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:*

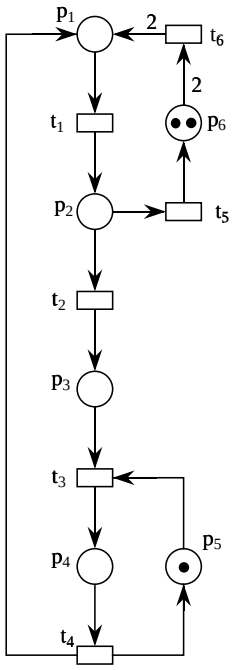
$$\mathcal{N} = \langle P, T, F, W \rangle$$

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.

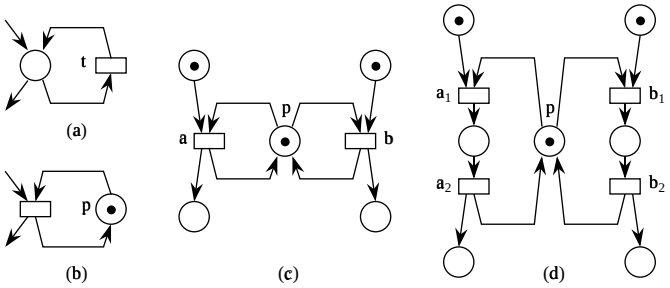


Figure 2.2: Self-loops.

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:

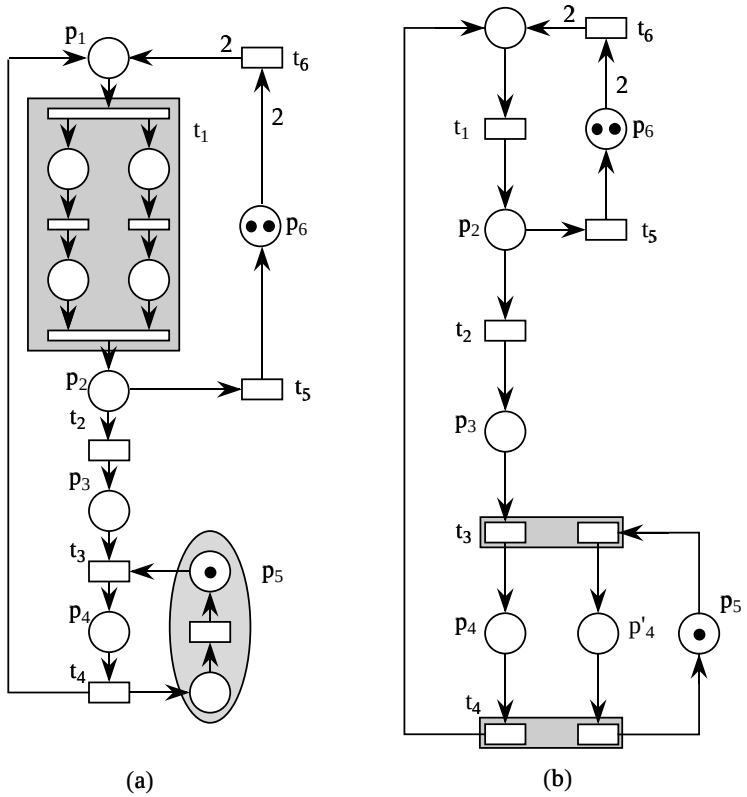


Figure 2.3: Synthesising the model of the multicomputer PLC of Example 2.1: (a) A refined model; (b) As a synchronisation of two sub-models.

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.

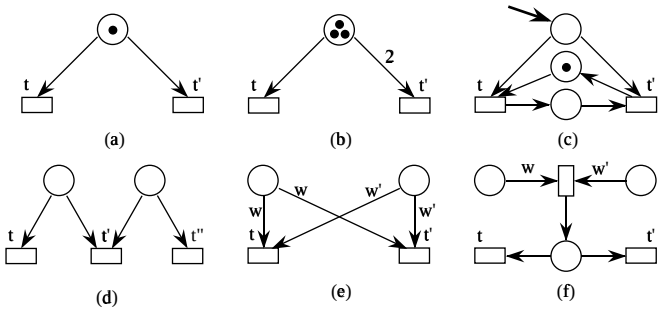


Figure 2.4: Conflicts and structural conflicts.

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'.

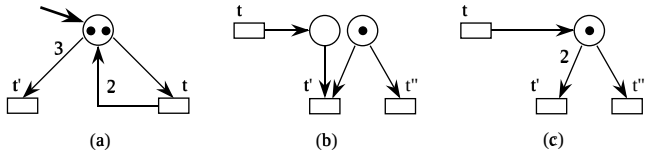


Figure 2.5: Non equal conflicts and confusion.

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.)

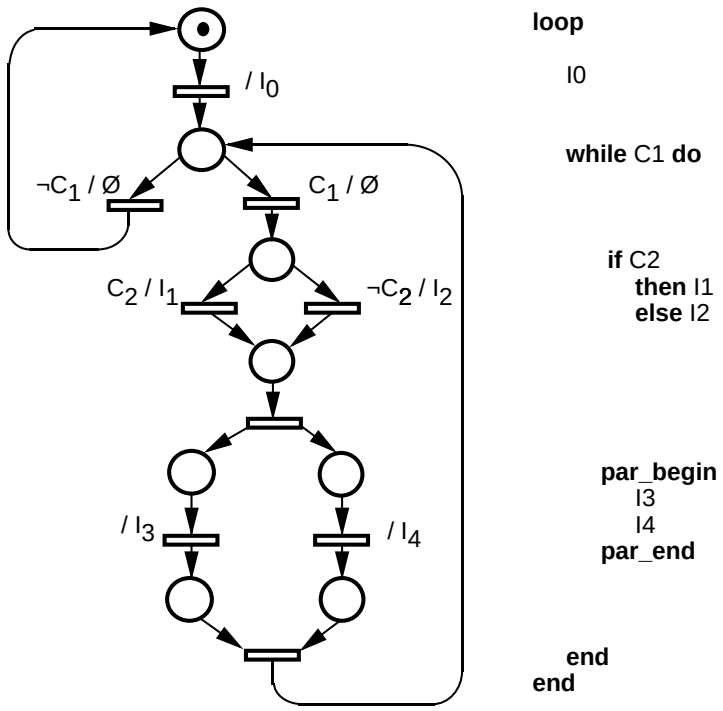


Figure 2.6: Control flow of a simple parallel-PASCAL-like program.

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

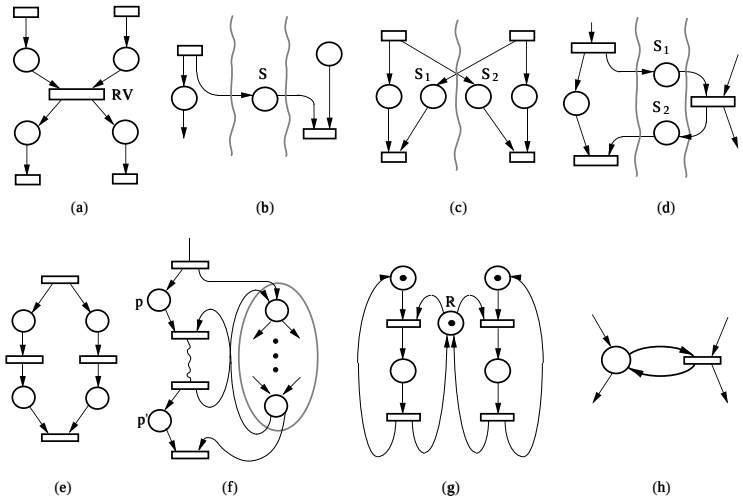


Figure 2.7: Conventional synchronisation schemas: (a) rendez-vous, (b) semaphore, (c) symmetric rendez-vous/semaphore, (d) asymmetric rendez-vous/semaphore (master/slave), (e) fork-join, (f) subprogram (p and p' must not be marked simultaneously), (g) mutex, (h) guard (condition reading).

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.

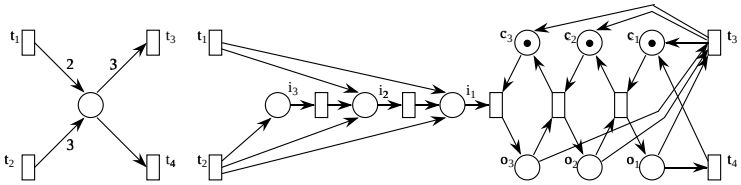


Figure 2.8: Ordinary implementation of a weighted net.

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.

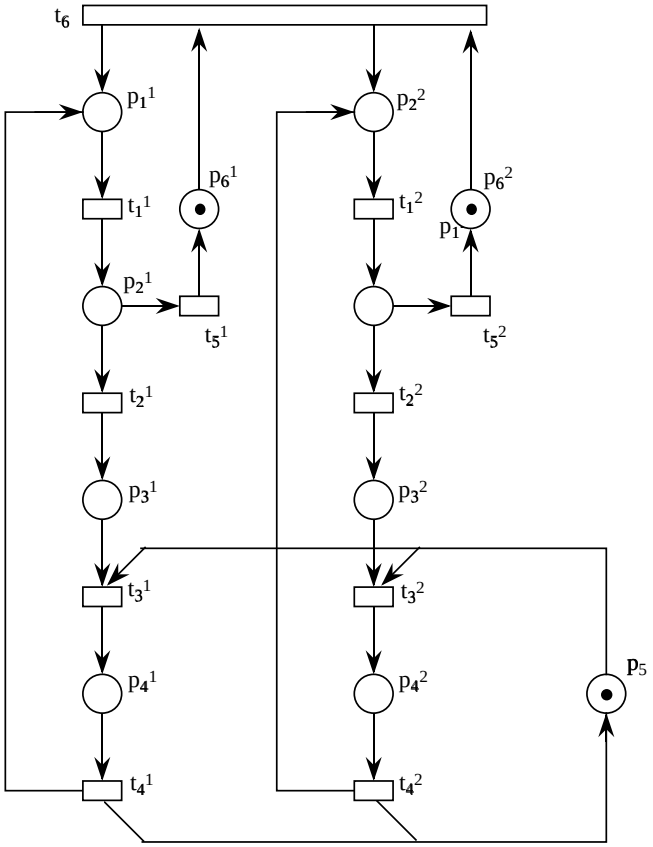


Figure 2.9: An EN model of the multicomputer PLC of Example 2.1.

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

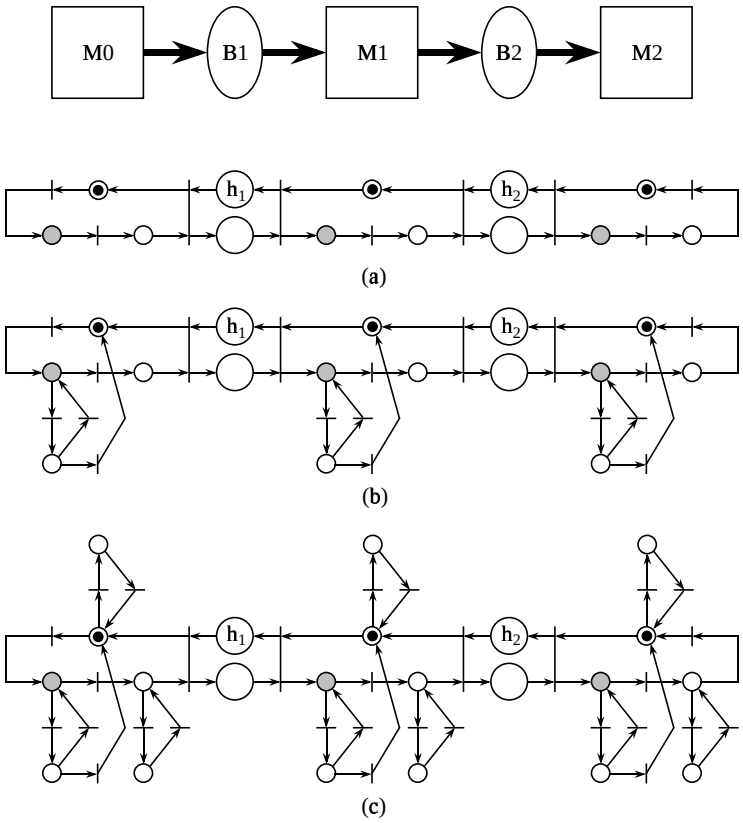


Figure 2.10: Modelling a flow line with three machines and two buffers. Each buffer is modelled by two places, for the parts and "holes", respectively (the later initially marked with h_i holes). Each machine is modelled by a state machine, initially idle, where the "working-state" is shaded; they follow 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.

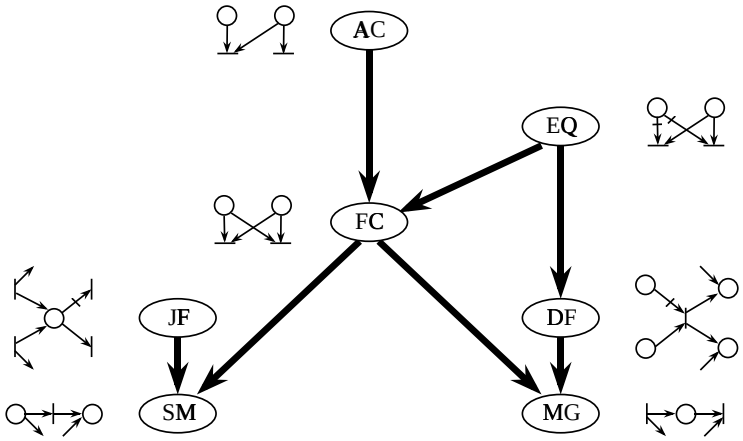


Figure 2.11: Relations between some basic syntactical subclasses.

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.

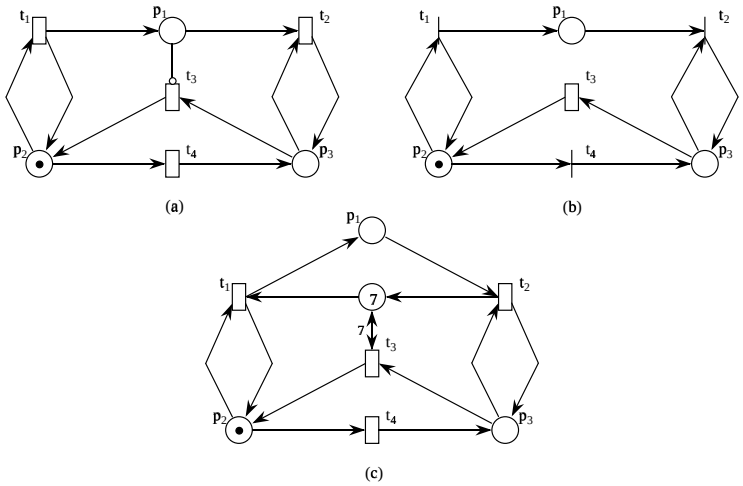


Figure 2.12: Three models of the lazy lad.

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.

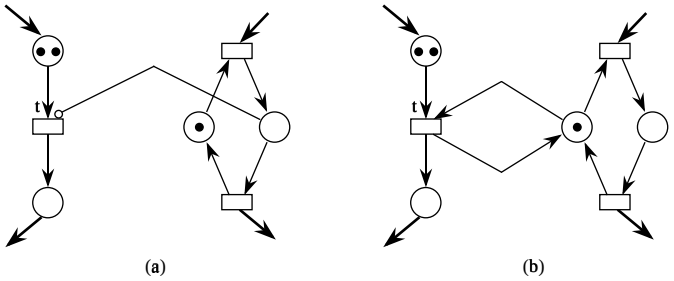


Figure 2.13: Inhibitor arcs and enabling degree. In (a) $\mathbf{e}(\mathbf{m})[t] = 2$, and in (b) $\mathbf{e}(\mathbf{m})[t] = 1$.

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.

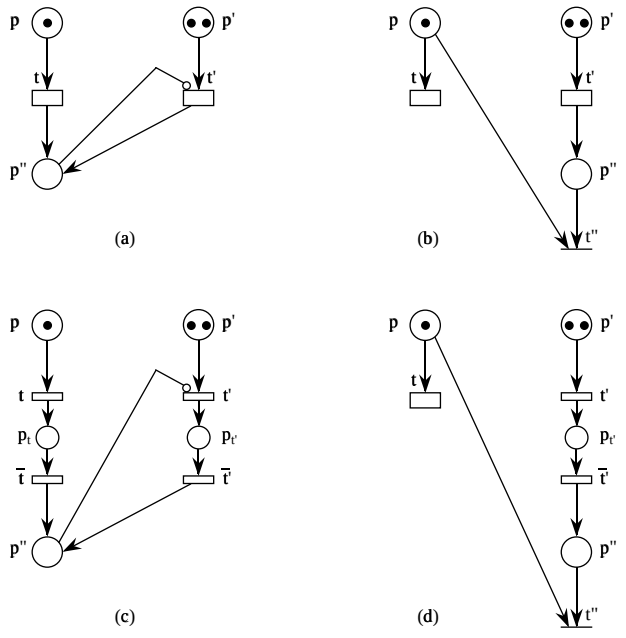


Figure 2.14: Inhibitor arcs, priorities, concurrency, and conflicts.

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.

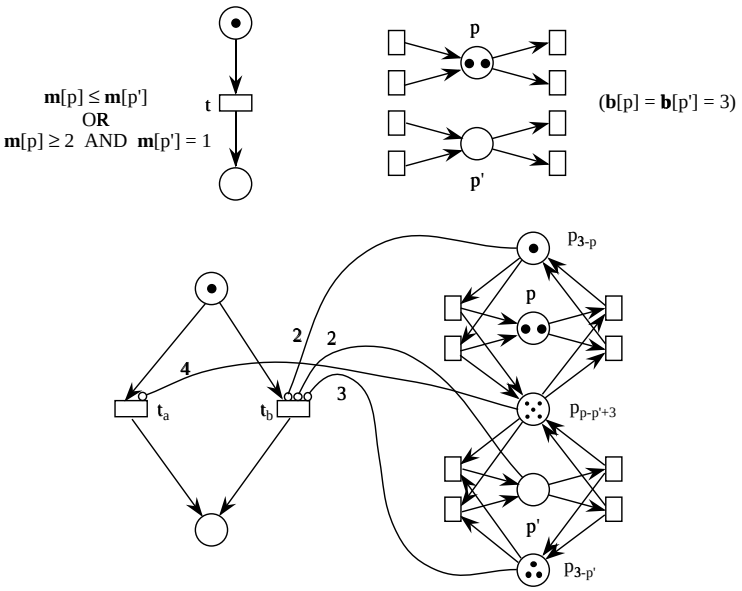


Figure 2.15: Implementing complex enabling conditions in bounded systems.

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).

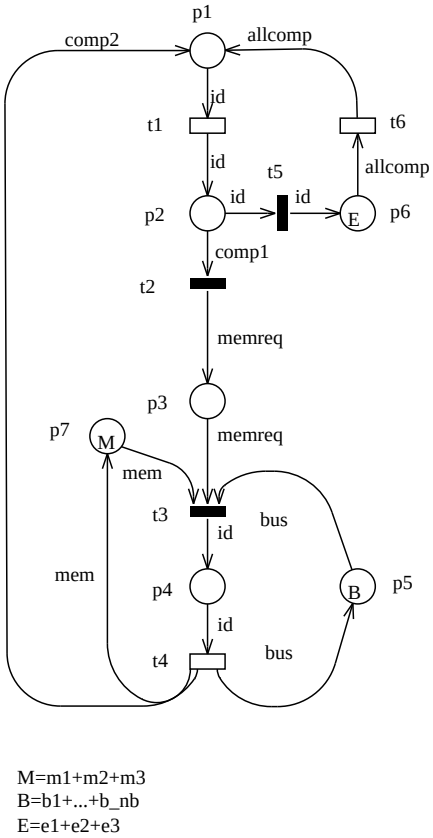


Figure 2.16: CPN model of a multicomputer PLC

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 $\mathbf{m}[p, c] = n$.

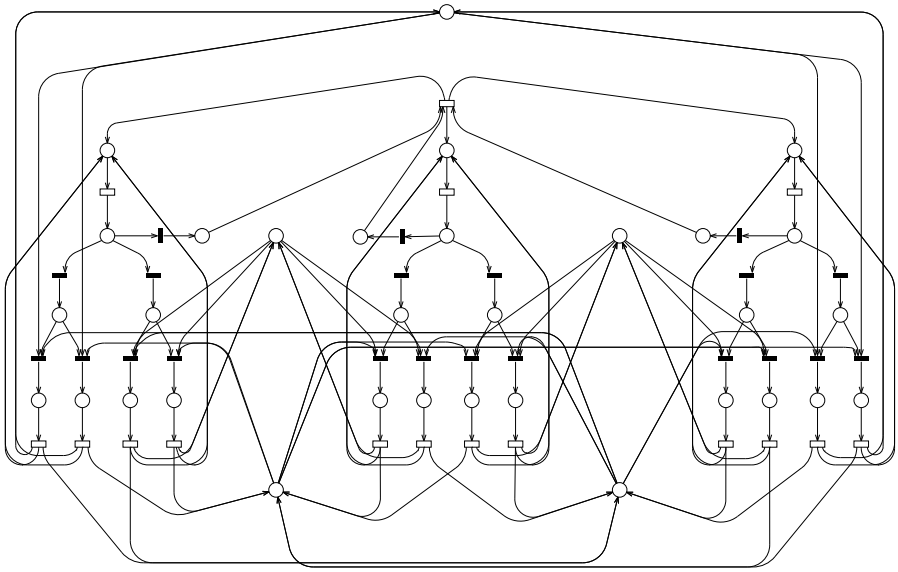


Figure 2.17: Unfolded model of a multicomputer PLC with three computers and two busses

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

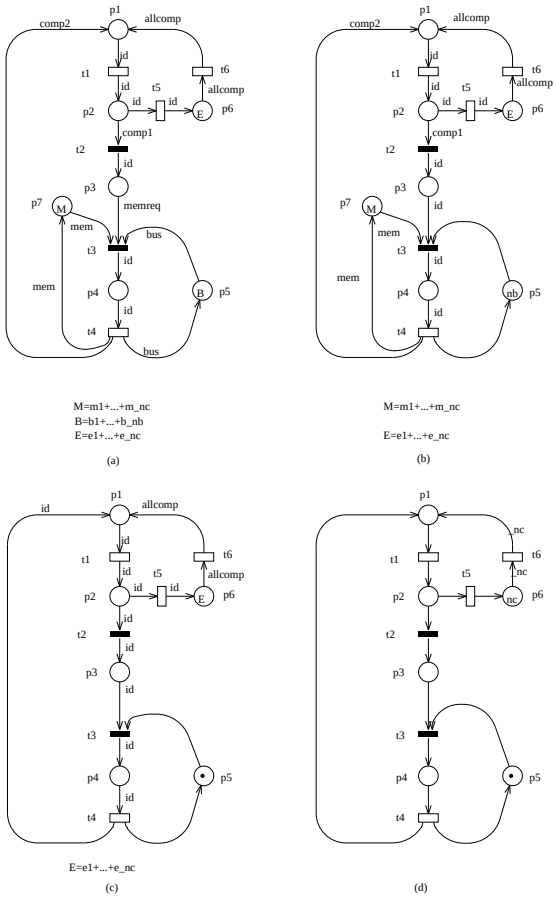
token game on the net (see Figure 2.18(b)). The second observation concerns the colour used to distinguish memories, in the particular case of $nb = 1$ (only one bus in the system): indeed in this situation, the place representing the memories not involved in any external access becomes *implicit* (i.e., 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.