG (only provided that it is 1–live). The formal proof of this (somewhat counter-intuitive) result stated in the next theorem is based on the definition of the family of random variables:

$$X_\mu^i(\epsilon) = \begin{cases} 0, & \text{with probability } 1-\epsilon^i \\ \mu/\epsilon^i, & \text{with probability } \epsilon^i \end{cases} \tag{17.32}$$

for $\mu \geq 0; 0 < \epsilon \leq 1; i \in \mathbb{N}$. It is straightforward to see that

$$\mathrm{E}\left[X_\mu^i(\epsilon)\right] = \mu \quad \text{and} \quad \mathrm{E}\left[X_\mu^i(\epsilon)^2\right] = \mu^2/\epsilon^i$$

This implies that the coefficient of variation is 0 for $\epsilon = 1$, and that it tends to ∞ as $\epsilon \to 0$ provided that $i > 0$. Then, the proof derives from considering that each transition t_j in the MG has $X_{\bar{\mathbf{s}}[t_j]}^{j-1}(\epsilon)$ as random service time distribution.

Theorem 17.16 *For any 1–live strongly-connected MG with a given specification of the average service times $\bar{\mathbf{s}}[t_j]$ for each $t_j \in T$, it is possible to assign probability distribution functions to the transition service times such that the average cycle time is $\Gamma = \sum_j \bar{\mathbf{s}}[t_j] - O(\epsilon)$, $\forall \epsilon : 0 < \epsilon \leq 1$, independently of the topology of the net (and thus independently of the potential maximum degree of parallelism intrinsic in the MG)* [2].

Proof:
By construction, we will show that the association of the family of random variables $X_{\bar{\mathbf{s}}[t_j]}^{j-1}(\epsilon)$, defined in (17.32), with each transition $t_j \in T$ yields exactly the cycle time Γ claimed by the theorem. To give the proof, we will consider a sequence of models ordered by the index of transitions, in which the qth model of the sequence has transitions $t_1, t_2, \ldots, t_q$ timed with the random variables $X_{\bar{\mathbf{s}}[t_j]}^{j-1}(\epsilon)$, and all other transitions immediate (firing in zero time); the $|T|$th model in the sequence represents an example of attainability of the lower bound on throughput (upper bound for the average cycle time), independent of the net topology. Now we will prove by induction that the qth model in the sequence has a cycle time $\Gamma_q = \sum_{j=1}^{q} \bar{\mathbf{s}}[t_j] - O(\epsilon)$.

Base ($q = 1$): trivial since the repetitive cycle that constitutes the steady-state behaviour of the MG contains only one (single-server) deterministic transition with average service time $\Gamma_1 = \bar{\mathbf{s}}[t_1]$.

Induction step ($q > 1$): taking the limit $\epsilon \to 0$, the newly timed transition t_q will fire most of the time with time zero, thus normally not contributing to the computation of the cycle time, that will be just $\Gamma_{q-1} = \sum_{j=1}^{q-1} \bar{\mathbf{s}}[t_j] - O(\epsilon)$ (as in the case of model $q-1$) with probability $1-\epsilon^{q-1}$. On the other hand, the newly timed transition has a (very small) probability ϵ^{q-1} of delaying its firing by a time $\bar{\mathbf{s}}[t_q]/\epsilon^{q-1}$, which is at least of order $1/\epsilon$ bigger than any other service time in the circuit, so that in this case all other transitions will wait for the firing of t_q, after having completed their possible current service in a time which is $O(\epsilon)$ lower than the service time of t_q itself (i.e., $\bar{\mathbf{s}}[t_q]/\epsilon^{q-1} = \Gamma_{q-1}/O(\epsilon)$). Therefore we obtain that $\Gamma_q = (1-\epsilon^{q-1})\Gamma_{q-1} + \epsilon^{q-1}(\frac{\bar{\mathbf{s}}[t_q]}{\epsilon^{q-1}} - O(\epsilon)) = \sum_{j=1}^{q} \bar{\mathbf{s}}[t_j] - O(\epsilon)$. $\diamondsuit$

[2] We use here the notation $O(f(x))$ to indicate any function $g(x)$ such that $\lim_{x\to 0} \frac{g(x)}{f(x)} \leq k \in \mathbb{R}$.

In the previous result, the upper bound for the average cycle time (thus lower bound on throughput) is reached in a limit case ($\epsilon \to 0$) in which the random variables associated with transitions have infinite coefficient of variation. This is a way to obtain the minimum throughput if service times associated with transitions are assumed mutually uncorrelated. It can be shown that it is also possible to reach the lower bound in performance for finite coefficient of variation if a *maximum negative correlation* is assumed among the service times of transitions.

Until now, we have shown that the trivial sum of the average service times of all transitions in the system constitutes a tight (attainable) lower bound for the performance of a live and safe MG (or more generally of a 1–live strongly-connected MG, but otherwise independently of the topology) in which only the mean values and neither the probability distribution functions nor the higher moments are specified for the transition service times. Let us now extend this result to the more general case of k–live strongly-connected MG's.

An intuitive idea is to derive a lower bound on throughput for an MG containing transitions with liveness bound $k \geq 1$ (remember that, for MG's, $\mathbf{lb} = \mathbf{seb}$) by taking the method used for the computation of the throughput upper bound in Section 17.2.3, and substitute in it the "max" operator for the sum of the service times of all transitions involved. After some manipulation to avoid counting more than once the contribution of the same transition, one can arrive at the formulation of the following value for the maximum cycle time:

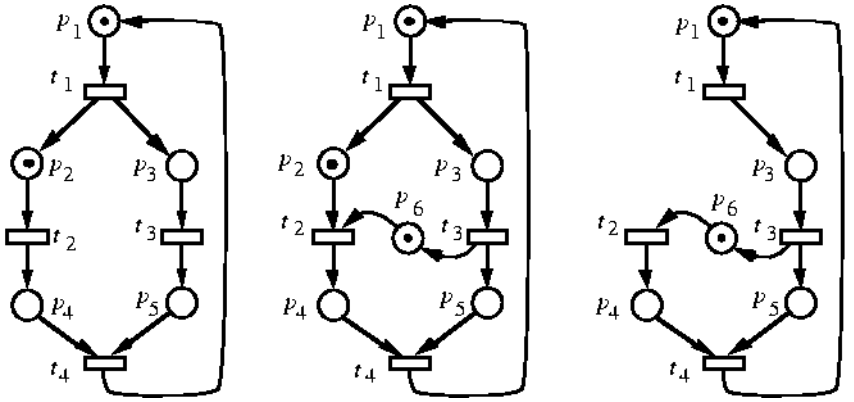
$$\Gamma \leq \sum_{t \in T} \frac{\bar{\mathbf{s}}[t]}{\mathbf{seb}[t]} \tag{17.33}$$

The proof of this result requires the following Lemma.

Lemma 17.17 *Any strongly-connected MG with arbitrary initial marking can be constrained to contain a main circuit including all transitions, without changing their liveness bound. This main circuit (which, in general, is not unique) contains a number of tokens equal to the maximum of the liveness bounds among all transitions. In addition there are other minor circuits that preserve the liveness bounds for transitions with bound lower than the maximum.*

Proof:
To construct an MG of the desired form we can apply the following iterative procedure that interleaves two non-disjoint circuits into a single one. Since the MG is strongly-connected each node belongs to at least one circuit; moreover, since the original MG is finite and each circuit cannot contain the same node more than once, this circuit interleaving procedure must terminate after a finite number of iterations. To reduce the number of circuits, implicit places created after each iteration can be removed. The iteration step is the following:

1. Take two arbitrary non-disjoint circuits (unless the MG already contains a main circuit including all nodes, there always exists such a pair of circuits because the MG is strongly-connected).



a) Original net with t_2 and t_3 concurrent.

b) Transformed net: t_2 and t_3 sequentialized and p_2 made implicit.

c) Elimination of implicit places. Main loop: p_1, p_3, p_6, p_4. Minor cycle: p_1, p_3, p_5.

Figure 17.6: Example of structural sequentialization.

2. Combine them in a single circuit in such a way that the partial order among transitions given by the two original circuits is substituted by a compatible but otherwise arbitrary total order. This combination can be obtained by adding new places that are connected as input for a transition of one circuit and output for a transition of the other circuit that we decide must follow in the sequence determined by the new circuit we are creating.

3. Mark the new added places in such a way that the new circuit contains the same number of tokens as the maximum of the number of tokens in the two original circuits.

The above procedure is applied iteratively until all transitions are constrained into a single main circuit. At this point, we can identify and eliminate the implicit places that have been created during the circuits interleaving procedure. We obtain then an MG composed of one main circuit containing $c = \max_{t \in T} \mathbf{seb}[t]$ tokens that connects all transitions, and a certain number of minor circuits containing less tokens than c that maintain the liveness bound of the other transitions. ◊

The idea behind this constraint is to introduce a structural sequentialization among all transitions, thus potentially reducing the degree of concurrency between the activities modeled by the transitions. In other words, from the partial order given by the initial MG structure, we try to derive a total order without changing the liveness bound.

An example of application of the Lemma follows, in order to clarify the procedure. Consider the system depicted in Figure 17.6.a. This system contains only two circuits, namely $\langle t_1, t_2, t_4 \rangle$, and $\langle t_1, t_3, t_4 \rangle$; we can then add either the

circuit $\langle t_1, t_2, t_3, t_4 \rangle$ or $\langle t_1, t_3, t_2, t_4 \rangle$; Figure 17.6.b depicts the resulting system in case we choose to add the second circuit. In this case only place p_6 (from t_3 to t_2) needs to be added to obtain the longer circuit, and it should be marked with one token, so that the new circuit comprising places $\langle p_1, p_3, p_6, p_4 \rangle$ contains two tokens, as the original circuit $\langle p_1, p_2, p_4 \rangle$ (while the other original circuit $\langle p_1, p_3, p_5 \rangle$ contained only one). In our example, we need not iterate the procedure since we have already obtained a circuit containing all transitions of the MG. At this point we can identify and eliminate the implicit places that have been created during the circuits interleaving procedure. In the present example, we can easily see that place p_2 becomes implicit in Figure 17.6.b, so that it can be eliminated, finally leading ourselves to the MG depicted in Figure 17.6.c.

It should be evident that the MG transformed by applying the above Lemma has an average cycle time which is greater than or equal to the average cycle time of the original one, since some additional constraints have been added to the enabling of transitions: hence the average cycle time of the transformed MG is a lower bound for the performance of the original one. Now if $c = \max_{t \in T} \mathbf{seb}[t] = 1$ in the above Lemma, we re-find the lower bound of Theorem 17.16. In the case of $c > 1$, we can show that the average cycle time of the transformed system cannot exceed $\sum_{t \in T} \bar{\mathbf{s}}[t]/\mathbf{seb}[t]$.

Theorem 17.18 *For any live and bounded marked graph, an upper bound for the average cycle time Γ can be computed as follows:*

$$\Gamma \leq \sum_{t \in T} \frac{\bar{\mathbf{s}}[t]}{\mathbf{seb}[t]} \tag{17.34}$$

Moreover, this upper bound for the average cycle time is reachable for any MG topology and for some assignment of probability distribution functions to the service time of transitions (i.e., the bound cannot be improved).

Proof:
Without loss of generality, assume that transitions in the system resulting from the application of Lemma 17.17 are partitioned in two classes S_2 and S_1, with liveness bounds $K_2 = c > 1$ and $K_1 < c$ (where $c = \max_{t \in T} \mathbf{seb}[t]$), respectively (the proof is easily extended to the case of more than two classes). Construct a new model containing only K_1 tokens in the main circuit; at this point all transitions behave as K_1–servers, so that the cycle time is given by the sum of the service times of all transitions, divided by the total number of customers in the main loop K_1; moreover, the delay time for the transitions belonging to class S_1 is simply given by $D_1 = \sum_{t \in S_1} \bar{\mathbf{s}}[t]$. Now if we increase the number of tokens in the main loop from K_1 to K_2, the delay time of S_1 cannot increase, so that the contribution of S_1 to the cycle time cannot exceed D_1 for each of the first K_1 tokens. Under the hypothesis that the throughput of the system $\boldsymbol{\chi}[t_1]$ is given by the inverse of $\sum_{t \in T} \bar{\mathbf{s}}[t]/\mathbf{seb}[t]$, the average number of tokens of the main loop computed using Little's formula cannot exceed $N_1 = \boldsymbol{\chi}[t_1] D_1$, therefore the average number of tokens available to fire transitions in S_2 cannot be lower than

$$N_2 = K_2 - N_1 = K_2 \frac{\frac{K_2-K_1}{K_1}\sum_{t\in S_1}\overline{\mathbf{s}}[t] + \sum_{t\in S_2}\overline{\mathbf{s}}[t]}{\sum_{t\in S_2}\overline{\mathbf{s}}[t] + \frac{K_2}{K_1}\sum_{t\in S_1}\overline{\mathbf{s}}[t]}$$

On the other hand, we need only

$$N_2 = \boldsymbol{\chi}[t_1]D_2 = K_2 \frac{D_2}{\sum_{t\in S_2}\overline{\mathbf{s}}[t] + \frac{K_2}{K_1}\sum_{t\in S_1}\overline{\mathbf{s}}[t]}$$

tokens to sustain throughput $\boldsymbol{\chi}[t_1]$ in subnet S_2, so that we are assuming a delay in S_2:

$$D_2 \leq \frac{K_2 - K_1}{K_1}\sum_{t\in S_1}\overline{\mathbf{s}}[t] + \sum_{t\in S_2}\overline{\mathbf{s}}[t]$$

Now we claim that this is the actual maximum delay because the first K_1 tokens can proceed at the maximum speed in the whole system, thus experiencing only delay $\sum_{t\in S_2}\overline{\mathbf{s}}[t]$ in subnet S_2, while the remaining $K_2 - K_1$ tokens can also queue up for traveling through S_1, thus experiencing an additional delay of $\frac{1}{K_1}\sum_{t\in S_1}\overline{\mathbf{s}}[t]$ each.

With respect to the reachability of the bound, we proceed by construction, in a way very similar to that of Theorem 17.16. The only technical difference is that now, without any loss of generality, we assume first of all to enumerate transitions in non-increasing order of liveness bound (or, equivalently, of structural enabling bound) i.e., rename the transitions in such a way that $\forall t_i, t_j \in T$, $i > j \implies \mathbf{seb}[t_i] \leq \mathbf{seb}[t_j]$. Then, as in the case of Theorem 17.16, we can show that the association of the family of random variables $X^{j-1}_{\overline{\mathbf{s}}[t_j]}(\epsilon)$ with each transition $t_j \in T$ yields exactly the bound for the cycle time claimed by the theorem. To give the proof we consider a sequence of models ordered by the index of transitions, in which the qth model of the sequence has transitions $t_1, t_2, \ldots, t_q$ timed with the random variables $X^{j-1}_{\overline{\mathbf{s}}[t_j]}(\epsilon)$, and all other transitions immediate (firing in zero time); the $|T|$th model in the sequence represents the resulting model that is expected to provide the example of attainability of the lower bound. By induction we prove that the qth model in the sequence has a cycle time:

$$\Gamma_q = \sum_{j=1}^{q} \frac{\overline{\mathbf{s}}[t_j]}{\mathbf{seb}[t_j]} - O(\epsilon)$$

Base ($q = 1$): trivial since the repetitive cycle that constitute the steady-state behavior of the MG contains only one ($\mathbf{seb}[t_1]$–server) deterministic transition with average service time $\Gamma_1 = \overline{\mathbf{s}}[t_1]/\mathbf{seb}[t_1]$.

Induction step ($q > 1$): taking the limit $\epsilon \to 0$, each server of the newly timed transition t_q will fire most of the times with time zero, thus normally not disturbing the behavior of the other timed transitions, and not contributing to

the computation of the cycle time, that will be just $\Gamma_q = \sum_{j=1}^{q-1} \frac{\bar{\mathbf{s}}[t_j]}{\mathbf{seb}[t_j]} - O(\epsilon)$ (as in the case of model $q-1$) with probability $1-\epsilon^{q-1}$. On the other hand, each of the servers of the newly timed transition has a (very small) probability ϵ^{q-1} of delaying its firing of a time $\bar{\mathbf{s}}[t_q]/\epsilon^{q-1}$, which is at least order of $1/\epsilon$ bigger than any other service time in the circuit. Now if $\mathbf{seb}[t_q] = 1$, then the proof is completed, since also $\forall j > q$, $\mathbf{seb}[t_j] = 1$ by hypothesis, and we reduce to the induction step of the proof of Theorem 17.16. Instead if $\mathbf{seb}[t_q] > 1$ then we can consider $\mathbf{seb}[t_q]$ consecutive firings of t_q, and compute the average service time as the total time to fire $\mathbf{seb}[t_q]$ times the transition, divided by $\mathbf{seb}[t_q]$. Now if we consider m consecutive firings of instances of transition t_q, we obtain an average delay:

$$\sum_{j=0}^{m-1} (1-\epsilon^{q-1})^j \epsilon^{(q-1)(m-j)} \frac{(m-j)\bar{\mathbf{s}}[t_q]}{\epsilon^{(q-1)}} = \bar{\mathbf{s}}[t_q](1+O(\epsilon))$$

Therefore, the average cycle time of the qth model will be:

$$\Gamma_q = (1-O(\epsilon^{q-1}))\Gamma_{q-1} + \frac{\bar{\mathbf{s}}[t_q]}{\mathbf{seb}[t_q]}(1+O(\epsilon)) = \sum_{j=1}^{q} \frac{\bar{\mathbf{s}}[t_j]}{\mathbf{seb}[t_j]} - O(\epsilon).$$

$\diamondsuit$

The same idea for the improvement of the lower bound based on liveness bounds of transitions that has been presented for marked graphs can be applied for live and bounded Free Choice systems in order to improve the trivial bound of Theorem 17.15.

Theorem 17.19 *For any live and bounded Free Choice system, an upper bound for the average interfiring time $\Gamma[t_1]$ of transition t_1 can be computed as follows:*

$$\Gamma[t_1] \leq \sum_{t \in T} \frac{\mathbf{v}[t]\bar{\mathbf{s}}[t]}{\mathbf{seb}[t]} = \sum_{t \in T} \frac{\overline{\mathbf{D}}[t]}{\mathbf{seb}[t]} \tag{17.35}$$

Proof:
Let us consider a deterministic conflicts resolution policy. A strongly connected MG with the same relative throughput vector can be constructed as follows (in fact, since for the MG $\mathbf{v} = \mathbf{1}$, what can appear are several instances of transitions to get the $\mathbf{v}$ of the original net):

1. Steady-state markings must be home states. Let $\mathbf{m_h}$ be one of the home states (there always exist some for live and bounded free choice systems, and substitute it to the initial marking (i.e., $\langle \mathcal{N}, \mathbf{m_h} \rangle$ is reversible).

2. From the live and bounded free choice system, a safe marking can be derived preserving liveness, removing tokens from $\mathbf{m_h}$.

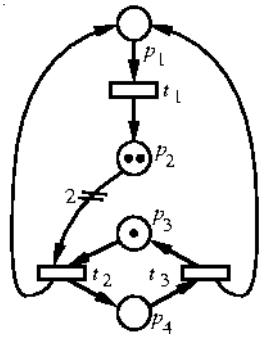


Figure 17.7: "Non-trivial" upper bound for the average interfiring time cannot be applied.

3. Develop the process, resolving the conflicts with the deterministic given policy, until cyclicity appears and the relative firing frequency holds. A safe MG is obtained in which transitions appear according to their relative firing frequencies.

4. The rest of tokens at each place in $\mathbf{m_h}$ in the original live and bounded free choice system can be added now in the corresponding place of the MG.

The actual interfiring time of the original free choice system (with deterministic conflicts resolution policy) is less than or equal to the one of the derived MG because the behaviour of the system has been constrained. Now, apply the bound obtained in Theorem 17.18. Different instances of a given transition are considered in the relative rate of the corresponding component in the relative firing frequency vector. Thus, the bound obtained for the derived MG applying Theorem 17.18 coincides with the bound obtained for the original system using the formula stated in this theorem. The theorem follows because $\mathbf{lb} = \mathbf{seb}$ for live and bounded free choice systems. ◊

Concerning non-free choice systems, only the trivial bound, given by the sum of the average service times of all transitions weighted by the vector of visit ratios, can be computed.

An example showing that the bound presented in Theorem 17.19 is not valid for non-free choice systems is depicted in Figure 17.7, where $\bar{\mathbf{s}}[t_1], \bar{\mathbf{s}}[t_2], \bar{\mathbf{s}}[t_3]$ are the average service times of transitions t_1, t_2, t_3, respectively. For this system, the vector of visit ratios normalized for transition t_2 (i.e., such that $\mathbf{v}[t_2] = 1$) is $\mathbf{v} = (2, 1, 1)$ and the liveness bounds of transitions are given by $\mathbf{lb}[t_1] = 2$, $\mathbf{lb}[t_2] = 1$, and $\mathbf{lb}[t_3] = 1$. Thus, the Theorem 17.19 would give the bound:

$$\Gamma[t_2] \leq \bar{\mathbf{s}}[t_1] + \bar{\mathbf{s}}[t_2] + \bar{\mathbf{s}}[t_3] \tag{17.36}$$

If exponentially distributed random variables (with means $\bar{\mathbf{s}}[t_1], \bar{\mathbf{s}}[t_2], \bar{\mathbf{s}}[t_3]$;

$\bar{\mathbf{s}}[t_1] \neq \bar{\mathbf{s}}[t_3]$) are associated with transitions, the average interfiring time for transition t_2 is

$$\Gamma[t_2] = \bar{\mathbf{s}}[t_1] + \bar{\mathbf{s}}[t_2] + \bar{\mathbf{s}}[t_3] + \frac{\bar{\mathbf{s}}[t_1]^2}{2(\bar{\mathbf{s}}[t_1] + \bar{\mathbf{s}}[t_3])} \tag{17.37}$$

which is greater than the value obtained from the Theorem 17.19, thus the "non-trivial" bound does not hold in general.

17.4 Additional Improvements: Non-Insensitive Bounds

A brief overview of three additional techniques for the improvement of the bounds presented before are included in this section. The common factor to these new techniques is that all of them need additional assumptions on the form of the probability distribution functions associated to the service of transitions or on the conflict resolution policies. In this sense, the obtained bounds are non-insensitive.

17.4.1 Linear Relations between Second Order Moments

In this section, some linear equations are derived between second order moments of marking of places for the particular case of *Exponential Petri Nets* (EPN). In this context, EPN are timed PN such that:

- all the transitions have (independent) exponential service time (in particular, immediate transitions are not allowed);
- a single-server semantics is assumed for transitions (or, an infinite-server semantics but with the assumption that every transition of the net has a self-loop place with multiplicity one and initially marked with one token);
- a race policy is assumed for the firing of transitions.

The obtained linear equations can be added as new constraints to the linear programming problem (17.1) of Section 17.1.1, therefore, the bounds for linear functions of average marking and throughput presented there can eventually be improved.

In order to get the new linear relations, we apply the *uniformization technique* [21] to the stochastic marking process $\{\overline{\boldsymbol{\mu}}(\tau)\}_{\tau \geq 0}$ (a more detailed discussion can be found in [24]).

Consider that each transition $t \in T$ of the EPN is continuously working with independent and exponentially distributed service time of rate $\lambda_t = 1/\bar{\mathbf{s}}[t]$. When a service at transition t is completed at instant τ there are two possibilities:

- either $\mathbf{e}[t](\tau) = 1$, i.e., t is enabled at instant τ and the completion of the service corresponds with a (real) firing of the transition; or

- $\mathbf{e}[t](\tau) = 0$, i.e., t is disabled at instant τ therefore nothing happens and we say that a *fictive firing* occurs.

Let $\{\tau_n\}_{n>0}$ be the sequence of time epochs of real or fictive service completions in the EPN. Then $\{\tau_n\}_{n>0}$ is a Poisson process with parameter $\lambda = \sum_{t\in T} \lambda_t$.

Denote $A_t(n)$ the indicator function defined as:

$$A_t(n) = \begin{cases} 1, & \text{if the } n\text{th real or fictive service completion occurs at } t \in T \\ 0, & \text{otherwise} \end{cases}$$

Then, $\sum_{t\in T} A_t(n) = 1$ and, for any $t \in T$, $\{A_t(n)\}_{n>0}$ is a sequence of independent and identically distributed random variables, independent of $\{\overline{\boldsymbol{\mu}}(\tau_n)\}_{n>0}$, and such that $\Pr\{A_t(n) = 1\} = \lambda_t/\lambda$.

If the system is in steady state then *PASTA property* (Poisson process see time average) [1] holds and in order to study the limit expected value $\overline{\boldsymbol{\mu}}$ of $\{\overline{\boldsymbol{\mu}}(\tau)\}_{\tau\geq 0}$, defined as:

$$\overline{\boldsymbol{\mu}}[p] = \lim_{\tau\to\infty} \mathrm{E}\,[\overline{\boldsymbol{\mu}}[p](\tau)]$$

it is enough to analyse the process $\{\overline{\boldsymbol{\mu}}(\tau_n)\}_{n>0}$.

First, notice that the evolution of $\{\overline{\boldsymbol{\mu}}(\tau_n)\}_{n>0}$ is determined by the following equation:

$$\overline{\boldsymbol{\mu}}[p](\tau_{n+1}) = \begin{cases} \overline{\boldsymbol{\mu}}[p](\tau_n), & \text{if } A_t(n) = 1 \text{ and } \mathbf{e}[t](\tau_n) = 0 \\ \overline{\boldsymbol{\mu}}[p](\tau_n) + \mathbf{C}[p,t], & \text{if } A_t(n) = 1 \text{ and } \mathbf{e}[t](\tau_n) = 1 \end{cases} \tag{17.38}$$

From this basic evolution equation, it is possible to compute linear relations between second order moments of $\{\overline{\boldsymbol{\mu}}(\tau_n)\}_{n>0}$.

For any pair of places $p_1, p_2 \in P$, the expectation of the product of $\overline{\boldsymbol{\mu}}[p_1](\tau_{n+1})$ and $\overline{\boldsymbol{\mu}}[p_2](\tau_{n+1})$ conditioned to $\mathcal{F}_n$ (where $\mathcal{F}_n$ is the σ-field generated by the events up to τ_n) can be computed from (17.38):

$$\begin{aligned}
&\mathrm{E}\,[\overline{\boldsymbol{\mu}}[p_1](\tau_{n+1})\overline{\boldsymbol{\mu}}[p_2](\tau_{n+1}) \mid \mathcal{F}_n] = \\
&\quad \sum_{t\in T} \frac{\lambda_t}{\lambda}(1 - \mathbf{e}[t](\tau_n))\overline{\boldsymbol{\mu}}[p_1](\tau_n)\overline{\boldsymbol{\mu}}[p_2](\tau_n) + \\
&\quad \sum_{t\in T} \frac{\lambda_t}{\lambda}\mathbf{e}[t](\tau_n)\,(\overline{\boldsymbol{\mu}}[p_1](\tau_n) + \mathbf{C}[p_1,t])\,(\overline{\boldsymbol{\mu}}[p_2](\tau_n) + \mathbf{C}[p_2,t]) = \\
&\quad \overline{\boldsymbol{\mu}}[p_1](\tau_n)\overline{\boldsymbol{\mu}}[p_2](\tau_n) + \sum_{t\in T} \frac{\lambda_t}{\lambda}\mathbf{e}[t](\tau_n)\mathbf{C}[p_1,t]\mathbf{C}[p_2,t] + \\
&\quad \sum_{t\in T} \frac{\lambda_t}{\lambda}\mathbf{e}[t](\tau_n)\overline{\boldsymbol{\mu}}[p_1](\tau_n)\mathbf{C}[p_2,t] + \sum_{t\in T} \frac{\lambda_t}{\lambda}\mathbf{e}[t](\tau_n)\overline{\boldsymbol{\mu}}[p_2](\tau_n)\mathbf{C}[p_1,t]
\end{aligned}$$

Then, taking the expectation in the above equation and later taking the limit $n \to \infty$ we obtain:

$$\sum_{t \in T} \lambda_t \mathbf{e}[t]\mathbf{C}[p_1,t]\mathbf{C}[p_2,t] + \sum_{t \in T} \lambda_t \mathbf{y}[p_1,t]\mathbf{C}[p_2,t] + \sum_{t \in T} \lambda_t \mathbf{y}[p_2,t]\mathbf{C}[p_1,t] = 0$$

where

$$\mathbf{e}[t] = \lim_{\tau \to \infty} \mathrm{E}\left[\mathbf{e}[t](\tau)\right]$$

$$\mathbf{y}[p,t] = \lim_{\tau \to \infty} \mathrm{E}\left[\boldsymbol{\mu}[p](\tau)\mathbf{e}[t](\tau)\right]$$

Now, since $\lambda_t \mathbf{e}[t] = \boldsymbol{\chi}[t]$ (*utilization law*) and changing $\mathbf{y}[p,t]$ to

$$\mathbf{z}[p,t] = \lambda_t \mathbf{y}[p,t]$$

we get:

$$\sum_{t \in T} \boldsymbol{\chi}[t]\mathbf{C}[p_1,t]\mathbf{C}[p_2,t] + \sum_{t \in T} \mathbf{z}[p_1,t]\mathbf{C}[p_2,t] + \sum_{t \in T} \mathbf{z}[p_2,t]\mathbf{C}[p_1,t] = 0$$

In the particular case that $p_1 = p_2$, the above equation takes the form:

$$\sum_{t \in T} \boldsymbol{\chi}[t]\mathbf{C}[p,t]^2 + 2\sum_{t \in T} \mathbf{z}[p,t]\mathbf{C}[p,t] = 0$$

In summary, the following set of linear constraints can be added to the linear programming problem (17.1) of Section 17.1.1 for the computation of upper or lower bounds for linear functions of average marking of places and throughput of transitions:

(c_9) $\sum_{t \in T} \boldsymbol{\chi}[t]\mathbf{C}[p,t]^2 + 2\sum_{t \in T} \mathbf{z}[p,t]\mathbf{C}[p,t] = 0, \qquad \forall p \in P$

(c_{10}) $\sum_{t \in T} \boldsymbol{\chi}[t]\mathbf{C}[p_1,t]\mathbf{C}[p_2,t]$

$+ \sum_{t \in T} \mathbf{z}[p_1,t]\mathbf{C}[p_2,t] + \sum_{t \in T} \mathbf{z}[p_2,t]\mathbf{C}[p_1,t] = 0, \quad \forall p_1, p_2 \in P, p_1 \neq p_2$

(c_{11}) $\mathbf{z} \geq \mathbf{0}$

The reader should notice that the new variables $\mathbf{z}[p,t], p \in P, t \in T$ have been added also to the linear programming problem (17.1).

17.4.2 Embedded PF-QN's

Insensitive lower bounds for the average interfiring time of transitions were introduced in Section 17.2.1 looking for the maximum of the average interfiring time of transitions of isolated subsystems generated by elementary P-semiflows. A more realistic computation of the average interfiring time of transitions of these subsystems than that obtained from the analysis in complete isolation is

considered now using, once more, the concept of liveness bound of transitions. The number of servers at each transition t of a given system in steady state is limited by its corresponding liveness bound $\mathbf{lb}[t]$ (or by its structural enabling bound which can always be computed in an efficient manner), because this bound is the *maximum reentrance* (or maximum self-concurrency) that the net structure and the marking allow for the transition.

The technique we are going to briefly present (a more detailed discussion can be found in [14]) is based on a *decomposition* of the original model in subsystems. In particular, we look for *embedded product-form closed monoclass queueing networks*. Well-known efficient algorithms exist for the computation of exact values or bounds for the throughput of such models [28, 32, 18].

Therefore, let us concentrate in the search of such subsystems. How are they structurally characterized? From a topological point of view, they are *P-components*: strongly connected State Machines. Timing of transitions must be done with exponentially distributed services. Moreover, conditional routing is modelled with decisions among immediate transitions, corresponding to generalized free conflicts in the whole system. In other words, if t_1 and t_2 are in conflict in the considered P-component, they should be in generalized free conflict in the original net: $\mathbf{Pre}[\cdot, t_1] = \mathbf{Pre}[\cdot, t_2]$. The reason for this constraint is that since we are going to consider P-components as product-form closed monoclass queueing networks with limited number of servers at stations (transitions), the throughput of these systems is *sensitive to the conflict resolution policy*, even if the relative firing rates are preserved. Therefore, conflicts in the P-component must be solved with exactly the same marking independent discrete probability distributions as in the whole net system, in order to obtain an optimistic bound for the throughput of the original net system. We call *RP-components* the subnets verifying the previous constraints.

Definition 17.20 *Let $\mathcal{N}$ be a net and $\mathcal{N}_i$ a P-component of $\mathcal{N}$ (strongly connected State Machine subnet). $\mathcal{N}_i$ is a routing preserving P-component, RP-component, iff for any pair of transitions, t_j and t_k, in conflict in $\mathcal{N}_i$, they are in generalized free (equal) conflict in the whole net $\mathcal{N}$:* $\mathbf{Pre}[\cdot, t_j] = \mathbf{Pre}[\cdot, t_k]$.

An improvement of the insensitive lower bound for the average interfiring time of a transition t_j computed in Theorem 17.2 can be eventually obtained computing the exact average interfiring time of that transition in the RP-component generated by a minimal P-semiflow $\mathbf{y}$, with $\mathbf{lb}[t]$–server semantics for each involved transition t (in fact, it is not necessary that t_j belongs to the P-component; the bound for other transition can be computed and then weighted according to the visit ratios in order to compute a bound for t_j). The P-semiflow $\mathbf{y}$ can be selected among the optimal solutions of (17.9) or it can be just a feasible *near-optimal* solution.

As an example, let us consider the net system depicted in Figure 17.8. Assume that routing probabilities are equal to $1/3$ for t_1, t_2, and t_3, and that t_7, t_8, t_9, t_{10}, t_{11}, t_{12} have exponentially distributed service times with mean values $\bar{\mathbf{s}}[t_7] = \bar{\mathbf{s}}[t_8] = \bar{\mathbf{s}}[t_9] = 10$, $\bar{\mathbf{s}}[t_{10}] = \bar{\mathbf{s}}[t_{11}] = \bar{\mathbf{s}}[t_{12}] = 1$. The elementary P-semiflows of the net are:

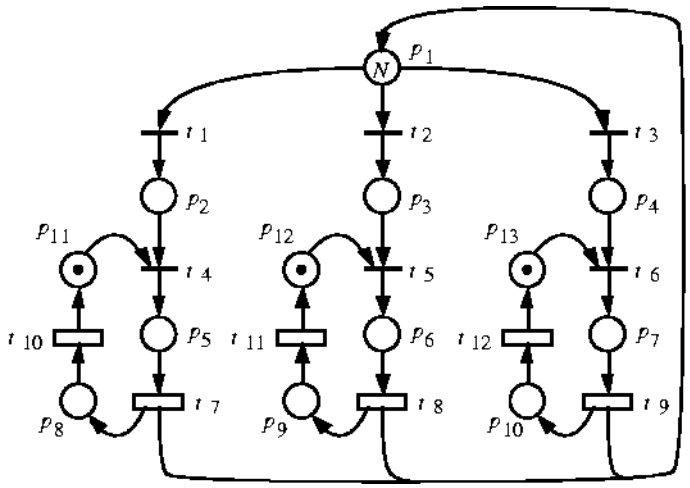


Figure 17.8: A live and bounded Free Choice system.

$$\begin{aligned}
\mathbf{y_1} &= (1,1,1,1,1,1,1,0,0,0,0,0,0) \\
\mathbf{y_2} &= (0,0,0,0,1,0,0,1,0,0,1,0,0) \\
\mathbf{y_3} &= (0,0,0,0,0,1,0,0,1,0,0,1,0) \\
\mathbf{y_4} &= (0,0,0,0,0,0,1,0,0,1,0,0,1)
\end{aligned} \tag{17.39}$$

Then, if the initial marking of p_{11}, p_{12}, and p_{13} is 1 token, and the initial marking of p_1 is N tokens, the lower bound for the average interfiring time derived from (17.9) is

$$\Gamma[t_1] \geq \max\{30/N, 11, 11, 11\} \tag{17.40}$$

For $N = 1$, the previous bound, obtained from $\mathbf{y_1}$, gives the value 30, while the exact average interfiring time is 31.06. For $N = 2$, the bound is 15 and it is derived also from $\mathbf{y_1}$ (average interfiring time of the P-component generated by $\mathbf{y_1}$, considered in isolation with infinite server semantics for transitions). This bound does not take into account the queueing time at places due to synchronizations (t_4, t_5, and t_6), and the exact average interfiring time of t_1 is $\Gamma[t_1] = 21.05$. For larger values of N, the bound obtained from (17.9) is equal to 11 (and is given by P-semiflows $\mathbf{y_2}$, $\mathbf{y_3}$ and $\mathbf{y_4}$). This bound can be improved if the P-component generated by $\mathbf{y_1}$ is considered with liveness bounds of transitions t_7, t_8, and t_9 reduced to 1 (which is the liveness bound of these transitions in the whole net).

The results obtained for different values of N are collected in Table 17.3. Exact values of average interfiring times for the P-component generated by $\mathbf{y_1}$ were computed using the *mean value analysis* algorithm [28]. This algorithm has $O(A^2B)$ worst case time complexity, where $A = \mathbf{y} \cdot \mathbf{m_0}$ is the number of

N	$\Gamma_{(17.9)}[t_1]$	$\Gamma_{(\mathbf{y_1})_{\mathbf{lb}}}[t_1]$	$\Gamma[t_1]$
1	30	30	31.06
2	15	20	21.05
3	11	16.67	17.71
4	11	15	16.03
5	11	14	15.03
10	11	12	13.02
15	11	11.34	12.35

Table 17.3: Bounds $\Gamma_{(17.9)}[t_1]$ obtained using (17.9), improvements for the bounds $\Gamma_{(\mathbf{y_1})_{\mathbf{lb}}}[t_1]$ presented in this section, and the exact average interfiring time $\Gamma[t_1]$ of t_1, for different initial markings N of p_1 in the net system of Figure 17.8.

tokens at the P-component and $B = \mathbf{y} \cdot \mathbf{Pre} \cdot \mathbf{1}$ is the number of involved transitions ($\mathbf{1}$ is a vector with all entries equal to 1). Exact computation on the original system takes several minutes in a *Sun SPARC Workstation* while bounds computation takes only a few seconds.

We also remark that other techniques for the computation of throughput upper bounds (instead of exact values) of closed product-form monoclass queueing networks could be used, such as, for instance, *balanced throughput upper bounds* [32] or *throughput upper bounds hierarchies* [18]. Hierarchies of bounds guarantee different levels of accuracy (including the exact solution), by investing the necessary computational effort. This provides also a hierarchy of bounds for the average interfiring time of transitions of Markovian Petri net systems.

Finally, the technique sketched in this section can be applied to the more general case of *Coxian* distributions (instead of exponential) for the service time of those transitions having either liveness bound equal to one (i.e., single-server stations) or liveness bound equal to the number of tokens in the RP-component (i.e., delay stations). The reason is that in these cases the embedded queueing network has also product-form solution, according to a classical theorem of queueing theory: the *BCMP theorem* [2].

17.4.3 Reduction and Transformation Techniques

The lower bounds for the throughput of transitions presented in previous sections are valid for any probability distribution function of service times but can be very pessimistic in some cases. In this section, an improvement of such results is briefly explained for the case of those net systems in which the following *performance monotonicity* property holds: *a local pessimistic transformation leads to a slower transformed net system* (i.e., a pessimistic local transformation guarantees a pessimistic global behaviour). Using the concept of *stochastic ordering* [29], a pessimistic transformation is, for example, to substitute the probability distribution function of a service (or token-subnet traversing) time

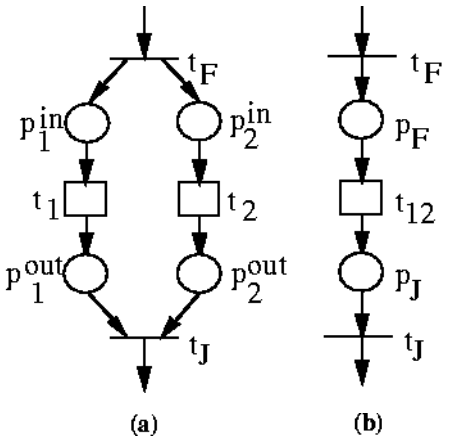


Figure 17.9: (a) Elementary fork-join and (b) its reduction.

by a *stochastically greater* probability distribution function. Live and bounded Free Choice is a class of systems for which the above performance monotonicity property holds. Details about the techniques presented here can be found in [11]. The basic ideas are:

1. To use local pessimistic transformation rules to obtain a net system "simpler" than the original (e.g., with smaller state space) and with equal or less performance.

2. To evaluate the performance for the derived net system, using insensitive bounds presented in previous sections, exact analysis, or any other applicable technique.

In order to obtain better bounds (after these two steps) than the values computed in previous sections, at least one of the transformation rules of item 1 must be less pessimistic than a total sequentialization of the involved transitions. We present first a rule whose application allows such *strict* improvement: the *fork-join rule*. Secondly, a rule that does not change at all the performance (*deletion of multistep preserving places*) is presented. Finally, a rule that does not follow the above ideas is also presented: the goal of this rule (*split of a transition*) is to make reapplicable the other transformation rules.

The most simple case of fork-join subnet that can be considered is depicted in Figure 17.9.a. In this case, if transitions t_1 and t_2 have exponential services X_1 and X_2 with means $\overline{s}[t_1]$ and $\overline{s}[t_2]$, respectively, they are reduced to a single transition (Figure 17.9.b) with exponential service time and mean:

$$\overline{s}[t_{12}] = \mathrm{E}[\max(X_1, X_2)] = \overline{s}[t_1] + \overline{s}[t_2] - \left(\frac{1}{\overline{s}[t_1]} + \frac{1}{\overline{s}[t_2]}\right)^{-1}$$

Therefore, even if the *mean traversing time* of the reduced subnet by a single token has been preserved, it has been substituted by a stochastically greater

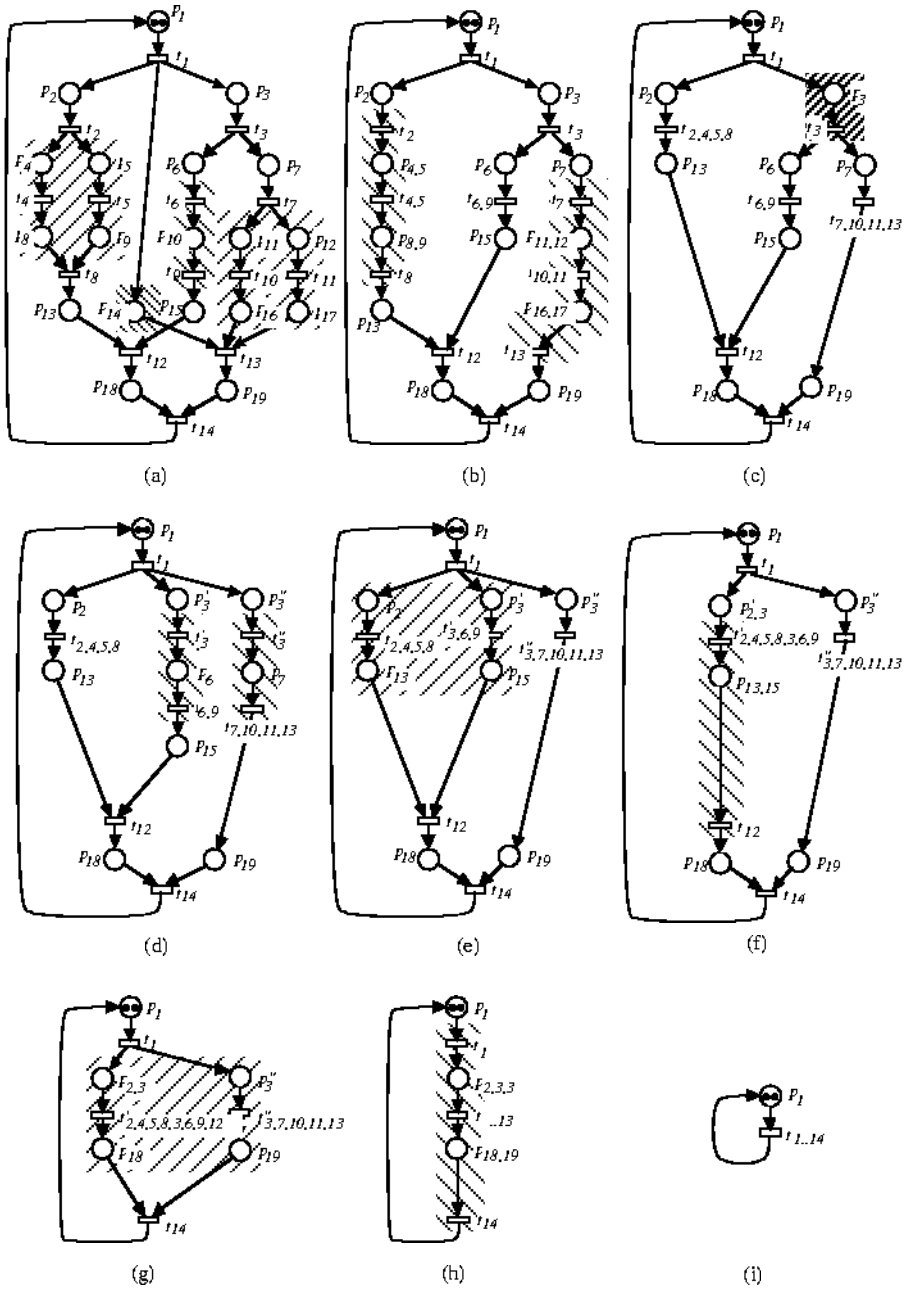


Figure 17.10: A complete reduction process. The relative error between insensitive bound and exact value diminishes from 140% to 35%.

variable. A trivial extension can be applied if the fork-join subnet includes more than two transitions in parallel.

Other transformation rules that have been presented in [11] are:

Deletion of a multistep preserving place: allows to remove some places without changing the *exact* performance indices of the stochastic net system. In fact the places that can be deleted are those whose elimination preserves the multisets of transitions simultaneously firable in all reachable markings (e.g., place p_{14} in Figure 17.10.a). The size of the state space of the model is preserved and also the exact throughput of transitions of the system.

Reduction of transitions in sequence: reduces a series of exponential services to a single exponential service with the same mean. Intuitively, this transformation makes indivisible the service time of two or more transitions representing elementary actions which always occur one after the other and lead to no side condition (e.g., transitions t_6 and t_9 in Figure 17.10.a). Therefore, the state space of the model is reduced. The throughput of transitions is, in general, reduced.

Split of a transition: this is not a state space reduction rule since it increases the state space of the transformed net system. The advantage of the rule is that it allows to proceed further in the reduction process using again the previous rules (e.g., transition t_3 in Figure 17.10.c).

An example of application of all above transformation rules is depicted in Figure 17.10 for a strongly connected marked graph with exponential timing. Let us assume that average service times of transitions are: $\overline{\mathbf{s}}[t_i] = 1$, $i = 1, 2, 3, 7, 8, 12, 13, 14$, and $\overline{\mathbf{s}}[t_i] = 10$ otherwise.

In order to compute firstly the insensitive lower bounds on throughput introduced in Section 17.3.2, it is necessary to derive the liveness bounds of transitions. In this case it is easy to see that $\mathbf{lb}[t_j] = 2$ for every transition t_j.

The vector of visit ratios of an MG is the unique minimal T-semiflow of the net: $\mathbf{v} = \mathbf{1}$. Therefore, the insensitive upper bound (valid for any probability distribution function of service times) of the average cycle time of the MG is $\Gamma \leq 34$. This value can be reached for some distributions of service times (see Theorem 17.18). Nevertheless, if services are exponential the exact average cycle time of the MG is $\Gamma = 14.15$.

The quantitative results of the transformation process illustrated in Figure 17.10 are shown in Table 17.4. We remark that the bound has been improved in polynomial time from 34 to 19.2.

17.5 Bibliographic Remarks

The general approach for the computation of insensitive performance bounds presented in Section 17.1 was introduced in [15].

The reinterpretation using Little's law and P-semiflows presented in Section 17.2.1 is in fact historically previous to the general approach, since it was

System	bound	error
Fig. 17.10.a	34	140 %
Fig. 17.10.b	29	105 %
Fig. 17.10.c	29	105 %
Fig. 17.10.d	29.5	108 %
Fig. 17.10.e	29.5	108 %
Fig. 17.10.f	24.8	75 %
Fig. 17.10.g	24.8	75 %
Fig. 17.10.h	19.2	35 %
Fig. 17.10.i	19.2	35 %
exact value: 14.15		

Table 17.4: Successive improvements of the upper bound for the average cycle time of the MG in Figure 17.10 and relative errors with respect to the exact value $\Gamma = 14.15$.

firstly introduced in [5] for marked graphs and in [7] for Petri nets with *unique consistent firing count vector*. Improved versions of those papers are [6] and [8], respectively. The technique was extended to Free Choice systems in [9] and to *FRT-net systems* (cfr. Chapter 8) in [4] and [13].

The relation, presented in Section 17.2.2, between the general technique of Section 17.1 and the P-semiflows based technique of Section 17.2.1 is original from this chapter.

The results on the reachability of the throughput upper bound for marked graphs of Section 17.2.3 have been taken from [5, 6].

Concerning the improvements of the bounds, that based on implicit places (Section 17.3.1) was published in [10]; the use of liveness bounds of transitions presented in Section 17.3.2 was introduced for marked graphs in [5, 6] and later extended to Free Choice systems in [9]. The uniformization technique used to compute linear relations between second order moments in Section 17.4.1 was proposed in this framework in [24]. The improvement of the bounds based on the consideration of embedded product-form queueing networks presented in Section 17.4.2 was published in [12] and later improved in [14]. Finally, the reduction and transformation techniques briefly presented in Section 17.4.3 have been taken from [11].

Concerning other works that are not considered at all in this chapter, a large number of bounding techniques have been proposed for the performance measures of classical (*synchronization-free*) queueing networks. The first family is that of *asymptotic bound analysis* [22, 17]. Asymptotic bounds are obtained by considering two extreme situations: (1) no queueing takes place at any node, and (2) at least one station is saturated. These bounds do not require the product form property to hold and their computation is very fast, but they are not accurate in general. The rest of bounds that have been introduced are tighter but do require the product form assumption. This is the case of

balanced job bounds [32, 23], which are based on the *mean value theorem* [28]. Finally, several schemes for the construction of *hierarchies of bounds* have been developed that guarantee any level of accuracy (including the exact solution), by investing the necessary computational effort: *performance bound hierarchies* [18, 19], *successively improving bounds* [30], *generalized quick bounds* [31]. All these techniques are derived from mean value theorem, thus they are valid only for product form networks.

With respect to timed Petri nets, M. Molloy [25] noted that the average token flows in an ordinary Markovian network at steady-state are conserved. Therefore, a series of *flow balance equations* can be written. Token flows are conserved in places so the sum of all flows into a place equals the sum of all flows out of the place. On the other hand, all token flows on the input and output arcs of a transition are equal. These equations determine the average token flows in the cycles of the net to within a constant. This constant cannot be determined without Markovian analysis at the reachability graph level. However, limit flows when the number of tokens tends to infinity can be computed. In order to do that, bottleneck transitions must be first located. Then, the actual flow through a bottleneck transition is (under saturation conditions) equal to its potential firing rate.

S. Bruell and S. Ghanta [3] developed algorithms for computing upper and lower bounds for the throughput of a restricted subclass of generalized stochastic Petri nets (with immediate and exponentially timed transitions). The considered nets include *control tokens* to model a physical restriction, such as semaphores, which is not a design parameter. The rest of tokens of such nets, grouped in *classes*, correspond to the notion of a job or customer in a monoclass queueing network, and its number is treated as a parameter of the net. The upper and lower bounds on throughput are computed hierarchically estimating maximum and minimum time of the path followed by each class of jobs.

In the paper of S. Islam and H. Ammar [20], methods to compute upper and lower bounds for the steady-state token probabilities of a subclass of generalized stochastic Petri nets were presented. The considered nets are obliged to admit a *time scale decomposition*. This means that the transitions of the net are supposed to be divided into two classes: slow and fast transitions, with several orders of magnitude of difference in the duration of activities. Moreover, the subnets obtained after removing all slow transitions with their input and output arcs must be conservative and admit a reversible initial marking. The computation is based on *near-completely decomposability* of Markov chains.

Bibliography

[1] F. Baccelli and P. Bremaud. *Elements of Queueing Theory.* Springer-Verlag, 1994.

[2] F. Baskett, K. M. Chandy, R. R. Muntz, and F. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, April 1975.

[3] S. C. Bruell and S. Ghanta. Throughput bounds for generalized stochastic Petri net models. In *Proceedings of the International Workshop on Timed Petri Nets*, pages 250–261, Torino, Italy, July 1985. IEEE Computer Society Press.

[4] J. Campos. *Performance Bounds for Synchronized Queueing Networks.* PhD thesis, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, Spain, October 1990. Research Report GISI-RR-90-20.

[5] J. Campos, G. Chiola, J. M. Colom, and M. Silva. Tight polynomial bounds for steady-state performance of marked graphs. In *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models*, pages 200–209, Kyoto, Japan, December 1989. IEEE Computer Society Press.

[6] J. Campos, G. Chiola, J. M. Colom, and M. Silva. Properties and performance bounds for timed marked graphs. *IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications*, 39(5):386–401, May 1992.

[7] J. Campos, G. Chiola, and M. Silva. Properties and steady-state performance bounds for Petri nets with unique repetitive firing count vector. In *Proceedings of the 3rd International Workshop on Petri Nets and Performance Models*, pages 210–220, Kyoto, Japan, December 1989. IEEE Computer Society Press.

[8] J. Campos, G. Chiola, and M. Silva. Ergodicity and throughput bounds of Petri nets with unique consistent firing count vector. *IEEE Transactions on Software Engineering*, 17(2):117–125, February 1991.

[9] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):1368–1382, December 1991.

[10] J. Campos, J. M. Colom, and M. Silva. Improving throughput upper bounds for net based models of manufacturing systems. In J. C. Gentina and S. G. Tzafestas, editors, *Robotics and Flexible Manufacturing Systems*, pages 281–294. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, The Netherlands, 1992.

[11] J. Campos, B. Sánchez, and M. Silva. Throughput lower bounds for Markovian Petri nets: Transformation techniques. In *Proceedings of the* 4^{rd} *International Workshop on Petri Nets and Performance Models*, pages 322–331, Melbourne, Australia, December 1991. IEEE-Computer Society Press.

[12] J. Campos and M. Silva. Throughput upper bounds for Markovian Petri nets: Embedded subnets and queueing networks. In *Proceedings of the* 4^{rd} *International Workshop on Petri Nets and Performance Models*, pages 312–321, Melbourne, Australia, December 1991. IEEE-Computer Society Press.

[13] J. Campos and M. Silva. Structural techniques and performance bounds of stochastic Petri net models. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 352–391. Springer-Verlag, Berlin, 1992.

[14] J. Campos and M. Silva. Embedded product-form queueing networks and the improvement of performance bounds for Petri net systems. *Performance Evaluation*, 18(1):3–19, July 1993.

[15] G. Chiola, C. Anglano, J. Campos, J. M. Colom, and M. Silva. Operational analysis of timed Petri nets and application to the computation of performance bounds. In *Proceedings of the* 5^{th} *International Workshop on Petri Nets and Performance Models*, pages 128–137, Toulouse, France, October 1993. IEEE-Computer Society Press.

[16] J. M. Colom and M. Silva. Improving the linearly based characterization of P/T nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 113–145. Springer-Verlag, Berlin, 1991.

[17] P. J. Denning and J. P. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, September 1978.

[18] D. L. Eager and K. C. Sevcik. Performance bound hierarchies for queueing networks. *ACM Transactions on Computer Systems*, 1(2):99–115, May 1983.

[19] D. L. Eager and K. C. Sevcik. Bound hierarchies for multiple-class queueing networks. *Journal of the ACM*, 33(1):179–206, January 1986.

[20] S. M. R. Islam and H. H. Ammar. On bounds for token probabilities in a class of generalized stochastic Petri nets. In *Proceedings of the* 3^{rd} *International Workshop on Petri Nets and Performance Models*, pages 221–227, Kyoto, Japan, December 1989. IEEE-Computer Society Press.

[21] J. Keilson. *Markov Chain Models. Rarity and Exponentiallity.* Springer-Verlag, 1979.

[22] L. Kleinrock. *Queueing Systems Volume II: Computer Applications.* John Wiley & Sons, New York, NY, 1976.

[23] J. Kriz. Throughput bounds for closed queueing networks. *Performance Evaluation*, 4:1–10, 1984.

[24] Z. Liu. Performance bounds for stochastic timed Petri nets. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995*, volume 935 of *Lecture Notes in Computer Science*, pages 316–334. Springer-Verlag, Berlin, 1995.

[25] M. K. Molloy. Fast bounds for stochastic Petri nets. In *Proceedings of the International Workshop on Timed Petri Nets*, pages 244–249, Torino, Italy, July 1985. IEEE-Computer Society Press.

[26] G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors. *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science.* North-Holland, Amsterdam, 1989.

[27] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Petri Nets.* PhD thesis, MIT, Cambridge, MA, USA, February 1974.

[28] M. Reiser and S. S. Lavenberg. Mean-value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):313–322, April 1980.

[29] S. M. Ross. *Stochastic Processes.* John Wiley & Sons, New York, NY, 1983.

[30] M. M. Srinivasan. Successively improving bounds on performance measures for single class product form queueing networks. *IEEE Transactions on Computers*, 36:1107–1112, September 1987.

[31] R. Suri. Generalized quick bounds for performance of queueing networks. *Computer Performance*, 5(2):116–120, June 1984.

[32] J. Zahorjan, K. C. Sevcik, D. L. Eager, and B. Galler. Balanced job bound analysis of queueing networks. *Communications of the ACM*, 25(2):134–141, February 1982.

Chapter 18

Product Form & Petri Nets

18.1 Introduction

The major problem in computing performance measures using Stochastic Petri Nets (SPNs) is the need to work with the equilibrium equations based on the reachability graph. The reachability graph size increases exponentially with both the number of tokens in the initial marking and the number of places and consequently, except for special classes of models, the dimension of this graph and the time complexity of the solution procedure preclude the exact analytical solution of many interesting models.

Queueing Networks (QNs) suffer from the same limitations, but the problem has been alleviated by the discovery of a class of QNs whose solution can be computed is an easy way (see for instance [2, 21, 23]). For this classes of QNs the solution can be factorised into terms that refer to each single queue of the network: it is the well known *Product Form Solution* (PFS).

Queueing networks with product form equilibrium distributions are well established and have applications in a wide range of fields. With few exceptions these networks provide the only efficient means of analysis in large systems. The advantage that the product form distribution provides is the availability of a large number of efficient computational algorithms for computing various performance measures either directly or through the normalisation constant (see for instance [7, 34] or [6] for a collection of these algorithms). These algorithms help to overcome the state space explosion problem which is inherent in large networks.

Recently there were several proposals to extent the PFS concepts to SPNs. The aim of this chapter and of the next one is to survey the product form results and the associated algorithms which currently exist in the SPN field. In particular we will focus the attention on the proposals that use the peculiarities of the Petri nets for recognising, computing, and using the product form equilibrium distribution for SPNs.

18.1.1 Outline of Chapter 18 and Chapter 19

The balance of this chapter is the following: Section 18.2 we briefly review the most part of the Product Form Solution criteria proposed in the literature. Section 18.3 introduces some basic concepts that will be used through the chapter. Section 18.4 presents the Product Form Solution for SPNs proposed by Henderson *et al.*. In Section 18.5 there is a comparison among the Product Form Solution for SPNs criteria proposed in the literature. In this section we also compare the criterion proposed by Henderson *et al.* with the Product Form for queueing networks (Gordon and Newell, BCMP, etc.).
For the class of SPNs characterised by the criterion presented in Section 18.4 several interesting results have been developed. In Section 18.6 some structural properties of this class of nets are presented.
Chapter 19 is dedicated to the presentation of the computational algorithms for this class of SPNs. In particular Section 19.1 and Section 19.2 of the next chapter present respectively a normalisation constant and a mean value analysis algorithms for the SPNs that have this type of product form solution. Concluding and bibliographical remarks are summarised in Sections 19.3 and 19.4.

18.2 Background and motivations

In the paper [33] Molloy pointed out that the real success of Petri net modeling will not occur until the fundamental problem of the reachability set explosion is solved. This is the same that plagued queueing networks for many years. It was overcome in some cases by the discovery of classes of models with product form equilibrium distributions.

One of the first work on product form solution was written by Jackson [21] who showed that the invariant measure for particular classes of queueing networks was a product of the marginal measures for each queue. Some of more noted results on product form queueing theory have appeared since the work of Jackson are by Gordon and Newell [16], Baskett *et al.* [2], and by Kelly [23]. One practical appeal of product form queueing networks is the availability of several efficient computational algorithms for computing the performance measures.

Recently, product form queueing networks have evolved beyond the single movement restriction to include batch movement of customers and correlated routing [19]. These type of networks are closely related with SPNs and hence starting from these product form results Henderson *et al.* have been shown that a certain class of SPNs have product form equilibrium distribution [18, 20]. During the last year several other proposals of product form results for SPNs have also appeared in the literature, some of them will be considered in the next section.

18.2.1 Lazar and Robertazzi Product Form criterion

The first product form result for SPNs can be attributed to Lazar and Robertazzi [27, 28] who consider SPNs at the marking level. They define a class of

safe and live SPNs whose equilibrium distributions satisfy partial balance equations and have product form equilibrium distributions. They assume that the transition firing rates of the SPNs are marking independent. Following earlier work in queueing theory [25] their approach was to look for the presence of *building blocks* in the reachability graph. The building blocks are closed finite sub-graphs of the reachability graph. From the topological point of view, the original reachability graph can be reconstructed by pasting the building blocks together. Mathematically, the sum of the global balance equations for the states within each building block, when considered in isolation, re-constitute the global balance equations for the entire reachability graph. The global balance equations for the states of each building block when considered in isolation are the possible set of partial balance equations. If each partial balance equation produces the same equilibrium probabilities, the equations are said to be consistent and moreover the solution of the partial balance equations is also a solution for the global balance equations. Partial balance has many meaning in the literature with many authors using different terminology for the same type of balance. In the following, by means of an example, we clarify the type of partial balance considered by Lazar and Robertazzi.

In any decomposition of the global balance equations into partial balance equations, each state may appear many times but each transition rate may appear only once. If this were not the case, the partial balance equations would not reform the global balance equations when summed together. It terms of building blocks, this means that each direct arc may only be used in one building block while each marking may appear in many building blocks.

One of the limitations of the Lazar and Robertazzi criterion is that the decomposition of the reachability graph into building blocks may be not unique and the partial balance equations for a given set of building blocks may not be consistent. Another strong limitation is the need to generate the reachability graph. In the paper [26] the Lazar and Robertazzi give necessary and sufficient conditions for the consistency of the partial balance equations derived from the building blocks. These conditions require certain flow relations, based on the transition rates, to hold for any closed path in the reachability graph. The consistency conditions are obtained through a structure that the authors called *consistency tree*. The relation between building blocks and product form has been called by the authors of this proposal *duality principle*.

18.2.2 Li and Georganas Product Form criterion

Li and Georganas [29] refer to SPNs which satisfy the product form criterion of Lazar and Robertazzi as Locally Balanced Stochastic Petri Nets (LBSPNs). The concept of local balance considered is equivalent to the concept of partial balance presented above.

The product form of Lazar and Robertazzi is base on a decomposition of the reachability graph into building blocks. Li and Georganas extend their definition of LBSPNs to a class of nets which are locally balanced based on a decomposition of the SPNs [29]. Based on the initial marking, they decompose a given SPN

according to its firable T-semiflows so that sub-nets have unique transitions but may have common places. Since each firable T-semiflow corresponds to a closed path in the reachability graph, the decomposition of Li and Georganas is a method for selecting building blocks. If the local balance equations based on the building blocks are consistent the nets are called LBSPNs. The approach of Li and Georganas extend the method of Lazar and Robertazzi to more general SPNs.

18.2.3 Frosch Product Form criterion

Frosch, in [15], considers a class of nets called closed synchronised systems of stochastic sequential processes (CS). He introduced the concept of *sequential transitions*, i.e., two transitions are sequential in the output vector of the first in the input vector of the second. Frosch's nets consist of cyclic state machines which can be joined together through common buffer places such that the flow of tokens through the buffer places is conserved for every pair of sequential transitions in any state machine. Each cyclic state machine is equivalent to a closed migration process such that considered by Kelly [24]. When the state machine are joined together by buffer places the result is a network of migration processes whose set transitions are restricted in a special way. The use of buffer tokens by one cyclic state machine may restrict the possible transition firings within other state machines. The product form which Frosch obtains is both a product over each state machines and over each place and is based on the structure of the net. Unlike Lazar and Robertazzi Frosch allows state dependent firing rates and does not restrict to safe nets. In order to guarantee product form, he assumes that every T-semiflow for each state machine is firable whenever one of the transitions in the T-semiflow is enabled. Although he does not prove the result, he claims that this assumption corresponds to the product form conditions of Lazar and Robertazzi. Frosch's assumption ensures that every T-semiflow is represented as a closed path in the reachability graph. Thus like Li and Georganas' result, this assumption is related to building blocks. It ensures that the building blocks for each T-semiflow are repeated as often as possible. Frosch's assumption however does not guarantee the existence of building blocks with consistent partial balance equations.

18.2.4 Henderson, Lucic, and Taylor Product Form criterion

Henderson, Lucic, and Taylor obtained a product form criterion based only on the structure of the net, without the need to generate the reachability graph. Their class of nets originates from results obtained within the framework of the batch movement queueing networks. The product form equilibrium distribution which they obtain is not the classical product form over the place but a product of two functions

$$\pi(\mathbf{m}) = \frac{1}{G}\Phi(\mathbf{m})\pi_f(\mathbf{m}), \tag{18.1}$$

where the function $\pi_f(\mathbf{m})$ is related to the routing while $\Phi(\mathbf{m})$ is related to transition firing rates.

This product form criterion is covered in more detail in the next sections where we examine the necessary and sufficient conditions for finding an invariant measure for the routing chain, necessary and sufficient conditions to have a product form over the places of the net, structural properties of the class of nets satisfying this product form criterion, and computational algorithms that allow the efficient computation of performance measures for the SPNs that satisfy this type of product form. In the rest of this chapter we call this criterion *structural product form solution.*

18.2.5 Boucherie Product Form Criterion

In the paper [3] Boucherie discusses product form results for SPNs using a Markov chain approach. He considers N continuous Markov chain which are related only through their transition rates. The state space for each Markov chain is partitioned into mutually exclusive sets each corresponding to a resource and the state of each Markov chain determines which resources are in use. Markov chain j uses resource i if the current state of the chain is in the partition corresponding to that resource. Dependence between the Markov chains only occurs though competition for resources as follows. For each resource there is a subset of chains which compete for it. Only one of the chain competing for a given resource can use that resource. A resource can be used by more than one chain only if the respective chains do not compete with each other for that resource.

The process whose state space is the union of the state spaces for each Markov chain is known as the product process. The product process has a state description given by

$$\overline{\mathbf{m}} = (\mathbf{m}_1, \ldots, \mathbf{m}_N)$$

where $\mathbf{m}_i$ is the state space of the i-th Markov chain. The product process is restricted in such a way that it can only change state through a change in state of one of its constituent Markov chains. Thus the transition rates are defined to be

$$q(\overline{\mathbf{m}}, \overline{\mathbf{m}'}) = \sum_{i=1}^{N} q_i(\overline{\mathbf{m}}, \overline{\mathbf{m}'}) \prod_{r=1,\, r\neq i}^{N} I[\mathbf{m}_r = \mathbf{m}'_r] I[\nu_r], \tag{18.2}$$

where $I[\cdot]$ is the indicator function and $q_i(\overline{\mathbf{m}}, \overline{\mathbf{m}'})$ is the transition rate within the i th chain. The first indicator function ensures that only chain i can change state while the condition ν_r ensures that no resource for which chain i competes is in use. The effect of this is that whenever just one of the resources for which chain j competes is in use this chain is stopped and no changes are allowed.

The equilibrium distribution of the product process is still shown however to factorise into a product of marginal distributions for each Markov chain. The reason for the product form equilibrium distribution is the restricted interaction between the individual Markov chains and the fact that each chain is either operational or stopped.

18.2.6 Florin and Natkin Product Form Criterion

In the paper [14] Florin and Natkin consider a class of synchronised queueing networks which can be modelled by ordinary and bounded SPNs with marking independent transition firing rates and a reachability graph which is strongly connected. For these nets they obtain a matrix geometric product form result where the equilibrium distribution is a sum of product forms. As far as we are aware there exist no efficient algorithm for computing the normalisation constant of their type of product form.

18.3 Peculiarities of the class of Stochastic Petri Nets

A Stochastic Petri net (SPN) is a triple $\langle \mathcal{N}, \mathbf{m_0}, \mathcal{T} \rangle$ where $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post} \rangle$ is a P/T net, $\mathbf{m_0}$ is the initial marking and $\mathcal{T}$ is the set of instantaneous firing rates of the transitions, i.e., the parameters of the exponential distributions of the firing delays associated with the transitions. The firing rates can be marking dependent.

In the following we denote the *input vector* of a transition t_j $\mathbf{Pre}[P, \cdot]$ as $\mathbf{i}(t_j)$ and the *output vector* $\mathbf{Post}[P, \cdot]$ as $\mathbf{o}(t_j)$.

18.3.1 Markov chain model

We assume that the SPN can be represented by a stable and regular continuous Markov chain $\mathbf{X} = \{X(\tau), \tau > 0\}$ with the state space $\mathrm{RS}(\mathbf{m_0})$. The rate at which a transition t_j fires in a marking $\mathbf{m}$ can be defined as

$$q(\mathbf{i}(t_j), \mathbf{o}(t_j); \mathbf{m} - \mathbf{i}(t_j)) = \mu(\mathbf{i}(t_j)) \frac{\psi(\mathbf{m} - \mathbf{i}(t_j))}{\Phi(\mathbf{m})} p(\mathbf{i}(t_j), \mathbf{o}(t_j)), \tag{18.3}$$

for all $t_j \in T$, $\mathbf{m} \in \mathrm{RS}(\mathbf{m_0})$ such that $\mathbf{m} \geq \mathbf{i}(t_j)$, where

$$\mu(\mathbf{i}(t_j)) = \sum_{\substack{t_h \in T \\ \mathbf{i}(t_h) = \mathbf{i}(t_j)}} \mu_h,$$

$\psi(\cdot)$ is an arbitrary non-negative function and $\Phi(\cdot)$ a positive function. In (18.3) the functions $\psi(\cdot)$ and $\Phi(\cdot)$ can be thought of as "potential functions", the state dependent firing rate of transition t_j being the product of its intrinsic firing rate $\mu(\mathbf{i}(t_j))$ and the ration of the functions $\psi(\cdot)$ and $\Phi(\cdot)$ evaluated at the states that exist after and before the firing, respectively. These function plus the rates $\mathcal{T}$ can be regarded as part of the specification of an SPN in that they define the transition firing rates. Marking independent firing rates can be modelled by choosing $\psi(\cdot) = \Phi(\cdot) = 1$.
A service function of the form (18.3) is common in the literature on product form queueing networks (see, for example [19] or [41]). In the SPN context, in

was first used in [18] where several examples are given which illustrate its range of application.
For single movement queueing networks, a form with $\psi = \Phi$ first appeared in [22], where it generalised the state dependent service rates introduced in [21].

The transition rate from marking $\mathbf{m}$ to marking $\mathbf{m}' = \mathbf{m} - \mathbf{i}(t_j) + \mathbf{o}(t_j)$ in the Markov chain $\mathbf{X}$ is

$$q(\mathbf{m}, \mathbf{m}') = \sum_{\substack{\mathbf{n} + \mathbf{i}(t_j) = \mathbf{m} \\ \mathbf{n} + \mathbf{o}(t_j) = \mathbf{m}'}} q(\mathbf{i}(t_j), \mathbf{o}(t_j); \mathbf{n}). \tag{18.4}$$

A collection of positive number $w = (w(\mathbf{m}), \mathbf{m} \in \mathrm{RS}(\mathbf{m_0}))$, is called an *invariant measure* for the Markov chain $\mathbf{X}$, if it satisfies the following equations

$$\sum_{\mathbf{m}' \in \mathrm{RS}(\mathbf{m_0})} \{w(\mathbf{m}) q(\mathbf{m}, \mathbf{m}') - w(\mathbf{m}') q(\mathbf{m}', \mathbf{m})\} = 0, \tag{18.5}$$

for all $\mathbf{m} \in \mathrm{RS}(\mathbf{m_0})$. The previous equations are called the *global balance equations* for $\mathbf{X}$. When w is a proper distribution over $\mathrm{RS}(\mathbf{m_0})$ it is called *equilibrium distribution*, in this case it will be denoted by $\pi = (\pi(\mathbf{m}), \mathbf{m} \in \mathrm{RS}(\mathbf{m_0}))$.

In the literature several more restrictive forms of balance exist. In particular, it is well known that a form of *local balance* is the common feature for all performance models with *product form equilibrium distribution*. In this survey we will use the *group-local-balance* (Boucherie and van Dijk [5]) to prove that the proposed product form distribution is an equilibrium distribution. A measure w satisfies the group-local-balance property for the Markov chain $\boldsymbol{X}$ if for all $\mathbf{g}, \mathbf{g}'$, $\mathbf{n}$, such that $\mathbf{n} + \mathbf{g} \in \mathrm{RS}(\mathbf{m_0})$

$$\sum_{\mathbf{g} \neq \mathbf{g}'} \{w(\mathbf{n} + \mathbf{g}) q(\mathbf{g}, \mathbf{g}'; \mathbf{n}) - w(\mathbf{n} + \mathbf{g}') q(\mathbf{g}', \mathbf{g}; \mathbf{n})\} = 0, \tag{18.6}$$

where $\mathbf{g}, \mathbf{g}'$ represent input and output vectors of transitions. If we sum the group-local-balance equations over all $\mathbf{g}, \mathbf{n}$ such that $\mathbf{g} + \mathbf{n} = \mathbf{m}$ and $\mathbf{m} \in \mathrm{RS}(\mathbf{m_0})$ we obtain the global balance equations.

18.4 A "Structural" Product Form Solution Criterion for SPNs

In this section we review the basic concepts of the class of stochastic Petri nets that have a *Product-Form Solution*. The criterion considered in this chapter for identifying a PF-SPN is that proposed by Henderson *et al.* [18, 20]; more comprehensive presentations of the results related with this topic can be found in the references [4, 10, 11]. The key for identifying the SPNs with a PFS is to consider the input and output vectors of the transitions of the SPN to be states of a Markov chain. This Markov chain has been called the *routing process* [18] and can be described using the following arguments.

Let $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_h}$ denote the minimal T-semiflows obtained from the incidence matrix of the net.

Definition 18.1 [Closed set of transitions] (From [4]) *For $T' \subseteq T$ define $\mathcal{K}(T')$, the set of input and output vectors for transitions in T', as*

$$\mathcal{K}(T') = \bigcup_{t_j \in T'} \{\mathbf{i}(t_j) \cup \mathbf{o}(t_j)\}.$$

The subset of transitions T' is said to be closed *if for any $\mathbf{l} \in \mathcal{K}(T')$ there exist two transitions $t_m, t_n \in T'$ such that $\mathbf{l} = \mathbf{i}(t_m)$, and $\mathbf{l} = \mathbf{o}(t_n)$. Transitions t_m and t_n are said to be* adjacent.

An alternative formulation of this definition is the following. A subset of transitions T' ($T' \subseteq T$) is said to be closed if

$$\bigcup_{t \in T'} \mathbf{i}(t) = \bigcup_{t \in T'} \mathbf{o}(t). \tag{18.7}$$

The following definition is the key for identifying (at the structural level) those SPNs that have a PFS.

Definition 18.2 *$\mathcal{N}$ is a* Π-net *if $\forall\, t \in T$ there exists a minimal T-semiflow $\mathbf{x}$ such that $t \in ||\mathbf{x}||$, and $||\mathbf{x}||$ is a closed set.*

In other words, $\mathcal{N}$ is a Π-net if all transition are covered by closed support minimal T-semiflows.

Among the minimal closed support T-semiflows we can identify a relation that can be used to derive the PFS. In the following we denote by $\mathcal{X}_{\rm cd}$ the set of closed support minimal T-semiflows of a net.

Definition 18.3 [Freely related T-semiflows] *Let $\mathcal{N} = (P, T, \mathbf{Pre}, \mathbf{Post})$ be a Π-net and $\mathbf{x}'$, $\mathbf{x}''$ be two different minimal closed support T-semiflows of $\mathcal{N}$. $\mathbf{x}'$ and $\mathbf{x}''$ are said to be* freely related, *denoted as $(\mathbf{x}', \mathbf{x}'') \in FR$, if there exist $t' \in ||\mathbf{x}'||$ and $t'' \in ||\mathbf{x}''||$ such that $\mathbf{i}(t') = \mathbf{i}(t'')$. The relation FR^* is the transitive closure of FR.*

The transitive closure of the relation FR is defined as follows: if $\mathbf{x_i}$, $\mathbf{x_j}$, $\mathbf{x_k}$, are three different minimal closed support T-semiflows of $\mathcal{N}$ such that $(\mathbf{x_i}, \mathbf{x_j}) \in FR$, and $(\mathbf{x_j}, \mathbf{x_k}) \in FR$. Then we say that $(\mathbf{x_i}, \mathbf{x_k}) \in FR^*$. This reflects the property that we can relate $\mathbf{x_i}$ to $\mathbf{x_k}$ via $\mathbf{x_j}$.

It is easy to see that the relation FR^* yields a partitioning of the set of the minimal closed support T-semiflows of $\mathcal{N}$ into equivalence classes. Let $[\mathbf{x_i}]$ be the equivalence class of $\mathbf{x_i}$, i.e., $[\mathbf{x_i}] = \{\mathbf{x}' : (\mathbf{x_i}, \mathbf{x}') \in FR^*\}$ with $[\mathbf{x_{i_h}}] \cap [\mathbf{x_{i_l}}] = \emptyset$, for any $(\mathbf{x_{i_h}}, \mathbf{x_{i_l}}) \notin FR^*$. Since any transition t can belong to one and only one FR^*-class, the partition of T-semiflows leads to a partition of the set of transitions. Let t', t'' be two transitions of $\mathcal{N}$, we can say that $(t', t'') \in FR^*$ iff $\mathbf{x}'$ and $\mathbf{x}''$ are two minimal closed support T-semiflows of $\mathcal{N}$ such that $t' \in ||\mathbf{x}'||$, $t'' \in ||\mathbf{x}''||$, and $(\mathbf{x}', \mathbf{x}'') \in FR^*$.

It is worthwhile pointing out that if there are two transitions having the same input bag, they both belong to the same FR^*-class. From this immediately follows that the partitioning of the set of transitions produces a partitioning of the set of input vectors as well.

In the following we denote as $\mathcal{C}_{FR^*} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_{nc}\}$ the partitioning of the set of transitions into FR^*-classes.

Having obtained these preliminary results, we can define the *routing process* as a Markov chain $\mathbf{y} = (\mathbf{y}(z),\ z \geq 0)$ whose state space is $\mathcal{S} = \{\mathbf{i}(t_j),\ t_j \in T\}$ and whose transition rates are $q(\mathbf{i}(t_j), \mathbf{i}(t_m)) = \mu_j P(\mathbf{i}(t_j), \mathbf{i}(t_m))$. The term $P(\mathbf{i}(t_j), \mathbf{i}(t_m))$ represents the probability that if transition t_j fires in a given marking $\mathbf{m}$, the next marking is $\mathbf{m}' = \mathbf{m} - \mathbf{i}(t_j) + \mathbf{o}(t_j) = \mathbf{m} - \mathbf{i}(t_j) + \mathbf{i}(t_m)$ where transition t_m becomes enabled. The *global balance equations* for the routing process $\mathbf{y}$ are

$$\sum_{t_h \in T} f(\mathbf{i}(t_j))\mu(\mathbf{i}(t_j))P(\mathbf{i}(t_j), \mathbf{i}(t_h)) = \\ = \sum_{t_m \in T} f(\mathbf{i}(t_m))\mu(\mathbf{i}(t_m))P(\mathbf{i}(t_m), \mathbf{i}(t_j)) \qquad \forall\ t_j \in T \tag{18.8}$$

which can be interpreted as the traffic equations for the SPN. This set of linear equations is the basis for the analysis of PF-SPNs. If the SPN is characterised by nc different FR^*-classes this system of equations can be partitioned into nc independent subsystems.

The following results establish the structural constraints (Definition 18.2) that an SPN must satisfy for a positive solution, $f(\cdot)$ of the traffic equations to exist.

Define $S(\mathbf{x}) \subset \mathcal{S}$, as set of the input vectors corresponding to $[\mathbf{x}]$, as

$$S(\mathbf{x}) = \{\mathbf{i}(t_j) \mid \exists\, \mathbf{x}' \in [\mathbf{x}] \text{ such that } \mathbf{x}'[j] > 0\}\,.$$

The next theorem shows that the partition of the set of closed support minimal T-semiflows $\mathcal{X}_{\text{cd}}$ into equivalence classes $[\mathbf{x}]_{\mathbf{x} \in \mathcal{X}_{\text{cd}}}$ induces a partition $\{S(\mathbf{x})\}_{\mathbf{x} \in \mathcal{X}_{\text{cd}}}$ of $\mathcal{S}$ into irreducible sets of the Markov chain $\mathbf{y}$ if and only if the net is a Π-net.

Theorem 18.4 [Existence of Solution of the Traffic Equations] (From [4]) *Let $\langle \mathcal{N}, \mathbf{m_0}, T\rangle$ be an SPN. A necessary and sufficient condition for the existence of an invariant measure for the routing chain $\mathbf{y}$ associated with the SPN is that $\mathcal{N}$ must be a Π-net.*

Proof:
We have to observe that the routing chain $\mathbf{y}$ has an invariant measure if and only if the state space $\mathcal{S}$ contains only irreducible sets. Therefore it is sufficient to prove that to be a Π-net is a necessary and sufficient condition for the partitioning of $\mathcal{S}$ into irreducible sets $\{S(\mathbf{x})\}_{\mathbf{x} \in \mathcal{X}_{\text{cd}}}$.
For any pair of closed support minimal T-semiflows $\mathbf{x_i}, \mathbf{x_j} \in \mathcal{X}_{\text{cd}}$ if $(\mathbf{x_i}, \mathbf{x_j}) \in FR^*$ then by definition of the sets $S(\cdot)$ we have that $S(\mathbf{x_i}) = S(\mathbf{x_j})$. If $S(\mathbf{x_i}) \cap$

$S(\mathbf{x_j}) \neq \emptyset$ then $\exists\ \mathbf{i}(t_h) \in S(\mathbf{x_i}) \cap S(\mathbf{x_j})$ and $\mathbf{i}(t_h)$ belongs to both the equivalence classes $[\mathbf{x_i}]$ and $[\mathbf{x_j}]$ and hence $[\mathbf{x_i}] = [\mathbf{x_j}]$. Hence this shows that $S(\mathbf{x_i}) \cap S(\mathbf{x_j})$ if $(\mathbf{x_i}, \mathbf{x_j}) \in FR^*$.

We can see that if an SPN is a Π-net then the relation FR^* induces a partition of $\mathcal{S}$ into irreducible sets. From Perron-Frobenius theorem (cf. Seneta [36]) it follows that there exists an invariant measure for the routing chain.

Conversely, assume that there exists an invariant measure for the routing chain. This imply that $\mathcal{S}$ is partitioned into irreducible sets. Let us denote by $\mathcal{S}_i$, $i = 1, \ldots, v$, these irreducible sets. Let $t_j \in T$ be a transition and i_0 an index such that $\mathbf{i}(t_j) \in \mathcal{S}_{i_0}$. Since $\mathcal{S}_{i_0}$ is an irreducible set then for all t_h such that $\mathbf{i}(t_h) \in \mathcal{S}_{i_0}$ there exist two firing sequences σ and σ' involving transitions with input vector in $\mathcal{S}_{i_0}$ such that $\mathbf{m} + \mathbf{i}(t_j)[\sigma > \mathbf{m} + \mathbf{i}(t_h)$, and $\mathbf{m} + \mathbf{i}(t_h)[\sigma' \mathbf{m} + \mathbf{i}(t_j)$ ($\mathbf{m} + \mathbf{i}(t_j)$ and $\mathbf{m} + \mathbf{i}(t_h)$ are markings belonging to the reachability set of the SPN). Thus $\overline{\sigma} = \sigma\sigma'$ is a closed support T-semiflow. From the irreducibility we may conclude that all T-semiflows contained in $\mathcal{S}_{i_0}$ have closed support. Each support of a T-semiflow can be decomposed into a union of minimal supports (cf. Memmi and Rucariol [32]) which implies that t_j is covered by a minimal closed support T-semiflow. ♢

The traffic equations balance the average at which the input vector $\mathbf{i}(t_j)$ (for $t_j \in T$) is "absorbed" with the average rate at which the input vector $\mathbf{i}(t_j)$ is "formed". We have to point out that this form of balance requires that $\mathbf{i}(t_j)$ there exists a transition t_h such that $\mathbf{i}(t_j) = \mathbf{o}(t_h)$.

The existence of a positive solution for the (traffic) Equations (18.8) is not sufficient to assert a Product-Form Solution for the SPN. In order for this result to be true the following two additional theorems must be satisfied.

Theorem 18.5 [Equilibrium Distribution of SPN] (From [18]) *Assume that $f(\cdot)$ is a solution for the traffic equations (18.8) and that there exists a function $\pi_f(\cdot) : \mathcal{R}(N, \mathbf{m_0}) \to I\!R$ such that for all $t_j, t_m \in T$ with $P(\mathbf{i}(t_j), \mathbf{i}(t_m)) > 0$, and for all $\mathbf{n} + \mathbf{i}(t_j) \in \mathcal{R}(N, \mathbf{m_0})$ the following relation holds*

$$\frac{\pi_f(\mathbf{n} + \mathbf{i}(t_j))}{\pi_f(\mathbf{n} + \mathbf{i}(t_m))} = \frac{f(\mathbf{i}(t_j))}{f(\mathbf{i}(t_m))}, \tag{18.9}$$

then, the equilibrium distribution of the SPN is given by

$$\pi(\mathbf{m}) = \frac{1}{G} \cdot \Phi(\mathbf{m}) \cdot \pi_f(\mathbf{m}) \qquad \mathbf{m} \in \mathcal{R}(N, \mathbf{m_0}), \tag{18.10}$$

where G is a normalisation constant.

Proof:
To prove this theorem we have to show that the measure $\Phi(\mathbf{m}) \cdot \pi_f(\mathbf{m})$ satisfies the group-local-balance property (Equation (18.6)). The substitution of $\Phi(\cdot)\pi_f(\cdot)$ and the expression of the rates (18.3) yields the derivations that are

shown below.

$$\sum_{t' \neq t''} \Big\{ \Phi(\mathbf{m}) \pi_f(\mathbf{m}) q(\mathbf{i}(t'), \mathbf{i}(t''); \mathbf{m} - \mathbf{i}(t')) + \\ -\Phi(\mathbf{m} - \mathbf{i}(t') + \mathbf{i}(t'')) \pi_f(\mathbf{m} - \mathbf{i}(t') + \mathbf{i}(t'')) q(\mathbf{i}(t''), \mathbf{i}(t'); \mathbf{m} - \mathbf{i}(t')) \Big\} =$$

$$= \psi(\mathbf{m} - \mathbf{i}(t')) \sum_{t' \neq t''} \Big\{ \pi_f(\mathbf{m}) \mu(\mathbf{i}(t')) p(\mathbf{i}(t'), \mathbf{i}(t'')) + \\ -\pi_f(\mathbf{m} - \mathbf{i}(t') + \mathbf{i}(t'')) \mu(\mathbf{i}(t'')) p(\mathbf{i}(t''), \mathbf{i}(t')) \Big\} =$$

$$= \psi(\mathbf{m} - \mathbf{i}(t')) \sum_{t' \neq t''} \Big\{ \mu(\mathbf{i}(t')) p(\mathbf{i}(t'), \mathbf{i}(t'')) + \\ -\frac{\pi_f(\mathbf{m} - \mathbf{i}(t') + \mathbf{i}(t''))}{\pi_f(\mathbf{m})} \mu(\mathbf{i}(t'')) p(\mathbf{i}(t''), \mathbf{i}(t')) \Big\} =$$

$$\overset{\text{Eq. (18.9)}}{=} \psi(\mathbf{m} - \mathbf{i}(t')) \sum_{t' \neq t''} \Big\{ \mu(\mathbf{i}(t')) p(\mathbf{i}(t'), \mathbf{i}(t'')) + \\ -\frac{f(\mathbf{i}(t''))}{f(\mathbf{i}(t'))} \mu(\mathbf{i}(t'')) p(\mathbf{i}(t''), \mathbf{i}(t')) \Big\} =$$

$$= \psi(\mathbf{m} - \mathbf{i}(t')) \sum_{t' \neq t''} \Big\{ f(\mathbf{i}(t')) \mu(\mathbf{i}(t')) p(\mathbf{i}(t'), \mathbf{i}(t'')) + \\ -f(\mathbf{i}(t'')) \mu(\mathbf{i}(t'')) p(\mathbf{i}(t''), \mathbf{i}(t')) \Big\} =$$

$$\overset{\text{Eq. (18.8)}}{=} 0.$$

$\Diamond$

In classical queueing networks, the functions $\pi_f(\cdot)$ and $\Phi(\cdot)$ are often product forms overs the stations of the network and hence the subsequent product form equilibrium distribution is amenable to convolution algorithms for finding normalisation constant. To use Theorem 18.5 it is advantageous if the functions $\pi_f(\cdot)$ and $\Phi(\cdot)$ have a form such that the equilibrium distribution can be used to derive computational algorithms. This would be possible if these functions are simple product form themselves either over the transitions or over the places of the SPN. Except for special cases, the condition of Equation (18.9) does not provide a constructive method for obtaining the explicit expression of the function $\pi_f(\cdot)$. The following theorem from Coleman *et al.* [11], states that $\pi_f(\cdot)$ has a product-form over the places of the SPN whenever one additional condition holds. Let $f(\cdot)$ be a solution of the traffic equations, and define the vector $\mathbf{w}_f$ as

$$\mathbf{w}_f = \left[\log\left(\frac{f(\mathbf{i}(t_1))}{f(\mathbf{o}(t_1))} \right), \log\left(\frac{f(\mathbf{i}(t_2))}{f(\mathbf{o}(t_2))} \right), \cdots, \log\left(\frac{f(\mathbf{i}(t_{nt}))}{f(\mathbf{o}(t_{nt}))} \right) \right]. \quad (18.11)$$

There may be many functions $f(\cdot)$ that are solutions of the traffic equations. However each one is unique up to a multiplicative constant in each FR^* class. This implies that the ratio $f(\mathbf{i}(t_i))/f(\mathbf{o}(t_i))$ is invariant.

Theorem 18.6 [Product-Form for Equilibrium Distribution of SPN] (From [11]) *Assume that $f(\cdot)$ is a solution for the traffic equations. The function $\pi_f(\cdot)$ satisfying Equation (18.9) has the form*

$$\pi_f(\mathbf{m}) = \prod_{i=1}^{np} y_i^{m_i} \qquad \forall\ \mathbf{m} \in \mathcal{R}(N, \mathbf{m_0}) \tag{18.12}$$

if and only if

$$Rank([\mathbf{C}]) = Rank([\mathbf{C} \mid \mathbf{w}_f]), \tag{18.13}$$

where $[\mathbf{C} \mid \mathbf{w}_f]$ is the matrix $\mathbf{C}$ augmented with the row $\mathbf{w}_f$. In this case the np-component vector $\mathbf{r} = [\log(y_1), \ldots, \log(y_{np})]$, satisfies the matrix equation

$$-\mathbf{r} \cdot \mathbf{C} = \mathbf{w}_f. \tag{18.14}$$

Proof:
To establish when the form of $\pi_f(\mathbf{m})$, given in Equation (18.12), satisfies the condition of Theorem 18.5, we substitute Equation (18.12) into Equation (18.9) and hence

$$y_1^{-\mathbf{C}[1,j]} y_2^{-\mathbf{C}[2,j]} \cdots y_{np}^{-\mathbf{C}[np,j]} = \frac{f(\mathbf{i}(t_j))}{f(\mathbf{i}(t_m))} \qquad \forall t_j, t_m\ : p(\mathbf{i}(t_j), \mathbf{i}(t_m)) > 0.$$

Taking logs we obtain

$$-\log(y_1)\mathbf{C}[1,j] - \log(y_2)\mathbf{C}[2,j] - \cdots - \log(y_{np})\mathbf{C}[np,j] = \log\left(\frac{f(\mathbf{i}(t_j))}{f(\mathbf{i}(t_m))}\right).$$

This is a system of nt equations in np unknown which can be expressed in matrix form as Equation (18.14) that is solvable if and only if

$$Rank([\mathbf{C}]) = Rank([\mathbf{C} \mid \mathbf{w}_f]).$$

Conversely if Equation (18.14) is satisfied then $\pi_f(\mathbf{m})$ given by Equation (18.12) satisfies Equation (18.9). ♢

We can observe that $\mathbf{w}_f$ depends on the firing rates through the definition of $f(\cdot)$. When Equation (18.13) is satisfied for any $\mathbf{w}_f$ of the form (18.11), there is no restriction of the firing rates for the product form (Equation (18.12)) to exist. A product form can still exist when Equation (18.13) is satisfied only for certain values of the firing rates. When the rank condition is satisfied, we can find the functions y_i by solving Equation (18.14). There may be solutions of this matrix equation when $\log(y_i) = 0$ for some i. In this case $y_i = 1$ and no term involving y_i appears in the product (Equation (18.12)).

18.4.1 Examples

The following examples show how to use all previous concepts to find the equilibrium distribution. All the SPN considered in the following examples have marking independent transition firing rates and hence we can set $\Phi(\mathbf{m}) = 1$ $\forall\ \mathbf{m} \in \mathrm{RS}(\mathbf{m_0})$.

Example 18.1 In this example we examine the single-class closed queueing networks with exponential servers. From the result of Gordon and Newell [16] it follows that these queueing networks have product form solution.
Consider a closed network of M queues with N customers circulating in the network. The customers are served with state independent rates. Let the set of queues be denoted by $\mathcal{M}$. The state of the process is given by $\boldsymbol{n} = [n_1, \ldots, n_M]$, where n_i represents the number of customers present at queue i. The customers are served at queue i with state independent rate q_i and then move to queue to queue j with probability $p(i,j)$.
We can build an SPN such that at each transition corresponds the event of a customer transferring between two queues, i and j, for some i and j. In a such SPN the places identify the queues. Let be p_i (for $i = 1, \ldots, M$) the place that represents the queue i. For any i and j such that $p(i,j) > 0$ there must be a transition t_{i_j} with ${}^\bullet t_{i_j} = \{p_i\}$ and $t_{i_j}^\bullet = \{p_j\}$, and with $\mu_{i_j} = p(i,j) \cdot q_i$, where q_i is the service rate of the i-th queue.

It is easy to check that the the SPN representing a closed queueing network is a Π-net. In this case there exists only FR^*-class because all the transitions $t_{i_j} \in p_i^\bullet$ have the same input bag.

Figure 18.1 shows an SPN that represents a closed queueing network with three queues. The routing probabilities are $p(1,2) = \alpha$, $p(1,3) = 1-\alpha$, $p(2,1) = 1$, and $p(3,1) = 1$.

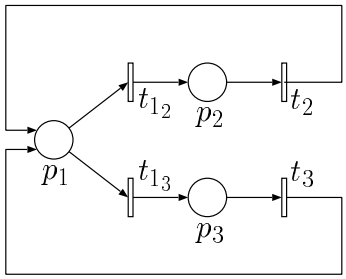


Figure 18.1: Example: An SPN that represents a closed queueing network with three queues

In the SPN of Figure 18.1 the rate of the transitions are $\mu_{1_2} = \alpha \cdot q_1$, $\mu_{1_3} = (1-\alpha) \cdot q_1$, $\mu_2 = q_2$, and $\mu_3 = q_3$. The incidence matrix $\mathbf{C}$ is given by

$$\mathbf{C} = \begin{pmatrix} -1 & -1 & 1 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{pmatrix}$$

This SPN has two minimal support T-semiflows whose support sets are $||\mathbf{x_1}|| = \{t_{1_2}, t_2\}$, and $||\mathbf{x_2}|| = \{t_{1_3}, t_2\}$. All the support sets of these T-semiflows satisfy Definition 18.1 then the SPN is a Π-net and hence there exists a positive solution for the traffic equations. Since $\mathbf{i}(t_{1_2}) = \mathbf{i}(t_{1_3})$ we simply denote as $\mathbf{i}(t_1)$ the input bag of transitions t_{1_2} and t_{1_3}. A solution for the traffic equations is $f(\mathbf{i}(t_1)) \cdot (\mu_{1_2} + \mu_{1_3}) = 1$, $f(\mathbf{i}(t_2)) \cdot \mu_2 = \alpha$, and $f(\mathbf{i}(t_3)) \cdot \mu_3 = 1 - \alpha$.
The vector $\mathbf{w}_f$ is

$$\begin{aligned}\mathbf{w}_f &= \left[\log\left(\frac{f(\mathbf{i}(t_1))}{f(\mathbf{i}(t_2))}\right), \log\left(\frac{f(\mathbf{i}(t_1))}{f(\mathbf{i}(t_3))}\right), \log\left(\frac{f(\mathbf{i}(t_2))}{f(\mathbf{i}(t_1))}\right), \log\left(\frac{f(\mathbf{i}(t_3))}{f(\mathbf{i}(t_1))}\right)\right] \\ &= \left[\log\left(\frac{\mu_2}{\alpha(\mu_{1_2}+\mu_{1_3})}\right), \log\left(\frac{\mu_3}{(1-\alpha)(\mu_{1_2}+\mu_{1_3})}\right), \log\left(\frac{\alpha(mu_{1_2}+\mu_{1_3})}{\mu_2}\right), \log\left(\frac{(1-\alpha)(\mu_{1_2}+\mu_{1_3})}{\mu_3}\right)\right].\end{aligned}$$

Let $\mathbf{w}_i$ be the i-th element of the vector $\mathbf{w}_f$. The augmented matrix $[\mathbf{C} \mid \mathbf{w}_f]$ is row equivalent to the fully row reduced matrix

$$\begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \\ \mathbf{w}_1 + \mathbf{w}_3 & \mathbf{w}_2 + \mathbf{w}_4 & \mathbf{w}_3 & \mathbf{w}_4 \end{pmatrix}$$

The two rank conditions are $\mathbf{w}_1 + \mathbf{w}_3 = 0$, and $\mathbf{w}_2 + \mathbf{w}_4 = 0$ which after the substitutions are translated in conditions on the μ.'s. In this case they are identities $\frac{\mu_2}{\alpha(\mu_{1_2}+\mu_{1_3})}\frac{\alpha(\mu_{1_2}+\mu_{1_3})}{\mu_2} = 1$, $\frac{\mu_3}{(1-\alpha)(\mu_{1_2}+\mu_{1_3})}\frac{(1-\alpha)(\mu_{1_2}+\mu_{1_3})}{\mu_3} = 1$, and imply that the PFS holds for any possible set of μ.'s. Letting $y_1 = 1$ we obtain that the product form equilibrium distribution of the SPN shown in Figure 18.1 is given by

$$\pi(\mathbf{m}) = \frac{1}{G}\left[\frac{\alpha(\mu_{1_2} + \mu_{1_3})}{\mu_2}\right]^{m_2}\left[\frac{(1 - \alpha(\mu_{1_2} + \mu_{1_3})}{\mu_3}\right]^{m_3}.$$

Example 18.2 Consider the Example taken from [39] shown in Figure 18.2.

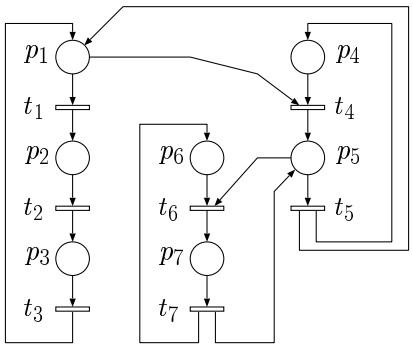


Figure 18.2: Example:

The incidence matrix $\mathbf{C}$ is given by

$$\mathbf{C} = \begin{pmatrix} -1 & 0 & 1 & -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{pmatrix}$$

This SPN has three minimal support T-semiflows whose support sets are $||\mathbf{x_1}|| = \{t_1, t_2, t_3\}, ||\mathbf{x_2}|| = \{t_4, t_5\}, ||\mathbf{x_3}|| = \{t_6, t_7\}$, all the support sets of these three T-semiflows satisfy Definition 18.1 then the SPN is a Π-net and hence there exists a positive solution for the traffic equations. A solution for these equations is $f(\mathbf{i}(t_i))\mu(\mathbf{i}(t_i)) = 1$ $(i = 1, \ldots, 7)$ and hence $f(\mathbf{i}(t_i)) = \frac{1}{\mu(\mathbf{i}(t_i))} = \frac{1}{\mu_i}$, for $i = 1, \ldots, 7$. The vector $\mathbf{w}_f$ is

$$\begin{aligned}\mathbf{w}_f &= \left[\log\left(\tfrac{f(\mathbf{i}(t_1))}{f(\mathbf{i}(t_2))}\right), \log\left(\tfrac{f(\mathbf{i}(t_2))}{f(\mathbf{i}(t_3))}\right), \log\left(\tfrac{f(\mathbf{i}(t_3))}{f(\mathbf{i}(t_1))}\right), \log\left(\tfrac{f(\mathbf{i}(t_4))}{f(\mathbf{i}(t_5))}\right), \log\left(\tfrac{f(\mathbf{i}(t_5))}{f(\mathbf{i}(t_4))}\right), \log\left(\tfrac{f(\mathbf{i}(t_6))}{f(\mathbf{i}(t_7))}\right), \log\left(\tfrac{f(\mathbf{i}(t_7))}{f(\mathbf{i}(t_6))}\right)\right] \\ &= \left[\log\left(\tfrac{\mu_2}{\mu_1}\right), \log\left(\tfrac{\mu_3}{\mu_2}\right), \log\left(\tfrac{\mu_1}{\mu_3}\right), \log\left(\tfrac{\mu_5}{\mu_4}\right), \log\left(\tfrac{\mu_4}{\mu_5}\right), \log\left(\tfrac{\mu_7}{\mu_6}\right), \log\left(\tfrac{\mu_6}{\mu_7}\right)\right].\end{aligned}$$

Let $\mathbf{w}_i$ be the i-th element of the vector $\mathbf{w}_f$. The augmented matrix $[\mathbf{C} \mid \mathbf{w}_f]$ is row equivalent to the fully row reduced matrix

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ \mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}_3 & \mathbf{w}_2 & \mathbf{w}_3 & \mathbf{w}_4 + \mathbf{w}_5 & \mathbf{w}_5 & \mathbf{w}_6 + \mathbf{w}_7 & \mathbf{w}_7 \end{pmatrix}$$

The three rank conditions are $\mathbf{w}_1 + \mathbf{w}_2 + \mathbf{w}[3] = 0$, $\mathbf{w}_4 + \mathbf{w}_5 = 0$, $\mathbf{w}[6] + \mathbf{w}[7] = 0$, which after the substitutions are translated in conditions on the μ.'s. In this case they are identities $\frac{\mu_2}{\mu_1}\frac{\mu_3}{\mu_2}\frac{\mu_1}{\mu_3} = 1$, $\frac{\mu_5}{\mu_4}\frac{\mu_4}{\mu_5} = 1$, $\frac{\mu_7}{\mu_6}\frac{\mu_6}{\mu_7} = 1$, and imply that the PFS holds for any possible set of μ.'s. Letting $y_1 = y_5 = y_7 = 1$ gives, we obtain that the product form equilibrium distribution of the SPN shown in Figure 18.2 is given by

$$\pi(\mathbf{m}) = \frac{1}{G}\left[\frac{\mu_1}{\mu_2}\right]^{m_2}\left[\frac{\mu_1}{\mu_3}\right]^{m_3}\left[\frac{\mu_5}{\mu_4}\right]^{m_4}\left[\frac{\mu_7}{\mu_6}\right]^{m_6}.$$

Example 18.3 The SPN shown in Figure 18.3 example, taken form [11], represents an SPN in which the rank condition is not trivially satisfied. The

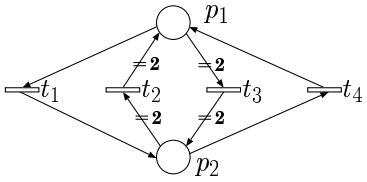


Figure 18.3: Example:

incidence matrix $\mathbf{C}$ is given by,

$$\mathbf{C} = \begin{pmatrix} -1 & 2 & -2 & 1 \\ 1 & -2 & 2 & -1 \end{pmatrix}$$

This SPN is covered by four minimal T-semiflows whose support sets are $||\mathbf{x_1}|| = \{t_1, t_4\}$, $||\mathbf{x_2}|| = \{t_2, t_3\}$, $||\mathbf{x_3}|| = \{2t_1, t_2\}$, and $||\mathbf{x_4}|| = \{t_3, 2t_4\}$. Only the T-semiflows $\mathbf{x_1}$ and $\mathbf{x_2}$ satisfy Definition 18.1 but all the transitions of the net

have been covered by a minimal closed support T-semiflow then the SPN is a Π-net and hence there exists a positive solution for the traffic equations. In particular we obtain $f(\mathbf{i}(t_i)) = \frac{1}{\mu_i}$ for $i = 1, \ldots, 4$. The vector $\mathbf{w}_f$ is given by

$$\begin{aligned} \mathbf{w}_f &= \left[\log\left(\tfrac{f(\mathbf{i}(t_1))}{f(\mathbf{i}(t_4))}\right), \log\left(\tfrac{f(\mathbf{i}(t_2))}{f(\mathbf{i}(t_3))}\right), \log\left(\tfrac{f(\mathbf{i}(t_3))}{f(\mathbf{i}(t_2))}\right), \log\left(\tfrac{f(\mathbf{i}(t_4))}{f(\mathbf{i}(t_1))}\right)\right] \\ &= \left[\log\left(\tfrac{\mu_4}{\mu_1}\right), \log\left(\tfrac{\mu_3}{\mu_2}\right), \log\left(\tfrac{\mu_2}{\mu_3}\right), \log\left(\tfrac{\mu_1}{\mu_4}\right)\right]. \end{aligned}$$

Let $\mathbf{w}_i$ be the i-th element of the vector $\mathbf{w}_f$. The augmented matrix $[\mathbf{C} \mid \mathbf{w}_f]$ is row equivalent to the fully row reduced matrix

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ \mathbf{w}_1 & \mathbf{w}_2 + 2\mathbf{w}_1 & \mathbf{w}_3 - 2\mathbf{w}_1 & \mathbf{w}_1 + \mathbf{w}_4 \end{pmatrix}.$$

The rank conditions are $\mathbf{w}_2 + 2\mathbf{w}_1 = 0$, $\mathbf{w}_3 - 2\mathbf{w}_1 = 0$, and $\mathbf{w}_1 + \mathbf{w}_4 = 0$, which implies, $\frac{f(\mathbf{i}(t_2))}{f(\mathbf{i}(t_3))}\left(\frac{f(\mathbf{i}(t_1))}{f(\mathbf{i}(t_4))}\right)^2 = 1$, $\frac{f(\mathbf{i}(t_3))}{f(\mathbf{i}(t_2))}\left(\frac{f(\mathbf{i}(t_4))}{f(\mathbf{i}(t_1))}\right)^2 = 1$, and $1 = 1$ respectively. The third rank condition is clearly always satisfied and the first and second are the same and arise because there is more than one way to effect the same change of marking. Substituting for the $f(\cdot)$, the rank condition becomes

$$\frac{\mu_2}{\mu_3} = \left(\frac{\mu_4}{\mu_1}\right)^2.$$

The previous equations is a conditional requirement upon the transition rates. Letting $y_2 = 1$ gives $y_1 = \frac{f(\mathbf{i}(t_1))}{f(\mathbf{i}(t_4))}$, and $y_2 = 1$ and hence,

$$\pi_f(\mathbf{m}) = \left[\frac{\mu_4}{\mu_1}\right]^{m_1}.$$

18.5 Considerations on the classes of PF-SPNs

We can classify the product form criteria in two types: *dynamical* and *statical*. To the first group belong the product form criteria proposed by Lazar and Robertazzi and by Li and Georganas. The term dynamical means that the product form depends on the property that can be verified at level of the state space. In principle, product forms of this type can be marking dependent properties. Donatelli and Sereno [13] give an example of a net that have product form solution, of the type identified at level of the state space, only for some initial markings and not others.

To the second group of product form criteria belong the proposals of Frosh, of Henderson *et al.*, and of Boucherie. In each case the structure of the SPN define the product form solution.

It is worthwhile to point out that the statical criteria are much more powerful that the dynamical counterpart. These criteria avoid the exploration of the state space and then they can be used to overcome the state space explosion problem.

Although there are several examples that can be characterised by all the PF criteria (the dining philosopher problem is one of these) a formal comparison among the different proposals has not been given yet. The only partial answer to this point has been done in [13]. In that paper there is an attempt to compare the Lazar and Robertazzi PF with the one proposed by Henderson *et al.*.
An issue worth being investigating is the search for a common environment among the different PF criteria proposed for queueing networks, for stochastic Petri nets, and recently also for stochastic process algebras [17, 37].

Another interesting problem is the comparison between product form criteria for SPNs and those for queueing networks. In particular what are the product form queueing networks that can be represented by means of PF-SPNs. From Example 18.1 we have seen that the queueing networks characterised by Gordon and Newell [16] can be represented by PF-SPNs. To be precise we have to point out the limitation on the service policy discipline, i. e., with SPNs we can use only random order server discipline. We can also note that in the class of PF-SPNs there are nets that are not characterised by the Gordon and Newell product form criterion. The nets of Figure 18.2 and Figure 18.3 are SPNs of this type. However we must point out that using SPNs it is not possible to represent multi-class queueing networks and hence due this limitation with PF-SPNs we cannot represent multi-class product form solution queueing networks.

In the following we focus our attention on the product form proposed by Henderson *et al.*. We call the SPNs characterised by this criterion PF-SPNs. It is rather natural to address on the modeling power of the PF-SPNs. In particular it useful to investigate the comparison of this class with the PF queueing networks. To this aim we can use the examples presented in Section 18.4.1. In particular

18.6 Structural Properties of Π-nets

In Petri net literature the classes of nets (or systems) that can be defined by imposing certain restrictions on the net are called *syntactical classes*. In this section we discuss the qualitative properties of the Π-nets.

Property 18.7 *A If* $\mathbf{x}$ *is a closed support minimal T-semiflow then* $x \in \{0,1\}^{|T|}$.

It is important to note that if a net is a Π-net this implies that T, the set of all transitions, is closed, but the reverse is not true. Indeed the set of transitions of net of Figure 18.4 is closed but the net does not satisfy Definition 18.2. This net is covered by 3 minimal T-semiflows $\mathbf{x_1} = [1,0,0,1,0]$, $\mathbf{x_2} = [0,0,1,0,1]$, and $\mathbf{x_3} = [1,2,0,0,1]$. We can see that $\mathbf{x_1}$ and $\mathbf{x_2}$ have closed support but the support of $\mathbf{x_3}$ is not closed. Definition 18.2 is not satisfied by transition t_2 that is covered only by $\mathbf{x_3}$. For this SPN it is possible to see that the traffic equations (Eqs. 18.8) do not have a solution and hence this net does not have PFS.

Proposition 18.8 *From the definition of* Π*-nets it follows that:*

1. *If* $\mathcal{N}$ *is a* Π*-net, the reverse* $\mathcal{N}^{-1}$ *belongs to the same class.*

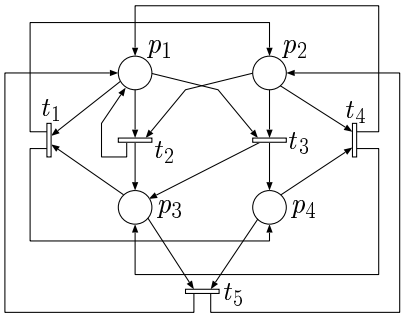


Figure 18.4: The set transitions is closed, but the net is not a Π-nets.

2. *The class of* Π*-nets is not immediately comparable with the well known classes of Petri nets:*
   - *if* $\mathcal{N}$ *is a WTS and a* Π*-net, then it is a MG (see Figure 18.5);*
   - $SM \subset \Pi - net$*;*
   - *there are MGs that are not* Π*-nets (see Figure 18.6(a));*
   - *there are* Π*-nets that are non EFC nets (see Figure 18.6(b)).*

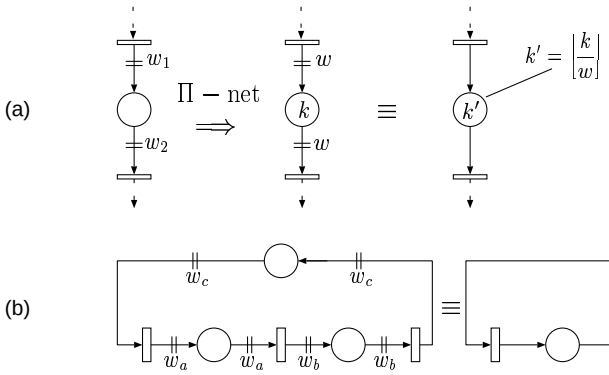


Figure 18.5: Weighted Π-cycles are just equivalent to ordinary cycles

Membership Problem

From the definition of Π-net we can easely decide if a given net falls in this class. The problem that arises is the complexity of a straightforward application of Definition 18.2 because the number of minimal T-semiflows can be exponential in the number of transitions (e.g., [31]). Now, using Property 18.7, we derive an algorithm (Table 18.1) that allows to recognise if a net is a Π-net in polynomial time.

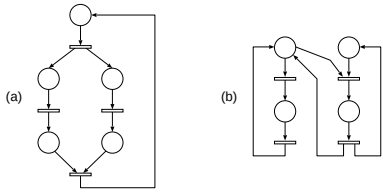


Figure 18.6: (a) A non Π-(marked) graph; (b) A non EFC Π-net

From simple considerations we can see that both the inner and the outer cycles require $O(|T|)$ steps. Hence the complexity of the algorithm that allows to recognise if a given net satisfy Definition 18.2 requires $O(|T|^2)$ steps.

18.6.1 Functional Properties of Π-nets

By definition the Π-nets are consistent.

Lemma 18.9 *Let $\Sigma = (\mathcal{N}, \mathbf{m_0})$ be a Π-system. If $t \in T$ is enabled at $\mathbf{m} \in R(\mathcal{N}, \mathbf{m_0})$, then for any t' belonging to the same closed support T-semiflow of t, $\exists\ \mathbf{m}' \in R(\mathcal{N}, \mathbf{m})$ such that t' is enabled in $\mathbf{m}'$. That is, if in a given marking $\mathbf{m} \in R(\mathcal{N}, \mathbf{m_0})$ a transition t belonging to a closed support T-semiflow is enabled then all the transitions of the same closed support T-semiflow can be fired.*

Proof:
Let $\mathbf{m}$ be a marking that enables transition t. From Definition 18.1 it follows that there exists a transition t' such that $\mathbf{o}(t) = \mathbf{i}(t')$, hence the firing of t yields the a marking $\mathbf{m}'$ that enables t'. The firing of t' enables t'' and so on. ♢

Corollary 18.10 *Let $\Sigma = (\mathcal{N}, \mathbf{m_0})$ be a Π-system. If $t \in T$ can be fired at $\mathbf{m} \in R(\mathcal{N}, \mathbf{m_0})$, then there is a firing sequence that fires all the remaining transitions in the FR^*-class at which t belongs.*

Proof:
The proof follows from the previous lemma from Definition 18.3 and from Property 18.7. ♢

Proposition 18.11 *Let $\mathcal{N}$ be a Π-net and $\Sigma = (\mathcal{N}, \mathbf{m_0})$ be a Π-system.*

1. *If $t \in T$ is enabled at $\mathbf{m_0}$ then Σ is DF.*
2. *If Σ is DF then it is reversible.*
3. *$\mathcal{N}$ is SL.*
4. *Σ is live live iff any FR^*-class can be firable.*

```
Procedure Verify Π-net
        L ← T
fail ← false
while (L ≠ ∅) and (fail ≠ true) do
        Let be t ∈ L
        M ← {t}
        t_start ← t
        t_c ← t
        cycle ← false
        repeat
                Let t_s ∈ T be such that i(t_s) = o(t_c)
                if (∄ such transition) then fail ← true
                else
                        if (t_s ∉ M) then
                          begin
                                M ← M ∪ {t_s}
                                t_c ← t_s
                          end
                        if (t_s ∈ M) and (t_s ≠ t_start) then fail ← true
                        if (t_s = t_start) then
                          begin
                                L ← L − M
                                cycle ← true
                          end
        until (cycle = true) or (fail = true)
endwhile
if (fail = false) then "The net is a Π-net"
else "The net is not a Π-net"
end Verify Π-net
```

Table 18.1: Procedure *Verify* Π-net

5. *If* Σ *is live then the* $\Sigma' = (\mathcal{N}, M_0')$ *with* $\mathbf{m_0} \leq M_0'$ *is live too (i.e., liveness monotonic w.r.t the marking in the net).*

Proof:

1. The proof of this statement immediately follows from Lemma 18.9 because all the transitions belonging to the same FR^*-class of t can be fired infinitely often.

2. For any $\mathbf{m} \in R(\mathcal{N}, \mathbf{m_0})$ there is a finite firing sequence $\sigma = t_{\delta_1}, t_{\delta_2}, \ldots, t_{\delta_l}$ such that $\mathbf{m_0}[t_{\delta_1} > \mathbf{m_1} \ldots \mathbf{m_{l-1}}[t_{\delta_l} > \mathbf{m}$. Now we have to prove that there is a finite firing sequence η such that $\mathbf{m}[\eta > \mathbf{m_0}$. Let $\mathbf{x}$ be a closed support T-semiflow (not necessarily minimal) such that $\mathbf{x} \geq \vec{\sigma}$. From Lemma 18.9 it follows that from $\mathbf{m}$, $\mathbf{x} - \vec{\sigma}$ must be firable and hence $\mathbf{m}[\mathbf{x} - \vec{\sigma} > \mathbf{m_0}$.

3. We have to prove that exists an initial marking $\mathbf{m_0}$ such that the Π-system is live. Let $\mathbf{m_0}$ be a marking that that enables at least a transition in each FR^*-class. In this case from Lemma 18.9 it follows that $\forall\, t \in T$ there exists a finite firing sequence δ such that $\mathbf{m_0}[\delta > \mathbf{m}$ and $\mathbf{m}$ enables t.
   We have that $\forall\, t \in T, \mathbf{m}' \in R(\mathcal{N}, \mathbf{m_0})$ from statement (2) of this proposition it follows that there is a finite firing sequence η such that $\mathbf{m}'[\eta > \mathbf{m_0}$.

Since we know that $\mathbf{m_0}$ enables at least a transition in each FR^*-class using Lemma 18.9 we can conclude the proof.

4. If the system is live then all transitions can be firable. If all the FR^*-classes can be firable the system should be live.

5. The liveness is equivalent to the firability of every FR^*-class. The firability of FR^*-classes is monotonic w.r.t. the marking in the net.

♢

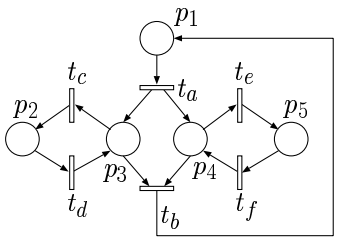


Figure 18.7:

Remark. It is important to point out that the deadlock freeness is different from the liveness. This can be observed in the Π-net of Figure 18.7 when the initial marking is $\mathbf{m_0} = [0, 1, 0, 0, 0]$.

Proposition 18.12 *(Reverse systems) Let* $\Sigma = (\mathcal{N}, \mathbf{m_0})$ *be a* Π*-system*

1. $\Sigma = (\mathcal{N}, \mathbf{m_0})$ *is deadlock free iff* $\Sigma^{-1} = (\mathcal{N}^{-1}, \mathbf{m_0})$ *is deadlock free.*

2. Π*-system* $\Sigma = (\mathcal{N}, \mathbf{m_0})$ *is live iff* $\Sigma^{-1} = (\mathcal{N}^{-1}, \mathbf{m_0})$ *is live.*

3. *The reachability graph of* $\Sigma = (\mathcal{N}, \mathbf{m_0})$ *is equal to the reverse of the reachability graph of* $\Sigma^{-1} = (\mathcal{N}^{-1}, \mathbf{m_0})$.

18.6.2 Reachability and Potential Reachability

In this section we discuss the properties of the Π-nets. These properties play an important role for the development of the computational algorithms for PF-SPNs. This role will be discussed in the next sub-section.

Let's start with showing that $R(\mathcal{N}, \mathbf{m_0}) \neq PR^B(\mathcal{N}, \mathbf{m_0})$ in Π-systems. This happens even in such simple case as the Π-cycle of Figure 18.8: the dead marking $\mathbf{m_1} = [1, 1]^T$ has the same dot-product with the P-semiflow $\mathbf{Y} = [1, 1]^T$ as the live one $\mathbf{m_0} = [2, 0]$ although there is no $\vec{\sigma} \in I\!N^{|T|}$ satisfying the state equation $\mathbf{m_1} = \mathbf{m_0} + \mathbf{C} \cdot \vec{\sigma}$.

We now address the problem of characterising the reachability set of a Π-net in terms of potential reachability set. The potential reachability set $PR(\mathcal{N}, \mathbf{m_0})$ is the set of vectors $\mathbf{m} \in I\!N^{|P|}$ such that $\exists\ \vec{\sigma}\ \in I\!N^{|T|} : \mathbf{m_0} + \mathbf{C} \cdot \vec{\sigma} = \mathbf{m}$.

Using the previous results it is possible to show that:

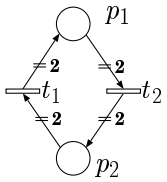


Figure 18.8: Π-net with $R(\mathcal{N}, \mathbf{m_0}) \neq PR^B(\mathcal{N}, \mathbf{m_0})$

Proposition 18.13 *1. The potential reachability graph of* $(\mathcal{N}, \mathbf{m_0})$ *is equal to the reverse of the potential reachability graph of* $(\mathcal{N}^{-1}, \mathbf{m_0})$.

2. The spurious solutions (if there are) cannot be transient, i.e., if $\mathbf{m} \in PR(\mathcal{N}, \mathbf{m_0})$ *but* $\mathbf{m} \notin R(\mathcal{N}, \mathbf{m_0})$, *then there is no firing sequence* σ *such that* $\mathbf{m}[\sigma > \mathbf{m}'$ *with* $\mathbf{m}' \in R(\mathcal{N}, \mathbf{m_0})$.

Proof:
(1) is true for any consistent net. To prove the (2) it is sufficient to observe that a transient spurious solution would be a deadlock marking in the reverse net, and this is in contradiction with Proposition 18.12. ♢

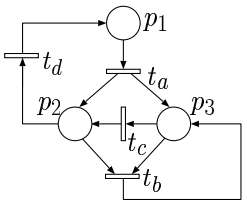


Figure 18.9: Unbounded Π-system may have killing spurious solution

The net of Figure 18.9 gives a partial negative result. For the unbounded Π-net it is possible to see that $\mathbf{m} = [0, 0, 0]$ is a spurious solution. We can see that for any initial marking $\mathbf{m_0} = [k_1, k_2, k_3]$ $\mathbf{m_0}[t_a^{k_1} > \mathbf{m_1} = [0, k_1 + k_2, k_1 + k_3][t_b^{k_1+k_2} > \mathbf{m_2} = [0, 0, 2k_1 + k_2 + k_3]$, setting $k = 2k_1 + k_2 + k_3$ we have that $\mathbf{m_2}[(t_c t_b)^{k-1} > \mathbf{m_3} = [0, 0, 1]$. Now "firing" $t_b t_c$ the null marking is spuriously reached.

The net of Figure 18.10 gives another and definitive negative result. This Π-net is bounded but it is possible to see that $M = [0, 0, 1, 0, 1]$ is a spurious solution. We can see that from the initial marking $M_0 = [0, 1, 0, 0, 1]$ and with the "firing" of t_b we obtain the marking $[0, 0, 1, 0, 1]$ that it is not a reachable marking. Hence we can state that $M \in PR(\mathcal{N}, M_0)$ but $M \notin R(\mathcal{N}, M_0)$.

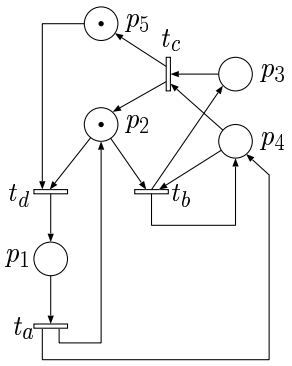


Figure 18.10: Bounded Π-system with killing spurious solution

Reachability properties and Computational Algorithms

In this section we discuss the role that the reachability properties for the development of the computational algorithms for PF-SPNs. For the development of these algorithms the reachability set of the SPN must be partitioned according certain criteria depending on the particular algorithm. For instance, the normalisation constant algorithm requires a partitioning of the reachability set that groups together all the markings that are characterised by a constant number of tokens in a given place. As can be seen in the next sections the reachability properties are used to characterise the subsets of markings corresponding to the partitions of the reachability set.

In the literature there are several proposals of computational algorithms for Product Form Solution SPNs. The reachability characterisation that is used is that one based on the minimal P-semiflows. As we have shown before (see Figure 18.8) there is no available characterisations of the reachability properties for this class of SPNs. The problems that arise from this peculiarity will be discussed in the next sections.

In the following we denote by $\mathbf{S}$ the matrix whose rows are the minimal P-semiflows of the net. In the following we will refer to this matrix as the *P-semiflow matrix*. In the rest of this chapter we restrict our attention to nets that are live and bounded and that are characterised by the fact that

$$PR^B(\mathcal{N}, \mathbf{m_0}) = R(\mathcal{N}, \mathbf{m_0}). \tag{18.15}$$

These SPNs are called *S-invariant reachable*. The term $\mathbf{S} \cdot \mathbf{m_0}$ identifies a set of invariant relations that are direct consequences of the P-semiflows of the net. The vector $\mathbf{k_0} = \mathbf{S} \cdot \mathbf{m_0}$ is called the *load vector*.

Given an S-invariant reachable Petri net $\mathcal{N}$ with initial marking $\mathbf{m_0}$ and load vector $\mathbf{k_0} = \mathbf{S} \cdot \mathbf{m_0}$, the following notation represents an alternative way of denoting its reachability set:

$$\mathcal{E}(\mathbf{k_0}, P) = \{\mathbf{m} | \mathbf{S} \cdot \mathbf{m} = \mathbf{k_0}\}.$$

Other useful definitions that derive from the reachability criterion based on the P-semiflow matrix are that one of *structural marking bound* and of *marking set*. The structural marking bound of an S-invariant reachable Petri net with initial marking $\mathbf{m_0}$, load vector $\mathbf{k_0}$, and P-semiflow matrix $\mathbf{S}$, the *structural marking bound* of a place p_i (denoted as $mx_i(\mathbf{k_0})$) is defined as

$$mx_i(\mathbf{k_0}) = \max_j \{j : \mathbf{k_0} - j \cdot \mathbf{e_i} \cdot \mathbf{S_i} \geq \mathbf{0}\}, \tag{18.16}$$

where $\mathbf{e_i}$ is the unit vector with a one in correspondence of place p_i, and $\mathbf{S_i}$ is the column corresponding to place p_i within matrix $\mathbf{S}$.
The marking set is defined as

$$M_i(\mathbf{k_0}) = \{l : \exists\, \mathbf{m} \in \mathcal{E}(\mathbf{k_0}, P) \text{ and } m_i = l\}.$$

The structural marking bound of a place in an S-invariant reachable marked net represents an upper bound for the number of tokens that can be contained in that place for any marking belonging to the potential reachability set, while in the marking set of a place there are all the possible number of tokens that can be in that place. A definition of structural marking bound based on a different reachability criterion can be found in [8].

For the development of the normalisation constant algorithm it is necessary an algebraic characterisation of the state space that accounts also subset of places. To this aim we generalise the notation of the state space to any subset of places $\wp \subseteq P$ and any load vector $\mathbf{k} \leq \mathbf{k_0}$ as follows

$$\mathcal{E}(\mathbf{k}, \wp) = \{\mathbf{m} | \mathbf{S} \cdot \mathbf{m} = \mathbf{k} \;:\; m_i = 0 \,\forall\, p_i \notin \wp\}.$$

Similar generalisations are needed also for the structural marking bound and for the marking set. In this case we have that

$$mx_i(\mathbf{k}, \wp) = \max_j \{j : \exists\, \mathbf{m} \in \mathcal{E}(\mathbf{k}, \wp) \text{ with } m_i = j\} \tag{18.17}$$

and

$$M_i(\mathbf{k}, \wp) = \{l : \exists\, \mathbf{m} \in \mathcal{E}(\mathbf{k}, \wp) \text{ and } m_i = l\}. \tag{18.18}$$

Chapter 19

Computational Algorithms for Product Form Solution Stochastic Petri Nets

19.1 The Normalisation Constant Calculus

The straightforward computation of the normalisation constant G that derives from Equation (18.12)

$$G = \sum_{\mathbf{m} \in \mathcal{R}(\mathcal{N}, \mathbf{m_0})} \prod_{i=1}^{np} y_i^{m_i} \tag{19.1}$$

is still exponentially complex and a calculus based on relationships among normalisation constants computed for smaller set of places of the net is needed in order to overcome this problem.

19.1.1 The Recursive Expressions

In a SPN satisfying the hypothesis of Theorems 18.5 and 18.6 we can express the equilibrium distribution as,

$$\pi(\mathbf{m}) = \frac{1}{G} \cdot \prod_{i=1}^{np} y_i^{m_i}, \tag{19.2}$$

and hence,

$$G = \sum_{\mathbf{m} \in \mathcal{E}(\mathbf{k_0}, P)} \prod_{i=1}^{np} y_i^{m_i}.$$

The normalisation constant depends on the number of places, and on the load vector $\mathbf{k_0}$, through the initial marking $\mathbf{m_0}$ (recall that $\mathbf{k_0} = \mathbf{S} \cdot \mathbf{m_0}$). We can

thus express this dependency explicitly by defining an auxiliary function $g(\mathbf{k}, \wp)$, where $\mathbf{k} \leq \mathbf{k_0}$ and $\wp \subseteq P$, as

$$g(\mathbf{k}, \wp) = \sum_{\mathbf{m} \in \mathcal{E}(\mathbf{k}, \wp)} \prod_{p_i \in \wp} y_i^{m_i}, \tag{19.3}$$

from which follows that

$$G = g(\mathbf{k_0}, P). \tag{19.4}$$

The following theorem derives a convolution equation by sub–dividing the state space and by conditioning on the number of tokens in one particular place.

Theorem 19.1 [Normalisation Constant (I)] *The following relationship exists between the normalisation constant of a S-invariant reachable PF-SPN with a set of places P and a load vector $\mathbf{k_0}$, and the normalisation constants of SPN with smaller set of places and smaller load vectors. Let be $\mathbf{k}$ and $\wp$ respectively a load vector and a subset of places, with $\mathbf{k} \leq \mathbf{k_0}$ and $\wp \subseteq P$, we have that*

$$g(\mathbf{k}, \wp) = \sum_{j \in M_i(\mathbf{k}, \wp)} y_i^j \, g(\mathbf{k} - j \cdot \mathbf{S_i}, \wp - \{p\}) \qquad p_i \in \wp \tag{19.5}$$

where

$$g(\mathbf{0}, \wp) = 1 \tag{19.6}$$

$$g(\mathbf{k}, \{p_i\}) = \sum_{j \in M_i(\mathbf{k}, \{p_i\})} y_i^j \,. \tag{19.7}$$

$$g(\mathbf{k}, \wp) = 0 \qquad \forall \qquad \mathbf{k} : \mathcal{E}(\mathbf{k}, \wp) = \emptyset. \tag{19.8}$$

Proof:
Consider the partitioning of the reachability set $\mathcal{E}(\mathbf{k}, \wp)$ given by

$$\mathcal{E}(\mathbf{k}, \wp) = \bigcup_{j \in M_i(\mathbf{k}, \wp)} \{\mathbf{m} \in \mathcal{E}(\mathbf{k}, \wp) \text{ with } m_i = j\}$$

The auxiliary function can be rewritten in the following manner

$$g(\mathbf{k}, \wp) \quad = \quad \sum_{j \in M_i(\mathbf{k}, \wp)} \left\{ \sum_{\substack{\mathbf{m} \in \mathcal{E}(\mathbf{k}, \wp) \\ m_i = j}} \prod_{p_l \in \wp} y_l^{m_l} \right\}.$$

The reachability set characterisation $R(\mathcal{N}, \mathbf{m_0}) \neq PR^B(\mathcal{N}, \mathbf{m_0})$ gives that

$$\{\mathbf{m} \in \mathcal{E}(\mathbf{k}, \wp) \text{ with } m_i = j\} = \{\mathbf{m}' + j \cdot \mathbf{e_i} \text{ with } \mathbf{m}' \in \mathcal{E}(\mathbf{k} - j \cdot \mathbf{S_i}, \wp - \{p_i\})$$

and hence

$$\begin{aligned} g(\mathbf{k}, \wp) \quad &= \quad \sum_{j \in M_i(\mathbf{k}, \wp)} y_i^j \left\{ \sum_{\mathbf{m} \in \mathcal{E}(\mathbf{k} - j \cdot \mathbf{S_i}, \wp - \{p_i\})} \prod_{p_l \in \wp - \{p_i\}} y_l^{m_l} \right\} \\ &= \quad \sum_{j \in M_i(\mathbf{k}, \wp)} y_i^j \, g(\mathbf{k} - j \cdot \mathbf{S_i}, \wp - \{p_i\}). \end{aligned}$$

When $\mathbf{k} = \mathbf{0}$ the only marking in $\mathcal{E}(\mathbf{k}, \wp)$ is the zero marking. Substituting the zero marking in

$$g(\mathbf{k}, \wp) = \sum_{\mathbf{m} \in \mathcal{E}(\mathbf{k}, \wp)} \prod_{p_l \in \wp} y_l^{m_l}$$

we obtain

$$g(\mathbf{0}, \wp) = 1.$$

When only place p_i is left in the sub net it is immediate to derive that

$$g(\mathbf{k}, \{p_i\}) = \sum_{j \in M_i(\mathbf{k}, \{p_i\})} y_i^j.$$

The last boundary condition follows when the load vector $\mathbf{k}$ is such that there is no marking $\mathbf{m}$ in the set $\mathcal{E}(\mathbf{k}, \wp)$. $\diamondsuit$

Equation (19.5) has the structure of a convolution and this is the reason for calling the algorithms derived from equations of this type *Convolution Algorithms*. It is important to note that in the previous theorem we have to compute the marking sets $M_i(\mathbf{k}, \wp)$ for each place $p_i \in \wp$. Finding these marking sets involves the solution of an equal number of systems of linear Diophantine equations. In the following we will show how it is possible to use the special feature of this kind of SPNs (S-invariant reachable) to make the solution of this problem computationally feasible (and simple).

Theorem 19.2 [Normalisation Constant (II)] *Given a S-invariant reachable SPN with PFS and load vector* $\mathbf{k_0}$. *Then the following relationship exists between the normalisation constant of a PFS S-invariant reachable SPN with a set of places* P *and a load vector* $\mathbf{k_0}$, *and the normalisation constants of SPN with smaller set of places and smaller load vectors. Let be* $\mathbf{k}$ *and* $\wp$ *respectively a load vector and a subset of places, with* $\mathbf{k} \leq \mathbf{k_0}$ *and* $\wp \subseteq P$, *we have that*

$$g(\mathbf{k}, \wp) = g(\mathbf{k}, \wp - \{p_i\}) + y_i \cdot g(\mathbf{k} - \mathbf{S_i}, \wp). \tag{19.9}$$

Proof:
From Theorem 19.1 we have that

$$g(\mathbf{k}, \wp) \quad = \quad \sum_{j \in M_i(\mathbf{k}, \wp)} y_i^j g(\mathbf{k} - j \cdot \mathbf{S_i}, \wp - \{p_i\}).$$

Let $mx_i(\mathbf{k}, \wp)$ be structural marking bound of the place p_i with a load vector $\mathbf{k}$ and a subset of places $\wp$,

$$CM_i(\mathbf{k}, \wp) = \{l : 0 \leq l \leq mx_i(\mathbf{k}, \wp)\}$$

candidate marking set of place p_i, given $\wp$ and $\mathbf{k}$. Obviously we have that $M_i(\mathbf{k}, \wp) \subseteq CM_i(\mathbf{k}, \wp)$. For any $h \in \{CM_i(\mathbf{k}, \wp) - M_i(\mathbf{k}, \wp)\}$ we have that

$\mathcal{E}(\mathbf{k} - h \cdot \mathbf{S_i}, \wp) = \emptyset$, and hence from (19.8) it follows that

$$\begin{aligned} g(\mathbf{k}, \wp) &= \sum_{j=0}^{mx_i(\mathbf{k},\wp)} y_i^j \, g(\mathbf{k} - j \cdot \mathbf{S_i}, \wp - \{p_i\}) \\ &= g(\mathbf{k}, \wp - \{p_i\}) + \sum_{j=1}^{mx_i(\mathbf{k},\wp)} y_i^j \, g(\mathbf{k} - j \cdot \mathbf{S_p}, \wp - \{p_i\}). \end{aligned}$$

Since we have that

$$\begin{aligned} \sum_{j=1}^{mx_i(\mathbf{k},\wp)} y_i^j \, g(\mathbf{k} - j \cdot \mathbf{S_i}, \wp - \{p_i\}) &= y_i \times \\ &\times \sum_{j=1}^{mx_i(\mathbf{k},\wp)} y_i^{j-1} g(\mathbf{k} - j \cdot \mathbf{S_i} - (j-1) \cdot \mathbf{S_i}, \wp - \{p_i\}) \\ &= y_i \cdot \sum_{h=0}^{mx_i(\mathbf{k},\wp)-1} y_i^h g(\mathbf{k}' - h \cdot \mathbf{S_i}, \wp - \{p_i\}) \\ &= y_i \cdot g(\mathbf{k}', \wp), \end{aligned}$$

with $\mathbf{k}' = \mathbf{k} - \mathbf{S_i}$, the theorem is proved. ♢

This recursive expression for $g(\mathbf{k}, \wp)$ provides the key for the construction of an algorithm to compute the desired normalisation constant without having to generate the whole reachability set of the SPN.

The Algorithm

Similarly to what has been done for the convolution method for Product Form Queueing Networks [7], the operations of the convolution algorithm for PF-SPNs can be described with the help of a two dimensional "tableau" with as many columns as there are places in the SPN and as many rows as there are possible load vectors from an empty net to a net with initial marking $\mathbf{m_0}$. The first row and the first column of the tableau are initialised according to Eq. (19.6) and (19.8). The tableau is filled column-wise starting from the upper left corner. Given an S-invariant PF-SPN with load vector $\mathbf{k_0}$, let $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l_{mx}}$ be the sequence of all load vectors $\mathbf{l}_i \leq \mathbf{k_0}$ such that $\mathbf{l_0} = [0, 0, \ldots, 0]$, $\mathbf{l_{mx}} = \mathbf{k_0} = [k_1, k_2, \ldots, k_{nq}]$ (nq is the number of minimal P-semiflow)[1]. The length of this sequence is $\prod_{i=1}^{nq}(k_i + 1)$. Using Theorems 19.1 and 19.2 we can derive the algorithm. At the step corresponding to the set of places $\{p_1, \ldots, p_i\}$, the normalisation constant $g(\mathbf{l_h}, \{p_1, \ldots, p_i\})$ can be computed using Theorem 19.2, and hence $g(\mathbf{l_h}, \{p_1, \ldots, p_i\}) = g(\mathbf{l_h}, \{p_1, p_2, \ldots, p_{i-1}\}) + y_i \cdot g(\mathbf{l_h} - \mathbf{S_i}, \{p_1, p_2, \ldots, p_i\})$.

[1] We say that $\mathbf{l}_i \leq \mathbf{k_0}$ when the relationship holds component wise, when it holds only for some of the components, but not for all, we say that $\mathbf{l}_i \not\leq \mathbf{k_0}$. The sequence $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l_{mx}}$ is ordered in such way that if $\mathbf{l}_i$ and$\mathbf{l}_j$ are vectors of the sequence with $i < j$ then either $\mathbf{l}_i \leq \mathbf{l}_j$ or $\mathbf{l}_i \not\leq \mathbf{l}_j$ but it cannot happen that $\mathbf{l}_i > \mathbf{l}_j$

The time complexity to compute the normalisation constant of a PFS SPN with a load vector $\mathbf{k_0} = [k_1, k_2, \ldots, k_{nq}]$ is $O(np \cdot mx)$, where $mx = \prod_{i=1}^{nq} (k_i + 1)$ and $np = |P|$. In terms of space the complexity is $O(mx)$.

Performance Indices using the Normalisation Constant

The previous algorithm can be used to find the normalisation constant of any S-invariant reachable SPN with PFS. The importance of this method is that the computation of the normalisation constant allows an easy evaluation of the SPN since many performance measures can be expressed in terms of normalisation constants with smaller set of places and load vectors. In this section we describe how to derive relationships for transition throughputs, place utilisations and average number of tokens in places.

Lemma 19.3 [Transition Utilisation] *Given a S-invariant reachable SPN with PFS, the steady state probability that a transition t_j is enabled, given a load vector $\mathbf{k_0}$, is provided by*

$$P([t>; \mathbf{k_0}) = \frac{g(\mathbf{k_0} - \mathbf{S} \cdot \mathbf{i}(t_j), P)}{g(\mathbf{k_0}, P)} \prod_{i=1}^{np} y_i^{iv_i(t_j)}. \tag{19.10}$$

Proof:
The steady state probability that a transition t_j is enabled is given by

$$\begin{aligned} P([t>; \mathbf{k_0}) &= \sum_{\substack{\mathbf{m} \in \mathcal{E}(\mathbf{k_0}, P) \\ \mathbf{m} \geq \mathbf{i}(t_j)}} \pi(\mathbf{m}) \\ &= \frac{1}{g(\mathbf{k_0}, P)} \sum_{\substack{\mathbf{m} \in \mathcal{E}(\mathbf{k_0}, P) \\ \mathbf{m} \geq \mathbf{i}(t_j)}} \prod_{i=1}^{np} y_i^{m_i}. \end{aligned}$$

The reachability set characterisation $R(\mathcal{N}, \mathbf{m_0}) \neq PR^B(\mathcal{N}, \mathbf{m_0})$ gives that

$$\{\mathbf{m} \in \mathcal{E}(\mathbf{k}, \wp) \text{ with } \mathbf{m} \geq \mathbf{i}(t_j)\} = \{\mathbf{m}' + \mathbf{i}(t_j) \text{with } \mathbf{m}' \in \mathcal{E}(\mathbf{k} - \mathbf{S_i} \cdot \mathbf{i}(t_j), \wp)\}$$

and hence

$$\begin{aligned} P([t>; \mathbf{k_0}) &= \frac{1}{g(\mathbf{k_0}, P)} \prod_{i=1}^{np} y_i^{iv_i(t_j)} \sum_{\mathbf{m} \in \mathcal{E}(\mathbf{k_0} - \mathbf{S} \cdot \mathbf{i}(t_j), P)} \prod_{i=1}^{np} y_i^{m_i} \\ &= \frac{1}{g(\mathbf{k_0}, P)} \prod_{i=1}^{np} y_i^{iv_i(t_j)} \cdot g(\mathbf{k_0} - \mathbf{S} \cdot \mathbf{i}(t_j), P) \\ &= \frac{g(\mathbf{k_0} - \mathbf{S} \cdot \mathbf{i}(t_j), P)}{g(\mathbf{k_0}, P)} \prod_{i=1}^{np} y_i^{iv_i(t_j)} \end{aligned}$$

where $iv_i(t_j)$ is the i^{th} element of the input vector for transition t_j. ♢

Since all transitions have independent marking firing rates, the utilisation and the throughput of a transition t_j, given a load vector $\mathbf{k_0}$, are respectively

$$U_j(\mathbf{k_0}) = P([t_j>; \mathbf{k_0}), \tag{19.11}$$

and

$$X_j(\mathbf{k_0}) = U_j(\mathbf{k_0}) \cdot \mu_j. \tag{19.12}$$

The proofs of the next two lemmas use the same techniques of Lemma 19.3, the details can be found in [39].

Lemma 19.4 [Place Utilisation] *Given a S-invariant reachable SPN with PFS, the steady state probability that a place p_i is not empty, given a load vector $\mathbf{k_0}$, is provided by*

$$P(m_i > 0; \mathbf{k_0}) = 1 - \frac{g(\mathbf{k_0}, P - \{p_i\})}{g(\mathbf{k_0}, P)}. \tag{19.13}$$

Lemma 19.5 [Place Marking] *Given a S-invariant reachable SPN with PFS, the steady state probability that a in place p_i there are l tokens, given a load vector $\mathbf{k_0}$, is provided by*

$$P(m_i = l; \mathbf{k_0}) = \frac{g(\mathbf{k_0} - l \cdot \mathbf{S_i}, P - \{p_i\})}{g(\mathbf{k_0}, P)} \cdot y_i^l. \tag{19.14}$$

Let us note that $g(\mathbf{k_0} - l \cdot \mathbf{S_i}, P - \{p_i\})$ is the normalisation constant of the net with the first $np - 1$ places and a load vector $\mathbf{k_0} - l \cdot \mathbf{S_i}$.

Theorem 19.2 gives us an efficient way to compute the distribution of the number of tokens in a place only for the place p_{np} (the last one of the SPN). Moreover if we have to compute the distribution of the number of tokens in a place $p_i \neq p_{np}$, we can change the order in which places are considered devising the computation of G putting the place p_i in the last position (place re-indexing). However, this problem can be solved in a computationally more efficient way by using the following inverse convolution algorithm. Let us denote the normalisation constant $g(\mathbf{k} - l \cdot \mathbf{S_i}, \wp - \{p_i\})$ by $g^{[p_i]}(\mathbf{k} - l \cdot \mathbf{S_i}, \wp)$, where $\wp \subseteq P$ and $\mathbf{k} \leq \mathbf{k_0}$.

Lemma 19.6 [Inverse Convolution] *The following relationship exists between the normalisation constant of a PFS S-invariant reachable SPN with a set of places P and a load vector $\mathbf{k_0}$, and the normalisation constants of SPN with smaller set of places and smaller load vectors. Let be $\mathbf{k}$ and $\wp$ respectively a load vector and a subset of places, with $\mathbf{k} \leq \mathbf{k_0}$ and $\wp \subseteq P$, we have that*

$$g^{[p_i]}(\mathbf{k}, \wp) = g(\mathbf{k}, \wp) - y_i \cdot g(\mathbf{k} - \mathbf{S_i}, \wp). \tag{19.15}$$

Proof:
The proof of this lemma follows from Theorem 19.2. The inverse convolution values can be iteratively computed starting with the initial condition

$$g^{[p_i]}(\mathbf{0}, \wp) = g(\mathbf{0}, \wp) = 1.$$

$\diamondsuit$

It is important to note that the algorithm for the inverse convolution can be numerically unstable. The problem follows from the cancellation error that may derive from the subtraction in (19.15) and suggests that the use of this expression must be carefully controlled in order to avoid unsafe situations.

Using Lemma 19.6 we can compute the average number of tokens in a place p_i:

$$\begin{aligned} n_i(\mathbf{k_0}) &= \sum_{l=0}^{mx_i(\mathbf{k_0},P)} l \cdot P(m_i = i; \mathbf{k_0}) \\ &= \frac{1}{g(\mathbf{k_0}, P)} \sum_{l=0}^{mx_i(\mathbf{k_0},P)} l \cdot g^{[p_i]}(\mathbf{k_0} - l \cdot \mathbf{S_i}, P) \cdot y_i^l, \end{aligned} \tag{19.16}$$

where $mx_i(\mathbf{k_0}, P)$ is the structural marking bound of the place p_i with a load vector $\mathbf{k_0}$.

Numerical Results

The following example shows how to use all previous concepts to find the equilibrium distribution and some performance indices for a PF-SPN. Let us consider the net of Figure 18.2.

The SPN has three minimal P-semiflows: $[1,1,1,0,1,0,1]$, $[0,0,0,1,1,0,1]$ and $[0,0,0,0,0,1,1]$.

Let us show now how it is possible to compute some performance indices for this PF-SPN. Assume that the following firing rates are associated with the transitions of the net $\mathcal{T} = \{1,2,3,5,6,7,8\}$. Let $\mathbf{m_0} = [3,0,0,2,0,1,0]$ be the initial marking of the SPN; the load vector in this case is $\mathbf{k_0} = [3,2,1]$. Table 19.1 corresponds to the "tableau" filled by the algorithm for computing the normalisation constant.

The normalisation constant in this particular case is $G = 14.381$. The sparseness of the "tableau" and the presence of many entries with identical values illustrate the behaviour of the algorithm and the peculiarities that distinguish SPNs of the type considered in this chapter, from multiclass product form queueing networks. In particular place p_1, p_2 and p_3 are covered by only one P-semiflows and the entries of the corresponding columns are obtained by summing the term corresponding to the same load vector in the previous column with the term corresponding to a load vector with one token less in the first P-semiflow. The large number of zero entries in the corresponding columns derives from the fact that many load vectors are "unfeasible" when a net comprising only these first three places is considered. Place p_5 is covered by two P-semiflows; some of the entries of column "p_5" are identical to those of column "p_4": this happens when the load vector is such that removing one token both from the first and the second P-semiflow yield a load vector that is unfeasible. Place p_7 is covered by all the three P-semiflows and the computation of each entry of the corresponding column depends on values that are quite distant in the tableau.

Load Vector	p_1	p_2	p_3	p_4	p_5	p_6	p_7
[0, 0, 0]	1.000	1.000	1.000	1.000	1.000	1.000	1.000
[0, 0, 1]	0.000	0.000	0.000	0.000	0.000	1.143	1.143
[0, 1, 0]	0.000	0.000	0.000	1.200	1.200	1.200	1.200
[0, 1, 1]	0.000	0.000	0.000	0.000	0.000	1.371	1.371
[0, 2, 0]	0.000	0.000	0.000	1.440	1.440	1.440	1.440
[0, 2, 1]	0.000	0.000	0.000	0.000	0.000	1.646	1.646
[1, 0, 0]	1.000	1.500	1.833	1.833	1.833	1.833	1.833
[1, 0, 1]	0.000	0.000	0.000	0.000	0.000	2.095	2.095
[1, 1, 0]	0.000	0.000	0.000	2.200	3.200	3.200	3.200
[1, 1, 1]	0.000	0.000	0.000	0.000	0.000	3.657	4.657
[1, 2, 0]	0.000	0.000	0.000	2.640	3.840	3.840	3.840
[1, 2, 1]	0.000	0.000	0.000	0.000	0.000	4.389	5.589
[2, 0, 0]	1.000	1.750	2.361	2.361	2.361	2.361	2.361
[2, 0, 1]	0.000	0.000	0.000	0.000	0.000	2.698	2.698
[2, 1, 0]	0.000	0.000	0.000	2.833	4.667	4.667	4.667
[2, 1, 1]	0.000	0.000	0.000	0.000	0.000	5.333	7.167
[2, 2, 0]	0.000	0.000	0.000	3.400	6.600	6.600	6.600
[2, 2, 1]	0.000	0.000	0.000	0.000	0.000	7.543	10.743
[3, 0, 0]	1.000	1.875	2.662	2.662	2.662	2.662	2.662
[3, 0, 1]	0.000	0.000	0.000	0.000	0.000	3.042	3.042
[3, 1, 0]	0.000	0.000	0.000	3.194	5.556	5.556	5.556
[3, 1, 1]	0.000	0.000	0.000	0.000	0.000	6.349	8.710
[3, 2, 0]	0.000	0.000	0.000	3.833	8.500	8.500	8.500
[3, 2, 1]	0.000	0.000	0.000	0.000	0.000	9.714	14.381

Table 19.1: "Tableau" for the SPN of Fig. 3 up to the load vector $\mathbf{k_0} = [3, 2, 1]$.

After computing the normalisation constant tableau, the throughput of all the transitions in the net and the average number of tokens in each place have been computed using Equations (19.10), (19.12), (19.14), and (19.16).

Other Normalisation Constant Algorithms

In [11] an algorithm for the computation of the normalisation constant that considers marking dependent firing rates has been proposed. In that paper the form of $\pi_f(\mathbf{m})$ is the one that follows from Equation (18.12), while the function $\Phi(\mathbf{m})$ (marking dependent service rate function) must be expressed as a product over the places of the SPN,

$$\Phi(\mathbf{m}) = \prod_{i=1}^{np} \Phi_i(m_i). \tag{19.17}$$

In this case the equations of Theorem 19.1 must account of this form of the function $\Phi(\mathbf{m})$. In particular Equation (19.5) of assumes the following form

$$g(\mathbf{k}, \wp) = \sum_{j \in M_i(\mathbf{k}, \wp)} y_i^j \Phi_i(m_i) g(\mathbf{k} - j \cdot \mathbf{S_i}, \wp - \{p\}) \qquad p_i \in \wp. \tag{19.18}$$

The algorithm for the computation of the normalisation constant with this type of rates has the same form of the analogous algorithm for PF-QNs with marking dependent service rates.

In [9] Coleman proposes an algorithm to compute the normalisation constant for PF-SPN with a time complexity that depends only on the structure of the net. Although this algorithm has this nice time complexity feature, there

are several problems that follow from the cancellation errors. These problem arise because the presence of several subtractions in the basic equations of this algorithm.

19.2 The Mean Value Analysis Algorithm

The steady state probability that in place p_i (with $1 \leq i \leq np$) there are at least h tokens is defined as follows:

$$P(m_i \geq h; \mathbf{k_0}) = \sum_{\substack{\mathbf{m} \in \mathcal{E}(\mathbf{k}_0,\, P) \\ m_i \geq h}} \pi(\mathbf{m}). \tag{19.19}$$

Similarly, the steady state probability that the marking is greater or equal than the row vector $\mathbf{iv} = [iv_1, iv_2, \ldots, iv_{np}]$ is denoted as:

$$\begin{aligned} P(\mathbf{m} \geq \mathbf{iv}; \mathbf{k_0}) &= P(m_1 \geq iv_1, m_2 \geq iv_2, \ldots, m_{np} \geq iv_{np}; \mathbf{k_0}) \\ &= \sum_{\substack{\mathbf{m} \in \mathcal{E}(\mathbf{k}_0,\, P) \\ \mathbf{m} \geq \mathbf{iv}}} \pi(\mathbf{m}). \end{aligned}$$

Using this last definition, the probability that transition t_j is enabled can also be written as

$$P([t_j>; \mathbf{k_0}) = P(\mathbf{m} \geq \mathbf{i}(t_j); \mathbf{k_0}),$$

where $\mathbf{i}(t_j)$ is the input bag of transition t_j. With this result we can observe that the utilisation $U_j(\mathbf{k_0})$ can be defined in the following way

$$U_j(\mathbf{k_0}) = P(\mathbf{m} \geq \mathbf{i}(t_j); \mathbf{k_0}). \tag{19.20}$$

The throughput $X_j(\mathbf{k_0})$ of a transition t_j is the average number of firings of t_j per unit of time. Since we have assumed that all transitions have marking independent firing rates, $X_j(\mathbf{k_0})$ can be easily derived from the knowledge of the utilisation of the same transition

$$X_j(\mathbf{k_0}) = U_j(\mathbf{k_0}) \cdot \mu_j. \tag{19.21}$$

In an SPN the *utilisation* of a place p_i, with a load vector $\mathbf{k_0}$, is given by the steady state probability that place p_i is non-empty:

$$u_i(\mathbf{k_0}) = P(m_i > 0; \mathbf{k_0}).$$

Let us define the *throughput of a place* p_i as the average number of tokens that depart from p_j per unit of time. This quantity can be obtained as a weighted sum of the throughputs of all transitions that belong to the output set of p_i

$$x_i(\mathbf{k_0}) = \sum_{t_j \in p_i^\bullet} X_j(\mathbf{k_0}) \cdot i_i(t_j), \tag{19.22}$$

where $i_i(t_j)$ is the i-th component of the input bag of t_j and corresponds to the weight of the arc from place p_i to transition t_j.
The probability distribution provided by Equation (19.19) allows to compute the *average number of tokens* in place p_i, given a load vector $\mathbf{k_0}$, that we denote by $n_i(\mathbf{k_0})$. Letting $\omega_i(\mathbf{k_0})$ represent the *average sojourn time* of a token in place p_i, this quantity can be computed using Little's formula [30] that in this case assumes the following expression:

$$\omega_i(\mathbf{k_0}) = \frac{n_i(\mathbf{k_0})}{x_i(\mathbf{k_0})}, \qquad \forall p_i \in P.$$

Recursive expressions for probability distributions

Let us prove now a few lemmas that provide some recursive relations for S-invariant reachable PF-SPNs. The method used for deriving these relations is based on the normalisation constant calculus presented before (Theorem 19.1). The proof of the next lemma uses the same techniques of Lemma 19.3, the details can be found in [40].

Lemma 19.7 *Given an S-invariant reachable PF-SPN and a load vector* $\mathbf{k_0}$*, the steady state probability that in place* p_i *there are* at least h *tokens, is expressed with the following formula*

$$P(m_i \geq h; \mathbf{k_0}) = \frac{g(\mathbf{k_0} - h \cdot \mathbf{S_i}, P)}{g(\mathbf{k_0}, P)} \cdot y_i^h, \tag{19.23}$$

where $\mathbf{S_i}$ *is the column corresponding to place* p_i *within the P-semiflow matrix* $\mathbf{S}$*.*

The previous result can be used to derive recursive expressions for different distributions of tokens in particular places of the net; these expressions have the nice property that the dependencies on the normalisation constant disappear, while they are defined in terms of expressions of other token distributions computed for smaller load vectors. In particular, an alternative expression for the result of Equation (19.23) can be easily obtained.

Corollary 19.8 *Given an S-invariant reachable PF-SPN and a load vector* $\mathbf{k_0}$*, the steady state probability that in place* p_i *there are* at least h *tokens* ($h > 0$) *is provided by the following relation*

$$P(m_i \geq h; \mathbf{k_0}) \quad = \quad u_i(\mathbf{k_0}) \cdot P(m_i \geq h - 1; \mathbf{k_0} - \mathbf{S_i}).$$

Proof:
From Lemma 19.7 we have that

$$P(m_i \geq h; \mathbf{k_0}) \quad = \quad \frac{g(\mathbf{k_0} - h \cdot \mathbf{S_i}, P)}{g(\mathbf{k_0}, P)} \cdot y_i^h$$

$$
\begin{aligned}
&= \frac{g(\mathbf{k_0} - \mathbf{S_i} - (h-1)\cdot\mathbf{S_i}, P)}{g(\mathbf{k_0}, P)} \cdot y_i^{h-1} \cdot y_i \\
&= \frac{g(\mathbf{k_0} - \mathbf{S_i} - (h-1)\cdot\mathbf{S_i}, P)}{g(\mathbf{k_0} - \mathbf{S_i}, P)} \cdot y_i^{h-1} \cdot \frac{g(\mathbf{k_0} - \mathbf{S_i}, P)}{g(\mathbf{k_0}, P)} \cdot y_i \\
&= P(m_i \geq h-1; \mathbf{k_0} - \mathbf{S_i}) \cdot P(m_i > 0; \mathbf{k_0}).
\end{aligned}
$$

♢

Corollary 19.8 can be generalised to obtain the steady state probabilities on any subset of places and any combination of restrictions; in particular the steady state probability that in place p_i there are *at least* h tokens and in place p_j there are *at least* l tokens ($h > 0$ and $l > 0$) is provided by the following relations:

$$
\begin{aligned}
P(m_i \geq h, m_j \geq l; \mathbf{k_0}) &= u_i(\mathbf{k_0}) P(m_i \geq h-1, m_j \geq l; \mathbf{k_0} - \mathbf{S_i}) \\
&= u_j(\mathbf{k_0}) P(m_i \geq h, m_j \geq l-1; \mathbf{k_0} - \mathbf{S_j}).
\end{aligned}
$$

Using the same ideas we can derive the following corollary.

Corollary 19.9 *Given an S-invariant reachable PF-SPN and a load vector* $\mathbf{k_0}$*, for the steady state probability that the marking is greater or equal than vector* $\mathbf{iv} = [iv_1, iv_2, \ldots, iv_{np}]$ *the following relation holds:*

$$
P(\mathbf{m} \geq \mathbf{iv}; \mathbf{k_0}) = u_i(\mathbf{k_0}) \cdot P(\mathbf{m} \geq \mathbf{iv} - \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}), \tag{19.24}
$$

for any $p_i \in P$ *such that* $iv_i > 0$.

When the vector $\mathbf{iv}$ is the input bag of transition t_j the previous result can be specialised as follows:

$$
\begin{aligned}
P([t_j>; \mathbf{k_0}) &= P(\mathbf{m} \geq \mathbf{i}(t_j); \mathbf{k_0}) \\
&= u_i(\mathbf{k_0}) \cdot P(\mathbf{m} \geq \mathbf{i}(t_j) - \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}) \qquad (19.25)
\end{aligned}
$$

where p_i is an input place of t_j.

Recursive expressions for average performance indices

The expressions obtained in the previous section are the basis for the derivation of recursive formulas for the average performance indices.

Theorem 19.10 *Given an S-invariant reachable PF-SPN and a load vector* $\mathbf{k_0}$*, the following recursive relations hold:*

$$
n_i(\mathbf{k_0}) = u_i(\mathbf{k_0}) \cdot (1 + n_i(\mathbf{k_0} - \mathbf{S_i}))\,; \tag{19.26}
$$

$$
x_i(\boldsymbol{k_0}) = u_i(\boldsymbol{k_0}) \cdot \left(\sum_{t_j \in p_i^\bullet} P\left(\boldsymbol{m} \geq \mathbf{i}(t_j) - \boldsymbol{e_i}; \boldsymbol{k_0} - \boldsymbol{S_i}\right) i_i(t_j) \mu_j \right); \tag{19.27}
$$

$$
\omega_i(\boldsymbol{k_0}) = \frac{1 + n_i(\boldsymbol{k_0} - \boldsymbol{S_i})}{\sum_{t_j \in p_i^\bullet} P\left(\boldsymbol{m} \geq \mathbf{i}(t_j) - \boldsymbol{e_i}; \boldsymbol{k_0} - \boldsymbol{S_i}\right) i_i(t_j) \mu_j}. \tag{19.28}
$$

Proof:
Recalling that $mx_i(\mathbf{k_0})$ is the structural marking bound of place p_i with load vector $\mathbf{k_0}$ (Definition 18.16 of Section 18.6.2 of Chapter 18), we have that

$$\begin{aligned}
n_i(\mathbf{k_0}) &= \sum_{j=1}^{mx_i(\mathbf{k}_0)} P(m_i \geq j; \mathbf{k_0}) \\
&= \sum_{j=1}^{mx_i(\mathbf{k}_0)} \frac{g(\mathbf{k_0} - j \cdot \mathbf{S_i}, P)}{g(\mathbf{k_0}, P)} y_i^j \\
&= \frac{g(\mathbf{k_0} - \mathbf{S_i}, P)}{g(\mathbf{k_0}, P)} y_i + \sum_{j=2}^{mx_i(\mathbf{k}_0)} \frac{g(\mathbf{k_0} - j \cdot \mathbf{S_i}, P)}{g(\mathbf{k_0}, P)} y_i^j \\
&= u_i(\mathbf{k_0}) + \sum_{j=2}^{mx_i(\mathbf{k}_0)} \frac{g(\mathbf{k_0} - i \cdot \mathbf{S_i}, P)}{g(\mathbf{k_0}, P)} y_i^j \\
&= u_i(\mathbf{k_0}) + \sum_{j=2}^{mx_i(\mathbf{k}_0)} y_i^{j-1} y_i \frac{g(\mathbf{k_0} - \mathbf{S_i}, P)}{g(\mathbf{k_0}, P)} \frac{g(\mathbf{k_0} - \mathbf{S_i} - (j-1) \cdot \mathbf{S_i}, P)}{g(\mathbf{k_0} - \mathbf{S_i}, P)}
\end{aligned}$$

and substituting $l = j - 1$ we obtain that

$$\begin{aligned}
n_i(\mathbf{k_0}) &= u_i(\mathbf{k_0}) + u_i(\mathbf{k_0}) \cdot \sum_{l=1}^{mx_i(\mathbf{k}_0)-1} \frac{g(\mathbf{k_0} - \mathbf{S_i} - l \cdot \mathbf{S_i}, P)}{g(\mathbf{k_0} - \mathbf{S_i}, P)} y_i^l \\
&= u_i(\mathbf{k_0}) \cdot (1 + n_i(\mathbf{k_0} - \mathbf{S_i})) .
\end{aligned}$$

The proof of Equation (19.27) follows from Equation (19.22) and Equation (19.25), while the proof of Equation (19.28) follows from Little's formula, Equation (19.26), and Equation (19.27). ♢

From Equation (19.28) we can observe that in an S-invariant reachable PF-SPN with a load vector $\mathbf{k_0}$, it is possible to compute the average sojourn time of a token in place p_i, using only quantities referring to the solution of the SPN with smaller load vectors. This equation was also derived in [1] by means of a series of *Arrival Theorems* for PF-SPNs.

Visit Ratios

In Section 18.4 of Chapter 18 we have seen that the structural constraints for the existence of a PFS summarised by Definition 18.2 imply that the system of linear equations represented by (18.8) admits a solution that is an invariant measure for the SPN. It is interesting to observe that the system of equations (18.8), that we repeat here setting $f(\mathbf{i}(t_j))\mu_j = v_j$ to make the explanation simpler, is homogeneous and hence admits an infinite number of solutions that differ for a multiplicative constant

$$v_j = \sum_{t_m \in T} v_m P(\mathbf{i}(t_m), \mathbf{i}(t_j)) \qquad \forall\, t_j \in T \tag{19.29}$$

For each FR^*-class $\mathcal{C}_c \in \mathcal{X}_{FR^*}$, we arbitrarily identify a reference transition $t_l^{[c]}$ (c is the class index in $\mathcal{X}_{FR^*}$ and l is the transition index in T) and we compute the solution of the system of equations (19.29), assuming $v_l^{[c]} = 1$ (with $1 \leq c \leq nc$). By doing so, we identify a particular vector $\mathbf{v}$ that is called *visit ratio vector* of the SPN.
Lemma 19.12 shows that the throughput of the transitions belonging to a same FR^*-class are related and that this relation is given by the visit ratio vector. To prove this lemma we require an intermediate result that is proved in Lemma 19.11.

Lemma 19.11 *Given an S-invariant reachable Petri net satisfying Definition 18.2 (Π-net). Let* $\mathbf{S}$ *be the P-semiflow matrix, then for any pairs of transitions* t_l, t_m *belonging to the same* FR^**-class* $\mathcal{C}_c$ *we have that*

$$\mathbf{S} \cdot \mathbf{i}(t_l) = \mathbf{S} \cdot \mathbf{i}(t_m).$$

Proof:
As first step of the proof, we have to show that $\forall t_j \in T$

$$\mathbf{S} \cdot \mathbf{i}(t_j) = \mathbf{S} \cdot \mathbf{o}(t_j).$$

The previous equality can be established by noticing that the firing of a transition t_j brings the net from marking $\mathbf{m} = \mathbf{n} + \mathbf{i}(t_j)$ to marking $\mathbf{m}' = \mathbf{n} + \mathbf{o}(t_j)$, with $\mathbf{m}, \mathbf{m}' \in \mathcal{E}(\mathbf{k_0}, P)$. This means that $\mathbf{S} \cdot \mathbf{m} = \mathbf{S} \cdot \mathbf{m}'$ and equivalently that $\mathbf{S} \cdot (\mathbf{n} + \mathbf{i}(t_j)) = \mathbf{S} \cdot (\mathbf{n} + \mathbf{o}(t_j))$, from which we can derive that $\mathbf{S} \cdot \mathbf{i}(t_j) = \mathbf{S} \cdot \mathbf{o}(t_j)$. By Definition 18.2 and Definition 18.3 (Section 18.4 of Chapter 18), we have that there exist a transition $t_n \in \mathcal{C}_c$ such that $\mathbf{o}(t_l) = \mathbf{i}(t_n)$. If $t_n = t_m$, we have that $\mathbf{o}(t_l) = \mathbf{i}(t_m)$. From this we can derive that $\mathbf{S} \cdot \mathbf{i}(t_l) = \mathbf{S} \cdot \mathbf{o}(t_l) = \mathbf{S} \cdot \mathbf{i}(t_m)$. Suppose that $t_n \neq t_m$, in this case we have that $\mathbf{S} \cdot \mathbf{i}(t_l) = \mathbf{S} \cdot \mathbf{i}(t_n)$. By Definition 18.2 there must be another transition t_h such that $\mathbf{o}(t_n) = \mathbf{i}(t_h)$, if $t_h = t_m$ the lemma is proved, otherwise we can repeatedly apply the same reasoning until we find a transition t_s such that $\mathbf{o}(t_s) = \mathbf{i}(t_m)$. ♢

Lemma 19.12 *Given a PF-SPN with a load vector* $\mathbf{k_0}$ *where* $\mathcal{X}_{FR^*} = \{\mathcal{C}_1, \ldots, \mathcal{C}_{nc}\}$ *is the set of* FR^**-classes and* $\mathbf{v}$ *is the visit ratio vector, then for any* $t_m, t_l \in \mathcal{C}_c$ *we have that*

$$\frac{X_m(\mathbf{k_0})}{X_l(\mathbf{k_0})} = \frac{v_m}{v_l}, \tag{19.30}$$

where v_l *and* v_m *are the components of vector* $\mathbf{v}$ *corresponding respectively to transition* t_l *and* t_m*.*

Proof:
When $\mathbf{o}(t_l) = \mathbf{i}(t_m)$ we said that t_m and t_l are adjacent in the SPN and we can

observe that

$$
\begin{aligned}
X_m(\mathbf{k_0}) &= \sum_{\substack{\mathbf{n} \\ \mathbf{n}+\mathbf{i}(t_m) \in \mathcal{E}(\mathbf{k}_0, P)}} \pi(\mathbf{n}+\mathbf{i}(t_m))\ \mu_m \\
&= \sum_{\substack{\mathbf{n} \\ \mathbf{n}+\mathbf{i}(t_m) \in \mathcal{E}(\mathbf{k}_0, P)}} \pi(\mathbf{n}+\mathbf{i}(t_l))\ \frac{\pi(\mathbf{n}+\mathbf{i}(t_m))}{\pi(\mathbf{n}+\mathbf{i}(t_l))}\ \mu_m \\
&\overset{\text{Teo. 18.5, Eq. (18.9)}}{=} \sum_{\substack{\mathbf{n} \\ \mathbf{n}+\mathbf{i}(t_m) \in \mathcal{E}(\mathbf{k}_0, P)}} \frac{f(\mathbf{i}(t_m))}{f(\mathbf{i}(t_l))}\ \pi(\mathbf{n}+\mathbf{i}(t_l))\ \mu_m \\
&= \sum_{\substack{\mathbf{n} \\ \mathbf{n}+\mathbf{i}(t_m) \in \mathcal{E}(\mathbf{k}_0, P)}} \frac{f(\mathbf{i}(t_m))}{f(\mathbf{i}(t_l))}\ \frac{\mu_l}{\mu_l}\ \pi(\mathbf{n}+\mathbf{i}(t_l))\ \mu_m \\
&= \frac{f(\mathbf{i}(t_m))}{f(\mathbf{i}(t_l))}\ \frac{\mu_m}{\mu_l} \sum_{\substack{\mathbf{n} \\ \mathbf{n}+\mathbf{i}(t_m) \in \mathcal{E}(\mathbf{k}_0, P)}} \pi(\mathbf{n}+\mathbf{i}(t_l))\ \mu_l \\
&\overset{\text{Lemma 19.11}}{=} \frac{f(\mathbf{i}(t_m))}{f(\mathbf{i}(t_l))}\ \frac{\mu_m}{\mu_l} \sum_{\substack{\mathbf{n} \\ \mathbf{n}+\mathbf{i}(t_l) \in \mathcal{E}(\mathbf{k}_0, P)}} \pi(\mathbf{n}+\mathbf{i}(t_l))\ \mu_l \\
&= \frac{f(\mathbf{i}(t_m))}{f(\mathbf{i}(t_l))}\ \frac{\mu_m}{\mu_l}\ X_l(\mathbf{k_0}) \\
&= \frac{v_m}{v_l}\ X_l(\mathbf{k_0}).
\end{aligned}
$$

Suppose now that $\mathbf{o}(t_l) \neq \mathbf{i}(t_m)$. By Definition 18.2 and by the fact that $t_l, t_m \in \mathcal{C}_c$, there must be a sequence of transitions $\sigma = \{t_{h_1}, t_{h_2}, \ldots, t_{h_n}\}$ such that $\mathbf{o}(t_l) = \mathbf{i}(t_{h_1})$, $\mathbf{o}(t_{h_j}) = \mathbf{i}(t_{h_{j+1}})$, for $j = 1, 2, \ldots, n-1$, and $\mathbf{o}(t_{h_n}) = \mathbf{i}(t_m)$. In this case, the result that we have just derived for adjacent transitions can be repeatedly used for each pair of transitions belonging to the sequence σ and we can thus write

$$
\begin{aligned}
X_{h_1}(\mathbf{k_0}) &= \frac{v_{h_1}}{v_l} X_l(\mathbf{k_0}) \\
X_{h_{j+1}}(\mathbf{k_0}) &= \frac{v_{h_{j+1}}}{v_{h_j}} X_{h_j}(\mathbf{k_0}) \qquad j = 1, 2, \ldots, n-1 \\
X_m(\mathbf{k_0}) &= \frac{v_m}{v_{h_n}} X_{h_n}(\mathbf{k_0}).
\end{aligned}
$$

By direct substitution we obtain that

$$
X_m(\mathbf{k_0}) = \frac{v_m}{v_{h_n}} \frac{v_{h_n}}{v_{h_{n-1}}} \cdots \frac{v_{h_2}}{v_{h_1}} \frac{v_{h_1}}{v_l} X_l(\mathbf{k_0})
$$

$$= \frac{v_m}{v_l} X_l(\mathbf{k_0}).$$

$\diamondsuit$

The result of Lemma 19.12 provides also, as a particular case:

$$X_j(\mathbf{k_0}) = X_l(\mathbf{k_0}) \cdot v_j,$$

where $t_j, t_l \in \mathcal{C}_c$, and $v_l = 1$.
It is worthwhile to point out that Lemma 19.12 asserts that the throughput ratio of any pair of transitions belonging to the same FR^*-class is independent of the load of the SPN. For transitions in different FR^*-classes, Equation (19.30) does not hold. This can be observed in the SPN of Figure 19.1. In this SPN there are two FR^*-classes: $\mathcal{C}_1 = \{t_1, t_3\}$ and $\mathcal{C}_2 = \{t_2, t_4\}$. We can easily see that the ratio $X_1(\mathbf{k_0})/X_2(\mathbf{k_0})$ depends on the initial marking (or equivalently on the load vector).

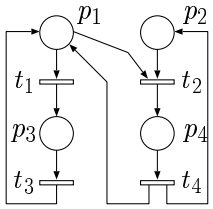


Figure 19.1: PF-SPN with two FR^*-classes.

In fact, let us denote the initial marking by $\mathbf{m_0} = [m_1, m_2, 0, 0]$, we can distinguish the following cases. With $m_2 = 0$ and $m_1 > 0$, the throughput of t_2 is null while the throughput of t_1 is different from zero. If instead we consider any $\mathbf{m_0}$ such that $m_2 > m_1$, the ratio $X_1(\mathbf{k_0})/X_2(\mathbf{k_0})$ is independent of the load because the place p_2 does not affect the behaviour of the net (since place p_2 is always marked independently of the evolution of the net). This shows that in general the ratio between the throughputs of two transitions belonging to different FR^*-classes changes its value depending on the initial marking of the net.
To deal with the relation among the throughputs of transitions belonging to different FR^*-classes, let us introduce another visit ratio concept that is the dual of the previous one since it refers to the throughput of places belonging to the support of the same P-semiflow. Let $p_i, p_r \in P$ be two places, the quantity

$$\nu_i(\mathbf{k_0}) = \frac{x_i(\mathbf{k_0})}{x_r(\mathbf{k_0})} \tag{19.31}$$

represents the visit ratio of tokens to place p_i with respect to the arrivals of tokens to place p_r, given a load vector $\mathbf{k_0}$. This kind of visit ratios plays an important role in the MVA algorithm that we are going to derive in the following sections; to distinguish these ratios from those defined by vector $\mathbf{v}$, we call them *P-visit ratios*.

Lemma 19.13 *Given an S-invariant reachable PF-SPN and a load vector* $\mathbf{k_0}$*, if the place* $p_r \in P$ *is covered by the P-semiflow* $\mathbf{s_h}$ *with support* $||\mathbf{s_h}||$ *then*

$$x_r(\mathbf{k_0}) = \frac{k_h}{\sum_{p_i \in ||\mathbf{s}_h||} s_i \cdot \nu_i(\mathbf{k_0}) \cdot \omega_i(\mathbf{k_0})}, \tag{19.32}$$

where k_h *is the component of the load vector* $\mathbf{k_0}$ *corresponding to the P-semiflow* $\mathbf{s_h}$*, and* s_i *is the component of the P-semiflow* $\mathbf{s_h}$ *corresponding to place* p_i*.*

Proof:
By definition of P-semiflow and taking expectations, we have that

$$k_h = \sum_{p_i \in ||\mathbf{s}_h||} s_i \cdot n_i(\mathbf{k_0}).$$

By Little's law, it follows that

$$\begin{aligned} k_h &= \sum_{p_i \in ||\mathbf{s}_h||} s_i \cdot \omega_i(\mathbf{k_0}) \cdot x_i(\mathbf{k_0}) \\ &= x_r(\mathbf{k_0}) \cdot \sum_{p_i \in ||\mathbf{s}_h||} s_i \cdot \omega_i(\mathbf{k_0}) \cdot \frac{x_i(\mathbf{k_0})}{x_r(\mathbf{k_0})}, \\ &= x_r(\mathbf{k_0}) \cdot \sum_{p_i \in ||\mathbf{s}_h||} s_i \cdot \omega_i(\mathbf{k_0}) \nu_i(\mathbf{k_0}), \end{aligned}$$

and hence

$$x_r(\mathbf{k_0}) = \frac{k_h}{\sum_{p_i \in ||\mathbf{s}_h||} s_i \cdot \nu_i(\mathbf{k_0}) \cdot \omega_i(\mathbf{k_0})}.$$

♢

From Equation (19.31) it follows that knowing the P-visit ratios for all the places of the net and knowing the throughput of the reference place of a P-semiflow, it is possible to compute the throughput of all the other places covered by the same P-semiflow in a straightforward manner.

The aim of the following lemma is to prove that, given an S-invariant reachable PF-SPN with a load vector $\mathbf{k_0}$, it is possible to compute the P-visit ratios using only quantities computed for the same SPN with smaller load vectors. By definition of throughput of a place and by Equation (19.25), Equation (19.31) becomes

$$\nu_i(\mathbf{k_0}) = \frac{u_i(\mathbf{k_0})}{u_r(\mathbf{k_0})} \frac{\sum_{t_a \in p_i^\bullet} P(\mathbf{m} \geq \mathbf{i}(t_a) - \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}) \cdot \mu_a \cdot i_i(t_a)}{\sum_{t_b \in p_r^\bullet} P(\mathbf{m} \geq \mathbf{i}(t_b) - \mathbf{e_r}; \mathbf{k_0} - \mathbf{S_r}) \cdot \mu_b \cdot i_r(t_b)}, \tag{19.33}$$

where $i_i(t_a)$ is the i-th component of the input bag of t_a and corresponds to the weight of the arc from place p_i to transition t_a, while $i_r(t_b)$ corresponds to the weight of the arc from place p_r to transition t_b. To show that the P-visit ratios, with a load vector $\mathbf{k_0}$, can be computed using only quantities computed for the same SPN with smaller load vectors, it is sufficient to prove that the same is true for the ratio $u_i(\mathbf{k_0})/u_r(\mathbf{k_0})$.

Lemma 19.14 *Given an S-invariant reachable PF-SPN with a load vector* $\mathbf{k_0}$*, let* $p_i, p_l \in P$ *be two arbitrary places of the net, the ratio*

$$\frac{u_l(\mathbf{k_0})}{u_i(\mathbf{k_0})}$$

can be computed using only quantities computed referring to the solution of the same SPN with smaller load vectors.

Proof:
First consider the case of adjacent places, i.e., there exists a transition t_j such that $t_j \in p_i^\bullet$ and $t_j \in {}^\bullet p_l$. From Equations (19.25) and (19.21), we have that

$$X_j(\mathbf{k_0}) = u_i(\mathbf{k_0}) \cdot P(\mathbf{m} \geq \mathbf{i}(t_j) - \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}) \cdot \mu_j$$

from this, it follows that

$$u_i(\mathbf{k_0}) = \frac{X_j(\mathbf{k_0})}{P(\mathbf{m} \geq \mathbf{i}(t_j) - \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}) \cdot \mu_j}.$$

Let t_m be a transition belonging to $p_l^\bullet$ such that $\mathbf{o}(t_j) = \mathbf{i}(t_m)$. In the same way we obtain that

$$X_m(\mathbf{k_0}) = u_l(\mathbf{k_0}) \cdot P(\mathbf{m} \geq \mathbf{i}(t_m) - \mathbf{e_l}; \mathbf{k_0} - \mathbf{S_l}) \cdot \mu_m$$

and thus

$$u_l(\mathbf{k_0}) = \frac{X_m(\mathbf{k_0})}{P(\mathbf{m} \geq \mathbf{i}(t_m) - \mathbf{e_l}; \mathbf{k_0} - \mathbf{S_l}) \cdot \mu_m}.$$

Since t_j, t_m belong to the same FR^*-class, from Lemma 19.12 we have that

$$\frac{X_m(\mathbf{k_0})}{X_j(\mathbf{k_0})} = \frac{v_m}{v_j}.$$

Combining all the previous derivations we obtain that

$$\frac{u_l(\mathbf{k_0})}{u_i(\mathbf{k_0})} = \frac{v_m \cdot P(\mathbf{m} \geq \mathbf{i}(t_j) - \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}) \cdot \mu_j}{v_j \cdot P(\mathbf{m} \geq \mathbf{i}(t_m) - \mathbf{e_l}; \mathbf{k_0} - \mathbf{S_l}) \cdot \mu_m},$$

which proves the lemma in this first case.

Consider now the case when p_i and p_l are two places that are not adjacent. For this second case the proof of the lemma is given by construction. Using the

fact that the SPN is live and bounded we know that the net is strongly connected and thus for any pair of places there exists a path that connects these places (see [42]). In particular we can say that there exists a sequence of length $n+1$ of adjacent places $p_{h_0}, p_{h_1}, \ldots, p_{h_n}$ such that $p_{h_0} = p_i$, $p_{h_n} = p_l$. Let us prove the lemma by induction on the length of this sequence. If $n = 1$ this means that p_i and p_l are adjacent places and we have already proved that the result holds in this case.
If $n > 1$, let us assume that we have already been able to show that the ratio between the utilisations of p_l and of any other place connected to p_l by a sequence of places of length n ($p_{h_0}, p_{h_1}, \ldots, p_{h_{n-1}}$ such that $p_{h_0} = p_i$) can be computed using only quantities computed referring to the solution of the same SPN with smaller load vectors. In this case we can write

$$\frac{u_{h_n}(\mathbf{k_0})}{u_{h_0}(\mathbf{k_0})} = \frac{u_{h_n}(\mathbf{k_0})}{u_{h_{n-1}}(\mathbf{k_0})} \frac{u_{h_{n-1}}(\mathbf{k_0})}{u_{h_0}(\mathbf{k_0})}.$$

But the first of the fractions on the right hand side involves places p_{h_n}, and $p_{h_{n-1}}$ that are adjacent and we have just proved that it can be computed in the desired manner. The same is true by assumption for the second fraction that involves places that are the extremes of a sequence of length n. By combining these two facts we can conclude that it is possible to compute the ratio between the utilisations of places that are connected by a sequence of length $n+1$ using only quantities computed referring to the solution of the same SPN with smaller load vectors. ♢

Remark. The computation of the P-visit ratios for load $\mathbf{k_0}$ in terms of quantities derived for smaller loads can be performed in a constructive and efficient manner using the following approach. First choose a place (say place p_r) as a reference place. Subsequently, the strong connectivity property of the net ensures that all its other places can be reached by means of a graph traversal algorithm (breadth first or depth first search) that yields a spanning tree having the root in place p_r. Every place of the net is thus visited once during this traversal and its P-visit ratio is computed in terms of the P-visit ratio of its ancestor in the spanning tree. From this observations we can see that while traversing the graph we visit place p_l coming from its parent place p_i (the parent/child relation is derived from the spanning tree). If t_j is the transition connecting p_i to p_l, i.e., $t_j \in p_i^\bullet$ and $t_j \in {}^\bullet p_l$, and t_m is a transition belonging to $p_l^\bullet$ such that $\mathbf{o}(t_j) = \mathbf{i}(t_m)$, we can write

$$\begin{aligned} \frac{u_l(\mathbf{k_0})}{u_r(\mathbf{k_0})} &= \frac{u_i(\mathbf{k_0})}{u_r(\mathbf{k_0})} \frac{u_l(\mathbf{k_0})}{u_i(\mathbf{k_0})} \\ &= \frac{u_i(\mathbf{k_0})}{u_r(\mathbf{k_0})} \frac{v_m \cdot P\left(\mathbf{m} \geq \mathbf{i}(t_j) - \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}\right) \mu_j}{v_j \cdot P\left(\mathbf{m} \geq \mathbf{i(t_m)} - \mathbf{e_l}; \mathbf{k_0} - \mathbf{S_l}\right) \mu_m}. \end{aligned} \tag{19.34}$$

The P-visit ratio $\nu_l(\mathbf{k_0})$ can be easily obtained as

$$\nu_l(\mathbf{k_0}) = \frac{u_l(\mathbf{k_0})}{u_r(\mathbf{k_0})} \frac{\sum_{t_a \in p_l^\bullet} P(\mathbf{m} \geq \mathbf{i}(t_a) - \mathbf{e_l}; \mathbf{k_0} - \mathbf{S_l}) \cdot \mu_a \cdot i_l(t_a)}{\sum_{t_b \in p_r^\bullet} P(\mathbf{m} \geq \mathbf{i}(t_b) - \mathbf{e_r}; \mathbf{k_0} - \mathbf{S_r}) \cdot \mu_b \cdot i_r(t_b)}. \quad (19.35)$$

From the previous considerations we can see that the computation of the P-visit ratios has the same time complexity of a visit-graph algorithm and is thus of the order of the number of places in the SPN.

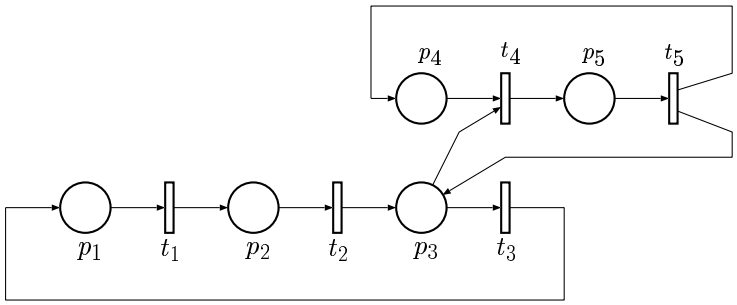


Figure 19.2: PF-SPN with two FR^*-classes used for the computation of the P-visit ratios.

Example of computation of the P-visit ratios. In this example we use the previous lemmas to compute the P-visit ratios for the PF-SPN showed in Figure 19.2 when $\mathbf{k_0}$ is the load vector.
In this SPN there are two FR^*-classes: $\mathcal{C}_1 = \{t_1, t_2, t_3\}; \mathcal{C}_2 = \{t_4, t_5\}$. Let us identify a transition inside each FR^*-class, for instance t_1 for $\mathcal{C}_1$, and t_4 for $\mathcal{C}_2$. Assuming $v_1 = v_4 = 1$ we obtain the visit ratio vector $\mathbf{v} = [1, 1, 1, 1, 1]$.
Let us assume place p_1 as the reference place; the P-visit ratios for all the other places of the net, computed with respect to the reference place p_1, assume the forms summarised in Table 19.2.
It is interesting to point out that the computation of $\nu_4(\mathbf{k_0})$ and $\nu_5(\mathbf{k_0})$ involves terms of the visit ratio vector $\mathbf{v}$ that refer to two different FR^*-classes. In fact, we can see that the terms $u_2(\mathbf{k_0})/u_1(\mathbf{k_0})$ and $u_3(\mathbf{k_0})/u_1(\mathbf{k_0})$ can be computed (step by step) using only terms of the vector $\mathbf{v}$ that refer to the FR^*-class $\mathcal{C}_1$, while the terms $u_4(\mathbf{k_0})/u_1(\mathbf{k_0})$ and $u_5(\mathbf{k_0})/u_1(\mathbf{k_0})$ require elements of the visit ratio vector $\mathbf{v}$ that refer to the FR^*-classes $\mathcal{C}_1$ and $\mathcal{C}_2$ (see the first column of Table 19.2).

Mean Value Analysis for Product Form Solution Stochastic Petri Nets: the algorithm

The relationships derived in this section can now be combined in a complete recursive scheme for the computation of all the performance indices of PF-SPNs

Place	$\frac{u_i(\mathbf{k_0})}{u_1(\mathbf{k_0})}$	$\nu_i(\mathbf{k_0}) = \frac{x_i(\mathbf{k_0})}{x_1(\mathbf{k_0})}$
p_2	$\frac{v_2\,\mu_1}{v_1\,\mu_2}$	$\frac{v_2}{v_1}$
p_3	$\frac{v_3\,\mu_1}{v_1\,\mu_3}$	$\frac{v_3}{v_1}$
p_5	$\frac{v_5\,P(\mathbf{m}\geq\mathbf{i}(t_4)-\mathbf{e_3};\mathbf{k_0}-\mathbf{S_3})\mu_4}{v_4\,\mu_5}\frac{v_3\,\mu_1}{v_1\,\mu_3}$	$\frac{P(\mathbf{m}\geq\mathbf{i}(t_4)-\mathbf{e_3};\mathbf{k_0}-\mathbf{S_3})\,v_3\,\mu_4}{v_1\,\mu_3}\frac{v_5}{v_4}$
p_4	$\frac{v_4\,\mu_5}{v_5\,P(\mathbf{m}\geq\mathbf{i}(t_4)-\mathbf{e_4};\mathbf{k_0}-\mathbf{S_4})\mu_4}\times$ $\times\frac{v_5\,P(\mathbf{m}\geq\mathbf{i}(t_4)-\mathbf{e_3};\mathbf{k_0}-\mathbf{S_3})\mu_4}{v_4\,\mu_5}\frac{v_3\,\mu_1}{v_1\,\mu_3}$	$\frac{P\,(\mathbf{m}\geq\mathbf{i}(t_4)-\mathbf{e_3};\mathbf{k_0}-\mathbf{S_3})\;v_3\,\mu_4}{v_1\,\mu_3}$

Table 19.2: Computation of the P-visit ratios for the SPN of Figure 19.2.

whose general organisation follows that of the MVA for product form Queueing Networks.
Equation (19.28) of Theorem 19.10 provides the way of computing the sojourn times of all the places of the net at load $\mathbf{k_0}$ in terms of quantities computed for the net with smaller loads. As we have just shown in the previous subsection, the same is true also for the P-visit ratios of all the places of the SPN. With these two type of quantities it is possible to compute the throughput of the reference place (arbitrarily chosen) at load $\mathbf{k_0}$ using Equation (19.32) of Lemma 19.13. The definition of P-visit ratios allows then the computation of the throughputs of all the places of the net. The throughput of each place p_i allows to compute the utilisation of the same place using Equation (19.27):

$$u_i(\mathbf{k_0}) = \frac{x_i(\mathbf{k_0})}{\sum_{t_j \in p_i^\bullet} P\left(\mathbf{m} \geq \mathbf{i}(t_j) - \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}\right) \cdot \mu_j \cdot i_i(t_j)}.$$

From the utilisation of p_i, using Equation (19.25), we can compute the utilisation of any transition $t_j \in p_i^\bullet$:

$$U_j(\mathbf{k_0}) = u_i(\mathbf{k_0}) \cdot P\left(\mathbf{m} \geq \mathbf{i}(t_j) - \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}\right),$$

and hence we can compute the throughput of t_j as indicated by Equation (19.21)

$$X_j(\mathbf{k_0}) = U_j(\mathbf{k_0}) \cdot \mu_j.$$

In order to complete the recursive scheme we need to compute the cumulative probabilities that appear in most of these formulas. In particular we must evaluate

$$P\left(\mathbf{m} \geq \mathbf{i}(t_j) - h \cdot \mathbf{e_i}; \mathbf{k_0}\right) \text{ with } 1 \leq i \leq np,\ 1 \leq j \leq nt, \text{ and } 0 \leq h \leq i_i(t_j),$$

where $i_i(t_j)$ is the component of $\mathbf{i}(t_j)$ corresponding to place p_i. This computation can be performed observing that there are three main cases:

- The case of $h = 0$ can be easily solved by using the definition of transition utilisation (Equation (19.20)) that yields

$$P(\mathbf{m} \geq \mathbf{i}(t_j); \mathbf{k_0}) = U_j(\mathbf{k_0}).$$

- Knowing the values of the place utilisations, the other cumulative probabilities can be computed using the following expression in which $0 < h < i_i(t_j)$ (Equation (19.24) of Corollary 19.9):

$$P(\mathbf{m} \geq \mathbf{i}(t_j) - h \cdot \mathbf{e_i}; \mathbf{k_0}) \quad = \quad u_i(\mathbf{k_0}) \cdot P(\mathbf{m} \geq \mathbf{i}(t_j) - (h+1) \cdot \mathbf{e_i}; \mathbf{k_0} - \mathbf{S_i}).$$

- Finally, for the case when $h = i_i(t_j)$ we have to distinguish two subcases:

  1. the place p_i is the unique place in ${}^\bullet t_j$, hence we have that

$$P(\mathbf{m} \geq \mathbf{i}(t_j) - i_i(t_j) \cdot \mathbf{e_i}; \mathbf{k_0}) = 1,$$

     because $\mathbf{i}(t_j) - i_i(t_j) \cdot \mathbf{e_i} = \mathbf{0}$ and hence $P(\mathbf{m} \geq \mathbf{0}; \mathbf{k_0}) = 1$;

  2. the place p_i is not the unique place in ${}^\bullet t_j$, in this case we have that

$$P(\mathbf{m} \geq \mathbf{i}(t_j) - i_i(t_j) \cdot \mathbf{e_i}; \mathbf{k_0}) \quad = \quad u_l(\mathbf{k_0}) \cdot P(\mathbf{m} \geq \mathbf{i}(t_j) - i_i(t_j) \cdot \mathbf{e_i} - \mathbf{e_l}; \mathbf{k_0} - \mathbf{S_l}),$$

     where p_l is a place in ${}^\bullet t_j$, with $p_l \neq p_i$.

All these computational steps can be integrated into a single algorithm that can be described as follows. Given an S-invariant PF-SPN with load vector $\mathbf{k_0}$, let $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l_{mx}}$ be the sequence of all load vectors $\mathbf{l}_i \leq \mathbf{k_0}$ such that $\mathbf{l_0} = [0, 0, \ldots, 0]$, $\mathbf{l_{mx}} = \mathbf{k_0} = [k_1, k_2, \ldots, k_{nq}]$ (nq is the number of minimal P-semiflow)[2].

The MVA algorithm computes the performance indices of the PF-SPN in the following way:

```
/* Input:    P, T, 𝒯, S, k_0;
   Output:   n_i(k_0), x_i(k_0), ω_i(k_0), X_j(k_0), U_j(k_0) with 1 ≤ i ≤ np e 1 ≤ j ≤ nt */
begin
      Initialise(l_0)
      for m := 1 to mx do
          One-Step-MVA(l_m)
end
```

The details of the procedures Initialise($\mathbf{l_0}$), One-Step-MVA($\mathbf{l_m}$), and P-visits($\mathbf{l_m}, p_r$) are provided in Tables 19.3, 19.4, and 19.5.

The derivation of the time complexity for this algorithm is quite simple. The procedure One-Step-MVA is executed mx times, where $mx = \prod_{i=1}^{nq}(k_i + 1)$. The value mx represents the length of the sequence of load vectors less than or equal

[2] We say that $\mathbf{l}_i \leq \mathbf{k_0}$ when the relationship holds component wise, when it holds only for some of the components, but not for all, we say that $\mathbf{l}_i \not\leq \mathbf{k_0}$. The sequence $\mathbf{l}_0, \mathbf{l}_1, \ldots, \mathbf{l_{mx}}$ is ordered in such way that if $\mathbf{l}_i$ and $\mathbf{l}_j$ are vectors of the sequence with $i < j$ then either $\mathbf{l}_i \leq \mathbf{l}_j$ or $\mathbf{l}_i \not\leq \mathbf{l}_j$ but it cannot happen that $\mathbf{l}_i > \mathbf{l}_j$

Procedure Initialise($\mathbf{l_0}$)

(1) **begin**
(2) Compute the vector visit ratios $\mathbf{v}$
(3) Initialise the values of the probabilities when the load vector is $\mathbf{l_0}$:
$P(\mathbf{m} \geq \mathbf{iv}; \mathbf{l_0}) = 1$ if $\mathbf{iv} = [0, 0, \ldots, 0]$
$P(\mathbf{m} \geq \mathbf{iv}; \mathbf{l_0}) = 0$ if $\mathbf{iv} \neq [0, 0, \ldots, 0]$
where $\mathbf{iv} = \mathbf{i}(t_j) - h \cdot \mathbf{e_i}$, $\forall p_i \in P$ with $t_j \in p_i^\bullet$ and $0 \leq h \leq i_i(t_j)$.
(4) **end** Initialise

Table 19.3: Procedure Initialise.

Procedure One-Step-MVA($\mathbf{l_m}$)

(1) **begin**
(2) **for each** $p_i \in P$ **do**
Computation of $\omega_i(\mathbf{l_m})$ (Eq. (19.28))
(3) *Let p_r be the reference place (arbitrarily chosen)*
P-visits($\mathbf{l_m}, p_r$)
(4) Computation of $x_r(\mathbf{l_m})$ (Eq. (19.32))
(5) **for each** $p_i \in P$ **do**
Computation of
$x_i(\mathbf{l_m})$ (Eq. (19.31)),
$u_i(\mathbf{l_m})$ (Eq. (19.27)),
$n_i(\mathbf{l_m})$ (Eq. (19.26))
(6) **for each** $t_j \in T$ **do**
Computation of
$U_j(\mathbf{l_m})$ (Eq. (19.25)),
$X_j(\mathbf{l_m})$ (Eq. (19.21))
(7) **for each** $p_i \in P$ **do**
Computation of $P(\mathbf{m} \geq \mathbf{iv}; \mathbf{l_m})$ (Eq. (19.24))
(8) **end** One-Step-MVA

Table 19.4: Procedure One-Step-MVA.

to $\mathbf{k_0}$ as well as the recursion depth of the algorithm. It is possible to see that the time complexity of the One-Step-MVA is $O(\max\{np, nt\})$ where $np = |P|$, and $nt = |T|$. From this it follows that the time complexity of the algorithm is $O(mx \cdot \max\{np, nt\})$.
To carry on the recursion we need the probabilities $P(\mathbf{m} \geq \mathbf{iv}; \mathbf{l_m})$, where the vectors $\mathbf{iv}$ are defined as

$$\mathbf{iv} = \mathbf{i}(t_j) - h \cdot \mathbf{e_i} \qquad \forall p_i \in P \text{ with } t_j \in p_i^\bullet \qquad \text{and } 0 \leq h \leq i_i(t_j).$$

From this it follows that the space complexity is $O(mx \cdot np)$. It is worthwhile to point out that at step corresponding to load vector $\mathbf{l_m}$ we do not need to know the probabilities for all $\mathbf{l}' < \mathbf{l_m}$, but only those corresponding to the load vectors $\mathbf{l_m} - \mathbf{S_i}$ $\forall\, p_i \in P$. Hence space savings could be obtained with a careful implementation of the algorithm (following ideas similar to those used for the implementation of the classical MVA algorithm for product form queueing networks) and the previous space complexity should be interpreted only as an upper bound.

```
Procedure P-visits(l_m, p_r)
(1)     begin
(2)         visit the reference place p_r
            set the values u_r(l_m)/u_r(l_m) = ν_r(l_m) = 1
(3)         for each p_h adjacent to p_r do
                DFS(h, r, l_m)
(4)     end  P-visits

Procedure DFS(l, i, l_m)
(1)     begin
(2)         visit the place p_l (from its ancestor p_i)
            compute the values
            u_l(l_m)/u_r(l_m) (Eq. (19.34)) and ν_l(l_m) (Eq. (19.35))
(3)         for each p_s adjacent to p_l do
                if p_s has not been visited yet then DFS(s, l, l_m)
(4)     end  DFS
```

Table 19.5: Procedure P-visits.

19.2.1 An Approximation Technique

The more appealing feature of the recursive formulation of MVA is the possibility of developing of approximation techniques that become quite convenient when the number of tokens per P-semiflows is very large. Indeed approximation techniques that allow the computation of the performance indices of PF-SPNs in these cases are relatively easy to obtain. In particular the MVA algorithm that has been described in this section can be modified to devise an approximation method inspired by similar heuristic techniques that have been developed for PF-QNs (e.g. see [35]). The idea behind this approximation algorithm is that of substituting the recursive structure of the algorithm of Section 19.2 with an iterative scheme. The heuristic method used to convert the recursion into an iteration is that of estimating the quantities that have been initialised at algorithm start on the basis of their values computed by the procedure One-Step-MVA. The basic scheme of the approximate algorithm derived from the MVA for PF-SPNs can be described as follows:

1. Let $\mathbf{k_0}$ be the load vector of the SPN. Initialise the values of the average number of tokens $n_i(\mathbf{k_0} - \mathbf{S_i}), \forall\, p_i \in P$ as well as those of the probabilities $P(\mathbf{m} \geq \mathbf{iv}; \mathbf{k_0} - \mathbf{S_i})$.
2. Compute the performance indices with the following procedure:

```
/* Input:    P, T, 𝒯, S, v, k_0;
   Output:   n_i(k_0), x_i(k_0), ω_i(k_0), X_j(k_0), U_j(k_0) */
begin
        repeat
            One-Step-MVA(k_0)
            Correction Phase: Recomputation of the values
                    P(m ≥ iv; k_0 − S_i) and n_i(k_0 − S_i)
        until Convergency is reached
end
```

The initialisation is performed as follows: the values $n_i(\mathbf{k_0} - \mathbf{S_i})$ are obtained spreading the tokens over the places of the net taking into account the P-semiflow constrains while the values of $P(\mathbf{m} \geq \mathbf{iv}; \mathbf{k_0} - \mathbf{S_i})$ are obtained by assigning to these probabilities any value belonging to the interval $(0, 1)$.

Both the above initialisations are non-critical, i.e., this step can be also performed according to different criteria and the difference between these possible choices is reflected only in the number of iteration steps needed to satisfy the convergency limit. As in all these methods, the critical step of this approximation technique is instead the *Correction phase* [35]. The method that has been used in [40] is based on the observation that for large load vectors the probabilities $P(\mathbf{m} \geq \mathbf{iv}; \mathbf{k_0})$ and $P(\mathbf{m} \geq \mathbf{iv}; \mathbf{k_0} - \mathbf{S_i})$ differ only for negligible amounts. This means that the contribution of an additional token (step from $\mathbf{k_0} - \mathbf{S_i}$ to $\mathbf{k_0}$) is marginal. From this, for the correction step of the probabilities, we set

$$P(\mathbf{m} \geq \mathbf{iv}; \mathbf{k_0} - \mathbf{S_i}) = P(\mathbf{m} \geq \mathbf{iv}; \mathbf{k_0}).$$

For the average number of tokens we correct the values as follow

$$n_i(\mathbf{k_0} - \mathbf{S_i}) = \alpha \cdot n_i(\mathbf{k_0}),$$

where α is a constant that normalises the sum of tokens in the corresponding P-semiflow.
In Table 19.6 we compare the approximation technique with the MVA algorithm, using as a test the SPN of Figure 18.2. This table compares the throughputs obtained with the two methods. The last two columns report the error. As one can see the errors are quite low, they decrease as the load of the net increases and the computational effort remains really negligible for all the cases that we have analyzed.

$\mathbf{k_0}$	Transition	MVA		Approximation			Error (%)	
		$X_j(\mathbf{k_0})$	Time	$X_j(\mathbf{k_0})$	Time	Iter.		Max
$[10, 9, 8]$			2.6		0.2	95		7
	t_1	0.940		0.923			2	
	t_4	4.572		4.329			5	
	t_6	5.314		4.927			7	
$[30, 20, 10]$			18.6		0.3	116		3
	t_1	1.000		0.990			1	
	t_4	4.961		4.862			2	
	t_6	5.728		5.547			3	
$[45, 30, 20]$			78.1		0.3	117		2
	t_1	1.000		0.995			1	
	t_4	4.993		4.934			1	
	t_6	5.823		5.720			2	

Iter. = Number of iterations required to reach the convergency.

Table 19.6: Comparisons between the MVA algorithm and the approximation technique.

The possibility of using the basic ideas contained in these derivations for devising computationally efficient methods for the analysis of certain classes of non-PFS SPNs is currently under study. Preliminary results can be found in [38].

19.3 Conclusions

In this chapter we have shown that there exists a class of SPNs whose steady state probability distribution of SPNs have product form solutions. This class of nets exhibits interesting properties. In Section 18.6 of Chapter 18 some qualitative properties of this class of SPNs have been presented. In Section 19.1 and Section 19.2 we have shown that for this class of nets the performance measures can be efficiently computed using algorithms whose space and time complexities are polynomial in the number of places and in the initial marking of the SPN.

Several open points remain, for instance the reachability properties are not well characterised. This problem does not allow to have a complete "structural" characterisation of the class of nets whose performance measures can be efficiently computed using algorithms like the convolution or the MVA algorithm.

Other interesting point that can be investigated concern the possibility of using the PFS developed for this class of SPNs as basis to obtain approximate techniques. In the queueing network environment there are several examples of this possibility.

19.4 Bibliographic Remarks

The first product form result for SPNs can be attributed to Lazar and Robertazzi [27, 28] who consider SPNs at the marking level. They define a class of safe and live SPNs whose equilibrium distributions satisfy partial balance equations and have product form equilibrium distributions. The main limitation of the Lazar and Robertazzi product form result is that this criterion requires the generation of the the reachability graph and hence it cab be used only to check the existence of the product form solution.
Li and Georganas [29] extend the result of Lazar and Robertazzi allowing more general SPNs.

Frosch, in [15], considers a class of nets called closed synchronised systems of stochastic sequential processes (CS). This class of nets consist of cyclic state machines which can be joined together through common buffer places such that the flow of tokens through the buffer places is conserved for every pair of sequential transitions in any state machine. The product form which Frosch obtains is both a product over each state machines and over each place and is based on the structure of the net. Unlike Lazar and Robertazzi Frosch allows state dependent firing rates and does not restrict to safe nets.
Henderson, Lucic, and Taylor in [18] obtained a product form criterion based only on the structure of the net, without the need to generate the reachability graph. Their class of nets originates from results obtained within the framework of the batch movement queueing networks. For the class of SPNs identified by this product form criterion several results have been proposed. In [11] a method

for computing the equilibrium distribution as a product of terms relating with the places of the SPN has been proposed. In [4] a necessary and sufficient condition for the existence of a positive solution for the traffic equations has been proposed. This condition is based on a structural analysis of the SPN and does not require the generation of the the reachability graph.
For the class of SPNs characterised by the Henderson *et al.* product form result several algorithms for an efficient computation of the performance measures have been proposed. In particular algorithms for the computation of the normalisation constant have been proposed in [9, 11, 39]. In the papers [1, 12, 40] there are several proposals for Mean Value Analysis algorithms for this class of PF-SPNs.

Boucherie, in [3], proposed a product form criterion for SPNs using a Markov chain approach. In this case equilibrium distribution is a product of terms relating to subnets instead of places or transitions.

Another product form criterion has been proposed by Florin and Natkin, in [14], where the authors consider a class of synchronised queueing networks which can be modelled by ordinary and bounded SPNs with marking independent transition firing rates and a reachability graph which is strongly connected. For these nets they obtain a matrix geometric product form result where the equilibrium distribution is a sum of product forms.

Bibliography

[1] G. Balbo, S. C. Bruell, and M. Sereno. Arrival theorems for product-form stochastic Petri nets. In *Proc. 1994 ACM SIGMETRICS Conference*, pages 87–97, Nashville, Tennessee, USA, May 1994. ACM.

[2] F. Baskett, K. M. Chandy, R. R. Muntz, and F. Palacios. Open, closed and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, April 1975.

[3] R. J. Boucherie. A characterization of independence for competing markov chain with applications to stochastic Petri nets. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press. IEEE.

[4] R. J. Boucherie and M. Sereno. On closed support t-invariants and traffic equations. Technical Report ERCIM-04/94-R032, European Research Consortium for Informatics and Mathematics, 1994. To appear on the journal *Adv. Appl. Prob.*

[5] R. J. Boucherie and N. M. van Dijk. Product forms for queueing networks with state dependent multiple job transitions. *Advances on Applied Probability*, 23, 1991.

[6] S. C. Bruell and G. Balbo. *Computational Algorithms for Closed Queueing Networks*. Elsevier North-Holland, New York, 1980.

[7] J. P. Buzen. Computational algorithms for closed queueing networks with exponential servers. *Communications of the ACM*, 16(9):527–531, September 1973.

[8] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):1368–1382, December 1991.

[9] J. L. Coleman. Algorithms for product-form stochastic Petri nets - a new approach. In *Proc.* 5^{th} *Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

[10] J. L. Coleman. *Stochastic Petri Nets with Product Form Equilibrium Distributions*. PhD thesis, University of Adelaide, Dec 1993. PhD Thesis.

[11] J. L. Coleman, W. Henderson, and P. G. Taylor. Product form equilibrium distributions and an algorithm for classes of batch movement queueing networks and stochastic Petri nets. *Performance Evaluation*, 26(3):159–180, September 1996.

[12] A. J. Coyle, W. Henderson, C. E. M. Pearce, and P. G. Taylor. A general formulation for the mean-value analysis in product form batch-movement queueing networks. *Queueing Systems*, 16:363–372, 1994.

[13] S. Donatelli and M. Sereno. On the product form solution for stochastic Petri nets. In *Application and Theory of Petri Nets 1992, Proc.* 13^{th} *Intern. Conference*, LNCS, N. 616, pages 154–172, Sheffield, England, June 1992. Springer Verlag.

[14] G. Florin and S. Natkin. Matrix product form solution for closed synchronized queueing networks. *IEEE Trans. Soft. Eng.*, 17(2), February 1991.

[15] D. Frosch. Product form solution for closed synchronized systems of stochastic sequential processes. In *Proc. Intern. Computer Symp.*, pages 392–402, Taichung, Taiwan, December 1992.

[16] W. J. Gordon and G. F. Newell. Closed queueing systems with exponential servers. *Operations Research*, 15:254–265, 1967.

[17] P. Harrison and J. Hillston. Exploiting quasi-reversible structures in Markovian process algebra models. *Computer Journal*, 38(7), 1995.

[18] W. Henderson, D. Lucic, and P.G. Taylor. A net level performance analysis of stochastic Petri nets. *Journal of Australian Mathematical Soc. Ser. B*, 31:176–187, 1989.

[19] W. Henderson, C. E. M. Pearce, P.G. Taylor, and N. M. van Dijk. Closed queueing networks and batch services. *Queueing Systems*, 6:59–70, 1990.

[20] W. Henderson and P.G. Taylor. Embedded processes in stochastic Petri nets. *IEEE Transactions on Software Engineering*, 17:108–116, February 1991.

[21] J. R. Jackson. Jobshop-like queueing systems. *Management Science*, 10(1):131–142, October 1963.

[22] F. P. Kelly. Markov processes and Markov random fields. *Bull. Inst. Int. Statist.*, 46:397–404, 1975.

[23] F. P. Kelly. Networks of queues with customers of different types. *J. Appl. Prob.*, 12:542–554, 1975.

[24] F. P. Kelly. *Reversibility and Stochastic Networks*. London: Wiley, 1979.

[25] A. A. Lazar. An alebraic topological approach to Markovian queueing networks. In *Proc. of the 1984 Information Science and Systems Conference*, Princeton, NJ, USA, March 1984.

[26] A. A. Lazar and T. G. Robertazzi. The geometry of lattices for multiclass Markovian queueing networks. In *Proc. 18-th Conference on Information Science and Systems*, pages 164–168, Princeton University, Princeton, NJ, USA, 1984.

[27] A. A. Lazar and T. G. Robertazzi. Markovian Petri net protocols with product form solution. In *Proc. of INFOCOM '87*, pages 1054–1062, San Francisco, CA, USA, 1987.

[28] A. A. Lazar and T. G. Robertazzi. Markovian Petri net protocols with product form solution. *Performance Evaluation*, 12:67–77, 1991.

[29] M. Li and N. D. Georganas. Parametric analysis of stochastic Petri nets. In *Fifth International Conference on Modelling and Tools for Computer Performance Evaluation*, Torino, Italy, February 1991.

[30] J. D. C. Little. A proof of the queueing formula $l = \lambda w$. *Operations Research*, 9:383–387, 1961.

[31] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In *Proc.* 2^{nd} *European Workshop on Application and Theory of Petri Nets*, Bad Honnef, West Germany, September 1981. Springer Verlag.

[32] G. Memmi and G. Roucairol. Linear algebra in net theore. In *Proceedings of the Advanced Course on General Net Theory of Processes and Systems*, LNCS, N. 84, Hamburg, Germany, 1979. Springer Verlag.

[33] M. K. Molloy. Petri net modelling, the past, the present, and the future. In *Proc.* 3^{th} *Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.

[34] M. Reiser and S. S. Lavenberg. Mean value analysis of closed multichain queueing networks. *Journal of the ACM*, 27(2):313–322, April 1980.

[35] P. J. Schweitzer. A survey of mean value analysis, its generalizations, and applications, for networks of queues. Technical report, University of Rochester, Rochester, USA, Dec 1990.

[36] E. Seneta. *Non-negative matrices and Markov Chains.* Springer Verlag, 1981.

[37] M. Sereno. Towards a product form solution for stochastic process algebras. *Computer Journal*, 38(7):622–632, 1995.

[38] M. Sereno. Approximate mean value analysis for stochastic marked graphs. *IEEE Transactions on Software Engineering*, 22(9):654–664, 1996.

[39] M. Sereno and G. Balbo. Computational algorithms for product form solution stochastic Petri nets. In *Proc. 5^{th} Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

[40] M. Sereno and G. Balbo. Mean value analysis of stochastic Petri nets. *Performance Evaluation*, 29(1):35–62, 1997.

[41] R. Serfozo. Markovian network processes: Congestion-dependent routing and processing. *Queueing Systems*, 5:5–36, 1989.

[42] M. W. Shields. *An Introduction to Automata Theory.* Blackwell Scientific Publication, 1987.

Part VI

Decomposition and Approximation

Chapter 20

Introduction to Net-Driven Decomposition Techniques

As stated in previous chapters, a fundamental question in the use of stochastic Petri net models for performance evaluation, even under Markovian stochastic interpretation, is the so called *state explosion problem*. A general approach to deal with (computational) complexity is to use a *divide and conquer* strategy, what requires the definition of a decomposition method and the subsequent composition of partial results to get the full solution. On the other hand, the trade-off between computational cost and accuracy of the solution leads to the use of approximation or bounding techniques (for instance, throughput bounds can be computed in polynomial time on the number of transitions and places, see Chapter 17). In this context, a pragmatic compromise to be handled by the analyzer of a system concerns the definition of faithful models, that may be very complex to exactly analyse (what may lead to the use of approximation or just bounding techniques), or simplified models, for which exact analysis can be, eventually, accomplished. Divide and conquer strategies can be used with exact, approximate, or bounding techniques.

The techniques for performance evaluation present in the literature consider either implicit or explicit decomposition of net models. In [7] (see Chapter 17), an implicit decomposition into P-semiflows is used for computing throughput bounds for arbitrary pdf of time durations. In [19] (see Chapter 12), a decomposition into disjoint *modules* (usually subnets generated by P-semiflows) is defined by the analyzer or provided by model construction; the computational technique uses directly the information provided by the modules to compute exact global limit probability distribution vector. In [8], a decomposition into *modules* (connected through buffers) should also be provided. In this case, the modules are complemented with an abstract view of their environment in the full model, and the approximate solution is computed through an *iterative technique* looking for a fixed point (details will be presented in Chapter 23). In the sequel of this Chapter and in a general context, *components* will refer to

(eventually) complemented modules. They are just the elements used to build the full solution.

In order to get efficient techniques, the decomposition and, eventually, complementation process should be net-driven (i.e., derived at net level). For this, we shall use PN's structure theory concepts and techniques (e.g., P-semiflows, implicit places, etc.).

The Chapter is organised as follows. The main ideas behind net-driven decompositions of PN's are introduced in Section 20.1: the conservative and consistent components (Section 20.1.1), an example of implicit search technique into conservative components (Section 20.1.2), an explicit decomposition of nets, designating the modules to be used (Section 20.1.3), and the complementation of modules to get components (that include information from the environment), using implicit places (Section 20.1.4). A taxonomy for net-driven decomposition techniques is proposed in Section 20.2, providing a framework for the consideration of a significant number of performance evaluation methods. Several representative examples of techniques present in the literature are briefly overviewed and classified according with the classification criteria. Some concluding remarks are included in Section 20.3.

20.1 Basic Concepts and Techniques

In Section 20.1.1, the basic standard components of nets, conservative and consistent components, are derived from vectors (P- and T-semiflows) defined from algebraic properties of the incidence matrix $\mathbf{C}$ of the net. An implicit search technique into conservative components of a net is presented in Section 20.1.2. In other cases the decomposition must be done explicit, designating clearly the modules to be used (Section 20.1.3). Finally, the obtained modules can be complemented in order to (partially) re-introduce *synchronic dependences* that the decomposition removed. This is the topic of Section 20.1.4, where implicit places are used for the mentioned purpose.

20.1.1 Standard components

When a transition t is enabled at $\mathbf{m}$ ($\mathbf{m}[p] \geq \mathbf{Pre}[p,t]$) the new marking reached by its firing ($\mathbf{m} \xrightarrow{t} \mathbf{m}'$) is $\mathbf{m}' = \mathbf{m} + \mathbf{C}[P,t]$. If we integrate it over a sequence σ of fired transitions from the initial state $\mathbf{m_0}$ and yielding $\mathbf{m}$, denoting by $\boldsymbol{\sigma}$ the *firing count vector* of sequence σ ($\boldsymbol{\sigma}[t] = \#(t,\sigma)$ is the number of times t occurs in the sequence), we obtain (see Chapter 2):

$$\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma} \tag{20.1}$$

This equation resembles the state equation of a discrete time linear system, therefore it is frequently referred as the *net state equation.* Observe that some integer solutions of this equation may be non reachable in the net system (those are called "spurious solutions").

Premultiplying the state equation by $\mathbf{y} \geq \mathbf{0}$ such that $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$ —vector $\mathbf{y}$ is called a *P-semiflow*— then, for every initial marking $\mathbf{m_0}$, every reachable marking $\mathbf{m}$ satisfies:

$$\mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0} + \mathbf{y} \cdot \mathbf{C} \cdot \boldsymbol{\sigma} = \mathbf{y} \cdot \mathbf{m_0} = k \tag{20.2}$$

This provides a "token conservation law": for every reachable marking the weighted sum of tokens in $\|\mathbf{y}\|$ remains constant, where $\|\mathbf{y}\| = \{p \mid \mathbf{y}[p] > 0\}$ is the *support* of vector $\mathbf{y}$.

Besides the invariant laws, a major interest of P-semiflows is the *decomposed view* of the model that they provide. The P-subnet generated by the support of a P-semiflow is called a *conservative component* of the net, meaning that it is a part of the net that conserves its weighted token content. A conservative component allows neither incoming nor outcoming of tokens (customers, resources, servers...). Thus conservative components are *closed* subsystems. If there exists a $\mathbf{y} > \mathbf{0}$ such that $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$, then the net is said to be *conservative*, thus bounded for every initial marking. It is important to realise that we introduce three notions that should be differentiated:

- the P-semiflow (a vector: $\mathbf{y} \geq \mathbf{0}$, $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$),
- the token conservation law or marking invariant (an equation: $\mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0}$), and
- the conservative component (a net: the subnet generated by $\|\mathbf{y}\|$ and its input and output transitions).

By definition of P-semiflow, if there is at least one for a given net, then there exists an infinite number of them. All the information contained in the token conservation laws can be extracted from a subset of P-semiflows called *minimal P-semiflows*. A P-semiflow is said to be minimal when its positive entries $\mathbf{y}_i$ are relatively prime and no P-semiflow $\mathbf{y}'$ exists such that $\|\mathbf{y}'\| \subset \|\mathbf{y}\|$. The set of all the minimal P-semiflows, called the *fundamental set* of P-semiflows, is unique (see [16] for its computation).

Let us consider the net depicted in Figure 20.1.a. The computation of minimal P-semiflows gives a fundamental set with two elements: $\mathbf{y_1} = (1, 0, 1, 1, 0)$ and $\mathbf{y_2} = (1, 1, 0, 0, 1)$. The corresponding token conservation laws are: $\mathbf{m}[\mathsf{p1}] + \mathbf{m}[\mathsf{p3}] + \mathbf{m}[\mathsf{p4}] = 2$ and $\mathbf{m}[\mathsf{p1}] + \mathbf{m}[\mathsf{p2}] + \mathbf{m}[\mathsf{p5}] = 1$. The two conservative components associated to the above P-semiflows are depicted in Figure 20.1.b and Figure 20.1.c.

The dual notion of P-semiflows are T-semiflows. If $\mathbf{x} \geq \mathbf{0}$ is such that $\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$:

$$\mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \mathbf{x} = \mathbf{m_0} \tag{20.3}$$

T-semiflows correspond to cyclic sequences in the sense that the firing count vector of a cyclic sequence of a bounded system is a T-semiflow. Nevertheless, it should be pointed out that eventually for a given initial marking may be *not* possible to fire a sequence whose firing count vector is a given T-semiflow.

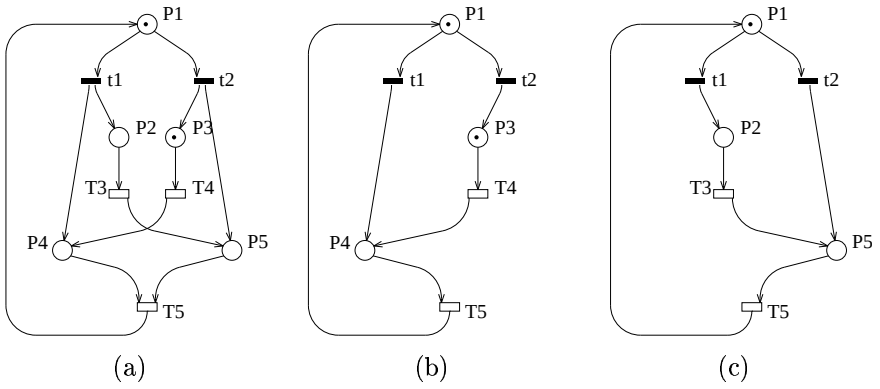


Figure 20.1: A net (a) and its conservative components (b) and (c).

For example, for the net in Figure 20.1.a., two minimal T-semiflows can be computed (using the same algorithm for computing P-semiflows but applied on the transposed incidence matrix): $\mathbf{x_1} = (1,0,1,0,1)$ and $\mathbf{x_2} = (0,1,0,1,1)$.

The T-subnet generated by the support of a T-semiflow is called a *consistent component* of the net. Consistent components provide an alternative decomposed view of the net.

In the case that $\mathbf{x} > \mathbf{0}$ such that $\mathbf{C} \cdot \mathbf{x} = \mathbf{0}$ exists, the whole net is a consistent component, and it is said that the net is *consistent*.

When a net is not consistent it cannot be lively and boundedly marked (see, for example, [32]).

20.1.2 P-Semiflows and Performance Bounds: An Implicit Search in Conservative Components.

The *visit ratio* of transition t_j with respect to t_i, $\mathbf{v}^{(i)}[t_j]$, is the average number of times t_j is visited (fired) for each visit to (firing of) the reference transition t_i (see Chapter 8). The vector of visit ratios of a live and bounded system must be a T-semiflow (the input weighted flow to each place must be equal to the output weighted flow): $\mathbf{C} \cdot \mathbf{v}^{(i)} = \mathbf{0}, \mathbf{v}^{(i)} \geq \mathbf{0}$ (otherwise stated: $\mathbf{v}^{(i)} = \sum_j \lambda_{ij}\mathbf{x_j}$, $\mathbf{v}^{(i)}[t_i] = 1$). The visit ratios of transitions in *equal conflict*, i.e., such that their preconditions are the same ($\mathbf{Pre}[t_j] = \mathbf{Pre}[t_k]$) must be fixed by the corresponding routing rates. For instance, for the net in Figure 20.1.a, the following equation holds: $r_1\mathbf{v}^{(i)}[\mathsf{t2}] - r_2\mathbf{v}^{(i)}[\mathsf{t1}] = 0$, where r_1 and r_2 are the routing rates of t1 and t2. In summary, the next result can be stated [12]: The vector of visit ratios with respect to transition t_i of a live and bounded net system must be a solution of:

$$\mathbf{C} \cdot \mathbf{v}^{(i)} = \mathbf{0}; \;\; \mathbf{R} \cdot \mathbf{v}^{(i)} = \mathbf{0}; \;\; \mathbf{v}^{(i)}[t_i] = 1 \tag{20.4}$$

where $\mathbf{R}$ is a matrix that relates the visit ratios of transitions in equal conflict ($\mathbf{Pre}[t_j] = \mathbf{Pre}[t_k]$) according to the corresponding routing rates.

Equations in the above statement have been shown to characterize a unique vector $\mathbf{v}^{(i)}$ for important net subclasses [7, 12] (a condition over the rank of $\mathbf{C}$ and the number of equal conflicts underlies these cases).

The *average service demand* from transition t_j with respect to t_i is defined as $\overline{\mathbf{D}}^{(i)}[t_j] = \mathbf{s}[t_j]\,\mathbf{v}^{(i)}[t_j]$.

From the classical Little's law (applied to the places of a SPN) and using the token conservation laws derived from P-semiflows the following result can be stated [7] (see Chapter 17): For any net system with infinite-server semantics assumed for transitions, a lower bound for the average interfiring time $\Gamma[t_i]$ of a transition t_i can be computed by solving the following linear programming problem (LPP):

$$\begin{array}{rlll} \Gamma[t_i] \geq & \text{maximum} & \mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}}^{(i)} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & & \mathbf{y} \cdot \mathbf{m_0} = 1 \\ & & \mathbf{y} \geq \mathbf{0} \end{array} \tag{20.5}$$

If the solution of the LPP (20.5) is unbounded and since it is a lower bound for the average interfiring time of transition t_i, the non-liveness can be assured (infinite interfiring time). If the visit ratios of all transitions are non-null (i.e., $\mathbf{v}^{(i)} > \mathbf{0}$), the unboundedness of the above problem implies that a total deadlock is reached by the net system. Anyhow, the unboundedness of the solution of (20.5) means that there exists an unmarked P-semiflow, and obviously the net system is non-live: if $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$ and $\mathbf{y} \cdot \mathbf{m_0} = 0$, then $\forall \mathbf{m}\ \forall p \in \|\mathbf{y}\|$: $\mathbf{m}[p] = 0$, and the input and output transitions of p are never firable.

The basic advantage of the LPP (20.5) lies in the fact that the *simplex method* for the solution of an LPP has almost linear complexity in practice, even if it has exponential worst case complexity. In any case, algorithms of polynomial worst case complexity can be found in [27].

In order to interpret the LPP (20.5), let us rewrite it as the following fractional programming problem where the interpretation in terms of P-semiflows is even more clear:

$$\begin{array}{rlll} \Gamma[t_i] \geq & \text{maximum} & \dfrac{\mathbf{y} \cdot \mathbf{Pre} \cdot \overline{\mathbf{D}}^{(i)}}{\mathbf{y} \cdot \mathbf{m_0}} \\ & \text{subject to} & \mathbf{y} \cdot \mathbf{C} = \mathbf{0} \\ & & \mathbf{y} \geq \mathbf{0} \end{array} \tag{20.6}$$

Therefore, the lower bound for the average interfiring time of t_i in the original net system given by (20.6) is computed *looking at the "slowest subsystem" generated by the P-semiflows*, considered in *isolation* (with delay nodes).

Let us consider again the net system of Figure 20.1.a. Assuming that the vector of average service times of transitions (for arbitrary pdf's; i.e., Markovian assumption is not needed) is $\mathbf{s} = [0, 0, 1, 10, 3]$ and that the routing rates associated with t1 and t2 are identical, then the system (20.4) gives $\mathbf{v}^{(5)} =$

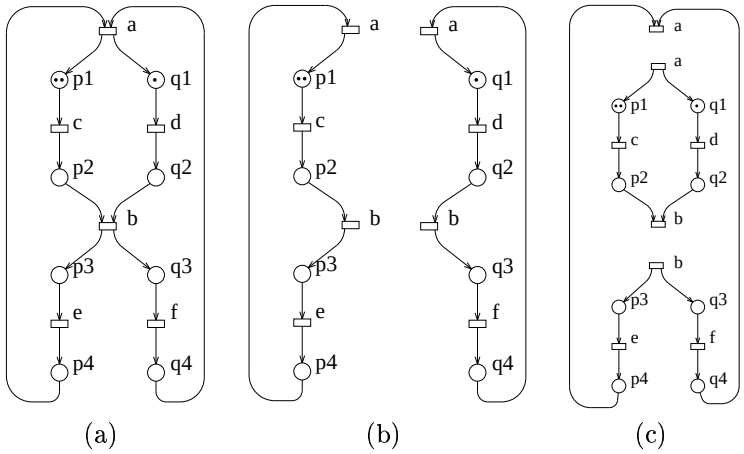


Figure 20.2: A SPN (a) and two possible decompositions (b,c) obtained by cutting through transitions a and b.

$(0.5, 0.5, 0.5, 0.5, 1)$ and the vector of average service demands for transitions (normalized for T5) is $\overline{\mathbf{D}}^{(5)} = (0, 0, 0.5, 5, 3)$.

The application of (20.5) or (20.6) to the minimal P-semiflows $\mathbf{y_1}$ and $\mathbf{y_2}$ gives:

$$\Gamma[\mathsf{T5}] \geq \max\{(5+3)/2, 0.5+3\} = 4 \tag{20.7}$$

The quantities under the max operator in (20.7) represent, for this particular case, the average interfiring time of transition T5 at each of the two subnets depicted in Figures 20.1.b and 20.1.c (embedded queueing networks in this particular case) assuming that all the nodes are delay stations (infinite-server semantics).

The reader should notice that the above linear programming problem makes an *implicit search* of the slowest subsystem among those defined by the minimal P-semiflows; it is a *bottleneck* analysis.

20.1.3 Explicit Decomposition of Nets: Modules

The first step in any technique based on the divide and conquer approach for the analysis on SPN's is the decomposition of the model into two or more "pieces" or *modules*. Since PN's are bipartite graphs, the decomposition will consist basically in cutting the graph by selecting a set of vertices (places or transitions) that are going to be the *interface* (or border) among the different modules.

A typical way of decomposing systems is to consider *rendez vous* synchronizations. In PN terms, the basic case is obtained by cutting a transition in a "parallel" way with respect to the flow of tokens and leaving a process in each side (the number of input and output places in each side is "balanced"). Consider, for instance, the system depicted in Figure 20.2.a and assume that

transitions a and b are going to define the cut. In Figure 20.2.b, the two obtained processes are depicted.

An alternative way of decomposing systems is to consider the splitting through buffers or mailboxes, leaving in one side the inputs and in the other the outputs. In this case, the cut is "orthogonal" with respect to the flow of tokens, in net terms. By refining the place that represents the buffer into a place-transition-place sequence, the above cut can be expressed by cutting also orthogonally to the flow of tokens in the introduced transition. For the same system in Figure 20.2, a cut orthogonal with respect to the flow of tokens through transitions a and b is depicted in Figure 20.2.c.

The first kind of compositions/decompositions introduced above are usually called *synchronous* while the second kind are referred as *asynchronous*. Figure 20.3.b shows an example of asynchronous decomposition of the system in Figure 20.3.a through places b1, b2, b3, b4. The model is split into three modules (subnets generated by places labeled with a, c and d, respectively).

A general cut of a PN through a transition can be defined by *partitioning* the set of input and output places of the transition into two or more subsets, each of one belonging to a different module. For example, a cut of the system in Figure 20.3.a through transitions Tl1 (b1,b2/c2,c4), Tl2 (b3,b4/c5,c6), Tl5 (b2/d1,d2,d6,d10), Tl6 (b4/d1,d5,d9,d13) is shown in Figure 20.3.c. The model is, in this case, also split into three modules (subnets generated by places starting with a or b for the first module, c for the second module, and d for the third one).

By means of T-P duality in nets, general cuts through places can also be defined. Nevertheless they need a more careful definition concerning the initial marking of the subsystems. In any case, they are usually less intuitive than the cuts through transitions.

20.1.4 Obtaining Components from Modules: Implicit Places

The above cuts decompose a net producing modules. In this section we present, in an informal way, the main ideas behind a technique for transforming modules into components. The main problem to solve is that "decomposition" means removing behavioural constraints, thus the state space of modules considered in isolation may be much larger (even unbounded) than when considered inside the full model (i.e., what corresponds to the projection of the full behaviour on the preserved nodes). Therefore, the natural idea is to restrict the behaviour of the modules, considered in isolation, with *synchronic constraints* inherited from the rest of the model. An alternative way of understanding the above complementation process of modules is to see that the added nodes implement a *reduction* of the behaviour of the rest of the model.

Using the above complementation process, the original system may be covered by components. In each component, *only one of the different modules of the original system is kept while the internal structure of the others is reduced as much as possible.*

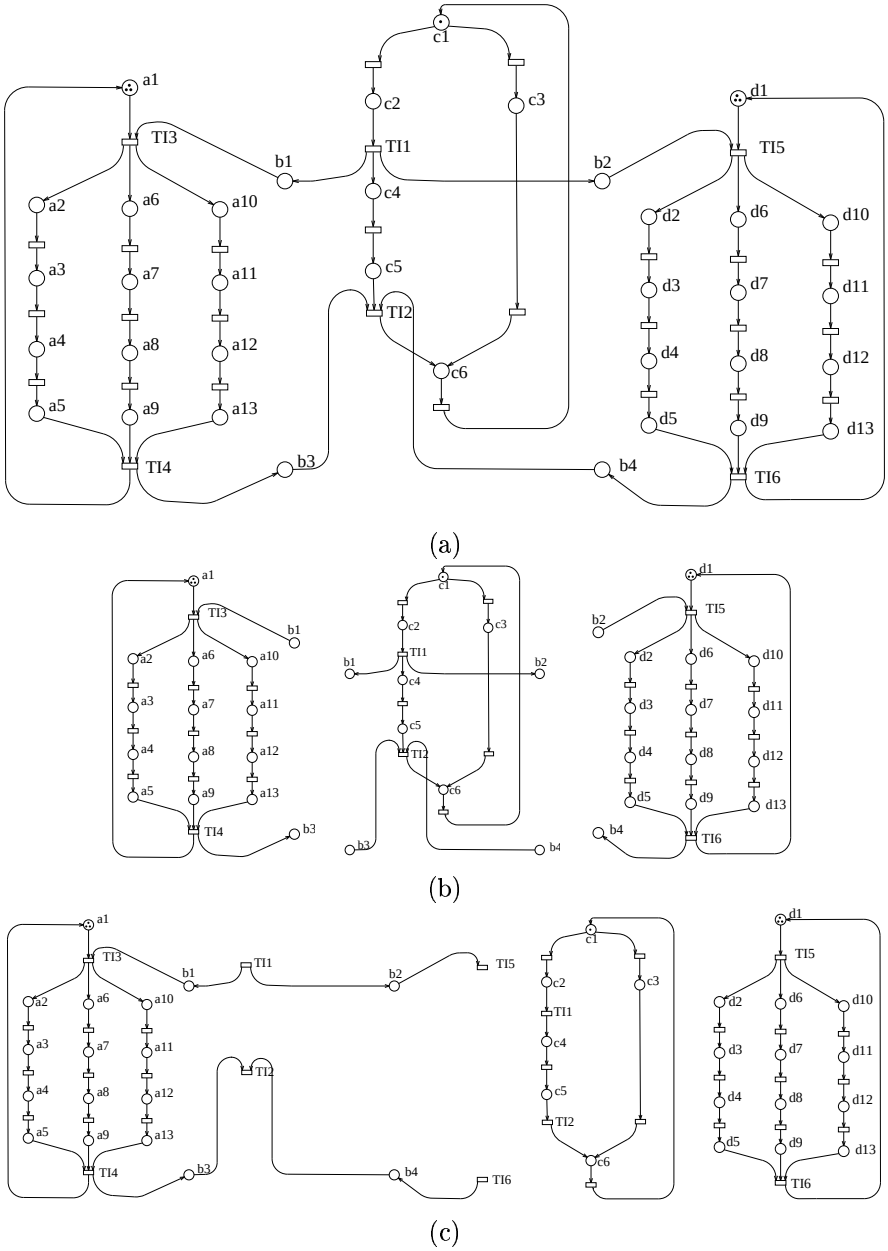


Figure 20.3: A SPN (a) and two possible decompositions obtained by cutting through (b) places b1, b2, b3, b4 and (c) transitions TI1, TI2, TI5, TI6.

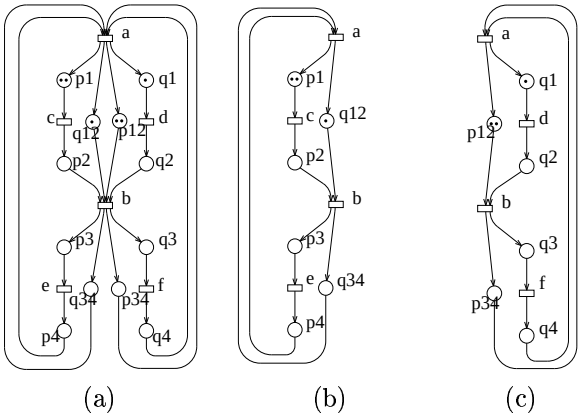


Figure 20.4: Deriving components from the modules in Figure 20.2.b.

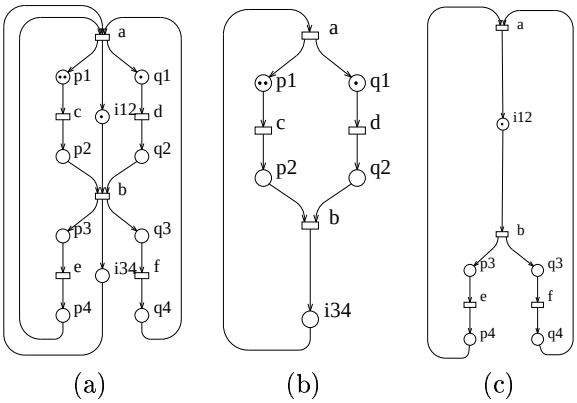


Figure 20.5: Deriving components from the modules in Figure 20.2.c.

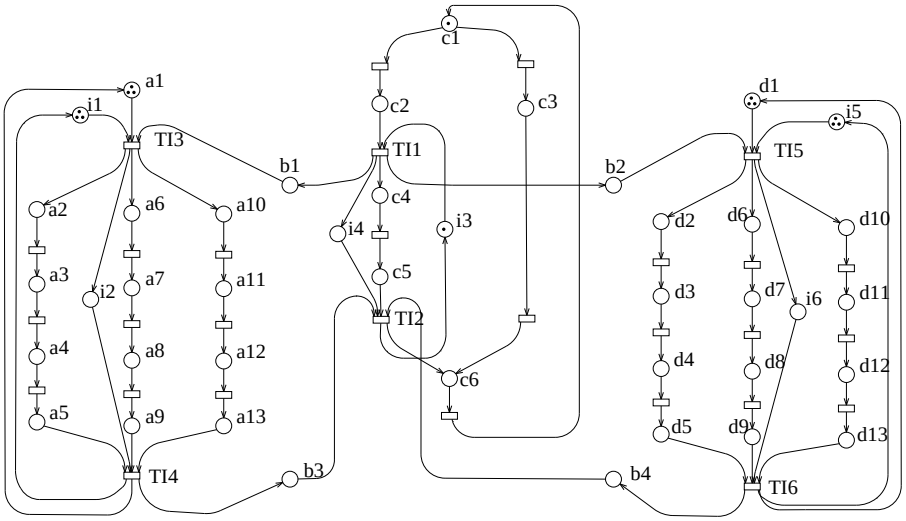


Figure 20.6: Extended system for the model in Figure 20.3.a.

Let us come back to the example in Figure 20.2. In order to complement the modules obtained in the first (synchronous) decomposition, depicted in Figure 20.2.b, an *extended system*, $\mathcal{ES}$, is derived from the original like that depicted in Figure 20.4.a. It consists of the original system plus the addition of some *implicit places*: q12 and q34 will be added to the module on the left (Figure 20.4.b) summarizing information from that on the right (respectively, p12 and p34 will be added to the module on the right as shown in Figure 20.4.c). An implicit place [17] (see Chapter 6) is one whose removal does not affect the behaviour of the system (therefore, behaviour of the original and of the extended systems is the same). Here, by behaviour we understand the *interleaving semantics*, i.e., the sequential observations or language [17], although the notion of implicit place can be directly extended to cope with a *step semantics* [15]. If a Markovian interpretation of PN's and single-server semantics of transitions is considered, the embedded CTMC of a system is preserved if implicit places are added or removed.

The extended system in Figure 20.4.a can be synchronously decomposed into the components in Figures 20.4.b and 20.4.c. The reader should notice that, except q12, the places computed as implicit in the original system (Figure 20.4.a) are also implicit in the components, so they could be deleted without changing their behaviour.

Consider now the second decomposition proposed in Figure 20.2.c for the same system in Figure 20.2.a. The modules can be complemented using the places i12 and i34 depicted in Figure 20.5.a. In this case, the addition of these places to the modules is crucial since it leads to bounded components (Figures 20.5.b and 20.5.c) while the original modules were unbounded.

After these introductory examples, let us concentrate in the decomposition process in a more general case, always staying semi-formal/illustrative. We adopt the so called SAM technique or view [11] (see Chapter 12), where modules are connected asynchronously through buffers (SAM stands for "System of Asynchronously communicating Modules"). Let us consider again the example in Figure 20.3.a distinguishing the three modules $\mathcal{N}_1$, $\mathcal{N}_2$, and $\mathcal{N}_3$ depicted in Figure 20.3.b, where places b1, b2, b3, b4 are the buffers. The extended system is depicted in Figure 20.6, where the labels of the added implicit places start with i. The way the implicit places to be added are computed is the following. First, an *equivalence relation* R is introduced in the set of places P_i of each module, partitioning P_i into equivalence classes P_i^j. Two places of a module are related by R iff there exists a non-directed path including only nodes from that module that does not include interface transitions. In the example, places in module $\mathcal{N}_1$ are partitioned into four equivalence classes, $P_1^1 = \{\mathsf{a1}\}$, $P_1^2 = \{\mathsf{a2}, \ldots, \mathsf{a5}\}$, $P_1^3 = \{\mathsf{a6}, \ldots, \mathsf{a9}\}$, and $P_1^4 = \{\mathsf{a10}, \ldots, \mathsf{a13}\}$, places in module $\mathcal{N}_2$ are partitioned into $P_2^1 = \{\mathsf{c1}, \mathsf{c2}, \mathsf{c3}, \mathsf{c6}\}$ and $P_2^2 = \{\mathsf{c4}, \mathsf{c5}\}$, while places in $\mathcal{N}_3$ are partitioned into $P_3^1 = \{\mathsf{d1}\}$, $P_3^2 = \{\mathsf{d2}, \ldots, \mathsf{d5}\}$, $P_3^3 = \{\mathsf{d6}, \ldots, \mathsf{d9}\}$, and $P_3^4 = \{\mathsf{d10}, \ldots, \mathsf{d13}\}$. Then, a set H_i^j (standing for "High-level places") of implicit places that include information of the behaviour on $\mathcal{N}_i$ is computed for each equivalence class P_i^j. For instance, in the example, $H_2^1 = \{\mathsf{i3}\}$ is the set (in this case only one place) of implicit places corresponding to the equivalence class $P_2^1 = \{\mathsf{c1}, \mathsf{c2}, \mathsf{c3}, \mathsf{c6}\}$. In some cases, some of the computed implicit places can be omitted. For instance, the implicit places corresponding to the equivalence classes P_1^2, P_1^3, and P_1^4 are identical (place i2), thus only one is added.

The components (low level systems, $\mathcal{LS}_i$, $i = 1, \ldots, K$) are derived by reducing all the modules $\mathcal{N}_j$, $j \neq i$, to their interface transitions and to the implicit places that were added in the extended system, while $\mathcal{N}_i$ is fully preserved. If needed, another component, the *basic skeleton*, $\mathcal{BS}$, can be derived by reducing *all* the modules in the same way as for $\mathcal{LS}_i$. The basic skeleton defines a more abstract view of the original system. Figure 20.7 shows the low level systems, $\mathcal{LS}_1$ $\mathcal{LS}_2$, and $\mathcal{LS}_3$ and the basic skeleton, $\mathcal{BS}$ for the running example. Notice that if $\mathcal{LS}_1$ $\mathcal{LS}_2$ and $\mathcal{LS}_3$ were synchronized by merging common transitions (interface transitions) and identifying common places (buffers and implicit places), the extended system (with equivalent behaviour to the original system) would be obtained. Notice also that the basic skeleton is the common abstraction among the three components.

20.2 A Taxonomy of Techniques

There exist a significant number of techniques proposed in the literature in order to compute performance indices (either as *bounds*, *approximations* or *exact* values), within divide and conquer strategies. In order to provide a framework for their consideration, we shall introduce some criteria inducing a certain taxonomy for the existing techniques. Representative examples of techniques will be briefly overviewed, providing some insight into the stochastic solution.

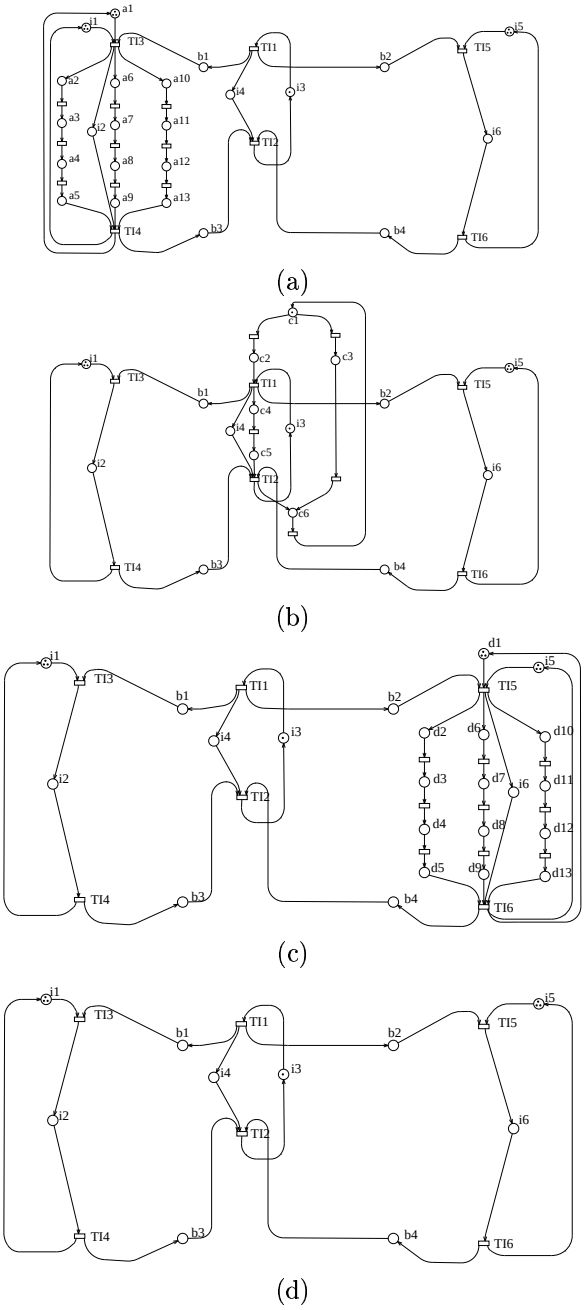


Figure 20.7: Low level systems (a,b,c) and basic skeleton (d) for the model in Figure 20.3.a.

20.2.1 The Criteria for Classification

A first criterium for the taxonomy used later concerns *the information that components have about their environment.* In some cases, the components are directly the modules obtained from the partition, thus they do not have any information of the rest of the system. Examples of this approach are: the exact analysis of SGSPN [19] (see Chapter 12), the flow equivalent aggregation for the approximate analysis of SPN's [20], or the linear programming based computation of performance bounds presented in Section 20.1.2 [7]. In other cases, modules are complemented in order to summarize the possible behaviour of the environment. A particular technique for complementing modules (for the case of the SAM view) was introduced in the previous Section, that has been used in [11] for exact analysis or in [28] for approximate analysis. Other examples where components are complemented modules appeared in [13] for the improvement of bounds or in [26] for approximate analysis.

In divide and conquer strategies there always exists an integration of partial solutions. Therefore, a second criterium for classification may be *the existence or not of an explicit global abstract view of the system* usable in the integration phase. In our case, the existence of a net model providing a global abstract view makes the difference. Examples of techniques in which no high level view of the system is used are: the bounds presented in Section 20.1.2 [7] and their improvement presented in [13], or the approximate technique proposed in [26]. On the other hand, two-level approaches (i.e., techniques that use both low and high level views of the system) for the analysis of SPN's are, for instance, the flow equivalent aggregation for the approximate analysis of SPN's [20], the product-form approximation techniques [1] presented in Chapter 22, the exact analysis of SAM models [11] (Chapter 12), and the approximate analysis of DSSP (Deterministically Synchronized Sequential Processes) [29, 30] or SAM models [28].

Since the structure of the SPN model is defined by a bipartite graph, an additional classification criterium could be *the selection of either places or transitions for the partition of the model into modules.* However, as we stressed in Section 20.1.3, both approaches can be translated one into the other, thus the selection of places or transitions for defining the cut plays a non-substantial (essentially syntactical) rule in the decomposition process.

In addition to the above criteria, all the analysis techniques for performance evaluation of models (and in particular those based on a net-driven decomposition) can be classified according to *the quality of results* into: exact, approximate and bounding techniques. The general cost/accuracy trade-off must be taken into account in the selection of the desired approach.

20.2.2 Isolated Modules and Single-Level Techniques

An example of solution method within a divide and conquer strategy with isolated modules and a single level of abstraction is the technique for computing performance bounds using P-semiflows presented in Section 20.1.2 [7], where

thanks to the LPP expression the decomposition is implicit, transparent to the analyzer.

Another example that makes use of isolated modules and a single level of abstraction is the exact analysis of SGSPN (Superposed Generalized Stochastic Petri Nets) [19] presented in Chapter 12. Here the decomposition must be made explicit. The basic idea behind the SGSPN technique can be explained using the model of Figure 20.2.a, that shows a GSPN $\mathcal{S}$ that can be considered as the composition of two GSPN's $\mathcal{S}_1$ and $\mathcal{S}_2$ (depicted in Figure 20.2.b) over two common transitions a and b. $\mathcal{S}_1$ has therefore a number of states equal to the number of ways in which 2 balls can be distributed into four boxes: 10 states. Analogously for $\mathcal{S}_2$ (4 boxes, 1 ball): 4 states. A product state space PS can then be defined as

$$\mathrm{PS} = \mathrm{RS}_1 \times \mathrm{RS}_2$$

and it is straightforward to observe that $\mathrm{RS} \subseteq \mathrm{PS}$. Note that PS has a number of states equal to the product of the above numbers (40 states), but the reachability set of $\mathcal{S}$ has only 14 states.

According to the techniques presented in [19] the following matrix $\mathbf{G}$ of size $|\mathrm{PS}| \times |\mathrm{PS}|$ can be constructed:

$$\mathbf{G} = \mathbf{Q}'_1 \oplus \mathbf{Q}'_2 \quad \begin{array}{l} -\sum_{t\in\{\mathsf{a},\mathsf{b}\}} w(t)[\mathbf{K}_1(t) \otimes \mathbf{K}_2(t)] \\ +\sum_{t\in\{\mathsf{a},\mathsf{b}\}} w(t)[\mathbf{K}'_1(t) \otimes \mathbf{K}'_2(t)] \end{array} \tag{20.8}$$

where the $\mathbf{Q}'_i$, $\mathbf{K}_i(t)$, and $\mathbf{K}'_i(t)$ (for $i \in \{1,2\}$) are $|\mathrm{RS}_i| \times |\mathrm{RS}_i|$ matrices, that can all be derived from the infinitesimal generator $\mathbf{Q}_i$ of $\mathcal{S}_i$.

The idea behind this formula is to split the behaviour of each component into *local behaviour* (related to transitions local to a single component), and *dependent behaviour* (related to "synchronizing transitions" a and b). The local behaviour of each GSPN is represented by $\mathbf{Q}'_i$, and since the local behaviour is independent, the global behaviour due to local transitions can be obtained as the tensor sum of the $\mathbf{Q}'_i$ matrices. The behaviour related to synchronization requires that, for a synchronization transition to fire, both $\mathcal{S}_i$ must be in a state that enables the transition. $\mathbf{K}_i(t)$, the correcting matrix for transition t, has a 1 in each entry of the matrix that corresponds to a change of state due to t in $\mathcal{S}_i$. The tensor product will indeed realize the required condition that *a synchronization transition fires only in global states whose corresponding local states in the $\mathcal{S}_i$ enable t.* The term with the $\mathbf{K}'_i(t)$ matrices is used to compute the portion of the diagonal elements expression that accounts for synchronization transitions.

By definition of the tensor sum and product, $\mathbf{G}$ is a $|\mathrm{PS}| \times |\mathrm{PS}|$ matrix, and it is shown in [19] how the non null entries of the vector $\boldsymbol{\pi}$, solution of the equation $\boldsymbol{\pi} \cdot \mathbf{G} = \mathbf{0}$, are the steady-state solution of $\mathcal{S}$. Moreover a solution process may be devised that does not require the explicit computation and storing of $\mathbf{G}$, so that the biggest memory requirement is that of the vector $\boldsymbol{\pi}$. The technique is extended to transient analysis in [24]. The computational cost, under full matrix implementation assumption, is smaller than the classical vector to matrix multiplication [31], while recent results have shown [5] that the viceversa is true

for matrices with a mean number of elements per rows less than $K^{\frac{1}{K-1}}$ (K being the number of components) under sparse matrix implementation.

The above technique produces a significant storage saving whenever the size of PS, and therefore that of $\boldsymbol{\pi}$, is inferior to the number of non null elements of $\mathbf{Q}$, since, otherwise, it would be better to store $\mathbf{Q}$ explicitly.

The solution procedure outlined before is the basic idea behind a number of works that have appeared in the literature. The work in [31] defines the basics of the method and applies it to networks of stochastic automata, while classes of SPN's for which a single level structured solution has been applied are Superposed Stochastic Automata [18] and Superposed GSPN (SGSPN) [19] (SGSPN nets that can be interpreted as the superposition over a subset of timed transitions of equal label and rates of a set of GSPN's).

The distance between RS and PS can limit the applicability of the technique, but if a bit vector of size |PS| can be allocated in memory, then a state space exploration can be performed [23], which, with an additional tree-like data structure of size $O(|\mathrm{RS}| \cdot \log |\mathrm{RS}_i|)$ (i is the index of the component whose state space is represented in the tree leaves), allows the definition of multiplication algorithms that consider only reachable states. The computational overhead is at most logarithmic in |RS| [5].

20.2.3 Complemented Modules and Single-Level Techniques

As we remarked in Section 20.1.4, the main idea behind the complementation of modules is to obtain components with a certain abstract view of its environment, the rest of the system. The goal is to find a process that leads to better (or just possible) quantitative evaluation techniques using the composition of partial results computed for the components.

A simple example of complementation of modules was proposed in [13] for the improvement of the linear programming based bounds presented in Section 20.1.2. In that section, bounds for the mean interfiring time of transitions of the modules in complete isolation is computed. A more realistic computation of the mean interfiring time can be considered using the concept of *liveness bound* of transitions. The liveness bound of a transition is the *maximum reentrance* (or maximum self-concurrency) that the net structure and the marking allow for the transition in steady-state. In other words, it gives the number of servers for transition t in steady state. Formally, $L(t) = \max\{k \mid \forall \mathbf{m}' : \mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}', \exists \mathbf{m} : \mathbf{m}' \xrightarrow{\sigma'} \mathbf{m} \wedge \mathbf{m} \geq k\ \mathbf{Pre}[t]\}$. In the case that the above quantity cannot be computed efficiently, an structural counterpart (that can always be computed in an efficient manner) can be considered: the *structural enabling bound* [13].

The technique we are going to briefly present by means of an example is based on the consideration of *embedded product-form closed monoclass queueing networks* of the SPN. From a topological point of view, these embedded networks are *P-components*: strongly connected state machines generated by P-semiflows. An improvement of the lower bound for the mean interfiring time

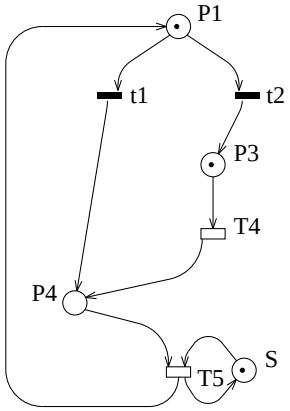


Figure 20.8: Complementation of the module in Figure 20.1.b.

of a transition t_i given by LPP (20.5) can be eventually obtained computing the exact mean interfiring time of that transition in the P-component generated by a minimal P-semiflow $\mathbf{y}$, with $L(t)$–server semantics for each involved transition t (in fact, it is not necessary that t_i belongs to the P-component; the bound for other transition can be computed and then weighted according to the visit ratios in order to compute a bound for t_i). The P-semiflow $\mathbf{y}$ can be selected among the optimal solutions of (20.5) or it can be just a feasible *near-optimal* solution.

Let us consider once again the net system depicted in Figure 20.1.a. The bound computed in (20.7) does not take into account the queueing time at places P4 and P5 due to synchronization (T5). That bound can be improved if the P-component generated by $\mathbf{y_1} = (1, 0, 1, 1, 0)$ (depicted in Figure 20.1.b) is considered with liveness bound of transition T5 reduced to 1 (which is the liveness bound of this transition in the whole net). That information may be introduced in the module of Figure 20.1.b by the addition of a place S that limits the number of servers of T5 using the information of the rest of the system, leading to the component shown in Figure 20.8. Notice that if place S were added in the original system it would be implicit. The system in Figure 20.8 is isomorphous to a product-form queueing network and it can be efficiently solved. The average interfiring time of T5 in this system gives the following improvement of the bound (20.7) for the average interfiring time of the same transition in the original system (Figure 20.1.a):

$$\Gamma[\mathsf{T5}] \geq 4.562502 \tag{20.9}$$

The exact average interfiring time of T5 is 4.978493, therefore by adding S and considering the exact performance of the component in isolation the relative error has been reduced from 19.65% to 8.36%.

Another interesting example of complemented modules using information of the rest of the system may be described by the tensor algebra approach

presented in the previous section for the exact solution of SGSPN models. If the components depicted in Figures 20.4.b and 20.4.c are used in equation (20.8) instead of modules depicted in Figure 20.2.b, the size of matrices $\mathbf{Q}_1'$, $\mathbf{K}_1(t)$, and $\mathbf{K}_1'(t)$ is reduced from 10 to 7, therefore, the difference between PS and RS is reduced. Notice that places q34, p12, and p34 are implicit what means, in particular, that the environment of $\mathcal{S}_2$ (subsystem $\mathcal{S}_1$) does not constraint its behaviour. If the other decomposition depicted in Figure 20.2.c is considered, the tensor algebra approach cannot be directly applied since the state spaces of the modules (thus the size of matrices involved in the tensor expression) are unbounded. Using implicit places to complement the modules, as explained in Section 20.1.4 and depicted in Figures 20.5.b and 20.5.c, the state spaces of the components are limited (finite) to 8 and 5, respectively, and the technique can be applied (with $|\mathrm{PS}| = 40$, while $\mathrm{RS} = 14$).

20.2.4 Isolated Modules and Two-Level Techniques

A classical technique for the analysis of queueing network models that can be also applied to SPN's is *flow equivalent aggregation* (FEA) [14, 25, 20, 22]. The basic approach is to replace a general server or subnetwork of queues by a Flow Equivalent Service Center (FESC). An arriving customer sees the FESC as a black box whose behaviour is completely characterized by a listing of the residence time (the inverse of the throughput) as a function of the possible customer population. In order to determine the state dependent service rates, the subsystem is studied off-line, i.e., without any interaction with the environment. In general, a FESC matches only the first moment of the probability distribution, as the higher moments are too cumbersome to obtain.

Two steps for constructing an FESC can be determined, once a *subsystem* to be analysed by FESC is defined:

1. Analyse the (low level) subsystem by maintaining the number of customers constant. This is done by shorting out all other parts of the network and varying the number of customers up to the maximum allowed in the subsystem.

2. The Aggregated Network (AN) or high level system is constructed as a server with a queue-dependent service rate. Let $\chi_{\mathrm{SW}}(k)$ be the conditional throughput of the subnetwork (SW) when k customers are present, and $\mu_{\mathrm{AN}}(k)$ the (state dependent) service rate of AN. The approximation is done through the following equality, under the assumption of exponential service time: $\chi_{\mathrm{SW}}(k) = \mu_{\mathrm{AN}}(k)$.

As an example, consider the SPN depicted in Figure 20.9. Service times of transitions are the following (single server semantics is assumed, and race policy at conflicts): $\mathbf{s}[\mathsf{Ti}] = 1.0$ for $i \neq 6, 7, 13$; $\mathbf{s}[\mathsf{T6}] = \mathbf{s}[\mathsf{T7}] = 2.0$; $\mathbf{s}[\mathsf{T13}] = 0.5$. Once the subsystem to be aggregated has been selected, the systems depicted in Figure 20.10.a and 20.10.b are derived. The first of them is the isolated subsystem or low level subsystem while the second one is the aggregated network or high level

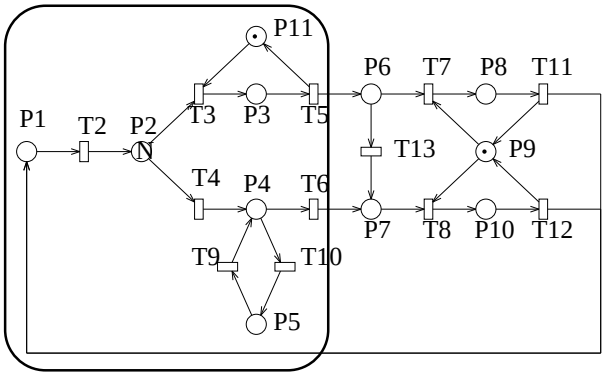
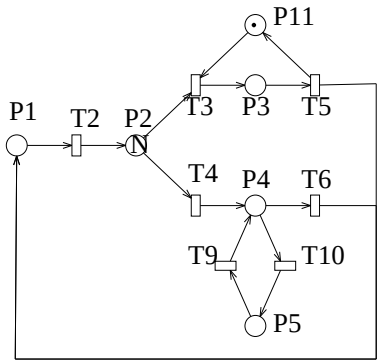
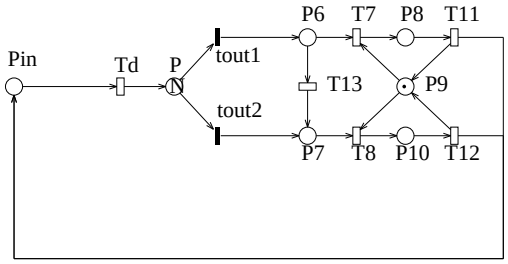


Figure 20.9: An example of stochastic Petri net system with a subsystem to be aggregated.



(a)



(b)

Figure 20.10: (a) Isolated subsystem and (b) aggregated system for the model in Figure 20.9.

N	r_5	r_6	$\chi(\mathsf{T2})$
1	0.500	0.500	0.400
2	0.431	0.569	0.640
3	0.403	0.597	0.780
4	0.389	0.611	0.863
5	0.382	0.618	0.914

Table 20.1: Solution of the isolated subsystem in Figure 20.10.a for different values of N.

N	# states		throughput		% error
	aggreg.	orig.	aggreg.	orig.	
1	5	9	0.232	0.232	0.00
2	12	41	0.381	0.384	0.78
3	22	131	0.470	0.474	0.84
4	35	336	0.521	0.523	0.38
5	51	742	0.548	0.547	< 0.10

Table 20.2: Solution of the aggregated system in Figure 20.10.b and of the original system (Figure 20.9) for different values of N.

system. The subsystem is evaluated in isolation with constant number of tokens varying from 1 to N (Figure 20.10.a). The outcome of the analysis is: (1) the relative throughput (visit ratios) between T5 and T6, and (2) the throughput of the subsystem (more precisely, the throughput of T2), for different values of the initial marking of P2. The obtained results are shown in Table 20.1.

When the subsystem is aggregated (Figure 20.10.b), routing at tout1 and tout2, and delay at Td are state dependent (they depend on $\mathbf{m}[\mathsf{Pin}]$). Table 20.2 shows the results obtained with flow equivalent aggregation for the system in Figure 20.9 for different values of the initial marking at P1 ($N = 1, \ldots, 5$). The exact results obtained by solving the CTMC of the original system and the relative error of the FEA are also included (together with the size of the state spaces of the original system and of the aggregated system).

In FEA, the behaviour of the subnet is assumed to be *independent* of the arrival process and to depend only on the number of customers in the system, i.e., the behaviour is completely independent of the environment. This condition is frequently violated. In [21], it is shown that even in very simple cases the mean completion (or traversing) time of a subnet (i.e., the average time spent by a token traversing the subnet) may depend on the token's interarrival process. This fact happens, for instance, if there exist internal loops in the subnet or if there are trapped tokens in a fork-join inside the subnet; when such situations arise, it becomes necessary to implement a less efficient (usually involving a fixed-point search iterative process) but more accurate approximation technique, like, for instance, response time approximation [22, 21, 8, 26, 29, 30, 28].

There are some cases where FEA leads to exact results. This happens for

the particular case of product-form queueing networks (Norton's theorem) [14], where the mean completion time of the subnet is strictly independent of the token's interarrival process.

20.2.5 Complemented Modules and Two-Level Techniques

In Section 20.2.3 we have illustrated how the use of complemented modules can improve the results obtained using isolated modules. In the particular case of the tensor algebra approach for the exact solution of models, we have seen that the distance between RS and PS can be reduced, including in each module some information (abstract view) of the other. It may be possible that PS includes a number of *spurious states* that are non-reachable in the original system (i.e., they do not belong to RS). To limit the number of spurious states, an abstract description of the system can be used to appropriately pre-select the subsets of the states that should enter in the Cartesian product.

For instance, for both decompositions of Figure 20.2.a (the one in Figures 20.4.b and 20.4.c, and that in Figures 20.5.b and 20.5.c), the basic skeleton $\mathcal{BS}$ (or high level view of the model) is just the cycle a-i12-b-i34 with one token initially in i12. The states of the low level systems $\mathcal{LS}_1$ and $\mathcal{LS}_2$ can be partitioned according to the states of $\mathcal{BS}$: those states with equal high level representation $\mathbf{z}$ (i.e., those whose projection on i12 and i34 is the same) belong to a single class $\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_1)$ ($\mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_2)$) in the state space of $\mathcal{S}_1$ (respectively, $\mathcal{S}_2$). The restricted product state space RPS can then be built as:

$$\mathrm{RPS}(\mathcal{S}) = \biguplus_{\mathbf{z} \in \mathrm{RS}(\mathcal{BS})} \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_1) \times \mathrm{RS}_{\mathbf{z}}(\mathcal{LS}_2)$$

where $\biguplus$ is the *disjoint* set union. Note that for the example in Figure 20.4 we obtain a precise characterization of the state space (i.e., there exist no spurious states; a property that holds in general for live and bounded marked graphs), but the union of Cartesian products can in general produce a superset of the reachable state space, depending on how precise is the abstract representation.

Since the state space is no longer the Cartesian product of sets of local state spaces, but the union of Cartesian products, we cannot expect to have for the infinitesimal generator a tensor expression as simple as before: we can build a matrix $\mathbf{G}$, of size $|\mathrm{RPS}| \times |\mathrm{RPS}|$, and then consider it as block structured according to the states of $\mathcal{BS}$. Each block will thus refer to a set of states obtained by a Cartesian product, and a tensor expression for each block can be derived. Diagonal blocks $\mathbf{G}(\mathbf{z}, \mathbf{z})$ can be expressed as

$$\mathbf{G}(\mathbf{z}, \mathbf{z}) = \mathbf{Q}_1(\mathbf{z}, \mathbf{z}) \oplus \mathbf{Q}_2(\mathbf{z}, \mathbf{z})$$

where the $\mathbf{Q}_i(\mathbf{z}, \mathbf{z})$ are the submatrices of the infinitesimal generator of $\mathcal{LS}_i$ determined by the states whose abstract representation is $\mathbf{z}$. Each $\mathbf{G}(\mathbf{z}, \mathbf{z}')$ block, with $\mathbf{z} \neq \mathbf{z}'$, describes instead the behaviour that changes the high level

	$\mathbf{m}(\mathsf{c1})=1$	$\mathbf{m}(\mathsf{c1})=2$	$\mathbf{m}(\mathsf{c1})=3$
$\mathcal{BS}$	10	46	146
$\mathcal{LS}_1$	199	6.985	108.945
$\mathcal{LS}_2$	199	6.985	108.945
$\mathcal{LS}_3$	22	190	1.032
RPS	8.716	3.872.341	431.270.717
RS	8.716	3.872.341	no memory
PS(case A)	11.426.400	no memory	no memory
PS(case B)	—	198.994.880	no memory

Table 20.3: SAM technique: State spaces computed for the model of Figure 20.3.a.

state; each block $\mathbf{Q}(\mathbf{z},\mathbf{z}')$ can be written as

$$\mathbf{G}(\mathbf{z},\mathbf{z}') = \sum_{t:\mathbf{z}\xrightarrow{t}\mathbf{z}'} w(t)[\mathbf{K}_1'(t)(\mathbf{z},\mathbf{z}') \otimes \mathbf{K}_2'(t)(\mathbf{z},\mathbf{z}')]$$

where the $\mathbf{K}_i'(t)(\mathbf{z},\mathbf{z}')$ are the submatrices of the infinitesimal generator of $\mathcal{LS}_i$ whose rows (columns) are abstractly represented by $\mathbf{z}$ ($\mathbf{z}'$), projected to include only the contribution due to t. Observe that, in our example, there is a single t for each given pair $(\mathbf{z},\mathbf{z}')$.

The above approach that has been applied here to SGSPN, was developed and used in the literature in the context of the solution of "asynchronous systems", SPN that can be seen as modules that interact by exchanging tokens [4, 2, 6, 10, 11]. For the example in Figure 20.3.a, considering the SAM view depicted in Figure 20.7, Table 20.3 shows the sizes of the state space of $\mathcal{LS}_i$ and $\mathcal{BS}$ as well as the size of RPS and RS of the complete system, for an initial marking with three tokens in a1 and d1, and corresponding implicit places, and of $1, 2$, and 3 tokens in place c1.

A straightforward comparison with single-level synchronous decomposition may not be very significant, since the division of the SAM into modules may not be the best one for SGSPN. We have tried two different synchronous decompositions: three components identified by places starting with a or b for the first component, c for the second, and d for the third (case **A**) (derived from modules in Figure 20.3.c by adding some appropriated places [11]), and two components identified by places starting with a or b or d for the first component, and c for the second (case **B**).

The size of the product state space is also shown in Table 20.3 (where — means that the experiment was not performed). For the **A** case it was not possible to increase to 2 the number of tokens in c1, since after the generation of the state spaces of the three components, of size $1.701, 5.161$, and 5.161, the tool stops, which is not surprising considering that, according to the size of the components, PS has a size of about $45 * 10^9$.

The decomposition of case **B** is indeed more favourable, since we were able to solve the $\mathbf{m}(\mathsf{c1}) = 2$ model.

In summary, if we have a construction rule for RS as a disjoint union of Cartesian products, we can limit the problem of the difference $|\mathrm{PS}| - |\mathrm{RS}|$, while still maintaining the benefits of the solution in structured form as in the SGSPN case: indeed it is only necessary to organize the classical vector by matrix multiplication $\boldsymbol{\pi} \cdot \mathbf{G}$ in terms of subvector by submatrix multiplication. Details for the computational costs of this technique under sparse and full matrix storage scheme have been reported in [3].

20.3 Concluding remarks

Divide and conquer is a classical and sometimes successful enough strategy to fight against computational complexity. Using stochastic PN models of concurrent systems we consider several model decomposition and subsequent solution techniques in order to compute performance figures.

Modules, implicit places and components are the basic elements for providing decomposed views of systems. Criteria like the addition of information summarizing the behaviour of the environment of a module or the eventual existence of a global abstract view allows to situate many sparse techniques in a certain conceptual framework. The introduced possibility for going to a two-level view of systems can be generalized by recursively applying the principles, leading to multiple-level views; these have been considered, for example, for approximation techniques [14, 21].

Finally, just to point out that a different use of structure theory (also using implicit places) in order to improve performance bounds is to add to the original net model some implicit places (in relation with some *traps* structures) in such a way that new components are found [9, 12]. If the new components are bottlenecks, the computed bound is improved. As an example, adding a place π to the net in Figure 20.1.a with input transitions T3 and T4 and output transition T5 (with ordinary arcs) and initially marked with zero tokens, an additional minimal P-semiflow is generated (with support $\{\mathsf{P1}, \mathsf{P2}, \mathsf{P3}, \pi\}$), and the application of (20.6) gives now $\Gamma[\mathsf{T5}] \geq \max\{(5+3)/2, 0.5+3, (0.5+5+3)/2\} = 4.25$ thus improving the first bound ($\Gamma[\mathsf{T5}] \geq 4$) given in (20.7). Moreover, if the new component generated by this additional P-semiflow is considered using the technique presented in Section 20.2.3, limiting the liveness bound of transition T5 to 1 (as it was performed in Figure 20.8), the bound is improved to $\Gamma[\mathsf{T5}] \geq 4.779406$ (instead of the value $\Gamma[\mathsf{T5}] \geq 4.562502$ obtained without the addition of the implicit place π, thus the relative error is reduced from 8.36% to 4%). Better quality results are obtained increasing the computational investments.

Bibliography

[1] B. Baynat and Y. Dallery. A product-form approximation method for general closed queueing networks with several classes of customers. *Performance Evaluation*, 24:165–188, 1996.

[2] P. Buchholz. A hierarchical view of GCSPN's and its impact on qualitative and quantitative analysis. *Journal of Parallel and Distributed Computing*, 15(3):207–224, July 1992.

[3] P. Buchholz. Numerical solution methods based on structured descriptions of Markovian models. In G. Balbo and G. Serazzi, editors, *Computer Performance Evaluation. Modeling Techniques and Tools*, pages 251–267. Elsevier, 1992.

[4] P. Buchholz. A class of hierarchical queueing networks and their analysis. *Queueing Systems*, 15:59–80, 1994.

[5] P. Buchholz, G. Ciardo, S Donatelli, and P. Kemper. Complexity of Kronecker operations on sparse matrices with applications to the solution of Markov models. Icase report 97-66, Institute for Computer Applications in Science and Engeneering, Hampton, VA, 1997.

[6] P. Buchholz and P. Kemper. Numerical analyisis of stochastic marked graphs. In *Proc. 6^{th} Intern. Workshop on Petri Nets and Performance Models*, pages 32–41, Durham, NC, USA, October 1995. IEEE-CS Press.

[7] J. Campos, G. Chiola, and M. Silva. Properties and performance bounds for closed free choice synchronized monoclass queueing networks. *IEEE Transactions on Automatic Control*, 36(12):1368–1382, December 1991.

[8] J. Campos, J. M. Colom, H. Jungnitz, and M. Silva. Approximate throughput computation of stochastic marked graphs. *IEEE Transactions on Software Engineering*, 20(7):526–535, July 1994.

[9] J. Campos, J. M. Colom, and M. Silva. Improving throughput upper bounds for net based models of manufacturing systems. In J. C. Gentina and S. G. Tzafestas, editors, *Robotics and Flexible Manufacturing Systems*, pages 281–294. Elsevier Science Publishers B.V. (North-Holland), Amsterdam, The Netherlands, 1992.

[10] J. Campos, S. Donatelli, and M. Silva. Structured solution of stochastic DSSP systems. In *Proceedings of the 7^{th} International Workshop on Petri Nets and Performance Models*, pages 91–100, Saint Malo, France, June 1997. IEEE Computer Society Press.

[11] J. Campos, S. Donatelli, and M. Silva. Structured solution of asynchronously communicating stochastic modules. Research Report RR-98-4, Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, María de Luna 3, 50015 Zaragoza (Spain), April 1998. Submitted paper.

[12] J. Campos and M. Silva. Structural techniques and performance bounds of stochastic Petri net models. In G. Rozenberg, editor, *Advances in Petri Nets 1992*, volume 609 of *Lecture Notes in Computer Science*, pages 352–391. Springer-Verlag, Berlin, 1992.

[13] J. Campos and M. Silva. Embedded product-form queueing networks and the improvement of performance bounds for Petri net systems. *Performance Evaluation*, 18(1):3–19, July 1993.

[14] K. M. Chandy, U. Herzog, and L. Woo. Parametric analysis of queueing networks. *IBM Journal of Res. Develop.*, 19:36–42, January 1975.

[15] J. M. Colom. *Análisis Estructural de Redes de Petri, Programación Lineal y Geometría Convexa.* PhD thesis, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, Spain, June 1989. Research Report. GISI-RR-89-11. In Spanish.

[16] J. M. Colom and M. Silva. Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal p-semiflows. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 79–112. Springer-Verlag, Berlin, 1991.

[17] J. M. Colom and M. Silva. Improving the linearly based characterization of P/T nets. In G. Rozenberg, editor, *Advances in Petri Nets 1990*, volume 483 of *Lecture Notes in Computer Science*, pages 113–145. Springer-Verlag, Berlin, 1991.

[18] S. Donatelli. Superposed stochastic automata: A class of stochastic Petri nets with parallel solution and distributed state space. *Performance Evaluation*, 18:21–36, 1993.

[19] S. Donatelli. Superposed generalized stochastic Petri nets: Definition and efficient solution. In R. Valette, editor, *Application and Theory of Petri Nets 1994*, volume 815 of *Lecture Notes in Computer Science*, pages 258–277. Springer-Verlag, Berlin, 1994.

[20] H. Jungnitz and A. A. Desrochers. Flow equivalent nets for the performance analysis of flexible manufacturing systems. In *Proceedings of the*

1991 IEEE International Conference on Robotics and Automation, pages 122–127, Sacramento, CA, USA, April 1991.

[21] H. Jungnitz, B. Sánchez, and M. Silva. Approximate throughput computation of stochastic marked graphs. *Journal of Parallel and Distributed Computing*, 15:282–295, 1992.

[22] H. J. Jungnitz. *Approximation Methods for Stochastic Petri Nets.* PhD thesis, Dept. of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY, USA, May 1992.

[23] P. Kemper. Numerical analyisis of superposed GSPN. *IEEE Transactions on Software Engineering*, 22(4):615–628, September 1996.

[24] P. Kemper. Transient analysis of superposed GSPNs. In *7-th International Conference on Petri Nets and Performance Models - PNPM97*, pages 101–110. IEEE Computer Society, 1997.

[25] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, Englewood Cliffs, NJ, 1984.

[26] Y. Li and C. M. Woodside. Complete decomposition of stochastic Petri nets representing generalized service networks. *IEEE Transactions on Computers*, 44(8):1031–1046, August 1995.

[27] G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd, editors. *Optimization*, volume 1 of *Handbooks in Operations Research and Management Science.* North-Holland, Amsterdam, 1989.

[28] C. J. Pérez-Jiménez and J. Campos. A response time approximation technique for stochastic general P/T systems. In *Proceedings of the 2nd IMACS International Multiconference on Computational Engineering in Systems Applications (CESA'98)*, Hammamet, Tunisia, April 1998. IEEE Systems, Man and Cybernetics.

[29] C. J. Pérez-Jiménez, J. Campos, and M. Silva. Approximate throughput computation of a class of cooperating sequential processes. In *Proceedings of the Rensselaer's Fifth International Conference on Computer Integrated Manufacturing and Automation Technology (CIMAT'96)*, pages 382–389, Grenoble, France, May 1996.

[30] C. J. Pérez-Jiménez, J. Campos, and M. Silva. On approximate performance evaluation of manufacturing systems modelled with weighted T-systems. In *Proceedings of the IMACS/IEEE-SMC Multiconference on Computational Engineering in Systems Applications (CESA'96)*, pages 201–207, Lille, France, July 1996.

[31] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *Proceedings of the 1985 SIGMETRICS Conference*, pages 147–154, Austin, TX, USA, August 1985. ACM.

[32] M. Silva. *Introducing Petri Nets*, chapter 1. Chapman & Hall, London, 1993.

Chapter 21

Queueing Networks with Blocking

Queueing network models in which buffers have finite capacity are often encountered in various applications such as manufacturing systems and communication networks. Different types of blocking mechanisms have been considered in the literature in particular the so-called blocking-after-service and blocking-before-service mechanisms (see Chapter 1). Queueing networks with finite buffers have received much attention in the past years. Refer to [6, 7, 15] for surveys of results and lists of references. Exact closed form analytical solutions, e.g. product-form solutions, are not in general obtainable for queueing networks with finite buffers. Also, exact numerical techniques can only be used for small size models. As a result, much effort has been devoted to obtaining approximate solutions. Various methods have been proposed that can handle different classes of queueing networks with finite buffers. Among them, the class of tandem queueing networks has received much attention. Some authors assume exponential service time distributions while others consider general service time distributions most often represented as phase type (PH) distributions [13]. Also, some authors consider the case where customers arrive to the network according to a given arrival process while others assume that the first server is saturated, i.e., there is an infinite supply of customers in front of it. A model with an external arrival process can be equivalently transformed into a saturated model if the external arrival process is Poisson; see e.g. [6].

Several different methods have been proposed for analyzing tandem queueing networks with blocking, e.g. [1, 2, 3, 5, 6, 10, 11, 16, 17]. All these methods are based on the decomposition of the original network into a set of subsystems. A comparison and unifying presentation of these methods can be found in [6]. It appears that there are three main approaches. Among them, the so-called symmetrical approach has the following interesting feature: starvation and blocking phenomena play a similar role in the method. A efficient decomposition method based on this symmetrical approach has been proposed independently by Altiok

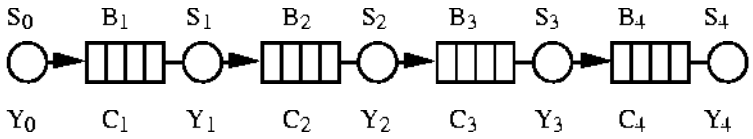


Figure 21.1: The tandem queueing network with finite buffer

and Ranjan [3] and Gün and Makowski [10] in the case of blocking-after-service and by Bouhchouch et al. [5] in the case of blocking-before-service. They consider a saturated model with servers having PH distributions. These methods have proved to perform very well and provide accurate results.

In this chapter, we present the decomposition method proposed in [5] to handle the case of the blocking-before-service mechanism, as an illustration of decomposition methods for queueing networks with blocking. The chapter is organized as follows. In Section 21.1, the model considered in this chapter as well as some relationships between the different blocking mechanisms are presented. A discussion pertaining to the behavior of the blocking-before-service mechanism is then provided. The decomposition method is then presented in Section 21.2.

21.1 Tandem Queueing Networks with Finite Buffers

21.1.1 The Tandem Queueing Network Model

We consider a tandem queueing network consisting of a series of $M+1$ single servers separated by M buffers as illustrated in Figure 21.1. The servers will be denoted by S_i, for $i = 0, ..., M$, and the buffers will be denoted by B_i, for $i = 1, ..., M$. Let C_i denote the capacity of buffer B_i, including the server space in front of server S_i, and let $\mathbf{C} = (C_1, C_2, ..., C_M)$ denote the buffer capacity vector. We assume that the successive service times at each server are independent and identically distributed (i.i.d.) random variables. We denote by Y_i the random variable representing the service time of server S_i. We assume that the first server is never starved and that the last server is never blocked. That is, we consider a saturated model.

Due to the finiteness of the buffers blocking may occur. Various blocking mechanisms have been considered in the literature. The most commonly used are blocking-after-service (BAS) and blocking-before-service (BBS). In blocking-after-service, upon completion of its service at server S_i a customer attemps to enter buffer B_{i+1}. If at that moment this buffer is full, the customer cannot be transfered until a space becomes available in buffer B_{i+1}. During this period of time server S_i is blocked, i.e., it can not serve any customer. In blocking- before-service, server S_i is allowed to start processing a customer only if there is a space available in buffer B_{i+1}. If buffer B_{i+1} is full, then server S_i is blocked and it will be allowed to start processing a customer when a space becomes available in

buffer B_{i+1}. Blocking-before-service is further classified according to whether the service area may be occupied or not while the server is blocked. These two cases are referred as: BBS-SO (blocking-before-service with service area occupied while the server is blocked) and BBS-SNO (blocking-before-service with service area non-occupied while the server is blocked) [15].

Let us briefly discuss the relationships between the different blocking mechanisms. First, there is an equivalence between BAS and BBS-SNO mechanisms [14, 8]. Let $\mathbf{1} = (1, 1, ..., 1)$ denote the unit vector. Consider a tandem queueing network with general service time distributions having buffer capacity vector $\mathbf{C}$ and operating under the BAS blocking mechanism. Consider the same tandem queueing network but with buffer capacity vector $\mathbf{C} + \mathbf{1}$ and operating under the BBS-SNO blocking mechanism. Then, it is established in [8] that these two networks have exactly the same behavior (in terms of sample paths). The performance parameters of one can easily be derived from those of the other. In particular, they have the same throughput.

There is no such equivalence between the BAS (or BBS-SNO) and BBS-SO mechanisms except in the case of a two server one buffer system, i.e., $M = 1$. Indeed, in that case, BBS-SO and BBS-SNO mechanisms are equivalent. Therefore, a two server one buffer system with capacity C and BAS mechanism is equivalent to a two server one buffer system with capacity $C + 1$ and BBS-SO mechanism. In the general case, i.e., $M > 1$, a relationship between the throughput of a tandem queueing network with BAS and BBS-SO mechanisms can be established. Let $X(\mathbf{C}, BAS)$ and $X(\mathbf{C}, BBS - SO)$ denote the throughputs of a tandem queueing network with general service time distributions and buffer capacity vector $\mathbf{C}$, operating under BAS and BBS-SO mechanisms, respectively. The following result is proved in [8]:

$$X(\mathbf{C}, BBS - SO) \leq X(\mathbf{C}, BAS) \leq X(\mathbf{C} + \mathbf{1}, BBS - SO) \leq X(\mathbf{C} + \mathbf{1}, BAS) \tag{21.1}$$

As a result of the above discussion, it appears that there is no need to develop specific methods for the case of BBS-SNO because of the equivalence with the BAS mechanism for which. In this chapter, we focus on tandem queueing networks with finite buffer under the BBS-SO blocking and present the decomposition method proposed in [5]. A similar decomposition method for the case of BAS can be found in [3, 10] Since in the rest of the chapter we only consider the BBS-SO blocking mechanism, we simply refer to it as blocking-before-service (BBS).

21.1.2 Behavior of the Blocking-Before-Service Mechanism

Before presenting the approximation method in Section 21.2, let us further investigate the behavior of tandem queueing networks with BBS mechanism. Let us first define starvation and blocking phenomena. Server S_i is said to be *starved* whenever buffer B_i is empty, that is there is no customers for the server to work on. Server S_i is said to be *blocked* whenever buffer B_{i+1} is full. According to the

definition of the BBS mechanism, the transfer instant of a customer into buffer B_{i+1} is exactly the same as the processing completion instant of this customer by server S_i. Let us consider the time between the processing completion instants of two successive customers, say x and $x+1$, at server S_i. Equivalently, it is also the time between the transfer instants of customers x and $x+1$ into buffer B_{i+1}. After transfer of customer x into buffer B_{i+1}, server S_i may be in four different situations: 1) neither blocked nor starved; 2) blocked but not starved; 3) starved but not blocked; 4) simultaneously starved and blocked. Let us consider these four situations.

- **Situation 1.** Server S_i is neither blocked nor starved at the processing completion instant of customer x. The server begins immediately the service of customer $x+1$. Therefore, the time between the transfer instants of customers x and $x+1$ is just the processing time of customer $x+1$ by server S_i, which is characterized by the random variable Y_i.

- **Situation 2.** Server S_i is blocked but not starved at the processing completion instant of customer x. Customer $x+1$ is already present in buffer B_i waiting for receiving service. However, server S_i cannot start its processing because buffer B_{i+1} is full. This will last until there is a space available in buffer B_{i+1}. Thus, the time between the transfer instants of customers x and $x+1$ is composed of a blocking time followed by a processing time of server S_i. The blocking time corresponds to the time to the next processing completion of server S_{i+1} at the processing completion instant of customer x at server S_i. This time is just the residual processing time of server S_{i+1} if server S_{i+1} is not blocked at the processing completion instant of customer x at server S_i. Otherwise, it is the sum of the time for server S_{i+1} to become unblocked plus the time for server S_{i+1} to process one customer.

- **Situation 3.** Server S_i is starved but not blocked at the processing completion instant of customer x. There is a space available in buffer B_{i+1}. However, customer $x+1$ is not yet present in buffer B_i and thus server S_i cannot start working. This will last until customer $x+1$ arrives into buffer B_i. Thus, the time between the transfer instants of customers x and $x+1$ is composed of a starvation time followed by a processing time of server S_i. The starvation time corresponds to the time to the next processing completion of server S_{i-1} at the processing completion instant of customer x at server S_i. This time is just the residual processing time of server S_{i-1} if server S_{i-1} is not blocked at the processing completion instant of customer x at server S_i. Otherwise, it is the sum of the time for server S_{i-1} to become unblocked plus the time for server S_{i-1} to process one customer.

- **Situation 4.** (Figure 2). Server S_i is starved and blocked at the processing completion instant of customer x. That is, customer $x+1$ is not yet present in buffer B_i and buffer B_{i+1} is full. The processing of customer

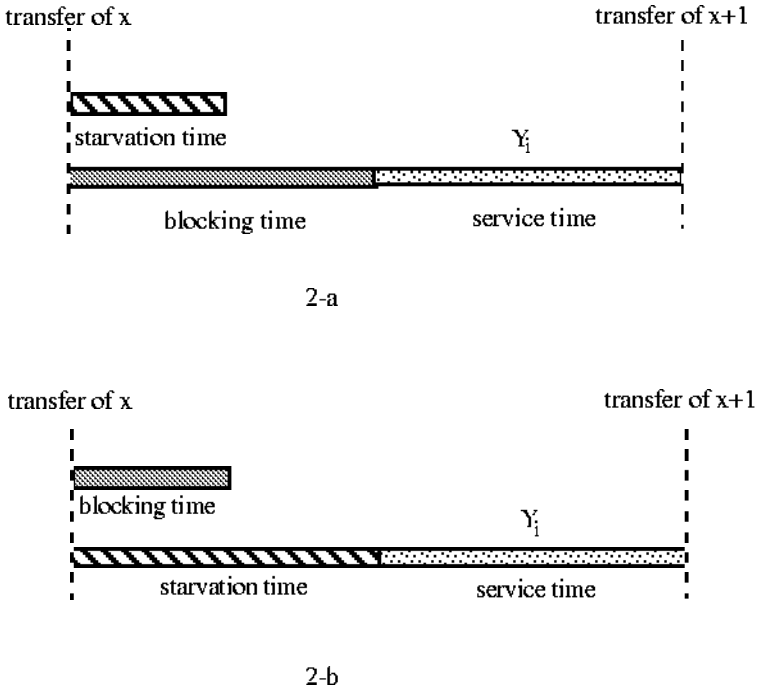


Figure 21.2: Time between transfer instants of customers x and $x+1$ (situation 4)

$x+1$ will begin when both the starvation and the blocking are over. Thus, the time between the transfer of customers x and $x+1$ is composed of the maximum between the starvation time and the blocking time followed by a service time of S_i. Two cases may happen. In the first case, the blocking time is larger than the starvation time (Fig. 2.a). In this case server S_i will still be blocked when customer $x+1$ arrives into buffer B_i. The processing of customer $x+1$ is then delayed by the residual blocking time at the end of starvation. In the second case, the starvation time is larger than the blocking time (Fig. 2.b). In this case the blocking of server S_i has not delayed the processing of customer $x+1$.

This fourth situation is an essential feature of the BBS mechanism: a server can be blocked and starved simultaneously. In BBS mechanism, the blocking of a server can affect the beginning of the processing of the next customer either totally (Situation 2), partially (Figure 21.2a), or not at all (Figure 21.2b). The aim of this chapter is to present an approximation that captures this feature.

Finally, let us discuss some important properties of tandem queueing networks, namely reversibility and duality. The reverse of a tandem queueing network is the network obtained by reversing the flow of customers. The so-called reversibility property states that the reverse network has the same throughput

as the original network. This property holds for the BAS mechanism [12] as well as for the BBS mechanism [8]. However, in the case of BBS, a stronger property so-called duality also holds [4, 8]. It states that the behavior of the reverse network is equivalent (in terms of sample-paths) to that of the original network, provided that the customers of the reverse network are interpreted as the holes (empty space) of the original network, and vice-versa. As a result, there is a one to one correspondance between the performance parameters of the two networks. In particular, the probability of starvation (resp. blocking) of a server in the reverse network is equal to the probability of blocking (resp. starvation) of the corresponding server in the original network.

21.2 The Decomposition Method

In this section, we present the approximation method. The decomposition principle is presented in Section 21.2.1. In Section 21.2.2, we focus our attention to a special case, namely the case of a three server network with exponential service times. This will allow us to present the basic ideas. In Section 21.2.3, we show how this extends to the general case. The computational algorithm is presented Section 21.2.4.

21.2.1 The Decomposition Principle

Consider the tandem queueing network model introduced in Section 21.1.1. As in [3, 10], the method is based on the decomposition of the original network consisting of $M+1$ servers into M subsystems (see Figure 21.3). Each subsystem $T(i)$, for $i = 1, ..., M$ is composed of an upstream server $S_u(i)$ and a downstream server $S_d(i)$ separated by a finite buffer $B(i)$. The first server, $S_u(i)$, is never starved and the second server, $S_d(i)$, is never blocked. The service times of server $S_u(i)$ and $S_d(i)$ are assumed to be independent and identically distributed (i.i.d.) random variables denoted by $Y_u(i)$ and $Y_d(i)$, respectively. Subsystem $T(i)$ approximates the flow of customers in buffer B_i of the original system. The intermediate buffer $B(i)$ has the same capacity C_i as that of buffer B_i. Server $S_u(i)$ represents the part of the original model upstream of buffer B_i while server $S_d(i)$ represents the part of the original model downstream of buffer B_i. Each subsystem $T(i)$ operates under the BBS blocking mechanism.

The goal of the method is to determine servers $S_u(i)$ and $S_d(i)$ in such a way that the flows into and out of buffer $B(i)$ in subsystem $T(i)$ closely match those of buffer B_i in system T, for all subsystems $T(i)$, $i = 1, ..., M$. In this decomposition method, the processing time distributions of the upstream and downstream servers of the subsystems are characterized by PH distributions.

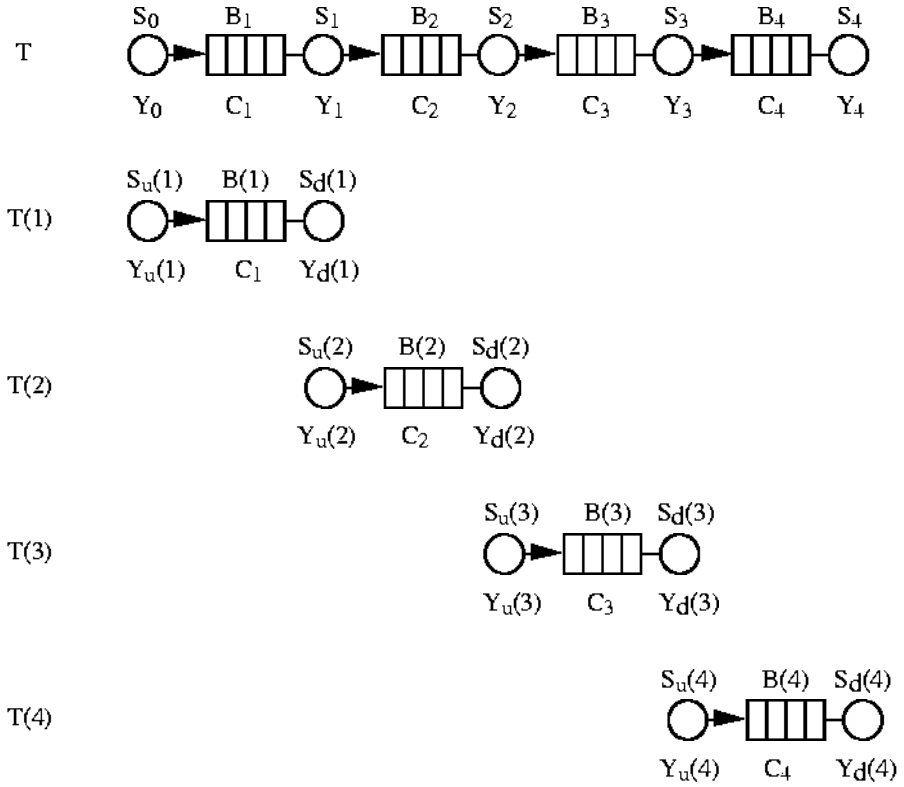


Figure 21.3: Decomposition of the tandem queueing network

21.2.2 A Three Server Network with Exponential Service Times

Let us first focus on a special case, namely a tandem queueing network consisting of 3 servers ($M = 2$). Furthermore, we assume that the random variables Y_i, $i = 0, 1, 2$, are exponentially distributed. Let μ_i be the service rate of server S_i, $i = 0, 1, 2$. According to the general principle of the method presented above, this system is decomposed into two subsystems: $T(1)$ and $T(2)$. We need to characterize the behavior of each of these two subsystems and in particular characterize the service processes of the upstream and the downstream servers of each subsystem, that is characterize the random variables $Y_u(1)$, $Y_d(1)$, $Y_u(2)$ and $Y_d(2)$. First, as usual in decomposition methods, we have boundary conditions. Indeed, $S_u(1)$ represents the part of the original model upstream of buffer B_1 which consists only of server S_0. As a result, the service times of $S_u(1)$ should be identical to those of S_0 and therefore $Y_u(1) = Y_0$. Similarly, $S_d(2)$ represents the part of the original model downstream of buffer B_2 which consists only of server S_2. As a result, the service times of $S_d(2)$ should be identical to those of S_2 and therefore $Y_d(2) = Y_2$. In other words the service times of servers $S_u(1)$ and $S_d(2)$ are simply exponentially distributed with rates μ_0 and μ_2, respectively.

Let us first consider the characterization of the service process of the downstream server of subsystem $T(1)$, that is $S_d(1)$. Server $S_d(1)$ represents (in an aggregate way) the part of the network downstream of buffer B_1, that is: the subnetwork consisting of server S_1, buffer B_2 and server S_2. Thus, the service process of server $S_d(1)$ must represent the effect of this subnetwork onto the behavior of the system. Therefore, it must reflect the service process of server S_1 as well as the possible blocking effect of server S_1 by server S_2. Ideally, the service process of server $S_d(1)$ should be such that the time between the departures of two successive customers from buffer $B(1)$ (that is from subsystem $T(1)$), is the same as the time between the instants of transfer of two successive customers into buffer B_2 in the original system. Now, we have seen in Section 21.1.2 that this time consists of a service time of server S_1 possibly preceded by either a starvation time and/or a blocking time. This should be reflected in subsystem $T(1)$. The starvation phenomena is captured in an explicit way in subsystem $T(1)$ since the part of the system upstream of server S_1 is actually represented in $T(1)$. Therefore what server $S_d(1)$ has to capture is the service time of server S_1 and the possible blocking time.

Thus, the service process of server $S_d(1)$ can be decomposed into two tasks that are performed successively. The first task represents the possible blocking time. It must be performed only if blocking occurs and its length must correspond to the time during which the server is blocked. The probability that this task need to be performed is approximated by the probability that buffer $B(2)$ becomes full at the service completion instant of server $S_u(2)$ in subsystem $T(2)$. Let $p_b(2)$ denote this probability. The blocking time is approximated by the residual service time of server $S_d(2)$ at the blocking instant of server $S_u(2)$ in subsystem $T(2)$. Since the service time of $S_d(2)$ is exponentially distributed,

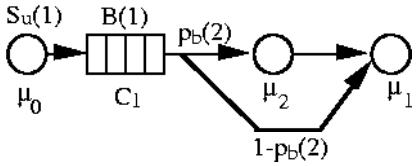


Figure 21.4: Subsystem $T(1)$ in the case of a three server network with exponential distributions

this residual service time is exponentially distributed with the same rate, that is μ_2. The second task simply corresponds to the actual service process of a customer at server S_1. Therefore, its associated service time is characterized by the random variable Y_1, which in the special case considered here has an exponential distribution with rate μ_1. Thus, the service process of server $S_d(1)$ can be represented by a two stage PH distribution as illustrated in Figure 21.4.

Now, as stated in Section 21.1.2, an important feature of the BBS mechanism is that starvation and blocking may occur simultaneously. This is the fourth situation we considered. In this case, the next operation of the server can only begin when both starvation and blocking are over. The idle time of the server is then the maximum of the starvation time and the blocking time. Starvation is over as soon as a service completion occurs at server S_0. Blocking is over as soon as a service completion occurs at server S_2. What is important to notice is that these two activities, namely bringing a new customer into buffer B_1 and creating a new space into buffer B_2, are performed simultaneously and not one after the other. This should be reflected in subsystem $T(1)$. This implies that the first part of the service process of server $S_d(1)$, i.e., the task of exponential length μ_2, must be performed even when buffer $B(1)$ is empty, that is when server $S_d(1)$ is starved. This is an important feature of this decomposition method which is not encountered in decomposition methods dealing with BAS mechanisms.

We note that due to the above feature, namely that the first task of server $S_d(1)$ is performed even though server $S_d(1)$ is starved, the behavior of subsystem $T(1)$ differs from that of the subsystems usually encountered in decomposition methods. In order to illustrate that, consider the case where buffer B_1 has capacity $C_1 = 3$. The markov chain associated to subsystem $T(1)$ is given Figure 21.5. The markovian state is (n, j) where n is the number of customers in buffer $B(1)$ (including that currently receiving service, if any) and j is the stage of the PH distribution of server $S_d(1)$, $j = 1, 2$. The special feature of this Markov chain is that it can be in two different states when buffer $B(1)$ is empty. State $(0, 1)$ corresponds to the state where server $S_d(1)$ is performing the first task, although buffer $B(1)$ is empty. State $(0, 2)$ corresponds to the state where server $S_d(1)$ is waiting for a customer to arrive in buffer $B(1)$ to begin to perform its second task.

Let us now consider the characterization of the service process of the upstream server of subsystem $T(2)$, that is $S_u(2)$. As will appear, the following discussion is very similar to that pertaining to the characterization of server

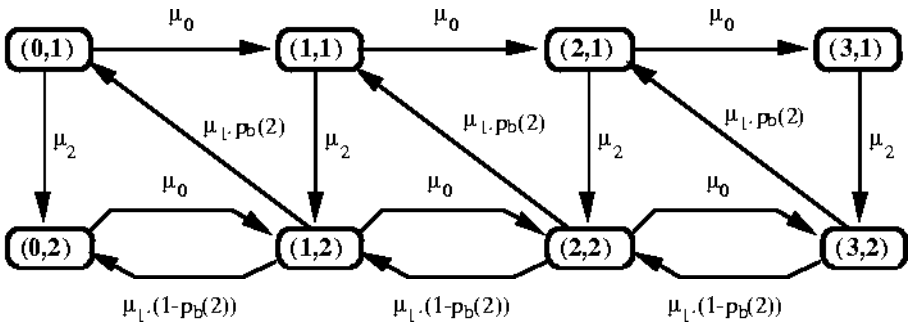


Figure 21.5: Markov chain of subsystem $T(1)$ ($C_1 = 3$)

$S_d(1)$. As a result, it will not be as detailed. Server $S_u(2)$ represents (in an aggregate way) the part of the network upstream of buffer B_2, that is: the sub-network consisting of server S_0, buffer B_1 and server S_1. Ideally, the service process of server $S_u(2)$ should be such that the time between the arrivals of two successive customers into buffer $B(2)$ is the same as the time between the instants of transfer of two successive customers into buffer B_2 in the original system. Again, this time consists of a service time of server S_1 possibly preceded by either a starvation time and/or a blocking time. The blocking phenomena is captured in an explicit way in subsystem $T(2)$ since the part of the system downstream of server S_1 is actually represented in $T(2)$. Therefore what server $S_u(2)$ has to capture is the service time of server S_1 and the possible starvation time.

Thus, the service process of server $S_u(2)$ can be decomposed into two tasks that are performed successively. The first task represents the possible starvation time. The probability that this task need to be performed is approximated by the probability that buffer $B(1)$ becomes empty at the service completion instant of server $S_d(1)$ in subsystem $T(1)$. Let $p_s(1)$ denote this probability. The starvation time is approximated by the residual service time of server $S_u(1)$ at the starvation instant of server $S_d(1)$ in subsystem $T(1)$. Since the service time of $S_u(1)$ is exponentially distributed, this residual service time is exponentially distributed with the same rate, that is μ_0. The second task simply corresponds to the actual service process of a customer at server S_1. Therefore, its associated service time is characterized by the random variable Y_1, which in the special case considered here has an exponential distribution with rate μ_1. Thus, the service process of server $S_u(2)$ can be represented by a two stage PH distribution.

Now, by using similar arguments as for the case of server $S_d(1)$, it appears that the first part of the service process of server $S_u(2)$, i.e., the task of exponential length μ_0, must be performed even when buffer $B(2)$ is full, that is when server $S_u(2)$ is blocked. Let us again illustrate this by considering the case where buffer B_2 has capacity $C_2 = 3$. A markov chain can be associated to subsystem $T(2)$. The markovian state is (n, j) where n is the number of customers in buffer $B(2)$ (including that currently receiving service, if any) and j

is the stage of the PH distribution of server $S_u(2)$, $j = 1, 2$. The special feature of this Markov chain is that it can be in two different states when buffer $B(2)$ is full. State $(3, 1)$ corresponds to the state where server $S_u(2)$ is performing the first task, although buffer $B(2)$ is full. State $(3, 2)$ corresponds to the state where server $S_u(2)$ is waiting for a customer to arrive in buffer $B(2)$ to begin to perform its second task.

21.2.3 The General Case

Consider again the tandem queueing network consisting of $M+1$ servers and M buffers as defined in Section 21.1.1 and its decomposition into M subsystems as presented in Section 21.2.1. Consider a particular subsystem, say $T(i)$. Recall that server $S_u(i)$ represents the part of the original model upstream of buffer B_i while server $S_d(i)$ represents the part of the original model downstream of buffer B_i. We need to generalize the analysis presented in Section 21.2.2 to handle the facts that: 1) service time distributions are of PH type; 2) except for subsystem $T(1)$, there is more than one machine upstream of buffer B_i; except for subsystem $T(M)$, there is more than one machine downstream of buffer B_i. First, we again have the following boundary conditions: the distribution of $Y_u(1)$ is identical to that of Y_0 (S_0 is never starved) and the distribution of $Y_d(M)$ is the same as that of Y_M (S_M is never blocked).

Let us consider the characterization of the service process of server $S_d(i)$, for any $i < M$. By generalizing the arguments given in Section 21.2.2, it appears that the service process of server $S_d(i)$ can be decomposed into two tasks that are performed successively. The first task represents the possible blocking time of server S_i. It must be performed only if blocking occurs and its length must correspond to the time during which the server is blocked. The probability that this task need to be performed is approximated by the probability that buffer $B(i+1)$ becomes full at the service completion instant of server $S_u(i+1)$ in subsystem $T(i+1)$. Let $p_b(i+1)$ denote this probability. The blocking time is approximated by the residual service time of server $S_d(i+1)$ at the blocking instant of server $S_u(i+1)$ in subsystem $T(i+1)$. (This is because server $S_d(i+1)$ represents the part of the system downstream of buffer B_{i+1}.) Let $Y_d^r(i+1)$ denote the random variable associated with this residual service time. The second task simply corresponds to the actual service process of a customer at server S_i. Therefore, its associated service time is characterized by the random variable Y_i.

Consider now the characterization of the service process of server $S_u(i)$, for any $i > 1$. In a similar way, the service process of server $S_u(i)$ can be decomposed into two tasks that are performed successively. The first task represents the possible starvation time of server S_{i-1}. It must be performed only if starvation occurs and its length must correspond to the time during which the server is starved. The probability that this task need to be performed is approximated by the probability that buffer $B(i-1)$ becomes empty at the service completion instant of server $S_d(i-1)$ in subsystem $T(i-1)$. Let $p_s(i-1)$ denote this probability. The starvation time is approximated by the residual service time of

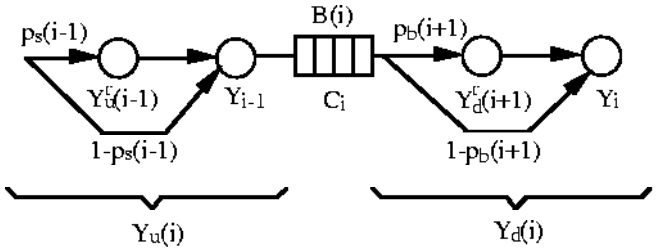


Figure 21.6: Subsystem $T(i)$ in the general case

server $S_u(i-1)$ at the starvation instant of server $S_d(i-1)$ in subsystem $T(i-1)$. This is because server $S_u(i-1)$ represents the part of the system upstream of buffer B_{i-1}. Let $Y_u^r(i-1)$ denote the random variable associated with this residual service time. The second task simply corresponds to the actual service process of a customer at server S_{i-1}. The resulting subsystem $T(i)$ is illustrated in Figure 21.6.

Again, the important feature that a server can be starved and blocked simultaneously (situation 4 described in Section 21.1.2) has to be captured in the decomposition. By generalizing the arguments of Section 21.2.2, it appears that the first task of server $S_u(i)$ should be performed (if required, accordingly to the probability $p_s(i-1)$), even when buffer $B(i)$ is full. The second task however, cannot begin before there is a space available in buffer $B(i)$. Similarly, the first task of server $S_d(i)$ should be performed (if required, accordingly to the probability $p_b(i+1)$), even when buffer $B(i)$ is empty. The second task however, cannot begin before there is a customer in buffer $B(i)$.

21.2.4 The Computational Procedure

In order to use the decomposition method presented above we need to be able: 1) to analyze each subsystem in isolation provided that the characterizations of its upstream and downstream servers are given; and 2) to determine the characterizations of the upstream and downstream servers of each subsystem.

Let us first address the first issue. Consider subsystem $T(i)$. Suppose the characterizations of its upstream and downstream servers are given, i.e., the probabilities $p_s(i-1)$ and $p_b(i+1)$ as well as the distributions $Y_u^r(i-1)$ and $Y_d^r(i+1)$ given under a phase type representation. Then, subsystem $T(i)$ can be analyzed by solving its underlying Markov chain. All performance parameters of interest can then be derived from the stationary probabilities of the Markov chain. The solution of the Markov chain can be obtained using any appropriate technique. However, it is important to recognize that the Markov chain has a special structure, namely a bloc tridiagonal structure. This is also the case in the method of Altiok and Ranjan [3] and Gün and Makowski [10]. The Markov chain however differs from that encoutered in their method because of the special feature pertaining to the simultaneous starvation and blocking phenomena (see

Sections 21.2.2 and 21.2.3). Gün and Makowski [9] used a matrix geometric technique [13]. An alternative technique that can be used in the case of bloc tridiagonal structures is the recursive technique.

Let us now address the second issue pertaining to the determination of the characterizations of the upstream and downstream servers of each subsystem. As it appears from Section 21.2.3, the characterization of the service process of server $S_u(i)$ depends on quantities pertaining to subsystem $T(i-1)$. These quantities, namely the starvation probability $p_s(i-1)$ and the residual service time distribution $Y_u^r(i-1)$, given by its PH representation, can easily be derived from the stationary probabilities of the Markov chain associated with subsystem $T(i-1)$. In a similar way, the characterization of the service process of server $S_d(i)$ depends on quantities pertaining to subsystem $T(i+1)$. These quantities, namely the blocking probability $p_b(i+1)$ and the residual service time distribution $Y_d^r(i+1)$, given by its PH representation, can again be derived from the stationary probabilities of the Markov chain associated with subsystem $T(i+1)$. Details can be found in [5].

Therefore, as usual in decomposition methods [7, 6], it turns out that the parameters of each subsystem $T(i)$ (required in order to characterize its upstream and downstream servers) depend on quantities pertaining to its neighbors: subsystems $T(i-1)$ and $T(i+1)$. As a result, an iterative procedure must be used to determine the unknown parameters. This procedure is similar to that used in other decomposition methods based on a symmetrical approach [3, 7, 6, 10]. The iterative algorithm consists of a forward pass (Step 1) and a backward pass (Step 2). The forward pass calculates new estimates of the service process of each upstream server $S_u(i)$, for $i = 2, ..., M$, while the backward pass calculates new estimates of the service process of each downstream server $S_d(i)$, for $i = M-1, ..., 1$. The iterative procedure is stopped when the parameters of all the servers have converged.

Computational Algorithm

- Step 0. Set $Y_u(1) = Y_0$ and $Y_d(M) = Y_M$. Initialize $Y_d(i) = Y_i$, for $i = 1, ..., M-1$.
- Step 1. For $i = 1, 2, ..., M-1$:
  - Step 1.1. Solve the Markov chain associated with subsystem $T(i)$.
  - Step 1.2. Obtain the starvation probability $p_s(i)$ and the PH distribution $Y_u^r(i)$.
- Step 2. For $i = M, M-1, ..., 2$:
  - Step 2.1. Solve the Markov chain associated with subsystem $T(i)$.
  - Step 2.2. Obtain the blocking probability $p_b(i)$ and the PH distribution $Y_d^r(i)$
- Go to Step 1 until convergence.

An important property of tandem queueing networks is the so-called conservation of flow [7]. It states that the throughputs of all the servers are identical. It is thus important that a decomposition method satisfies this property as well in the sense that all the subsystems should have the same throughput. As pointed out in [6], the symmetrical approach does not involve an equation that prescribes in an explicit way that this is true. However, it is shown in [6] that, in the case of tandem queueing networks with general service times and BAS mechanism, this property is satisfied by the symmetrical methods. This property also holds true in the case of BBS mechanism [5]

Bibliography

[1] T. Altiok, "Approximate analysis of exponential tandem queues with blocking, *Euro. J. Oper. Res.* **11**, pp. 390-398, 1982.

[2] T. Altiok, " Approximate analysis of queues in series with phase-type service times and blocking, *Operations Research* **37**, pp. 601-610, 1989.

[3] T. Altiok and R. Ranjan, " Analysis of production lines with general service times and finite buffers: a two-node decomposition approach, Technical Report, Industrial Engineering Dept., Rutgers University, 1987.

[4] M. H. Ammar, S. B. Gershwin, "Equivalence Relations in Queuing Models of Fork/Join Queueing Networks with Blocking", *Performance Evaluation* **10**, pp. 233-245, 1989.

[5] A. Bouhchouch, Y. Frein, Y. Dallery " A decomposition method for the analysis of tandem queueing networks with blocking before service ", *Queueing Networks with Finite Capacity"*, (ed. R.O. Onvural and I.F. Akyildiz), Elsevier Science Publishers, 1993.

[6] Y. Dallery, Y. Frein, " On decomposition methods for tandem queueing networks with blocking, *Operations Research* , **41**, 2, pp. 386-399, 1993.

[7] Y. Dallery, S. B. Gershwin, "Manufacturing Flow Line Systems: a Review of Models and Analytical Results," *Queueing Systems Theory and Applications (QUESTA)*, **12**, 3-94, 1992.

[8] Y. Dallery, Z. Liu, D. Towsley, "Properties of Fork/Join Queueing Networks with Blocking under Various Operating Mechanisms," *IEEE Transactions on Robotics and Automation*, **13**, 4, pp. 503-518, 1997.

[9] L. Gün, A. M. Makowski, "Matrix-geometric solution for finite queues with phase-type distribution", *Performance'87*, (eds. P. Courtois and G. Latouche), North-Holland, Amsterdam, pp. 269-282, 1987.

[10] L. Gün, A. M. Makowski, "An approximation method for general tandem queueing systems subject to blocking", *Proc. First International Workshop on Queueing networks with blocking*, (eds. H. Perros and T. Altiok), North-Holland, Amsterdam), pp. 147-171, 1989.

[11] K. P. Jun, H. G. Perros, "An approximate analysis of open tandem queueing networks with blocking and general service times", *European Journal of Operational Research*, **46**, pp. 123-135, 1990.

[12] E. J. Muth, "The Reversibility Property of Production Lines", *Management Science*, **25**, pp. 152-158, 1979.

[13] M. F. Neuts, "Matrix-geometric solutions in stochastic models: an algorithmic approach", The John Hopkins Univ. Press, Baltimore, 1981.

[14] R. O. Onvural, H. G. Perros, "On equivalencies of blocking mechanisms in queueing networks with blocking", *Oper. Res. Letters*, **5**, pp. 293 297, 1986.

[15] H. G. Perros, "Open Queueing Networks with Blocking", *Stochastic Analysis of Computer and Communication Systems*, (ed. H. Takagi), North-Holland, Amsterdam, Netherlands, 1989.

[16] , H. G. Perros, T. Altiok, T., "An Approximate analysis of open networks of queues with blocking: tandem configurations", *IEEE Trans. Soft. Eng.* **12**, pp. 450-461, 1986.

[17] Y. Takahashi, H. Miyahara, T. and Hasegawa, "An approximation method for open restricted queueing networks", *Operations Research* **28**, pp. 594-602, 1980.

Chapter 22

Product-form approximation methods for general closed queueing networks

22.1 Basic single-class technique

In this section, we present the general principle of the approximation technique on a very restricted class of single-class closed queueing networks. Let us consider a closed queueing network consisting of M stations and N customers. Each station of this network has an infinite capacity (or equivalently a capacity greater than the population N of the network), and a single server, i.e. $K_i = \infty$ and $C_i = 1, i = 1, \ldots, M$. The scheduling discipline of each queue is FIFO. Let t_i be the mean service-time of station i. We assume that the service-times are i.i.d. random variables. Finally we assume that the routing of the customers through the network is probabilistic. Let $r_{i,j}$ be the probability for a customer completing a service at station i to go to station j. An illustrative example is given in Figure 22.1, where $M = 4, r_{2,3} = r_{2,4} = r_{3,1} = r_{3,2} = 0.5$ and $r_{1,3} = r_{4,2} = 1$. This example will be referred to as Example 1.

22.1.1 Single-class product-form networks

If, in addition to the previous assumptions, every station of the network has an exponential service-time distribution with state-independent rates $\mu_i = \frac{1}{t_i}$, the steady-state probabilities $p(\mathbf{n}) = p(n_1, \ldots, n_M)$ of having at a long-term run n_i customers in station $i, i = 1, \ldots, M$, are known to have the following product-form solution [8, 17, 10]:

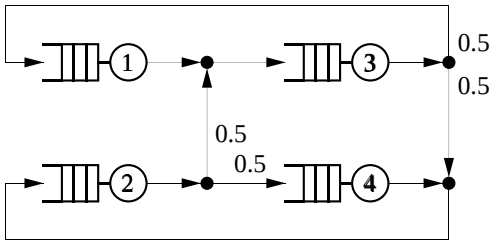


Figure 22.1: Example 1.

$$p(\mathbf{n}) = p(n_1, \ldots, n_M) = \frac{1}{G(M,N)} \prod_{i=1}^{M} \left[\frac{v_i}{\mu_i}\right]^{n_i} \tag{22.1}$$

where v_i is the mean visit ratio of station i and $G(M,N)$ is a normalization constant. The set $\{v_i\}_{i=1,\ldots,M}$ is defined within a multiplicative constant by the following system:

$$v_i = \sum_{j=1}^{M} v_j r_{j,i} \qquad \text{for } i = 1, \ldots, M \tag{22.2}$$

One solution of this system for the example in Figure 22.1 is: $v_1 = 1; v_2 = 2; v_3 = 2$ and $v_4 = 1$ and can be interpreted as follows: between two successive visits at station 1, a customer visits, in average, twice stations 2 and 3, and once station 4. It is important to note that the visit ratios can be calculated whether the network is product-form or not, as soon as the routing of the customers is probabilistic.

$G(M,N)$ is such that the probabilities $p(\mathbf{n})$ sum up to 1 over all the possible vectors $\mathbf{n}$ such that $\sum_{i=1}^{M} n_i = N$, where n_i is the number of customers in station i. A direct derivation of the constant $G(M,N)$ is not feasible as it involves multiple summations, which leads obviously to an exponential complexity. Fortunately by means of the convolution algorithm [9], this constant can efficiently be derived with a polynomial complexity. Finally, with the convolution algorithm or some other very efficient algorithms such as the Mean Value Algorithm [11], one can easily obtain the local performance parameters such as, U_i: the utilization of the server of station i, Q_i: the average number of customers in station i, R_i: the mean residence time of a customer in station i and X_i: the average throughput of station i, for all $i = 1, \ldots, M$.

The network in Figure 22.1, together with a population $N = 10$ and stations having exponential distributions with parameters given in Table 22.1, will be referred to as Example 1A. The exact performance parameters of Example 1A are given in Table 22.2.

Exp. parameters	Station 1	Station 2	Station 3	Station 4
μ_i	1	1.25	1.6666	0.6666
t_i	1	0.8	0.6	1.5

Table 22.1: Parameters of the stations of Example 1A.

Average performance parameters	Station 1	Station 2	Station 3	Station 4
U_i	0.55201	0.88321	0.66241	0.82801
Q_i	1.15663	3.93285	1.72646	3.18406
R_i	2.09531	3.56232	1.56381	5.76816
X_i	0.55201	1.10401	1.10401	0.55201

Table 22.2: Exact average performance parameters of Example 1A.

The basic product-form 22.1, can be generalized by introducing state-dependent service rates. In that case, each station i of the network has a markovian distribution with load-dependent service rates $\mu_i(n_i)$, for $n_i = 1, \ldots, N$. This means that as long as the number of customers in station i is n_i, the time to the next departure is exponentially distributed with a rate $\mu_i(n_i)$. These networks are often referred to as Gordon-Newell's networks [17]. The steady-state solution of Gordon-Newell's networks are known to have the following product-form:

$$p(\mathbf{n}) = p(n_1, \ldots, n_M) = \frac{1}{G(M,N)} \prod_{i=1}^{M} \left[\prod_{n=1}^{n_i} \frac{v_i}{\mu_i(n)} \right] \tag{22.3}$$

Obviously the basic closed product-form networks are a particular case of these state-dependent closed product-form networks. Indeed, if for all station i, $\mu_i(n_i) = \mu_i$, for all $n_i = 1, \ldots, N$, the product-form solution 22.3 is equivalent to that in relation 22.1. It is also interesting to see the multiple-servers case as a special case of this class of network. Indeed, if station i has now C_i identical servers, the output rates of station i are load-dependent and given by:

$$\mu_i(n_i) = \begin{cases} n_i \mu_i & \text{for } n_i \leq C_i \\ C_i \mu_i & \text{for } n_i \geq C_i \end{cases}$$

22.1.2 Non product-form networks

If some stations of the network do not have exponential (or state-dependent markovian) service-time distributions, except in some very special cases, the network no longer has a product-form solution. Then, obtaining the performance parameters of this network requires to solve the underlying Markov chain (provided that the service-time distribution of the stations are described by means of a PH-distribution). Clearly, this is only feasible for very small networks and

for a very small population N, which is generally not the case for models of real systems.

Let us consider again the network in Figure 22.1 but in which all the stations have now a Coxian-2 service-time distribution. This example will be referred to as Example 1B. Let us remind that a Coxian-2 distribution with parameters (μ_1, a_1, μ_2) is constituted of a first exponential stage with parameter μ_1 followed with a probability a_1 by a second exponential stage with parameter μ_2 (see Figure 22.2). The parameters of the stations of Example 1B are given in Table 22.3. It is important to note that stations in Examples 1A and 1B have exactly the same mean service-time.

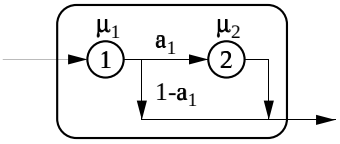


Figure 22.2: Coxian-2 distribution.

Cox-2 parameters	Station 1	Station 2	Station 3	Station 4
$(\mu_{i1}, a_{i1}, \mu_{i2})$	(2, 1, 2)	(5, 0.3, 0.5)	(4, 0.35, 1)	(2, 0.1, 0.1)
t_i	1	0.8	0.6	1.5
cv_i^2	0.5	3.25	1.78	8.55

Table 22.3: Parameters of the stations of Example 1B.

Table 22.4 shows the performance parameters of Example 1B obtained by simulation with QNAP2 package. All the parameters, excepted the throughputs, are given together with their 95% confidence interval. At that point it is interesting to compare the performance parameters of Examples 1A and 1B. Clearly they are very far from each other. Especially the throughputs differ from more than 20% and the mean response time of station 4, from about 30 %.The conclusion is thus that the performance of a closed queueing network having generally distributed service-time cannot be accurately estimated by that of a state-independent product-form network having the same mean service-time. This is a well-known result that has been confirmed by many numerical tests.

22.1.3 Principle

The idea is then to estimate the performance of a general closed queueing network having general service-time distributions by that of product-form network having load-dependent markovian service-times. The question is then: how can we estimate the load-dependent rates of the associated product-form network? One possibility is provided by Marie's method [12] and will be developed in the following.

Average performance parameters	Station 1	Station 2	Station 3	Station 4
U_i	0.4532 ±0.004	0.7262 ±0.004	0.5440 ±0.004	0.6842 ±0.004
Q_i	0.9017 ±0.014	3.873 ±0.029	1.591 ±0.022	3.634 ±0.046
R_i	1.987 ±0.012	4.265 ±0.040	1.753 ±0.012	8.004 ±0.156
X_i	0.4537	0.9081	0.9077	0.4540

Table 22.4: Average performance of Example 1B obtained by simulation.

To each station i of the original network having a general service-time distribution, is associated in the product-form network, a station i having state-dependent rates $\mu_i(n_i)$, n_i being the number of customers currently present in station i. The associated network is then a Gordon-Newell network [17] and has the product-form solution 22.3.

Ideally, the service rates $\mu_i(n_i)$ of station i in the associated product-form network should be equal to the conditional throughputs $\nu_i(n_i)$ of the corresponding station in the original network [5]. The conditional throughput $\nu_i(n_i)$ is defined as the average number of customers by unit of time leaving station i while it contains n_i customers. The conditional throughput $\nu_i(n_i)$ can alternatively be viewed as the inverse of the pseudo-service-time $S_i(n_i)$ introduced in [20] and defined as the average time station i spends in state n_i between two departures from station i.

It must be emphasized that obtaining the exact values of the conditional throughputs would, in general, require an exact solution of the original network [20]. Alternatively, if the conditional throughputs can be estimated, then the approximate product-form solution given by equation 22.3 can still be used [5]. In that case, the overall method will include two approximations: the first one is due to the product-form approximation and the second one involves the estimation of the conditional throughputs.

The problem then reduces in the estimation of the conditional throughputs. These estimates will be obtained by analyzing each station of the original network in isolation. Thus, the analysis of the whole system is replaced by the analysis of the M stations in isolation which hopefully will be much simpler. The price to pay is that, in general, only approximate values of the conditional throughputs will be obtained.

22.1.4 Analysis in isolation

One way of estimating the conditional throughputs is provided by Marie's method [12, 5]. The idea is to analyze each station of the original network in isolation under a state-dependent Poisson arrival process depending on the number of customers present in the station (see Figure 22.3). Let $\lambda_i(n_i)$ be the

state-dependent arrival rates in station i. This means that when the station is in state n_i, the time to the next arrival is exponentially distributed with rate $\lambda_i(n_i)$. Note that $\lambda_i(n) = 0$ for any $n \geq N$ since no new customer can arrive when station i contains the N customers of the system. This arrival process approximates the actual flow of customers entering station i in the original system. (Note that the actual flow is neither a Poisson process nor a renewal process).

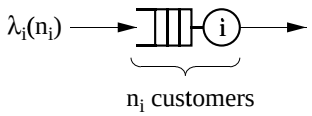


Figure 22.3: Analysis in isolation of station i.

Suppose first that the quantities $\lambda_i(n_i)$, for $n_i = 0, \ldots, N-1$, are given. Then, station i can be analyzed in isolation using any appropriate technique. Provided that the service-time is described by a PH-type distribution, the analysis in isolation of station i correspond to a $\lambda(n)/PH/1/N$ queue. Note that, as $\lambda_i(n) = 0$ for any $n \geq N$, the underlying Markov chain is finite and can easily be solved with direct (for some special cases, e.g. if the service-time is modeled by a Coxian distribution [13]) or iterative techniques [18]. Especially, the steady-state probability of having n_i customers in this isolated station, say $\tilde{p}_i(n_i)$, can be derived. Let $\tilde{\nu}_i(n_i)$ be the conditional throughput of this isolated station with population n_i. It can be obtained as [20]:

$$\tilde{\nu}_i(n_i) = \lambda_i(n_i - 1)\frac{\tilde{p}_i(n_i - 1)}{\tilde{p}_i(n_i)} \qquad \text{for } n_i = 1, \ldots, N \tag{22.4}$$

Once the conditional throughputs have been obtained, the load-dependent service rates of the i-th station of the associated product-form network are set equal to the conditional throughputs of the corresponding station analyzed in isolation, i.e.:

$$\mu_i(n_i) = \tilde{\nu}_i(n_i) \qquad \text{for } n_i = 1, \ldots, N \tag{22.5}$$

Now, in order to use this approach, the state-dependent arrival rates are required. It turns out that these quantities can be obtained from the product-form solution of the associated network given by relation 22.3. Indeed, provided the $\mu_i(n_i)$ are given, the state-dependent arrival rates of each station i can be obtained as [20]:

$$\lambda_i(n_i) = \mu_i(n_i + 1)\frac{p_i(n_i + 1)}{p_i(n_i)} \qquad \text{for } n_i = 0, \ldots, N-1 \tag{22.6}$$

In this equation, the probabilities $p_i(n_i)$ are the marginal probabilities of station i in the product-form network. Note that equations 22.4, 22.5, and 22.6 imply that the marginal probabilities in the product-form network are the same as those of the corresponding station in isolation, i.e.:

$$p_i(n_i) = \tilde{p}_i(n_i) \tag{22.7}$$

Now, from classical formulas on product-form networks [16], equation 22.7 can equivalently be written as:

$$\lambda_i(n_i) = v_i \frac{G_i(M-1, N-n_i-1)}{G_i(M-1, N-n_i)} \qquad \text{for } n_i = 0, \ldots, N-1 \tag{22.8}$$

where $G_i(M-1, n)$ is the normalization constant of the equivalent network with station i removed and a population n. Now, $G_i(M-1, n)$ is a function of the parameters $\mu_j(n_j)$, for all $j \neq i$ and $n_j = 1, ..., N$. These complementary normalization constants can efficiently be computed by means of the convolution algorithm [16].

22.1.5 Algorithm

So, it appears that the parameters $\mu_i(n_i)$ are the solution of a fixed-point problem defined by the above set of equations. In other words, the vector of unknowns μ, where $\mu = (\mu_1(1), \ldots, \mu_1(N), \ldots, \mu_M(1), \ldots, \mu_M(N))$, is solution of a system of equations of the form: $\mu = F(\mu)$. The number of unknowns is MN. Therefore an iterative procedure must be used to determine these quantities. The principle of this procedure is, as usual in fixed-point problems, to calculate a series of vectors $\mu^{(n)}$, where $\mu^{(n)} = F(\mu^{(n-1)})$, starting from an initial vector $\mu^{(0)}$. The procedure is stopped when $\mu^{(n)}$ and $\mu^{(n-1)}$ are close enough to each other. The vector $\mu^{(n)}$ then provides the estimation of the unknown vector μ. This procedure is described by Algorithm 22.1.

Step 1 of the algorithm calculates new values of the state-dependent arrival rates at the different stations, given the current values of the load-dependent service rates. Step 2 yields new estimates of the conditional throughputs, which are used in Step 3 to update the values of the load-dependent service rates. The number of iterations to achieve convergence is usually very reasonable (less than 10 on most examples). Note that it is possible to reduce the number of iterations of the algorithm, and then speed up the overall method, by solving the fixed-point iteration using a slightly different algorithm [1]. The idea is similar to that introduced in [15], and consists in calculating the state-dependent arrival rates of an isolated station just before its analysis in isolation using the most recent updated values of the load-dependent service rates corresponding to the other stations. It was shown in [1] that this may reduce the number of iterations by a factor of 2 without increasing the computational complexity of each iteration. When convergence has been achieved, all the performance parameters of interest can be obtained by using any computational method for obtaining the performance parameters of product-form closed queueing networks with load-dependent service rates [16].

Algorithm 22.1 Marie's method

Step 0. Set the $\mu_i(n_i)$ to some initial values, for example $\frac{1}{t_i}$,
for all $i = 1, \ldots, M$ and $n_i = 1, \ldots, N$.

Step 1. For $i = 1, \ldots, M$:
Calculate the state-dependent arrival rates $\lambda_i(n_i)$,
for $n_i = 0, \ldots, N-1$, from relation 22.8.

Step 2. For $i = 1, \ldots, M$:
- **a.** Analyze station i in isolation.
- **b.** Derive the marginal probabilities $\tilde{p}_i(n_i)$ of this isolated station, for all $n_i = 0, \ldots, N$.
- **c.** Calculate the conditional throughputs $\tilde{\nu}_i(n_i)$ of this isolated station, for all $n_i = 1, \ldots, N$, from relation 22.4.

Step 3. For $i = 1, \ldots, M$:
Update the load-dependent service rates of station i in the product-form network: $\mu_i(n_i) = \tilde{\nu}_i(n_i)$, for all $n_i = 1, \ldots, N$.

Step 4. Go to Step 1 until convergence of the parameters $\mu_i(n_i)$.

Step 5. Calculate the average performance parameters
from the product-form network.

22.1.6 Numerical results

The results of Example 1B analyzed by Marie's method are given in Table 22.5 together with the relative error with respect to the simulation results. As we can see, they are much better than thus provided by the product-form network with state-independent rates, as the throughputs are now estimated with an error lower than 3% and the average error on the other parameter is about 5%. All the numerical tests that have been run on Marie's method show that the throughputs are better estimated (in general with a relative error lower than 5%) than the other parameters (for which the relative error is generally between 5 to 15%). This is not a surprising result as the objective of the technique is to replace each station of the original network by a flow equivalent service center in the product-form network.

Average performance parameters	Station 1	Station 2	Station 3	Station 4
U_i	0.44247 -2.4 %	0.70795 -2.5 %	0.53097 -2.4 %	0.66371 -3 %
Q_i	0.82156 -8.9 %	3.81796 -1.4 %	1.68935 +6.2 %	3.67580 +1.2 %
R_i	1.85676 -6.5 %	4.31436 +1.1 %	1.90899 +8.9 %	8.30744 +3.8 %
X_i	0.44247 -2.5 %	0.88494 -2.6 %	0.88494 -2.5 %	0.44247 -2.6 %

Table 22.5: Approximate average performance of Example 1B.

22.2 Extended Single-class technique

The objective of this section is to extend the results of Marie's method, originally devoted to the analysis of closed queueing networks with general service-time distributions and presented in the previous section, to much more general single-class closed queueing networks. The networks we are now going to consider are general in the sense that, in addition to general service-time distributions, they may have features such as: blocking due to finite capacity queues, synchronization constraints, simultaneous resource possessions, population constraints, state-dependent service rates, etc.

The extension will consist in decomposing the network into a set of subsystems, associating with this partition of the system a product-form network, then applying the principle of Marie's method in order to estimate the parameters of this product-form network. As we will see, the crucial point in the extension of the basic technique lies in the decomposition procedure.

22.2.1 Example of a general closed queueing network

Before proceeding further it is interesting to present an example of what we consider as a very general single-class closed queueing networks. This network includes several non-classical mechanisms, each of them being obviously enough to result in a network that do not have a product-form solution. This example will be referred to as Example 2 and is shown in Figure 22.4. It is made of M = 19 service stations and has the following features:

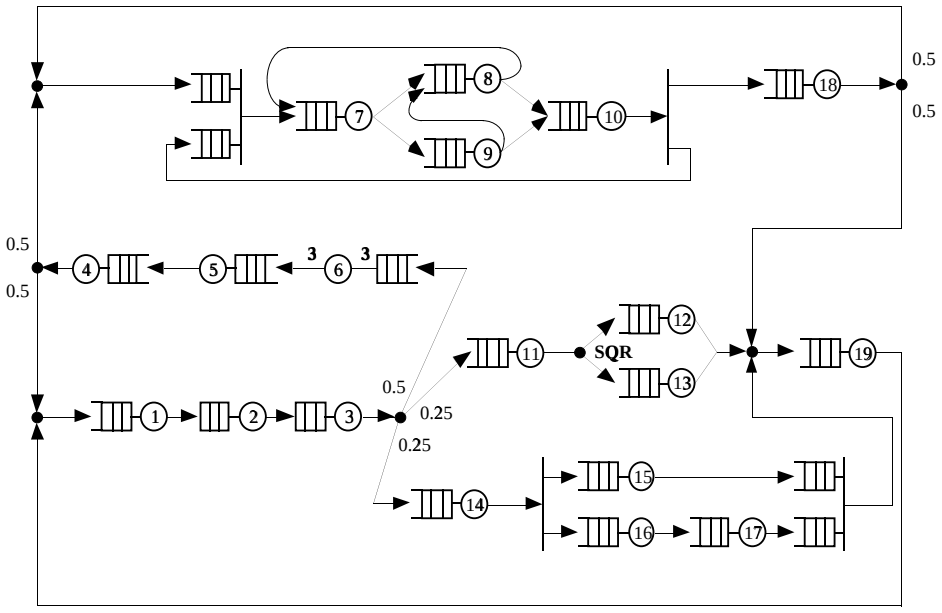


Figure 22.4: Example 2.

- Each station of the network may have a general service-time distribution modeled by a Coxian or a PH distribution.
- Stations 2 and 3 have buffers of finite capacity and thus blocking may occur. If we consider for instance the so-called blocking-after-service, a server is blocked if after completion of the service of a customer, the next queue is full. If this happens, the customer cannot be transferred until a space becomes available in the next queue. During this period of time, the server is blocked. So, servers 1 and 2 may be blocked because of the finiteness of the buffer of stations 2 and 3, respectively.
- Station 6 processes customers in batch. That is, server 6 does not start the processing of jobs as long as the number of customers in the queue is less than three. Now, as soon as three customers are present in the queue, server 6 starts the processing of a batch of three customers. At service completion, the three customers are simultaneously released in the queue of station 5.
- Stations 7, 8, 9, and 10 have a population constraint. That is, the total number of customers that can be simultaneously present in the subnetwork consisting of this set of stations cannot exceed a fixed value referred to as the capacity of the subnetwork. As explained in the Section 22.2.3, this population constraint will be modeled by a set of non consumable resources (or tokens), the total number of resources remaining constant and equal to the capacity of the subnetwork.
- Customers leaving station 11 can join either station 12 or station 13 according to a shortest queue routing (SQR) mechanism.
- A customer leaving station 14 is split into two lower-level customers (or sons). The first one then proceeds to station 15 while the second one proceeds successively to stations 16 and 17. As soon as the two lower-level customers have completed their service at stations 15 and 17, respectively, they are merged together and the resulting customer then joins station 19.

22.2.2 Principle

The idea is again to estimate the performance of a general closed queueing network, such as the network in Figure 22.4, by that of a product-form network with load-dependent service-rates. But obviously the product-form network cannot contain as many stations as the original network. The reasons for this are multiple:

- In a product-form network all the stations have an infinite capacity (or equivalently a capacity greater than the population of the network). Thus no blocking can occur. So if we wanted to associate with stations 1, 2 and 3 of Example 2, as many stations in the product-form network, the effects of the blocking that may occur at stations 1 and 2 in the original system,

could not be reflected by the product-form network. As a matter of fact the approximation will surely be very poor.

- A closed product-form network is first of all a closed network. This means that all the entities that circulates in it (the customers) are statistically identical and the number of these entities remains constant. Thus if we wanted to associate with stations 14 to 17 of the original network, four stations in the product-form network, it would be impossible to reflect the fact that, in the original network, a customer who completes its service at station 14 gives rise to two entities (called lower-level customers), one going to station 15 and the other going to station 16.

- In a product-form network the routing are probabilistic. If we wanted to associate with station 11, 12 and 13 of the original network, three stations in the product-form network, the state-dependent routing of the original network would have to be replaced by a probabilistic routing in the product-form network, which, again, would result in a bad approximation.

- In a product-form network there is no batch departures. If we wanted to associate with station 5 and 6 of the original network, two stations in the product-form network, we would need to transform the batch departures from station 6 into single-customer departures. This is, for sure, a bad idea.

The objective is then to hide in subsystems all these features that cannot be taken into account in a product-form network. This is the aim of the first step of the technique, namely the decomposition step. This decomposition will be obtained with respect to a set of four basic assumptions, as explained in the next section, and will result into what we call a feasible partition. The idea is then to associate with this feasible partition of the system, a product-form network containing as many stations as the number of subsystems, each one being associated to a given subsystem. The product-form network will model the global behavior of the system, that is the behavior of the network where each subsystem has been replaced by a black-box. The input/output behavior of a customer through a given station, in the product-form network, should thus reflect the input/output behavior of a customer through the corresponding subsystem, in the initial network. We believe that the following decomposition rules are such that the performance of the product-form network are close to that of the original network.

22.2.3 Decomposition

Let us consider a partition of the M stations of the system into K subsystems (or subnetworks): $\{R^1, ..., R^k, ..., R^K\}$ where R^k denotes the k-th subsystem. Let I^k be the set of stations belonging to subsystem R^k. Let n^k be the global state of subsystem R^k, i.e., the total number of customers in subsystem R^k, and

let $\mathbf{n} = (n^1, \ldots, n^k, \ldots, n^K)$. $\mathbf{n}$ will be referred to as the global state-vector of the system.

In order to be able to use product-form approximations, and as a consequence of the remarks given in the previous section, the partition must satisfy some basic assumptions. Indeed, we need four basic assumptions related to single-step transitions, state-dependent behavior, routing, and assembly/disassembly mechanisms. They are introduced below.

Assumption 22.1 *The partition is such that all transitions between subsystems are single step transitions.*

This means that any transition between two subsystems involves only a single customer. On the other hand, multiple customer transitions are allowed within a subsystem. Multiple customer transitions are encountered for instance in the case of bulk service. In this case, assumption A1 prescribes that all the stations that can be reached from a station with bulk service belong to the same subsystem as this station.

In Example 2, the only non-single step transitions that occur in the network are between stations 6 and 5 as a consequence of the batch processing of server 6. As a result, Assumption 22.1 is satisfied if stations 5 and 6 belong to the same subsystem, say R^1.

Assumption 22.2 *The partition is such that the behavior of each subsystem depends only on the state of this subsystem and is otherwise independent of the state of the rest of the network.*

Therefore, some subsystems may have explicit state-dependent behaviors. We allow for instance the service rates of a station of a subsystem to be dependent on the current (detailed) state of this subsystem. Blocking is an other form of state-dependent behavior. In that case, the above assumption prescribes that all the stations that have non-zero transition probabilities toward a finite capacity station belonging to a subsystem R^k, must be part of this subsystem.

In example 2, because of the blocking that may occur, the behavior of station 1 depends on the state of station 2. Assumption 22.2 then prescribes that these two stations must belong to a same subsystem. By a similar argument, stations 2 and 3 must also belong to the same subsystem. As a result, Assumption 22.2 is satisfied if stations 1, 2, and 3 belong to the same subsystem, say R^2.

We also need to consider the routing of the customers throughout the network. We restrict our attention to the case of probabilistic routing. Upon service completion at a station, a customer joins another station according to some routing decision. This routing decision may or may not be dependent on the state of the system. A state-dependent routing (SDR) is such that the next

station to be visited depends in part on the current state of the system. Shortest queue routing (SQR) is probably the most common form of state-dependent routing where the customer joins the station that has the smallest number of customers among a set of possible destination stations. State-dependent routing can be viewed as a form of state-dependent behavior. In this case, Assumption 22.2 prescribes that all possible destination stations as well as all stations involved in the state-dependent decision are part of the same subsystem. We need a further assumption on the routing which is now presented.

Assumption 22.3 *The partition is such that the input/output behavior of a customer through each subsystem is independent of the state-dependent routing decisions taken during its sojourn in the subsystem.*

Let us elaborate on this assumption. Consider a customer entering subsystem R^k. Let i ($i \in I^k$), be the station from which this customer came from. Then Assumption 22.3 requires that the probability that the customer will join a station j ($j \notin I^k$) just after departure from subsystem R^k is independent of any state-dependent routing decision that the customer may have taken during its sojourn inside the subsystem. This must be true for any station i from which a customer can enter subsystem R^k. Assumption 22.3 is obviously satisfied in the case where the original network has state-independent routing. Assumption 22.3 is also satisfied in the case of a subsystem that has a single input station and a single output station. Indeed, any state-dependent routing mechanism in between these two stations is allowed. Another example with more than one input and one output station is given in [5] and shows that Assumption 22.3 enables us to consider more general state-dependent routing schemes.

In Example 2, since customers are routed to stations 12 and 13 according to a state-dependent routing mechanism, Assumption 22.3 prescribes that these two stations must belong to the same subsystem, say R^3. Now this condition is sufficient for Assumption 22.3 to be satisfied since subsystem R^3 has a single upstream station, station 11, and a single downstream station, station 19.

The fourth assumption is related to assembly and disassembly mechanisms. Up to now, the only entities we have considered are customers. However, in some queueing network models of real systems, entities other than customers may be encountered: either lower-level customers or higher-level customers. Lower level customers (LL-customers) arise from a disassembly of customers. That is, at some point in the network, a customer is split into several LL-customers. These LL-customers then travel through a set of stations, independently of each other. At some point later, the LL-customers are merged into a single customer. The splitting corresponds to the disassembly (or fork) operation, and the merging to the reassembly (or join) operation. An illustrative example is given on Figure 22.5. Various cases may be encountered. The number of LL-customers resulting from the disassembly operation may be fixed or variable. The LL-customers

may be either of different types, having specific routing and/or service requirements, or undifferentiated. Finally, the reassembly may concern LL-customers corresponding to the disassembly of the same original customer, which is the most common case, or LL-customers generated by different customers. In the former case, the original customer is often referred to as the father and the LL-customers resulting from the disassembly operation as the sons.

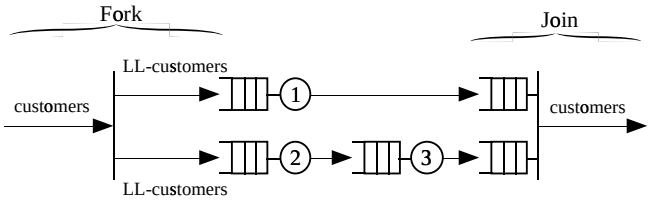


Figure 22.5: Illustration of LL-customers.

In some systems, customers may require resources other than servers. These resources are sometime referred to as passive resources whereas the servers are referred to as active resources. Passive resources can be classified into consumable and non-consumable resources. Let us first consider the case of non-consumable resources (NC-resources). In this case, there is a finite number of resources. A customer that requires an NC-resource must wait until such a resource is available. Then, the customer holds the resource for a certain period of time after which the resource is released. The synchronization of a customer and an NC-resource (or more generally several NC-resources) can be viewed as an assembly operation. The resulting entity, i.e., the customer and the NC-resource, will be called a higher-level customer (HL-customer). The release of the NC-resource can then be viewed as a disassembly operation performed on the HL-customer. The resulting entities, the original customer and the resource, then proceed independently. Once the resource is released, it can be immediately available for another customer or may have to visit a set of stations before. An illustrative example is given on Figure 22.6.

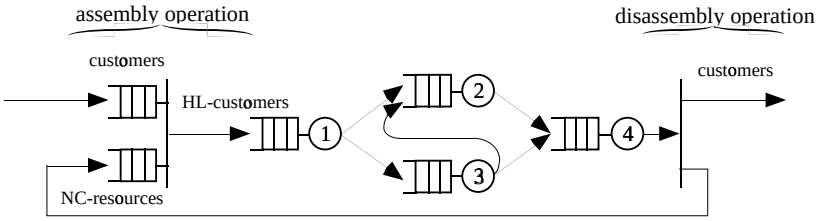


Figure 22.6: Illustration of HL-customers.

A very useful case which can be modeled by means of NC-resources is the case of population constraints. Consider a subnetwork where the total number of customers simultaneously present cannot exceed a fixed value called the capacity

of the subnetwork. A customer that attempts to enter this subnetwork while it is full is forced to wait in an external queue. As soon as a customer leaves the subnetwork, a new customer, if any in the external queue, is allowed to enter the subnetwork. Population constraints can easily be modeled as NC-resources. Indeed, a resource type is associated with each subnetwork having a population constraint. The total number of resources is then equal to the capacity of the subnetwork.

Let us now consider the case of consumable resources (C-resources). In that case, the number of resources is not a constant. C-resources are released to the system according to a given external arrival process and are consumed upon synchronization with a customer. As for NC-resources, a customer that requires a C-resource must wait until such a resource is available. As soon as the resource is available, the customer can resume its activity and the resource is consumed. The synchronization of a customer and a C-resource can again be viewed as an assembly operation. However, since the resource is consumed, the entity resulting from the assembly is the original customer. Indeed, as opposed to the case of NC-resources, no HL-customer is created. As a result, there is no corresponding disassembly operation.

Assumption 22.4 *The partition is such that the only transitions that may occur between subsystems are customer transitions.*

Assumption 22.4 states that transitions of either LL-customers or HL-customers between subsystems are not allowed. However, such transitions may occur within a subsystem. This means that, for any fork/join mechanism of a customer, the disassembly and the reassembly operations as well as all stations visited by the associated LL-customers belongs to a same subsystem. In this way, transitions of LL-customers between subsystems cannot be encountered. Similarly for any customer/NC-resource synchronization, the assembly and disassembly operations as well as all stations visited by the associated HL-customer belong to a same subsystem. In this way, transitions of HL-customers between subsystems cannot be encountered. Moreover, if the resource is not immediately available upon release, then the stations visited by the resources must also be part of this subsystem since Assumption 22.4 also prohibits resource transitions between subsystems. On the other hand, synchronization of customers and C-resources does not imply any constraint on the partition since no HL-customers are involved in this case.

In example 2, LL-customers are encountered in between stations 14 and 19. Thus, Assumption 22.4 implies that stations 15, 16, and 17, as well as the fork and join mechanisms, must belong to the same subsystem, say R^4. HL-customers are encountered as a result of the assembly of the customers with the NC-resources that correspond to the population constraint. Thus, Assumption 22.4 implies that stations 7, 8, 9, and 10, as well as the assembly and disassembly operations between customers and NC-resources, must belong to the same subsystem, say R^5.

A partition that satisfies Assumptions 22.1 through 22.4 will be called a feasible partition. It should be pointed out that a partition of the network can only be done once the customers have been defined. Indeed, there may be several entities in the network and one may have different choices for defining which entity is considered to be the customers of the network. A discussion pertaining to this issue is provided in [5]. It is also important to note that, for a given network and a given choice of the customers, there are several feasible decompositions. Two of them are of particular interest: 1) the minimal partition consists of a single subsystem that includes all the stations of the network and is obviously feasible, and 2) the maximal partition is such that no subsystem can be split into two subsystems without violating one of the assumption. It can be shown that, there is a unique maximal partition (provided that the customers of the network have been preliminary defined) and any feasible partition of the system can be obtained by merging some subsystems in the maximal partition. The important issue is thus to choose a particular partition among all feasible decompositions of the original closed queueing network. As discussed in [5] this choice is based on a trade-off between computational complexity and accuracy. As in general, the maximal partition is the one who allows to reduce the more the computational complexity, it is generally chosen.

In example 2, it appears that Assumptions 22.1 through 22.4 do not prescribe any constraint on stations 4, 11, 14, 18, and 19. Therefore, a subsystem can be associated with each of these stations. In this case, the corresponding partition is the maximal partition since no subsystem can be split into two subsystems without violating one of the assumptions. The total number of subsystems is thus K = 10. The maximal partition of Example 2 is given on Figure 22.7.

22.2.4 Approximate product-form solution

Let $\{R^1, ..., R^K\}$ be a feasible partition of the network. Let v^k be the average visit rate of a customer at subsystem R^k (or the mean visit ratio of subsystem R^k). v^k characterizes, in a aggregate way, the routing of the customers in between subsystems. As the network is closed, it is worthwhile noticing that visit rates are relative and thus the set $\{v^k\}_{k=1,...,K}$ is defined within a multiplicative constant.

When all subsystems have a single input station and a single output station, which is the case in Example 2, the average visit rates v^k can be obtained as one solution of a system similar to that in relation 22.2:

$$v^k = \sum_{k=1}^{K} v^k r^{k,h} \qquad \text{for } k = 1, \ldots, K$$

where $r^{k,h}$ is the mean routing probabilities in between subsystems k and h. Now it must be emphasized that the average visit rates can always be derived (provided that the network can be decomposed according to Assumptions 22.1

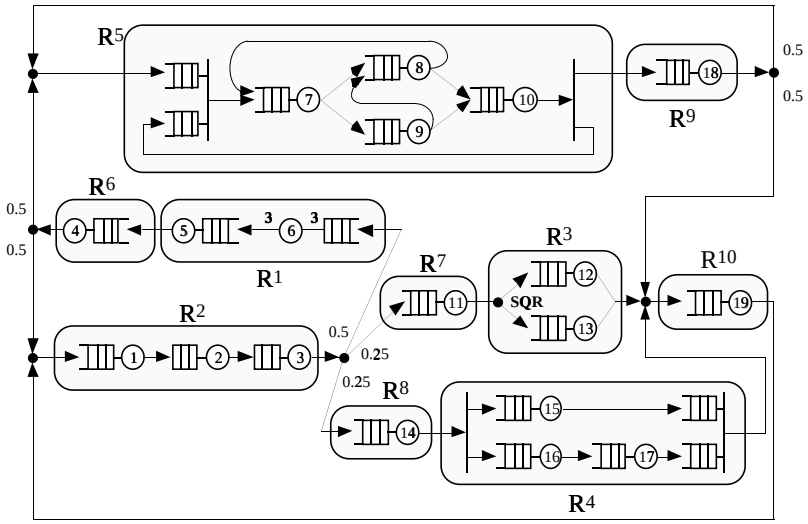


Figure 22.7: The maximal partition for the network of Fig. 22.4.

to 22.4) from the routing of the initial network [5]. This is specially easy in the case of state-independent routing (but remains true, thanks to Assumption 22.3, in the case of state-dependent routing).

For Example 2 decomposed according to the maximal partition and the routing probabilities shown in Figure 22.7, one possible set for the average visit rates is: $v^1 = 1; v^2 = 2; v^3 = 0.5; v^4 = 0.5; v^5 = 1; v^6 = 1; v^7 = 0.5; v^8 = 0.5; v^9 = 1$ and $v^{10} = 1.5$

The principle is then to associate with this partition of the system a product-form network. With each subsystem R^k of the partition is then associated, in the product-form network, a station k having state-dependent rates $\mu^k(n^k)$. The steady-state probabilities of this network have the product-form solution given in relation 22.9.

$$p(\mathbf{n}) = p(n^1, \ldots, n^K) = \frac{1}{G(K,N)} \prod_{k=1}^{K} \left[\prod_{n=1}^{n^k} \frac{v^k}{\mu^k(n)} \right] \tag{22.9}$$

where $\mathbf{n}$ is the global state-vector of the network, n^k is the number of customers in station k and $G(K,N)$ is a normalization constant.

With this approximation, each subsystem R^k is thus only characterized by the parameters v^k and $\mu^k(n^k)$ for $n^k = 1, \ldots, N$. v^k is nothing but the mean visit rate of subsystem R^k, that, as stated before, can be easily derived from the routing probabilities of the initial system. The only unknowns in this product-form approximation are then the state-dependent rates $\mu^k(n^k)$.

The product-form network associated with the partition of Example 2 in Figure 22.7 is shown in Figure 22.8. This network has $K = 10$ stations. It is important to note that, the routing of this network is only shown for picture conveniences, as the only parameters involved in the steady-state solution are the mean visit rates v^k (and not the mean routing probabilities $r^{h,k}$).

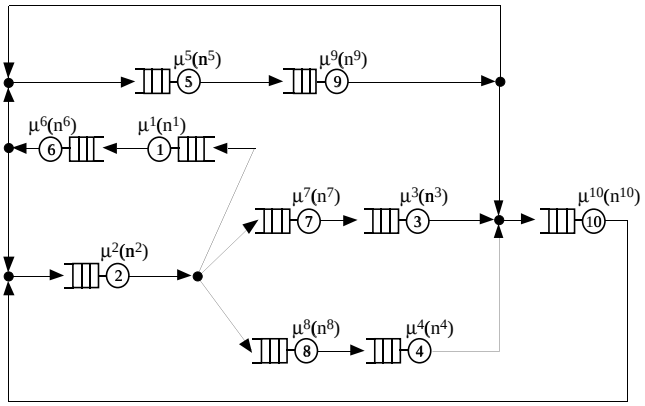


Figure 22.8: The product-form network associated with the partition in Fig. 22.7.

As in the basic case, the objective of the technique is to estimate the unknown state-dependent rates of the product-form approximation. Again, the best would be to estimate the service rates $\mu^k(n^k)$ of station k in the product-form network by the conditional throughputs $\nu^k(n^k)$ of the corresponding subsystem, R^k, in the original network; $\nu^k(n^k)$ being defined as the average number of customers by unit of time leaving subsystem R^k while it contains n^k customers. But again, obtaining the exact values of the conditional throughputs needs, in general, to have an exact solution for the original network, what, of course, we do not have. The idea is then to estimate the conditional throughputs of each subsystem in the original network, by the conditional throughputs of the subsystems analyzed in isolation. Once again, the overall method will include two approximations: the first one coming from the product-form approximation and the second one coming from the estimation of the conditional throughputs.

Each subsystem R^k will then be analyzed in isolation as an open model, in which customers arrive according to a state-dependent Poisson arrival process, with rates $\lambda^k(n^k)$ that depend on n^k, the total number of customers currently present in the isolated subsystem (see Figure 22.9 a.). As explained in Section 22.1.4, $\lambda^k(n) = 0$ for any $n \geq N$. As a consequence the isolated subsystem is totally equivalent to a closed network with one extra station (that actually represents subsystem-R^k complement in an aggregate and approximate way). This closed equivalent view is illustrated in Figure 22.9 b. The extra station has exponentially distributed service-times with load-dependent service rates given

by:

$$\mu_0^k(n_0^k) = \lambda^k(N - n_0^k), \text{for } n_0^k = 1, \ldots, N \tag{22.10}$$

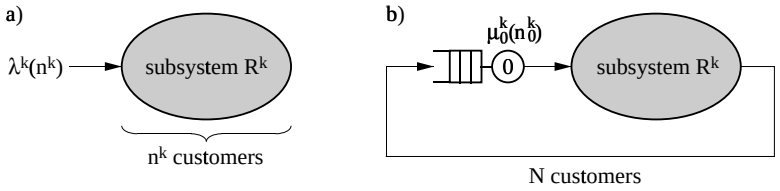


Figure 22.9: Analysis of subsystem R^k in isolation

In order to analyze in isolation each subsystem, one has to know the values of the state-dependent arrival rates $\lambda^k(n^k)$. We then extend the different steps of Marie's procedure presented in Section 22.1.4:

If we first assume that these quantities are given, then subsystem R^k can be analyzed in isolation using any appropriate technique. We will come back on this crucial point in the next section. The aim of analyzing subsystem R^k in isolation is, as explained in section 22.1.4, to obtain the steady-state probabilities $\tilde{p}^k(n^k)$ of having n^k customers in this isolated open subsystem. If the alternative equivalent closed view of subsystem R^k in isolation is used (see Figure 22.9), the probability $\tilde{p}^k(n^k)$ must now be interpreted as the probability of having n^k customers in subsystem R^k in the closed model. Note that in this case $\tilde{p}^k(n^k) = \tilde{p}_0^k(N - n^k)$, where $\tilde{p}_0^k(n_0^k)$ is the probability of having n_0^k customers at the extra station. The conditional throughputs $\tilde{\nu}^k(n^k)$ of this isolated subsystem can thus be derived as [20]:

$$\tilde{\nu}^k(n^k) = \lambda^k(n^k - 1)\frac{\tilde{p}^k(n^k - 1)}{\tilde{p}^k(n^k)} \qquad \text{for } n^k = 1, \ldots, N \tag{22.11}$$

Once the conditional throughputs have been obtained, the load-dependent service rates of the station k in the associated product-form network are set equal to the conditional throughputs of the corresponding subsystem analyzed in isolation:

$$\mu^k(n^k) = \tilde{\nu}^k(n^k) \qquad \text{for } n^k = 1, \ldots, N \tag{22.12}$$

Once all the subsystems have been analyzed in isolation, all the state-dependent rates $\mu^k(n^k)$ have been set up. Conversely that all the parameters of the product-form solution are given, the state-dependent arrival rates of each subsystem can be derived from the product-form solution, using relation 22.13, where $p^k(n^k)$ are the marginal probabilities of station k in the product-form network.

$$\lambda^k(n^k) = \mu^k(n^k + 1)\frac{p^k(n^k + 1)}{p^k(n^k)} \qquad \text{for } n^k = 0, \ldots, N - 1 \tag{22.13}$$

Algorithm 22.2 Single-class technique

Step 0. a. Decompose the network according to Assumptions 22.1 to 22.4 and obtain a feasible partition $\{R^1, ..., R^K\}$.
b. Set the $\mu^k(n^k)$ to some initial values for all $k = 1, \ldots, K$ and $n^k = 1, \ldots, N$.
Step 1. For $k = 1, \ldots, K$:
Calculate the state-dependent arrival rates $\lambda^k(n^k)$, for $n^k = 0, \ldots, N-1$, from relation 22.13.
Step 2. For $k = 1, \ldots, K$:
a. Analyze subsystem R^k in isolation.
b. Derive the marginal probabilities $\tilde{p}^k(n^k)$ of this isolated subsystem, for all $n^k = 0, \ldots, N$.
c. Calculate the conditional throughputs $\tilde{\nu}^k(n^k)$ of this isolated subsystem, for all $n^k = 1, \ldots, N$, from relation 22.11.
Step 3. For $k = 1, \ldots, K$:
Update the load-dependent service rates of station k in the product-form network: $\mu^k(n^k) = \tilde{\nu}^k(n^k)$, for all $n^k = 1, \ldots, N$.
Step 4. Go to Step 1 until convergence of the parameters $\mu^k(n^k)$.
Step 5. Calculate the global average performance parameters from the product-form network.
Step 6. Calculate the local average performance parameters from the last analysis in isolation of the subsystems.

As for the basic technique, the overall procedure is thus iterative, and consists in solving the set of equations 22.11, 22.12, and 22.13. Algorithm 22.2 describe the general single-class procedure. In comparison with the basic algorithm 22.1, two steps have been added: Step 0.a that consists in decomposing the network and Step 6 for the derivation of the local performance parameters of the stations. Step 0.a must be done according to the four assumptions described in Section 22.2.3. Step 5 consists in determining the global average performance of the decomposition, that is: the average number of customers in subsystem R^k, the mean residence time of a customer in subsystem R^k and the average throughput of subsystem R^k, for all subsystems ($k = 1, \ldots, K$). Step 6 consists in obtaining the local average performance parameters of each subsystem, that is: for each subsystem R^k, the average number of customers in station i, the mean residence time of a customer in a station i and the average throughput of station i, for each station i belonging to subnetwork R^k ($i \in I^k$).

22.2.5 Analysis in isolation

In order to obtain the conditional throughput estimates $\tilde{\nu}^k(n^k)$, each subsystem needs to be analyzed in isolation. However, since the procedure is iterative, this needs to be done as many times as the number of iterations required to reach the convergence. Moreover, at each iteration, new values of the state-dependent arrival rates $\lambda^k(n^k)$ are calculated. This involves some computations

in product-form expressions that can be done efficiently using, for example, the convolution algorithm [9]. As explained in [5], excepted in some cases (where the subsystems are very simple, e.g., subsystems consisting of a single station with a coxian-2 distribution), the main complexity is thus due to the analysis of the subsystems in isolation and not to the calculation of the state-dependent arrival rates.

It appears that a subsystem in isolation can always be viewed as a closed network. So the complexity of analyzing any subsystem in isolation is that of analyzing a closed network. Now, each subsystem in isolation may be solved exactly or approximately. An exact solution is obviously preferable but again may be difficult (and even sometimes infeasible) except if the subsystem is very simple. A typical case for which an exact solution is feasible is when the subsystem consists of a single station (which corresponds to the basic technique). On the other hand, using an approximate solution of the subsystem will reduce the computational complexity. The price to pay is that the estimation of the conditional throughputs will be less accurate, which will have an impact on the accuracy of the overall analysis. Since the objective is to reduce the computational complexity, this second choice is often made.

If the subsystem in isolation is to be analyzed approximately, we are again faced with the problem of obtaining an approximate solution of a closed network. The first idea is to try to use the same approach as for the original network, whose first step is to partition the closed subsystem. Now, it is easy to show that this closed subsystem is partitionable with respect to the required assumptions if the original partition of the overall closed network is not maximal with respect to this subsystem. In that case, the partition of the system will be hierarchical. That is, there is a first partition (global partition) of the system into a set of subsystems and each subsystem when analyzed in isolation is again partitioned (local partition).

In the case where the closed subsystem is not partitionable, it may still be possible to use the general approximate technique considered here. The trick is to transform the closed subsystem into an equivalent closed network which is now partitionable with respect to Assumptions 22.1 to 22.4. A typical case where this approach can be successfully used is when the subsystem corresponds to a subnetwork having a population constraint. This will be described in Section 22.2.6. Finally, if neither of the preceding approaches is applicable, any other appropriate approximate techniques may be used.

22.2.6 Example of application

Closed networks including subnetworks with restricted capacity

We consider closed queueing networks with general service-time distributions and subnetworks having a population constraint. This particular application of the general technique developed in the previous sections is described in detail in [3].

Let us consider a closed queueing network consisting of M stations and N

customers. The routing of customers through the system is assumed to be probabilistic and independent of the state of the system. We assume that the service-time distribution of each station is given by a Coxian distribution (or more generally a Phase-type distribution) and that the service disciplines are FIFO. We further assume that some stations belong to subnetworks having a population constraint. In such a subnetwork the total number of customers that can be simultaneously present is limited to a fixed value called the capacity of the subnetwork. We assume that a station can only be part of a single subnetwork. Let C be the total number of subnetworks having a population constraint and C^k, $k = 1, \ldots, C$, be a subnetwork with population constraint N^k. As described in Section 22.2.3, the population constraint of a given subnetwork is modeled by a set of non consumable resources (NC-resources), the total number of NC-resources being equal to the capacity of the subnetwork. Finally, there may be stations that do not belong to any subnetwork. These stations will be referred to as isolated stations. Let L be the total number of isolated stations. An example of a system containing subnetworks with population constraints is presented in Figure 22.10 and referred to as Example 3. This example is made of three isolated stations ($L = 3$) and two subnetworks ($C = 2$), say C^1 and C^2, with population constraints N^1 and N^2. Subnetwork C^1 consists of stations 1, 2 and 3, and subnetwork C^2 consists of stations 4, 5 and 6.

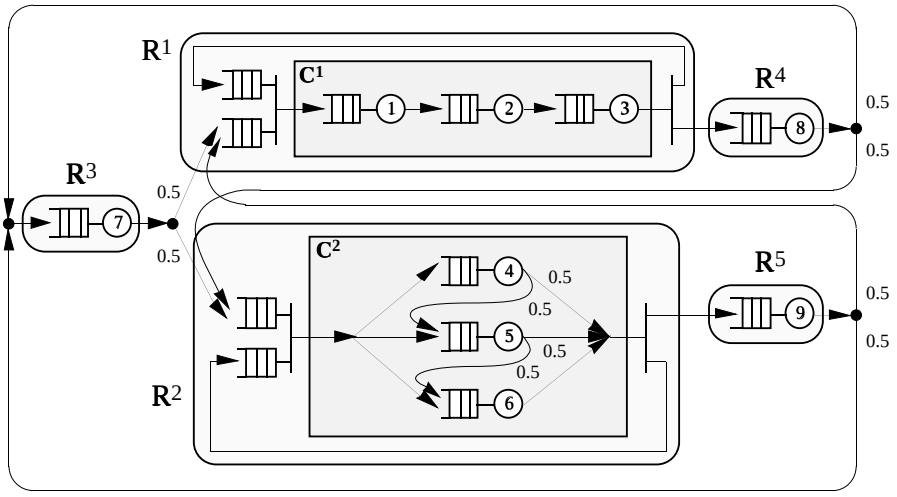


Figure 22.10: Example 3.

We are now going to show how the general technique can be used to analyze this kind of models. The first step of the technique is to derive a feasible partition of the system. From the developments in Section 22.2.3, it is easy to check that the partition such that every subsystem R^k consists of either an isolated station or a subnetwork C^k with population constraint N^k (see Figure 22.10) is a feasible partition for the system (actually it is the maximal partition).

The total number of subsystems is thus: $K = L + C$.

The second step of the method is to associate with the system partitioned into K subsystems, a product-form network consisting of K stations. The third step is then to estimate the parameters of these stations. In this way, each subsystem R^k will have to be studied in isolation. For a subsystem consisting of a single general service-time station, the analysis in isolation corresponds to a $\lambda(n)/PH/1/N$ queue and can be led using efficient algorithm (see Section 22.1.4). Now, let us consider a subsystem R^k containing a subnetwork C^k with population constraint N^k. The analysis in isolation of such a subsystem is shown in Figure 22.11. In this open model, customers coming from the outside assemble with a finite number N^k of NC-resources. The higher-level customer (HL-customer) resulting from the assembly travel through the subnetwork C^k before splitting into a customer that leaves the system and a NC-resource that becomes available for another customer to enter.

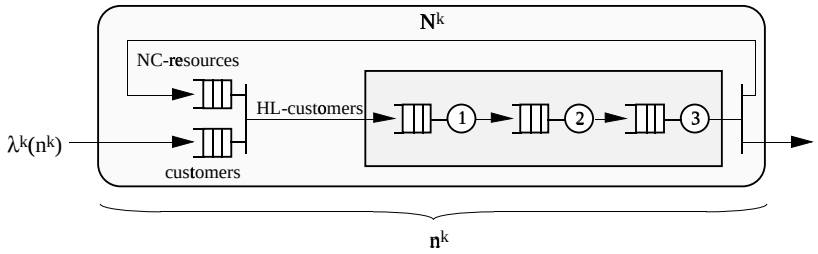


Figure 22.11: Analysis in isolation of a subnetwork with restricted capacity.

The idea is to transform the isolated subnetwork in Figure 22.11 into that in Figure 22.12, by exchanging the roles of the customers and the resources. The idea comes from [19] and was originally devoted to the analysis of open queueing networks with restricted capacity. It is important to note that this transformation is exact (see [19, 3]) as it is just another way of considering the isolated subnetwork ,which is actually a mixed network (open with regards to the customers and closed with regards to the NC-resources). The customers of the equivalent model are the NC-resources of the previous one. As the number of NC-resources of the first model is constant, the second model is closed and contains N^k customers. Now by transforming the customers of the first model into consumable resources (C-resources) in the second one, the assembly mechanism (customer + NC-resource $\rightarrow$ HL-customer) is transformed into a synchronization mechanism between a customer and a C-resource. As the resulting entity is again a customer, the disassembly mechanism (HL-customer $\rightarrow$ customer + NC-resource) of the first model disappear in the second one.

The problem is now reduced to the analysis of a closed network. This network consists of the set of stations of subnetwork C^k and of an extra synchronization station. To analyze this closed network, we are going to use again the general method described in Sections 22.2.2 to 22.2.5, at a local level. The first point of this method is to get a feasible partition for the system. A possible

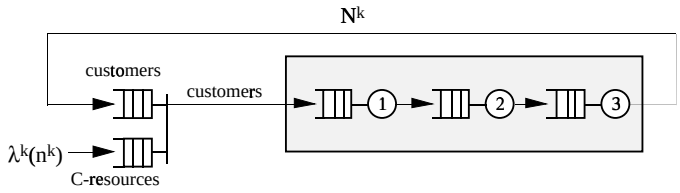


Figure 22.12: Equivalent transformation.

partition is given in Figure 22.13. In this partition, subsystem R_0^k contains the synchronization station, whereas each subsystem R_i^k contains one of the stations belonging to the set of the stations of C^k. Again, we can easily check that this partition is a feasible partition for the network. Indeed a synchronization mechanism between a customer and a C-resource does not imply any constraint on the partition with respect to Assumption 22.4, since in such mechanism no HL-customer is created. However, another assumption must now be taken into consideration. Indeed, Assumption 22.2 states that the partition must be such that the behavior of each subsystem only depends on the state of this subsystem, and is otherwise independent of the state of the rest of the network. Thus, we have to check that subsystem R_0^k satisfies this assumption. Indeed, subsystem R_0^k includes a state dependent behavior in relation to the global state n^k, as the arrival rates of C-resources depends on n^k. But it is easy to see that n^k can only be obtained from the detailed state of subsystem R_0^k, namely (n_0^k, n_a^k) where n_0^k is the number of customers and n_a^k is the number of C-resources in R_0^k (for more details see [3]):

$$n^k = \begin{cases} N^k - n_0^k & \text{if } n_0^k \neq 0 \\ N^k + n_a^k & \text{if } n_0^k = 0 \end{cases}$$

Thus, the behavior of subsystem R_0^k only depends on the state of this single subsystem. Assumption 22.2 is satisfied. And so the partition of Figure 22.13 is a feasible partition for the system. Obviously it is actually the maximal partition. It is important to note that the trick of transforming the open network in Figure 22.11 into that equivalent closed network in Figure 22.12 has allowed to decompose the subsystem more than we were able to do in the original partition of the whole network.

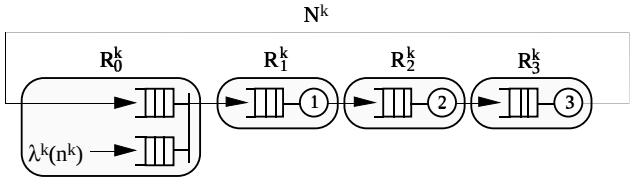


Figure 22.13: Decomposition.

Once the network has been decomposed, the general procedure described in Section 22.2.4 can be applied. A product-form network will then be associated with this partition of the system and the parameters of the product-form network will be estimated by analyzing in isolation each subsystem. The analysis in isolation of a subsystem consisting of a generally distributed station has been already discussed. Now it is easy to show that the Markov chain associated with the analysis in isolation of subsystem R_0^k is linear (and finite). This analysis is developed in [3]. The overall method is thus hierarchical. A global iteration must be achieved in order to estimate the conditional throughputs of each subsystem of the global partition (illustrated in Figure 22.10). And for each iteration of this global loop, a local iteration must be implemented in order to analyze each subnetwork with population constraint. All the technical details about this example of application can be found in [3].

Let us consider Example 3 with stations having Coxian-2 distribution given in Table 22.6 and a population $N = 10$. We varied the capacities N^1 and N^2. Table 22.7 only gives the throughput of this closed system for the different configurations. The results obtained with the approximate technique are compared to the simulation results. More results can be found in [3].

Cox-2 parameters	Stations 1, 2 and 3	Stations 4, 5 and 6
$(\mu_{i1}, a_{i1}, \mu_{i2})$	(2, 0.2, 0.4)	(1, 0.2, 0.2)
t_i	1	2
cv_i^2	2.5	2.5

	Station 7	Station 8	Station 9
$(\mu_{i1}, a_{i1}, \mu_{i2})$	(2, 0.1, 0.2)	(3, 0.2, 0.3)	(4, 0.3, 0.4)
t_i	1	1	1
cv_i^2	5	4.11	3.25

Table 22.6: Parameters of the stations of Example 3.

	$N^1 = 2$ $N^2 = 2$	$N^1 = 2$ $N^2 = 8$	$N^1 = 8$ $N^2 = 2$	$N^1 = 5$ $N^2 = 5$	$N^1 = 8$ $N^2 = 8$
Approx.	0.3804 +0.1 %	0.4018 -0.9 %	0.4117 +1.0 %	0.4432 -0.4 %	0.4577 +0.7 %
Simul.	0.3799	0.4056	0.4076	0.4454	0.4544

Table 22.7: Average throughput of Example 3.

Closed networks including fork/join subnetworks

We now consider closed queueing networks with general service-time distributions and fork/join subnetworks. This particular application of the general technique is described in detail in [4].

Let us consider a closed queueing network consisting of M stations and N customers. The routing of customers through the system is assumed to be probabilistic and independent of the state of the system. We assume that the service-time distribution of each station is given by a Coxian distribution (or more generally a Phase-type distribution) and that the service disciplines are FIFO. Now, at some point in the network, a customer can be split into several lower-level customers (LL-customers). These LL-customers then travel through a set of stations, independently of each other. At some point later, the LL-customers are merged into a single customer. The splitting of the customer corresponds to the disassembly (or fork) operation while the merging of the LL-customers corresponds to the reassembly (or join) operation. There may be several such fork/join mechanisms in the network. For each fork/join mechanism we assume that:

- There is a fixed number of LL-customers resulting from the fork operation.
- Each LL-customer visits an arbitrary number of stations.
- The stations visited by the different LL-customers are distinct (this assumption can be relaxed and is made here for ease of presentation).
- The fork and the join operation are instantaneous.

The original customer is often referred to as the father and the LL-customers resulting from the fork operation as the sons. A subsystem consisting of a fork/join mechanism as well as all the stations visited by the corresponding LL-customers will be referred to as a fork/join subnetwork. Note that the numbers of stations visited by the different LL-customers need not be the same. Let F be the total number of fork/join subnetworks and F^k, $k = 1, \ldots, F$, be a given fork/join subnetwork. Now some stations do not belong to any fork/join subnetwork and are thus visited by customers (and not LL-customers). Again, these stations will be referred to as isolated stations. Let L be the total number of isolated stations. An example of a system containing fork/join subnetworks is given in Figure 22.14 and referred to as Example 4. This example is made of four isolated stations ($L = 4$) and two fork/join subnetworks ($F = 2$), say F^1 and F^2. Upon arrival of a customer to subnetwork F^1, this customer is instantaneously split into three LL-customers. It is convenient to assign a type to each one of the LL-customers. In this particular example, each original customer gives rise to one type-1 LL-customer, one type-2 LL-customer and one type-3 LL-customer. The type-1 LL-customer visits stations 1 and 2 successively and then goes to the join station; the type-2 LL-customer visits station 3 and then goes to the join station; the type-3 LL-customer visits stations 4 and 5 successively and then goes to the join station. As soon as the three LL-customers (one of each type) are present at the join station, they are merged into a single customer that leaves the fork/join subnetwork (and goes to station 9). Again, this join operation is instantaneous. In subnetwork F^2 a customer is split into two LL-customers, the first one going to station 6 and the second one going to station 7 then to station 8.

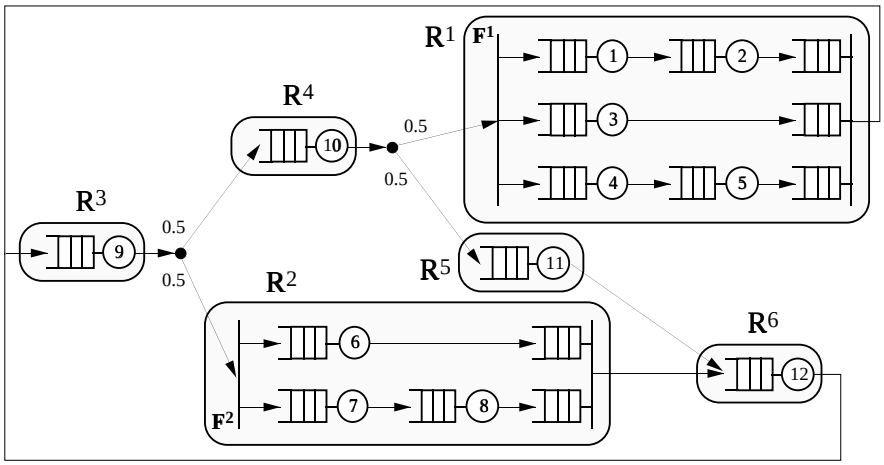


Figure 22.14: Example 4.

As in the previous example, we are going to show how the general technique can be used to analyze a closed queueing network with subnetworks having population constraints. The first step consists in obtaining a feasible partition of the system. The following partition such that every subsystem R^k consists of either an isolated station or a fork/join subnetwork F^k (see Figure 22.14) is, from the developments in Section 22.2.3, the maximal (feasible) partition of the system. The total number of subsystems is then: $K = L + F$.

The idea is then to associate with this partition a product-form network consisting of K stations. In order to estimate the parameters of these stations, each subsystem R^k needs to be analyzed in isolation. For a subsystem consisting of a single station, the analysis in isolation corresponds to a $\lambda(n)/PH/1/N$ queue (see Section 22.1.4). Now, for a subsystem R^k containing a fork/join subnetwork F^k the analysis in isolation is shown in Figure 22.15. Part a. and b. of this figure show the two equivalent way of considering the analysis in isolation of a given subsystem (see Section 22.2.4). In the open model of Figure 22.15a, upon arrival to the network (according to a state-dependent Markovian process with rates $\lambda^k(n^k)$), a customer is split into several LL-customers that independently travel through the network after which they are reassembled into the original customer that leaves the system. In the equivalent closed network of Figure 22.15b, the extra station models the arrival process of the customers into subsystem R^k. The server of this station is a Markovian server with load-dependent service rates given by relation 22.10.

In this last closed network, a customer that has been reassembled by the join mechanism, goes to the extra station before being split into several LL-customers by the fork mechanism. We are now going to see that this model is totally equivalent to a synchronized multi-class closed queueing network. This equivalent view can be obtained in three steps:

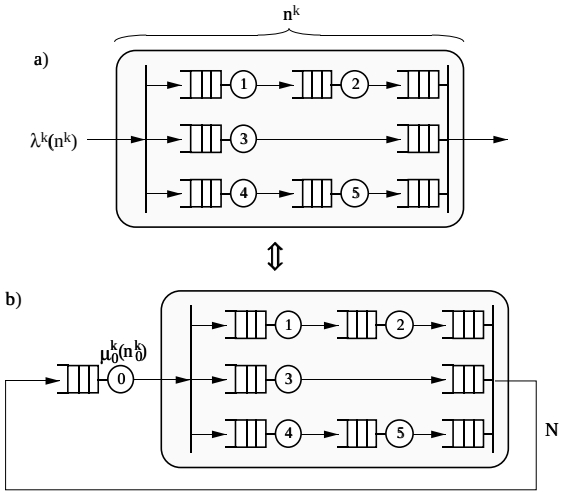


Figure 22.15: Analysis in isolation of a fork/join subnetwork.

- First, the network in Figure 22.15b is transformed into the synchronized closed queueing network in Figure 22.16a. The transformation is only obtained by putting together the fork and the join operation. It is worthwhile noticing that this transformation is exact. The only difference is that the entities of this equivalent view are now the LL-customers (of the different types).
- Now it is easy to check that the network in Figure 22.16a has the same behavior as that of the synchronized closed queueing network represented in Figure 22.16b. The synchronization station of this network will be referred to as a timed-synchronization station. The time to the next release of a group of LL-customers (one of each type) is exponentially distributed with rate $\mu_0^k(n_0^k)$, where n_0^k is the minimum number of LL-customers of each type in the input queues of the synchronization station: $n_0^k = \min(n_{01}^k, n_{02}^k, n_{03}^k)$.
- Finally, the network in Figure 22.16b, is equivalently transformed into the multi-class network shown in Figure 22.17. This is only done by considering each type of LL-customers as customers of a given class.

We are now facing the problem of analyzing a multi-class synchronized closed queueing network. This network is visited by as many classes as the number of types of LL-customers in fork/join subnetwork F^k, and is made of the set of service-stations of F^k plus an extra timed-synchronization station between the different classes. Each class has the same population, namely N. To analyze this multi-class closed queueing network, we will use an extension of the general single-class technique presented in the previous sections, to multi-class closed queueing networks with synchronization stations. This extension is presented

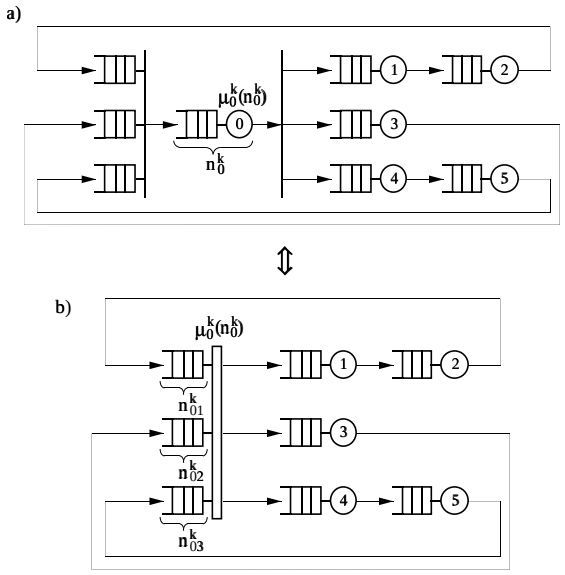


Figure 22.16: Equivalent transformation.

in Section 22.4. It is again based on the principle of the decomposition of the system into subsystems and is iterative. Here, we just show in Figure 22.17 the decomposition step of this multi-class technique. Once more, the trick of transforming the isolated fork/join subnetwork in Figure 22.15 into that equivalent multi-class closed network in Figure 22.17 has allowed to decompose the subsystem more than we were able to do in the original partition of the whole network.

The overall method is thus again hierarchical. A global iteration is achieved in order to estimate the conditional throughputs of each subsystem of the global partition. And for each iteration of this global loop, a local iteration is implemented in order to analyze each fork/join subnetwork. All the technical details about this example of application can be found in [4].

Let us consider Example 4 with stations having Coxian-2 distribution with parameters shown in Table 22.8. Table 22.9 gives the throughput of this closed system for three different populations, $N = 5$, $N = 10$ and $N = 20$. Table 22.10 gives the mean response times of a customer (or a LL-customer) at each station of the network for a population $N = 10$. The results obtained with the approximate technique are compared to the simulation results. The approximate results are given together with the relative error with respect to the simulation results. The simulations results are given, in the second table, with their 95% confident interval. More results can be found in [4].

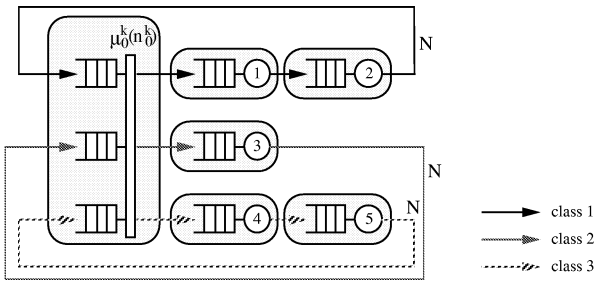


Figure 22.17: Decomposition.

Cox-2 par.	St. 1	St. 2	St. 3	St. 4	St. 5
$(\mu_{i1}, a_{i1}, \mu_{i2})$	(2, 0.1, 0.2)	(1, 0.1, 0.1)	(0.5, 0, -)	(1, 0, -)	(1, 0.2, 0.4)
t_i	1	2	2	1	1.5
cv_i^2	5	5	1	1	1.44

Cox-2 par.	St. 6	St. 7	St. 8
$(\mu_{i1}, a_{i1}, \mu_{i2})$	(0.67, 0.1, 0.2)	(1.5, 0, -)	(0.5, 0, -)
t_i	2	0.67	2
cv_i^2	1.75	1	1

Cox-2 par.	St. 9	St. 10	St. 11	St. 12
$(\mu_{i1}, a_{i1}, \mu_{i2})$	(1, 0, -)	(2, 0.5, 2)	(0.5, 0, -)	(1, 0.1, 0.2)
t_i	1	0.75	2	1.5
cv_i^2	1	0.78	1	2.56

Table 22.8: Parameters of the stations of Example 4.

22.3 Basic multi-class technique

In this section, we present an extension of the basic single-class technique described in Section 22.1, to multi-class closed queueing networks. As in the single-class case we are first going to present the general principle of this approximation technique on a restricted class of multi-class closed queueing networks. So, let us first consider a closed queueing network consisting of M stations and R classes of customers. Customers are not allowed to change class, so the number of customers in each class is constant. Let N_r be the total number of class-r customers, $\mathbf{N} = (N_1, \ldots, N_R)$ be the population-vector and $N = \sum_{r=1}^{R} N_r$ be the total population of the system. Each station i of this network has an infinite capacity (or equivalently a capacity greater than N), i.e. $K_i = \infty$ (or $K_i \geq N$), and C_i identical servers. Let $t_{r,i}$ be the mean service-time of a class-r customer in station i. We assume that the service-times are i.i.d. random variables. Finally we assume that the routing of the customers through the network is probabilistic. Let $r_{r,i,j}$ be the probability for a class-r

	$N = 5$	$N = 10$	$N = 20$
Approx.	0.509	0.674	0.796
	+1.2 %	+0.7 %	+0.8 %
Simul.	0.503	0.669	0.790

Table 22.9: Average throughput of Example 4.

	St. 1	St. 2	St. 3	St. 4	St. 5
Approx.	1.591	4.663	3.000	1.202	2.111
	+2.0 %	+1.8 %	-1.8 %	+0.3 %	+0.9 %
Simul.	1.560	4.580	3.055	1.198	2.092
	±0.035	±0.058	±0.024	±0.006	±0.016

	St. 6	St. 7	St. 8
Approx.	6.327	0.8587	5.529
	+3.0 %	-1.7 %	+0.5 %
Simul.	6.143	0.8734	5.504
	±0.080	±0.005	±0.053

	St. 9	St. 10	St. 11	St. 12
Approx.	2.747	0.9742	3.000	5.722
	-7.3 %	-0.9 %	-0.4 %	+1.7 %
Simul.	2.963	0.9835	3.012	5.628
	±0.020	±0.007	±0.033	±0.055

Table 22.10: Mean response times of Example 4, with $N = 10$.

customer completing a service at station i to go to station j.

An illustrative example with $M = 12$ stations and $R = 3$ classes is given in Figure 22.18. Class-1 customers visit stations 1, 2, 3, 5, 6, 7, 8, 9, 11, and 12. Class-2 customers visit stations 1, 3, 5, 6, 8, 9, 11, and 12. Class-3 customers visit stations 1, 3, 4, 5, 6, 10, and 12. The routing probabilities are shown on the figure. This example will be referred to as Example 5.

22.3.1 Multi-class product-form networks

In order for the multi-class network to have a product-form solution we need to add the following assumptions [8]. The scheduling discipline of each station can be of one of the following four types: 1) FCFS (first-come first-served), 2) LCFS (last-come first-served preemptive-resume), 3) PS (processor sharing) or 4) IS (infinite server). In addition, if the scheduling discipline of a given station i is FCFS, the service-time must be exponentially distributed and independent of class of the customer in service, i.e. if $\mu_{r,i} = \frac{1}{t_{r,i}}$ is the service-rate of class-r customer in station i, $\mu_{r,i} = \mu_{s,i} = \mu_i$ for all r and s. Now if a station i has one of the three other scheduling discipline, it may have a class-dependent general service-time distribution. Let again $\mu_{r,i} = \frac{1}{t_{r,i}}$ be the service-rate of class-r

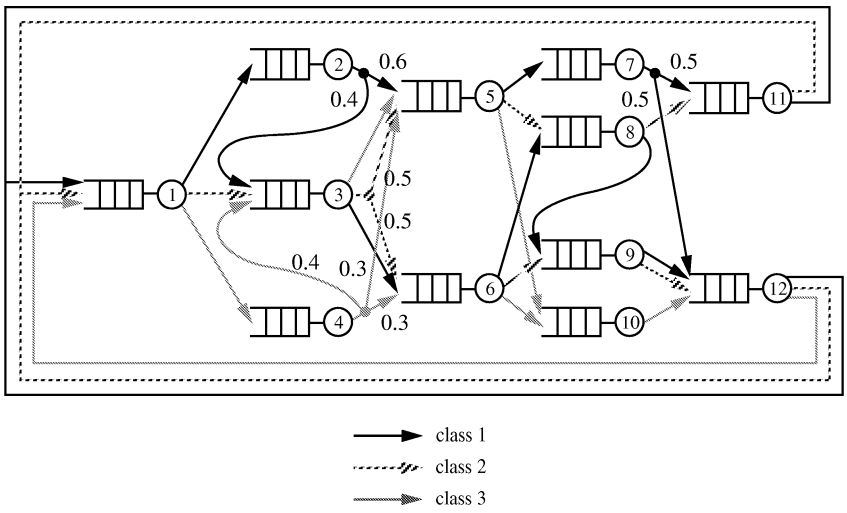


Figure 22.18: Example 5.

customer in station i.

Let $S(r) \in 1, ..., M$ be the set of indexes of the stations that are visited by class-r customers, $r = 1, \ldots, R$. Let $R(i) \in 1, ..., R$ be the set of classes of customers that visit station i, $i = 1, \ldots, M$. Let $n_{r,i}$ denote the total number of class-r customers currently present at station i. Let n_i be the total number of customers currently present in station i: $n_i = \sum_{r=1}^{R} n_{r,i}$. Let $\mathbf{n_i} = (n_{1,i}, \ldots, n_{r,i}, \ldots, n_{R,i})^T$; $\mathbf{n_i}$ will be referred to as the state-vector of station i. Let $\mathbf{n_r} = (n_{r,1}, \ldots, n_{r,i}, \ldots, n_{r,M})$; $\mathbf{n_r}$ will be referred to as the state-vector of class-r customers in the system. Note that $\sum_{i=1}^{M} n_{r,i} = N_r$. Finally, let $\mathbf{n} = (\mathbf{n_1}, \ldots, \mathbf{n_i}, \ldots, \mathbf{n_M}) = (\mathbf{n_1}, \ldots, \mathbf{n_r}, \ldots, \mathbf{n_R})^T$; $\mathbf{n}$ will be referred to as the state-matrix of the system. This network is known to have the following product-form solution, i. e., the equilibrium probability of the state $\mathbf{n}$ is given by [8]:

$$p(\mathbf{n}) = \frac{1}{G} \prod_{i=1}^{M} f_i(\mathbf{n_i}) \tag{22.14}$$

where

$$f_i(\mathbf{n_i}) = \begin{cases} \dfrac{n_i!}{\prod_{k=1}^{n_i} \min(k, C_i)} \prod_{r=1}^{R} \dfrac{{v_{r,i}}^{n_{r,i}}}{n_{r,i}!\ {\mu_{r,i}}^{n_{r,i}}} & \text{if station } i \text{ has a FCFS, LCFS or PS discipline} \\ \prod_{r=1}^{R} \dfrac{{v_{r,i}}^{n_{r,i}}}{n_{r,i}!\ {\mu_{r,i}}^{n_{r,i}}} & \text{if station } i \text{ has a IS discipline} \end{cases}$$

$v_{r,i}$ is the mean visit ratio of class-r customers at station i and G is a normalization constant. The mean visit ratios can be obtained, for each class r, from a relation similar to that in equation 22.2 (where the index r is added). The mean visit ratios of the different stations of Example 5 are given in Table 22.11.

Station	1	2	3	4	5	6	7	8	9	10	11	12
Class 1	1	1	0.4	-	0.6	0.4	0.6	0.4	0.4	-	0.3	0.7
Class 2	1	-	1	-	0.5	0.5	-	0.5	0.5	-	0.5	0.5
Class 3	1	-	0.4	1	0.7	0.3	-	-	-	1	-	1

Table 22.11: Mean visit ratios of the stations of Example 5.

For this particular class of networks, the performance parameters can be calculated using a variety of different computational algorithms such as the convolution algorithm [9, 16] and the mean-value-analysis (MVA) algorithm [11]. In theory, these algorithms enable us to obtain the average performance parameters such as, $Q_{r,i}$: the average number of class-r customers in station i, $R_{r,i}$: the mean residence time of a class-r customer in station i and $X_{r,i}$: the average throughput of class-r customers out of station i, for all $i = 1, \ldots, M$ and $r \in R(i)$.

The network in Figure 22.18 with stations having exponential distributions with parameters given in Table 22.12, will be referred to as Example 5A. The exact throughputs of Example 5A, for three different population-vectors, (5, 4, 3), (6, 8, 10) and (14, 16, 10) are given in Table 22.13. The exact mean response-times of Example 5A, for a single population-vector (6, 8, 10), are given in Table 22.14.

In practice, there are two limitations of multi-class product-form closed queueing networks. First, the assumptions of product-form networks are often too restrictive for modeling and analysis of real systems. Second, even in the case where product-form networks are appropriate models, calculating the performance parameters of the network using any of the available computational algorithms may not be feasible. This is due either to memory constraints or to prohibitive computational times, particularly when the network has many classes of customers [2]. In fact, the computational complexity grows drastically

Station i	1	2	3	4	5	6
Discipline	PS	FCFS	FCFS	FCFS	PS	PS
$t_{1,i}$	1	2	1	-	1	0.75
$t_{2,i}$	0.5	-	1	-	1	0.4
$t_{3,i}$	0.25	-	1	0.5	0.5	0.9

Station i	7	8	9	10	11	12
Discipline	FCFS	FCFS	FCFS	FCFS	PS	PS
$t_{1,i}$	0.5	1	1.25	-	0.25	0.75
$t_{2,i}$	-	1	1.25	-	2	0.5
$t_{3,i}$	-	-	-	1.5	-	1

Table 22.12: Parameters of the stations of Example 5A.

Throughputs	$N_1 = 5$ $N_2 = 4$ $N_3 = 3$	$N_1 = 6$ $N_2 = 8$ $N_3 = 10$	$N_1 = 14$ $N_2 = 16$ $N_3 = 10$
Class 1	0.3800	0.3427	0.4538
Class 2	0.4372	0.5467	0.6073
Class 3	0.3748	0.5806	0.4997

Table 22.13: Exact throughput of Example 5A.

with the number of stations, the number of classes, and the number of customers per class. A further feature which makes the models even more difficult to analyze is the presence of stations with multiple servers. Unfortunately, the size of most models encountered in real applications makes them intractable using exact computational algorithms. In order to overcome either of the above limitations, much work has been devoted to developing approximation methods that can handle both non-product-form networks and large product-form networks. In the following, we present an extension of the single-class technique presented in Section 22.1 that can efficiently be used for the approximate analysis of both kinds of networks.

22.3.2 Approximate technique

Principle

As in the single-class case, the idea of the method is to estimate the performance of the original network by using approximate product-form solutions. But now, as the original network is visited by R classes of customers, we are going to associate with this network, R single-class product-form networks, each of them corresponding to a particular class of customers. The objective is that a product-form network of a given class r will accurately reflect the behavior of only class-r customers in the initial system. Of course the interactions between class-r and the other classes must be taken into account in the derivation of the product-

	St. 1	St. 2	St. 3	St. 4	St. 5
Class 1	3.314	4.252	6.260	-	2.880
Class 2	1.767	-	5.481	-	2.906
Class 3	0.9130	-	6.266	0.6924	1.490

	St. 6	St. 7	St. 8
Class 1	1.173	0.5499	1.652
Class 2	0.6293	-	1.639
Class 3	1.416	-	-

	St. 9	St. 10	St. 11	St. 12
Class 1	2.444	-	0.5513	4.436
Class 2	2.416	-	4.055	3.123
Class 3	-	5.979	-	5.664

Table 22.14: Mean response times of Example 5A, with $\mathbf{N} = (6, 8, 10)$.

form approximations.

Consider the single-class network associated with a particular class r. With each station i of the original network visited by class-r customers, i.e., $i \in S(r)$, is associated a load-dependent exponential station. Thus, the class-r product-form network consists of $|S(r)|$ stations, where $|S(r)|$ denotes the cardinal of $S(r)$. Let $\mu_{r,i}(n_{r,i})$, $n_{r,i} = 1, ..., N_r$, denote the load-dependent service rates of station i, in the class-r product-form network ($i \in S(r)$). The visit ratios in the r-th isolated network, $v_{r,i}$ for $i = 1, ..., M$, are defined to be exactly those visit ratios for the r-th class in the original network. Since the single-class networks are Gordon-Newell's networks (see Section 22.1.1), the state-probabilities $p(\mathbf{n_r})$ for the r-th single-class network, $r = 1, \ldots, R$, have the following product-form solution:

$$p(\mathbf{n_r}) = p(n_{r,1}, \ldots, n_{r,M}) = \frac{1}{G_r} \prod_{i \in S(r)} \left[\prod_{n=1}^{n_{r,i}} \frac{v_{r,i}}{\mu_{r,i}(n)} \right] \tag{22.15}$$

where G_r is the normalization constant associated with class-r network.

As an illustration, the three product-form networks associated with Example 5 (in Figure 22.18) are shown in Figure 22.19. Again, It is important to note that, the routing of these networks are only shown for picture conveniences, as the only parameters involved in the product-form solutions are mean visit ratios (see Section 22.2.4).

Thus, the performance of the r-th class in the original network is approximated by the performance of the single-class in the r-th single-class network. It remains to specify the load-dependent service rates $\mu_{r,i}(n_{r,i})$, $n_{r,i} = 1, ..., N_r$. Ideally, the service rates $\mu_{r,i}(n_{r,i})$ of the flow equivalent service centers of the class-r product-form network should be equal to the conditional throughputs

$\nu_{r,i}(n_{r,i})$ of class-r customers at the corresponding station (station i) in the original network [20, 7]. The conditional throughput $\nu_{r,i}(n_{r,i})$ is defined as the average flow of class-r customers out of station i, given that $n_{r,i}$ class-r customers are present at the station (regardless of the number of customers of the other classes simultaneously present at the station). However, obtaining the exact values of the conditional throughputs would, in general, require an exact solution of the original system, which is precisely what we want to avoid. Therefore, the idea is to approximate the conditional throughputs. These approximations will be used instead of the exact values in order to obtain the service rates of the flow equivalent service centers. This step constitutes the second level of the approximation of the method.

Of course, in order for the behavior of the customers in the r-th single-class network to be close to that of the class-r customers in the original network, the influence of the other classes has to be accurately reflected in the approximations of the conditional throughputs. Therefore, a crucial issue in this approximate procedure is to approximate the conditional throughputs of the different classes of customers at the different stations of the system. These approximations will be obtained by analyzing each station of the system in isolation.

Analysis in isolation

In order to approximate the conditional throughputs of each class at every station in the original system, the stations are analyzed in isolation. Each station i is analyzed as a queue fed by external state-dependent processes, one for each class of customers. In particular, customers of class r have exponential inter-arrival times with arrival rates $\lambda_{r,i}(n_{r,i})$, $n_{r,i} = 0, \ldots, N_r - 1$, where $n_{r,i}$ is the number of class-r customers present at station i (see Figure 22.20). Let $\tilde{\nu}_{r,i}(n_{r,i})$ be the conditional throughput (output rate) of class-r customers at this isolated queue, when there are $n_{r,i}$ class-r customers present (and again, whatever are the numbers of customers of the other classes). These conditional throughputs will provide approximations of the exact conditional throughputs $\nu_{r,i}(n_{r,i})$ of class-r customers at station i in the original system.

In order to get these estimated conditional throughputs we are going to follow the same lines as for the basic single-class technique. So we first suppose that we know the arrival rates for the different classes that visit a given station i, that are the $\lambda_{r,i}(n_{r,i})$ for all $r \in R(i)$, and $n_{r,i} = 0, \ldots, N_r - 1$. Station i can then be analyzed in isolation using any appropriate technique. We will come back on that crucial point in the following section. For each class of customers, the marginal probabilities $\tilde{p}_{r,i}(n_{r,i})$ that $n_{r,i}$ customers are present in the isolated station, $n_{r,i} = 0, \ldots, N_r$, can then be derived. Then, the conditional throughputs $\tilde{\nu}_{r,i}(n_{r,i})$ for the isolated station can be obtained by means of relation 22.16 [7]. Indeed, if we consider a particular class of customers that visit station i, that is a class r such that $r \in R(i)$, station i in isolation can be viewed as a single-class queue, in which the customers that arrive are the class-r customers of the initial system. Customers of the other classes must be considered as entities that interact and slow down class-r customers. Then, the

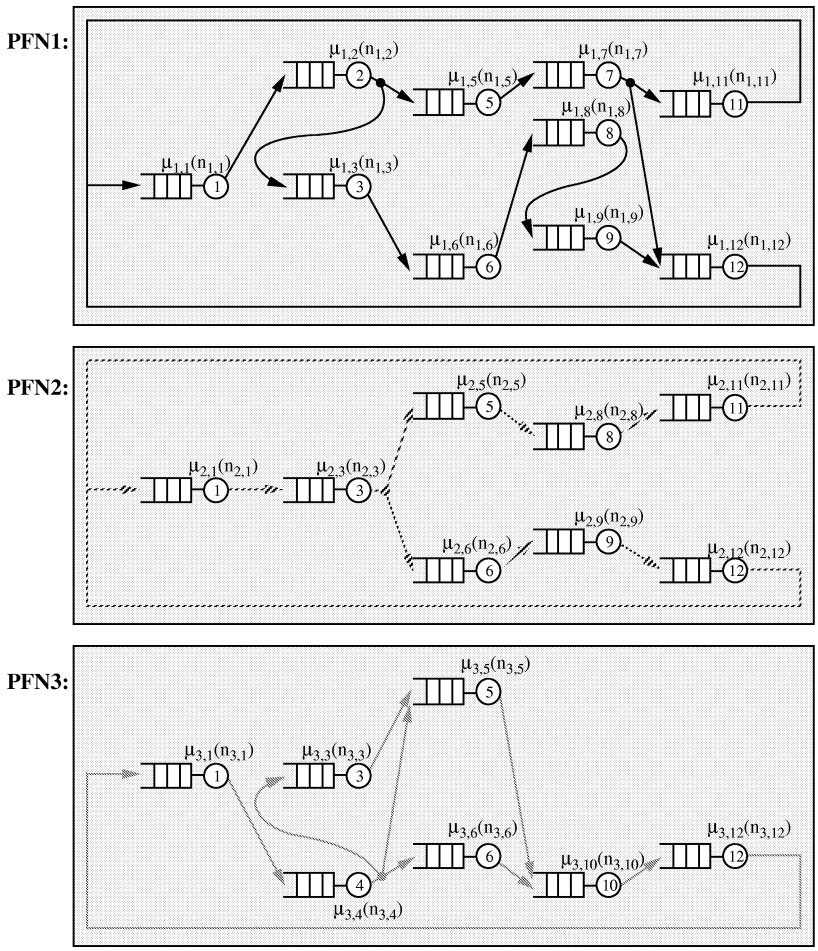


Figure 22.19: Single-class product-form networks associated with Example 5.

marginal probabilities $\tilde{p}_{r,i}(n_{r,i})$ represent the probabilities for this single-class queue to be in state $n_{r,i}$. Therefore, the conditional throughputs $\tilde{\nu}_{r,i}(n_{r,i})$ can be obtained by means of the single-class relation 22.4. By the way, note that relation 22.16 is exactly the same as relation 22.4 with some additional class indexes. Of course this relation has to be applied $|R(i)|$ times in order to get the approximate conditional throughputs of all the classes of customers that visit station i, i.e., the $\tilde{\nu}_{r,i}(n_{r,i})$ for all $r \in R(i)$.

$$\tilde{\nu}_{r,i}(n_{r,i}) = \lambda_{r,i}(n_{r,i}-1)\frac{\tilde{p}_{r,i}(n_{r,i}-1)}{\tilde{p}_{r,i}(n_{r,i})} \qquad \text{for } n_{r,i} = 1,\ldots,N_r \tag{22.16}$$

The principle of the method is then to set the load-dependent service rates of all the stations in the different single-class product-form networks, associated

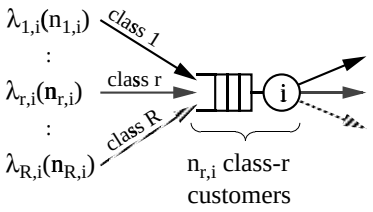


Figure 22.20: Analysis in isolation of station i.

with station i, equal to these approximated conditional throughputs, i.e.:

$$\mu_{r,i}(n_{r,i}) = \tilde{\nu}_{r,i}(n_{r,i}) \qquad \text{for all } r \in R(i) \text{ and } n_{r,i} = 1, \ldots, N_r \tag{22.17}$$

Now, in order to use this approach, the state-dependent arrival rates $\lambda_{r,i}(n_{r,i})$ of each class of customers at the different stations are required. It turns out that these quantities can be obtained from the product-form solutions of the single-class networks given by 22.15. Indeed, the arrival rates of a given class r of customers at station i ($r \in R(i)$), i.e., $\lambda_{r,i}(n_{r,i})$ for $n_{r,i} = 0, \ldots, N_r - 1$, only depend on the number of class-r customers currently present at this station. Provided that the $\mu_{r,i}(n_{r,i})$ are given, these state-dependent arrival rates can be obtained from the expression of the product-form solution of the class-r single-class network (see Section 22.1.4):

$$\lambda_{r,i}(n_{r,i}) = \mu_{r,i}(n_{r,i}+1)\frac{p_{r,i}(n_{r,i}+1)}{p_{r,i}(n_{r,i})} \qquad \text{for } n_{r,i} = 0, \ldots, N_r - 1 \tag{22.18}$$

Algorithm

As in the single-class case, the service rates $\mu_{r,i}(n_{r,i})$ are solutions of a fixed-point problem defined by the set of equations 22.16, 22.17 and 22.18. The number of unknowns is now: $\prod_{r=1}^{R} |S(r)|N_r$, where $|S(r)|$ is the number of stations of the class-r product-form network. An iterative procedure is used to determine these quantities. This procedure is described by means of Algorithm 22.3.

The remarks that have been made on the basic single-class algorithm 22.1 (see Section 22.1.5), remain true here. In addition, we must note that the solution of product-form networks (Steps 1 and 5) can easily be obtained since the only networks involved in our method are single-class networks. So, in general, the main computational burden of this method is due to the analysis of the stations in isolation. This issue is investigated in the next section. Finally we must say that the number of iterations to reach the convergence is, as in the single-class case, very low (of the order of 10 on most examples) and that it is still possible to reduce the number of iterations of the algorithm by using the

Algorithm 22.3 Multi-class technique

Step 0. For $r = 1, \ldots, R$:
Set the $\mu_{r,i}(n_{r,i})$ to some initial values, for example $\frac{1}{t_{r,i}}$,
for all $i \in S(r)$ and $n_{r,i} = 1, \ldots, N_r$.
Step 1. For $i = 1, \ldots, M$:
Calculate the state-dependent arrival rates $\lambda_{r,i}(n_{r,i})$,
for all $r \in R(i)$ and $n_{r,i} = 0, \ldots, N_r - 1$, from relation 22.18.
Step 2. For $i = 1, \ldots, M$:
a. Analyze station i in isolation.
b. Derive the marginal probabilities $\tilde{p}_{r,i}(n_{r,i})$ of this isolated station,
for all $r \in R(i)$ and $n_{r,i} = 0, \ldots, N_r$.
c. Calculate the conditional throughputs $\tilde{\nu}_{r,i}(n_{r,i})$ of this isolated
station, for all $r \in R(i)$ and $n_{r,i} = 1, \ldots, N_r$, from relation 22.16.
Step 3. For $r = 1, \ldots, R$:
Update the load-dependent service rates of station i, for all $i \in S(r)$,
in the r-th product-form network:
$\mu_i(n_i) = \tilde{\nu}_i(n_i)$, for all $n_{r,i} = 1, \ldots, N_r$.
Step 4. Go to Step 1 until convergence of the parameters $\mu_i(n_i)$.
Step 5. For $r = 1, \ldots, R$:
Calculate the average performance parameters of class-r customers
from the product-form expression of the class-r product-form network.

most updated values of the load-dependent arrival rates when analyzing it in isolation (see Section 22.1.5).

22.3.3 Analysis of a station in isolation

As it appears in the previous section, a crucial step in this method is the analysis of each station in isolation. As a matter of fact, analyzing a multi-class station in isolation constitutes the most costly part of the technique in terms of computational complexity. Indeed, as discussed in the previous section, in between the analysis in isolation of the different stations, one has to handle with calculations in single-class product-form networks that have very low computational requirements. In most of the cases, the cost of calculating the state-dependent arrival rates to each station, is thus insignificant compared to the cost of analyzing a station in isolation. That is why, focusing on this problem and trying to reduce as much as possible the cost of the analysis in isolation, is so important.

The goal of the analysis of a given station, say station i, is to obtain the marginal probabilities $\tilde{p}_{r,i}(n_{r,i})$ that $n_{r,i}$ class-r customers are present in the queue for all possible values of $n_{r,i}$, i.e., $n_{r,i} = 0, \ldots, N_r$, and this for each class r that visits station i, i.e., $r \in R(i)$. From these marginal probabilities, the conditional throughputs $\tilde{\nu}_{r,i}(n_{r,i})$ of class-r customers at station i can then be derived using Equation 22.16.

An alternative and equivalent way of looking at this system is to notice that the isolated multi-class station is equivalent to a closed network containing

$|R(i)|+1$ stations, namely station i plus one extra station per class of customers that visit station i. This equivalent view of the analysis in isolation of a given station i is illustrated in Figure 22.21. Each of the extra stations of this equivalent closed network models the arrival process of a particular class of customers at station i. These stations are single-class queues, as they are only visited by a particular class of customers. Moreover, they have exponentially distributed service-times with load-dependent service rates given by a relation similar to Relation 22.10.

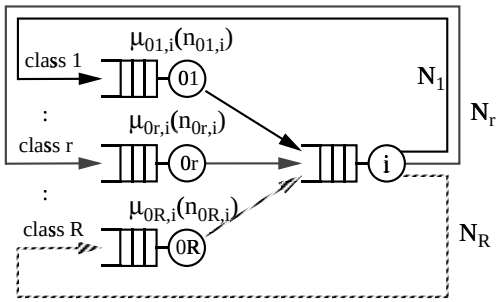


Figure 22.21: Equivalent closed view of the analysis in isolation of station i.

Ideally, an exact solution of each station in isolation should be derived. A typical case where an exact solution can efficiently be obtained is when the station is BCMP. This issue is described in the first part of this section. However, in certain cases, obtaining an exact solution would remain too costly. In these cases, the only alternative is to obtain an approximate solution. That is, instead of deriving exact values of the marginal probabilities, approximations of these quantities are used. The example of the approximate analysis of a FIFO (and non-BCMP) queue is given in the second part of this section. Finally a class-aggregation technique has been proposed in [7] in order to reduce tremendously the computational effort associated with the analyses in isolation. This technique is presented in the next section.

BCMP station

A typical example for which an exact analysis is feasible, is when station i is a BCMP station [8]. The application of the multi-class technique presented in the previous section to BCMP networks has been developed in [2]. It has been shown that very efficient computational algorithms can be developed for the analysis in isolation of a BCMP station, which make the general method attractive for product-form networks. Here we briefly present the general ideas of this work. For any technical details and algorithms see [2].

The key idea is that the analysis in isolation of a BCMP station constitutes a special case of a BCMP network. Indeed, in the equivalent closed view of the analysis in isolation of station i (see Figure 22.21) all the stations are BCMP: station i, by assumption, but also all the $|R(i)|$ extra stations that are single-class

exponential stations with state-dependent rates. Thus the marginal probabilities $\tilde{p}_{r,i}(n_{r,i})$ required by the overall method have the following product-form expression (see [2]):

$$\tilde{p}_{r,i}(n_{r,i}) = \frac{f_{r,i}(n_{r,i})}{G} \sum_{n=0}^{N-N_r} \frac{(n+n_{r,i})!}{\prod_{k=1}^{n+n_{r,i}} \min(k, C_i)} F_i^r(n) \qquad (22.19)$$

where

$$f_{r,i}(n_{r,i}) = \frac{\prod_{n=0}^{n_{r,i}-1} \lambda_{r,i}(n)}{n_{r,i}! {\mu_{r,i}}^{n_{r,i}}}$$

and

$$F_i^r = \bigotimes_{s \in R(i),\ s \neq r} f_{s,i}$$

The quantities $F_i^r(n)$, $n = 0, \ldots, N - N_r$, can be calculated by successive convolutions on the $\{f_{s,i}\}_{s \in R(i),\ s \neq r}$. The marginal probabilities $\tilde{p}_{r,i}(n_{r,i})$ can then be derived using Equation 22.19. Note that a normalization constant G appears in equation 22.19. There is however no need to calculate G since we only need to obtain the values $\tilde{p}_{r,i}(n_{r,i})$ up to a multiplicative constant. Indeed, our final goal is to derive the conditional throughputs $\tilde{\nu}_{r,i}(n_{r,i})$, which only involves the ratio of two marginal probabilities, namely $\tilde{p}_{r,i}(n_{r,i}-1)/\tilde{p}_{r,i}(n_{r,i})$. Finally it has been shown in [2] that the computational effort to solve this isolated system is $O(R^3 \bar{N}^2)$, where $\bar{N} = \max_{r \in R(i)} N_r$.

As stated before, the multi-class technique presented in Section 22.3.2 was not originally devoted to the analysis of BCMP networks, as the primary purpose was to provide a general framework for the analysis of general multi-class closed queueing networks. However it is interesting to see that, by using this convolution procedure to analyze each station in isolation, the technique competes favorably with the existing techniques that are only devoted to the approximate analysis of BCMP networks (see [2]). In addition it was important to test the accuracy of the technique when exact analyses in isolation are run, in order to estimate the basic error of the technique, that is the error generated by the use of single-class product-form approximations.

As an illustration of the accuracy, we have tested Example 5A which exact performance parameters are given in Tables 22.13 and 22.14. Tables 22.15 shows the results pertaining to the throughput of each class of customers (measured at station 1), for the three different population-vectors (5, 4, 3), (6, 8, 10) and (14, 16, 10). Table 22.16 shows the results pertaining to the mean response-time of each class of customers at the different stations, for a single population-vector (6, 8, 10). Both tables give the relative error with respect to the exact values. It appears that the results obtained by our method are quite accurate: there

is less than 1% error for the throughputs and less than 5% errors for the mean response-time. More examples are given in [2] and similar results were obtained.

Throughputs	$N_1 = 5$ $N_2 = 4$ $N_3 = 3$	$N_1 = 6$ $N_2 = 8$ $N_3 = 10$	$N_1 = 14$ $N_2 = 16$ $N_3 = 10$
Class 1	0.3813 +0.3 %	0.3460 +1.0 %	0.4536 0 %
Class 2	0.4417 +1.0 %	0.5520 +1.0 %	0.6097 +0.4 %
Class 3	0.3777 +0.8 %	0.5811 +0.9 %	0.5026 +0.6 %

Table 22.15: Approximate throughput of Example 5A.

Non-BCMP stations

We now consider the case where station i is not a BCMP station. This is the case for example when station i is either a FIFO queue with non-exponential and/or class-dependent service-time distribution, or has a scheduling discipline which is of none of the BCMP types. As an example, station i may be a priority queue. In this case some classes of customers have priority over other classes and bypass them in the buffer of the station. In addition one has to define when a high-priority customer arrive, if this last preempt the service (PR queue) of the lower-priority customer in service, or is put Head On Line (HOL queue) in the buffer to be the next customer to be served, provided no new higher-priority customer arrive within the beginning of his service.

The analysis in isolation of station i no longer has a product-form expression. Now provided that the service-time distribution of each class of customers is represented as a Phase-Type (PH) distribution, it is always possible to obtain an exact solution of the multi-class station by solving the underlying Markov chain. This can be achieved using any appropriate numerical technique [18]. Such a numerical solution will be feasible only if the number of states of the Markov chain is not too large. This is the case for priority queues visited by few classes. Note that for two classes one can use a direct technique, as described in [1].

Now the number of states of the Markov chain associated with the analysis in isolation of a given station increases very rapidly with the number of classes and the number of customers of each class. This is specially true for a FIFO queue. Consider for instance a FIFO queue with class-dependent exponential service-time distributions. The state-vector is $(x_1, \ldots, x_n)$ where x_j is the class of the customer who is the j-th position in the queue and n is the total number of customers currently present. If station i is visited by 4 classes of customers and if the population vector is $\mathbf{N} = (N_1, N_2, N_3, N_4) = (5, 4, 3, 2)$, the Markov chain associated with the analysis of station i in isolation has 5,783,545 states.

	St. 1	St. 2	St. 3	St. 4	St. 5
Class 1	3.252 -1.9 %	4.380 +3.0 %	6.014 -3.9 %	-	2.812 -2.4 %
Class 2	1.721 -2.7 %	-	5.447 -0.6 %	-	2.849 -2.0 %
Class 3	0.8878 -2.8 %	-	6.139 -2.0 %	0.6937 +0.2 %	1.460 -2.0 %

	St. 6	St. 7	St. 8
Class 1	1.168 -0.4 %	0.5512 +0.2 %	1.645 -0.4 %
Class 2	0.6270 -0.4 %	-	1.641 +0.1 %
Class 3	1.414 -0.1 %	-	-

	St. 9	St. 10	St. 11	St. 12
Class 1	2.428 -0.7 %	-	0.5463 -0.9 %	4.320 -2.6 %
Class 2	2.417 0 %	-	4.125 +1.7 %	2.988 -4.0 %
Class 3	-	6.132 +2.6 %	-	5.593 -0.2 %

Table 22.16: Approximate mean response times of Example 5A, with $\mathbf{N} = (6, 8, 10)$.

None of the numerical techniques available can solve such a large Markov chain. Thus, exact analysis of FIFO queues will not in general be achievable. In this case, the only viable alternative is to derive an approximate solution of the multi-class queue. In order to reduce the complexity of the analysis, the idea is to transform the multi-class station in such a way that the behavior of the new station is close to that of the original station and its analysis is much easier.

A first attempt to reduce the complexity of the analysis is to replace the FIFO scheduling discipline by a RANDOM scheduling discipline. In a RANDOM discipline, the next customer to be served is chosen randomly among the customers currently present in the queue. This means that the probability for a class-r customer to receive service when the server becomes available is proportional to the number of class-r customers waiting in the queue. This transformation is supported by two theoretical justifications. On the one hand, in a BCMP network, a FIFO queue with exponential class-independent service-time distribution can be replaced exactly by a RANDOM queue having the same service time distribution. On the other hand, it has been proved that in the case of an open FIFO queue having class-dependent general service-time distributions and Poisson arrival processes for all the classes, the transformation of the FIFO scheduling discipline into a RANDOM is again exact [14]. Moreover, Marie and

Rubino [14] noticed that replacing a FIFO queue by a RANDOM queue having the same service-time distribution, in a closed network, is in general a good approximation.

This first approximation leads to a significant reduction in the complexity of the analysis of the Markov chain associated with station i. Indeed, by transforming the FIFO discipline into a RANDOM one, the state-vector of the Markov chain is simply given by $(n_1, \ldots, n_r, \ldots, n_R, s)$ where n_r is the number of class-r customers in the queue and s is the class of customer currently in service. Now, let us consider again the previous example of a FIFO queue visited by 4 classes and a population vector $\mathbf{N} = (N_1, N_2, N_3, N_4) = (5, 4, 3, 2)$. The number of states of the associated RANDOM queue has been drastically reduced from 5,783,545 to 1,099. Now, Markov chains with about a thousand of states are fairly easy to solve by means of numerical techniques. However, if the number of classes and/or the number of customers per class is increased, the number of states of the Markov chain associated with the FIFO queue will significantly increase. As a result, the RANDOM multi-class queue may again become too costly to solve exactly. In the next section, we present a further approximation that enables us to significantly reduce the cost of the analysis in isolation of a given station. It is presented in the case of FIFO or RANDOM queues. However, it extends easily to priority queues as briefly discussed at the end of the next section.

In order to illustrate the results obtained by the proposed method, we consider again Example 5 illustrated in Figure 22.18 with parameters given in Table 22.17. This example will be referred to as Example 5B. The network consists of different types of stations: station 1 has FIFO discipline and class-dependent exponential service-time distribution; stations 5 and 6 have FIFO discipline and class-dependent general service-time distributions given as Coxian-2 distributions; stations 11 and 12 have PR discipline and class-dependent exponential service-time distributions; at station 11, class 1 has priority over class 2, while at station 12, class 1 has priority over class 2 and 3, and class 2 has priority over class 3; all the other stations have FIFO discipline and exponential service-time distributions (BCMP stations). We again tested three different configurations corresponding to three different population vectors: (5, 4, 3), (6, 8, 10) and (14, 16, 10). Tables 22.18 shows the results pertaining to the throughput of each class of customers (measured at station 1), for the three different population-vectors while Table 22.19 shows the results pertaining to the mean response-time of each class of customers at the different stations, for the single population-vector (6, 8, 10). Both tables give the simulation results and the approximate values when an exact analysis in isolation of each station is done and when the class-aggregation technique developed in the next section is used. It appears that the results obtained by our method are still accurate: there is less than 5% error for the throughputs and less than 20% errors for the mean response-time. More examples are given in [7] and similar results were obtained.

Mean service-time	St. 1 FIFO non-BCMP Expo.	St. 2 FIFO BCMP	St. 3 FIFO BCMP	St. 4 FIFO BCMP
Class 1	1	2	1	-
Class 2	0.5	-	1	-
Class 3	0.25	-	1	0.5

Cox-2 par. (μ_1, a_1, μ_2) $(Mean, cv^2)$	St. 5 FIFO non-BCMP Cox-2 distr.	St. 6 FIFO non-BCMP Cox-2 distr.
Class 1	(2, 1, 2) (1, 0.5)	(4, 0.1, 0.2) (0.75, 8.55)
Class 2	(2, 0.1, 0.2) (1, 5)	(5, 0.2, 1) (0.4, 2.5)
Class 3	(5, 0.3, 1) (0.5, 2.2)	(2, 0.2, 0.5) (0.9, 2.1)

Mean service-time	St. 7 FIFO BCMP	St. 8 FIFO BCMP	St. 9 FIFO BCMP	St. 10 FIFO BCMP	St. 11 PR non-BCMP	St. 12 PR non-BCMP
Class 1	0.5	1	1.25	-	0.25	0.75
Class 2	-	1	1.25	-	2	0.5
Class 3	-	-	-	1.5	-	1

Table 22.17: Parameters of the stations of Example 5B.

22.3.4 Class-aggregation technique

A class-aggregation technique has been proposed in [7] to approximately analyze a station in isolation. The idea of this technique is to reduce the analysis in isolation of a given station i fed by the arrival of $|R(i)|$ classes of customers, by $|R(i)|$ analyses in isolation of a station fed by the arrival of only two classes of customers. It has been shown that this technique allows us to reduce tremendously the computational effort associated with the analyses in isolation and thus the computational effort of the overall method, and introduce only an insignificant additional error. The basic ideas of this class-aggregation technique are briefly presented in this section.

To understand the general principle of this technique let us consider a particular class of customers, say r, that visit station i, that is $r \in R(i)$. With respect to class-r customers, the customers of the other classes can be considered as entities that slow down the traffic of class-r customers. Now, in a RANDOM (or FIFO) queue, the customers of the other classes have the same effect on the behavior of class-r customers, with respect to the scheduling discipline. The idea is then to aggregate all these other classes into an equivalent aggregate class, referred to as class-a^r. In other words the original multi-class station hav-

Throughputs	$N_1 = 5$	$N_2 = 4$	$N_3 = 3$	$N_1 = 6$	$N_2 = 8$	$N_3 = 10$
	Simul.	Exact Anal. in Isol.	Class-aggreg.	Simul.	Exact Anal. in Isol.	Class-aggreg.
Class 1	0.3969	0.3981 +0.3 %	0.3983 +0.3 %	0.3847	0.3854 +0.2 %	0.3861 +0.4 %
Class 2	0.4059	0.4206 +3.6 %	0.4212 +3.7 %	0.5267	0.5412 +2.8 %	0.5402 +2.5 %
Class 3	0.2948	0.3154 +7.0 %	0.3161 +7.2 %	0.5056	0.5226 +3.4 %	0.5231 +3.4 %

Throughputs	$N_1 = 14$	$N_2 = 16$	$N_3 = 10$
	Simul.	Exact Anal. in Isol.	Class-aggreg.
Class 1	0.4763	0.4729 -0.7 %	0.4734 -0.6
Class 2	0.6084	0.6164 +1.3 %	0.6172 +1.4
Class 3	0.4058	0.4266 +5.1 %	0.4278 +5.4

Table 22.18: Approximate throughputs of Example 5B.

ing $|R(i)|$ classes of customers is transformed into an aggregate station having two classes of customers, namely class r and class a^r (see Figure 22.22).

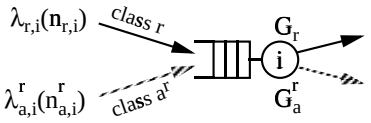


Figure 22.22: Illustration of the class-aggregation technique

Before proceeding further, it is worthwhile noticing that this transformation has to be done for all the classes in turn. Indeed, the aggregate station with respect to class-r customers will enable us to obtain approximate values for the marginal probabilities of only class-r customers, that is the $\tilde{p}_{r,i}(n_{r,i})$ for $n_{r,i} = 0, \ldots, N_r$. Thus, this transformation has to be performed $|R(i)|$ times, each one with respect to a particular class of customers.

We now describe the characterization of the aggregate station and quickly show how its parameters can be determined. (Technical details can be found in [7].) Consider again any class $r \in R(i)$ and consider the aggregate station with respect to class r. The arrival process and service time distribution of class-r customers are the same as in the original multi-class station. The scheduling

discipline is also the same as in the original multi-class station, namely in our case a RANDOM discipline. Thus, we only need to characterize the arrival and service processes of the customers of the aggregate class.

Let us first consider the arrival process of the aggregate class. The arrival of class-a^r customers is assumed to be a Markovian process whose rate $\lambda^r_{a,i}(n^r_{a,i})$ depends on the total number $n^r_{a,i}$ of class-a^r customers currently present in the queue. Note that the maximum number of customers of class a^r that can be simultaneously present is given by:

$$N^r_a = \sum_{s \in R(i),\ s \neq r} N_s$$

Therefore, $\lambda^r_{a,i}(N^r_a) = 0$. We thus have to determine the quantities $\lambda^r_{a,i}(n^r_{a,i})$, for $n^r_{a,i} = 0, \ldots, N^r_a - 1$. In order to determine these quantities, we need to interpret a population of $n^r_{a,i}$ customers of class a^r as consisting of a number of customers of the different classes other than class r. Let $\mathbf{n^r} = [n_s]_{s \in R(i),\ s \neq r}$ denote the state-vector representing the number of customers of all the classes (that visit station i) but class r. Let $E^r(n^r_{a,i})$ denote the set of all possible vectors $\mathbf{n^r}$ such that the sum of their components are equal to $n^r_{a,i}$. Let $P_i(\mathbf{n^r}|n^r_{a,i})$ be the conditional probability that the classes other than class r are in state $\mathbf{n^r}$ given that the total number of customers in these classes is $n^r_{a,i}$. It has been shown in [7] that the $\lambda^r_{a,i}(n^r_{a,i})$ can be obtained by the following relation:

$$\lambda^r_{a,i}(n^r_{a,i}) = \sum_{\mathbf{n^r} \in E^r(n^r_{a,i})} \left[\sum_{s \in R(i),\ s \neq r} \lambda_{s,i}(n_{s,i}) \right] P_i(\mathbf{n^r}|n^r_{a,i})$$
$$\text{for } n^r_{a,i} = 0, \ldots, N^r_a - 1 \qquad (22.20)$$

The sum is over all the possible arrangements of the $n^r_{a,i}$ customers. For each arrangement, the global arrival rate equal to the sum of the corresponding rates, $\lambda_{s,i}(n_{s,i})$ for $s \in R(i)$ and $s \neq r$, weighted by the conditional probability of this arrangement. This conditional probability can equivalently be written as:

$$P_i(\mathbf{n^r}|n^r_{a,i}) = \frac{P_i(\mathbf{n^r})}{P^r_{a,i}(n^r_{a,i})}$$

where

$$P^r_{a,i}(n^r_{a,i}) = \sum_{\mathbf{n^r} \in E^r(n^r_{a,i})} P_i(\mathbf{n^r})$$

In this expression $P_i(\mathbf{n^r})$ is the detailed probability of the distribution of the different classes of customers (other than class r) in station i and $P^r_{a,i}(n^r_{a,i})$ is the marginal probability of class-a^r customers. Of course, the actual values of $P_i(\mathbf{n^r})$ are unknown. We approximate these quantities by the product of the marginal probabilities $P_{s,i}(n_{s,i})$ of each class s that visits station i other than class r:

$$P_i(\mathbf{n^r}) = \prod_{s \in R(i),\ s \neq r} P_{s,i}(n_{s,i})$$

Of course, the probabilities $P_{s,i}(n_{s,i})$ are performance parameters for the system and are thus unknown. Since our general algorithm is iterative, one can use the most recent values of the marginal probabilities $P_{s,i}(n_{s,i})$ calculated at the previous step of the algorithm. Now in order to avoid the enumeration in the multiple sum in 22.20, it has been shown in [2] that the arrival rates of the aggregate classes can be calculated using relation 22.21:

$$\lambda^r_{a,i}(n^r_{a,i}) = \frac{1}{q^r_i(n^r_{a,i})} \sum_{s \neq r} \sum_{n_s = \max(0, n^r_{a,i} - N^r_a + N_s)}^{\min(n^r_{a,i}, N_s)} \lambda_{s,i}(n_{s,i}) P_{s,i}(n_{s,i}) q_i^{r,s}(n^r_{a,i} - n_{s,i})$$

$$\text{for } n^r_{a,i} = 0, \ldots, N^r_a - 1 \tag{22.21}$$

with

$$q^r_i = \bigotimes_{s \in R(i),\ s \neq r} P_{s,i} \text{ and } q_i^{r,s} = \bigotimes_{t \in R(i),\ t \neq r,s} P_{s,i}$$

Let us now turn our attention to the characterization of the service times distribution of the aggregate class. Let G_r and $\{G_s\}_{s \in R(i),\ s \neq r}$ denote the service time distributions of class r and the other classes that visit station i. Let G^r_a be the service time distribution of the r-th aggregate class. G^r_a can be described as a probabilistic combination of the distribution of the classes different from class r, as shown in Figure 22.23. In this Figure p^r_s is the probability for a customer of the aggregate class entering the service stage to be actually a class-s customer. The probabilities p^r_s can be derived using relation 22.22:

$$p^r_s = \frac{X_{s,i}}{\sum_{t \in R(i),\ t \neq r} X_{t,i}} \tag{22.22}$$

where $X_{t,i}$ denotes the throughput of class-t customers at station i. Again, since our general algorithm is an iterative algorithm, the values of p^r_s can be calculated using the most recent values of the throughputs that have been calculated at the previous step.

In order to illustrate the reduction in terms of computational cost achieved by the class-aggregation technique, we consider an example of a RANDOM queue visited by R classes of customers and having class-dependent exponential service times. Each class has 10 customers. In the case of an exact analysis, the state-vector is that described in Section 22.3.3. Using the class-aggregation technique, each of the aggregation station is visited by two classes: the first class has 10 customers and the second one has $10(R-1)$ customers. The state vector associated with the Markov chain is now (n_r, n_a, s) where n_r is the number of

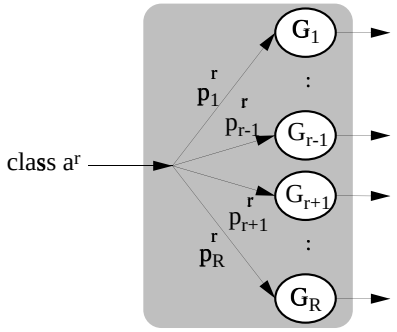


Figure 22.23: General service-time distribution G_a^r of the aggregate class a^r

class-r customers in the queue, n_a is the number of customers of the corresponding aggregate class. The parameters s can take any value within $R(i)$. $s = r$ means that the customer in service is a class-r customer. $s = t$ $(t \neq r)$ means that the customer in service is a customer of the aggregate class and furthermore that its exponential service time distribution is that of a customer of class t of the original multi-class queue. Table 22.20 gives the number of states of the Markov chain associated with the exact analysis of the original multi-class station and of that associated with any of the aggregate station. The number of classes $|R(i)|$ varies from 1 to 6.

It is clear that as the number of classes increases, the number of states of the Markov chain associated with the aggregate station becomes much smaller than that of the Markov chain associated with the original station. As a result, although it has to be performed $|R(i)|$ times, the numerical analysis of the aggregate station is much simpler than that of the original station. Indeed, for a RANDOM queue visited by 6 classes of customers and 10 customers by class, the approximation enables us to replace a single analysis of a Markov chain having nearly 10 million states by 6 successive analyses of a Markov chain with a little more than 3 000 states. And as illustrated on Example 5B (see Tables 22.18 and 22.19), this does not introduce any significant error in the overall method. Finally, it is worth noticing that the computational complexity could further be reduced by replacing the general service time distribution G_a^r of the aggregate class a^r by a simple PH distribution (e.g., Erlang or Cox2) having the same first two (or three) moments.

22.4 Multi-class technique for synchronized networks

In this section we extend the results from the previous section by adding some synchronization stations in between the different classes of customers. We again consider a multi-class closed queueing network consisting of M stations, R

classes of customers and a constant number N_r of class-r customers. (Customers are still not allowed to change class.) But now a station is either a service center (a multi-class station as described in the previous sections) or a synchronization station between different classes of customers. Figure 22.26 gives an illustration of a synchronization station involving four classes of customers. It has a dedicated buffer for each class. A customer of each class is released simultaneously through the synchronization station as soon as there is at least one customer in each of the input buffers of the synchronization station. As a consequence the only feasible states of this synchronization station are states where at least one input buffer is empty.

Let S be the number of synchronization stations and L the number of service centers ($S+L = M$). Without lost of generality, we assume that synchronization stations are indexed from 1 to S and service centers from $S + 1$ to M.

Such networks will be referred to as multi-class synchronized closed queueing networks (MSCQNs). An example of a MSCQN is shown in Figure 22.24 and will be referred to as Example 6. There are $R = 4$ classes of customers and $M = 9$ stations divided up into $S = 2$ synchronization stations (stations 1 and 2) and $L = 7$ service centers (stations 3 to 9). In this example the service center stations are only visited by a single class of customers. Synchronization station 1 involves two classes of customers (classes 1 and 2), while synchronization station 2 involves three classes of customers (classes 2, 3 and 4). Figure 22.24 also shows the routing of the customers through the network. For instance, a customer completing its service at station 6 has a probability 0.5 to be sent to synchronization station 2 and a probability 0.5 to be sent to station 9.

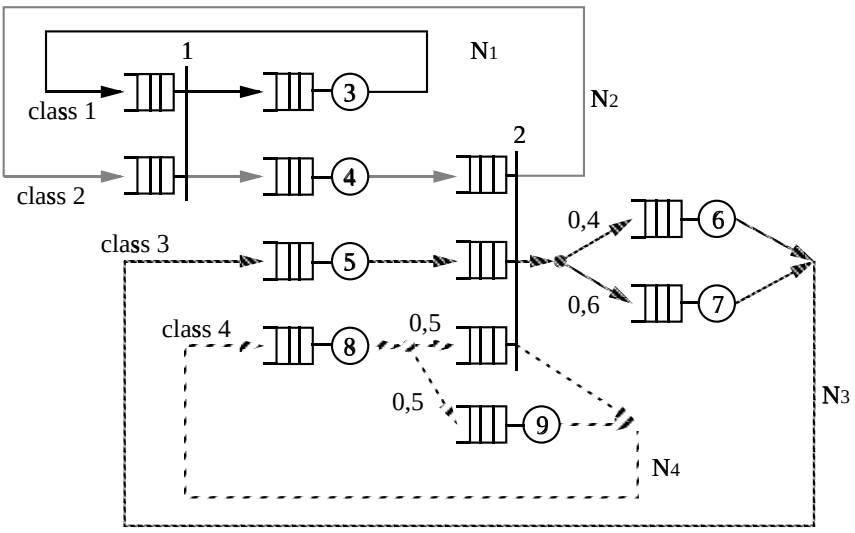


Figure 22.24: Example 6.

22.4.1 Restriction on the Model

Before proceeding further it is important to note that the steady-state behavior of a MSCQN may depend on the initial distribution of the customers in the network (initial state). As an illustration let us consider the network shown in Figure 22.25. There are two classes of customers and two synchronization stations, both between customers of class 1 and class 2. If the initial state is such that all customers of class 1 are in synchronization station 1 and all customers of class 2 are in synchronization station 2, it is clear that the system is in a deadlock. Actually, class-1 customers are blocked at synchronization station 1 awaiting arrivals of class-2 customers and class-2 customers are blocked at synchronization station 2 awaiting arrivals of class-1 customers. If the initial state is such that all customers of class 1 are in station 3 and all customers of class 2 are in station 4, it is easy to show that the network is alive. Moreover, for both classes, any placement of the customers among the different stations is reachable. Of course, any initial placement between this two extremes schemes will lead to a different stationary behavior.

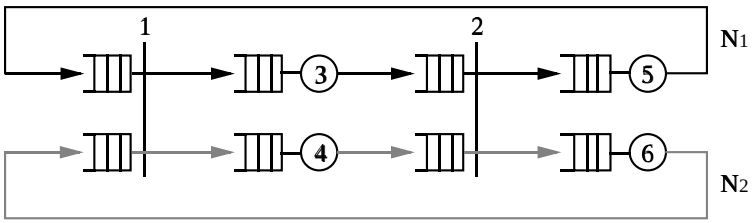


Figure 22.25: Illustration of a MSCQN whose behavior depends on the initial distribution of the customers.

The general product-form approximation technique presented in Section 22.3 doesn't take into account the initial state. Indeed, the performance parameters calculated using this technique do only depend on the populations of the different classes of customers but are otherwise independent of the initial state of the system. Thus the class of MSCQNs solvable using this technique must be restricted. This restriction is presented in detail in [6]. A *simple* MSCQN is then defined as a MSCQN whose steady-state performance parameters do not depend on the initial distribution of the N_r customers of each class r among the stations belonging to $S(r)$. A very simple procedure is described in [6] in order to check whether a MSCQN is simple or not. The idea behind this procedure is to see if there exists an initial distribution that may lead to a deadlock. To do so we only need to consider the synchronization stations and check if there is a cycle in between the classes that are involved in the synchronization stations. As an illustration, for the example in Figure 22.25 there are two synchronization stations 1/2 (between class 1 and class 2), and thus a cycle 1-2-1. This network is not simple. In Example 6 there are two synchronization stations, 1/2 and 2/3/4. Obviously this does not create any cycle in between classes 1, 2, 3 and 4. Example 6 is thus a simple MSCQN. Now if we add to Example 6 a

synchronization station 4/1, we create a cycle 1-2-3-4-1 and the network is not simple anymore.

22.4.2 Application of the multi-class technique

In this section, we apply the principle of the basic multi-class technique described in Section 22.3 to the class of simple MSCQNs. To do so we first associate with this network, R single-class product-form networks, each of them corresponding to a particular class of customers. The class-r product-form network still consists of $|S(r)|$ stations, where $S(r)$ is the set of stations of the original network visited by class-r customers, this set including now the synchronization stations. In order to estimate the parameters of each of the product-form networks, one has to analyze each station in isolation as described in Section 22.3.3. Recall that the aim of the analysis in isolation of a given station i, is to obtain the marginal probabilities $\tilde{p}_{r,i}(n_{r,i})$ that $n_{r,i}$ class-r customers are present in the station for all possible values of $n_{r,i}$, i.e., $n_{r,i} = 0, ..., N_r$, and this for each class r that visits station i, i.e., $r \in R(i)$. From these marginal probabilities, the conditional throughputs $\tilde{\nu}_{r,i}(n_{r,i})$ of class-r customers at station i can then be derived using equation 22.16.

Now in a (simple) MSCQN a station is either a multi-class service center or a synchronization station between different classes of customers. The analysis in isolation of a multi-class service center i has already been discussed in Sections 22.3.3 and 22.3.4. We now only focus on the analysis in isolation of a given synchronization station i synchronizing the set of classes in $R(i)$.

22.4.3 Exact analysis in isolation of a synchronization station

For the sake of simplicity, let us assume that synchronization station i is visited by the first $|R(i)|$ classes of customers, that is every class r, $r = 1, \ldots, |R(i)|$. Customers of class r arrive to this queue according to a state-dependent Markovian process with arrival rates $\lambda_{r,i}(n_{r,i})$, $n_{r,i} = 0, \ldots, N_r - 1$, that only depend on $n_{r,i}$, the number of class-r customers present at this station (see Figure 22.26). The behavior of this synchronization station can be represented by a continuous-time Markov chain whose state-vector is given by: $(n_{1,i}, \ldots, n_{|R(i)|,i})$ with $n_{r,i} = 0, \ldots, N_r$ for any $r = 1, ..., |R(i)|$. Note that, since the synchronization operation is instantaneous, the only feasible states are such that at least one input queue is empty. The steady-state probabilities $\tilde{p}_i(n_{1,i}, \ldots, n_{|R(i)|,i})$ of this Markov chain can be calculated using any appropriate numerical technique. It is important to note that for a two-class synchronization station, the associated Markov chain is linear [6]. As a result the steady-state probabilities $\tilde{p}_i(n_{1,i}, n_{2,i})$ can be obtained by means of a single formula. Now for any synchronization involving at least three classes of customers, one can use any iterative technique. Once the steady-state probabilities have been obtained, the marginal probabilities $\tilde{p}_{r,i}(n_{r,i})$ required by the approximate method can simply be derived by summing up these detailed probabilities.

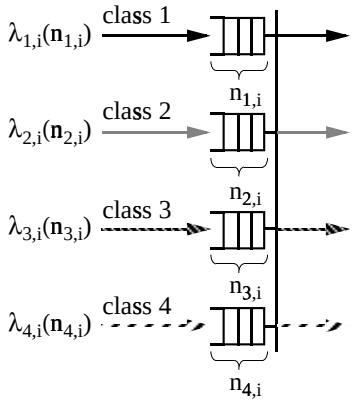


Figure 22.26: Analysis in isolation of a multi-class synchronization station

To illustrate the accuracy of the technique when applied to (simple) MSC-QNs we test Example 6 with stations having Coxian-2 service-time distributions given in Table 22.21. Tables 22.22 gives the throughput of each class of customers (measured at station 1), for two different population-vectors (5, 7, 9, 6) and (10, 15, 11, 13). Table 22.23 gives the mean response-time of each class of customers at the different stations, for a single population-vector (5, 7, 9, 6). Both tables give the results when an exact analysis in isolation of each synchronization station is derived and when the class-aggregation technique presented in the next section is used. For all approximate parameters the relative errors with respect to the exact values obtained by simulation, are also given. It appears that the results obtained by the technique remain accurate: the mean error is about 1% for the throughputs and 5% for the mean response-time. More examples are given in [6] and similar conclusions were drawn.

22.4.4 Class-aggregation technique

When $|R(i)|$ is large, calculating the steady-state probabilities of the Markov chain associated with the synchronization station may be very tedious. In this section, we present a class-aggregation technique for the analysis in isolation of a synchronization station. Even if it uses the same basic idea as the class-aggregation technique presented in Section 22.3.4 for the analysis in isolation of a multi-class service center, this technique differs significantly from the earlier one. Again the class-aggregation principle will allow to drastically reduce the complexity of the analysis in isolation of a synchronization station without introducing any significant additional error. Here we will just present the basic ideas of the class-aggregation technique. All the technical details concerning this technique can be found in [6].

To understand the general principle of this technique let us consider a par-

ticular class r of customers that visits synchronization station i, i.e. $r \in R(i)$. With respect to class-r customers, the customers of the other classes can be considered as resources that are required in order for class-r customers to proceed through the synchronization station. Indeed, the synchronization mechanism forces every class-r customer to synchronize with a customer of every other class. The only condition that must be satisfied for a class-r customer to be allowed to depart from the synchronization station, is that (at least) one customer of each of the other classes be present. This condition can be represented by a permit for class r. Let $n^r_{a,i}$ denote the number of class-r permits available at any time. It is given by:

$$n^r_{a,i} = \min_{s \in R(i),\ s \neq r} n_{s,i}$$

A class-r permit can be viewed as a join of $|R(i)| - 1$ customers, one of each class s that visits station i, $s \neq r$. This is illustrated in Figure 22.27 for the case $|R(i)| = 4$ and $r = 1$.

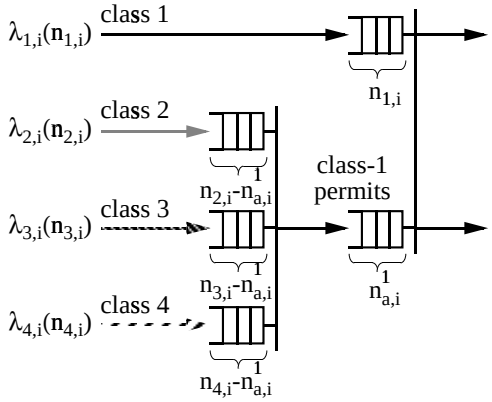


Figure 22.27: Class-aggregation technique applied to synchronization stations.

Note that at least one of the quantities $n_{r,i}$ and $n^r_{a,i}$ is zero at any time. In the case where $n_{r,i} > 0$ (and thus $n^r_{a,i} = 0$), every class-r customer must wait until a new permit becomes available. Therefore, the effect of the customers of the other classes on class-r customers can be represented in an aggregate way by the effect of the class-r permits. The class-aggregation idea can then be applied as follows. For each class of customers, say r, transform the original multi-class synchronization station into a two-class synchronization station in which the effect of the other classes is represented in an aggregate way by means of an aggregate class, referred to as class a^r. The class-a^r customers represent the class-r permits. The two-class synchronization station corresponding to class r will be referred to as the r-th aggregate synchronization station and is illustrated in Figure 22.28. Note that this transformation has to be performed $|R(i)|$ times, each one with respect to a particular class of customers, in order

to obtain estimations of the marginal probabilities of all the classes.

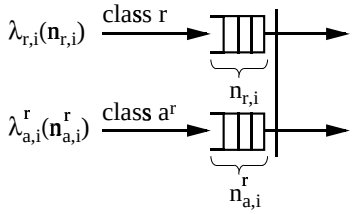


Figure 22.28: The r-th aggregate synchronization station.

We now need to define the parameters of the r-th aggregate synchronization station, for each $r \in R(i)$. Let us consider a particular aggregate station corresponding to a given r. The arrival process of class-r customers is obviously the same as that of the original synchronization station. The only parameter to be define is thus the arrival process of the aggregate class a^r. As for the class-aggregation in the context of service center stations, the arrival process of class-a^r customers is modeled as a Markovian process whose rate $\lambda^r_{a,i}(n^r_{a,i})$ depends on the total number $n^r_{a,i}$ of class-a^r customers currently present in the station.

If the rates $\lambda^r_{a,i}(n^r_{a,i})$ are given, the r-th aggregate synchronization station can straightforwardly be analyzed. Indeed as stated in the previous section the steady-state performance parameters of a two-class synchronization station analyzed in isolation have a closed form. From the detailed joint probabilities $\tilde{p}^r_i(n_{r,i}, n^r_{a,i})$ of this 2-class synchronization station one can easily derive an estimation of the required marginal probabilities $\tilde{p}_{r,i}(n_{r,i})$ that $n_{r,i}$ class-r customers are present in the station for this given class r. These estimation will be good provided the rates of the aggregate class are well estimated.

All the problem thus consists in estimating the rates $\lambda^r_{a,i}(n^r_{a,i})$ of the aggregate class a^r. Note that the maximum number of customers of class a^r that can be simultaneously present is now given by:

$$N^r_a = \min_{s \in R(i),\ s \neq r} N_s$$

For each class a^r, we just have to determine the quantities $\lambda^r_{a,i}(n^r_{a,i})$, for $n^r_{a,i} = 0, \ldots, N^r_a - 1$. Let $\mathbf{n^r} = [n_s]_{s \in R(i),\ s \neq r}$ denote the state-vector representing the number of customers of all the classes (that visit station i) but class r. A given number of class-a^r customers may actually represent different vectors $\mathbf{n^r}$. Let $E^r(n^r_{a,i})$ denote the set of all possible vectors $\mathbf{n^r}$ having the associated number of class-a^r customers :

$$E^r(n^r_{a,i}) = \{\mathbf{n^r} = [n_s]_{s \in R(i),\ s \neq r} \mid n_s \geq n^r_{a,i} \text{ and } \exists t \neq r \mid n_t = n^r_{a,i}\}$$

Let $P_i(\mathbf{n^r} | n^r_{a,i})$ be the conditional probability that the classes other than class r are in state $\mathbf{n^r}$ given that the total number of class-r permits (class-a^r

customers) is $n^r_{a,i}$. It has been shown in [6] that the $\lambda^r_{a,i}(n^r_{a,i})$ can be obtained by the following relation:

$$\lambda^r_{a,i}(n^r_{a,i}) = \sum_{\mathbf{n^r} \in E^r(n^r_{a,i})} \lambda^r_{a,i}(\mathbf{n^r}) P_i(\mathbf{n^r} | n^r_{a,i}) \qquad \text{for } n^r_{a,i} = 0, \ldots, N^r_a - 1 \quad (22.23)$$

where

$$\lambda^r_{a,i}(\mathbf{n^r}) = \begin{cases} \lambda_{s,i}(n_s) & \text{if } n_s = n^r_{a,i} \text{ and } n_t > n^r_{a,i} \text{ for all } t \neq r,\ s \\ 0 & \text{else} \end{cases}$$

$\lambda^r_{a,i}(\mathbf{n^r})$ denote the arrival rate of class-a^r customers corresponding to a particular state $\mathbf{n^r}$. There are two cases to be considered. On the one hand, if $\mathbf{n^r}$ is such that exactly one class, say s, is such that $n_s = n^r_{a,i}$, while all the other classes $t \neq s$ are such that $n_t > n^r_{a,i}$, one new permit will become available as soon as one customer of class s arrives (regardless of arrivals of customers of other classes). Thus in that case $\lambda^r_{a,i}(\mathbf{n^r}) = \lambda_{s,i}(n_s)$. On the other hand, if there are at least two classes of customers, say s and t, such that $n_s = n_t = n^r_{a,i}$, a single arrival (either of a class-s customer or of a class-t customer) will not increase the number of permits available. Thus in that case $\lambda^r_{a,i}(\mathbf{n^r}) = 0$. Now the conditional probabilities $P_i(\mathbf{n^r} | n^r_{a,i})$ appearing in Relation 22.23 can be calculated in the same way as in Section 22.4.4.

The accuracy of the class-aggregation technique when applied to synchronization stations is illustrated on Example 6 (see Tables 22.22 and 22.23). It is clear for this example that the use of the class-aggregation principle does not introduce any significant additional error. This is a general conclusion that has been verified on all the examples we have run.

Mean response-time		$N_1 = 6$	$N_2 = 8$	$N_3 = 10$
		Simul.	Exact Anal. in Isol.	Class-aggreg.
Station 1	Class 1	2.729	2.750	2.743
		± 0.024	+0.8 %	+0.5 %
	Class 2	2.549	2.356	2.390
		± 0.025	-7.5 %	-6.2 %
	Class 3	2.435	2.183	2.170
		± 0.025	-10.3 %	-10.9 %
Station 2	Class 1	4.868	5.042	5.053
		± 0.029	+3.6 %	+3.8 %
Station 3	Class 1	5.917	5.760	5.758
		± 0.044	-2.6 %	-2.7 %
	Class 2	5.286	5.314	5.317
		± 0.032	+0.5 %	+0.6 %
	Class 3	6.395	5.918	5.917
		± 0.041	-7.4 %	-7.4 %
Station 4	Class 3	0.8222	0.6707	0.6708
		± 0.005	-18.4 %	-18.4 %
Station 5	Class 1	3.452	3.466	3.419
		± 0.052	+0.4 %	-0.9 %
	Class 2	3.545	3.511	3.496
		± 0.059	-1.0 %	-1.3 %
	Class 3	3.272	3.122	3.078
		± 0.053	-4.6 %	-5.9 %
Station 6	Class 1	1.735	1.669	1.661
		± 0.051	-3.8 %	-4.2 %
	Class 2	1.411	1.362	1.359
		± 0.034	-3.4 %	-3.7 %
	Class 3	2.098	1.909	1.904
		± 0.048	-9.0 %	-9.2 %
Station 7	Class 1	0.5456	0.5587	0.5590
		± 0.002	+2.4 %	+2.5 %
Station 8	Class 1	1.734	1.679	1.678
		± 0.011	-3.2 %	-3.2 %
	Class 2	1.744	1.680	1.680
		± 0.013	-3.7 %	-3.7 %
Station 9	Class 1	2.832	2.501	2.501
		± 0.018	-11.7 %	-11.7 %
	Class 2	2.746	2.502	2.503
		± 0.016	-8.9 %	-8.9 %
Station 10	Class 3	5.141	5.111	5.121
		± 0.023	-0.6 %	-0.4 %
Station 11	Class 1	0.2554	0.2569	0.2573
		± 0.002	+0.6 %	+0.7 %
	Class 2	4.212	4.157	4.154
		± 0.033	-1.3 %	-1.4 %
Station 12	Class 1	0.9163	0.9170	0.9176
		± 0.006	+0.1 %	+0.1 %
	Class 2	1.055	1.012	1.013
		± 0.008	-4.0 %	-4.0 %
	Class 3	5.905	6.043	6.062
		± 0.052	+2.3 %	+2.7 %

Table 22.19: Approximate mean response-times of Example 5B, with $\mathbf{N} = (6, 8, 10)$.

R	Exact Anal.	Class Aggreg.
1	11	11
2	221	2 × 221
3	3 631	3 × 651
4	53 241	4 × 1 301
5	732 051	5 × 2 171
6	9 663 061	6 × 3 261

Table 22.20: Number of states of the Markov chain associated with the analysis in isolation of a RANDOM queue with R classes of customers and class-dependent exponential service time distributions.

Cox-2 parameters	Station 3	Station 4	Station 5	Station 6
$(\mu_{i1}, a_{i1}, \mu_{i2})$	(0.5, 0, -)	(2, 0.5, 2)	(2, 0.1, 0.1)	(0.67, 0.1, 0.2)
t_i	2	0.75	1	2
cv_i^2	1	0.78	5	1.75

	Station 7	Station 8	Station 9
$(\mu_{i1}, a_{i1}, \mu_{i2})$	(0.5, 0.1, 0.1)	(1.33, 0.5, 0.2)	(0.67, 0.1, 0.2)
t_i	3	1	2
cv_i^2	2.55	0.75	1.75

Table 22.21: Parameters of the stations of Example 6.

Throughputs	$N_1 = 5$	$N_2 = 7$ $N_4 = 6$	$N_3 = 9$	$N_1 = 10$	$N_2 = 15$ $N_4 = 13$	$N_3 = 11$
	Simul.	Exact Anal. in Isol.	Class-aggreg.	Simul.	Exact Anal. in Isol.	Class-aggreg.
Class 1	0.3932	0.3999 +1.7 %	0.3959 +0.5 %	0.4397	0.4421 +0.5 %	0.4407 +0.2 %
Class 2	0.3932	0.3999 +1.7 %	0.3959 +0.5 %	0.4397	0.4420 +0.5 %	0.4407 +0.2 %
Class 3	0.3932	0.3999 +1.7 %	0.3959 +0.5 %	0.4397	0.4420 +0.5 %	0.4407 +0.2 %
Class 4	0.3932	0.3998 +1.7 %	0.3959 +0.5 %	0.4397	0.4421 +0.5 %	0.4407 +0.2 %

Table 22.22: Approximate throughputs of Example 6.

Mean response-time		$N_1 = 5$ $N_2 = 7$ $N_3 = 9$ $N_4 = 6$		
		Simul.	Exact Anal. in Isol.	Class-aggreg.
Station 1	Class 1	5.565 ± 0.099	5.862 + 5.3 %	6.039 + 8.5 %
	Class 2	2.938 ± 0.067	2.109 - 28 %	2.180 - 26 %
Station 2	Class 1	13.82 ± 0.134	14.37 4.0 %	14.47 + 4.7 %
	Class 2	11.55 ± 0.107	11.05 - 4.3 %	11.34 - 1.8 %
	Class 3	1.906 ± 0.046	1.841 - 3.4 %	2.096 + 10 %
Station 3	Class 1	7.151 ± 0.053	6.642 - 7.1 %	6.589 - 7.8 %
Station 4	Class 2	1.049 ± 0.008	1.028 - 2.0 %	1.033 - 1.5 %
Station 5	Class 3	2.855 ± 0.041	2.797 - 2.0 %	2.786 - 2.4 %
Station 6	Class 3	3.369 ± 0.041	3.361 - 0.2 %	3.348 - 0.6 %
Station 7	Class 3	11.89 ± 0.185	12.19 + 2.5 %	12.12 + 1.9 %
Station 8	Class 4	3.377 ± 0.013	3.299 - 2.3 %	3.264 - 3.3 %
Station 9	Class 4	6.593 ± 0.050	6.568 - 0.4 %	6.529 - 1.0 %

Table 22.23: Approximate mean response-times of Example 6, with $\mathbf{N} = (5, 7, 9, 6)$.

Bibliography

[1] Baynat B. *Une méthode approximative d'analyse des réseaux de files d'attentes fermés multiclasses.* PhD thesis, Universit é Pierre & Marie Curie, Paris 6, 1991.

[2] Baynat B., Ross K., and Dallery Y. A decomposition approximation method for multiclass bcmp queueing networks with multiple-server stations. *Annals of Operations Research*, 48:273–294, 1994.

[3] Baynat B. and Dallery Y. Approximate techniques for general closed queueing networks with subnetworks having population constraints. *European Journal of Operational Research*, 69:250–264, 1993.

[4] Baynat B. and Dallery Y. A decomposition approximation method for closed queueing networks with fork/join subnetworks. In *International Conference on Decentralized and Distributed Systems (ICDDS'93), Palma de Mallorca*, pages 311–322, 1993.

[5] Baynat B. and Dallery Y. A unified view of product-form approximation techniques for general closed queueing networks. *Performance Evaluation*, 18:205–224, 1993.

[6] Baynat B. and Dallery Y. Approximate analysis of multi-class synchronized closed queueing networks. In *International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'95), Durham, North Carolina*, 1995.

[7] Baynat B. and Dallery Y. A product-form approximation method for general closed queueing networks with several classes of customers. *Performance Evaluation*, 24:165–188, 1996.

[8] Baskett F., Chandy K.M., Muntz R.R., and Palacios F. Open, closed and mixed networks of queues with differents classes of customers. *Journal of the ACM*, 22:248–260, 1975.

[9] Buzen J.P. Computational algorithms for closed queueing networks with exponential servers. *Communication of the ACM*, 16:527–531, 1973.

[10] Jackson J.R. Jobshop-like queueing systems. *Management Science*, 10:131–142, 1963.

[11] Reiser M. and Lavenberg S.S. Mean value analysis of closed multichain queuing networks. *Journal of the ACM*, 27:313–322, 1980.

[12] Marie R. An approximate analytical method for general queueing networks. *Trans. on Software Engineering*, 5:530–538, 1979.

[13] Marie R. Calculating equilibrium probabilities for $\lambda(n)/c_k/1/n$ queues. *Performance Evaluation*, 9:117–125, 1980.

[14] Marie R. and Rubino G. An approximation for a multiclass ./m/fifo queue imbedded in a closed queueing network. In *Journal of Systems and Software*, pages 31–39, 1986.

[15] Marie R., Snyder P., and Stewart W.J. Extensions and computational aspects of an iterative method. In *ACM Sigmetrics*, 1982.

[16] Bruel S.C. and Balbo G. *Computational algorithms for closed queueing networks*. North-Holland, Amsterdam, 1980.

[17] Gordon W.J. and Newell G.F. Closed queuing networks with exponential servers. *Operations Research*, 15:254–265, 1967.

[18] Stewart W.J. A comparison of numerical techniques in markov modeling. *Communication of the ACM*, 21:144–152, 1978.

[19] Dallery Y. Approximate analysis of general open queueing networks with restricted capacity. *Performance Evaluation*, 11:209–222, 1990.

[20] Dallery Y. and Cao X. Operational analysis of stochastic closed queueing networks. *Performance Evaluation*, 14:43–61, 1992.

Chapter 23

Response Time Approximation for Stochastic Marked Graphs

A general iterative technique for approximate throughput computation of strongly connected stochastic marked graphs (SMG's) is presented in this Chapter. We consider SMG's with time and marking independent *exponentially distributed* service times associated with transitions.

The approach has two basic foundations. First, it is deeply based on *qualitative theory of MG's*. More precisely, given an arbitrary cut (subset of places producing a net partition), a *structural decomposition* technique is developed that allows us to split a strongly connected MG into two *aggregated subsystems* and a *basic skeleton system*. And what is more important, *the behaviours of the subsystems, including steps, language of firing sequences and reachable markings, are equivalent to the whole system behaviour* (projected on the corresponding subsets of nodes). Second, after the decomposition phase, an iterative *response time approximation* method is applied for the computation of the throughput. Experimental results on several examples generally have an error of less than 3%. The state space is usually reduced by more than one order of magnitude; therefore the analysis of otherwise intractable systems is possible.

The Chapter is organized as follows. In Section 23.1, some fundamental properties on MG's and *implicit places* are presented. Section 23.2 includes the structural decomposition of MG's used in the rest of the Chapter. The iterative technique for approximate throughput computation is described in Section 23.3. Section 23.4 includes several application examples to illustrate the introduced technique. Finally, some bibliographic remarks are included in Section 23.5.

23.1 Implicit Places and MG's

Let $\mathcal{N} = \langle P, T, F \rangle$ be a net. A *path* of $\mathcal{N}$ is a sequence $x_1 \ldots x_k$ of elements (places and transitions) of $\mathcal{N}$ satisfying $\langle x_1, x_2 \rangle, \ldots, \langle x_{k-1}, x_k \rangle \in F$. It is a *circuit* if $\langle x_k, x_1 \rangle \in F$. A path (circuit) is called *simple* if all elements in the sequence defining the path (circuit) are different. In this Chapter we only consider simple paths and circuits. We denote by $\mathcal{P}(x, y)$, $x, y \in P \cup T$, the set of simple paths from x to y. This notion is extended to sets of elements: $\mathcal{P}(X, Y)$ is the union of the $\mathcal{P}(x, y)$ for all $x \in X$ and for all $y \in Y$.

A *marked graph* (MG) is a Petri net such that each place has exactly one input transition and exactly one output transition (see Chapter 2). MG's allow synchronization but no choice. MG's are a subclass of ordinary Petri nets for which a simple, powerful, and elegant theory allows very efficient analysis and synthesis algorithms. A summary of structure theory of MG's can be found in Chapter 6.

An *implicit place* never is the unique restricting the firing of its output transitions (see Chapter 6). Let $\mathcal{N}$ be any net and $\mathcal{N}^p$ be the net resulting from adding a place p to $\mathcal{N}$. If $\mathbf{m_0}$ is an initial marking of $\mathcal{N}$, $\mathbf{m_0}^p$ denotes the initial marking of $\mathcal{N}^p$. The incidence matrix of $\mathcal{N}$ is $\mathbf{C}$ and $\mathbf{l_p}$ is the incidence vector of place p.

Definition 23.1 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a net system and $p \notin P$ be a place to be added. Then p is an implicit place (IP) with respect to $\langle \mathcal{N}, \mathbf{m_0} \rangle$ (or equivalently, it is an implicit place in $\langle \mathcal{N}^p, \mathbf{m_0}^p \rangle$) iff the languages of firing sequences of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ and $\langle \mathcal{N}^p, \mathbf{m_0}^p \rangle$ coincide. That is,* $\mathrm{L}(\mathcal{N}, \mathbf{m_0}) = \mathrm{L}(\mathcal{N}^p, \mathbf{m_0}^p)$.

A place is an IP depending on the initial marking. Places which can be implicit for any initial marking are said to be *structurally implicit* (SIP). Inside the class of SIP's we are interested in the so called *marking structurally implicit places* (MSIP) whose structural characterization is given, as definition, in the following statement.

Definition 23.2 *Let $\mathcal{N}$ be a net and p be a place with incidence vector $\mathbf{l_p}$. The place p is an MSIP in $\mathcal{N}^p$ if there exists $\mathbf{y} \geq \mathbf{0}$ such that $\mathbf{y} \cdot \mathbf{C} = \mathbf{l_p}$.*

From this characterization of an MSIP, p, a method to compute an initial marking of p making it implicit with respect to $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is presented in Chapter 6.

In the following, we characterize a special class of MSIP's with respect to strongly connected MG's called *TT-MSIP's*. These places have only one input arc and one output arc and therefore, $\mathcal{N}^p$ will be also an MG. The row of the incidence matrix corresponding to a TT-MSIP can be obtained from the summation of rows corresponding to the places in any path from the input transition to the output transition of the place. Moreover, we characterize the minimum initial marking making these places implicit with respect to $\langle \mathcal{N}, \mathbf{m_0} \rangle$ and preserving its steps.

Theorem 23.3 *Let $\mathcal{N} = \langle P, T, F\rangle$ be a strongly connected MG and $p \notin P$ be a place to be added with one input transition $t_i \in T$ ($^\bullet p = \{t_i\}$) and one output transition $t_o \in T$ ($p^\bullet = \{t_o\}$). The place p is an MSIP (called TT-MSIP) with respect to $\mathcal{N}$ and $\forall \pi \in \mathcal{P}(t_i, t_o)$, $\mathbf{l_p} = \sum_{p_j \in \pi} \mathbf{l_{p_j}}$.*

Proof:
If $\mathcal{N}$ is a strongly connected MG, for all path, $\pi \in \mathcal{P}(t_i, t_o)$, of the form $t_i(= t_1)p_1t_2 \ldots t_{k-1}p_{k-1}t_k(= t_o)$: $^\bullet p_j = \{t_j\}$, $p_j^\bullet = \{t_{j+1}\}$, $j = 1, \ldots, k-1$. Therefore, the summation of the rows in the incidence matrix corresponding to the places in π, $\mathbf{v} = \sum_{p_j \in \pi} \mathbf{l_{p_j}}$, verifies:

(1) $\mathbf{v}[t] = 0, \forall t \notin \pi$;

(2) $\mathbf{v}[t_i] = \mathbf{v}[t_1] = \sum_{p_j \in \pi} \mathbf{l_{p_j}}[t_1] =$ `if` $t_i \neq t_o$ `then` $\mathbf{l_{p_1}}[t_1] = 1$ `else` $\mathbf{l_{p_1}}[t_1] + \mathbf{l_{p_{k-1}}}[t_o] = 0$;

(3) $\mathbf{v}[t_r] = \sum_{p_j \in \pi} \mathbf{l_{p_j}}[t_r] = \mathbf{l_{p_{r-1}}}[t_r] + \mathbf{l_{p_r}}[t_r] = 0, \forall t_r \in \pi, r = 2 \ldots (k-1)$;

(4) $\mathbf{v}[t_o] = \mathbf{v}[t_k] = \sum_{p_j \in \pi} \mathbf{l_{p_j}}[t_k] =$ `if` $t_i \neq t_o$ `then` $\mathbf{l_{p_{k-1}}}[t_k] = -1$ `else` $\mathbf{l_{p_1}}[t_i] + \mathbf{l_{p_{k-1}}}[t_k] = 0$.

That is, vector $\mathbf{v}$ coincides with the incidence vector, $\mathbf{l_p}$, of p, and according to Definition 23.2, p is an MSIP (with $\mathbf{y}[p_j] =$ `if` $p_j \in \pi$ `then` 1 `else` $0, \forall p_j \in P$) and also a TT-MSIP. ♢

The following result characterizes the minimum initial marking of a TT-MSIP to be implicit *preserving all steps of the net system* $\langle \mathcal{N}, \mathbf{m_0}\rangle$. This marking is computed from the contents of tokens of the existing paths from the input transition of p to its output transition.

Theorem 23.4 *Let $\langle \mathcal{N}, \mathbf{m_0}\rangle$ be a strongly connected and live MG, and $p \notin P$ be a TT-MSIP to be added with $^\bullet p = \{t_i\}$ and $p^\bullet = \{t_o\}$. The minimum initial marking of p to be an IP in $\langle \mathcal{N}^p, \mathbf{m_0}^p\rangle$ preserving all steps of $\langle \mathcal{N}, \mathbf{m_0}\rangle$ is*

$$m_0^{\min}(p) = \min\{\sum_{p_j \in \pi} \mathbf{m_0}[p_j] \mid \pi \in \mathcal{P}(t_i, t_o)\}.$$

Proof:
First we prove that p is an IP with an initial marking $\mathbf{m_0}^p[p] = m_0^{\min}(p)$ (i.e., $\mathrm{L}(\mathcal{N}, \mathbf{m_0}) = \mathrm{L}(\mathcal{N}^p, \mathbf{m_0}^p)$).

$\mathrm{L}(\mathcal{N}^p, \mathbf{m_0}^p) \subseteq \mathrm{L}(\mathcal{N}, \mathbf{m_0})$. Removing place p from $\mathcal{N}^p$, we remove constraints for firing transitions. Therefore, all sequence $\sigma \in \mathrm{L}(\mathcal{N}^p, \mathbf{m_0}^p)$ are also firable in $\langle \mathcal{N}, \mathbf{m_0}\rangle$.

$\mathrm{L}(\mathcal{N}, \mathbf{m_0}) \subseteq \mathrm{L}(\mathcal{N}^p, \mathbf{m_0}^p)$. We prove this part by contradiction. Let σ be a sequence firable in $\langle \mathcal{N}, \mathbf{m_0}\rangle$ but not firable in $\langle \mathcal{N}^p, \mathbf{m_0}^p\rangle$. Let σ_1 be the maximal prefix of σ firable in $\langle \mathcal{N}, \mathbf{m_0}\rangle$ and $\langle \mathcal{N}^p, \mathbf{m_0}^p\rangle$: $\mathbf{m_0} \xrightarrow{\sigma_1} \mathbf{m}$ and $\mathbf{m_0}^p \xrightarrow{\sigma_1} \mathbf{m}^p$. Obviously, $\mathbf{m}^p[p_i] = \mathbf{m}[p_i]$ for all $p_i \in P$. The only transition preventing to

finish the firing of σ after the firing of σ_1 in $\langle \mathcal{N}^p, \mathbf{m_0}^p \rangle$ is t_o. This means that $\mathbf{m}^p[p] = \mathbf{m_0}^p[p] + \mathbf{l_p} \cdot \boldsymbol{\sigma}_1 = 0$. Now, we select a path $\pi \in \mathcal{P}(t_i, t_o)$ such that $\mathbf{m_0}^p[p] = \sum_{p_j \in \pi} \mathbf{m_0}[p_j]$. Moreover, according to Theorem 23.3, $\mathbf{l_p} = \sum_{p_j \in \pi} \mathbf{l_{p_j}}$. Therefore, substituting these last expressions in the above expression of $\mathbf{m}^p[p]$ we obtain, $0 = \mathbf{m}^p[p] = \sum_{p_j \in \pi} \mathbf{m_0}[p_j] + \sum_{p_j \in \pi} \mathbf{l_{p_j}} \cdot \boldsymbol{\sigma}_1 = \sum_{p_j \in \pi} \mathbf{m}[p_j]$. But this contradicts the hypothesis from which σ is firable in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ and therefore the place $p_j \in {}^\bullet t_o$ in the path π must contain at least one token.

In order to prove that $m_0^{\min}(p)$ is the minimum initial marking making p an IP preserving the steps of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ we distinguish two cases.

Case 1 ($t_i \neq t_o$, i.e., p is self-loop free). In this case, since p is an IP for $\mathbf{m_0}^p[p] = m_0^{\min}(p)$, it is step preserving [8]. We prove that $m_0^{\min}(p)$ is the minimum initial marking.

First we build a sequence, σ, of maximal length in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ firing only transitions of $T \setminus \{t_i\}$. All reached markings throughout the sequence are different, on the contrary we have a reproducible sequence without transition t_i and this is not possible in MG's. This sequence is finite because the number of different markings in a bounded net is finite. Since the sequence is maximal, we reach a marking $\mathbf{m}$ from which t_i is the unique firable transition (the net system is live), $\mathbf{m_0} \xrightarrow{\sigma} \mathbf{m}$.

In $\langle \mathcal{N}, \mathbf{m} \rangle$ there exists a path, π', from t_i to t_o where all places contain zero tokens. In effect, the only firable transition from $\mathbf{m}$ is t_i, then all transitions of $T \setminus \{t_i\}$ have at least one input place with zero tokens. Therefore, t_o has an empty input place with an input transition that has another empty input place, and so on. This sequence cannot be a circuit because the MG is live and then one of the places in the sequence is an output place of t_i.

Taking into account that p is an IP with respect to $\langle \mathcal{N}, \mathbf{m_0} \rangle$, σ is also firable in $\langle \mathcal{N}^p, \mathbf{m_0}^p \rangle$ and the number of tokens in p is: $\mathbf{m}^p[p] = \mathbf{m_0}^p[p] + \mathbf{l_p} \cdot \boldsymbol{\sigma}$. Let π be a path of $\mathcal{P}(t_i, t_o)$ such that $\mathbf{m_0}^p[p] = \sum_{p_j \in \pi} \mathbf{m_0}[p_j]$. Moreover, according to Theorem 23.3, $\mathbf{l_p} = \sum_{p_j \in \pi} \mathbf{l_{p_j}} = \sum_{p_k \in \pi'} \mathbf{l_{p_k}}$, because $\pi' \in \mathcal{P}(t_i, t_o)$, but in general $\mathbf{m_0}^p[p] \leq \sum_{p_k \in \pi'} \mathbf{m_0}[p_k]$. Considering these expressions, we can rewrite the contents of tokens of p in the following way: $\mathbf{m}^p[p] = \sum_{p_j \in \pi} \mathbf{m_0}[p_j] + \left(\sum_{p_j \in \pi} \mathbf{l_{p_j}}\right) \cdot \boldsymbol{\sigma} \leq \sum_{p_k \in \pi'} \mathbf{m_0}[p_k] + \left(\sum_{p_k \in \pi'} \mathbf{l_{p_k}}\right) \cdot \boldsymbol{\sigma} = \sum_{p_k \in \pi'} \mathbf{m}[p_k] = 0$.

Therefore, $m_0^{\min}(p)$ is a minimal initial marking because there exists a firable sequence in $\langle \mathcal{N}^p, \mathbf{m_0}^p \rangle$ that empties the place.

Case 2 ($t_i = t_o$, i.e., p is a self-loop). In this case, the minimal initial marking to make p an IP is equal to one. We prove that in order to preserve the steps of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ we need at least the initial marking stated.

From $\langle \mathcal{N}, \mathbf{m_0} \rangle$ we can obtain a new net $\langle \mathcal{N}', \mathbf{m_0}' \rangle$ by splitting the transition t_i into two transitions t and t' such that: ${}^\bullet t = {}^\bullet t_i$ and ${t'}^\bullet = t_i{}^\bullet$; and a new ordinary place p_i such that: ${}^\bullet p_i = \{t\}$ and $p_i{}^\bullet = \{t'\}$ and $\mathbf{m_0}'[p_i] = 0$. Let RS$'$ be the set of reachable markings of $\langle \mathcal{N}', \mathbf{m_0}' \rangle$ in which the marking of place p_i is equal to zero. It is trivial to verify that the set RS$'$ projected with respect to the set of places P coincides with the set of reachable markings of $\langle \mathcal{N}, \mathbf{m_0} \rangle$. Therefore, the set of steps of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ is enclosed in the set of steps of $\langle \mathcal{N}', \mathbf{m_0}' \rangle$

renaming the appearings of t by t_i and removing the appearings of t'.

Let us consider a place p with ${}^\bullet p = \{t'\}$ and $p^\bullet = \{t\}$ with respect to $\langle \mathcal{N}', \mathbf{m_0}' \rangle$. Applying the previous Case 1 to place p we conclude that the minimum initial marking to make implicit place p with respect to $\langle \mathcal{N}', \mathbf{m_0}' \rangle$ and preserving the steps of the net is equal to the minimal contents of tokens in the paths from t' to t (i.e., the circuits of the net system $\langle \mathcal{N}, \mathbf{m_0} \rangle$ traversing the transition t_i). Therefore, according to the previous paragraph a self-loop, p, with this initial marking preserves the steps of $\langle \mathcal{N}, \mathbf{m_0} \rangle$. Moreover, it is minimal because the steps requiring the maximum amount of tokens in p are the steps of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ (they contain the output transition of p). ♢

Corollary 23.5 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a strongly connected and live MG, and $p \notin P$ be a TT-MSIP to be added with ${}^\bullet p = \{t_i\}$ and $p^\bullet = \{t_o\}$. The place p is an IP in $\langle \mathcal{N}^p, \mathbf{m_0}^p \rangle$ preserving all steps of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ for all initial marking $\mathbf{m_0}^p[p] \geq m_0^{\min}(p)$.*

Proof:
If we remove $\mathbf{m_0}^p[p] - m_0^{\min}(p)$ tokens from p, then p is an IP in $\langle \mathcal{N}^p, \mathbf{m_0}^p \rangle$ preserving all steps of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ (Theorem 23.4). Therefore, all sequences and steps of $\langle \mathcal{N}, \mathbf{m_0} \rangle$ are firable in $\langle \mathcal{N}^p, \mathbf{m_0}^p \rangle$. On the other hand, $\mathbf{m_0}^p[p] - m_0^{\min}(p)$ tokens in p are frozen, hence the sequences and steps of $\langle \mathcal{N}^p, \mathbf{m_0}^p \rangle$ coincide with those of $\langle \mathcal{N}, \mathbf{m_0} \rangle$. In effect, add the place p to a net system $\langle \mathcal{N}^{p'}, \mathbf{m_0}^{p'} \rangle$ where p' is a place such that ${}^\bullet p' = \{t_i\}$, $p'^\bullet = \{t_o\}$ and its initial marking is $m_0^{\min}(p')$. $\langle \mathcal{N}^{p'}, \mathbf{m_0}^{p'} \rangle$ has the same sequences and steps that $\langle \mathcal{N}, \mathbf{m_0} \rangle$ (Theorem 23.4), and there exists a sequence in $\langle \mathcal{N}^{p'}, \mathbf{m_0}^{p'} \rangle$ that empties the place p'. The place p is identical to the place p', hence the minimum marking of p is reached when p' is empty (i.e. it contains $\mathbf{m_0}^p[p] - m_0^{\min}(p)$ tokens). ♢

The Theorem 23.4 characterizes the minimum initial marking of a TT-MSIP to be an IP with respect to $\langle \mathcal{N}, \mathbf{m_0} \rangle$ in terms of the contents of tokens of the paths $\mathcal{P}(t_i, t_o)$. The computation of this minimum initial marking can be done applying an algorithm from the graph theory to determine the cost of the *shortest path from a source vertex to a sink vertex of a directed graph*, $G = (V, E)$, obtained from the original MG (see [2] for implementations of these algorithms). In this graph, each vertex corresponds to a transition of the net. There exists a directed arc between two vertices if and only if there exists a place in the net connecting the two transitions that represent the two vertices. The sense of the arc is the sense of the tokens' flow between the transitions through the place. Each arc has a non-negative cost equal to the initial marking of the place that represents. Moreover, we add an arc $t \to t$ for each vertex t with a cost equal to ∞.

Therefore, if we apply the algorithm to solve the shortest path problem in the directed graph G, we obtain the smallest length of any path from t_i to t_o, denoted length(t_i, t_o). Observe, that length$(t_i, t_o) = \min\{\sum_{p_j \in \pi} \mathbf{m_0}[p_j] \mid \pi \in \mathcal{P}(t_i, t_o)\} = m_0^{\min}(p)$.

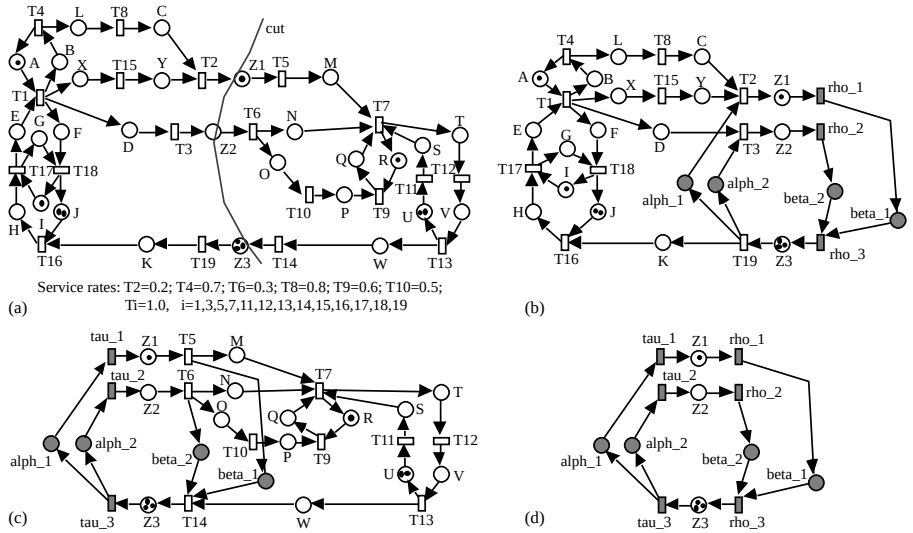


Figure 23.1: An SMG (a), its decomposition in aggregated systems $\mathcal{AS}_1$ (b), $\mathcal{AS}_2$ (c), and the basic skeleton (d).

23.2 Structural Decomposition of MG's

The basic idea is the following: a strongly connected and live MG (see Fig. 23.1.a) is split into two subnets by a *cut* Q defined through some places ($Q = \{Z1, Z2, Z3\}$, in Fig. 23.1.a). From the cut we define three nets: two *aggregated subnets* ($\mathcal{AN}_1$ and $\mathcal{AN}_2$; see Figs. 23.1.b and 23.1.c) and a *basic skeleton* net ($\mathcal{BN}$; see Fig. 23.1.d). These nets will be obtained by substitution of the so called aggregable subnets, defined from the cut Q, by a set of places. We select an initial marking for each added place such that the behaviour of the aggregated subnet is the behaviour of the original MG hiding the behaviour of the aggregable subnet.

Definition 23.6 *Let $\mathcal{N} = \langle P, T, F\rangle$ be a strongly connected MG. A subset of places, $Q \subseteq P$, is said to be a cut of $\mathcal{N}$ iff there exist two subnets, $\mathcal{N}_1 = \langle P_1, T_1, F_1\rangle$ and $\mathcal{N}_2 = \langle P_2, T_2, F_2\rangle$, of $\mathcal{N}$ verifying*

i) $T_1 \cup T_2 = T$, $T_1 \cap T_2 = \emptyset$

ii) $P_1 = T_1^{\bullet} \cup {}^{\bullet}T_1$, $P_2 = T_2^{\bullet} \cup {}^{\bullet}T_2$

iii) $P_1 \cup P_2 = P$, $P_1 \cap P_2 = Q$

iv) $F_i = F \cap ((P_i \times T_i) \cup (T_i \times P_i))$, $i \in \{1, 2\}$

Definition 23.7 *Let $\mathcal{N} = \langle P, T, F\rangle$ be a strongly connected MG, $Q \subseteq P$ a cut of $\mathcal{N}$, and $\mathcal{N}_1 = \langle P_1, T_1, F_1\rangle$, $\mathcal{N}_2 = \langle P_2, T_2, F_2\rangle$ the two subnets associated with*

the cut (by Def. 23.6). The subnets $\mathcal{N}_{A_i} = \langle P_{A_i}, T_{A_i}, F_{A_i} \rangle$, $i \in \{1,2\}$, *are called the aggregable subnets of the cut* Q, *where*

i) $P_{A_i} = P_i \setminus Q$

ii) $T_{A_i} = T_i \setminus T_Q$, *where* $T_Q = {}^\bullet Q \cup Q^\bullet$

iii) $F_{A_i} = F_i \cap ((P_{A_i} \times T_{A_i}) \cup (T_{A_i} \times P_{A_i}))$

The places $p \in P_{A_i}$ *such that* ${}^\bullet p \cap T_{A_i} = \emptyset$ *(resp.,* $p^\bullet \cap T_{A_i} = \emptyset$*) are called source places (resp., sink places) of* $\mathcal{N}_{A_i}$. *The set of input transitions of the source places and output transitions of the sink places are called interface transitions of* $\mathcal{N}_{A_i}$.

We denote $\mathcal{P}_{A_i}$ the set of paths in the net $\mathcal{N}_{A_i}$ from a source place to a sink place. $\mathcal{IP}_{A_i}$ denotes the set of TT-MSIP's with respect to $\mathcal{N}$ obtained from each path of $\mathcal{P}_{A_i}$ by the linear combination of the rows in the incidence matrix corresponding to the path's places. In the sequel, we define the so called *aggregated subnets* of an MG $\langle \mathcal{N}, \mathbf{m_0} \rangle$ with respect to a cut Q. These subnets will be obtained by substituting in $\mathcal{N}$ of an aggregable subnet $\mathcal{N}_{A_i}$ by the set of places $\mathcal{IP}_{A_i}$. This substitution is an abstraction of the subnet $\mathcal{N}_{A_i}$. We select an initial marking for each place $p \in \mathcal{IP}_{A_i}$ (called *aggregation's initial marking*, $\mathbf{m_0}^a[p]$) equal to $\mathbf{m_0}^a[p] = \min\{\sum_{p_j \in \pi} \mathbf{m_0}[p_j] \,|\, \mathbf{l_p} = \sum_{p_j \in \pi} \mathbf{l_{p_j}} \text{ and } \pi \in \mathcal{P}_{A_i}\}$. With this initial marking we prove that the behaviour of the aggregated subnet is the behaviour of the original MG by hiding the behaviour of $\mathcal{N}_{A_i}$.

Definition 23.8 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be a strongly connected and live MG,* $Q \subseteq P$ *a cut of* $\mathcal{N}$, *and* $\mathcal{N}_{A_i}$, $i = 1,2$, *be the aggregable subnets defined by the cut* Q. *The aggregated subsystem* $\mathcal{AS}_i = \langle \mathcal{AN}_i, \mathbf{m_0}^{\mathcal{AN}_i} \rangle$ *is the net system obtained from* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *by substituting the subnet* $\mathcal{N}_{A_j}$ *by the set of places* $\mathcal{IP}_{A_j}$ *with* $\mathbf{m_0}[p] = \mathbf{m_0}^a[p]$, *for all* $p \in \mathcal{IP}_{A_j}$, $i = 1,2; j = 1,2$ *and* $j \neq i$. *The basic skeleton system,* $\mathcal{BS} = \langle \mathcal{BN}, \mathbf{m_0}^{\mathcal{BN}} \rangle$, *is the system obtained from* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *by substituting the subnets* $\mathcal{N}_{A_1}$ *and* $\mathcal{N}_{A_2}$ *by the set of places* $\mathcal{IP}_{A_1}$ *and* $\mathcal{IP}_{A_2}$ *with* $\mathbf{m_0}[p] = \mathbf{m_0}^a[p] = \min\{\sum_{p_j \in \pi} \mathbf{m_0}[p_j] \,|\, \mathbf{l_p} = \sum_{p_j \in \pi} \mathbf{l_{p_j}}$ *and* $\pi \in \mathcal{P}_{A_i}\}$, *for all* $p \in \mathcal{IP}_{A_1} \cup \mathcal{IP}_{A_2}$.

Theorem 23.9 *Let* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *be a strongly connected and live MG,* $Q \subseteq P$ *a cut of* $\mathcal{N}$ *and* $\mathcal{AS}_i$ *be the aggregated subsystem obtained from* $\langle \mathcal{N}, \mathbf{m_0} \rangle$ *by substituting the subnet* $\mathcal{N}_{A_j}$ *by the set of places* $\mathcal{IP}_{A_j}$ *with* $\mathbf{m_0}^{\mathcal{AN}_i}[p] =$ `if` $p \in \mathcal{IP}_{A_j}$ `then` $\mathbf{m_0}^a[p]$ `else` $\mathbf{m_0}[p]$, $i = 1,2; j = 1,2$, *and* $j \neq i$.

i) $\mathrm{L}(\mathcal{N}, \mathbf{m_0})|_{T \setminus T_{A_j}} = \mathrm{L}(\mathcal{AN}_i, \mathbf{m_0}^{\mathcal{AN}_i})$.

ii) $\mathrm{RS}(\mathcal{N}, \mathbf{m_0})|_{P \setminus P_{A_j}} = \mathrm{RS}(\mathcal{AN}_i, \mathbf{m_0}^{\mathcal{AN}_i})|_{P_{\mathcal{AN}_i} \setminus \mathcal{IP}_{A_j}}$.

Proof:
$\mathrm{L}(\mathcal{N}, \mathbf{m_0})|_{T \setminus T_{A_j}} \subseteq L(\mathcal{AN}_i, \mathbf{m_0}^{\mathcal{AN}_i})$. If we add the places of $\mathcal{IP}_{A_j}$ to $\langle \mathcal{N}, \mathbf{m_0} \rangle$ then $\mathrm{L}(\mathcal{N}, \mathbf{m_0})$ is preserved because all places of $\mathcal{IP}_{A_j}$ are IP with respect to

$\langle \mathcal{N}, \mathbf{m_0} \rangle$ preserving its steps (Corollary 23.5, taking into account that ${\mathbf{m_0}}^a[p] \geq m_0^{\min}(p)$). All sequences firable in this net are also firable in the net $\mathcal{AS}_i$ after the removing of transitions in T_{A_j}. This is because in $\mathcal{AS}_i$ we have removed all firing constraints appearing in $\langle \mathcal{N}, \mathbf{m_0} \rangle$ imposed by $\mathcal{N}_{A_j}$.

$\mathrm{L}(\mathcal{AN}_i, {\mathbf{m_0}}^{\mathcal{AN}_i}) \subseteq \mathrm{L}(\mathcal{N}, \mathbf{m_0})|_{T \setminus T_{A_j}}$. We prove this part by contradiction. Let σ be a sequence of $\mathrm{L}(\mathcal{AN}_i, {\mathbf{m_0}}^{\mathcal{AN}_i})$ for which there is no $\sigma' \in \mathrm{L}(\mathcal{N}, \mathbf{m_0})$ such that $\sigma = \sigma'|_{T \setminus T_{A_j}}$. Let σ_0 be the maximal prefix of σ for which there is a sequence $\sigma'_0 \in \mathrm{L}(\mathcal{N}, \mathbf{m_0})$ verifying $\sigma_0 = \sigma'_0|_{T \setminus T_{A_j}}$. If $\mathbf{m_0} \xrightarrow{\sigma'_0} \mathbf{m}$ and ${\mathbf{m_0}}^{\mathcal{AN}_i} \xrightarrow{\sigma_0} {\mathbf{m}}^{\mathcal{AN}_i}$, it is trivial to verify that $\mathbf{m}[p] = {\mathbf{m}}^{\mathcal{AN}_i}[p]$ for all $p \in (P \setminus P_{A_j})$. The next transition to σ_0, t, in σ must be an output transition of a sink place of $\mathcal{N}_{A_j}$, because these transitions are the unique transitions of $\mathcal{AS}_i$ with additional constraints to fire in $\langle \mathcal{N}, \mathbf{m_0} \rangle$. These constraints arise from $\mathcal{N}_{A_j}$ but not from the places $\mathcal{IP}_{A_j}$ because they are implicit with respect to $\langle \mathcal{N}, \mathbf{m_0} \rangle$. All maximal firable sequences in $\langle \mathcal{N}, \mathbf{m} \rangle$ containing only transitions of $\mathcal{N}_{A_j}$ never can enable the transition t because σ_0 is the maximal prefix of σ for which there is a sequence $\sigma'_0 \in \mathrm{L}(\mathcal{N}, \mathbf{m_0})$ verifying $\sigma_0 = \sigma'_0|_{T \setminus T_{A_j}}$. Let $\mathbf{m}'$ be a marking reachable in $\langle \mathcal{N}, \mathbf{m} \rangle$ firing a maximal sequence, σ_1, in $\langle \mathcal{N}, \mathbf{m} \rangle$ containing only transitions of $\mathcal{N}_{A_j}$. At $\mathbf{m}'$ there exists an empty path in the $\mathcal{N}_{A_j}$ from a source place to a sink place that inputs to transition t. In effect, at $\mathbf{m}'$ all transitions of $\mathcal{N}_{A_j}$ are not enabled, hence have at least one empty input place. Moreover, t has at least one empty input place being a sink place of $\mathcal{N}_{A_j}$ because t is not enabled at $\mathbf{m}'$. Therefore, t has an empty input place whose input transition has an empty input place, and so on, until we reach a source place of $\mathcal{N}_{A_j}$. This means that a place in $\mathcal{IP}_{A_j}$ corresponding to this path is an input place of t containing zero tokens, but this contradicts the hypothesis from which t is firable in $\mathcal{AS}_i$.

$\mathrm{RS}(\mathcal{N}, \mathbf{m_0})|_{P \setminus P_{A_j}} = \mathrm{RS}(\mathcal{AN}_i, {\mathbf{m_0}}^{\mathcal{AN}_i})|_{P_{\mathcal{AN}_i} \setminus \mathcal{IP}_{A_j}}$. To prove this, observe that $P \setminus P_{A_j} = P_{\mathcal{AN}_i} \setminus \mathcal{IP}_{A_j}$ from the definition of $\mathcal{AN}_i$. Taking into account the part (i) of this theorem, the stated equality of markings' sets holds. $\diamondsuit$

Corollary 23.10 *Let $\langle \mathcal{N}, \mathbf{m_0} \rangle$ be a strongly connected and live MG, $Q \subseteq P$ a cut of $\mathcal{N}$, and $\mathcal{BS}$ the basic skeleton system obtained from $\langle \mathcal{N}, \mathbf{m_0} \rangle$ by substituting the subnets $\mathcal{N}_{A_1}$ and $\mathcal{N}_{A_2}$ by the set of places $\mathcal{IP}_{A_1}$ and $\mathcal{IP}_{A_2}$, respectively, and ${\mathbf{m_0}}^{\mathcal{BN}}[p] =$ `if` $p \in \mathcal{IP}_{A_1} \cup \mathcal{IP}_{A_2}$ `then` ${\mathbf{m_0}}^a[p]$ `else` $\mathbf{m_0}[p]$.*

i) $\mathrm{L}(\mathcal{N}, \mathbf{m_0})|_{T \setminus (T_{A_1} \cup T_{A_2})} = \mathrm{L}(\mathcal{BN}, {\mathbf{m_0}}^{\mathcal{BN}})$.

ii) $\mathrm{RS}(\mathcal{N}, \mathbf{m_0})|_{P \setminus (P_{A_1} \cup P_{A_2})} = \mathrm{RS}(\mathcal{BN}, {\mathbf{m_0}}^{\mathcal{BS}})|_{P_{\mathcal{BN}} \setminus (\mathcal{IP}_{A_1} \cup \mathcal{IP}_{A_2})}$.

Proof:
The proof of the corollary can be decomposed into two steps: (1) The proof of the behaviour equivalence between $\langle \mathcal{N}, \mathbf{m_0} \rangle$ and $\mathcal{AS}_1$ (that is, the previous theorem); (2) The proof of the behaviour equivalence between $\mathcal{AS}_1$ and $\mathcal{BS}$. Taking into account that $\mathcal{AS}_i$ is a strongly connected and live MG, the proof

of this second part is the same that the above theorem renaming, for example, $\mathcal{AS}_1$ as $\langle \mathcal{N}, \mathbf{m_0} \rangle$ and $\mathcal{BS}$ as $\mathcal{AS}_2$. $\Diamond$

The main drawback of the above theorems concerns the great number (exponential in the worst case) of places in $\mathcal{IP}_{A_i}$. In the following we present a method to reduce the number of places to add, characterizing a subset of $\mathcal{IP}_{A_i}$, denoted $\mathcal{BIP}_{A_i}$, with the property that all places of $\mathcal{IP}_{A_i} \setminus \mathcal{BIP}_{A_i}$ are implicit with respect to the places $\mathcal{BIP}_{A_i}$. Therefore, in order to build the aggregated subnet we only add the set of places $\mathcal{BIP}_{A_i}$ instead of $\mathcal{IP}_{A_i}$.

Let us consider the aggregable subnet $\mathcal{N}_{A_i}$ together with its interface transitions. We derive from this net a directed graph $G_{A_i} = (V, E)$ in the same way to that presented at the end of previous section.

If we apply the algorithm of R.W. Floyd to solve the *all-pairs shortest paths* problem (see [2] for implementations of this algorithm) to the directed graph G_{A_i}, we obtain for each ordered pair of vertices (i.e., transitions) (t, t') the smallest length of any path from t to t', denoted length(t, t') (if this value is equal to ∞, there is no path from t to t'). Observe, that length$(t, t') = \min\{\sum_{p_j \in \pi} \mathbf{m_0}[p_j] \mid \pi$ is a path from t to $t'\}$. The computational complexity of this algorithm is $O(m^3)$, where m is the number of transitions of the considered net. From this values we define the set of places $\mathcal{BIP}_{A_i}$ as $\mathcal{BIP}_{A_i} = \{p \mid {}^\bullet p = \{t\}; p^\bullet = \{t'\}; t, t' \in T_Q; \text{length}(t, t') \neq \infty\}$.

For all $p \in \mathcal{BIP}_{A_i}$ we select an initial marking $\mathbf{m_0}[p] = \text{length}(t, t')$. It is trivial to verify that this initial marking coincides with the previously defined *aggregation's initial marking*, ${\mathbf{m_0}}^a[p]$. For instance, in the case of Fig. 23.1.b, $\mathcal{BIP}_{A_2} = \{beta\_1, beta\_2\}$ and $\mathbf{m_0}[beta\_1] = \mathbf{m_0}[beta\_2] = 0$.

The following result states that all places of $\mathcal{IP}_{A_i} \setminus \mathcal{BIP}_{A_i}$ are implicit with respect to the places $\mathcal{BIP}_{A_i}$. Therefore, in order to build the aggregated subsystem we only add the set of places $\mathcal{BIP}_{A_i}$ instead of $\mathcal{IP}_{A_i}$.

Property 23.11 *Each place $p \in \mathcal{IP}_{A_i} \setminus \mathcal{BIP}_{A_i}$ with an initial marking equal to ${\mathbf{m_0}}^a[p]$ is implicit with respect to the set of places $\mathcal{BIP}_{A_i}$ each one with an initial marking equal to the aggregation's marking.*

Proof:
Let $p \in \mathcal{IP}_{A_i} \setminus \mathcal{BIP}_{A_i}$ be a place obtained from the summation of the rows in the incidence matrix corresponding to the places of a path. Let $t, t' \in T_Q$ be the interface transitions of this path. Because of the existence of this path, after the application of the Floyd's algorithm we have length$(t, t') \neq \infty$, therefore there exists an identical place in $\mathcal{BIP}_{A_i}$ with the same initial marking. $\Diamond$

In many cases the set $\mathcal{BIP}_{A_i}$ is bigger than necessary because some places can be implicit in $\mathcal{AS}_i$. In order to remove one of these unnecessary places, p, we can apply the method described at the end of the previous section to compute the shortest path from ${}^\bullet p$ to $p^\bullet$. The place p can be removed if the output of this algorithm is less than or equal to the aggregation's marking of p. Observe that in the case of Fig. 23.2.b, the set $\mathcal{BIP}_{A_2}$ contains 16 places but a further

removing of places leads to a minimum set of 6 places, named β_i, $i = 1, \ldots, 6$ in the figure.

23.3 Iterative Throughput Approximation

In previous section, an algorithm to decompose an MG into two aggregated subsystems and a basic skeleton system (being also MG's) has been presented. In aggregated subsystem $\mathcal{AS}_i$ $(i = 1, 2)$, the subnet $\mathcal{N}_j$ $(j \neq i)$ is represented by the places in the cut Q, by the interface transitions of $\mathcal{N}_j$, $T_{I_j} = T_Q \cap T_j$, and by the new places that substitute the subnet $\mathcal{N}_{A_j}$.

The technique for an approximate computation of the throughput that we present now is, basically, a *response time approximation* method [1, 10, 11]. The interface transitions of $\mathcal{N}_j$ in $\mathcal{AS}_i$ approximate the response time of all the subsystem $\mathcal{N}_j$ $(i = 1, 2; j \neq i)$. A direct (non-iterative) method to compute the constant service rates of such interface transitions in order to represent the aggregation of the subnet gives, in general, low accuracy. Therefore, we are forced to define a *fixed-point search iterative process*, with the possible drawback of the presence of convergence and efficiency problems.

23.3.1 First Approach: Ping-Pong Algorithm

The first algorithm that we explored, called "Ping-Pong", follows.

```
select a cut Q;
derive aggregated subsystems AS_i, i = 1,2;
give value μ_t^0 for each t ∈ T_I1 in AS_2;
compute value of throughput χ_2^0 of AS_2;
k := 0;     {counter for iteration steps}
repeat
  k := k + 1;
  compute μ_t^k for each t ∈ T_I2 such that the throughput χ_1^k of
          AS_1 is close enough to χ_2^(k-1);
  compute μ_t^k for each t ∈ T_I1 such that the throughput χ_2^k of
          AS_2 is close enough to χ_1^k;
until convergence of χ_1^k and χ_2^k;
```

In the above procedure, once a cut has been selected and given some initial values for the service rates of interface transitions of $\mathcal{N}_1$ (which approximate the response time of all the subsystem $\mathcal{N}_1$), the underlying CTMC of aggregated subsystem $\mathcal{AS}_2$ is solved. From the solution of that CTMC, the first estimation χ_2^0 of the throughput of $\mathcal{AS}_2$ can be computed. Then, the initial estimated values of service rates of interface transitions that approximate the response time of subsystem $\mathcal{N}_2$ must be derived. This must be done in such a way that the throughput χ_1^1 of $\mathcal{AS}_1$ is "close enough" to χ_2^0. Then, a better estimation of rates μ_t^k for each $t \in T_{I_1}$ must be computed such that the throughput χ_2^1 of $\mathcal{AS}_2$ is close enough to χ_1^1. The process is iterated until χ_1^k and χ_2^k converge.

The first problem of the above sketch of approximation algorithm is that a *multidimensional search on the parameters* of a complex CTMC in order to get a given throughput cannot be done in an efficient way. A possible solution to this problem is the following. In the iterative process, each time that an aggregated subsystem $\mathcal{AS}_i, i = 1, 2$, is solved, *the ratios* among the service rates μ_t^k of all the transitions in T_{I_i} are estimated. After that, when the other subsystem $\mathcal{AS}_j, j \neq i$, is solved, only a *scale factor* for these service rates must be computed. The goal is to find a scale factor of μ_t^k for all $t \in T_{I_j}$ (and fixed k) such that the throughput of $\mathcal{AS}_j$ and the throughput of $\mathcal{AS}_i$, computed before, are the same. And this can be achieved with a linear search of the scale factor in $\mathcal{AS}_j$.

At this point, the main technical problem is the following: How to estimate from the solution of $\mathcal{AS}_i$ the ratios among the service rates of all transitions in T_{I_i} that in the next step (solution of $\mathcal{AS}_j$) will be scaled to obtain an approximation of the response time of the subsystem $\mathcal{N}_i$?

We explain our answer to this question by means of the example depicted in Fig. 23.1. Figure 23.1.b represents the aggregated subsystem $\mathcal{AS}_1$ derived from the original MG. It is necessary to compute the ratio between the service rate of $T2$ and $T3$ to be used as input data for the linear search of the scale factor in $\mathcal{AS}_2$ (Fig. 23.1.c). In order to do that, the aggregated subsystem $\mathcal{AS}_1$ is transformed (as depicted in Fig. 23.1.b) with the addition of places $\mathcal{BIP}_{A_1} = \{alph\_1, alph\_2\}$. The obtained system is behaviourally equivalent to $\mathcal{AS}_1$ because the added places (which are those that will substitute $\mathcal{N}_{A_1}$), are implicit. These new places allow to estimate the ratio between the "aggregated service times" of transitions $T2$ and $T3$ (representing the response time approximation of $\mathcal{N}_1$), as the quotient of the mean marking of *alph_1* by the mean marking of *alph_2*, because the throughput of all transitions is the same.

Now, two problems arise. First, the linear search of the scale factor must be done in the aggregated subsystems, that can have a considerably large state space, thus the efficiency of the method falls down. Additionally, we have found convergence problems in many cases. A solution for both problems is proposed in the next subsection.

23.3.2 A Solution: Pelota[1] Algorithm

The more practical solution of the problem we found makes use of the third system (another MG) derived from the original one, in previous section: *the basic skeleton.* The basic skeleton contains the interface subsystem and a simplified view (using the places $\mathcal{BIP}_{A_i}, i = 1, 2$, computed by the algorithm in previous section) of subsystems $\mathcal{N}_{A_i}, i = 1, 2$.

The idea is to use the basic skeleton as an intermediate point (*fronton*) between the two aggregated subnets (rackets), as explained in this algorithm:

[1] Game played by two players who use a basket strapped to their wrists or a wooden racket to propel a ball against a specially marked wall, called *fronton*.

```
select a cut Q;
derive AS_i, i = 1,2 and BS;
give initial value μ_t^0 for each t ∈ T_{I_2};
k := 0;    {counter for iteration steps}
repeat
  k := k + 1;
  solve aggregated subsystem AS_1 with
    input:   μ_t^{k-1} for each t ∈ T_{I_2},
    output:  ratios among μ_t^k of t ∈ T_{I_1} and χ_1^k;
  solve basic skeleton system BS with
    input:   μ_t^{k-1} for each t ∈ T_{I_2},
             ratios among μ_t^k of t ∈ T_{I_1}, and χ_1^k,
    output:  scale factor of μ_t^k of t ∈ T_{I_1};
  solve aggregated subsystem AS_2 with
    input:   μ_t^k for each t ∈ T_{I_1},
    output:  ratios among μ_t^k of t ∈ T_{I_2} and χ_2^k;
  solve basic skeleton system BS with
    input:   μ_t^k for each t ∈ T_{I_1},
             ratios among μ_t^k of t ∈ T_{I_2}, and χ_2^k,
    output:  scale factor of μ_t^k of t ∈ T_{I_2};
until convergence of χ_1^k and χ_2^k;
```

In this iterative process, each time that an aggregated subsystem $\mathcal{AS}_i, i = 1,2$, is solved, only the throughput χ_i^k and the ratios among the service rates μ_t^k of all the transitions in T_{I_i} are estimated (with the method explained in previous subsection). After that, a scale factor for these service rates must be computed. This is achieved by using the basic skeleton system $\mathcal{BS}$. The goal is to find a scale factor of μ_t^k for all $t \in T_{I_i}$ such that the throughput of the basic skeleton and the throughput of $\mathcal{AS}_i$, computed before, are the same. A linear search of the scale factor must be implemented, but now in a net system with considerably fewer states (the basic skeleton). In each iteration of this linear search, the basic skeleton is solved by deriving the underlying CTMC.

Now, the existence and uniqueness of the solution, and the convergence of the method should be addressed. Although no formal proof gives positive answers so far to the above questions, extensive testing allows the conjecture that there exists one and only one solution, computable in a finite number of steps, typically between 2 and 5 if the convergence criterion is that the difference between the two last estimations of the throughput is less than 0.1%.

23.4 Examples

In this section we present several numerical results of the application of the iterative technique previously introduced. Among all the tested examples, we have selected two different Petri net structures because of their following characteristics: the first one (already introduced in Fig. 23.1) is structurally asymmetric

$\mathcal{AS}_1$					$\mathcal{AS}_2$				
χ_1	tau_1	tau_2	tau_3	coeff	χ_2	rho_1	rho_2	rho_3	coeff
0.17352	0.05170	0.16810	0.88873	1.01167	0.12714	0.89026	0.21861	0.14354	0.98468
0.14093	0.06265	0.19707	0.91895	1.01318	0.13795	0.88267	0.21363	0.13509	0.98582
0.13856	0.06325	0.19821	0.92054	1.01306	0.13841	0.88239	0.21343	0.13467	0.98592
0.13844	0.06328	0.19827	0.92062	1.01306	0.13843	0.88237	0.21342	0.13465	0.98592
0.13843	0.06328	0.19827	0.92064	1.01307	0.13843	0.88238	0.21342	0.13465	0.98593

Table 23.1: Iteration results for the SMG in Fig. 23.1.

while the second has symmetries; for the second one, the effect of timing asymmetries on the iterative algorithm can be studied by changing the service rates of transitions (preserving the strong structural symmetry). In all cases, the obtained approximations are compared with exact values obtained from the numerical solution of the underlying CTMC (*GreatSPN* package was used [7]).

Let us consider again the SMG depicted in Fig. 23.1.a. The exact value of the throughput is equal to 0.138341 (if single-server semantics is assumed). The underlying CTMC has 89358 states. The aggregated systems $\mathcal{AS}_1$ and $\mathcal{AS}_2$ are depicted in Figs. 23.1.b and 23.1.c, respectively. The corresponding basic skeleton system is that in Fig. 23.1.d.

Table 23.1 shows the iterative results obtained for this example. The values in $\mathcal{AS}_1$ columns have been obtained from the solution of the aggregated system in Fig. 23.1.b: χ_1 is the throughput of $\mathcal{AS}_1$; columns *tau_1*, *tau_2*, and *tau_3* are the estimated values of the service rates of the aggregated transition tau_1, tau_2, and tau_3, computed in $\mathcal{AS}_1$; column *coeff* is the scale factor of previous estimated service rates, obtained by the linear search in the basic skeleton of Fig. 23.1.d. Columns related with $\mathcal{AS}_2$ represent the analogous values for the aggregated system in Fig. 23.1.c. Convergence of the method can be observed from the third iteration step. The error is -0.064333%, after the fifth step. The following additional fact must be remarked: the underlying CTMC's of $\mathcal{AS}_1$, $\mathcal{AS}_2$, and the basic skeleton have 8288, 3440, and 231 states, respectively, while the original SMG has 89358 states.

As a second example, let us consider the SMG depicted in Fig. 23.2.a. Any splitting of the net will generate two strongly coupled aggregated subnets. We select the following cut: $Q = \{P21, P22, P23, P24, P25, P26, P27, P28\}$. The corresponding aggregated systems are depicted in Figs. 23.2.b and 23.2.c. The basic skeleton is that in Fig. 23.2.d. The CTMC underlying the original SMG has 49398, while those underlying the aggregated systems have 6748. The basic skeleton has 771 reachable states.

We consider three different situations arising from different transition service rates (we assume infinite-server semantics in all cases). In the first case, we suppose that the service rates of all transitions are equal to 1.0. In this case, the exact throughput of the SMG is 0.295945.

Table 23.2 shows the iteration results for three different selections of initial values of aggregated service rates of transitions *rho_1*, *rho_2*, *rho_3*, and *rho_4*. It can be seen that in all cases convergence occurs at the third iteration step, independently of the initial values given to the aggregated service rates. This

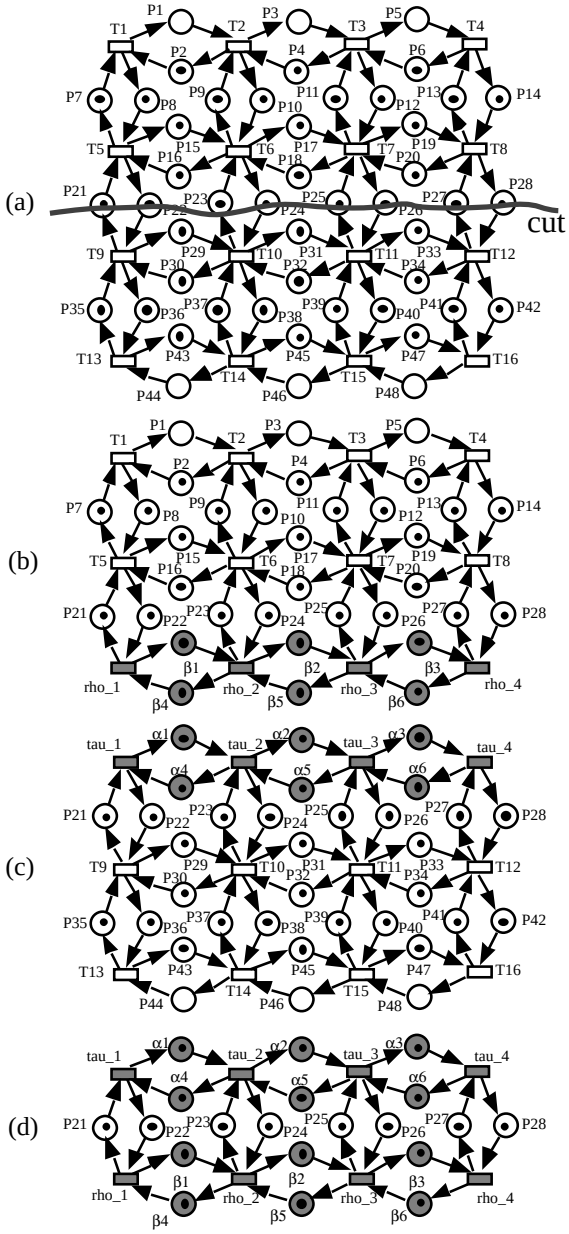


Figure 23.2: A second example of SMG and its decomposition.

Initial values of service rates of rho_1, rho_2, rho_3, and rho_4 equal to 0.1

$\mathcal{AS}_1$					$\mathcal{AS}_2$					
χ_1	tau_1	tau_2	tau_3	tau_4	coeff	χ_2	rho_1	rho_2	rho_3	rho_4	coeff
0.07614	1.02121	1.02452	1.01112	0.80930	1.06357	0.33294	0.29834	0.50973	0.61599	0.71668	1.03611
0.29244	0.84574	0.72462	0.55755	0.30802	1.05833	0.30079	0.29864	0.54035	0.70609	0.83610	1.06250
0.29710	0.84301	0.71383	0.54364	0.29813	1.06382	0.29733	0.29758	0.54270	0.71310	0.84299	1.06427
0.29711	0.84340	0.71354	0.54286	0.29751	1.06436	0.29711	0.29747	0.54281	0.71352	0.84343	1.06440

Initial values of service rates of rho_1, rho_2, rho_3, and rho_4 equal to 1.0

$\mathcal{AS}_1$					$\mathcal{AS}_2$					
χ_1	tau_1	tau_2	tau_3	tau_4	coeff	χ_2	rho_1	rho_2	rho_3	rho_4	coeff
0.33318	0.70982	0.61546	0.51044	0.29917	1.03518	0.29265	0.30871	0.55771	0.72423	0.84521	1.05804
0.30095	0.83571	0.70581	0.54034	0.29877	1.06233	0.29712	0.29817	0.54366	0.71378	0.84293	1.06378
0.29734	0.84296	0.71307	0.54270	0.29759	1.06425	0.29712	0.29751	0.54286	0.71354	0.84339	1.06436
0.29712	0.84343	0.71352	0.54281	0.29747	1.06440	0.29710	0.29746	0.54282	0.71354	0.84345	1.06439

Initial values of service rates of rho_1, rho_2, rho_3, and rho_4 equal to 10.0

$\mathcal{AS}_1$					$\mathcal{AS}_2$					
χ_1	tau_1	tau_2	tau_3	tau_4	coeff	χ_2	rho_1	rho_2	rho_3	rho_4	coeff
0.33419	0.68611	0.59756	0.49474	0.28053	1.03311	0.28561	0.30812	0.56325	0.73687	0.85741	1.06091
0.30136	0.83550	0.70455	0.53890	0.29791	1.06293	0.29679	0.29807	0.54392	0.71447	0.84356	1.06392
0.29735	0.84299	0.71304	0.54263	0.29753	1.06430	0.29710	0.29750	0.54287	0.71358	0.84343	1.06437
0.29711	0.84343	0.71352	0.54281	0.29747	1.06440	0.29710	0.29746	0.54282	0.71355	0.84346	1.06441
0.29710	0.84346	0.71355	0.54282	0.29746	1.06441	0.29710	0.29745	0.54281	0.71355	0.84346	1.06441
0.29710	0.84346	0.71356	0.54282	0.29746	1.06441	0.29710	0.29746	0.54282	0.71355	0.84346	1.06441

Table 23.2: Iteration results for the SMG in Fig. 23.2 with all service rates of transition equal to 1.0.

AS_1					AS_2					
χ_1	tau_1	tau_2	tau_3	tau_4	coeff	χ_2	rho_1	rho_2	rho_3	rho_4	coeff
0.33318	0.70983	0.61546	0.51045	0.29917	1.03519	0.34424	0.70118	1.49390	1.84123	1.92737	1.06187
0.33352	0.71500	0.60522	0.49835	0.28554	1.03637	0.33345	0.68342	1.50320	1.85362	1.93598	1.06255
0.33345	0.71616	0.60538	0.49834	0.28550	1.03656	0.33345	0.68281	1.50288	1.85352	1.93592	1.06251
0.33345	0.71621	0.60539	0.49834	0.28550	1.03656	0.33345	0.68278	1.50284	1.85348	1.93588	1.06249

(a)

AS_1					AS_2					
χ_1	tau_1	tau_2	tau_3	tau_4	coeff	χ_2	rho_1	rho_2	rho_3	rho_4	coeff
0.40526	1.64486	1.58029	0.60759	0.36042	1.00568	0.35214	0.36948	0.69530	0.61363	0.80667	1.07872
0.36392	1.81297	1.72253	0.66348	0.38291	1.01927	0.36239	0.37446	0.68764	0.59809	0.79673	1.08484
0.36326	1.80988	1.72268	0.66584	0.38570	1.01711	0.36321	0.37514	0.68748	0.59702	0.79565	1.08508
0.36328	1.80942	1.72245	0.66596	0.38596	1.01688	0.36328	0.37520	0.68748	0.59694	0.79556	1.08510
0.36329	1.80938	1.72243	0.66598	0.38599	1.01686	0.36329	0.37521	0.68747	0.59693	0.79555	1.08510
0.36329	1.80938	1.72243	0.66598	0.38599	1.01686	0.36329	0.37521	0.68748	0.59694	0.79555	1.08510

(b)

Table 23.3: Iteration results for the SMG in Fig. 23.2 with: (a) service rates of transition $T1$ to $T8$ equal to 1.0 and of transitions $T9$ to $T16$ equal to 2.0; and (b) service rates of transition $T1$, $T2$, $T5$, $T6$, $T9$, $T10$, $T13$, and $T14$ equal to 2.0, and the rest equal to 1.0.

fact illustrates the robustness of the method with respect to the seed. The error of the approximation in all cases is 0.4%.

As a second case, consider again the SMG in Fig. 23.2.a but now with asymmetric service rates associated with transitions. Assume that the service rates of transitions $T1$ to $T8$ are all equal to 1.0, while service rates of transition $T9$ to $T16$ are equal to 2.0. In this case the exact throughput of the original system is 0.333356. The iteration results are shown in Table 23.3.a. Now, the initial values of aggregated service rates of transitions *rho_1*, *rho_2*, *rho_3*, and *rho_3* are equal to 1.0. Convergence can be observed from the second iteration step and the error of the obtained value is 0.02%.

Finally, consider once more the SMG of Fig. 23.2.a, but now with the following service rates associated with transitions: the rates of $T1$, $T2$, $T5$, $T6$, $T9$, $T10$, $T13$, and $T14$ are equal to 2.0, while the rest are equal to 1.0. In this case, the exact throughput is 0.362586. The iteration results are shown in Table 23.3.b. Again, convergence can be observed from the second iteration step. The error is now 0.19%.

23.5 Bibliographic Remarks

The general approach for the approximate throughput computation of strongly connected stochastic marked graphs presented in this Chapter was introduced in [5, 6]. It generalized a previous technique based on net decomposition through a single input-single output cut [10], allowing the split of the model through any cut.

The divide and conquer principle, used here, underlies other works on the topic [3, 9, 10, 11, 13]). In [3], some particular queueing networks with subnetworks having *population constraints* were analyzed using *flow equivalent aggregation* (i.e., a non-iterative technique) and Marie's method [14] (the idea was to replace a subsystem by an equivalent exponential service station with load-dependent service rates obtained by analyzing the subsystem in isolation under a load-dependent Poisson arrival process). An alternative approach was presented in [13] to compute approximate throughput for SMG's. In that work, the original system was also *split in subsystems* and a *delay equivalence* criterion was used for throughput approximation. The service rates for the aggregated subsystems were *marking dependent*. In [10], *response time approximation* was applied for an iterative computation of the throughput of SMG's.

A discussion of the above recalled techniques was presented in [11] for the throughput approximation of SMG's. We summarize now some of the conclusions. Flow equivalent aggregation is clearly the most efficient method (it is not an iterative method). In this method, the behaviour of the subsystem is assumed to be independent of the arrival process and depends only on the number of customers in the system. In many cases, this assumption is violated (see [10]), therefore the method cannot be applied. Marie's method behaves correctly in many cases. As with many iterative methods, the uniqueness of the solution cannot be proven although numerical experience has shown that a unique point

does indeed exist. The main drawback is that convergence sometimes presents a problem [4]. Concerning the delay equivalence technique presented in [13], its convergence may sometimes constitute a problem. The robustness of the method was improved in [12], where the service rates of the aggregated subsystems were made constant. Some problems of this approach were reported in [10] where it was shown that the speed of convergence strongly depends on the initial values estimated for the service rates that represent the aggregated subsystem.

An obvious generalization of the technique presented in this Chapter can be derived if the original system is partitioned into more than two subsystems, leading to the classical tradeoff between efficiency and accuracy. Concerning the extension of the technique presented here to more general net subclasses, in [18] the case of *weighted T-systems*, the weighted extension of marked graphs, was studied. The application of the technique to *Deterministically Synchronized Sequential Processes* (DSSP) was achieved in [16, 19, 17]. Finally, in [15], the technique was generalized to *arbitrary* stochastic P/T systems.

Bibliography

[1] S. C. Agrawal, J. P. Buzen, and A. W. Shum. Response time preservation: A general technique for developing approximate algorithms for queueing networks. In *Proceedings of the 1984 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, pages 63–77, Cambridge, MA, August 1984.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[3] B. Baynat and Y. Dallery. Approximate techniques for general closed queueing networks with subnetworks having population constraints. *European Journal of Operational Research*, 69:250–263, 1993.

[4] B. Baynat and Y. Dallery. A unified view of product-form approximation techniques for general closed queueing networks. *Performance Evaluation*, 18(3):205–224, November 1993.

[5] J. Campos, J. M. Colom, H. Jungnitz, and M. Silva. A general iterative technique for approximate throughput computation of stochastic marked graphs. In *Proceedings of the* 5^{th} *International Workshop on Petri Nets and Performance Models*, pages 138–147, Toulouse, France, October 1993. IEEE-Computer Society Press.

[6] J. Campos, J. M. Colom, H. Jungnitz, and M. Silva. Approximate throughput computation of stochastic marked graphs. *IEEE Transactions on Software Engineering*, 20(7):526–535, July 1994.

[7] G. Chiola. A graphical Petri net tool for performance analysis. In *Proceedings of the* 3^{rd} *International Workshop on Modeling Techniques and Performance Evaluation*, Paris, France, March 1987. AFCET.

[8] J. M. Colom. *Análisis Estructural de Redes de Petri, Programación Lineal y Geometría Convexa*. PhD thesis, Departamento de Ingeniería Eléctrica e Informática, Universidad de Zaragoza, Spain, June 1989. Research Report. GISI-RR-89-11. In Spanish.

[9] A. Desrochers, H. Jungnitz, and M. Silva. An approximation method for the performance analysis of manufacturing systems based on GSPNs. In *Proceedings of the Rensselaer's Third International Conference on Computer Integrated Manufacturing*, pages 46–55, Rensselaer Polytechnic Institute, Troy, NY, USA, May 1992. IEEE-Computer Society Press.

[10] H. Jungnitz, B. Sánchez, and M. Silva. Approximate throughput computation of stochastic marked graphs. *Journal of Parallel and Distributed Computing*, 15:282–295, 1992.

[11] H. J. Jungnitz. *Approximation Methods for Stochastic Petri Nets.* PhD thesis, Dept. of Electrical, Computer and Systems Engineering, Rensselaer Polytechnic Institute, Troy, NY, USA, May 1992.

[12] Y. Li and C. M. Woodside. Performance Petri net analysis of communications protocol software by delay-equivalent aggregation. In *Proceedings of the* 4^{th} *International Workshop on Petri Nets and Performance Models*, pages 64–73, Melbourne, Australia, December 1991. IEEE-Computer Society Press.

[13] Y. Li and C. M. Woodside. Complete decomposition of stochastic Petri nets representing generalized service networks. *IEEE Transactions on Computers*, 44(8):1031–1046, August 1995.

[14] R. A. Marie. An approximate analytical method for general queueing networks. *IEEE Transactions on Software Engineering*, 5(5):530–538, September 1979.

[15] C. J. Pérez-Jiménez and J. Campos. A response time approximation technique for stochastic general P/T systems. In *Proceedings of the 2nd IMACS International Multiconference on Computational Engineering in Systems Applications (CESA'98)*, Hammamet, Tunisia, April 1998. IEEE Systems, Man and Cybernetics.

[16] C. J. Pérez-Jiménez, J. Campos, and M. Silva. On approximate throughput computation of deterministic systems of sequential processes. In *Actas de las IV Jornadas de Concurrencia*, pages 156–171, El Escorial, Spain, June 1995.

[17] C. J. Pérez-Jiménez, J. Campos, and M. Silva. Approximate throughput computation of a class of cooperating sequential processes. In *Proceedings of the Rensselaer's Fifth International Conference on Computer Integrated Manufacturing and Automation Technology (CIMAT'96)*, pages 382–389, Grenoble, France, May 1996.

[18] C. J. Pérez-Jiménez, J. Campos, and M. Silva. On approximate performance evaluation of manufacturing systems modelled with weighted T-systems. In *Proceedings of the IMACS/IEEE-SMC Multiconference on Computational Engineering in Systems Applications (CESA'96)*, pages 201–207, Lille, France, July 1996.

[19] C. J. Pérez-Jiménez, J. Campos, and M. Silva. State machine reduction for the approximate performance evaluation of manufacturing systems modelled with cooperating sequential processes. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 1159–1165, Minneapolis, Minnesota, USA, April 1996.

Part VII

Simulation

Chapter 24

Simulation Principles

24.1 Simulation Modelling and Analysis

We consider a *simulation* (or computer simulation) as the execution of an abstraction (or *system model*) of an existing or nonexisting real system by means of (computer) software. As an alternative to the analysis techniques developed in previous chapters, simulation appears as a good compromise between mathematical/analytical models often being too abstract with respect to assumptions on system behavior, and the need for building experimental real systems often being too expensive. It supplements the mathematical/analytical modeling approach in cases where the real system is not in existence so that measurements cannot be applied, but a realistic abstraction of the hypothecial real system behavior cannot be expressed or evaluated analytically. On the other hand it supplements measurements whenever they are too costly, dangerous, inapropriate or impossible.

To understand the operation of a real system, to evaluate its performance, to select and tune design parameters, to verify and validate desgins, to compare alternatives, etc., the analyst is generally interested in its *behaviour*, i.e. the output it generates based on a certain input (see Figure 24.1).

For the development of simulation software it is now important, to adequately express a *real system* in a *system model*, the abstraction process of which we call *system modeling*. Accordingly, the *input* to the real system requires an appropriate representation for or within the *system model*. Issues related to the modeling the input for a system simulation is referred to *input modelling* in the literature [16, 51]. The analysis of the model output usually involves thorough statistical investigation of the output data, the exploration of large

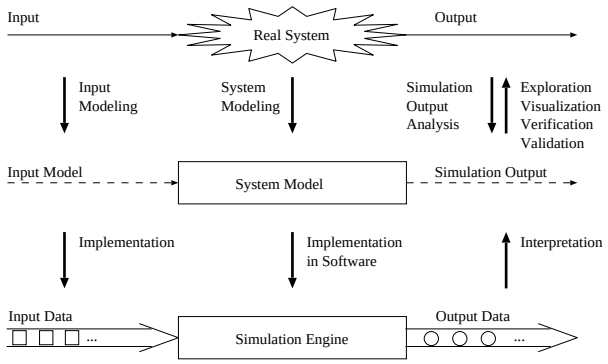


Figure 24.1: Computer Simulation

data spaces, the appropriate visualization, and the verification and validation of simulation experiments. *Simulation output analysis* [48] is the term in the literature addressing those issues.

Once a *system model* with the respective *input model* has been developed or defined, the implementation of those models into software becomes topical. Generally, a *simulation engine* or simulation program is implemented taking input data to "drive" the execution, and by that generating *output data* to be interpreted as the simulation output (see Figure 24.1).

24.1.1 System Models

Basically, a *system model* is a specification of phenomena present in the real (physical) system (or at least some of its components) in terms of a set of *states* and *events* – at an appropriate level of abstraction or detail. In its software implementation (*simulation engine*), states are usually realized as *(state) variables*, who's values define the current state of the simulated system, whereas *events* are usually software components (often called "event procedures") that change the values of *state variables*, thus indicating the chage of the state of the system. From a software technology viewpoint, performing a simulation thus means mimicking the occurrence of events as they evolve, and recognizing their effects as represented by states.

Simulation as a research area and engineering discipline has established a whole spectrum of different kinds of system models. In simulations of *dynamic system models* the evolution of system states is considered over *time*, whereas time is not a variable in a *static system model* simulation. In *time* oriented simulations, the primary distinction is with respect to the metric of the time

and the state parameter. In *continuous-time* system models time is considered real valued, and the system state is defined at all times (e.g. the number of tokens in place of an SPN). A *discrete-time* system model implements time as a discrete variable, and the system state is defined at the respective time instants only. Accordingly, state variables have continuous values in a *continuous-state* system model, and integer values in a *discrete-state* system model.

With respect to the input-output relation of a simulation we can distinguish *deterministic system models* where simulation output can be predicted with certainty, from *probabilistic system models* where one and the same input can produce a manifold of different outputs. If the simulation output is a linear dependency of the input, the literature talks about *linear system models*, and about *nonlinear system models* in other cases.

24.1.2 Types of Simulation

For the performance analyst, three different types of *simulation engines*, i.e. software implementations of system models have become prevalent.

Monte Carlo simulations are used to simulate systems of probabilistic nature, that do not change characteristics over time. As such, a Monte Carlo simulation appears as a *static simulation*, with no explicit time axis present in the system model. As an illustration take the integral evaluation via Monte Carlo simulation as developed in [43]: In order to evaluate $I = \int_0^2 e^{-x^2} dx$ we can consider a uniformly distributed random number x with probability density function $f(x) = 1/2$ iff $0 \leq x \leq 2$, and compute another random variable y as $y = 2e^{-x^2}$. it appears that the expected value of y is $E(y) = \int_0^2 2e^{-x^2} f(x) dx = \int_0^2 2e^{-x^2} \frac{1}{2} dx = \int_0^2 e^{-x^2} dx = I$, the value of the integral. We can thus evaluate the integral I via simulation, or more precisely, by drawing random variables $x_i \sim$ Uniform$(0, 2)$, transforming them to $y_i = 2e^{-x_i^2}$, and computing the expected value $E(y) = \frac{1}{n} \sum_{i=1}^{n} y_i = I$. Note again, that the concept of time is not present in this kind of simulation (*static simulation*).

A very important approach for *dynamic simulations* is the use of a time ordered sequence of (real) events as the input to a simulation. In a *trace driven simulation*, an execution trace collected during the execution of the real system is used to "drive" the simulation engine. This simulation methodology has become particularly popular for the analysis and tuning of resource management algorithms in operating systems, like e.g. CPU scheduling or caching. The main advantage of trace driven simulation has been considered to be the preservation of correlation and coinfluence of effects in workloads when using e.g. real resource

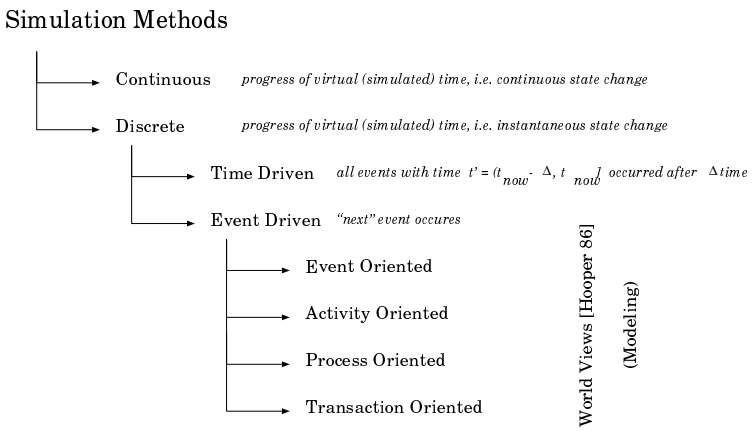


Figure 24.2: "World Views" and Simulation Paradigms

reference patterns (as given by the trace). The high level of detail potentially attainable with trace driven simulations however at the same time introduces the problem of complexity and tractability (very long traces).

An *event driven simulation* uses a discrete state model and an event structure as described in the system model to "drive" the simulation. As opposed to trace driven simulations therefore, causal dependencies among events and their occurrence in time are responsible for triggering the simulation engine rather than an external (event) trace. An event driven simulation engine hence employs an *event scheduler* which maintains a time ordered linked list of events waiting to happen, a *simulation clock* which advances in time units or time jumps, *state variables* reflecting the system state, and event routines to simulate event occurrences. Because of its natural conformance to the discrete event nature of Petri net (system) models, the rest of this text will be focused on discrete event simulation.

24.1.3 World Views

A variety of orthogonal views with respect to simulation has been developed in the literature, which are summarized in Figure 24.2 as "Simulation Methods" [42]. Related to our terminology, a simulation method is the software model underlying the *system model* and implemented in the *simulation engine*.

In this classification, a discrete event simulation can be triggered by the progression of time (*time-driven*) or by the occurrence of events (*event-driven*) (we will develop an example for both in the context of Petri net simulations).

The "translation" (or better: modelling) process of *real system* phenomena into *system models* is considered to happen along the events that can occur in reality (*event oriented*), along the duration of activities as defined by the time interval between a "start-activity" and "end-activity" event (*activity oriented*), along the real world "processes" that the change system state (*process oriented*), or along the transactions accepted and executed by a real world entity (*transaction oriented*).

Historically, the event oriented "world view" has significantly influenced the design and implementation of simulation engines for its clear separation of event structure and system behavior descriptions, the static model components (objects) like e.g. a FCFS queue, and the dynamic model components (events) like e.g. a customer arrival. Event types are identified and program routine is developed for every event type. Each event routine implements the modification of object attributes upon an event occurrence, and a control program (event scheduler) "calls" those event routines. Execution speed attained via efficient event scheduler implementations has always been a favoring argument for event oriented simulations.

Quite recently, with the upcoming object oriented software development methodologies, the process oriented "world view" has gained considerable attention. The behavior of system model components are expressed as *process* and coded as *objects* in the object oriented sense. As such, *processes* (objects) appear as collections of behaviors (methods). Processes (objects) can become `active`, can `hold` for a certain period of time, or can `await` the occurrence of an event, etc. Process suspension (or reinvocation) is handled by the runtime system. The process oriented "world view" convincingly conforms an object oriented *system model* development process, which seamlessly supports object oriented programming methodologies. Poor execution performance has been considered a potential liability of this approach.

24.2 PNs and Event Oriented System Models

The PN execution semantics (e.g. transition firing policies) as developed earlier in this text in a natural way correspond to an event oriented world view. The occurrence of events in a a *real system* are modeled by the transition firing occurrences (or token arrivals, etc.) in a PN based *system model*. For the *quality* analysis of systems the consideration of a nontimed PN was sufficient (see Chapter ??). To make the PN formalism adequate also for *quantitative* (i.e. performance) analysis, finite timing of activities can be expressed by associating

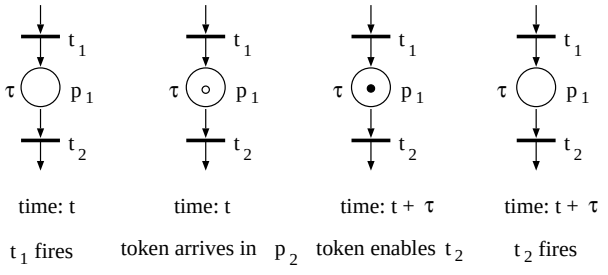


Figure 24.3: Firing in Place-Timed PNs (PTPNs)

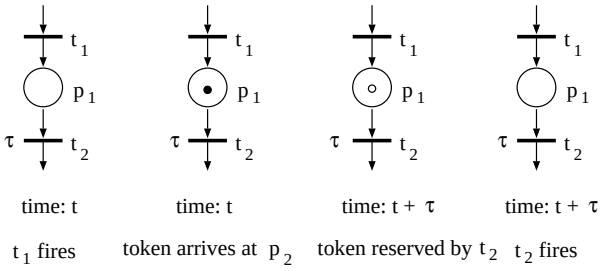


Figure 24.4: Firing in Transition-Timed PNs (TTPNs)

a time concept to places, transitions, arcs, tokens, or any combination of them. Although several of them have been studied throughout this text, we shall review some of them from a discrete event world view point. We shall first look at some notions of time and their impact on discrete event simulations of PNs, and then come to the expressive power of the formalism to serve as a system model to support discrete event simulations.

24.2.1 Notions of Time

Popular approaches are to assign *firing times* to transitions [69] representing the time interval during which a corresponding activity is going on, or *holding times* to places [72] modeling the time expired between the occurrence of events (an arriving token has to reside in the place for the holding time before it can enable a transition). Several equivalences among timing notions have been proven [25].

Figure 24.3 shows the firing semantics in a Place-Timed PN, i.e. $\tau : P \mapsto I\!R$ assigns *holding times* τ_i to places $p_i \in P$, whereas Figure 24.4 shows the firing for a Transition-Timed PN, i.e. $\tau : T \mapsto I\!R$ assigns *firing delays* τ_i to transitions $t_i \in T$. Figure 24.5 demonstrates how PTPNs can be transformed to TTPNs of

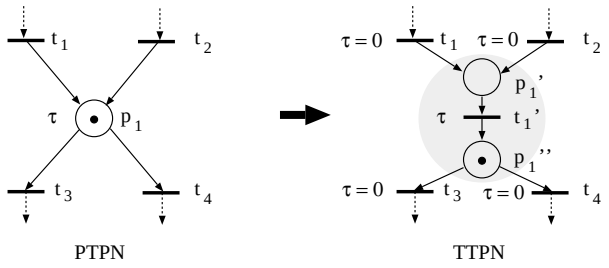


Figure 24.5: Equivalence Transformation from PTPN to TTPN

equivalent modelling power.

24.2.2 Timed Enabling and Firing Semantics

Important for modeling, analysis and simulation is how the timing concept attached to PNs modifies the enabling and firing rule of PNs: If $\forall p \in I(t)$, $\mu(p) \geq c\ w((p,t))\ c > 1$ in μ then t is said to be *multiply enabled* at degree c. If t can only "fire" (serve) one enabling at a time, we talk about *single server* (SS) (enabling) semantics, and *infinite server* (IS) semantics if *any* amount of enablings can be served at a time. If t_1 in the SPN in Figure 24.7 followed SS, then the *occurrence time* $ot(t_1(\bullet_i))$ of t_1 with the i-th token out of the four in p_1 would be $ot(t_1(\bullet_i)) = ot(t_1(\bullet_{i-1})) + X_i$, where $X_i \sim exp(\lambda_1)$ is an exponential variate. If t_1 applied IS, then $ot(t_1(\bullet_i)) = X_i \quad \forall \bullet_i$, expressing a notion of parallelism among the tokens. The latter is useful in our interpretation of the net in Figure 24.7 where tokens model machines subject to failure (t_1) and repair (t_2). Note that for memoryless distributions, a t with IS can be modeled as an appropriately "faster" SS transition by choosing a marking dependent rate, e.g. $\mu_1\lambda_1$ for t_1. In a PN with IS, SS can always be modeled by assigning to transitions a loop-back place marked with a single token.

Both the enabling and the firing can be timed in a TPN. Timing the firing is straightforward. If the enabling is timed, then for SMs, FCNs and GNs (but not for MGs and CFNs) two questions arise related to timing: (i) should it be allowed to preempt the enabling, and if so, (ii) how should intermediate "work" be memorized. In a policy with non-preemptive enabling semantics, once t becomes enabled it is decided whether it should fire. If t fires, it removes tokens from $I(t)$ (*start-firing event*), "hides" tokens during the firing period, and after that releases tokens to $O(t)$ (*end-firing event*). This policy is know as *preselection* (PS), since in cases of conflict among t and t' one of them is

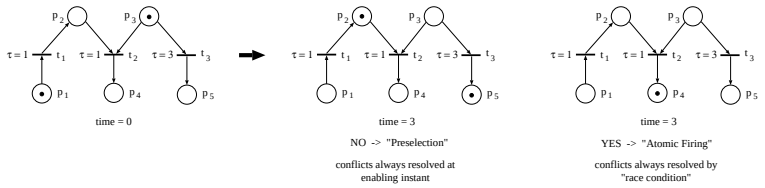


Figure 24.6: SMs, FCNs, GNs: Preemption possible? How Memorize?

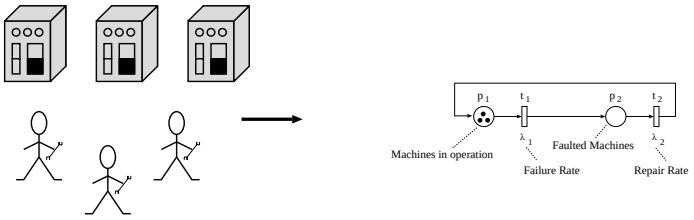


Figure 24.7: A TPN with Model Parallelism

preselected for firing. *Atomic firing* (AF) refers to a policy where t, once it becomes enabled, its firing is delayed to the instant when the enabling time $\tau(t)$ has expired. If t is still enabled at that time, it fires exactly like in a PN. For the case where t looses enabling during that period, either a new, *resampled* $\tau(t)$, or the non expired part of $\tau(t)$ (*age memory*) is used for the eventual new enabling of t. While with PS conflicts (among timed transitions) are resolved by predefined mechanisms like random switches or priorities (from "*outside*"), AF resolves (timed) conflicts always and entirely by the race condition (from "*inside*"). Real systems with race conditions, like timers controling activities, *cannot* be modeled by TPNs with PS. (See Figure 24.6 for an illustration.)

Note: Opposed to timing the enabling, timing the firing does not impose any difficulty (always non-preemptive).

24.2.3 PNs and Discrete Event System Models

The execution of a TPN can now be interpreted as a time dynamic discrete event system in various different ways. If oriented along the places $p \in P$, the token *arrival* and *departure* events (in case of the PS policy also the token *reservation* events), characterize the dynamic behavior of a system. More natural is the transition oriented view, where transition firings are related to event occurrences, while places represent the conditions that need to hold for the event to occur; e.g. in Figure 24.7 the firing of t_1 represents the occurrence

of a "*machine failure*" event, etc. In the sequel therefore, we shall consider only "transition firings" (in a TPN based system model) to represent events in real systems. This is no restriction with respect to the generality of the presentation of TPN simulations, i.e. system models could equivalently be based on events like "token arrivals", "enabling preemptions", etc.

For the development of TPN based system models considering "transition firings" as the underlying events, event structures can be derived from the TNP graph and the respective transition firing semantics.

Figure 24.8 recalls modelling components that encode event structures like event sequencing, event interleaving, mutual exclusion, etc. by relating the transition firings of t_i to event occurrences e_i in an event structure $[\ldots, e_i, \ldots]$

24.2.4 Why Discrete Event Simulation of PNs?

Both the net class as well as the timing notion in PNs have impact on their qualitative and quantitative analysis. E.g., working with PN system models for *realistically sized* systems one frequently arrives at the limits of practical tractability of the analytical evaluation. Therefore, and due to the limitations of modeling power of net classes with tractable analytical solutions, simulation often remains as the only practical technique to obtain performance indices. The tables in Figure 24.9 summarizes this situation. Taking just the class of timed Petri nets having an underlying stochastic process which is Markov [59] semi-Markov [1] or semi-regenerative [20] as an argument, it requires the knowledge of the infinitesimal generator Q (to solve e.g. for $\vec{\pi}Q = \vec{0}$ subject to $\sum_{s_i \in RS(\mu^{(0)})} \pi_i = 1$), i.e. the enumeration of all states $RS(\mu^{(0)})$ reachable from the initial marking $\mu^{(0)}$. $RS(\mu^{(0)})$, however, is potentially exponential in the number of places $|P|$ and the number of tokens in the initial marking, yielding a space and a computational problem for analytical evaluation. The complexity of generating the state space is reflected in the computational complexity of deciding whether given a PN and a marking μ: $\exists \sigma \; s.t. \; \mu^{(0)} \xrightarrow{\sigma} \mu$? ("reachability problem"). Although decidable [57], this problem is known to be exp-space-complete for symmetric PNs, Pspace-complete if $|I(t)| = |O(t)| = c \;\; \forall t \in T$ and for *safe* PNs, NP-complete for PNs without cycles and CFNs, and polynomial only for bounded CFNs, for MGs, and for live, bounded, strongly connected FCNs.

Having motivated the need for simulations as an evaluation mechanism for PNs, we can now switch to the respective discrete event simulation concepts.

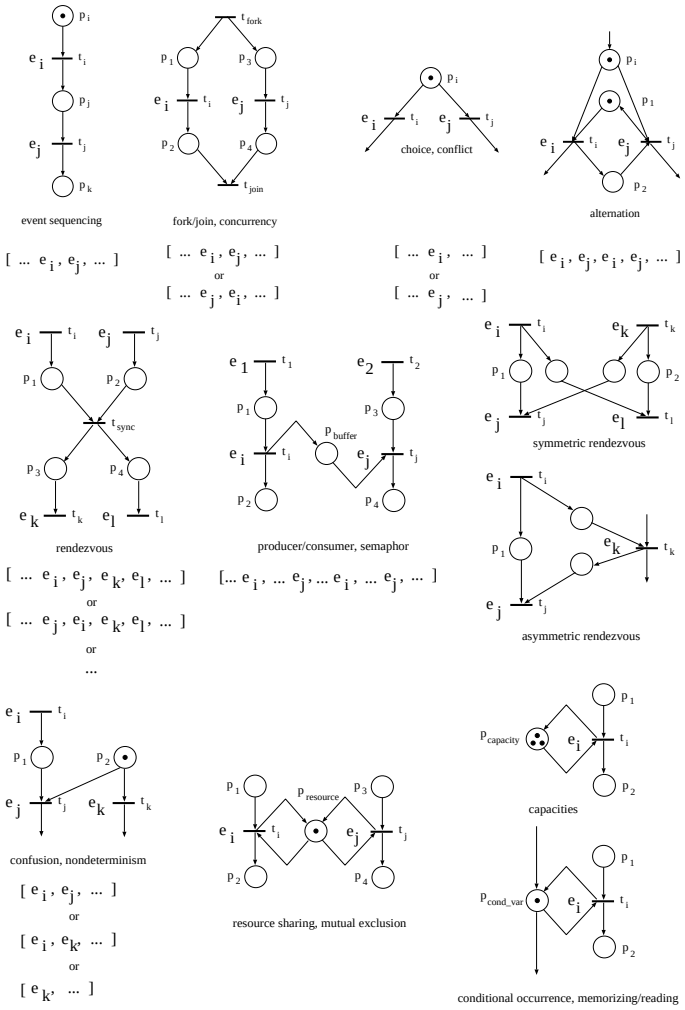


Figure 24.8: TPN Models and Event Structures

class	Definition	Modeling Power
State Machines (SM)	$\|I(t)\| = \|O(t)\| = 1 \quad \forall t \in T$	concurrency, conflict
Marked Graphs (MG)	$\|I(p)\| = \|O(p)\| = 1 \quad \forall p \in P$	concurrency, synchronization
Conflict Free nets (CFN)	$\|O(p)\| = 1 \quad \forall p \in P$	concurrency, synchronization, conflict excluded
Free Choice nets (FCN)	$O(p_i) \cap O(p_j) \neq \emptyset \Rightarrow$ $\|O(p_i)\| = \|O(p_j)\| = 1 \quad \forall p_i, p_j \in P, i \neq j$	concurrency, conflict, synchronization
General nets (GN)	no restriction	concurrency, conflict, synchronization, confusion

Class	Type of Enabling Delay $\tau(t_i)$	Analysis Techniques
nontimed PN	$\tau(t_i) = 0$	no quantitative analysis possible
Timed MG	$\tau(t_i) \in$ $\{\mathbb{N}, \mathbb{R}, \text{geom}(\alpha), \exp(\lambda), \text{discr.-general}, \text{cont.-general}\}$	recurrence equations
Timed GN	$\tau(t_i) \in$ $\{\mathbb{N}, \mathbb{R}, \text{geom}(\alpha), \exp(\lambda), \text{discr.-general}, \text{cont.-general}\}$	simulation
Discr./Cont. Time Stochastic PN	$\tau(t_i) \sim \text{geom}(\alpha) \ \forall \ t_i \in T$ $\tau(t_i) \sim \exp(\lambda) \ \forall \ t_i \in T$	isomorphic to DTMC/CTMC standard steady-state/transient anal.
Discr./Cont. Time Generalized SPN	$\tau(t_i) \in \{0, \text{geom}(\alpha)\} \ \forall \ t_i \in T$, "memoryless" RET $\tau(t_i) \in \{0, \exp(\lambda)\} \ \forall \ t_i \in T$, "memoryless" RET	isomorphic to DTMC/CTMC standard steady-state/transient anal.
Non-memoryless RET D/C GSPNs	as with D/C GSPNs, but firingpolicy with absence of memory at regeneration points	semi-Markov (DTMC/CTMC) analysis on embedded MC
Deterministic and Stochastic PN	$\tau(t_i) \in \{0, \mathbb{R}, \exp(\lambda)\}$, at most 1 determ. transition enabled in each marking	steady-state analysis by method of supplementary variables
GN	$\tau(t_i) = f(\mu)$ (user defined, arbitrary timing)	simulation

Figure 24.9: Modeling Power versus Complexity of Solution

24.2.5 Discrete Event Simulation and PNs

In a *discrete event* simulation the occurrence of an event is instantaneous and fixed to a particular point in time. Two kinds of discrete simulations of TPNs appear possible (distinguished with respect to the way simulation time is progressed): In *time driven* discrete simulation simulated time is advanced in *time steps* (or ticks) of constant size Δ, or in other words the, observation of the simulated dynamic system is discretized by unitary time intervals. The choice of Δ interchanges simulation accuracy and elapsed simulation time: ticks short enough to guarantee the required precision generally imply longer simulation time. Intuitively, for event structures irregularly dispersed over time, the time-driven concept generates inefficient simulation algorithms.

Event driven discrete simulation discretizes the observation of the simulated system at event occurrence instants. We shall refer to this kind of simulation as *discrete event simulation* (DES) subsequently. A DES, when executed sequentially repeatedly processes the occurrence of events in simulated time (often called *"virtual time"*, VT) by maintaining a time ordered *event list* (EVL) holding timestamped events scheduled to occur in the future, a (global) *clock* indicating the current time and *state variables* $S = (s_1, s_2, \ldots s_n)$ defining the

current state of the system (see Figure 24.10). A *simulation engine* (SE) drives the simulation by continuously taking the first event out of the event list (i.e. the one with the lowest timestamp), simulating the effect of the event by changing the state variables and/or scheduling new events in EVL – possibly also removing obsolete events. This is performed until some pre-defined endtime is reached, or there are no further events to occur.

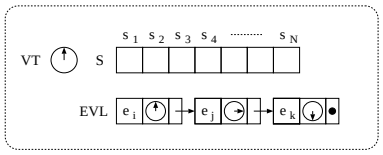


Figure 24.10: Simulation Engine for Discrete Event Simulation

As an example (see Figure 24.11), assume a *real system* of two machines participating in a manufacturing process. In a preprocessing step, one machine produces two subparts A1 and A2 of a product A, both of which can be assembled concurrently. Part A1 requires a single, whereas A2 takes a nonpredictable amount of assembly steps. Once one A1 and one A2 are assembled (irrespective of the machine that assembled it) one piece of a A is produced.

The system is modelled in terms of a TPN: transition t_1 models the preprocessing step and the forking of independent postprocessing steps, t_2 and t_3. Machines in the preprocessing phase are represented by tokens in p_1, finished parts A1 by tokens in p_5 and finished or "still in assembly" parts A2 by tokens in p_4. Once there is at least one token in p_5 and at least one token in p_4, the assembly process stops or repeats with equal probability (conflict among t_4 and t_5). Once the assembly process terminates yielding one A, one machine is released (t_5). The time behavior of the real system is modelled by associating timing information to transitions ($\tau(t_1) = 3$, $\tau(t_2) = \tau(t_3) = 2$ and $\tau(t_4) = \tau(t_5) = 0$). This means that a transition t_i that became enabled by the arrival of tokens in the input places at time t and remained enabled (by the presence of tokens in the input places) during $[t, t + \tau(t_i))$. It fires at time $t + \tau(t_i)$ by removing tokens from input places and depositing tokens in t_i's output places. The initial state of the system is represented by the marking of the PN where place $p1$ has 2 tokens (for 2 machines), and no tokens are available in any other place. Both the time driven and the event driven DES of the PN are illustrated in Figure 24.12.

The time driven DES increments VT (denoted by a watch symbol in the table) by one time unit each step, and collects the state vector S as observed at

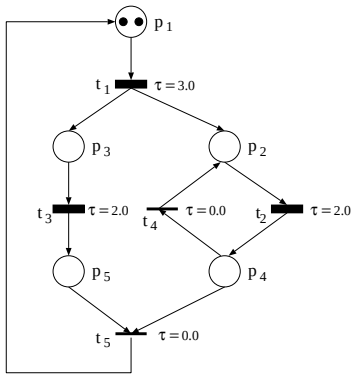


Figure 24.11: A Sample Simulation Model described as Timed Petri Net

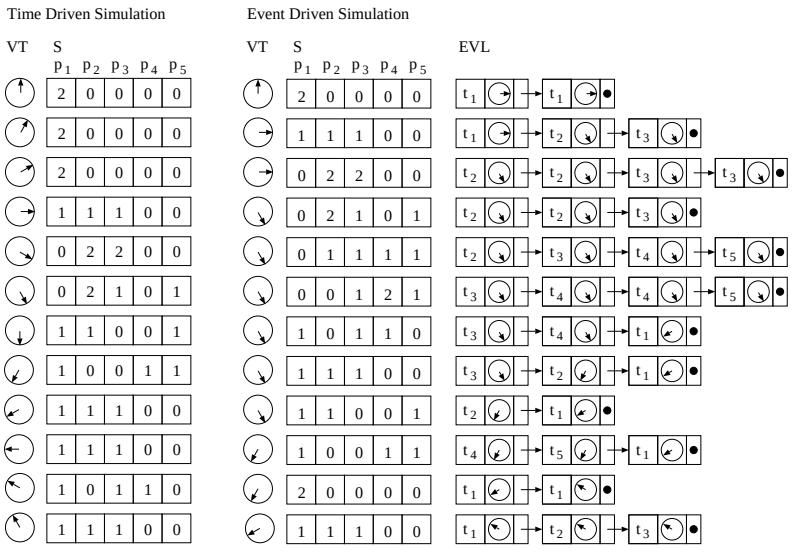


Figure 24.12: Event Driven vs. Time Driven Execution of the Simulation Model

that time. Due to time resolution and non time consuming state changes of the system, not all the relevant information could be collected with this simulation strategy.

The event driven DES employs a simulation engine as in Figure 24.10 and exploits a natural correspondence among event occurrences in the real system and transition firings in the PN model by relating them: whenever an event occurs in the real (physical) system, a transition is fired in the model. The event list hence carries transitions and the time instant at which they will fire, given that the firing is not preempted by the firing of another transition in the meantime. The state of the system is represented by the current PN marking (S), which is changed by the processing of an event, i.e. the firing of a transition: the transition with the smallest timestamp is withdrawn from the event list, and S is changed according to the corresponding token moves. The new state, however, can enable new transitions (in some cases maybe even disable enabled transitions), such that EVL has to be corrected accordingly: new enabled transitions are scheduled with their firing time to occur in the future by inserting them into EVL (while disabled transitions are removed). Finally the VT is set to the timestamp (ts) of the transition just fired.

Related now to the example in Figure 24.11 we have: Before the first step of the simulation starts, VT is set to 0 and transition t_1 is scheduled twice for firing at time $0 + \tau(t_1) = 3$ according to the initial state $S = (2, 0, 0, 0, 0)$. There are two identical event entries with identical timestamps in the EVL, announcing two event occurrences at that time. In the first simulation step, one of these events (since both have identical, lowest timestamps) is taken out of EVL arbitrarily, and the state is changed to $S = (1, 1, 1, 0, 0)$ since firing t_1 removes one token from p_1 and generates one token for both p_2 and p_3. Finally VT is adjusted. The new marking now enables transitions t_2 and t_3, both with firing time 2. Hence, new event tuples $\langle t_2@\mathrm{VT} + \tau(t_2)\rangle$ and $\langle t_3@\mathrm{VT} + \tau(t_3)\rangle$ are generated and scheduled, i.e. inserted in EVL in increasing order of timestamp. In step 2, again, the event with smallest timestamp is taken from EVL and processed in the same manner, etc.

Chapter 25

Centralized Schemes

25.1 Simulation Input Modelling

When developing discrete event *system models* (see Figure 24.1) for *real systems* which exhibit stochastic behavior, a close match between the uncertainty mechanisms underlying the true *input* and the *input model* is desirable [16] [51]. Realistic input models therefore need to provide for variable input as well, to be able to "mimic" probabilistic nature of system under consideration. The central issue in *input modelling* is the identification of such models (mostly based on empirical data collected from observations of the real system), and the assessment of the effect of errors observed in the simulation output due to making inappropriate choices. As an example, the variance of service time distribution for a single server queue determines simulation output performance parameters like the average queue length or waiting time. The input model for such a queue hence (*i*) must not ignore the chance of fluctuation possibly present in the real input process, and (*ii*) must allow for inferring "tolerances" for the simulation output parameters (by statistical analysis).

A discrete event simulation study in the input modelling process is concerned with collecting the right (and enough) data from the true input, choosing an one out of a range of possible (theoretical) input models, the exploitation of the true data to parametrize the input model (in case of a parametric input model), and to perform a sensitivity analysis of the model for validation purposes.

Collecting the right data from the (true) input process is preferably done by conducting a designed experiment, i.e. the analyst controls the data collection process according to statistical criteria. In many cases however, data sampled from the true input process is already provided, i.e. the analyst has no hand

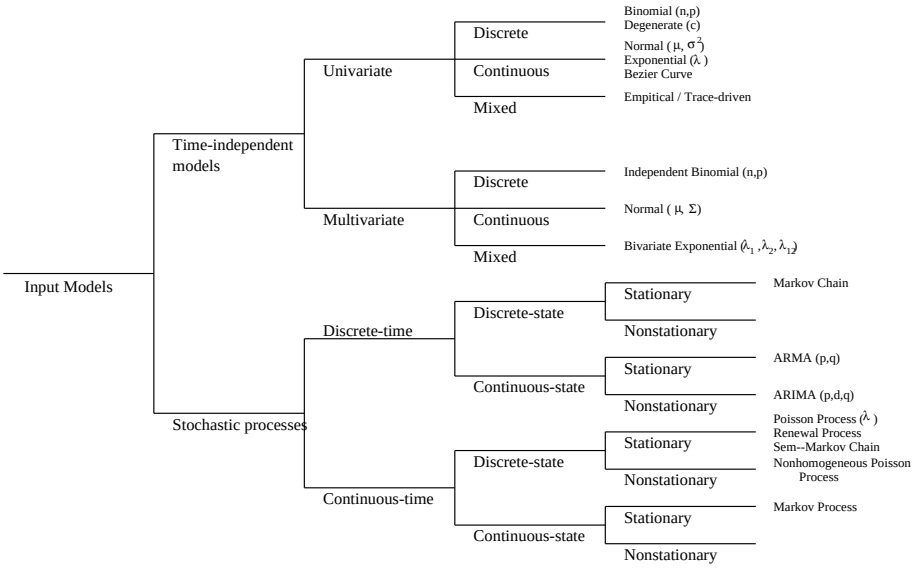


Figure 25.1: Leemis' Taxonomy of Input Models

on collecting it. An exploratory data analysis must assess the "suitability" (or quality) of the data in these cases.

A wide variety of (theoretical) input models have emerged in the simulation community. Figure 25.1 for example gives a taxonomy developed by Leemis [51]. Input models for systems not related to time (Monte Carlo simulations, see above) are given in the upper branch of this taxonomy. A distinction is made between univariate and multivariate models, and random variates drawn from theoretical distributions (line binomial, normal, exponential, etc.) are used to serve as the simulation input. In Leemis' taxonomy, *trace drive simulation* is considered a special case of a time-independent, univariate input model. The empirical (true) input data is used here as the simulation input. Stochastic process models define the lower branch of the taxonomy. Here the input process is understood as an unknown stochastic process (with continuous/discrete time and continuous/discrete state). Usually the empirical (true input) data is used to fit process models like Markov chains or autoregressive moving average (ARMA) time series models, which then drive the simulation.

A prerequisite for the "safe" use of an input model of the above kind is a careful statistical analysis of the empirical data set. Since most of the theoretical input models are assuming the iid property (independent, identically distributed) of random variables, *independence testing* of the observations is de-

manded. Hypothesis testing and correlation analysis are the statistical means to assess independence, besides visual tests like scatter or quantile-quantile plots. If no empirical evidence can be given for the iid property, methods for modelling stochastic dependencies must be consulted [77]. To be able to use a parametric model for simulation input, fitting a theoretical distribution from the empirical data often becomes an issue, e.g. fitting the arrivals of customers at a teller to a normal, Weibull or Gamma distribution, and estimating the respective parameters (mean, variance, etc.) accordingly. Basically, for *distribution fitting* the empirical cumulative distribution function is used to fit the cumulative distribution function of a theoretical distribution, and the adequacy is tested with a variety of the classical goodness-of-fit tests like the Kolmogorov-Smirnoff, the Chi-Square or the Cramer von Mises test.

Running the simulation now based on an input model of the above kind will yield certain responses as the simulation output. The analysis of the sensitivity of the simulation output on the configuration of the input model may give indication that further data collection of the (true) input is warranted, to develop an input model that induces less variance in the simulation output.

25.2 Simulation Output Analysis

When analysing the output of a simulation [48] [3], two kinds of system behaviour must be distinguished: if the dynamic behavior as reflected in the output parameters settles down a state independent of time ("in the long run") we talk about *steady state simulations*. Systems, on the other hand, that always operate under transient conditions, i.e. systems that never reach steady state it is necessary to study the system in a transient state. The respective simulations are called *transient* or *terminating simulations*.

25.2.1 Terminating (Transient) Simulations

Output analysis of terminating simulations is concerned with the statistical analysis of the output of *independent replications* of simulation runs, i.e. the application of standard statistical methods to the results *across* independent replications. Suppose a single simulation run yielding a sample of *size* n, i.e. n output data $X_1, X_2, \ldots, X_n$ has been collected. Then the objective is to estimate the unknown (true) mean $\mu = E(\overline{X}_n)$ with $\overline{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$. Clearly, $\overline{X}_n$ is an unbiased estimator for μ. For estimating the variance, the observations X_i generally cannot be assumed to be iid $\rightarrow$, making the estimation of

$Var(\overline{X})$ nontrivial from a single simulation run. Taking now k **independent replications** (runs), and assuming that run i produces: $X_{i,1}, X_{i,2}, \ldots, X_{i,n}$, then the sample means $Y_i = \frac{1}{n}\sum_{j=1}^{n} X_{i,j}$ are iid and an unbiased estimator for μ is obtained as $\overline{Y}_k = \frac{1}{k}\sum_{i=1}^{k} Y_i$. At the same time now, an unbiased estimator for $Var(\overline{X})$ is derived as

$$Var(\overline{X}) = \frac{1}{k-1}\sum_{i=1}^{k}(Y_i - \overline{Y}_k)^2.$$

If the sample size n and the number of replications k is "sufficiently large", we can also compute approximate $100(1-\alpha)$ confidence intervals for μ:

$$\overline{Y}_k - t_{k-1,1-\frac{\alpha}{2}}\sqrt{\frac{Var(\overline{X})}{k}} \leq \mu \leq \overline{Y}_k + t_{k-1,1-\frac{\alpha}{2}}\sqrt{\frac{Var(\overline{X})}{k}}$$

where $t_{d,\gamma}$ is the γ-quantile of the t-distribution with d degrees of freedom. Many procedures have been developed to determine the number of replications needed to estimate μ with a fixed error [3].

25.2.2 Steady State Simulations

A severe problem in the steady state analysis of simulation output is the removal of data related to the *initial transient* or *initial bias* the simulation traversed before settling in steady state. More formally, assuming the simulation output to be a discrete-time (stochastic) process $\{X_i : i \geq 1\}$ which according to the initial condition I converges to the steady state random variable x ($P(X_n \leq x \mid I) \rightarrow P(X_n \leq x)$) as $n \rightarrow \infty$, the we are interested in the estimator of the (true) steady state mean $\mu = lim_{n\rightarrow\infty} E(X_n \mid I)$. Ad hoc solutions to the problem would be to conduct "very" long simulation runs such that the impact of the *initial transient* becomes negligible, to start the simulation in with proper initialization (i.e. in a state close to steady state), to truncate $\{X_i : i \geq 1\}$ after observing a "pilot" run, or to truncate after an index i such that X_i is neither the minimum ($min_{(j\leq i)}(X_j)$) nor the maximum ($max_{(j\leq i)}(X_j)$) of the preceeding observations (min/max test [43]). None of these approaches can however be considered convincing. Again, a commonly used method for truncating the initial bias is based on considering k independent replications (simulation runs). Assume run i produces $X_{i,1}, X_{i,2}, \ldots, X_{i,n}$, then compute the averages $\overline{X}_j = \frac{1}{k}\sum_{i=1}^{k} X_{i,j}$ for $j = 1, 2, \ldots, n$. A visual truncation method now plots **moving averages** of the $\overline{X}_j$s for *time windows* of size ω (against j)

$$\overline{X}_j(\omega) = \begin{cases} \frac{1}{2\omega+1}\sum_{m=-\omega}^{\omega} \overline{X}_{j+m} & \omega+1 \leq j \leq n-\omega \\ \frac{1}{2j-1}\sum_{m=-j+1}^{j-1} \overline{X}_{j+m} & 1 \leq j \leq \omega \end{cases}$$

and decides to truncate at j if the plot is "reasonably" smooth. If not, "smoothness" is inspected for a different time window size ω.

25.2.3 Steady State Analysis

Once the initial bias has been removed from the observations from a steady state simulation, the estimation of the (true) unknown steady state mean μ can be done according to a variety of methods. We shall summarize the most important ones here, and generally assume a "bias free" discrete time simulation output $\{X_i : i \geq 1\}$ [1].

The Replication/Deletion Approach

The simplest method for estimating μ is to execute k independent replications of length n and truncate i observations (from each run) according to the procedure presented above. Then for sufficiently large k and n the sample means can be assumed to be iid, and μ can be estimated as

$$Y_i = \frac{1}{n-l} \sum_{j=l+1}^{n} X_{i,j}$$

The Method of Regeneration

For simulations that exhibit repeated regenerations in the sense that $\{X_i\}$ probabilistically "starts over" after reaching so called "regeneration points", the regeneration epochs can be used to obtain iid random variables for the estimation of μ.

Assume time indices $1 \leq T_1 < T_2 < \ldots$ s.t. $\{X_{T_i} + j, j > 0\}$ has the same distribution for each i and is independent to the protion prior to time T_i. Let ow $Y_i = \sum_{j=T_i}^{T_{i+1}} X_j$, $Z_i = T_{i+1} - T_i$, $(E(Z_i) < \infty) \ \forall \ i = 1, 2, ldots$, then

$$\mu = \frac{E(Y_1)}{E(Z_1)}.$$

To estimate the steady state mean therefore we can simulate for n cycles and collect $Y_1, Y_2, \ldots, Y_n$, $Z_1, Z_2, \ldots, Z_n$. Then the estimator

[1]

- $\{X_i\}$ is *strictly stationary* if joint distribution $X_{i+j_1}, X_{i+j_2}, \ldots, X_{i+j_k}$ is independent of i for all indices $j_1, j_2, \ldots, j_k$
- $\{X_i\}$ is *covariance stationary* if $E(X_i) = \mu$, $Var(X_i) < \infty$ for all i, and $C_j = Cov(X_i, X_{i+j})$ is independent of i

$$\widehat{\mu} = \frac{\overline{Y}_n}{\overline{Z}_n}$$

is strongly consistent (although typically biased for finite n). In practice, the method of regeneration is difficult to apply due to the non-existence of regeneration points in many simulations, and the identification of regeneration points in case of regenerative simulation behaviour.

The Batch Means Method

A simple, yet effective method for steady state parameter estimation is the method of batch means. Assuming $\{X_i\}$ is covariance stationary, the method basically divides n outputs $X_1, X_2, \ldots, X_n$ of a single but *long* simulation run into k batches of size b: $X_{(i-1)b+1}, X_{(i-1)b+2}, \ldots X_{ib} \quad i = 1, 2, \ldots, k \quad (n = kb)$. Then it computes the batch means as $Y_i(b) = \frac{1}{b}\sum_{j=1}^{b} X_{(i-1)b+j} \quad i = 1, 2, \ldots, k$, and since $\{X_i\}$ is covariance stationary:

$$C_l(b) = Cov(Y_i(b), Y_{i+l}(b)) = \frac{1}{b}\sum_{j=1-b}^{b-1}(1 - \frac{\mid j \mid}{b})\, C_{lb+j}$$

is independent of i. If $C_l(b) \to 0$ as b increases one can find a batch size b such that $Y_i(b), \quad i = 1, 2, \ldots, k$ are approximately iid. The estimate is thus obtained by computing the grand batch mean (and variance) as:

$$\overline{Y}_k = \overline{X}_n = \frac{1}{n}\sum_{i=1}^{k} Y_i(b) \qquad Var(\overline{X}) = \frac{1}{k-1}\sum_{i=1}^{k}(Y_i(b) - \overline{Y}_k)^2$$

The problem with the batch means method is the choice of b (or equivalently k): b must be big enough s.t. $\overline{Y}_k$ are approximately uncorrelated. Empirical evidence has been given that it may never pay to have $b > 20 \sim 30$ (due to autocorrelation, splitting into a larger number of smaller batches degrades quality (variability) of each individual batch).

The Standardized Time Series Method

This method assumes $\{X_i\}$ to be strictly stationary and again considers the observation sequence $X_1, X_2, \ldots, X_n$ to be divided into k contiguos batches of size b. Letting $Y_1(b), Y_2(b), \ldots, Y_k(b)$ again be the batch means then it can be shown that the random variables

$$A_i = \sum_{j=1}^{b}(\frac{n+1}{2} - j)X_{(i-1)b+j} \qquad i = 1, 2, \ldots, k$$

are approximately iid following a normal distribution. A $100(1-\alpha)$ percent confidence interval for the true parameter can thus be given as

$$\overline{Y}_k - t_{k,1-\frac{\alpha}{2}}\sqrt{\frac{\hat{V}_T}{n}} \leq \mu \leq \overline{Y}_k + t_{k,1-\frac{\alpha}{2}}\sqrt{\frac{\hat{V}_T}{n}} \qquad \text{where} \qquad \hat{V}_T = \frac{12}{b_3 - b} k \sum_{i=1}^{k} A_i^2$$

Easy to implementation and asymptotic advantages over batch means method have made the time series method a frequent choice in steady state output analysis.

The Autoregressive Method

The autoregressive method aims at an interpretation of of the simulation outputs as an autoregressive process of order p. Assuming $\{X_i\}$ to covariance stationary with mean μ and $\sum_{j=-\infty}^{\infty} \mid C_j \mid \; < \; \infty$, then $\{X_i\}$ is presented as:

$$\sum_{j=0}^{p} b_j (X_{i-j} - \mu) = \epsilon_i$$

where $b_0 = 1$, ϵ_i a sequence of error terms with $E(\epsilon_i) = 0$ and $Var(\epsilon_i) = \sigma_\epsilon^2$. For large n, a $100(1-\alpha)$ percent confidence interval for μ is given as

$$\overline{X}_n - t_{d,1-\frac{\alpha}{2}}\sqrt{\frac{\hat{V}_A}{n}} \leq \mu \leq \overline{X}_n + t_{d,1-\frac{\alpha}{2}}\sqrt{\frac{\hat{V}_A}{n}}$$

where

$$\hat{V}_A = \frac{\hat{\sigma_\epsilon}^2}{n(\sum_{j=0}^{p} \hat{b}_j)^2} \qquad d = \frac{n(\sum_{j=0}^{p} \hat{b}_j)}{2\sum_{j=0}^{p}(p-2j)\hat{b}_j}$$

Certainly is the validity of the autoregressive model assumption the most critical issue in practice.

The Spectral Estimation Method

Under the assumption of $\{X_i\}$ being covariance stationary, this method uses the spectral property of the variance of $\overline{X}_n$

$$Var(\overline{X}_n) = \frac{1}{n}(C_0 + 2\sum_{j=1}^{n-1}(1 - \frac{j}{n})C_j) :$$

If $\sum_{j=-\infty}^{\infty} \mid C_j \mid \; < \; \infty$ then $nVar(\overline{X}_n) \rightarrow 2\pi g(0)$ as $n \rightarrow \infty$. $g(\lambda)$ is the *spectrum* of the process at frequency λ defined as

$$g(\lambda) = \frac{1}{2\pi} \sum_{j=-\infty}^{\infty} C_j \; e^{-i\lambda j} \qquad \mid \lambda \mid \leq \pi, \quad i = \sqrt{-1}$$

For large n, the estimation of $Var(\overline{X}_n)$ can viewed as estimating the spectrum $g(0)$. The typical form of variance estimators according to the spectral estimation method is

$$\hat{V}_S = \frac{1}{n}(\hat{C}_0 + 2\sum_{j=1}^{p-1} w_j \hat{C}_j).$$

Computational Efficiency

The computational efficiency of the overlapping and non-overlapping batch means method, the spectral method and the regenerative method are investigated in [22]. A thorough computational efficiency analysis for various variations of the batch means methods is studied in [4] and [36].

25.3 Accelerating Simulations: Efficiency Improvement

The problem of assessing the statistical efficiency of estimators for true unknown system parameters from simulations is generally referred to as *efficiency improvement* [50]. Considering stochastic simulation as a means to generate realizations of a random variable X to serve as an estimator for the true unknown μ, then efficiency improvement is concerned with finding *computationally efficient* estimators. Assuming X and Y being such estimators, then X is said to be computationally *more efficient* than Y if the *efficiency* of X, Eff(X), is higher than that of Y: Eff(X) > Eff(Y). Many definitions of Eff(X) have appeared in the literature (all of them of course in some way considering the computational cost (e.g. CPU time)) We will present a very intuitive one [50].

Consider the random variable X for which realizations are collected during a stochastic simulation, taken as estimator for μ. The *bias* of this estimator is defined to be $\beta = E[X] - \mu$, the variance $\sigma^2 = Var(X) = E[(X - E[X])^2]$ and the mean square error as $MSE[X] = E[(X-\mu)^2] = \beta^2 + \sigma^2$. Let now the CPU *cost* (for computing X) be denoted by $C(X)$, then the efficiency of X can be expressed as:

$$\text{Eff}(X) = \frac{1}{MSE[X]\ \ C(X)}.$$

From this definition it is easy seen that, assuming the two estimators X and Y both being unbiased ($\beta = 0$) and having about the same computational cost ($C(X) \sim C(Y)$), that improving the efficiency means reducing the mean

square error, i.e. *reducing the variance* σ^2. This is also the reason why efficiency improvement is often addressed with *variance reduction techniques* (VRTs).

A more recent definition of efficiency considers *asymptotic efficiency*, as CPU power goes to infinity. Let $\overline{X}(t)$ be an estimator obtained with CPU budget t ($C(\overline{X}(t)) = t$), then there exist a constant γ and a random variable Z such that $t^{2\gamma}(\overline{X}(t) - \mu) \Rightarrow Z$ ($\Rightarrow$ here stands for *"convergence in distribution"*) and $t^{2\gamma}\ MSE[\overline{X}(t)] = \nu + o(1)$ where $o(1) \to 0$ as $t \to \infty$. Now the *asymptotically most efficient estimator* is the one with largest value γ, in case of a tie, the one with the smallest value of the constant ν).

The remainder of this Chapter will be devoted to a selection of relevant variance reduction techniques.

25.3.1 The Common Random Numbers Method

A VRT that is particularly appropriate when comparing the performance parameters of two (or more) physical systems μ_1, μ_2 by estimating the difference between the two respective performance estimators X_1, X_2 is called the common random numbers (CRN) method. For estimating the true difference of the unknowns $\mu_1 - \mu_2$ we take a random variable $Z = X_1 - X_2$ and estimate the difference by $E[Z] = \mu_1 - \mu_2$. Since then variance of Z is $Var[Z] = Var[X_1] + Var[X_2] - 2Cov[X_1, X_2]$ ($Cov[X_1, X_2]$ disappears if X_1, X_2 are generated independently), the method aims to induce a positive covariance $Cov[X_1, X_2]$ to reduce $Var[Z]$. The approach for achieving a variance reduction is to use same underlying uniform random numbers (RNs) in the (stochastic) simulations that generate X_1 and X_2, with the rationale of exposing the two systems to the same random noise ("experimental conditions"). The observed differences can thus be assumed to be due to differences between the systems, and not due to "lucky" random numbers in one simulation that are not present in the other.

A critical problem with the CRN method is that using the same set of (*common*) random numbers cannot guarantee $Cov[X_1, X_2] > 0$. A sufficient (but not necessary) condition for $Cov[X_1, X_2] > 0$ is that X_1, X_2 are both monotonically increasing/decreasing for **any** underlying uniform RN. If there is monotonicity only for some of the RNs, **independent** RNs must be used for cases where X_1, X_2 appear nonmonotone. Clearly, monotonicity in this context is not always easy to check.

Evidence has been delivered that the CRN method works well (even in absence of monotonicity) if a parametrized (θ) system reacts similar to close values

of θ. Let X_i be the response obtained for a system at parameter θ_i, then the correlation between X_1, X_2 approaches zero as $\mid \theta_1 - \theta_2 \mid \rightarrow 0$. This means that the CRN method will reduce $Cov[X_1, X_2]$ if the parameters θ_1, θ_2 are "close enough" to each other.

Besides difference estimating, the CRN method can also be used to for reducing the variance of general estimators $g(X_1, X_2, \ldots, X_n)$, like e.g. the fraction of system parameters $g(\mu_1, \mu_2) = \frac{\mu_1}{\mu_2}$.

25.3.2 The Antithetic Variates Method

The antithetic variates (AV) method resembles the idea of the CRN method for cases where one wants to estimate *single* unknown μ using a pair of estimators X_1, X_2. If one uses the average as an (unbiased) estimator $X = \frac{X_1 + X_2}{2}$ for μ, the variance of X is

$$Var[X] = \frac{Var[X_1] + Var[X_2]}{4} + \frac{Cov[X_1, X_2]}{2}$$

Assuming now $Var[X_1] = Var[X_2]$ and X_1, X_2 being independent, then $Var[X] = \frac{Var[X_1]}{2}$. Assuming again $Var[X_1] = Var[X_2]$, but $Cov[X_1, X_2] < 0$ (X_1, X_2 not being independent), then X has smaller variance. The AV method now tries to induce a negative covariance $Cov[X_1, X_2]$ to reduce the variance $Var[X]$.

Tehcnically, a sequence of iid uniform RNs $\omega_1 \equiv \{U_k, k \geq 1\}$ is used to drive simulation for obtaining X_1 ($X_1 = h(\omega_1)$), while the respective *"antithetic"* sequence of RNs $1 - \omega_1 \equiv \{1 - U_k, k \geq 1\}$ is used to drive the simulation for obtaining X_2 ($X_2 = h(1 - \omega_1)$). The rationale here is that "disastrous" events that occured in the first simulation should be compensated by "antithetic" lucky ones in the second, thus reducing the variance on average.

Like CRN, AV does not guarantee a negative covariance ($Cov[X_1, X_2] < 0$), nor a variance reduction in general. A sufficient (but not necessary) condition for $Cov[X_1, X_2] < 0$ is that h, is monotonically increasing/decreasing for **each** underlying uniform RN. Again, if h is monotone only for some RNs ω_i, **independent** RNs must be used $\overline{\omega_i}$ to compute $X_k = h(\overline{\omega_i})$, $X_l = h(1 - \overline{\omega_i})$.

The worst case scenario for the AV method is where X_1, X_2 are perfectly correlated; here variance (instead of being reduced) doubles. The best case is where the response is linear function of all underlying uniforms; here the variance reduced to 0.

25.3.3 Latin Hypercube Sampling

Latin hypercube sampling is not a variance reduction technique as such, but a generalization of the variate sampling technique used in the AV method. In the negative correlation induction framework of [Avramidis/Wilson 94] n dependent replications performed with i-th replication using uniform random variates $\omega_i \equiv \{U_{i,k}, k \geq 1\}$. Considering the k-th random number of each replication as a vector $U_k = (U_{1,k}, U_{2,k}, \ldots, U_{n,k},$ then U_k follows a multivariate distribution where each univariate marginal is uniformly distributed $U(0,1)$, and each bivariate marginal (Y_1, Y_2) is negatively quadrat dependent $(P[Y_1 \leq y_1, Y_2 \leq y_2] \leq P[Y_1 \leq y_1] \cdot P[Y_2 \leq y_2] \quad \forall\, y_1, y_2)$. To induce negative correlation (and thus reduce variance), random numbers can be sampled using the following procedure (latin hypercube sampling (LHS) method):

(i) select finite subset Ψ of RNs

(ii) for each $k \in \Psi$ generate a random permutation of integers $\{1, 2, \ldots, n\}$ ($\pi_{i,k}$ ... i-th element in permutation k)

(iii) for each (i,k) generate $U_{i,k}$ uniformly over the interval $((\pi_{i,k}-1)/k, \pi_{i,k}/k)$

(iv) for each $k \notin \Psi$ generate $U_{i,k}$'s independently from $U(0,1)$ distribution

Then each $\omega_i \equiv \{U_{i,k}, k \geq 1\}$ is a sequence of iid uniformly distributed variates. For each $k \in \Psi$ the interval $(0,1)$ is partitioned into n equal pieces across the n replications, i.e. $U_{i,k}$ form a stratified sample over $(0,1)$. Variance reduction can be guaranteed under certain conditions using random numbers coming from the LHS method.

25.3.4 Control Variables

The control variables (CV) variance reduction method basically exploits information on random events being more/less "favorable" in influencing sample performance, and makes appropriate corrections. The method, for the estimator X defines a vector of *control variables* $Y = (Y_1, \ldots, Y_q)$ with known expectation $E(Y) = \nu = (\nu_1, \ldots, \nu_q)$. A *controlled estimator*

$$X_c = X - \beta(Y - \nu) = X - \sum_{k=1}^{q} \beta_k (Y_k - \nu_k)$$

is defined, using a vector of constants $\beta = (\beta_1, \ldots, \beta_q)$ to control (reduce) variance in the following way. Let $\Sigma_Y = Cov[Y]$ be the covariance matrix

of the vector of control variables (with $Cov[Y_i, Y_j]$ as its element (i, j)), and $\sigma_{XY} = (Cov(X, Y_1), \ldots, Cov(X, Y_q))$, then the expectation of the controlled estimator can be used as the estimator ($E[X_c] = E[X] = \mu$), where the variance of of the controlled estimator X_c is $Var[X_c] = Var[X] + \beta\Sigma_Y\beta - 2\beta\sigma_{XY}$. The variance hence is minimized with $\beta = \beta^* = \Sigma_Y^{-1}\sigma_{XY}$, in which case

$$Var[X_c] = (1 - R^2_{XY})Var[X]$$

where $R^2_{XY} = \frac{\sigma_{XY}\Sigma_Y^{-1}\sigma_{XY}}{var[X]}$ is the coefficient of determination. Variance is by either positive or negative correlation, and R^2_{XY} indicates the fraction of variance that is reduced.

In the totally correlated case (multiple correlation is ± 1) variance reduced to 0, in the uncorrelated case (multiple correlation is 0) the variance remains unchanged. A problem with the CV method is β^* being unknown, and $\sum_Y$ and $\sum_{XY}$ must be estimated for the computation of β^*. This is usually done by using the samples that were obtained in the n independent replications to compute $\hat{\sum}_Y$ and $\hat{\sum}_{XY}$, and from that use $\hat{\beta} = \hat{\sum}_Y^{-1}\hat{\sigma}_{XY}$ instead of β^*. Now let $X_{ce,i}$, $i = 1, \ldots, n$ denote the n replicates of the controlled estimator $X_{ce,i} = X_i - \hat{\beta}(Y_i - \nu)$ where (X_i, Y_i) is the i-th replicate of (X, Y), and $\overline{X}_{ce}$ and s^2_{ce} be the sample average and the sample variance of the $X_{ce,i}$, the the CV estimator for μ is $\overline{X}_{ce}$.

25.3.5 Importance Sampling

The rationale of the importance sampling (IS) method is to concentrate the sampling effort in the "most important parts of the sample space. For this reason, IS is particularly effective for dealing with *rare events* in simulation (concentrating in the "areas" where rare events are most likely to occur).

25.3.6 Stratification

To reduce variance, stratification attempts to partition the sample space into disjoint strata such that that variance within individual strata is smaller than the "non-stratified" variance. To compute a *stratified estimator*, the observations from N simulation runs are "stratified" according to S strata, such that N_s runs fall into strata s ($N = \sum_{s=1}^{S} N_s$). If $X_{s,i}$ denotes the i-th observation from strata s, and p_s the probability of falling into strata s under the original distribution, then the stratified estimator is

$$X_s = \sum_{s=1}^{S} p_s(\frac{1}{N_s}\sum_{i=1}^{N_s} X_{s,i}).$$

Depending on whether stratification is done before or after the execution of the N simulation runs, two cases are distinguished: In the *post-stratification* approach the observations are "stratified" (N_s is chosen) such that $E[N_s] = p_s N$. *Pre-stratification* decides to which stratum a simulation run will belong to before the simulation is executed. N_ss are chosen so as to minimize the variance X_s, i.e.

$$N_s = \frac{N p_s \sigma_s}{\sum_{j=1}^{S} p_j \sigma_j}$$

where $\sigma_s^2 = Var[X_{s,i}]$ is the variance within stratum$s$. The unknown σ_s's are estimated from observations in pilot simulations.

Chapter 26

Accelerating with Multiple Processors

The statistical acceleration methods for simulation model executions as presented in the previous chapter are mostly aimed to avoid the generation of "statistically unnecessary", thus effectively reducing computational efforts. Naturally, however, faster simulations can be obtained by using more computational resources, particularly multiple processors operating in parallel. It seems obvious at least for simulation models reflecting real life systems constituted by components operating in parallel, that this inherent model parallelism could be exploited to make the use of a parallel computer potentially effective. Moreover, for the execution of independent replications of the same simulation model with different parametrizations the parallelization appears to be trivial. In this chapter we shall systematically describe ways of accelerating simulations using multiprocessor systems with focus on the synchronization of *logical simulation processes* executing in parallel on different processing nodes in a parallel or distributed environment.

In the example of Figure 24.11, there are situations where several transitions have identical smallest timestamps, e.g. in step 5 where all scheduled transitions have identical end firing time instants. This is not an exceptional situation but appears whenever (*i*) two or more events *(potentially) can* occur at the same time but are mutually exclusive in their occurrence, or (*ii*) *(actually) do* occur simultaneously in the real system. The latter distinction is very important with respect to the construction of parallel or distributed simulation engines: t_2 and t_3 are scheduled to fire at time 5 (their enabling lasted for the whole period

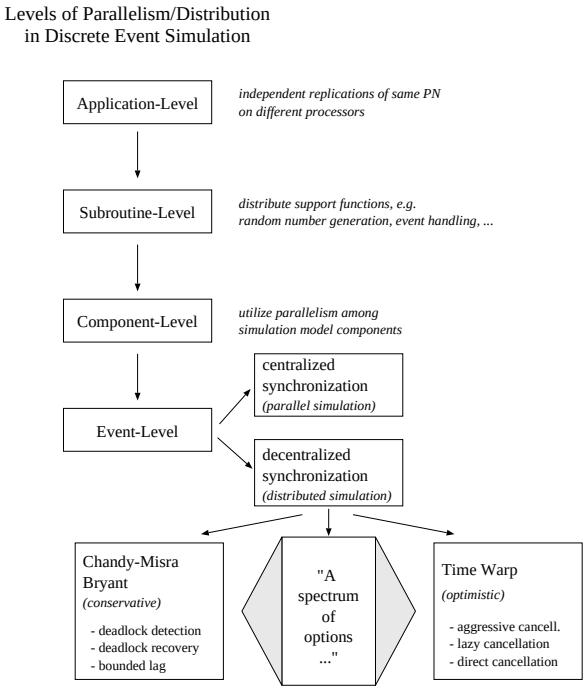


Figure 26.1: Options to Parallelize/Distribute Discrete Event Simulations.

of their firing time $\tau(t_2) = \tau(t_3) = 2$), where the firing of one of them will not interfere the firing of the other one. t_2 and t_3 are said to be *concurrent events* since their occurrences are not interrelated. Obviously t_2 and t_3 could be simulated in parallel, say t_2 by some processor P1 and t_3 by another processor P2. As an improvement of the sequential simulation on the other hand, they could both be removed from EVL in a single simulation step. The situation is somewhat different with t_4 and t_5, since the occurrence of one of them will disable the other one – t_4 and t_5 are said to be *conflicting events*. The effect of simulating one of them would (besides changing the state) also be to remove the other one from EVL. t_4 and t_5 are *mutually exclusive* and preclude parallel simulation.

Before following the idea of simulating a single simulation model (like the example PN) in parallel, we will first take a more systematic look at the possibilities to accelerate the execution of simulations using P processors.

26.1 Levels of Parallelism/Distribution

What are the options for the acceleration of discrete event simulation executions? Figure 26.1 shows that *application-level* distribution is feasible if large parameter spaces have to be searched (execute the PN with several initial markings). *Subroutine-level* distribution is feasible if iteration dependencies (output values of replication i = input parameters for replication $i+1$) hinders application-level distribution, but is limited in the degree of exploitable parallelism by the number of simulation support subroutines. *Component-level* distribution exploits the natural parallelism of simulation model components. Most challenging from a scientific viewpoint is the parallelization at the *event-level*, where basically two strategies of event list (EVL) management are feasible. In a *centralized* EVL management approach, a master process maintains the EVL and virtual time (VT) progression, while workers execute concurrent events. This approach is particularly appropriate for UMA and COMA architectures with the EVL organized as a shared data structure, and often makes use of a centralized control (synchronization) unit. A *decentralized* EVL management strategy allows the concurrent simulation of events with different timestamps, thus necessitating a local synchronization protocol. We shall distinguish *parallel simulation* that follows a SIMD operated synchronization, from *distributed simulation* (or *Logical Process Simulation*) which operates completely asynchronous.

26.2 Logical Process Simulation

Common to all simulation strategies with distribution at the event level is their aim to divide a global simulation task into a set of communicating *logical processes* (LPs), trying to exploit the parallelism inherent among the respective model components with the concurrent execution of these processes. We can thus view a *logical process simulation* (LP simulation) as the cooperation of an arrangement of interacting LPs, each of them simulating a subspace of the space-time which we will call an event structure *region*. In our example a region would be a spatial part of the PN topology. Generally a region is represented by the set of all events in a sub-epoch of the simulation time, or the set of all events in a certain subspace of the simulation space.

The basic architecture of an LP simulation can be viewed as in Figure 26.2:

- A *set of LPs* is devised to execute event occurrences synchronously or asynchronously in parallel.

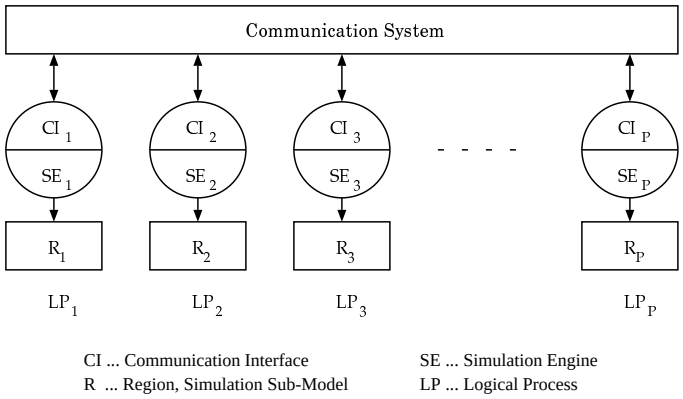


Figure 26.2: Architecture of a Logical Process Simulation

- A *communication system* (CS) provides the possibility to LPs to exchange local data, but also to synchronize local activities.
- Every LP_i has assigned a *region* R_i as part of the simulation model, upon which a simulation engine SE_i operating in event driven mode (Figure 24.10) executes *local* (and generates *remote*) event occurrences, thus progressing a *local clock* (local virtual time, LVT).
- Each LP_i (SE_i) has access only to a statically partitioned *subset of the state variables* $S_i \subset S$, disjoint to state variables assigned to other LPs.
- Two kinds of events are processed in each LP_i: *internal events* which have causal impact only to $S_i \subset S$, and *external events* which also affect $S_j \subset S$ ($i \neq j$) the local states of other LPs.
- A *communication interface* CI_i attached to the SE takes care for the propagation of effects causal to events to be simulated by remote LPs, and the proper inclusion of causal effects to the local simulation as produced by remote LPs. The main mechanism for this is the sending, receiving and processing of event messages piggybacked with copies of the senders LVT at the sending instant.

Basically two classes of CIs have been studied for LP simulation, either taking a *conservative* or an *optimistic* position with respect to the advancement of event executions. Both are based on the sending of messages carrying causality information that has been created by one LP and affects one or more other LPs. On the other hand, the CI is also responsible for preventing global event

causality violations. In the first case, the conservative protocol, the CI triggers the SE in a way which prevents from causality errors ever occuring (by blocking the SE if there is the chance to process an 'unsafe' event, i.e. one for which causal dependencies are still pending). In the optimistic protocol, the CI triggers the SE to redo the simulation of an event should it detect that premature processing of local events is inconsistent with causality conditions produced by other LPs. In both cases, messages are invoked and collected by the CIs of LPs, the propagation of which consumes real time dependent on the technology the communication system is based on.

For the representation and advancement of simulated time (VT) in an LP simulation we can devise two possibilities[66]: a *synchronous LP simulation* implements VT as a global clock, which is either represented explicitly as a centralized data structure, or implicitly implemented by a time-stepped execution procedure – the key characteristic being that each LP (at any point in real time) faces the same VT. This restriction is relaxed in an *asynchronous LP simulation*, where every LP maintains a *local* VT (LVT) with generally different clock values at a given point in real time.

26.2.1 Synchronous LP Simulation

In a *time-stepped* LP simulation [66], all the LPs' local clocks are kept at the same value at every point in real time, i.e. every local clock evolves on a sequence of discrete values $(0, \Delta, 2\Delta, 3\Delta, \ldots)$. In other words, simulation proceeds according to a global clock since all local clocks appear to be just a copy of the global clock value. Every LP must process all events in the time interval $[i\Delta, (i+1)\Delta)$ (time step i) before any of the LPs are allowed to begin processing events with occurrence time $(i+1)\Delta$ and after. This strategy considerably simplifies the implementation of correct simulations by avoiding problems of deadlock and possibly overwhelming message traffic and/or memory requirements as will be seen with synchronization protocols for asynchronous simulation. Moreover, it can efficiently use the barrier synchronization mechanisms available in almost every parallel processing environment. The imbalance of work across the LPs in certain time steps on the other hand naturally leads to idle times and thus represents a source of inefficiency.

Both centralized and decentralized approaches of implementing global clocks have been followed. In [76], a centralized implementation with one dedicated processor controlling the global clock is proposed. To overcome stepping the time at instances where no events are occuring, algorithms to determine for

time-stepped LP simulation [Peacock 79]:

- local clock evolves on a sequence of discrete values $(0, \Delta, 2\Delta, 3\Delta, \ldots)$
- Every LP must process all events in the time interval $[i\Delta, (i+1)\Delta)$ (time step i) before any of the LPs are allowed to begin processing events with occurrence time $(i+1)\Delta$ and after.
- Reliant to efficient barrier synchronization.
- Imbalance of simulation work = source of inefficiency

global clock [Venkatesh 86]:

- One dedicated processor controls the global clock.
- A structured (hierarchical) LP organization can be used to determine the minimum next event time.
- Global clock can be advanced by $\Delta(S)$, i.e. an amount depending on state S
- Reliant to parallel min-reduction operation.

combinations synchronous LP simulation + event driven global clock progression

- "Bounded Lag" [Lubachevsky 88]
- "Moving Time Windows" [Sokol/Briscoe/Wieland 88]

Figure 26.3: Synchronous Logical Process Simulation: Summary

every LP at what point in time the next interaction with another LP shall occur have been developed. Once the minimum timestamp of possible next external events is determined, the global clock can be advanced by $\Delta(S)$, i.e. an amount which depends on the particular state S. For a distributed implementation of a global clock [66], a structured (hierarchical) LP organization can be used [21] to determine the minimum next event time. A parallel min-reduction operation can bring this timestamp to the root of a process tree [7], which can then be propagated down the tree. Another possibility is to apply a distributed snapshot algorithm in order to avoid the bottleneck of a centralized global clock coordinator.

Combinations of synchronous LP simulation with event-driven global clock progression have also been studied. Although the global clock is advanced to the minimum next event time as in the event driven scheme, LPs are only allowed to simulate within a Δ-tick of time, called a bounded lag by Lubachevsky [54] or a Moving Time Window by [73]. Synchronous LP simulation is summarized

in Figure 26.3.

26.2.2 Asynchronous LP Simulation

Independent Events Relies on the presence of events occuring at different simulated time instants that do not affect one another.

Critical Problem: chance of *causality errors*

Correctness assured if the (total) event ordering as produced by a sequential DES is consistent with the (partial) event ordering as generated by the parallel execution.

Causality Errors can never occur if and only if every LP adheres to processing events in nondecreasing timestamp order only (*local causality constraint* [Fujimoto 90]).

lcc: Although sufficient, it is not always necessary to obey *lcc*, because two events occuring within one and the same LP may be concurrent (independent of each other) and could thus be processed in any order.

Mechanisms: Two main categories of mechanisms adhere to the *lcc* in different ways:

- **conservative** methods strictly avoid *lcc* violations, even if there is some nonzero probability that an event ordering mismatch will *not* occur.
- **optimistic** methods hazardously use the chance of processing events even if there *is* nonzero probability for an event ordering mismatch.

Figure 26.4: Asynchronous Logical Process Simulation: Summary

Asynchronous LP simulation relies on the presence of events occuring at different simulated times that do not affect one another. Concurrent processing of those events thus effectively accelerates sequential simulation execution time.

The critical problem, however, which asynchronous LP simulation poses is the chance of *causality errors*. Indeed, an asynchronous LP simulation insures correctness if the (total) event ordering as produced by a sequential DES is consistent with the (partial) event ordering as generated by the distributed execution. Jefferson [47] recognized this problem to be the inverse of Lamport's logical clock problem [49], i.e. providing clock values for events occuring in a distributed system such that all events appear ordered in logical time.

It is intuitively convincing and has been shown in [58] that no causality error can ever occur in an asynchronous LP simulation if and only if every LP adheres to processing events in nondecreasing timestamp order only (*local causality*

constraint (*lcc*) as formulated in [33]). Although sufficient, it is not always necessary to obey the *lcc*, because two events occuring within one and the same LP may be concurrent (independent of each other) and could thus be processed in any order. The two main categories of mechanisms for asynchronous LP simulation already mentioned adhere to the *lcc* in different ways: conservative methods strictly avoid *lcc* violations, even if there is some nonzero probability that an event ordering mismatch will *not* occur; whereas optimistic methods hazardously use the chance of processing events even if there *is* nonzero probability for an event ordering mismatch. Asynchronous LP simulation is summarized in Figure 26.4.

26.2.3 Synchronous vs. Asynchronous LP Simulation: A Performance Comparison

In a comparison of synchronous and asynchronous LP simulation schemes it has been shown [27], that the potential performance improvement of an asynchronous LP simulation strategy over the time-stepped variant is at most $O(log\, P)$, P being the number of LPs executing concurrently on independent processors. The analysis assumes each time step to take an exponentially distributed amount of execution time $T_{step,i} \sim exp(\lambda)$ in every LP$_i$ ($E[T_{step,i}] = \frac{1}{\lambda}$). As a consequence, the expected simulation time $E[T^{sync}]$ for a k time step synchronous simulation compares to the expected simulation time $E[T^{async}]$ of an asynchronous simulation as

$$E[T^{sync}] = k\ E[\max_{i=1..P}\ (T_{step,i})] = k\ \frac{1}{\lambda}\sum_{i=1}^{P}\frac{1}{i}\ \leq \frac{k}{\lambda}log(P)$$

$$E[T^{async}] = E[\max_{i=1..P}(k\ T_{step,i})] > \frac{k}{\lambda}$$

Therefore

$$\lim_{k\to\infty, P\to\infty}\frac{E[T^{sync}]}{E[T^{async}]} \approx log(P)$$

i.e., with increasing simulation size k, an asynchronous simulation could complete (at most) $log(P)$ times as fast as the synchronous simulation, and the maximum attainable speedup of any time stepped simulation is $\frac{P}{log(P)}$. These results, however, are a direct consequence of the exponential step execution time assumption, i.e. comparing the expectation of the k-fold sum over the max of exponential random variates (synchronous) with the expectation of the max over P k-stage Erlang random variates.

For a step execution time uniformly distributed over $[l, u]$ we have

$$\lim_{k\to\infty, P\to\infty} \frac{E[T^{sync}]}{E[T^{async}]} \approx 2$$

or intuitively: $T^{sync} \leq k\ u$,

$$E[T^{async}] \geq k\frac{(l+u)}{2} \quad \Rightarrow \quad \frac{E[T^{sync}]}{E[T^{async}]} = \frac{2}{(k\ u)(k\ (l+u))} \leq 2,$$

i.e. constant. Therefore for a local event processing time distribution with finite support the improvement of an asynchronous strategy reduces to an amount independent of P.

Certainly the model assumptions are far from what would be observed in real implementations on certain platforms, but the results might help to rank the two approaches at least from a statistical viewpoint.

26.2.4 An LP Simulation Framework for TPNs

For the concurrent execution of TPNs using logical processes we can use the following framework: A *Logical Process* is a tuple $LP_k = (TTPNR_k, I_k, SE)$ where

(i) **region**
$TTPNR_k = (P_k, T_k, F_k \subseteq (P_k \times T_k) \cup (T_k \times P_k), W, \Pi, \tau, M_0)$ is a (spatial) *region* of some $TTPN$ s.t. $P = \bigcup_{k=1}^{N_{LP}} P_k$, $T = \bigcup_{k=1}^{N_{LP}} T_k$,
$F = \bigcup_{k=1}^{N_{LP}} F_k \bigcup_{i\in \text{neighborhood}(TTPNR_k)}((P_k \times T_i) \cup (T_k \times P_i))$, which are again TTPNs.

(ii) **communication interface**
$I_k = (CH, m)$, the *communication interface* of LP_k is a set of directed communication channels $CH = \bigcup_{k,i} ch_{k,i}$ where $ch_{k,i} = (LP_k, LP_i)$ connects LP_k to LP_i, corresponding to an arc $f \in F$ and carrying messages $m = \langle w(f), D, TT\rangle$.
$w(f)$ ... number of tokens transferred,
D ... identifier of the destination (place or transition) and
TT (token time) ... timestamp of the local virtual time of the sending LP_k.
The CI implements a DDES synchronization protocol

(iii) **simulation engine**
SE is an event driven *simulation engine* implementing the simulation strategy.

The set of all LPs $LP = \bigcup_k LP_k$ together with directed communication channels $CH = \bigcup_{k,i} ch_{k,i} = \bigcup_{k,i}(LP_k, LP_i)$ constitute the *Graph of Logical Processes* $GLP = (LP, CH)$.

Correspondingly, in a *logical process simulation* of TPNs a set of LPs devised to execute event occurrences of a spatially partitioned TPN graph asynchronously in parallel on different nodes of a multiprocessor system. The software architecture of a logical process LP_i comprises (i) an event list (EVL) based *simulation engine* SE_i, where transitions $t_i \in T$ are scheduled for firing at their occurrence time $ot(t_i)$, (ii) a *communication interface* CI_i connecting LP_i to other LPs and providing the possibility to synchronize local events with remote events, and (iii) an assigned *region* R_i as part of the TPN simulation model. SE_i operates on R_i in an event driven mode, i.e. it executes *local* and generates *local* and *remote*) future events (transition firings), while progressing a *local clock* (local virtual time, LVT). Each LP_i (SE_i) has access only to a statically partitioned *subset of the state variables* $S_i \subset S$, constituted by the token counts in places $p \in P$ contained in R_i (S_i is disjoint to state variables assigned to other LPs). Two kinds of events processed in each LP_i are distinguished: *internal events* have causal impact only to $S_i \subset S$, whereas *external events* also affect $S_j \subset S$ $(i \neq j)$, the local states of other LPs. The communication interface CI_i takes care for the propagation of effects causal to events to be simulated by remote LPs, and the proper inclusion of external causal effects. This is done by "encoding" the token(s) released by a transition upon firing for a place (or places) in a remote LP (or LPs) into a token message m (piggybacked with a copy of the senders LVT at the sending instant), the delivery of m to the destination LP(s), and the "decoding" of m into the enabling of transitions and their scheduling in the receivers EVL according to the message time stamp value $t(m)$, relative to the receivers LVT.

Chapter 27

Synchronized (Parallel) Schemes

27.1 Parallel Simulation of TPNs

Parallel discrete event simulation (PDES) of TPNs aims to exploit the specific features offered by SIMD operated environments, especially the hardware support for fast communication in a static, regular interconnection network of processors controlled by a centralized control unit. Not only the one-to-all broadcast operation is possible in $O(logN)$ time, N being the number of processors, but also the reduction of data values d_i from the N processors with respect to any binary associative operator $\circ$ (e.g. +, min, etc.), and, most importantly, the N partial products $\bigcirc_{j=1}^{i} d_j \quad i = 1 \ldots N$ (*parallel prefix* operation).

27.2 Time Stepping

Obviously, TPNs with DT and PS find a straightforward *parallel* execution implementation, since start-firing and end-firing events can only occur at instants $k\Delta_t, \quad k = 0, 1, \ldots$ of simulated time. Denote the state of such a TPN by (μ, ϱ), where μ is the marking and ϱ is a $\mathcal{T} \times 1$ vector holding for every $t \in T$ the *remaining time to next event* (start-firing or end firing), then the algorithm at time $k\Delta_t$ executes events related to $t_i \in T$ for which $\varrho_i = 0$ in parallel (as long as there are $t_i \in T$ with $\varrho_i = 0$). After that, for all $\varrho_i > 0$ it sets $\varrho_i = \varrho_i - \Delta_t$ and steps to time $(k+1)\Delta_t$. Clearly, time gaps containing no event occurrences can be "overjumped" employing a *min*-reduction over ϱ to determine the

earliest next event time. In principle, the same execution strategy applies to TPNs with AF and/or CT, but is likely to be less effective due to handling the descheduling with AF, and the far smaller probability of two or more events occuring at exactly the same instant of time (parallelism) with CT. Time stepping will perform best for TPNs with a very high concentration of events at certain points in virtual time, and finds an efficient implementation for SMs, MGs, and CFNs. A distributed conflict resolution scheme necessary for FCNs and GNs can severely complicate the implementation and degrade performance.

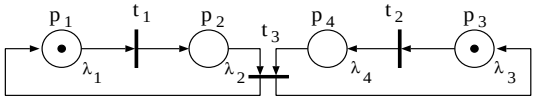


Figure 27.1: PN3: MG with Stochastic Holding Times

27.3 Recurrence Equations

A more general way to execute certain classes of TPNs concurrently was developed by Baccelli and Canales [6], describing in terms of recurrence equations how the TPN dynamically evolves. Consider exponential holding times associated with places p_i as $h_i(1), h_i(2), \ldots$, and (for simplicity) zero enabling/firing times in the MG in Figure 27.1. Then the occurrence of the k-th firing of $t_i \in T$, $ot_i(k) = ot(t_i(\bullet_k))$ can be written as:

$$
\begin{aligned}
ot_1(k) &= h_1(k) \otimes ot_3(k-1), \\
ot_2(k) &= h_3(k) \otimes ot_3(k-1), \\
ot_3(k) &= h_2(k) \otimes ot_1(k) \oplus h_4(k) \otimes ot_2(k),
\end{aligned}
$$

where $\oplus$ denotes max as an associative infix operator, and $\otimes$ denotes the $+$ operator. This set of equations together with $ot_1(0) = ot_2(0) = ot_3(0) = 0$ defines the evolution of PN3, and can, with $ot_3(k) = h_2(k) \otimes h_1(k) \otimes ot_3(k-1) \oplus h_4(k) \otimes h_3(k) \otimes ot_3(k-1)$, be written in matrix form as:

$$
ot(k) = ot(k-1) \otimes A_1(k) \otimes (E \oplus A_0(k)), \tag{27.1}
$$

$$
A_0(k) = \begin{pmatrix} \epsilon & \epsilon & h_2(k) \\ \epsilon & \epsilon & h_4(k) \\ \epsilon & \epsilon & \epsilon \end{pmatrix} \quad A_1(k) = \begin{pmatrix} \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon \\ h_1(k) & h_3(k) & \epsilon \end{pmatrix}
$$

where $\otimes$ and $\oplus$ are operations in the semiring $(\mathcal{R}, \oplus, \otimes)$ with null elements $\epsilon = -\infty$ for $\oplus$ and $e = 0$ for $\otimes$; E is the identity matrix in $(\mathcal{R}, \oplus, \otimes)$. Indeed, (27.1) is a special form of

$$\begin{aligned} ot(k) &= (ot(k-1) \otimes A_1(k) \oplus \cdots \oplus ot(k-M) \otimes A_M(k)) \\ &\quad \otimes (E \oplus A_0(k) \oplus A_0(k)^2 \oplus \cdots \oplus A_0(k)^L) \end{aligned} \tag{27.2}$$

where $A_n(k)$ are matrices representing with their elements ij the holding time of the k-th token in $p \in P$ with $O(t_i) = I(t_j) = p$ having n tokens in $\mu^{(0)}$, L is the maximum number of places on a directed cycle in MG not having a token in $\mu^{(0)}$, and M is the maximum number of tokens in any $p \in P$ in $\mu^{(0)}$. By appropriately rewriting (27.2) and using the associativity of $\otimes$ in $(\mathcal{R}, \oplus, \otimes)$, the underlying MG can be executed either in a *spatially decomposed* way (parallelism from the MG size), or in a *temporally decomposed* way by executing matrix multiplications that simulate the evolving of MG in distinct *epochs of time*, the results of which are *parallel prefixed* with $\otimes$ eventually. Bachelli and Canales [6] achieved excellent performance for MGs with DT or CT for both transitions and places, and FIFO token arrival/consumption. The extension of the approach to higher net classes is possible in principle, but not within $(\mathcal{R}, \oplus, \otimes)$.

27.4 Time Parallelism

A similar approach of exploiting parallelism in time, *time parallelism* (instead of *space parallelism*), for massively parallel simulation has been developed by [Chandy/Sherman 89], and extended by e.g. Lubachevsky, Greenberg and Mitrani [37, 38, 39] or Nicol [64]. In [64] a time simulation is formulated as a fixed-point computations allowing for an almost unlimited parallelism degree. The key to the practical exploitation is the efficient use of the parallel prefix operation, supported in hardware on some massively parallel SIMD multiprocessors.

Consider as an example the simulation of FCFS G/G/1 queue. Let the inter-arrival times be $\{\alpha_i\}_{i \geq 1}$ and the service times be $\{s_i\}_{i \geq 1}$. Now the delay times $\{\delta_i\}_{i \geq 1}$ can be computed according to $\delta_i = \max\{0, (\delta_{i-1} + s_{i-1} - \alpha_i)\}$. This operation has a matrix recurrence formulation in the $(max, +)$ semiring as:

$$D_i = M_i D_{i-1} = M_i M_{i-1} \cdots D_1 \qquad with$$

$$D_i = \begin{bmatrix} \delta_i \\ 0 \end{bmatrix}, \; D_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \; M_i = \begin{bmatrix} \delta_{i-1} - \alpha_i & 0 \\ \infty & 0 \end{bmatrix}$$

A massively parallel simulation scheme now executes this model as follows:

(1) compute partial matrix products $M_i' = M_i M_{i-1} \cdots$ (utilizing parallel prefix)

(2) compute $D_i = M_i' D_1$ for $i = 1, 2, \ldots$ (in parallel)

Chapter 28

Asynchronous (Distributed) Schemes

28.1 Event Driven Simulation Engines

Opposed to PDES, distributed discrete event simulations (DDES) execute TPNs as a spatially decomposed event system dispersed over processors that operate *asynchronously* in parallel (but not necessarily; note the implementation of a DDES scheme for MGs on a SIMD platform by Sellami et.al. [71]). As such, DDES of TPNs is more general than the time stepping approach, since also transition firings occuring at different instants of virtual time are considered for parallel execution. However, DDES potentially induces higher overheads for event list management and complex synchronization protocols not present in PDES.

The basic idea of a DDES is to separate topological TPN parts to be simulated by *logical processes* (LPs). A synchronization protocol provides for a proper synchronization of intra and inter LP event relations with respect to simulated time and causal interdependencies. A DDES of a TPN is organized as in Figure 28.1 (see also [17]), where LPs comprise

R a spatial *region* of TPN: $R_i = (P_i \subseteq P,\ T_i \subseteq T,\ F_i \subseteq F)$ (all event occurrences in R_i will be simulated by LP_i),

SE an event driven *simulation engine* maintaining an event list (EVL), a *local* virtual time (LVT), and a static subset of the state variables $S_i \subseteq S$ defining the state of R_i at any time LVT (we will study cases with $S_i = \cup_j \mu_j \quad \forall p_j \in P_i$), and

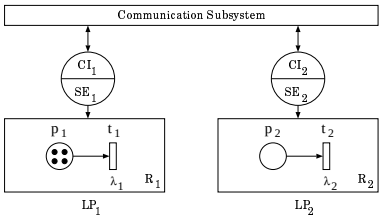


Figure 28.1: Logical Process simulation of TPNs

CI a *communication interface* triggering the local SE and synchronizing with other LPs. CI implements a synchronization protocol either based on the Chandy-Misra-Bryant scheme (conservative), or on Time Warp (optimistic). (Both protocols will be described in detail in the sequel.)

At simulated time LVT, EVL in SE_i holds pending future event occurrences, organized in increasing timestamp order. In case of e.g. AF semantics, these are tuples $\langle t_j @ ot(t_j) \rangle$ denoting the scheduled firing of t_j at time $ot(t_j)$. Once SE_i gets permission from CI_i to fire the first (lowest occurrence time) transition t_j in state S_i, it (i) removes $\langle t_j @ ot(t_j) \rangle$ from EVL, (ii) modifies the state variables according to $S \xrightarrow{t_j} S_i'$, (iii) schedules new tuples $\langle t_k @ ot(t_k) \rangle$ for all $t_k \in E(S_i')$, and (iv), removes tuples $\langle t_l @ ot(t_l) \rangle$ which have lost enabling in S_i' (only in the AF policy). Note that in target environments lacking a shared address space, for performance reasons it is very important to be able to verify $t_k \in E(S_i)$ without having to consult data residing in remote LPs (implicitly assumed here with the notation), thus restricting the freedom in partitioning.

28.2 Partitioning

Thomas and Lazowska [74, 75]) have proposed $R_i = (p \in P)$ (P-LPs) or $R_i = (t \in T)$ (T-LPs), setting up $\mathcal{P} + \mathcal{T}$ LPs. His "Transition Firing Protocol" (TFP) operates in two phases: (i) verify $t_k \in E(\mu)$, t_k residing in LP_{t_k}, and (ii) fire t_k. A fairly communication intensive double-handshake protocol is used for (i), involving the announcement of available tokens by P-LPs, the requesting of a certain amount of tokens necessary for enabling by T-LPs, the granting of the requested amount of tokens by P-LPs, and the confirmation of absorbation of that amount of tokens again by T-LPs. Basically TFP implements a (distributed) token competition resolution policy, but not a conflict resolution mechanism in the sense of the GSPN definition, where user defined *random switches* for competing transition selection can be specified. (ii) im-

plements the removal (deposit) of tokens from (to) P-LPs. P-LPs and T-LPs employ different SEs, which in combination behave *conservatively* (P-LPs do only serve competing *earliest* requests).

Ammar and Deng [5] allow a totally general decomposition of TPNs into regions, but with redundant proxy representations of places that are cut away from their output transitions. A Time Warp based communication interface managing five different types of messages is used to maintain consistency in the "overlapped" state representation (note that here S_i is not a pairwise disjoint partition of μ).

It has been seen by Thomas [74], Nicol and Roy [65] and Chiola and Ferscha [17], that for performance reasons the generality of R_i should be limited in such a way that conflicting transitions together with all their input places should always reside in the same LP, i.e. for $t_k \in T_i$, $t_k \in E(S_i)$ can always be verified without communication. A *minimum region partitioning* and *grain packing* strategy was proposed in [18], that uses two sources of partitioning information: (i) the TPN topology (P, T, F), and (ii) structural properties of the TPN in combination with $\mu^{(0)}$ (if available). Consider the following relations among transitions:

SC $t_i, t_j \in T$ are in *structural conflict* $(t_i \ SC \ t_j)$, iff $I(t_i) \cap I(t_j) \neq \emptyset$

EC $t_i, t_j \in T$ are in *effective conflict* $(t_i \ EC \ t_j)$, iff $t_i, t_j \in E(\mu)$, and $\mu \xrightarrow{t_i} \mu'$ might cause that $t_j \notin E(\mu')$. SC is necessary but not sufficient for EC. Note that in Figure 28.2 $(t_1 \ SC \ t_2)$, but in the interpretation as a GSPN with AF, the probability for $(t_1 \ EC \ t_2)$ is zero for all $\mu^{(i)} \in RS([0,0,0,0,4,1,2]^T)$.

CC $t_i, t_j \in T$ are *causally connected* $(t_i \ CC \ t_j)$, iff $t_i \in E(\mu)$ and $t_j \notin E(\mu)$, $\mu \xrightarrow{t_i} \mu'$ might cause that $t_j \in E(\mu')$.

ME $t_i, t_j \in T$ are *mutually exclusive*, $(t_i \ ME \ t_j)$, iff $\nexists \mu$ s.t. $t_i, t_j \in E(\mu)$. i.e. they cannot be simultaneously enabled simultaneously ME is symmetric and naturally not reflexive.

A sufficient conditions for $(t_i \ ME \ t_j)$ is that the number of tokens in a P-invariant out of which t_i and t_j share places prohibits a simultaneous enabling.

Another sufficient conditions for $(t_i \ ME \ t_j)$ is that $\exists t_k \ : \ \pi_k > \pi_j$ such that $\forall p \in {}^\bullet t_k \ p \in {}^\bullet t_i \cup {}^\bullet t_j \wedge w(t_k, p) \leq \max(w(t_i, p), w(t_j, p))$.

CN $t_i, t_j \in T$ are *concurrent* $(t_i \ CN \ t_j)$, if they are neither EC, nor CC, nor ME. CN is symmetric and reflexive, but not transitive.

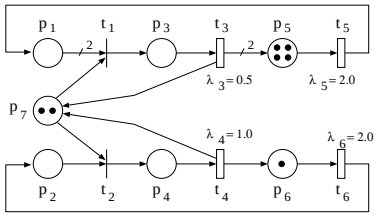


Figure 28.2: Communicating Processes with Resource Contention

$t_i, t_j \in T$ are in *symmetric* SC ($t_i\ SSC\ t_j$) iff $((t_i\ SC\ t_j) \vee (t_j\ SC\ t_i)) \wedge \neg(t_i\ ME\ t_j)$. Note that ($t_i\ SC\ t_i$), ($t_i\ SSC\ t_i$). The transitivity and reflexivity of SSC allows to partition T into equivalence classes of potentially conflicting transitions by computing the transitive closures for $t \in T$ with respect to SSC, (denoted by $ECS(t)$, the extended conflict set of t), which gives the *minimun region partitioning* (MRP) of TPN. (We neglect the possibility of indirect conflict for simplicity here). $ECS(t)$ can be computed in polynomial time in a pre-partitioning analysis for GNs, and is trivially obtained for SMs, MGs, CFNs and FCNs. The MRP for PN1 is $\mathrm{R}_1 = ((p_1, p_7, p_2), (t_1, t_2), F_1)$, $\mathrm{R}_2 = ((p_3), (t_3), F_2)$, $\mathrm{R}_3 = ((p_4), (t_4), F_3)$, $\mathrm{R}_4 = ((p_5), (t_5), F_4)$, $\mathrm{R}_5 = ((p_6), (t_6), F_5)$; for the failure/repair model of Figure 24.7 it is depicted in Figure 28.1.

As another example we look at the net in Figure 28.2: The netmodels the behavior of a system of four computational (c-)processes (p_5), operating in parallel to each other and to one I/O process (p_6). Two c-processes after having done local computations (t_5) compete with another c-process pair and the I/O process for two shared communication devices (e.g. channels, p_7) to be able to communicate (t_3). We obtain the following properties: The net is 4-bounded, not conservative over P, but over (p_2, p_4, p_6) and (p_3, p_4, p_7). It is persistent over (t_3, t_4, t_5, t_6), while t_1 and t_2 are not persistent: firing t_2 in $\mu = (2, 1, 1, 0, 0, 0, 1)^T$ will disable t_1 and vice versa. The initial marking $\mu^{(0)} = [0, 0, 0, 0, 4, 1, 2]^T$ is a homestate. We find the minimum support P-invariants $y_1 = [1, 0, 2, 0, 1, 0, 0]^T$, $y_2 = [0, 0, 1, 1, 0, 0, 1]^T$ and $y_3 = [0, 1, 0, 1, 0, 1, 0]^T$, saying that irrespective of any transition firing $\mu_1^{(i)} + 2\mu_3^{(i)} + \mu_5^{(i)} = 4$, $\mu_3^{(i)} + \mu_4^{(i)} + \mu_7^{(i)} = 2$, and $\mu_2^{(i)} + \mu_4^{(i)} + \mu_6^{(i)} = 1$ in any reachable marking $\mu^{(i)} \in RS(\mu^{(0)})$. Two T-invariants exist: $x_1 = [1, 0, 1, 0, 2, 0]$ and $x_2 = [0, 1, 0, 1, 0, 1]$, expressing that if in $\mu^{(i)}$ there exists a sequence to (exclusively) fire t_1, t_3 and two times t_5 yielding $\mu^{(i+4)}$, $\mu^{(i)} \stackrel{\sigma'}{\rightarrow} \mu^{(i+4)}$, or a sequence σ'' containing t t_2, t_4 and t_6 s.t. $\mu^{(i)} \stackrel{\sigma''}{\rightarrow} \mu^{(i+3)}$, yielding $\mu^{(i+3)}$, then $\mu^{(i)} = \mu^{(i+4)}$ or $\mu^{(i)} = \mu^{(i+3)}$.

Starting from the automatically generated MRP as depicted in Figure 28.3,

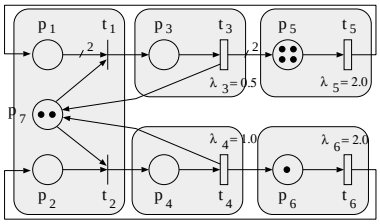


Figure 28.3: MRP of Communicating Processes Net

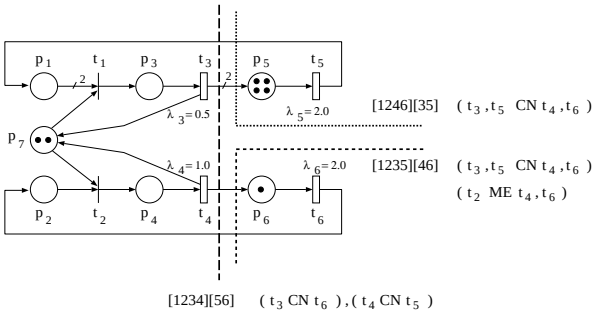


Figure 28.4: Possible Partitionings of Communicating Processes Net

regions of larger "grain size" can be obtained by collapsing R_is using "grain packing" heuristics. A few arguments are: (i) ME transitions do not bear any parallelism, s.t. it might not make sense to separate them into different regions: consider $\mu^{(0)} = [0,0,0,0,4,1,1]^T$, then together with the P-invariant $y_2 = [0,0,1,1,0,0,1]^T$ we have $\mu_3^{(i)} + \mu_4^{(i)} + \mu_7^{(i)} = 1$, which is a sufficient condition for $(t_3\ ME\ t_4)$, and we would "pack" R_2 and R_3 together. Moreover this is sufficient for $(t_1\ ME\ t_3)$ and $(t_2\ ME\ t_4)$ and we could collapse even R_1, R_2 and R_3. (ii) From $\mu^{(0)} = [0,0,0,0,4,1,2]^T$ (two communication devices) together with $y_3 = [0,1,0,1,0,1,0]^T$ we have $\mu_2^{(i)} + \mu_4^{(i)} + \mu_6^{(i)} = 1$ which is a sufficient condition for $(t_4\ CC\ t_6)$ (even $(t_4\ ME\ t_6)$), s.t. the corresponding regions R_3 and R_5 could be collapsed. (iii) From $\mu^{(0)} = [0,0,0,0,4,1,2]^T$ it is trivially seen that $(t_5\ CN\ t_6)$, i.e. executing firings of t_5 and t_6 can indeed exploit model parallelism. Note that also $(t_4\ CN\ t_5)$, $(t_3\ CN\ t_6)$ and $(t_3\ CN\ t_4)$ in $\mu^{(0)} = [0,0,0,0,4,1,2]^T$. As a principle, grainpacking should be done so as to separate CN transitions.

A few "meaningful" partitionings are outlined in Figure 28.4, and will be studied later on.

28.3 Communication Interfaces

The purpose of the CI is to synchronize the transition firings local to the LP's region with firings in remote regions (LPs). The main mechanism for this is to communicate messages among the LPs carrying tokens along with their creation time. These *tokenmessages* are of the form *tokenmessages* of the form $m = \langle \#, i, ts \rangle$ among the LPs, where # is the number of tokens to be deposited in place p_i in the receiver LP, respecting its timestamp ts, i.e. a copy of the senders LVT at the sending instant. Once a tokenmessage is received by the CI of an LP, it integrates the token in the local event structure at LVT = ts to preserve global causalities. Both classical synchronization protocols can be applied for this integration: While Chandy-Misra-Bryant (CMB) based CIs (CI^{CMB}) would prevent the local SE from simulating local transition firings surpassing a LVT horizon for which the LP has been guaranteed not to receive a tokenmessage with ts less than that horizon (by simply blocking SE), a Time Warp based CI (CI^{TW}) would relax this restriction allowing SE to progress local simulation as far into the simulated future as there are scheduled events in EVL. Once a straggler tokenmessage (i.e. one with timestamp less than the current LVT) is received, the CI^{TW} would recover from the causality violation by undoing both local and consequential remote "overoptimistic" simulations in the rollback procedure. Data and control structures for CI^{CMB} and CI^{TW} were worked out in detail in [17].

CMB based CIs rely on model specific information called *lookahead*, that can be obtained from a *future list* [61] of presampled random variates for transition firing times $\tau(t)$. Assume for example that in PN1 exponential variates have been drawn as $\tau(t_4)$ and $\tau(t_6)$ then the lookahead $la(t_2, (t_6, p_2))$ of transition t_2 imposed on the arc (t_6, p_2) is $la(t_2, (t_6, p_2)) = \tau(t_4) + \tau(t_6)$, because in $\mu^{(0)} = [0, 0, 0, 0, 4, 1, 2]^T$ both t_4 and t_6 are *persistent*. This means that once t_2 is fired at LVT, then eventually a token will be generated on arc (t_6, p_2) with timestamp LVT+$la(t_2, (t_6, p_2))$, and no token with a smaller timestamp will be seen on (t_6, p_2). Exactly this information needs to be propagated among LPs using CMB nullmessages to *unblock* an LP, i.e. help to determine an LP when it is *safe* to process an event in the local EVL. A restriction imposed by a CMB CI on the partitioning is, that a nonzero time increment must be passed along with every token message leaving the LP, in order to avoid deadlock due to cyclic waiting for unblocking.

A CI based on *conservative time windows* (to be explained in detail later) has been proposed in [60], that lets SE execute all events e $VT \leq ot(e) < VT + \omega$

without incurring any further communication among LPs, where VT is the (*global*) virtual time present in every LP after a barrier synchronization, and ω is the precomputed (global) time window. With respect to partitioning, this work avoids to make use of information other than the static net topology (P, T, F), but periodically attempts to *dynamically* redistributed LPs after monitoring the intensity and distribution of the workload. The static premapping is based on a sophisticated heuristic for weighted directed graph partitioning.

Time Warp based CIs bear the danger of aggressive memory consumption and the chance of rollback thrashing due to "overoptimism". Adaptive memory management and *lazy cancellation rollback* can improve the shortcomings of this strategy (a detailed view of the optimizations of CMB and Time Warp that can be applied to CIs is given in [29]). Time Warp will perform best if the tokenmessages exchanged among LPs are not widely dispersed over time because rollback probabilities will become unacceptably high. The impact on grain packing is thus to try to balance the average time increment per tokenmessage, which cannot be derived trivially from the net structure.

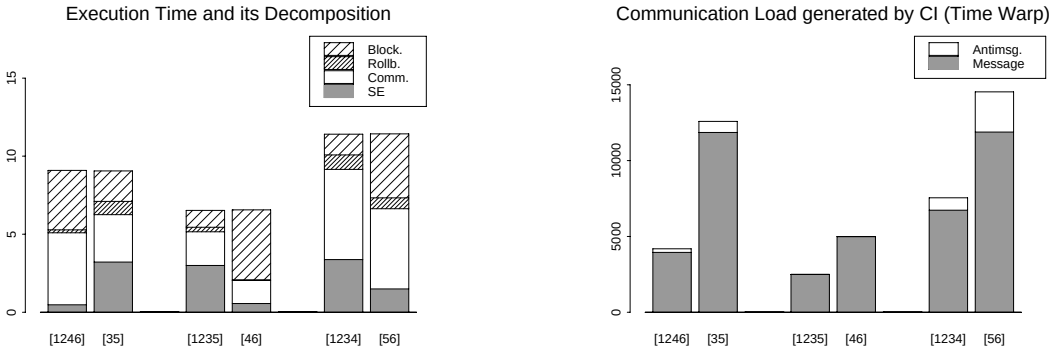


Figure 28.5: Performance of CI^{TW} on the CM-5

Example The performance of a DDES of PN1 as executing on the CM-5 with a Time Warp based, lazy cancellation CI is shown in Figure 28.5. Let $\Pi_1 =$ [1246][35] be a shorthand notation for the partitioning $\text{R}_1 = ((p_1, p_2, p_4, p_6, p_7), (t_1, t_2, t_4, t_6), F_1)$, $\text{R}_2 = ((p_3, p_5), (t_3, t_5), F_2)$, i.e. LP_1 is assigned t_1, t_2, t_4, t_6 with the respective inputplaces and LP_2 is assigned t_3 and t_5 (see Figure 28.4 for an illustration). Similarly let $\Pi_2 =$ [1235][46], $\Pi_3 =$ [1234][56]. The potential gain from Π_1 and Π_2 is $(t_3, t_5\ CN\ t_4, t_6)$, i.e. t_3, t_5 and t_4, t_6 are pairwise CN in $\mu^{(0)} = [0, 0, 0, 0, 4, 1, 2]^T$. The difference among Π_1 and Π_2 is that in Π_2 ME transitions on the same T-invariant $x_2 = [0, 1, 0, 1, 0, 1]$ are separated

from each other: t_2 with (t_2 ME t_4, t_6) is separated from t_4, t_6 (transitions in $x_1 = [1, 0, 1, 0, 2, 0]$ (in case of Π_1) are not pairwise ME). A consequence of this is, that rollback *can never occur* in LP_2 with Π_2 (see middle columns in Figure 28.5). Moreover, no antimessage will ever be generated for (sent to) LP_2. The hope of Π_3 is (t_3 CN t_6) and (t_4 CN t_5). Note that Π_2, despite the violation of the grain packing argument (i), but due to avoiding communication induced by rollback (but suffering from blocking instead) outperforms Π_1 and Π_3 on the CM-5 for the particular $\mu^{(0)}$. The situation would change with a different $\mu^{(0)}$ (e.g. more than one I/O-processes in PN1), and might be different on a platform with a smaller communication-computation speed ratio.

We shall look at the two possibilities for implementing communication interfaces in more detail.

28.4 Chandy-Misra-Bryant Communication Interfaces

LP simulations following a conservative strategy date back to original works by Chandy and Misra [13] and Bryant [11], and are often referred to as the Chandy-Misra-Bryant (CMB) protocols. As described by [58], in CMB causality of events across LPs is preserved by sending timestamped (external) event messages of type $\langle ee@t \rangle$, where ee denotes the event and t is a copy of LVT of the sending LP at (@) the instant when the message was created and sent. $t = ts(ee)$ is also called the *timestamp* of the event. A logical process following the conservative protocol (subsequently denoted by LP^{CMB}) is allowed to process *safe* events only, i.e. events up to a LVT for which the LP has been guaranteed not to receive (external event) messages with LVT $< t$ (timestamp "in the past"). Moreover, all events (internal and external) must be processed in chronological order. This guarantees that the message stream produced by an LP^{CMB} is in turn in chronological order, and a communication system (Figure 26.2) preserving the order of messages sent from LP_i^{CMB} to LP_j^{CMB} (FIFO) is sufficient to guarantee that no out of chronological order message can ever arrive in any LP_i^{CMB} (necessary for correctness). A conservative LP simulation can thus be seen as a set of all LPs $\mathrm{LP}^{CMB} = \bigcup_k \mathrm{LP}_k^{CMB}$ together with a set of directed, reliable, FIFO communication channels $\mathrm{CH} = \bigcup_{k,i\ (k \neq i)} ch_{k,i} = (\mathrm{LP}_k, \mathrm{LP}_i)$ that constitute the *Graph of Logical Processes* $\mathrm{GLP}^{CMB} = (\mathrm{LP}, \mathrm{CH})$. (It is important to note, that GLP^{CMB} has a *static* toplogy, which compared to optimistic protocols, prohibits dynamic (re-)scheduling of LPs in a set of physical proces-

sors.)

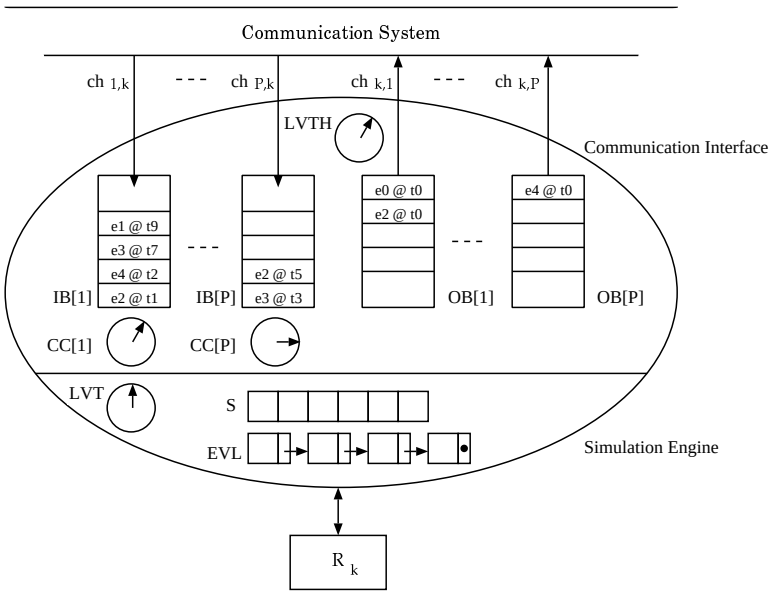


Figure 28.6: Architecture of a *Conservative* Logical Process

The communication interface CI^{CMB} of an LP^{CMB} on the input side maintains one input buffer IB[i] and a channel (or link) clock CC[i] for every channel $ch_{i,k} \in \mathrm{CH}$ pointing to LP_k^{CMB} (Figure 28.6). IB[i] intermediately stores arriving messages in FIFO order, whereas CC[i] holds a copy of the timestamp of the message at the head of IB[i]; initially CC[i] is set to zero. LVTH = $\min_i$ CC[i] is the time horizon up until which LVT is allowed to progress by simulating internal or external events, since no external event can arrive with a timestamp smaller than LVTH. CI now triggers the SE to conduct event processing just like a (sequential) event driven SE (Figure 24.10) based on (internal) events in the EVL, but also process (external) events from the corresponding IBs respecting chronological order and only up until LVT meets LVTH. During this, SE might have produced future events for remote LPs. For each of those, a message is constructed by adding a copy of LVT to the event, and deposited into FIFO output buffers OB[i] to be picked up there and delivered by the communication system. CI maintains individual output buffers OB[i] for every outgoing channel $ch_{k,l}$CH to subsequent LPs LP_l. The basic algorithm is sketched in Figure 28.7.

Given now that within the horizon LVTH neither internal nor external events are available to process, then LP_k^{CMB} *blocks* processing, and idles to receive

```
program LP^CMB(R_k)
S1      LVT = 0; EVL = {}; S = initialstate();
S2      for all CC[i] do (CC[i] = 0) od;
S3      for all ie_i caused by S do chronological_insert(⟨ie_i@occurrence_time(ie_i)⟩, EVL) od;
S4      while LVT ≤ endtime do
S4.1        for all IB[i] do await not_empty(IB[i]) od;
S4.2        for all CC[i] do CC[i] = ts(first(IB[i])) od;
S4.3        LVTH = min_i CC[i];
S4.4        min_channel_index = i | CC[i] == min_channel_clock;
S4.5        if ts(first(EVL)) ≤ LVTH
                then /* select first internal event*/
                     e = remove_first(EVL) ;
                else /* select first external event*/
                     e = remove_first(IB[min_channel_index]);
            end if;
            /* now process the selected event */
S4.6        LVT = ts(e);
S4.7        if not nullmessage(e) then
S4.7.1        S = modified_by_occurrence_of(e);
S4.7.2        for all ie_i caused by S do chronological_insert(⟨ie_i@occurrence_time(ie_i)⟩, EVL) od;
S4.7.3        for all ie_i preempted by S do remove(ie_i, EVL) od;
S4.7.4        for all ee_i caused by S do deposit(⟨ee_i@LVT⟩, corresponding(OB[j])) od;
            end if;
S4.11       for all empty(OB[i]) do deposit(⟨0@LVT + lookahead(ch_{k,i})⟩, OB[i]) od;
S4.12       for all OB[i] do send_out_contents(OB[i]) od;
        od while;
```

Figure 28.7: Conservative LP Simulation Algorithm Sketch.

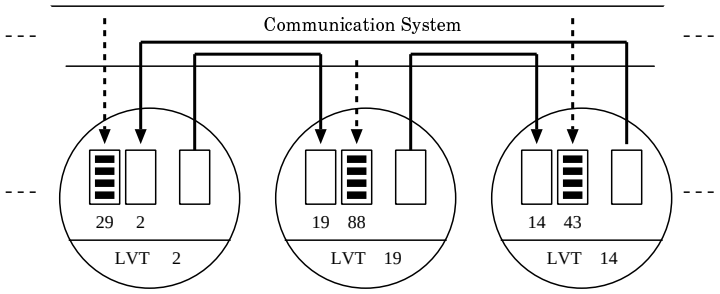


Figure 28.8: Deadlock and Memory Overflow

new messages potentially widening the time horizon. Two key problems appear with this policy of "blocking-until-safe-to-process", namely *deadlock* and *memory overflow* as explained with Figure 28.8: Each LP is waiting for a message to arrive, however, from an LP that is blocked itself (deadlock). Moreover, the cyclic waiting of the LPs involved in deadlock leaves events unprocessed in their input buffers, the amount of which can grow unpredictably even in the absence of deadlock (memory overflow). Several methods have been proposed to overcome the vulnerability of the CMB protocol to deadlock, falling into the two principle categories of *deadlock avoidance* and *deadlock detection/recory*.

28.4.1 Deadlock Avoidance

Deadlock as in Figure 28.8 can be prevented by modifying the communication protocol based on the sending of *nullmessages* [58] of the form $\langle 0@t \rangle$, where 0 denotes a nullevent (event without effect). A nullmessages is *not* related to the simulated model and only serves for synchronization purposes. Essentially it is sent on every output channel as a promise not send any other message with smaller timestamp in the future. It is launched whenever an LP processed an event that did not generate an event message for some corresponding target LP. The receiver LP can use this implicit information to extend its LVTH and by that become unblocked. In our example (Figure 28.8), after the LP in the middle would have broadcasted $\langle 0@19 \rangle$ to the neighboring LPs, both of them would have chance to progress their LVT up until time 19, and in turn issue new event messages expanding the LVTHs of other LPs etc. The nullmessage based protocol can be guaranteed to be deadlock free as long as there are no closed cycles of channels, for which a message traversing this cycle cannot increment its timestamp. This implies, that simulation models whose event structure cannot

be decomposed into regions such that for every directed channel cycle there is at least one LP to put a nonzero time increment on traversing messages cannot be simulated using CMB with nullmessages.

Although the protocol extension is straight-forward to implement, it can put a dramatic burden of nullmessage overhead on the performance of the LP simulation. Optimizations of the protocol to reduce the frequency and amount of nullmessages, e.g. sending them only on *demand* (upon request), delayed until some timeout, or only when an LP becomes blocked have been proposed [58]. An approach where additional information (essentially the routing path as observed during traversal) is attached to the nullmessage, is the *carrier nullmessage protocol* [12].

One problem that still remains with conservative LPs is the determination of when it is safe to process an event. The degree to which LPs can *look ahead* and predict future events plays a critical role in the safety verification and as a consequence for the performance of conservative LP simulations. In the example in Figure 28.8, if the LP with LVT 19 could know that processing the next event will certainly increment LVT to 22, then nullmessages $\langle 0@22 \rangle$ (so called *lookahead* of 3) could have been broadcasted as further improvement on the LVTH of the receivers.

Lookahead must come directly from the underlying simulation model and enhances the prediction of future events, which is – as seen – necessary to determine when it is safe to process an event. The ability to exploit lookahead from FCFS queueing network simulations was originally demonstrated by Nicol [61], the basic idea being that the simulation of a job arriving at a FCFS queue will certainly increment LVT by the service time, which can already be determined, e.g. by random variate presampling, upon arrival since the number of queued jobs is known and preemption is not possible.

Example: Conservative LP Simulation of a PN with Model Parallelism The failure/repair PN model in Figure 24.7 comprises balanced firing delays ($\tau(\mathrm{T1}) \sim exp(0.5)$, $\tau(\mathrm{T2}) \sim exp(0.5)$), i.e. time operating is approximately the same as time being maintained. Related to those firing delays and the number of machines being represented by circulating tokens, a certain amount of model parallelism can be exploited when partitioning the net into two LPs (Figure 28.9), such that the individual PN regions of LP_1 and LP_2 are: $R_1 = (\{\mathrm{T1}\}, \{\mathrm{P1}\}, \{(\mathrm{P1}, \mathrm{T1})\}, \tau(\mathrm{T1}) \sim exp(\lambda = 0.5))$, and $R_2 = (\{\mathrm{T2}\}, \{\mathrm{P2}\}, \{(\mathrm{P2}, \mathrm{T2})\}, \tau(\mathrm{T2}) \sim exp(\lambda = 0.5))$.

Let the *future list* [61], a sequence of exponentially distributed random fir-

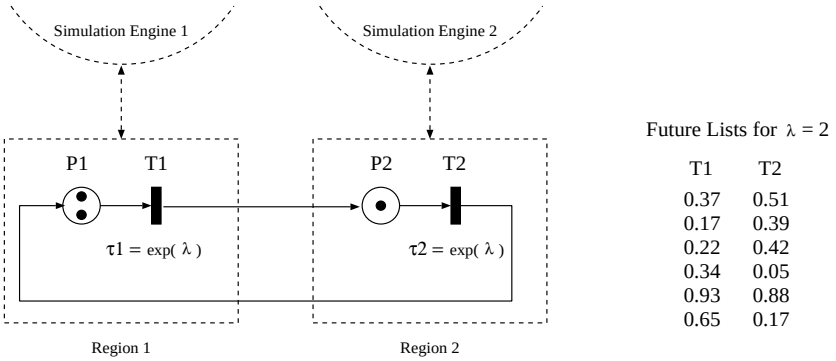


Figure 28.9: LP Simulation of PN1

Step	VT	S	EVL	T
0	0.00	(2,1)	T1@0.17; T1@0.37; T2@0.51	—
1	0.17	(1,2)	T1@0.37; T2@0.51; T2@0.56	T1
2	0.37	(0,3)	T2@0.51; T2@0.56; T2@0.79	T1
3	0.51	(1,2)	T2@0.56; T1@0.73; T2@0.79	T2
4	0.56	(2,1)	T1@0.73; T2@0.79; T1@0.90	T2
5	0.73	(1,2)	T2@0.78; T2@0.79; T1@0.90	T1

Table 28.1: Sequential DES of a PN with Model Parallelism

ing times (random variates), for T1 and T2 be as in the table of Figure 28.9. The sequential simulation would then sequence the variates according to their resulting scheduling in virtual time units when simulating the timed behavior of the PN as in Table 28.1. This sequencing stems from the policy of always using the next free variate from the future list to schedule the occurrence of the next event in EVL.

To explain *model parallelism*, observe that the firing of a scheduled transition (internal event) always generates an external event, namely a message carrying a *token* as the event description (*tokenmessage*), and a *timestamp* equal to the *local virtual time* LVT of the sending LP. On the other hand, the receipt of an event message (external event) always causes a new internal event to the receiving LP, namely the scheduling of a new transition firing in the local EVL. By just looking at the PN model and the variates sampled in the future list (Figure 28.9), we observe that the first occurrence of T1 and the first occurrence of T2 could be simulated in a time overlapped way.

This is explained as follows (Figure 28.10): Both T1 and T2 have infinite

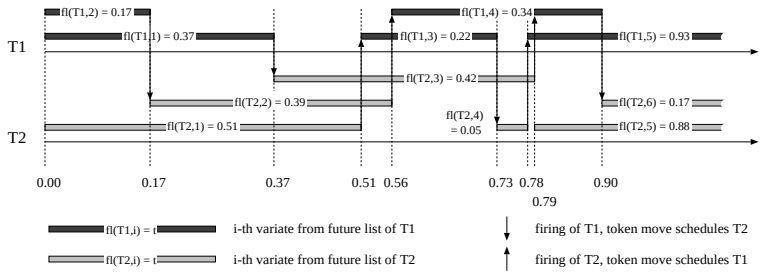


Figure 28.10: Model Parallelism Observed in the PN execution

server firing semantics, i.e. whenever a token arrives in P1 or P2, T1 (or T2) is enabled with a scheduled firing at LVT plus the transitions next future variate. There are constantly $M = 3$ tokens in the PN model, therefore the maximum degree of enabling is M for both T1 and T2. Considering now the initial state $S = (2, 1)$ (two tokens in P_1 and one in P_2), one occurrence of T1 is scheduled for time $t_{\mathrm{T1}} = 0.17$, and another one for $t'_{\mathrm{T1}} = 0.37$. One occurrence of T2 is scheduled for time $t_{\mathrm{T2}} = 0.51$. The next variate for T1 is 0.22, the one for T2 is 0.39. A token can be expected in P1 at $\min(0.51, 0.39, 0.42) = 39$ at the earliest, leading to a new (the third) scheduling of T1 at $0.39 + 0.22 = 0.61$ at the earliest, maybe later. Consequently the *first* occurrence of T1 must be at $t(\mathrm{T1}_1) = 0.17$, and the *second* occurrence of T1 must be $t(\mathrm{T1}_2) = 0.37$. The first occurrence of T2 can be either the one scheduled at 0.51, or the one invoked by the first occurrence of T1 at $0.17 + 0.39 = 0.56$, or the one invoked by the second occurrence of T1 at $0.37 + 0.42 = 0.78$. Clearly, the *first* occurrence of T2 must be at $t(\mathrm{T2}_1) = 0.51$, and the *second* occurrence of T2 must be at $t(\mathrm{T2}_2) = 0.17 + 0.39 = 0.56$, etc. Since $\mathrm{T1}_1 \rightarrow \mathrm{T2}_2$ with $t(\mathrm{T2}_1) < t(\mathrm{T2}_2)$ and $\mathrm{T2}_1 \rightarrow \mathrm{T1}_3$ with $t(\mathrm{T1}_1) < t(\mathrm{T1}_3)$, $\mathrm{T1}_1$ and $\mathrm{T2}_1$ do not interfere with each other and can therefore be simulated independently ($\mathrm{T1}_i \rightarrow \mathrm{T2}_j$ denotes the *direct scheduling* causality of the $i-$th occurrence of T1 onto the $j-$th occurrence of T2).

As was seen, PN1 provides inherent model parallelism. In order to exploit this model parallelism in a CMB simulation, the PN model is decomposed into two regions R_1 and R_2, which are assigned to two LPs LP_1 and LP_2, such that GLP= $(\{\mathrm{LP}_1, \mathrm{LP}_2\}, \{ch_{1,2}, ch_{2,1}\})$, where the channels $ch_{1,2}$ and $ch_{2,1}$ are supposed to substitute the PN arcs $(\mathrm{T1}, \mathrm{P2})$ and $(\mathrm{T2}, \mathrm{P1})$ respectively. Both $ch_{1,2}$ and $ch_{2,1}$ carry messages containing tokens that were generated by the firing of a transition in a remote LP. Consequently, $ch_{1,2}$ propagates a message of the form $m = \langle 1, \mathrm{P2}, t\rangle$ from LP_1 to LP_2 on the occurrence of a firing of

T1, in order to deposit 1 (first component of m) token into place P2 (second component of m) at time t (third component). The timestamp t is produced as a copy of the LVT of LP_1 at the instant of that firing of T1, that produced the token.

A CMB *parallel* execution of the LP simulation model developed above, since operating in a synchronous way in two phases (first simulate one event locally, then transmit messages), generates the trace in Table 28.2. In step 0, both LPs use precomputed random variates from their individual future lists and schedule events. In step 1, no event processing can happen due to LVTH = 0.0, LPs are blocked (see indication in B column. Generally in such a situation every LP_i computes its lookahead $la(ch_{i,j})$ imposed on the individual outputchannels j. In the example we have

$$la(ch_{i,j}) = min(\ (\mathrm{LVT}_i - min_{k=1..S_i}(st_k))\ ,\ min_{k=1..(M-S_i)}\ fl_k\ )$$

where st_k is the scheduled occurrence time of the k-th entry in EVL, fl_k is the k-th free variate in the future list, and M is the maximum enabling degree (tokens in the PN model). For example, the lookahead in LP_1 in the state of step 1 imposed on the channel to LP_2 is $la(ch_{1,2}) = 0.17$, whereas $la(ch_{2,1}) = 0.39$. la is now attached to the LP's LVT, giving the timestamps for the nullmessage $\langle$ 0; P2; 0.17 $\rangle$ sent from LP_1 to LP_2, and $\langle$ 0; P1; 0.39 $\rangle$ sent from LP_2 to LP_1. The latter, when arriving at LP_1, unblocks the SE_1, such that the first event out of EVL_1 can be processed, generating the event message $\langle$ 1; P1; 0.17 $\rangle$. This message, however, as received by LP_2 still cannot unblock LP_2 since it carries the same timestamp as the previous nullmessage; also the local lookahead cannot be improved and $\langle$ 0; P1; 0.51 $\rangle$ is resent. It takes another iteration to finally unblock LP_2, which can then process its first event in step 5, etc. It is easy seen from the example, that the CMB protocol (for the particular example) forces a 'logical' barrier synchronization whenever the sequential DES (see trace in Table 28.1) switches from processing a T1 related event to a T2 related one and vice versa (at VT 0.17, 0.51, 0.73, etc.). In the diagram in Figure 28.10, this is at points where the arrow denoting a token move from T1 (T2) to T2 (T1) has the opposite direction that the previous one.

28.4.2 Deadlock Detection/Recovery

An alternative to the Chandy-Misra-Bryant protocol avoiding nullmessages has also been proposed by Chandy and Misra [14], allowing deadlocks to occur, but providing a mechanism to detect it and recover from it. Their algorithm

Step	LP_1						LP_2					
	IB	LVT	S_{P1}	EVL	OB	B	IB	LVT	S_{P2}	EVL	OB	B
0	—	0.00	2	T1@0.17; T1@0.37	—		—	0.00	1	T2@0.51	—	
1	—	0.00	2	T1@0.17; T1@0.37	⟨ 0; P2; 0.17 ⟩	•	—	0.00	1	T2@0.51	⟨ 0; P1; 0.39 ⟩	•
2	⟨ 0; P1; 0.39 ⟩	0.17	1	T1@0.37	⟨ 1; P2; 0.17 ⟩		⟨ 0; P2; 0.17 ⟩	0.17	1	T2@0.51	⟨ 0; P1; 0.51 ⟩	•
3	⟨ 0; P1; 0.51 ⟩	0.37	0	—	⟨ 1; P2; 0.37 ⟩		⟨ 1; P2; 0.17 ⟩	0.17	2	T2@0.51; T2@0.56	⟨ 0; P1; 0.51 ⟩	•
4	⟨ 0; P1; 0.51 ⟩	0.51	0	—	⟨ 0; P2; 0.73 ⟩	•	⟨ 1; P2; 0.37 ⟩	0.37	3	T2@0.51; T2@0.56; T2@0.79	⟨ 0; P1; 0.51 ⟩	•
5	⟨ 0; P1; 0.51 ⟩	0.51	0	—	⟨ 0; P2; 0.73 ⟩	•	⟨ 0; P2; 0.73 ⟩	0.51	2	T2@0.56; T2@1.79	⟨ 1; P1; 0.51 ⟩	
6	⟨ 1; P1; 0.51 ⟩	0.51	1	T1@0.73	⟨ 0; P2; 0.73 ⟩	•	⟨ 0; P2; 0.73 ⟩	0.56	1	T2@0.79	⟨ 1; P1; 0.56 ⟩	
7	⟨ 1; P1; 0.56 ⟩	0.56	2	T1@0.73; T1@0.90	⟨ 0; P2; 0.73 ⟩	•	⟨ 0; P2; 0.73 ⟩	0.73	1	T2@0.79	⟨ 0; P1; 0.78 ⟩	•
8	⟨ 0; P1; 0.78 ⟩	0.73	1	T1@0.90	⟨ 1; P2; 0.73 ⟩		⟨ 0; P2; 0.73 ⟩	0.73	1	T2@0.79	⟨ 0; P1; 0.78 ⟩	•

Table 28.2: Parallel Conservative LP Simulation of a PN with Model Parallelism

runs in two phases: (*i*) *parallel phase*, in which the simulation runs until it deadlocks, and (*ii*) *phase interface*, which initiates a computation allowing some LP to advance LVT. They prove, that in every parallel phase at least *one* event will be processed generating at least *one* event message, which will also be propagated before the next deadlock. A central *controller* is assumed in their algorithm, thus violating a distributed computing principle. To avoid a single resource (controller) to become a communication performance bottleneck during deadlock detection, any general distributed termination detection algorithm [56] or distributed deadlock detection algorithm [15] could be used instead.

In an algorithm described by Misra [58], a special message called *marker* circulates through GLP to detect and correct deadlock. A cyclic path for traversing all $ch_{i,j} \in CH$ is precomputed and LPs are initially colored *white*. An LP that received the marker takes the color *white* and is supposed to route it along the cycle in *finite* time. Once an LP has either received or sent an event message since passing the marker, it turns to *red*. The marker identifies deadlock if the last N LPs visited were all *white*. Deadlock is properly detected as long as for any $ch_{i,j} \in CH$ all messages sent over $ch_{i,j}$ arrive at LP_j in the time order as sent by LP_i. If the marker also carries the next event times of visited *white* LPs, it knows upon detection of deadlock the smallest next event time as well as the LP in which this event is supposed to occur. To recover from deadlock, this LP is invoked to process its first event. Obviously message lengths in this algorithm grow proportionally to the number of nodes in GLP.

Bain and Scott [8] propose an algorithm for demand driven deadlock free synchronization in conservative LP simulation that avoids message lengths to grow with the size of GLP. If an LP wants to process an event with timestamp t, but is prohibited to do so because $\mathrm{CC}[j] < t$ for some j, then it sends *time requests* containing the sender's process id and the requested time t to all predecessor LPs with this property. (The predecessors, however, may have already advanced their LVT in the mean time.) Predecessors are supposed to inform the sender LP when they can guarantee that they will not emit an event message at a time lower than the requested time t. Three types of reply types are used to avoid repeated polling in the presence of cycles: a *yes* indicates that the predecessor has reached the requested time, a *no* indicates that it has not (in which case another request must be made), and a *ryes* ("reflected yes") indicates that it has conditionally reached t. *Ryes* replys, together with a *request queue* maintained in every LP, essentially have the purpose to detect cycles and to minimize the number of subsequent requests sent to predecessors. If the process id and time of a request received match any request already in the request

queue, a cycle is detected and *ryes* is replied. Otherwise, if the LP's LVT equals or exceeds the requested time a *yes* is replied, whereas if the LP's LVT is less the requested time the request is enqueued in the request queue, and request copies are recursively sent to the receiver's predecessors with CC[i]'s $< t$, etc. The request is complete when all channels have responded, and the request reached the head of the request queue. At this time the request is removed from the request queue and a reply is sent to the requesting LP. The reply to the successor from which the request was received is *no* (*ryes*), if any request to a predecessor was answered with *no* (*ryes*), otherwise *yes* is sent. If *no* was received in an LP initiating a request, the LP has to restart the time request with lower channel clocks.

The *time-of-next-event algorithm* as proposed by Groselj and Tropper [40] assumes more than one LP mapped onto a single physical processor, and computes the greatest lower bound of the timestamps of the event messages expected to arrive next at *all empty* links on the LPs located at that processor. It thus helps to unblock LPs within one processor, but does not prevent deadlocks across processors. The lower bound algorithm is an instance of the single source shortest path problem.

28.4.3 Conservative Time Windows

Conservative LP simulations as presented above are distributed in nature since LPs can operate in a totally asynchronous way. One way to make these algorithms more synchronous in order to gain from the availability of fast synchronization hardware in multiprocessors is to introduce a *window* W_i in simulated time for each LP_i, such that events within this *time window* are *safe* (events in W_i are independent of events in W_j, $i \neq j$) and can be processed concurrently across all LP_i [54], [62].

A conservative time window (CTW) *parallel* LP simulation synchronously operates in two phases. In phase (i) (*window identification*) for every LP_i a chronological set of events W_i is identified such that for every event $e \in \mathrm{W}_i$, e is causally independent of any $e' \in \mathrm{W}_j$, $j \neq i$. Phase (i) is accomplished by a barrier synchronization over all LPs. In phase (ii) (*event processing*) every LP_i processes events $e \in \mathrm{W}_i$ sequentially in chronological order. Again, phase (ii) is accomplished by a barrier synchronization. Since the algorithm iteratively lock-steps over the two consecutive phases, the hope to gain speedup over a purely sequential DES heavily depends on the efficiency of the synchronization operation on the target architecture, but also on the event structure in the

simulation model. Different windows will generally have different cardinality of the covered event set, maybe some windows will remain empty after the identification phase for one cycle. In this case the corresponding LPs would idle for that cycle.

A considerable overhead can be imposed on the algorithm by the identification of when it is safe to process an event within LP_i (window identification phase). Lubachevsky [54] proposes to reduce the complexity of this operation by restricting the *lag* on the LP simulation, i.e. the difference in occurrence time of events being processed concurrently is bounded from above by a known finite constant (*bounded lag* protocol). By this restriction, and assuming a "reasonable" amount of dispersion of events in space and time, the execution of the algorithm on N processors in parallel will have one event processed in $O(logN)$ time on average.

28.5 Time Warp Communication Interfaces

Optimistic LP simulation strategies, in contrast to conservative ones, do not strictly adhere to the local causality constraint *lcc* (see Section 26.2.2), but allow the occurrence of causality errors and provide a mechanism to recover from *lcc* violations. In order to avoid *blocking* and *safe-to-process* determination which are serious performance pitfalls in the conservative approach, an optimistic LP progresses simulation (and by that advances LVT) as far into the simulated future as possible, without warranty that the set of generated (internal and external) events is consistent with *lcc*, and regardless to the possibility of the arrival of an external event with a timestamp in the local past.

Pioneering work in optimistic LP simulation was done by Jefferson and Sowizral [46, 47] in the definition of the Time Warp (TW) mechanism, which like the Chandy-Misra-Bryant protocol uses the sending of messages for synchronization. Time Warp employs a *rollback* (in time) mechanism to take care of proper synchronization with respect to *lcc*. If an external event arrives with timestamp in the local past, i.e. out of chronological order (*straggler message*), then the Time Warp scheme *rolls back* to the most recently saved state in the simulation history consistent with the timestamp of the arriving external event, and restarts simulation from that state on as a matter of *lcc* violation correction. Rollback, however, requires a record of the LP's history with respect to the simulation of internal *and* external events. Hence, an LP^{TW} has to keep sufficient internal state information, say a *state stack* SS, which allows for restoring a past state. Furthermore, it has to administrate an *input queue* IQ and an *output*

queue OQ for storing messages received and sent. For reasons to be seen, this logging of the LP's communication history must be done in chronological order. Since the arrival of event messages in increasing time stamp order cannot be guaranteed, two different kinds of messages are necessary to implement the synchronization protocol: first the usual external event messages ($m^+ = \langle ee@t, + \rangle$), (where again ee is the external event and t is a copy of the senders LVT at the sending instant) which will subsequently call *positive* messages. Opposed to that are messages of type ($m^- = \langle ee@t, - \rangle$) called *negative*- or *antimessages*, which are transmitted among LPs as a request to annihilate the prematurely sent positive message containing ee, but for which it meanwhile turned out that it was computed based on a causally erroneous state.

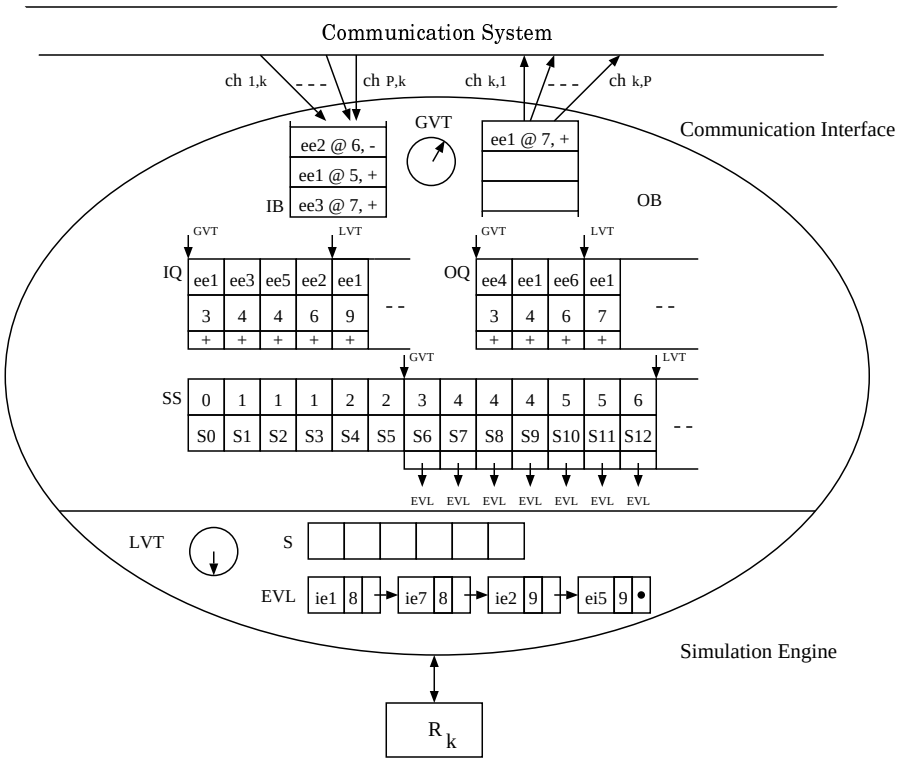


Figure 28.11: Architecture of an *Optimistic* Logical Process

The basic architecture of an optimistic LP employing the Time Warp rollback mechanism is outlined in Figure 28.11. External events are brought to some LP_k by the communication system in much the same way as in the conservative

protocol. Messages, however, are not required to arrive in the sending order (FIFO) in the optimistic protocol, which weakens the hardware requirements for executing Time Warp. Moreover, the separation of arrival streams is also not necessary, and so there is only a single IB and a single OB (assuming that the routing path can be deduced from the message itself). The communication related history of LP_k is kept in IQ and OQ, whereas the state related history is maintained in the SS data structure. All those together represent CI_k; SE_k is an event driven simulation engine equivalent to the one in LP^{CMB}.

The triggering of CI to SE is sketched with the basic algorithm for LP^{TW} in Figure 28.12. The LP mainly loops (*S3*) over four parts: (*i*) an input-synchronization to other LPs (*S3.1*), (*ii*) local event processing (*S3.2 – S3.8*), (*iii*) the propagation of external effects (*S3.9*) and (*iv*) the (global) confirmation of locally simulated events (*S3.10 – S3.11*). Part (*ii*) and (*iii*) are almost the same as was seen with LP^{CMB}. The input synchronization (*Rollback and Annihilation*) and confirmation (*GVT*) part however are the key mechanisms in optimistic LP simulation.

28.5.1 Rollback and Annihilation Mechanisms

The input synchronization of LP^{TW} (rollback mechanism) relates arriving messages to the current value of the LP's LVT and reacts accordingly (see Figure 28.13). A message affecting the LP's "local future" is moved from the IB to the IQ respecting the timestamp order, and the encoded external event will be processed as soon as LVT advances to that time. $\langle ee3@7, + \rangle$ in the IB in Figure 28.11 is an example of such an unproblematic message (LVT = 6). A message timestamped in the "local past" however is an indicator of a causality violation due to tentative event processing. The rollback mechanism (*S3.1.1*) in this case restores the most recent *lcc*-consistent state, by reconstructing S and EVL in the simulation engine from copies attached to the SS in the communication interface. Also LVT is warped back to the timestamp of the straggler message. This so far has compensated the *local* effects of the *lcc* violation; the *external* effects are annihilated by sending an antimessage for all previously sent outputmessages (in the example $\langle ee6@6, - \rangle$ and $\langle ee1@7, - \rangle$ are generated and sent out, while at the same time $\langle ee6@6, + \rangle$ and $\langle ee1@7, + \rangle$ are removed from OQ). Finally, if a negative message is received (e.g. $\langle ee2@6, - \rangle$) it is used to annihilate the dual positive message ($\langle ee2@6, + \rangle$) in the local IQ. Two cases for the negative messages must be distinguished: (*i*) If the dual positive message is present in the receiver IQ, then this entry is deleted as an annihilation. This can

```
program LP^TW(R_k)
S1      GVT = 0; LVT = 0; EVL = {}; S = initialstate();
S2      for all ie_i caused by S do chronological_insert(⟨ie_i@occurrence_time(ie_i)⟩, EVL) od;
S3      while GVT ≤ endtime do
S3.1        for all m ∈ IB do
S3.1.1          if ts(m) < LVT /* m potentially affects local past */
                  then
                      if (positive(m) and dual(m) ∉ IQ) or (negative(m) and dual(m) ∈ IQ)
                        then /* rollback */
                            restore_earliest_state_before(ts(m));
                            generate_and_sendout(antimessages);
                            LVT = earliest_state_timestamp_before(m);
                      endif;
                endif;
                /* irrespective of how m is related to LVT */
S3.1.2          if dual(m) ∈ IQ
                  then remove(dual(m), IQ);   /* annihilate */
                  else  chronological_insert(external_event(m)@ts(m), sign(m)), IQ);
                endif;
            od;
S3.2        if ts(first(EVL)) ≤ ts(first_nonnegative(IQ))
                then e = remove_first(EVL);                 /* select first internal event*/
                else  e = first_nonnegative(IQ);            /* select first external event*/
            endif;
            /* now process the selected event */
S3.3        LVT = ts(e);
S3.4        S = modified_by_occurrence_of(e);
S3.5        for all ie_i caused by S do chronological_insert(⟨ie_i@occurrence_time(ie_i)⟩, EVL) od;
S3.6        for all ie_i preempted by S do remove(ie_i, EVL) od;
S3.7        log_new_state(⟨LVT, S, copy_of(EVL)⟩, SS);
S3.8        for all ee_i caused by S do
                deposit(⟨ee_i@LVT, +⟩, OB);
                chronological_insert(⟨ie_i@LVT, +⟩, OQ);
            od;
S3.9        send_out_contents(OB);
S3.10       GVT = advance_GVT();
S3.11       fossil_collection(GVT);
        od while;
```

Figure 28.12: Optimistic LP Simulation Algorithm Sketch.

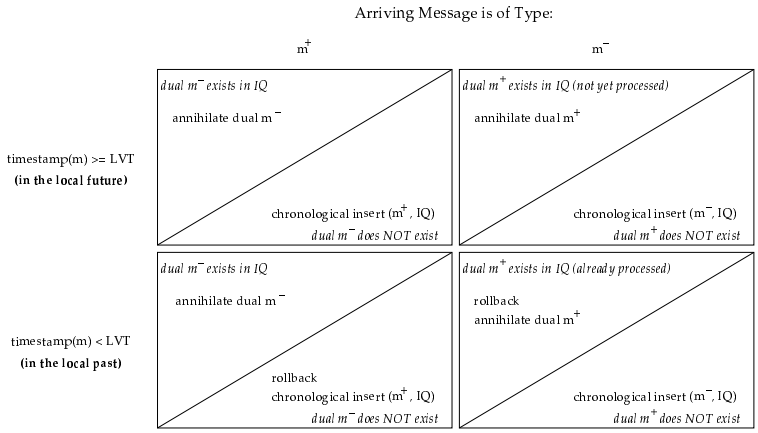


Figure 28.13: The Time Warp Message based Synchronization Mechanism

be easily done if the positive message has not yet been processed, but requires rollback if it was. (*ii*), if a dual positive message is not present (this case can only arise if the communication system does not deliver messages in a FIFO fashion), then the negative message is inserted in IQ (irrespective of its relation to LVT) to be annihilated later by the (delayed) positive message still in traffic.

As is seen now, the rollback mechanism requires a periodic saving of the states of SE (LVT, S and EVL) in order to able to restore a past state (*S3.7*), and to log output messages in OQ to be able to undo propagated external events (*S3.8*). Since antimessages can also cause rollback, there is the chance of *rollback chains*, even recursive rollback if the cascade unrolls sufficiently deep on a directed cycle of GLP. The protocol however guarantees, although consuming considerable memory and communication resources, that *any* rollback chain eventually terminates whatever its length or recursive depth is.

Lazy Cancellation

In the original Time Warp protocol as described above, an LP receiving a *straggler message* initiates sending antimessages immediately when executing the rollback procedure. This behavior is called *aggressive cancellation*. As a performance improvement over *aggressive cancellation*, the *lazy cancellation* policy does not send an antimessage (m^-) for m^+ immediately upon receipt of a straggler. Instead, it delays its propagation until the resimulation after rollback has progressed to LVT = ts(m^+) producing $m^{+'} \neq m^+$. If the resimulation produced $m^{+'} = m^+$, no antimessage has to be sent at all [34]. Lazy cancellation

thus avoids unnecessary cancelling of correct messages, but has the liability of additional memory and bookkeeping overhead (potential antimessages must be maintained in a rollback queue) and delaying the annihilation of actually wrong simulations.

Lazy cancellation can also be based on the utilization of lookahead available in the simulation model. If a straggler $m^+ <$ LVT is received, than obviously antimessages do not have to be sent for messages m with timestamp, $ts(m^+) \leq ts(m) < ts(m^+) + la$. Moreover, if $ts(m^+) + la \geq$ LVT even rollback does not need to be invoked. As opposed to lookahead computation in the CMB protocol, lazy cancellation can exploit implicit lookahead, i.e. does not require its explicit computation.

The traces in Table 28.3 represent the behavior of the optimistic protocol with the lazy cancellation message annihilation in a *parallel* LP simulation of PN1. (The trace table is to be read in the same way as the one in Table 28.2, except that there is a rollback indicator column RB instead of a blocking column B.) In step 2, for example, LP_2 receives the straggler $\langle 1; P1; 0.17\rangle$ at LVT $= 0.51$. Message annihilation and rollback can be avoided due to the exploitation of the lookahead from the next random variate in the future list, 0.39. The effect of the straggler is in the future of LP_2 (0.56).

It has been shown [45] that Time Warp with lazy cancellation can produce so called "supercritical speedup", i.e. surpass the simulations critical path by the chance of having wrong computations produce correct results. By immediately discarding rolled back computations this chance is lost for the aggressive cancellation policy. A performance comparison of the two, however, is related to the simulation model. Analysis by Reiher and Fujimoto [70] shows that lazy cancellation can arbitrarily outperform aggressive cancellation and vice versa, i.e. one can construct extreme cases for lazy and aggressive cancellation such that if one protocol executes in α time using N processors, the other uses αN time. Nevertheless, empirical evidence is reported "slightly" in favor of lazy cancellation for certain simulation applications.

Lazy Reevaluation

Much like *lazy cancellation* delays the annihilation of external effects upon receiving a straggler at LVT, *lazy re-evaluation* delays discarding entries on the state stack SS. Should the recomputation after rollback to time $t <$ LVT reach a state that exactly matches one logged in SS *and* the IQ is the same as the one at that state, then immediately *jump forward* to LVT, the time before rollback occured. Thus, lazy reevaluation prevents from the unnecessary recomputation

Step	LP_1						LP_2					
	IB	LVT	S	EVL	OB	RB	IB	LVT	S	EVL	OB	RB
0	—	0.00	2	T1@0.17; T1@0.37	—		—	0.00	1	T2@0.51	—	
1	—	0.17	1	T1@0.37	⟨ 1; P2; 0.17 ⟩		—	0.51	0	—	⟨ 1; P1; 0.51 ⟩	
2	⟨ 1; P1; 0.51 ⟩	0.37	1	T1@0.73	⟨ 1; P2; 0.37 ⟩		⟨ 1; P2; 0.17 ⟩	0.56	0	—	⟨ 1; P1; 0.56 ⟩	
3	⟨ 1; P1; 0.56 ⟩	0.73	1	T1@0.90	⟨ 1; P2; 0.73 ⟩		⟨ 1; P2; 0.37 ⟩	0.79	0	—	⟨ 1; P1; 0.79 ⟩	
4	⟨ 1; P1; 0.79 ⟩	0.90	1	T1@1.72	⟨ 1; P2; 0.90 ⟩		⟨ 1; P2; 0.73 ⟩	0.73	2	T2@0.78; T2@0.79	⟨ -1; P1; 0.79 ⟩	•
5	⟨ -1; P1; 0.79 ⟩	0.90	0	—	—	•	⟨ 1; P2; 0.90 ⟩	0.78	2	T2@0.79; T2@1.78	⟨ 1; P1; 0.78 ⟩	

Table 28.3: Parallel Optimistic LP Simulation of a PN with Model Parallelism

of correct states and is therefore promising in simulation models where events do not modify states ("read-only" events). A serious liability of this optimization is again additional memory and bookkeeping overhead, but also (and mainly) the considerable complication of the Time Warp code [33]. To verify equivalence of IQ's the protocol must draw and log copies of the IQ in every state saving step (*S3.7*). In a weaker *lazy re-evaluation* strategy one could allow jumping forward only if no message has arrived since rollback.

Lazy Rollback

The difference of virtual time in between the straggler m^*, $ts(m^*)$, and its actual effect at time $ts(m^*) + la(ee) \geq$ LVT can again be overjumped, saving the computation time for the resimulation of events in between $[ts(m^*), ts(m^*) + la(ee))$. $la(ee)$ is the lookahead imposed by the external event carried by m^*.

Breaking/Preventing Rollback Chains

Besides the postponing of erroneous message and state annihilation until it turns out that they are not reproduced in the repeated simulation, other techniques have been studied to break cascades of rollbacks as early as possible. Prakash and Subramanian [67], comparable to the carrier null message approach, attach a limited amount of state information to messages to prevent recursive rollbacks in cyclic GLPs. This information allows LPs to filter out messages based on preempted (obsolete) states to be eventually annihilated by chasing antimessages currently in transit. Related to the (conservative) *bounded lag* algorithm, Lubachevsky, Shwartz and Weiss have developed a *filtered rollback* protocol [55] that allows optimistically crossing the lag bound, but only up to a time window upper edge. Causality violations can only affect the time period in between the window edge and the lag bound, thus limiting (the relative) length of rollback chains. The SRADS protocol by Dickens and Reynolds [26], although allowing optimistic simulation progression, prohibits the propagation of uncommitted events to other LPs. Therefore, rollback can only be *local* to some LP and cascades of rollback can never occur. Madisetti, Walrand and Messerschmitt with their protocol called *Wolf-calls* freeze the spatial spreading of uncommitted events in so called *spheres of influence* $W(\mathrm{LP}_i, \tau)$, defined as the set of LPs that can be influenced by a message from LP_i at time $ts(m) + \tau$ respecting computation times a and communication times b. The Wolf algorithm ensures that the effects of an uncommitted event generated by LP_i are limited to a sphere of a computable (or selectable) radius around LP_i, and the

number of broadcasts necessary for a complete annihilation within the sphere is bounded by a computable (or chooseable) number of steps B (B being provably smaller than for the standard Time Warp protocol).

28.5.2 Optimistic Time Windows

A similar idea of "limiting the optimism" to overcome rollback overhead potentials is to advance computations by "windows" moving over simulated time. In the original work of Sokol, Briscoe and Wieland [73], the *moving time window* (MTW) protocol, neither internal nor external events e with $ts(e) > t + \Delta$ are allowed to be simulated in the time window $[t, t + \Delta)$, but are postponed for the next time window $[t + \Delta, t + 2\Delta)$. Two events e and e' timestamped $ts(e)$ and $ts(e')$ respectively therefore can only be simulated in parallel iff $\mid ts(e) - ts(e') \mid < \Delta$. Naturally, the protocol is in favor of simulation models with a low variation of event occurrence distances relative to the window size. Compared to a time-stepped simulation, MTW does not await the completion of *all* events e with $t \leq ts(e) < t + \Delta$ which would cause idle processors at the end of each time window, but invokes an attempt to move the window as soon as the number of events to be executed falls below a certain threshold. In order to keep moving the time window, LPs are polled for the timestamp of their earliest next event $t_i(e)$ (polling takes place simultaneously with event processing) and the window is advanced to $\min_i t_i(e), \min_i t_i(e) + \Delta$. (The next section will show the equivalence of the window lower edge determination to GVT computation.) Obviously the advantage of MTW and related protocols is the potential effective implementation as a *parallel* LP simulation, either on a SIMD architecture or in a MIMD environment where the reduction operation $\min_i t_i(e)$ can be computed utilizing synchronization hardware. Points of criticism are the assumption of approximately uniform distribution of event occurrence times in space and the ignorance with respect to potentially "good" optimism beyond the upper window edge. Furthermore, a natural difficulty is the determination of the Δ admitting enough events to make the simulation efficient.

The latter is addressed with the *adaptive Time Warp concurrency control algorithm* (ATW) proposed by Ball and Hyot [9], allowing the window size $\Delta(t)$ be adapted at any point t in simulation time. ATW aims to temporarily suspend event processing if it has observed a certain amount of *lcc* violations in the past. In this case the LP would conclude that it progresses LVT too fast compared to the predecessor LPs and would therefore stop LVT advancement

for a time period called the *blocking window* (BW). BW is determined based on the minimum of a function describing wasted computation in terms of time spent in a (conservatively) blocked mode, or a fault recovery mode as induced by the Time Warp rollback mechanism.

28.5.3 The Limited Memory Dilemma

All arguments on the execution of the Time Warp protocol so far assumed the availability of a *sufficient* amount of free memory to record internal and external effect history for pending rollbacks, and all arguments were related to the time complexity. Indeed, Time Warp with certain memory management strategies to be described in the sequel can be proven to work correctly when executed with $O(M^{seq})$ memory, where M^{seq} is the number of memory locations utilized by the corresponding sequential DES. Opposed to that, the CMB protocol may require $O(kM^{seq})$ space, but may also use less storage than sequential simulation, depending on the simulation model (it can even be proven that simulation models exist such that the space complexity of CMB is $O((M^{seq})^k)$). Time Warp *always* consumes more memory than sequential simulation [53], and a memory limitation imposes a performance decrease on Time Warp: providing just the minimum of memory necessary may cause the protocol to execute fairly slow, such that the memory/performance tradeoff becomes an issue.

Memory management in Time Warp follows two goals: (*i*) to make the protocol *operable* on real multiprocessors with bounded memory, and (*ii*) to make the execution of Time Warp *performance efficient* by providing "sufficient" memory. An *infrequent* or *incremental* saving of history information in some cases can *prevent*, maybe more effectively than one of the techniques presented for *limiting the optimism* in Time Warp, aggressive memory consumption. Once, despite the application of those techniques, available memory is exhausted, *fossil collection* could be applied as a technique to recover memory used for history recording that will definitely not be used anymore due to an assured lower bound on the timestamp of any possible future rollback (GVT). Finally, if even fossil collection fails to recover enough memory to proceed with the protocol, additional memory could be freed by returning messages from the IQ (*message sendback, cancelback*) or invoking an *artificial rollback* reducing space used for storing the OQ.

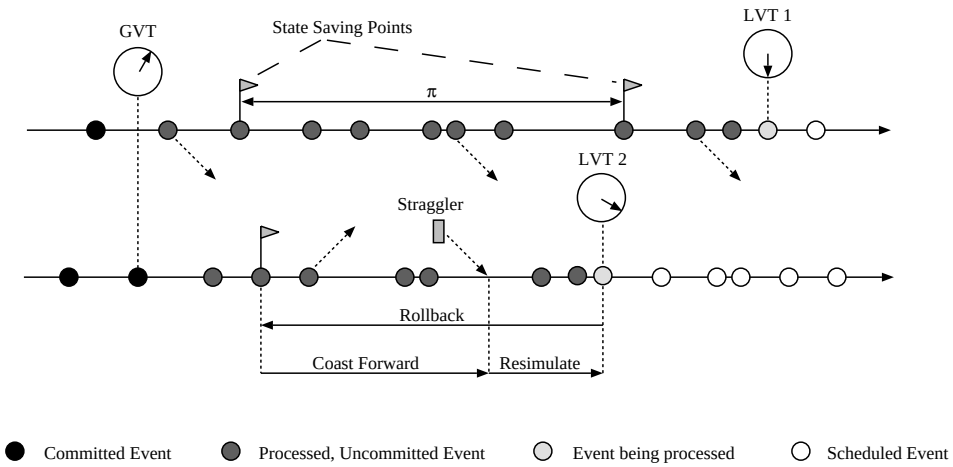


Figure 28.14: Interleaved State Saving

28.5.4 Incremental and Interleaved State Saving

Minimizing the storage space required for simulation models with complex sets of state variables $S_i \subset S$, S_i being the subset stored and maintained by LP_i that do not extensively change values over LVT progression, can be accomplished effectively by just saving the variables $s_j \in S_i$ affected by a state change. This is mainly an implementation optimization upon step *S3.4* in the algorithm in Figure 28.12. This *incremental* state saving can also improve the execution complexity in step *S3.7*, since generally less data has to be copied into the logrecord. Obviously the same strategy could be followed for the EVL, or the IQ in a lazy reevaluation protocol. Alternatively, imposing a condition upon step *S3.7*:

S3.7 **if** (step_count *modulo* π) == 0 **then** log_new_state(⟨LVT, S, copy_of(EVL)⟩, SS);

could be used to *interleave* the continuity of saved states and thus on the average reduce the storage requirement to $\frac{1}{\pi}$ of the noninterleaved case.

Both optimizations, however, increase the execution complexity of rollback. In incremental state saving protocols, desired states have to be reconstructed from increments following back a path further into the simulated past than required by rollback itself. The same is true for interleaved state saving, where the most recent *saved* state older than the straggler must be searched for, a reexecution up until the timestamp of the straggler (*coast forward*) must be

started which is a clear waste of CPU cycles since it just reproduces states that have already been computed but were not saved, and finally the straggler integration and usual reexecution are necessary (Figure 28.14). The tradeoff between state saving costs and the coast forward overhead has been studied (as reported by [64] in reference to Lin and Lazowska) based on expected event processing time ($\epsilon = E[exec(e)]$) and state saving costs (σ), giving an optimal *interleaving factor* π^* as

$$\lfloor\sqrt{(\alpha-1)\beta}\rfloor < \pi^* < \lceil\sqrt{(2\alpha+1)\beta}\rceil$$

where α is the average number of rollbacks with $\pi = 1$ and $\beta = \frac{\sigma}{\epsilon}$. The result expresses that an overestimation of π^* is more severe to performance than an underestimation by the same (absolute) amount. In a study of the optimal checkpointing interval explicitly considering state saving and restoration costs while assuming π does neither affect the number of rollbacks nor the number of rolled back events in [52], an algorithm is developed that, integrated into the protocol, "on-the-fly", within a few iterations, automatically adjusts π to π^*. It has been shown that some π, though increasing the rollback overhead, can reduce overall execution time.

28.5.5 Fossil Collection

Opposed to techniques that reclaim memory temporarily used for storing events and messages related to the *future* of some LP, fossil collection aims to return space used by history records that will no longer be used by the rollback synchronization mechanism. To assure from which state in the history (and back) computations can be considered fully committed, the determination of the value of *global virtual time* (GVT) is necessary.

Consider the tuple

$$\Sigma_i(T) = (\mathrm{LVT}_i(T), \mathrm{IQ}_i(T), \mathrm{SS}_i(T), \mathrm{OQ}_i(T))$$

to be a *local snapshot* of LP_i at *real time* T, i.e. LVT, IQ_i is the input queue as seen by an external observer at *real time* T, etc., and $\Sigma(T) = \bigcup_{i=1}^{N} \Sigma_i(T) \cup \mathrm{CS}(T)$ be the *global snapshot* of GLP. Further let $\mathrm{LVT}_i(T)$ be the local virtual time in LP_i, i.e. the timestamp of the event being processed at the observation instant T, and $UM_{i,j}(T)$ the set of external events imposed by LP_i upon LP_j encoded as messages m in the snapshot Σ. This means m is either in transit on channel $ch_{i,j}$ in CS or stored in some IQ_j, but not yet processed at time T. Then the GVT at real time T is defined to be:

$$\mathrm{GVT}(T) = \min(\min_i \mathrm{LVT}_i(T), \min_{i,j;m \in UM_{i,j}(T)} ts(m))$$

It should be clear even by intuition, that at *any* (real time) T, GVT(T) represents the maximum lower bound to which any rollback could ever backdate LVT_i ($\forall i$). An obvious consequence is that any processed event e with $ts(e) < \mathrm{GVT}(T)$ can never (at no instant T) be rolled back, and can therefore be considered as *(irrevocably) committed* [53] (Figure 28.14). Further consequences (for all LP_i) are that:

(i) messages $m \in \mathrm{IQ}_i$ with $ts(m) \leq \mathrm{GVT}(T)$, as well as messages $m \in \mathrm{OQ}_i$ with $ts(m) \leq \mathrm{GVT}(T)$ are obsolete and can be discarded (from IQ, OQ) after real time T.

(ii) state variables $s \in S_i$ stored in SS_i as with $ts(s) \leq \mathrm{GVT}(T)$ are obsolete and can be discarded after real time T.

Making use of these possibilities, i.e. getting rid of external event history according to (i) and of internal event history according to (ii) that is no longer needed to reclaim memory space is the idea behind *fossil collection*. It is called as a procedure in the abstracted Time Warp algorithm (Figure 28.12) in step *S3.11*. The idea of reclaiming memory for history earlier than GVT is also expressed in Figure 28.11, which shows IQ and OQ sections for entries with timestamp later than GVT only, and copies of EVL in SS if not older than GVT (also the rest of SS beyond GVT could be purged as irrelevant for Time Warp, but we assume here that the state trace is required for a post-simulation analysis).

Generally, a combination of fossil collection with any of the incremental or interleaved state saving schemes is recommended. Related to interleaving, however, rollback might be induced to events beyond the momentary committed GVT, with an average overhead directly proportional π. Not only that the interleaving of state recording is prohibiting fossil collection for states timestamped in the gap between GVT and the most recent saved state chronologically before GVT, it is also contraproductive to GVT computation which is comparably more expensive than state saving as will be seen soon.

28.5.6 Freeing Memory by Returning Messages

Previous strategies (interleaved, incremental state saving as well as fossil collection) are merely able to reduce the chance of memory exhaustion, but cannot

actually prevent such situations from occurring. In cases where memory is already completely allocated, only additional techniques, mostly based on returning messages to senders or artificially initiating rollback, can help to escape from deadlocks due to waiting for free memory:

Message Sendback

The first approach to recover from memory overflow in Time Warp was proposed by the *message sendback* mechanism by Jefferson [47]. Here, whenever the system runs out of memory on the occasion of an arriving message, part or all af the space used for saving the *input history* is used to recover free memory by returning *unprocessed* input messages (not necessarily including the one just received) back to the sender and relocating the freed (local) memory. By intuition, input messages with the highest send timestamps are returned first, since it is more likely that they carry incorrect information compared to "older" input messages, and since the annihilation of their effects can be expected not to disperse as much in virtual time, thus restricting annihilation influence spheres. Related to the original definition of the Time Warp protocol which distinguishes the *send time* (ST) and *receive time* (RT) ($\mathrm{ST}(m) \leq \mathrm{RT}(m)$) of messages, only messages with $\mathrm{ST}(m) > \mathrm{LVT}$ (local *future messages*) are considered for returning. An indirect effect of the sendback could also be storage release in remote LPs due to annihilation of messages triggered by the original sender's rollback procedure.

Gafni's Protocol

In a message traffic study of *aggressive* and *lazy cancellation*, Gafni [35] notes that *past* ($\mathrm{RT}(m) < \mathrm{GVT}$) *and present* messages ($\mathrm{ST}(m) < \mathrm{GVT} < \mathrm{RT}(m)$) and events accumulate in IQ, OQ, SS for the two annihilation mechanisms at the same rate, pointing out also the interweaving of messages and events in memory consumption. Past messages and events can be fossil collected as soon as a new value of GVT is available. The amount of "present" messages and events present in LP_i reflects the difference of LVT_i to the global GVT directly expressing the asynchrony or "imbalance" of LVT progression. This fact gives an intuitive explanation of the *message sendback*'s attempt to *balance* LVT progression across LPs, i.e. intentionally rollback those LPs that have progressed LVT ahead of others. Gafni, considering this asynchrony to be exactly the source from which Time Warp can gain real execution speedup, states that LVT progression balancing is does not solve the storage overflow problem. His algorithm reclaims

memory by relocating space used for saving the *input* or *state* or *output* history in the following way: Whether the overflow condition is raised by an arriving input message, the request to log a new state or the creation of a new output message, the element (message or event) with the *largest* timestamp is selected irrespective of its type.

- If it is an *output message*, a corresponding antimessage is sent, the element is removed from OQ and the state before sending the original message is restored. The antimessage arriving at the receiver will find its annihilation partner in the receiver's IQ upon arrival (at least in FIFO CSs), so memory is also reclaimed in the receiver LP.
- If it is an *input message*, it is removed from IQ and returned to the original sender to be annihilated with its dual in the OQ, perhaps invoking rollback there. Again also the receiver LP relocates memory.
- If it is a *state* in SS, it is discarded (and will be recomputed in case of local rollback).

The desirable property of both *message sendback* and Gafni's protocol is that LPs that ran out of memory can be relieved without shifting the overflow condition to another LP. So, given a certain minimum but limited amount of memory, both protocols make Time Warp "operable".

Cancelback

An LP simulation memory management scheme is considered to be storage *optimal* iff it consumes $O(M^{seq})$ *constant bounded* memory [53]. The worst case space complexity of Gafni's protocol is $O(NM^{seq}) = O(N^2)$ (irrespective of whether memory is shared or distributed), the reason for this being that it can only cancel elements within the individual LPs. *Cancelback* is the first optimal memory management protocol [44], and was developed targeting Time Warp implementations on shared memory systems. As opposed to Gafni's protocol, in Cancelback elements can be *canceled* in *any* LP_i (not necessarily in the one that observed memory overflow), whereas the element selection scheme is the same. Cancelback thus allows to selectively reclaim those memory spaces that are used for the *very* most recent (globally seen) input-, state- or output-history records, whichever LP maintains this data. An obvious implementation of Cancelback is therefore for shared memory environments and making use of system level interrupts. A Markov chain model of Cancelback [2] predicting speedup as the

amount of available memory beyond M^{seq} is varied, revealed that even with small fractions of additional memory the protocol performs about as well as with unlimited memory. The model assumes totally symmetric workload and a constant number of messages, but is verified with empirical observations.

Artificial Rollback

Although Cancelback theoretically solves the memory management dilemma of Time Warp since it produces correct simulations in real, limited memory environments with the same order of storage requirement as the sequential DES, it has been criticized for its implementation not being straightforward, especially in distributed memory environments. Lin [53] describes a Time Warp management scheme that is in turn memory *optimal* (there exists a shared memory implementation of Time Warp with space complexity $O(M^{seq})$ [1]), but has a simpler implementation. Lin's protocol is called *artificial rollback* for provoking the rollback procedure not only for the purpose of *lcc*-violation restoration, but also for its side effect of reclaiming memory (since rollback as such does not affect operational correctness of Time Warp, it can also be invoked *artificially*, i.e. even in the absence of a straggler). Equivalent to Cancelback in effect (cancelling an element generated by LP_j from IQ_i is equivalent to a rollback in LP_j, whereas cancelling an element from OQ_i or SS_i is equivalent to a rollback in LP_i), artificial rollback has a simpler implementation since the rollback procedure already available can be used together with an artificial rollback trigger causing only very little overhead. Determining, however, in which LP_i to invoke artificial rollback, to what LVT to rollback and at what instant of real time T to trigger it is not trivial (except the triggering, which can be related to the overflow condition and the failure of fossil collection). In the implementation proposed by Lin and Preiss [53], the two other issues are coupled to a processor scheduling policy in order to guarantee a certain amount of free memory (called *salvage parameter* in [64]), while following the "cancel-furthest-ahead" principle.

28.6 Adaptive Simulation Protocols

A general argument on the relative performances of CMB and TW protocol implementations cannot be given due to an overwhelming degree of interdepen-

[1] For implementations in distributed memory environments, Time Warp with artificial rollback cannot guarantee a space complexity of $O(M^{seq})$. *Cancelback* and Artificial Rollback in achieving the sequential DES storage complexity bound rely on the availability of a global, shared pool of (free) memory.

dencies among performance influencing factors [29], mainly coming from the *event structure* of the *simulation model* (inherent model parallelism), the *partitioning* of the simulation model into submodels to be executed by LPs (locality of causal effects, balance of virtual time progression, balance of load across LPs, etc.), the performance characteristics of the target *hardware* (processor speed, memory size, cache hierarchies, memory access speed communication latency, etc.), the *communication and synchronization model* on the target machine (routing strategy, nonblocking communication modes, hardware support for multicast and reduction operations, buffering, etc.), and last but not least implementation *optimizations*. (A "performance comparable implementation" is proposed in [19].) Experience, however, with implementations of parallel SEs for distributed memory environments that necessarily need to employ synchronization schemes based on the exchange of time stamped messages often reveals communication overhead to be the dominating performance factor. The reduction of overhead induced by the sending of messages is therefore of outstanding practical interest. Protocols that regulate the degree of "optimism" underlying TW according to hypotheses that LPs are establishing during simulation on the amount of available model parallelism have therefore been studied. The "aggressiveness" of TW can be adaptively throttled to that point in the spectrum between unlimited optimism and extreme pessimism, that is the best compromise considering a particular simulation model. This has lead to so called *adaptive protocols* (see Figure 28.15).

In [68] the possibilities of options for combining CMB and TW to attain the capabilities of both are classified as (*i*) *relaxing conservativism* in CMB protocols, (*ii*) *limiting the optimism* in TW protocols, and *seamlessly switching* between optimistic and conservative schemes. As already mentioned, the Adaptive Time Warp (ATW) protocol [9] allows the execution of events e with time stamp $ts(e)$ only if $ts(e) \in [t, t + \Delta)$, where Δ is adapted dynamically according to the number of local causality constraint violations in the past. In the Global Synchronization protocol [63], LPs exchange send/receive counts based on which it is decided whether to advance simulation to the next global synchronization step, or to await and process forthcoming message. In [30] a protocol is proposed that temporarily blocks an LP if a potential straggler messages (that would induce rollback) is anticipated. The Local Adaptive Protocol (LAP) [41] in much the same way uses statistical information from observing the simulation performance on the fly to dynamically adjust optimism control parameters. An explicit cost function to determine whether it is more "cost effective" to execute an arriving message instantaneously despite rollback hazard,

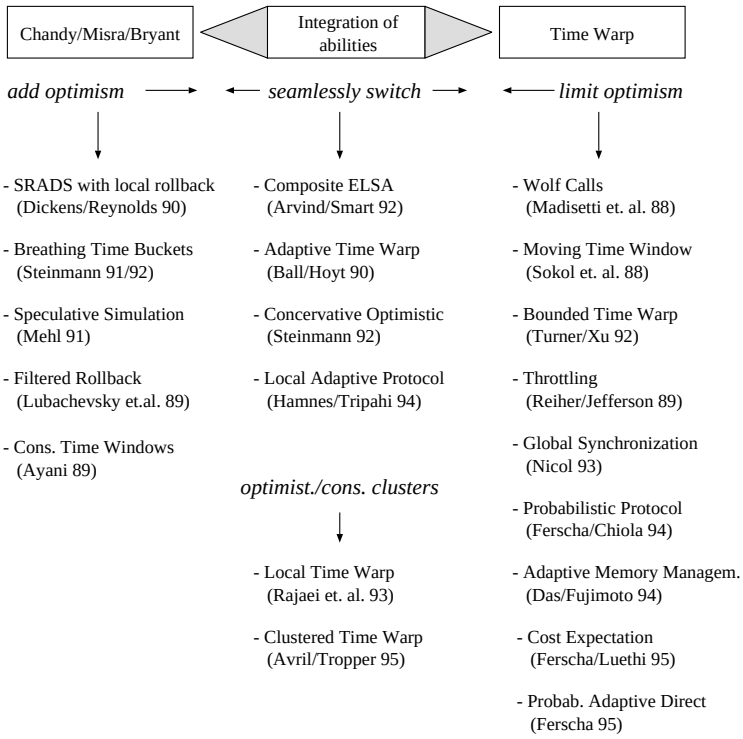


Figure 28.15: A Spectrum of Options inbetween TW and CMB.

or to delay its execution to avoid potential rollback/antimessage is reported in [32]. All these protocols use statistical data that reflects the central tendency (e.g. average increase of local virtual time and average (real and simulated) message interarrival time in LAP; average commitment rate in AMM) to determine parameters for optimism control in TW.

28.6.1 Adaptive Memory Management

The *adaptive memory management* (AMM) scheme proposed by Das and Fujimoto [23] attempts a combination of controling optimism in Time Warp and an *automatic* adjustment of the amount of memory in order to optimize fossil collection, Cancelback and rollback overheads. Analytical performance models of Time Warp with Cancelback [2] for homogeneous (artificial) workloads have shown that at a certain amount of available free memory fossil collection is sufficient to allocate enough memory. With a decreasing amount of available memory, absolute execution performance decreases due to more frequent can-

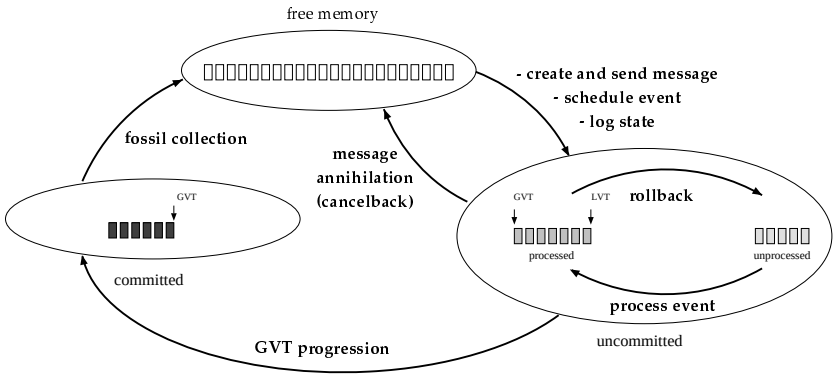


Figure 28.16: "Flow of buffers" in the AMM protocol

celbacks until it becomes frozen at some point. Strong empirical evidence has been given as a support to this analytical observations. The motivation now for an *adaptive* mechanism to control memory is twofold: (*i*) absolute performance is supposed to have negative increments after reducing memory even further. Indeed, one would like to run Time Warp in the area of the "knee-point" of absolute performance. A successive adaptation to that performance optimal point is desired. (*ii*) the location of the knee might vary during the course of simulation due to the underlying simulation model. A runtime adaptation to follow movements of the knee is desired.

The AMM protocol [24] for automatic adjustment of available storage uses a memory *flow model* that divides the available (limited) memory space M into three "pools", $M = M^c + M^{uc} + M^f$. M^c is the set of all memory locations used to store committed elements ($t(e) \leq$ GVT), M^{uc} is its analogy for uncommitted events (in IQ, OQ, or SS with $t(e) >$ GVT) and M^f holds temporarily unused (free) memory. The behavior of Time Warp can now be described in terms of flows of (fixed sized) memory buffers (able to record one message or event for simplicity) from one pool into the other (Figure 28.16): Free memory must be allocated for every message created/sent, every state logged or any future event scheduled, causing buffer moves from M^f to M^{uc}. Fossil collection on the other hand returns buffers from M^c as invoked upon exhaustion of M^f, whereas M^c is being supplied by the progression of GVT. Buffers move from M^{uc} to M^f with each message annihilation, either incurred by rollback or by Cancelback. A *Cancelback cycle* is defined by two consecutive invocations of cancelback. A cycle starts where Cancelback was called due to failure of fossil collection to reclaim memory; at this point there are no buffers in M^c. Progression of

LVT will move buffers to M^{uc}, rollback of LVT will occasionally return free memory, progression of GVT will deposit into M^c to be depleted again by fossil collection, but in tendency the free pool will be drained, thus necessitating a new cancelback.

Time Warp can now be controlled by two (mutually dependent) parameters: (i) α, the amount of processed but uncommitted buffers left behind after cancelback, as a parameter to control optimism; and (ii) β, the amount fo buffers freed by Cancelback, as a parameter to control the cycle length. Obviously, α has to be chosen small enough to avoid rollback thrashing and overly aggressive memory consumption, but not too small in order to prevent rollbacks of states that are most likely to be confirmed (events on the critical path). β should be chosen in such a way as to minimize the overhead caused by unnecessary frequent cancelback (and fossil collection) calls. The AMM protocol now by monitoring the Time Warp execution behavior during one cycle, attempts to simultaneously minimize the values of α and β, but respecting the constraints above. It assumes Cancelback (and fossil collection) overhead to be directly proportional to the Cancelback invocation frequency. Let $\varrho_{M^{uc}} = \frac{e_{committed}N}{T_{process}} - \varrho_{FC}$ be the rate of growth of M^{uc}, where $e_{committed}$ is the fraction of processed events also committed during the last cycle, $T_{process}$ is the average (real) time to process an event and ϱ_{FC} is the rate of depletion of M^{uc} due to fossil collection. (Estimates for the right hand side are generated from monitoring the simulation execution.) β is then approximated by

$$\beta = (T_{cycle} - T_{CB,FC})\varrho_{M^{uc}}$$

where $T_{CB,FC}$ is the overhead incurred by Cancelback and fossil collection in real time units, and T_{cycle} is the current invocation interval. Indeed, α is a parameter to control the upper tolerable bound for the progression of LVT. To set α appropriately, AMM records by a marking mechanism whether an event was rolled-back by Cancelback. A global (across all LPs) counting mechanism lets AMM determine the number $\#(e_{cp})$ of events that should *not* have been rolled back by Cancelback, since they were located on the critical path, and by that causing a definitive performance degrade[2]. Starting now with a high parameter value for α (which will give an observation $\#(e_{cp}) \simeq 0$), α is continuously

[2]The *Critical Path* of a DES is computed in terms of the (*real*) processing time on a certain target architecture respecting *lcc*. Traditionally, *critical path analysis* has been used to study the performance of distributed DES as reference to an "ideal", fastest possible asynchronous distributed execution of the simulation model. Indeed, it has been shown that the length of the critical path is a lower bound on the execution time of *any* conservative protocol, but *some* optimistic protocols do exist (Time Warp with lazy cancellation, Time Warp with

reduced as long as $\#(e_{cp})$ remains negligible. Rollback thrashing is explicitly tackled by a third mechanism that monitors $e_{committed}$ and reduces α and β to their halves when the decrease of $e_{committed}$ hits a predefined threshold.

Experiments with the AMM protcol have shown that both the claimed needs can be achieved: Limiting optimism in Time Warp *indirectly* by controlling the rate of drain of free memory can be accomplished effectively by a dynamically adaptive mechanism. AMM adapts this rate towards the performance knee-point automatically, and adjusts it to follow dynamical movements of that point due to workloads varying (linearly) over time.

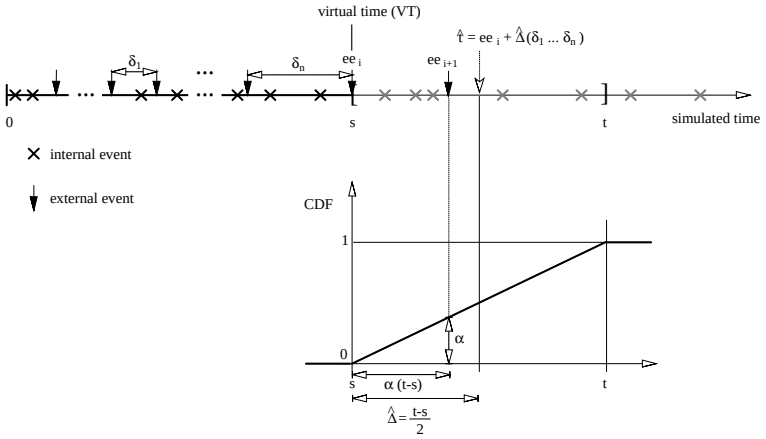


Figure 28.17: Motivation for the Probabilistic DDES

28.6.2 Probabilistic Optimism

A communication interface CI that considers the CMB protocol and Time Warp as two extremes in a spectrum of possibilities to synchronize LP's LVT advancements is the *probabilistic* distributed DES protocol [31]. Let the occurrence of an event e, $t(e) = \mathrm{LVT}_i$, in some LP_i be *causal* ($\rightarrow$) for a future event e' with $t(e') = \mathrm{LVT}_i + \delta(t)$ in LP_j with probability $P[e \rightarrow e']$, i.e. event e with some probability changes state variables in S_j with influence on e'. Then the CMB "block until safe-to-process"-rule appears adequate for $P[e \rightarrow e'] = 1$, and is

lazy rollback, Time Warp with phase decomposition, and the Chandy-Sherman Space-Time Method [45], which can surpass the critical path. The possibility of so called *supercritical speedup*, and as a consequence its nonsuitability as an *absolute* lower bound reference, however, has made critical path analysis less attractive.

overly pessimistic for cases $P[e \rightarrow e'] < 1$ since awaiting the decision whether $(e \rightarrow e')$ or $(e \not\rightarrow e')$ hinders the (probably correct) concurrent execution of events e and e' in different LPs. Clearly, if $P[e \rightarrow e'] \ll 1$ an optimistic strategy could have executed e' concurrently to e most of the time. More specifically (see Figure /refmotprob), assume the timestamp of the *next* (token-)message m_{i+1} to arrive at LP_j in the time interval $[a, b]$ (b can be arbitrarily large), i.e. $a \leq ts(m_{i+1}) \leq b$. Then LP_j could have safely simulated all the scheduled transition firings $\langle t_k @ ot(t_k) \rangle$ with $ot(t_k) < ts(m_{i+1})$; CI^{CMB} however will block at some $\mathrm{LVT}_j = a$ (Figure 28.21). As an example, let the probability of the actual timestamp of the forthcoming message be $P[ts(m_i) = x] = \frac{1}{b-a} \quad \forall x \in [b, a]$, then CMB, by blocking, fails to use a $(1 - \alpha)$ chance to produce useful simulation work by progressing over a and executing scheduled transition firings in the time interval $[a, a + \alpha(b - a)]$. This is a clear *overpessimism* at least for small values of α. On the other hand, CI^{TW} would allow executing even scheduled firings $ot(t_k) > b$, which is a clear *overoptimism* because every firing $ot(t_k) > b$ definitely will have to be rolled back, incurring annihilation messages that could have been avoided.

These arguments mainly motivate the use of an optimistic CI for simulation models with nondeterministic event causalities. On the other hand, as already seen, an optimistic LP might tend towards overoptimism, i.e. optimism lacking rational justification. The *probabilistic* protocol aims to exploit information on the dynamic simulation behavior in order to be able to allow in every simulation step just that amount of optimism that can be justified.

To give an intuition on the justification of optimism look again at the parallel simulation of the small simulation model in Figure 24.7 together with the future list, and the parallel execution of *lazy cancellation* Time Warp in Figure 28.3. At the beginning of step 3 LP_2 at LVT = 0.56 (= LVT at the end of step 2) faces the straggler $\langle$ 1; P2; 0.37 $\rangle$ in its IQ; the next element in LP_2's future list is 0.42. Since the effect of the straggler is in the local future of LP_2, i.e. $\langle$ T2@(0.37+0.42) $\rangle$, the *lazy rollback* strategy applies and rollback is avoided at all. The event $\langle$ T2@0.79 $\rangle$ is executed in that step, setting LVT = 0.79, and the outputmessage $\langle$ 1; P1; 0.79 $\rangle$ is generated (OQ) and sent at the end of the step. Unfortunately, in step 4 a new straggler $\langle$ 1; P2; 0.73 $\rangle$ is observed in IQ of LP_2, but now with the effect that at time $t = 0.73 + 0.05 <$ LVT = 0.79 LP_2 is forced to roll back (Figure 28.3). Indeed, LP_2 in step 3 generated and sent out $\langle$ 1; P1; 0.79 $\rangle$ without considering any information whether the implicit optimism is justified or not. If LP_2 would have observed that it received "on the average" one input message per step, with an "average" timestamp increment of 0.185, it might have

Step	LP_1					LP_1				
	IB	LVT	S	EVL	OB	IB	LVT	S	EVL	OB
0	—	0.00	2	T1@0.17; T1@0.37	—	—	0.00	1	T2@0.51	—
1	—	0.17	1	T1@0.37	⟨ 1; P2; 0.17 ⟩	—	0.51	0	—	⟨ 1; P1; 0.51 ⟩
2	⟨ 1; P1; 0.51 ⟩	0.37	1	T1@0.73	⟨ 1; P2; 0.37 ⟩	⟨ 1; P2; 0.17 ⟩	0.56	0	—	⟨ 1; P1; 0.56 ⟩
3	⟨ 1; P1; 0.56 ⟩	0.73	1	T1@0.90	⟨ 1; P2; 0.73 ⟩	⟨ 1; P2; 0.37 ⟩	0.56	1	T2@0.79	—
4	—	0.90	0	—	⟨ 1; P2; 0.90 ⟩	⟨ 1; P2; 0.73 ⟩	0.78	1	T2@0.79	⟨ 1; P1; 0.78 ⟩

Table 28.4: Motivation for Probabilistic LP Simulation

established a hypothesis that in step 4 a message is expected to arrive with an estimated timestamp of 0.37+0.185 = 0.555 (= timestamp of previous message + average increment). Taking this as an alarm for potential rollback, LP_2 could have avoided the propagation of the local optimistic simulation progression by e.g. delaying the sendout of ⟨ 1; P1; 0.79 ⟩ for one step. This is illustrated in Figure 28.4: LP_2 just takes the input message from IQ and schedules the event ⟨ T2@0.79 ⟩ in EVL, but does *not* process it. Instead, the execution is delayed until the hypothesis upon the next message's timestamp is verified. The next message is ⟨ 1; P1; 0.73 ⟩, the hypothesis can be dropped, and a new event ⟨ T2@0.78 ⟩ is scheduled and processed next. Actually two rollbacks and the corresponding sending of antimessages could be avoided by applying the probabilistic simulation scheme.

To be able to directly control the optimism in TW, each LP in the PADOC (Probabilistic Adaptive Direct Optimism Control) [28] approach monitors the LVT progression on each of its incident channels, i.e. logs the timestamps of messages as they arrive. From the observed message arrival patterns, each LP formulates a hypothesis on the timestamp of the next message expected to arrive, and – related to statistical confidence in the forecast value – by means of throttling *adapts* to a synchronization behavior that is presumably the best tradeoff among blocking (CMB) and optimistically progressing (TW) with respect to this hypothesis in the current situation. Throttling is done *probabilistically* in the sense that *blocking* is induced with a certain probability.

Assume that the history over the last n message arrivals $A_k = (ts(m_{i-n+1}), ts(m_{i-n+2}), \ldots ts(m_i))$ is maintained in LP_l for every (input) channel $ch_{k,l}$, and that $\widehat{ts}(m_{i+1}))$ is an estimate for the timestamp of the forthcoming message m_{i+1}. Let the confidence $0 \leq \zeta(\widehat{ts}(m_{i+1}))) \leq 1$. express the "trust" in this estimate. Then LP_l having scheduled t_r as the transition to fire next, say at $ot(t_r)$, would execute the occurrence of t_r with some probability P_ζ[execute $\langle t_r @ ot(t_r)\rangle$], but would block for the average amount of CPU time $\overline{s}$ (used to simulate one transition firing) with probability $1 - P_\zeta$. The algorithm sketch of the PADOC (Probabilistic Direct Optimism Control) LP simulation engine in Figure 28.18 explains further details.

Note that in contrast to other adaptive TW mechanisms that compute an optimal delay window for blocking the simulation engine [9, 41, 32], the PADOC engine blocks for a fixed amount of real time (i.e. $\overline{s}$), but loops over the blocking decision, incrementally establishing longer blocking periods. By this discretization of the "blocking window" PADOC preserves the possibility to use information on the arrival process encoded in the timestamps of messages that arrive in

```
program PADOC_Simulation_Engine( )
1       initialize();
2       while GVT ≤ endtime do
2.1           for all arriving messages m do
                  update(arrivalstatistics, m);
                  if ts(m) < LVT /* m affects local past */
                    then /* rollback */
                           restore_earliest_state_before(ts(m));
                           generate_and_sendout(antimessages);
                    else  chronological_insert(m, IQ);
                  endif;
              od;
2.2           ts^ = forecast(arrivalstatistics);
2.3           ζ(ts^) = confidence_in_forecast(arrivalstatistics);
2.4           if ts(first(EVL)) ≤ ts(first_nonnegative(IQ))
                then
                     if (1 − P_ζ[exec first(EVL)]) ≤ rand()
                       then /* delay execution */
                              delay(s̄);
                       else  process(first(EVL));
                     endif;
                else  process(first_nonnegative(IQ));
              endif;
2.5           sendout(outputmessages);
2.6           fossil_collection(advance_GVT());
        od while;
```

Figure 28.18: PADOC Simulation Engine.

between blocking phases. Algorithms based on variable size blocking windows fail to make use of intermediate message arrivals.

Incremental Forecast Methods

Predicting the timestamp of the forthcoming message m_{i+1} after having observed n arrivals is explained in Figure 28.19. Basically, by statistically analyzing the arrival instants $ts(m_{i-n+1}), ts(m_{i-n+2}), \ldots ts(m_i)$, an estimate $\widehat{ts}(m_{i+1}) = ts(m_i) + \widehat{\Delta}(\delta_1, \delta_2, \ldots \delta_n)$ is generated, where $\delta_k = ts(m_{i-n+k}) - ts(m_{i-n+k-1})$ is the difference in timestamps of two consecutive messages. (Note that δ_k is negative if m_{i-n+k} is a straggler.)

The choice of the size of the observation history n as well as the selection of the forecast procedure is critical for the performance of the PADOC engine for two reasons: (*i*) the achievable prediction *accuracy* and (*ii*) the computa-

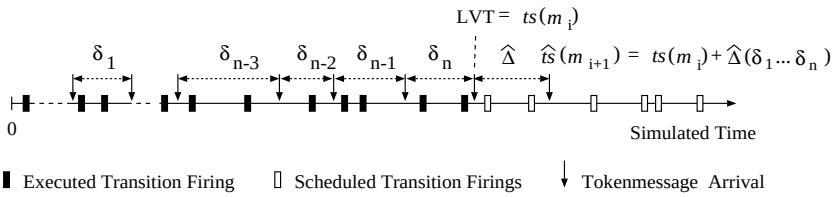


Figure 28.19: Message Timestamp Forecast

tional and space complexity of the forecast method. Generally, the larger n, the more information on the arrival history is available in the statistical sense. Considering much of the arrival history will at least theoretically give a higher prediction precision, but will also consume more memory space. Intuitively, complex forecast methods could give "better" predictions than trivial ones, but are liable to intrude on the distributed simulation protocol with an unacceptable amount of computational resource consumption. Therefore, *incremental* forecast methods of low memory complexity are recommended, i.e. procedures where $\widehat{ts}(m_{i+2})$ can be computed from the previous forecast $\widehat{ts}(m_{i+1})$ and the actual observation $ts(m_{i+1})$ in $\mathcal{O}(c)$ instead of $\mathcal{O}(cn)$ time.

Arithmetic Mean If no observation window is imposed on the arrival history, but all observed δ_j's are considered, then the observed mean $\widehat{\Delta}_i = \frac{1}{n}\sum_{j=1}^{n}\delta_j$ as an estimate of the timestamp of the forthcoming message has a recursive form. Upon the availability of the next time difference δ_{n+1}, $\widehat{ts}(m_{i+1})$ can be computed incrementally as:

$$\widehat{\Delta}_{n+1} = \frac{n\widehat{\Delta}_n + \delta_{n+1}}{n+1}.$$

Exponential Smoothing The arithmetic mean based forecast considers all observations δ_j as equally "important". A possibility to express the history as an exponentially weighted sum (e.g. give recent history higher "importance" than past history) is the exponential smoothing of the observation vector by a smoothing factor α ($|1-\alpha| < 1$). $\widehat{\Delta}$ in this case has the incremental form

$$\widehat{\Delta}_{n+1} = \alpha\delta_{n+1} + (1-\alpha)\widehat{\Delta}_n$$

$\alpha \approx 1$ gives a high weight to the last observation, and potentially yields a high variation in the forecasts. $\alpha \approx 0$ causes intense smoothing, making forecasts less reactive to shocks in the arrival process. We use the smoothing factor

$$\alpha = min_{\overline{\alpha}\in(0.1,0.2,\ldots,0.9)}(\sum_{i=1}^{n-1}[\delta_{i+1} - \widehat{\Delta}_{i+1}(\overline{\alpha})]^2),$$

		Execution Time	%-Simulation		%-Rollback	
			LP1	LP2	LP1	LP2
DD	TW	0.46	12.7	13.7	5.8	16.1
	TW+M	0.60	12.5	33.4	12.1	5.7
	TW+S	0.65	16.4	35.9	11.5	6.3
	TW+A	0.62	23.7	41.0	6.8	5.4
SD	TW	0.68	10.9	13.8	9.8	17.2
	TW+M	0.84	19.2	35.0	6.8	8.9
	TW+S	0.96	19.4	37.7	6.8	8.4
	TW+A	0.64	19.8	41.8	5.8	6.8
DS	TW	0.64	10.7	13.8	8.4	17.9
	TW+M	1.51	22.1	35.6	8.2	8.7
	TW+S	0.66	17.1	39.2	6.0	8.2
	TW+A	1.17	24.3	43.5	5.8	9.5
SS	TW	0.91	11.0	15.0	11.7	15.3
	TW+M	0.57	16.6	37.6	7.3	6.9
	TW+S	1.03	19.8	37.9	7.4	9.6
	TW+A	0.91	23.1	41.6	6.8	9.3

Table 28.5: TW with M, S and A for CM-5.

which is periodically readjusted during the simulation.

Median Approximation The virtual time increments in general cannot be assumed to yield a nonskewed, unimodal distribution of values, as is implicitly assumed when the arithmetic mean is used as an index of central tendency. Particularly, if the frequency of time increments has a pdf skewed to the left, then the arithmetic mean is higher in value than the median, and would thus overestimate the next message's timestamp. A consequence would be "overpessimism" in the blocking policy. Forecast based on the median would use the estimate

$$\widehat{\Delta} = \begin{cases} \delta_{\left(\frac{n-1}{2}\right)} & \text{n odd} \\ \frac{1}{2}\left(\delta_{\left(\frac{n}{2}\right)} + \delta_{\left(\frac{n+1}{2}\right)}\right) & \text{else} \end{cases}$$

which cannot be computed incrementally, as new timestamp increments have to be inserted in a sorted list of δ_i's to find the value of the median afterwards. As an approximation for the median we have developed the following supplement. Let $\mathcal{M} = \frac{\text{median}}{\text{mean}}$, which is a constant for every distribution (e.g. for the exponential distribution we have $(\frac{1}{\lambda}\ln 2)/\frac{1}{\lambda} = \ln 2$). Then with

$$\widehat{\mathcal{M}}(s) = \frac{\text{Median}(\delta_s + \delta_{s-1} + \ldots + \delta_{s-n+1})}{\frac{1}{n}(\delta_s + \delta_{s-1} + \ldots + \delta_{s-n+1})}$$

we find a forecast based on a median which is approximated by the arithmetic mean as

$$\widehat{\Delta} = \widehat{\mathcal{M}}\frac{1}{n}(\delta_1 + \ldots + \delta_n)$$

The performance of the three "straightforward" forecast methods (arithmetic mean (M), exponential smoothing (S) and approximated median (A)) applied

to the SPN in Figure 24.7 with 4 tokens in the initial marking and different timing scenarios is summarized in Table 28.5. In the scenario referred to as DD both T1 and T2 obey deterministic, but imbalanced timing (τ(T1) = 1, τ(T2) = 8). In the second case, SD, T1 has stochastic timing with τ(T1) $\sim$ $exp(1)$, but T2 is deterministically timed as τ(T2) = 8. Similarly, DS represents τ(T1) = 1 and τ(T2) $\sim$ $exp(1/8)$, whereas SS represents τ(T1) $\sim$ $exp(1)$, τ(T2) $\sim$ $exp(1/8)$. Note that in any case LVT progression in LP_2 is (on average) eight times higher than in LP_1 causing significant load imbalance and rollback (communication) overhead. (All forecast confidences are kept constant at ζ = 0.9 for comparability, (TW) refers to TW with unlimited optimism.) Sample arrival process traces as collected on the CM-5 are depicted in Figure 28.20 for the two LPs for DD (left), SD (half-left), DS (half-right) and SS (right).

Since the timestamp differences in DD toggle between $\delta = 0$ and $\delta = 9 = 8 + 1$, this deterministic behavior represents a neutral case for all methods, e.g. method M repeatedly overestimates and underestimates the next timestamp and thus cannot gain over TW in the long run. Overall execution time grows, however, since forecasting intrudes the simulation engine (stalls CPU cycles). The first quartuple of lines in Table 28.5 explains the order of intrusion induced by the various methods. In the case SD (second quartuple of lines in Table 28.5), method A finds the highest chances to avoid rollbacks and can outperform U. For DS, method M finds an absolute stress case, yielding a slowdown as compared to U. If the arrival process has two stochastic components (SS) both M and A can outperform U. The most important observation from Table 28.5 is, that all the methods are able to increase the percentage of overall execution time spent for simulating events over the communication overhead induced. Thus the optimism control schemes are even more promising for distributed memory environments, for which the communication/computation speed ratio is smaller than on the CM-5, e.g. a cluster of RISC workstations.

The main drawback of the forecast schemes M, S and A are that they cannot cope with transient "patterns" of arrivals, but do respect only a central tendency of timestamp increments. Arrival patterns that show certain regularities, or at least some correlation in the time increments, can yield to stress cases as was seen above. Therefore forecast methods able to identify correlations and to predict next events at the maximum likelihood of those correlations are demanded.

ARIMA Forecasts

In this section we follow the idea of considering the arrival process as an unknown stochastic processes $\{X_t\} = (X_1, X_2, \ldots X_n)$, where $X_1, X_2, \ldots X_n$ are a series of instances of a random variable. Specifically $X_t = \delta_t - \overline{\delta}$ are the empirically observed timestamp differences, transformed by the series mean $\overline{\delta}$. If $\{X_t\}$ are statistically dependent variables, the arrival process can be modeled by an integrated autoregressive moving average process ARIMA$[p, d, q]$ (see e.g. [10])

$$\phi(B)\nabla^d X_t = \theta(B)\epsilon_t \tag{28.1}$$

where $\nabla^d = (1 - B)^d$ is the d-fold differencing operator, and B the backward shift operator defined by $B^i X_t = X_{t-i} \quad i = 0, \pm 1, \pm 2, \ldots$. This means that for e.g. $d = 2$, the process $Y_t = \nabla^2 X_t = X_t - 2X_{t-1} + X_{t-2}$ is assumed to be a *stationary* ARMA$[p, q]$ (=ARIMA$[p, 0, q]$), composed by a pure autoregressive process of order p (AR$[p]$) explaining Y_t as a dependency $Y_t = \phi_1 Y_{t-1} + \phi_2 Y_{t-2} + \ldots + \phi_p Y_{t-p} + \epsilon_t$, ϵ_t being a white noise random error, and a pure moving average process of order q (MA$[q]$) that explains Y_t as a series of i.i.d. white noise errors $Y_t = \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + \ldots + \theta_q \epsilon_{t-q}$ with $\mathrm{E}(\epsilon_i) = 0$, $\mathrm{Var}(\epsilon_i) = \sigma_\epsilon^2$ and $\mathrm{E}(Y_t) = 0$.

Looking at the arrival process as obtained on the CM-5 for LP_2 of the SPN in Figure 24.7 (Figure 28.20 top) and the corresponding autocorrelation (ACF) and partial ACF (Figure 28.20 middle and bottom), we find high positive correlation after every fourth lag for DD (Figure 28.20, left) obviously due to the four tokens in the SPN. The case SD (Figure 28.20, half-left) leads us to hypothesize that the arrival process is AR, since we find ACF dying down in an oscillating damped exponential fashion. This is also intuitive because the deterministic component in the process ($\tau(\mathrm{T2}) = 8$) dominates the stochastic one ($\tau(\mathrm{T1}) \sim exp(1)$). DS (Figure 28.20, half-right) gives evidence for a suitable representation of the the arrivals as a MA process, because ACF has a single spike at lag 1, and PACF dies down, etc.

For the automated characterization of the arrival process as an ARIMA$[p, d, q]$ process, the classical Box-Jenkins procedure can be adapted:

1. **Model Order Identification** First, the order of the ARIMA model, $[p, d, q]$ is identified. Since the theoretical partial autocorrelations $\varrho_{k-1}(k)$ to the lag k vanish after some $k > p$ for a pure AR$[p]$, the order of such a process can be approximated from the empirical partial autocorrelations $r_{k-1}(k)$. Similarly, for a pure MA$[q]$, the theoretical autocorrelations $\varrho_(k)$

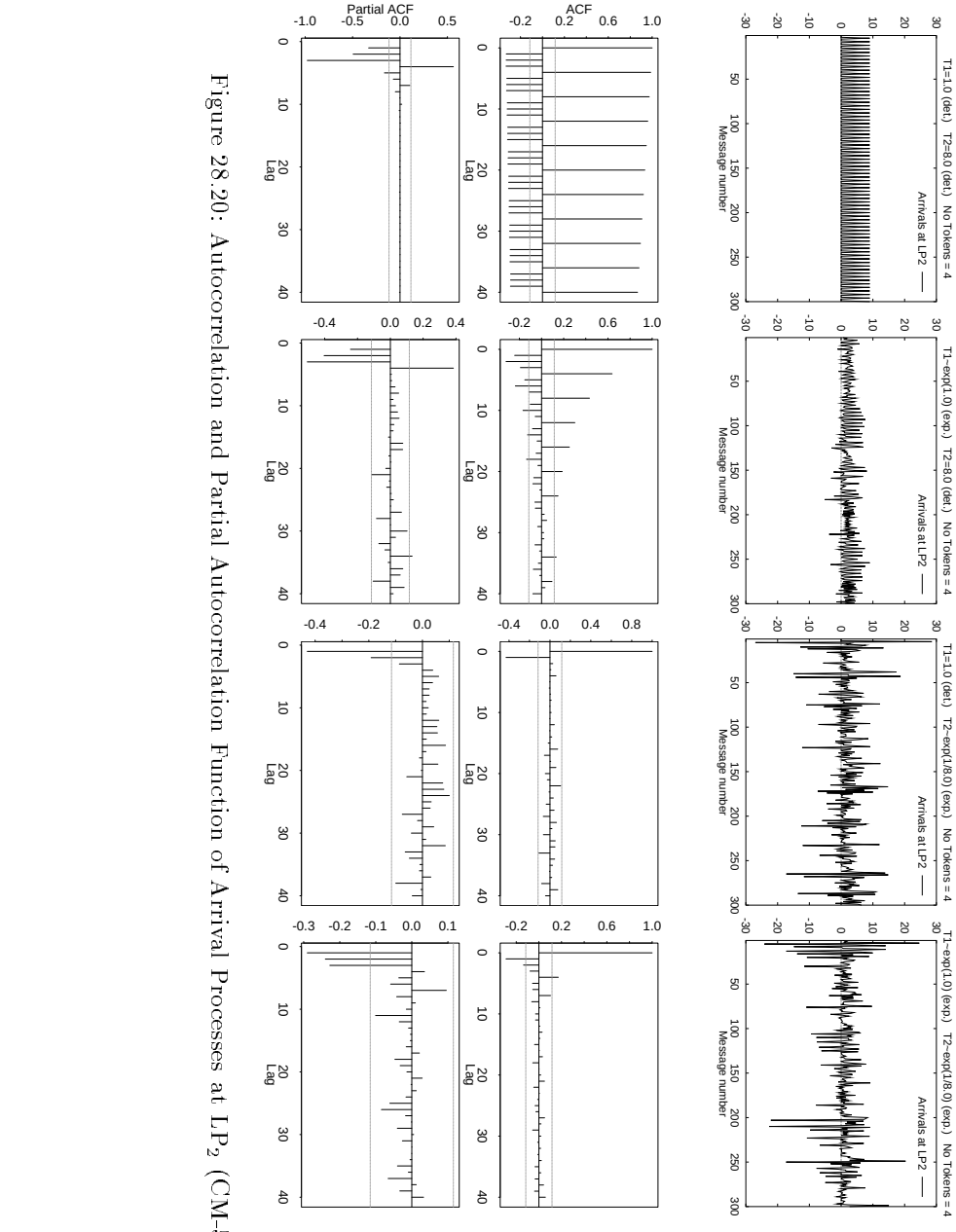


Figure 28.20: Autocorrelation and Partial Autocorrelation Function of Arrival Processes at LP_2 (CM-5)

vanish after some $k > q$, such that again the empirical data (autocorrelations $r(k)$) can be used to approximate the order. For a combined ARMA$[p, q]$ process, the Akaike-criterion, i.e. the combination (p, q) that minimizes $AIC(p, q) = \log \widehat{\sigma}^2_{p,q} + \frac{2}{n}(p + q)$ approximates the order. ($\widehat{\sigma}^2_{p,q}$ is a Maximum-Likelihood estimate of the variances σ^2_ϵ of the underlying white noise error.) The Akaike-criterion has a more general form for ARIMA$[p, d, q]$ processes. Indeed, as intuitively recognized, we find best order fittings e.g. for SD as ARIMA$[3, 0, 0]$ or for DS as ARIMA$[0, 0, 4]$.

2. **Model Parameter Estimation** In the next step, the parameters in (1) ($\phi_1, \ldots, \phi_p$ and $\theta_1, \ldots, \theta_q$) are determined as maximum likelihood estimates from the empirical data, i.e. the estimates $\hat{\phi}_1, \ldots, \hat{\phi}_p$ and $\hat{\theta}_1, \ldots, \hat{\theta}_q$ that minimize the square sum $S^2(\hat{\phi}_1, \ldots, \hat{\phi}_p, \hat{\theta}_1, \ldots, \hat{\theta}_q) = \sum_{t=-\infty}^{n} \tilde{\varepsilon}_t^2$ of the residuals $\tilde{\varepsilon}_t = \delta_t - \hat{\phi}_1 \delta_{t-1} - \ldots - \hat{\phi}_p \delta_{t-p} - \hat{\theta}_1 \tilde{\varepsilon}_{t-1} - \ldots - \hat{\theta}_q \tilde{\varepsilon}_{t-q}$ are used as the model parameters.

3. **Model Diagnostics/Verification** A well know method to validate the model with the estimates $\hat{\phi}_1, \ldots, \hat{\phi}_p$ and $\hat{\theta}_1, \ldots, \hat{\theta}_q$ is the Portmonteau lack-of-fit-test, which tests whether the residuals $\tilde{\varepsilon}_t$ are realizations of a white noise process. The confidence level $(1 - \alpha)$ of the test can be used as a measure to quantify the "trust" in the model and, as a consequence, in the forecast.

4. **Forecast** Finally, the (recursive) Durbin-Levinson method provides an algorithm for the one-step (or k-step) best linear prediction for $\widehat{X}_{t+1}$.

At a confidence level $\zeta = (1 - \alpha)$, the PADOC simulation engine (Figure 28.18, in Step 2.4) executes the next scheduled a transition firing with probability

$$P_\zeta[\text{exec first(EVL)}] = 1 - (1 + e^{-(\frac{LVT - \widehat{ts}}{\zeta(1-\zeta)100})})^{-1}, \tag{28.2}$$

otherwise the CPU is blocked for $\overline{s}$ time units.

Figure 28.21 explains the blocking probability (28.2) related to the confidence level $\zeta = (1 - \alpha)$: The higher the confidence ζ, the steeper the ascent of the delay probability as LVT progresses towards $\widehat{ts}$. (Steepness of the sigmoid function in Figure 28.21 (left) with $\zeta = 0.95$ is higher than in Figure 28.21 (right) $\zeta = 0.90$). Note also that after LVT progression in LP$_j$ has surpassed the estimate $\widehat{ts}$ (Figure 28.21, right), delays become more and more probable, expressing the increasing rollback hazard the LP runs into. The performance

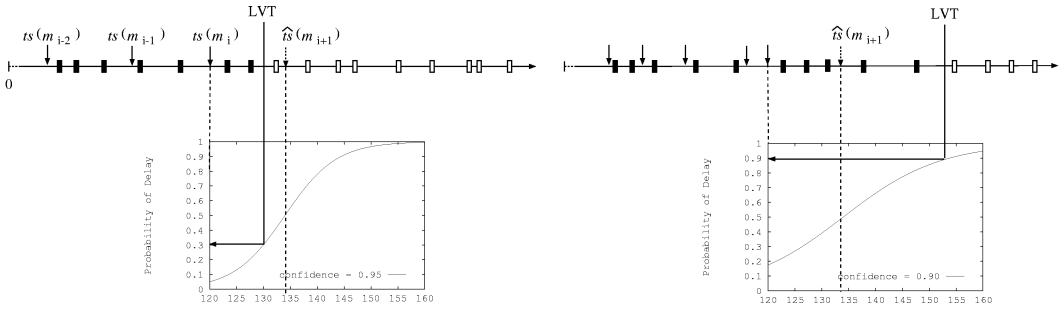


Figure 28.21: Probabilistic Direct Optimism Control with ARIMA Forecasting

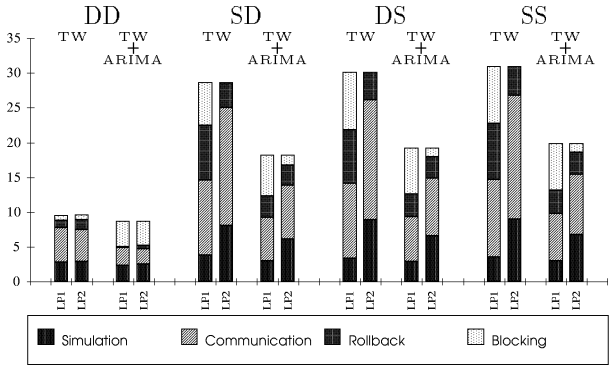


Figure 28.22: TW and TW with ARIMA forcasting for RS6K Cluster

gain of PADOC for the example in a distributed memory environment with comparably high communication latencies is illustrated in Figure 28.22.

A general observation is that with $\zeta \approx 1$, PADOC imposes a synchronization behavior close to CMB, whereas with $\zeta \approx 0$, optimism is as unlimited as in (plain) TW. Moreover, by periodically rebuilding the ARIMA model, the PADOC scheme adapts the LP to a synchronization behavior directly reflecting the inherent model parallelism, and also copes with *transient* arrival processes.

To demonstrate this by example, we study PN4 (Figure 28.23), with $\tau_1 = \tau_2 = \ldots = \tau_8 = 0.0$ and $\tau_{10} = 10.0$, $\tau_{12} = 1.0$, while $\tau_9 \sim exp(1.0)$ and $\tau_{11} \sim exp(0.1)$. Let the net be spatially decomposed into two LPs where LP_1 (LP_2) holds the transitions $\{t_1, t_2, t_3, t_4, t_9, t_{10}\}$ ($\{t_5, t_6, t_7, t_8, t_{11}, t_{12}\}$) and the respective input places.

Since TW performance is known to degrade in case of imbalancements in the local virtual time (LVT) progression of different LPs. In the example we

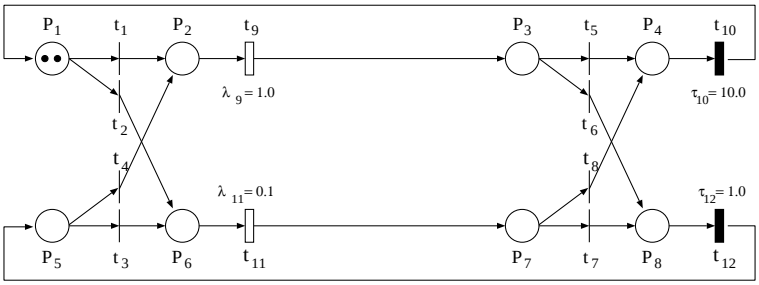


Figure 28.23: PN4: A PN with virtual time progression in phases

have intentionally constructed imbalancement by having LP_2 progress LVT per simulation step at a tenfold higher rate than LP_1 if transitions fire along the T-invariant $x_6 = [1,0,0,0,1,0,0,0,1,1,0,0]$. Clearly, LP_1 will permanently force LP_2 to rollback, thus causing overhead. If the net executes along $x_3 = [0,0,1,0,0,0,1,0,0,0,1,1]$, the opposite is the case, with the only difference that the dominating timing component is not deterministic ($t_1 0$), but of stochastic nature ($t_1 1$). We introduce a switching probability for the conflicting transitions (t_1, t_2), (t_3, t_4), etc. to be able to control the switching of tokens from cycle C_1 defined by x_6 to C_2 defined by x_3 and vice versa. $p = 50$ e.g. means that for a token at any decision point it is equally likely to switch the cycle or to stay on the current one, whereas $p = 95$ expresses just a 5 % "willingness" of tokens to switch cycles. For an initial marking $\sum_{i=1}^{\mathcal{P}} \mu^{(0)}(i) = 2$ the LVT progression in the LPs reveals three different *phases*: 2 tokens in C_1 (phase 1), 1 token in each cylce C_1 and C_2 (phase 2), and 2 tokens in C_2 (phase 3), thus naturally representing a *stress case* for TW in general, and adaptive TW protocols with optimism control parameters based on averages (ATW, LAP, AMM) in particular.

The LVT progression traces as obtained for PN4 (Figure 28.23) with $p = 95$ on the CM-5 are depicted in Figure 28.24. It is seen that plain TW considerably suffers from the (artificial) straggler bulks, yielding an oscillating LVT progression. TW with tokentime forecasts based on the arithmetic mean can cope with this destructive situation to some extent, but is still prone to pathological behavior. The TW+ARIMA approach, however, gains from consistent forecasts and reveals the best adaptation.

The major strength of the PADOC protocol is that the optimism of Time Warp can be automatically controlled by the model parallelism available as expressed by the likelihood of future messages, and can even adapt to a transient

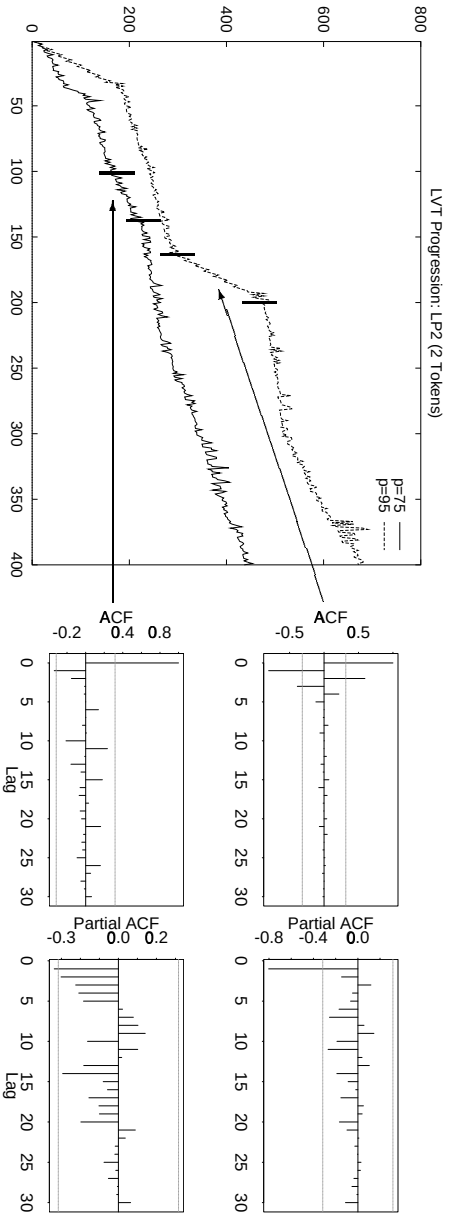


Figure 28.24: Message No vs. message time stamp as observed on CM-5; ARIMA characteristics

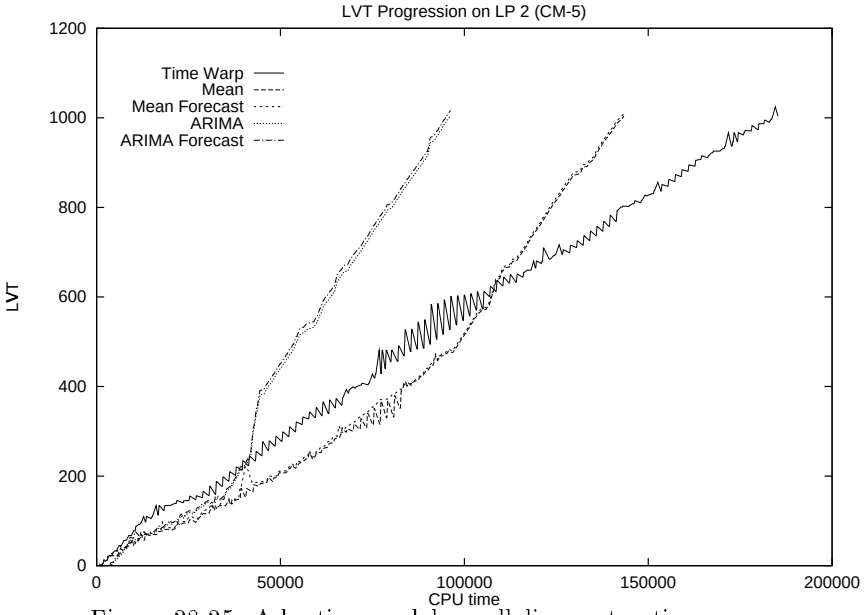


Figure 28.25: Adaptive model parallelism extraction

behavior of the simulated model. The *asymptotic* behavior of the protocol for simulation models with $P[e \rightarrow e'] \simeq 1$ (for all pairs (e, e')) is close to CMB, while for models with $P[e \rightarrow e'] \simeq 0$ it is arbitrarily close to (a throttled) Time Warp. The PADOC mechanism gains adaptiveness in the sense that, independent of the ratio of the communication and computation speed of the target platform, the synchronisation policy is adjusted automatically to that point in the continuum between TW and CMB protocols, that is most appropriate for the parallelism inherent in the simulation model.

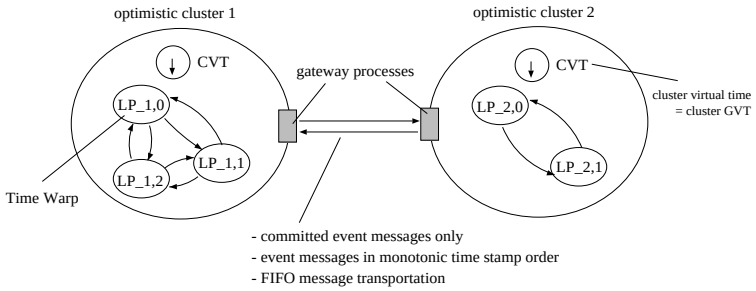
28.6.3 Other Adaptive Approaches

- window-based / penalty-based throttling [Reiher/Jefferson 89]
- adaptive blocking windows in TW [Ball/Hoyt 90]
- dynamically sizing checkpointing intervals [Palaniswamy/Wilsey 93]
- explicit rollback cost expectation function [Ferscha/Lüthi 94]
- *indirect* optimism control in TW via available memory [Das/Fujimoto 94]
- local adaptive optimism control in TW [Hamnes/Tripathi 94]

- adaptive LP scheduling [Palaniswamy/Wilsey 94]
- dynamically switching between lazy and aggressive cancellation [Rajan/Wilsey 95]
- probabilisitic adaptive *direct* optimism control in TW [Ferscha 95]
- near-perfect state information adaptive synchronization [Srinivasan/Reynolds 95]
- optimistic fossil collection [Young/Wilsey 96]

28.7 Hybrid Protocols

- Hierarchical Distributed Simulation [Prakash/Ramamoorthy 88]
- Hierarchical PDES in Composite ELSA [Arvind/Smart 92]
- Local Time Warp [Rajaei/Ayani/Thorelli 93]



- Clustered Time Warp [Arvind/Tropper 95]
  (Time Warp between LP clusters, sequential within clusters)

Bibliography

[1] M. Ajmone Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The Effect of Execution Policies on the Semantics and Analysis of Stochastic Petri Nets. *IEEE Transactions on Software Engineering*, 15(7):832–846, July 1989.

[2] I. F. Akyildiz, L. Chen, R. Das, R. M. Fujimoto, and R. F. Serfozo. The Effect of Memory Capacity on Time Warp Performance. *Journal of Parallel and Distributed Computing*, 18(4):411–422, August 1993.

[3] Ch. Alexopoulos. A Review of Advanced Methods for Simulation Output Analysis. In J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, editors, *Proceedings of the 1994 Winter Simulation Conference*, pages 133–140, 1994.

[4] Ch. Alexopoulos, G. S. Fishmann, and A. F. Seila. Computational Experience with the Batch Means Method. In S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, editors, *Proceedings of the 1997 Winter Simulation Conference*, pages 194–201, 1997.

[5] H. H. Ammar and S. Deng. Time Warp Simulation of Stochastic Petri Nets. In *Proc. 4th Intern. Workshop on Petri Nets and Performance Models*, pages 186 – 195. IEEE-CS Press, 1991.

[6] F. Baccelli and M. Canales. Parallel Simulation of Stochastic Petri Nets Using Recurrence Equations. *acm Transactions on Modeling and Computer Simulation*, 3(1):20 – 41, January 1993.

[7] D. Baik and B. P. Zeigler. Performance Evaluation of Hierarchical Distributed Simulators. In *Proc. of the 1985 Winter Simulation Conference*, pages 421 – 427. SCS, 1985.

[8] W. L. Bain and D. S. Scott. An Algorithm for Time Synchronization in Distributed Discrete Event Simulation. In B. Unger and D. Jefferson, editors, *Proceedings of the SCS Multiconference on Distributed Simulation, 19 (3)*, pages 30–33. SCS, February 1988.

[9] D. Ball and S. Hoyt. The Adaptive Time-Warp Concurrency Control Algorithm. In D. Nicol, editor, *Distributed Simulation. Proceedings of the SCS Multiconference on Distributed Simulation*, pages 174 – 177, San Diego, California, 1990. Society for Computer Simulation. Simulation Series, Vol. 22, No. 1.

[10] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods.* Springer Verlag, New York, 1991.

[11] R. E. Bryant. A Switch-Level Model and Simulator for MOS Digital Systems. *IEEE Transactions on Computers*, C-33(2):160–177, February 1984.

[12] W. Cai and St. J. Turner. An Algorithm for Distributed Discrete-Event Simulation - The 'Carrier Null Message' Approach. In *Proceedings of the SCS Multiconference on Distributed Simulation Vol. 22 (1)*, pages 3–8. SCS, January 1990.

[13] K. M. Chandy and J. Misra. Distributed Simulation: A Case Study in Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, September 1979.

[14] K. M. Chandy and J. Misra. Asynchronous Distributed Simulation via a Sequence of Parallel Computations. *Communications of the ACM*, 24(11):198–206, November 1981.

[15] K. M. Chandy, J. Misra, and L. M. Haas. Distributed Deadlock Detection. *ACM Transactions On Computer Systems*, 1(2):144–156, May 1983.

[16] R. Cheng. Selecting Input Models. In J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, editors, *Proceedings of the 1994 Winter Simulation Conference*, pages 184–191, 1994.

[17] G. Chiola and A. Ferscha. Distributed Simulation of Petri Nets. *IEEE Parallel and Distributed Technology*, 1(3):33 – 50, August 1993.

[18] G. Chiola and A. Ferscha. Distributed Simulation of Timed Petri Nets: Exploiting the Net Structure to Obtain Efficiency. In M. Ajmone Marsan, editor, *Proc. of the 14^{th} Int. Conf. on Application and Theory of Petri Nets*

1993, Chicago, June 1993, Lecture Notes in Computer Science 691, pages 146 – 165, Berlin, 1993. Springer Verlag.

[19] G. Chiola and A. Ferscha. Performance Comparable Design of Efficient Synchronization Protocols for Distributed Simulation. In *Proc. of MASCOTS'95*, pages 343 – 348. IEEE Computer Society Press, 1995.

[20] G. Ciardo, R. German, and Ch. Lindemann. A Characterization of the Stochastic Process Underlying a Stochastic Petri Net. In *Proc. of the 5th International Workshop on Petri Nets and Performance Models, October 19-22, 1993, Toulouse, France.* IEEE Computer Society Press, 1993.

[21] A. I. Concepcion. Mapping Distributed Simulators onto the Hierarchical Multibus Multiprocessor Architecture. In P. Reynolds, editor, *Proc. of the SCS Multiconference on Distributed Simulation*, pages 8–13. Society for Computer Simulation, 1985.

[22] H. Damerdji, S. G. Henderson, and P. W. Glynn. Computational Efficiency Evaluation in Output Analysis. In S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, editors, *Proceedings of the 1997 Winter Simulation Conference*, pages 208–215, 1997.

[23] S. R. Das and R. M. Fujimoto. An Adaptive Memory Management Protocol for Time Warp Parallel Simulation. In *Proc. of the 1994 ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems, Nashville, 1994*, pages 201–210. ACM, 1994.

[24] Samir Ranjan Das and Richard Fujimoto. Adaptive Memory Management and Optimism Control in Time Warp. *ACM Transactions on Modeling and Computer Simulation*, 7(2):239–271, April 1997.

[25] R. David and H. Alla. *Petri Nets & Grafcet. Tools for Modelling Discrete Event Systems.* Prentice Hall, New York, 1992.

[26] Ph. M. Dickens and P. F. Reynolds. SRADS with Local Rollback. In *Proceedings of the SCS Multiconference on Distributed Simulation Vol. 22 (1)*, pages 161–164. SCS, January 1990.

[27] R. E. Felderman and L. Kleinrock. An Upper Bound on the Improvement of Asynchronous versus Synchronous Distributed Processing. In D. Nicol, editor, *Proc. of the SCS Multiconf. on Dist. Sim.*, volume 22, pages 131 – 136, Jan 1990.

[28] A. Ferscha. Probabilistic Adaptive Direct Optimism Control in Time Warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95)*, pages 120 – 129, 1995.

[29] A. Ferscha. Parallel and Distributed Simulation of Discrete Event Systems. In A. Y. Zomaya, editor, *Parallel and Distributed Computing Handbook*, pages 1003 – 1041. McGraw-Hill, 1996.

[30] A. Ferscha and G. Chiola. Self Adaptive Logical Processes: The Probabilistic Distributed Simulation Protocol. In *Proc. of the 27^{th} Annual Simulation Symposium*, pages 78–88, Los Alamitos, California, 1994. IEEE Computer Society Press.

[31] A. Ferscha and G. Chiola. Self Adaptive Logical Processes: The Probabilistic Distributed Simulation Protocol. In *Proc. of the 27^{th} Annual Simulation Symposium*, pages 78–88, Los Alamitos, California, 1994. IEEE Computer Society Press.

[32] A. Ferscha and J. Lüthi. Estimating Rollback Overhead for Optimism Control in Time Warp. In *Proc. of the 28^{th} Annual Simulation Symposium*, pages 2–12, Los Alamitos, California, 1995. IEEE Computer Society Press.

[33] R. M. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):30–53, October 1990.

[34] A. Gafni. Rollback Mechanisms for Optimistic Distributed Simulation Systems. In *Proc. of the SCS Multiconference on Distributed Simulation 19*, pages 61–67, 1988.

[35] A. Gafni. Rollback Mechanisms for Optimistic Distributed Simulation Systems. In B. Unger and D. Jefferson, editors, *Proceedings of the SCS Multiconference on Distributed Simulation, 19 (3)*, pages 61–67. SCS, February 1988.

[36] D. Goldsman and B. W. Schmeiser. Computational Efficiency of Batching Methods. In S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, editors, *Proceedings of the 1997 Winter Simulation Conference*, pages 202–207, 1997.

[37] A. G. Greenberg, B. D. Lubachevsky, and I. Mitrani. Unboundedly Parallel Simulations Via Recurrence Relations. *ACM*, pages 1–12, 1990.

[38] A. G. Greenberg, B. D. Lubachevsky, and I. Mitrani. Algorithms for Unboundedly Parallel Simulations. *ACM Transactions on Computer Systems*, 9(3):201 – 221, Aug 1991.

[39] Albert G. Greenberg, Boris D. Lubachevsky, and Isi Mitrani. Superfast Parallel Discrete Event Simulations. *ACM Transactions on Modeling and Computer Simulation*, 6(2):107–136, April 1996.

[40] B. Groselj and C. Tropper. The Time-of-next-event Algorithm. In B. Unger and D. Jefferson, editors, *Proceedings of the SCS Multiconference on Distributed Simulation, 19 (3)*, pages 25–29. SCS, February 1988.

[41] Donald O. Hamnes and Anand Tripathi. Investigations in Adaptive Distributed Simulation. In D. K. Arvind, Rajive Bagrodia, and Jason Yi-Bing Lin, editors, *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94)*, pages 20–23, July 1994.

[42] J. Hooper. Strategy-related Characteristics of Discrete Event Languages and Models. *Simulation*, 46(4):153–159, 1986.

[43] Raj Jain. *The Art of Computer System Performance Analysis. Techniques for Experimental Design, Measurement, Simulaiton, and Modeling.* John Wiley and Sons, Inc., 1991.

[44] D. Jefferson. Virtual Time II: The Cancelback Protocol for Storage Management in Time Warp. In *Proc. of the 9th Annual ACM Symposium on Principles of Distributed Computing*, pages 75–90, ACM, New York, 1990.

[45] D. Jefferson and P. Reiher. Supercritical Speedup. In A. H. Rutan, editor, *Proceedings of the 24^{th} Annual Simulation Symposium, New Orleans, Louisiana, USA, April 1-5, 1991.*, pages 159–168. IEEE Computer Society Press, 1991.

[46] D. Jefferson and H. Sowizral. Fast Concurrent Simulation Using the Time Warp Mechanism. In P. Reynolds, editor, *Distributed Simulation 1985*, pages 63–69, La Jolla, California, 1985. SCS-The Society for Computer Simulation, Simulation Councils, Inc.

[47] D. A. Jefferson. Virtual Time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, July 1985.

[48] D. W. Kelton. Statistical Analysis of Simulation Output. In S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, editors, *Proceedings of the 1997 Winter Simulation Conference*, pages 23–30, 1997.

[49] L. Lamport. Time, Clocks, and the Ordering of Events in Distributed Systems. *Communications of the ACM*, 21(7):558 – 565, Jul 1978.

[50] P. L. L'Ecuyer. Efficiency Improvement and Variance Reduction. In J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, editors, *Proceedings of the 1994 Winter Simulation Conference*, pages 122–132, 1994.

[51] L. Leemis. Seven Habits of Highly Successful Input Modelers. In S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, editors, *Proceedings of the 1997 Winter Simulation Conference*, pages 39–46, 1997.

[52] Y-B. Lin, B. Preiss, W. Loucks, and E. Lazowska. Selecting the Checkpoint Interval in Time Warp Simulation. In R. Bagrodia and D. Jefferson, editors, *Proc. of the 7^{th} Workshop on Parallel and Distributed Simulation*, pages 3–10, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[53] Yi-Bing Lin and Bruno R. Preiss. Optimal Memory Management for Time Warp Parallel Simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(4):283–307, October 1991.

[54] B. D. Lubachevsky. Bounded Lag Distributed Discrete Event Simulation. In B. Unger and D. Jefferson, editors, *Proceedings of the SCS Multiconference on Distributed Simulation, 19 (3)*, pages 183–191. SCS, February 1988.

[55] B. D. Lubachevsky, A. Weiss, and A. Shwartz. An Analysis of Rollback-Based Simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(2):154–193, April 1991.

[56] F. Mattern. Algorithms for distributed termination detection. *Distributed Computing*, 2:161–175, 1987.

[57] E. Mayr and A. Meyer. The Complexity of the Finite Containment Problem for Petri Nets. *Journal of the ACM*, 28(3):561 – 576, 1981.

[58] J. Misra. Distributed Discrete-Event Simulation. *ACM Computing Surveys*, 18(1):39–65, 1986.

[59] M. K. Molloy. Performance Analysis Using Stochastic Petri Nets. *IEEE Transactions on Computers*, C-31(9):913 – 917, September 1982.

[60] D. Nicol and W. Mao. Automated Parallelization of Timed Petri-Net Simulations. submitted for publication, 1994.

[61] D. M. Nicol. Parallel Discrete-Event Simulation OF FCFS Stochastic Queueing Networks. In *Proceedings of the ACM/SIGPLAN PPEALS 1988*, pages 124 – 137, 1988.

[62] D. M. Nicol. Performance Bounds on Parallel Self-initiating Discrete Event Simulations. *ACM Transactions on Modeling and Computer Simulation*, 1(1):24 – 50, Jan 1991.

[63] D. M. Nicol. Global Synchronization for Optimistic Parallel Discrete Event Simulation. In R. Bagrodia and D. Jefferson, editors, *Proc. of the 7^{th} Workshop on Parallel and Distributed Simulation*, pages 27–34, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[64] D. M. Nicol and R. M. Fujimoto. Parallel Simulation Today. *Operations Research*, 1994.

[65] D. M. Nicol and S. Roy. Parallel Simulation of Timed Petri-Nets. In B. Nelson, D. Kelton, and G. Clark, editors, *Proceedings of the 1991 Winter Simulation Conference*, pages 574 – 583, 1991.

[66] J. K. Peacock, J. W. Wong, and E. G. Manning. Distributed Simulation using a Network of Processors. *Computer Networks*, 3(1):44–56, 1979.

[67] A. Prakash and R. Subramanian. Filter: An Algorithm for Reducing Cascaded Rollbacks in Optimistic Distributed Simulation. In A. H. Rutan, editor, *Proceedings of the 24^{th} Annual Simulation Symposium, New Orleans, Louisiana, USA, April 1-5, 1991.*, pages 123–132. IEEE Computer Society Press, 1991.

[68] H. Rajaei, R. Ayani, and L. E. Thorelli. The Local Time Warp Approach to Parallel Simulation. In R. Bagrodia and D. Jefferson, editors, *Proc. of the 7^{th} Workshop on Parallel and Distributed Simulation*, pages 119–126, Los Alamitos, CA, 1993. IEEE Computer Society Press.

[69] Ch. Ramchandani. Analysis of Asynchronous Concurrent Systems by Petri Nets. Technical report, MIT, Laboratory of Computer Science, Cambridge, Massachusetts, February 1974.

[70] P. L. Reiher, R. M. Fujimoto, S. Bellenot, and D. Jefferson. Cancellation Strategies in Optimistic Execution Systems. In *Proceedings of the SCS Multiconference on Distributed Simulation Vol. 22 (1)*, pages 112–121. SCS, January 1990.

[71] H. Sellami, J. D. Allen, D.E. Schimmel, and S. Yalamanchili. Simulation of Marked Graphs on SIMD Architectures Using Efficient Memory Management. In *Proc. of MASCOTS'94*, pages 343 – 348. IEEE Computer Society Press, 1994.

[72] J. Sifakis. Use of Petri Nets for Performance Evaluation. In H. Beilner and E. Gelenbe, editors, *Measuring, modelling and evaluating computer systems*, pages 75 – 93. North-Holland, 1977.

[73] L. M. Sokol, D. P. Briscoe, and A. P. Wieland. MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution. In *Proc. of the SCS Multiconf. on Distributed Simulation*, pages 34 – 42, 1988.

[74] G. S. Thomas. Parallel Simulation of Petri Nets. Technical Report TR 91-05-05, Dep. of Computer Science, University of Washington, May 1991.

[75] G. S. Thomas and J. Zahorjan. Parallel Simulation of Performance Petri Nets: Extending the Domain of Parallel Simulation. In *Proc. of the 1991 Winter Simulation Conference*, 1991.

[76] K. Venkatesh, T. Radhakrishnan, and H. F. Li. Discrete Event Simulation in a Distributed System. In *IEEE COMPSAC*, pages 123 – 129. IEEE Computer Society Press, 1986.

[77] J. R. Wilson. Modeling Dependencies in Stochastic Simulation Inputs. In S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, editors, *Proceedings of the 1997 Winter Simulation Conference*, pages 47–52, 1997.