Programación de Sistemas Concurrentes y Distribuidos 1ª Convocatoria curso 17/18

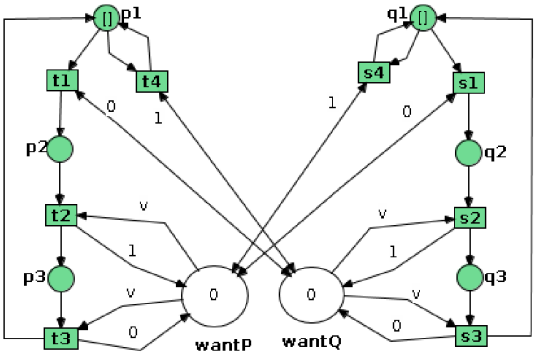
27 de enero de 2018

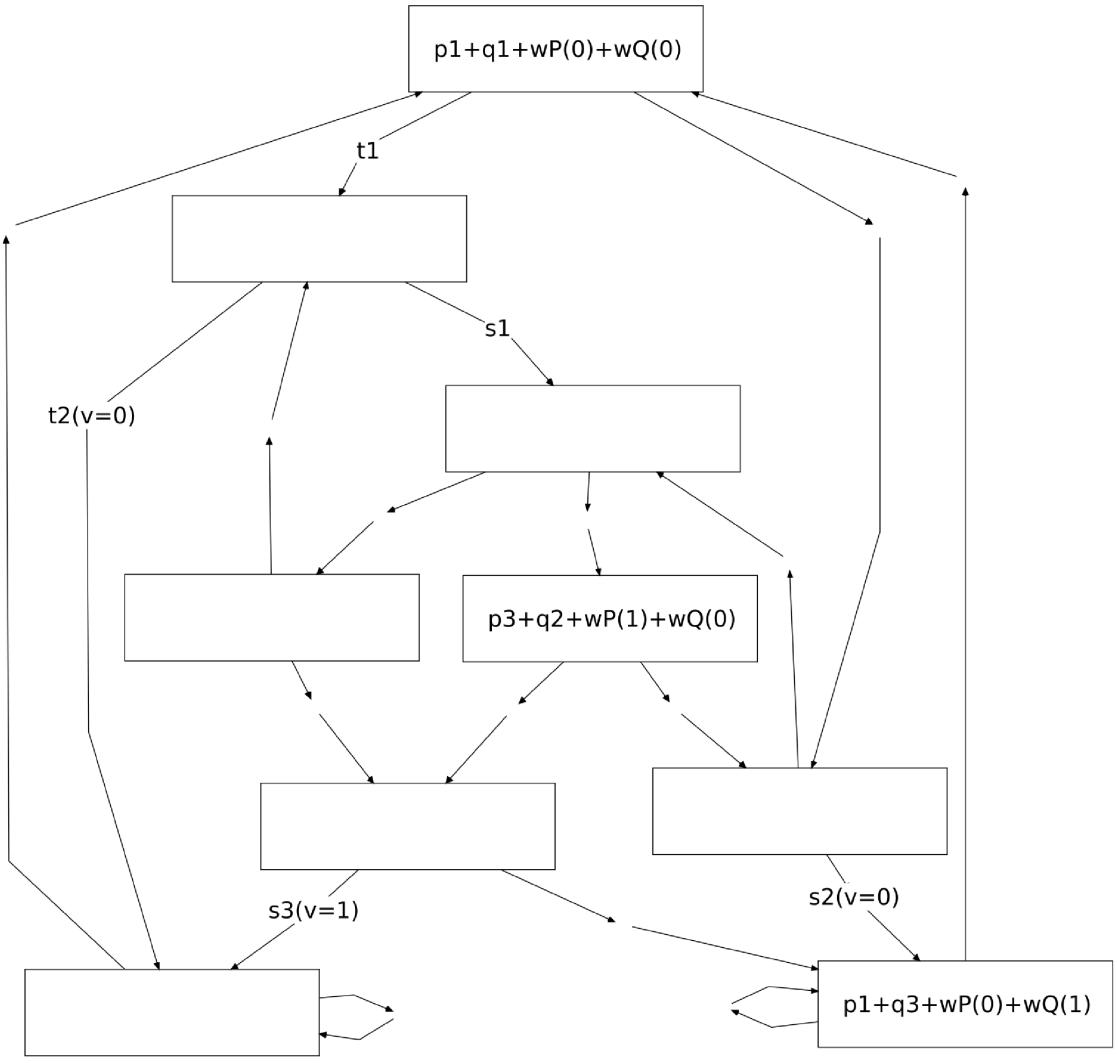
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza

Ejercicio 1 (1.5 ptos.)

Considérese el modelo que se muestra a continuación. El ejercicio pide:

- Completar, sobre la propia hoja del enunciado, el grafo de alcanzabilidad del modelo
- Razonando sobre el grafo, respóndase a las siguientes cuestiones:
  - ¿Tiene el programa un comportamiento equitativo? En caso negativo, evaluar el número de posibles historias no equitativas y mostrar al menos una.
  - ¿Hay problemas de bloqueo?
  - ¿Se cumple la exclusión mutua entre los estados de los procesos modelados, respectivamente, con los lugares `p3` y `q3`?
  - Probar que los estados modelados por los lugares `p1`, `p2` y `p3`, están en exclusión mutua.


p1
q1
t1
t4
s4
s1
p2
q2
t2
s2
p3
q3
t3
s3
wantP
wantQ


p1+q1+wP(0)+wQ(0)
t1
s1
t2(v=0)
p3+q2+wP(1)+wQ(0)
s3(v=1)
s2(v=0)
p1+q3+wP(0)+wQ(1)

Ejercicio 2 (1.5 ptos.)

Cuatro generales **A**, **B**, **C** y **D** deciden, respectivamente **A**, **R**, **A**, y **R**. Considérese la siguiente situación:

- **A** envía su decisión en primera vuelta a **B** y **C**, y cae antes de enviársela a **D**
- **B** miente en segunda vuelta a **C** sobre **A**

El ejercicio pide completar las tablas en las que los generales leales basan su decisión, y determinar si llegan a consenso.

C					
General	**Plan**	**Reported by**			**Majority**
		A	B	D	
A					
B					
C					
D					
Majority					

D					
General	**Plan**	**Reported by**			**Majority**
		A	B	C	
A					
B					
C					
D					
Majority					

Ejercicio 3 (3.5 ptos.)

Un sistema distribuido contabiliza y gestiona las partidas de un juego en red. El número máximo de partidas que se pueden estar jugando simultáneamente es de 25. En cada partida se enfrentan dos jugadores cualesquiera de entre los registrados en el sistema. Antes de iniciar una partida es necesario crear una pareja de jugadores. Para ello, un primer jugador solicita iniciar una nueva partida y espera a que realice la misma acción un posible contrincante. Cuando lo hace, se ponen de acuerdo para comenzar a jugar. En un momento determinado, el jugador de menor identificador de la pareja decide terminar el juego, se lo comunica a su pareja, espera su confirmación y actualiza el estado del sistema. El sistema consta de 100 procesos *Jugador*, que participan de forma continuada en distintas partidas, y un proceso *Gestor*, que inicializa el sistema y muestra periódicamente, por la salida estándar, el estado del sistema (se limita a informar del número de parejas que están jugando).

Se pide completar el código de los procesos involucrados, utilizando la notación presentada en clase, y resolviendo las cuestiones de comunicación y sincronización del sistema por medio de Linda.

Además, la solución debe satisfacer las siguientes restricciones: **1)** No podrá estar basada en un proceso servidor que encapsule el estado del sistema y sea el responsable de gestionar la creación de las parejas, y **2)** Deberá contener una descripción clara y concisa de la sintaxis y la semántica de las tuplas que se utilizan para coordinar los procesos involucrados.

```
process Jugador(i::1..100)
  -- Declaración de variables locales
  loop forever
     DESCANSANDO() --implementado
     -- Buscar contrincante
     -- Sincronizar el inicio de la partida
     JUGAR() --implementado
     -- Sincronizar el final de la partida
     -- Finalizar de la partida (proceso con menor identificador)
  end loop
end process

process Gestor
  -- Declaración de variables locales
  -- Establecer el estado inicial del sistema
  loop forever
     -- Mostrar por salida estándar el número de parejas jugando
     ESPERANDO()
  end loop
end process
```

Ejercicio 4 (3.5 ptos.)

Se trata de coordinar la ejecución de determinada tarea, que debe ser llevada a cabo por 5 parejas, de manera simultánea. Para ello, 50 procesos ejecutan el código que se muestra debajo. El ejercicio pide desarrollar el código del monitor Coordinador. Se debe explicar cada variable utilizada: qué codifica y cómo lo hace.

```
process P(i::1..50)
   integer miPareja --ID de mi pareja

   loop forever
      --formar las parejas, CONFORME VAN LLEGANDO A ESTE PUNTO
      --se bloquea si nadie está esperando o ya hay 10 dentro
      --asigna a "miPareja" el id de la pareja asociada
      Coordinador.damePareja(i, miPareja)

      --bloquear los procesos hasta que estén las 5 parejas formadas
      Coordinador.esperar5()

      EJECUTAR_TAREA() --implementado

      if (i < miPareja)
         Coordinador.avisaFinTarea()
      end
   end loop
end process

monitor Coordinador
   ...
end
```
