PROGRAMACIÓN CONCURRENTE. EXAMEN FINAL 14-06-04

Apellidos: PLANTILLA **Nombre:** _____

PRIMERA PARTE

1 **[0.5]** El siguiente algoritmo,

a) ¿cumple las propiedades de exclusión mutua de los procesos en el acceso y la de alcanzabilidad de la sección crítica? Para cada una de estas propiedades se pide su demostración, o bien un escenario de ejecución de los procesos en el que no se cumpla la citada propiedad.

b) Qué tipo de propiedad de equidad habría que suponerle a la planificación de los procesos para que ambos procesos consiguieran entrar un número ilimitado de veces en sección crítica.

```
P0::=                          P1::=
1 c0:= 0;                      1 c1:= 0;
2 turno:=0;                    2 turno:= 1;
3 while c1=0 do                3 while (c0≠1 and
4   begin                        turno=1) do
5   c0:=1;                       nothing;
6   while c1=0 do              enddo;
7     nothing;enddo;           Sección Crítica;
8   c0:=0;                     c1:=1;
9 enddo;
10 Sección Crítica;
11 C0:=1;
        (i)                            (ii)
```

Figura 1

```
Pi
<<resto de instrucciones>>
(1) repeat
(2) flag[i]:= solicitando;
(3) j:= turn;
(4) while j≠i do
(5)   if flag[j]≠pasivo then j:= turn;
(6)      else j:= (j-1) mod n;endif;
(7) enddo;
(8)   flag[i]:= en_SC;
(9) k:= 0;
(10) while(k<n)∧(k=ivflag[k]≠ en_SC) do
(11) k:=k+1 enddo;
(12) until k≥n;
(13) <<sección crítica>>
(14) turn:= (turn-1) mod n;
(15) flag[i]:= pasivo;
  ...
```

2 El algoritmo de la figura 1 pretende resolver el problema de la *exclusión mutua para n-procesos*, para lo cual sólo utiliza n variables `flag` que pueden tomar 3 posibles valores, `(pasivo,solicitando,en_SC)`, 1 variable `turn: 0..n-1` y la variable local a los procesos `j`. Las variables globales del algoritmo se inicializan a `pasivo` y `0`.

Se pide:

a) **[0.25]** Demostrar la propiedad de *alcanzabilidad* de la sección crítica para cualquier proceso no pasivo.

b) **[0.75]** Comprobar que la máxima espera de un proceso P_i que intenta entrar en sección crítica (ésto es, con flag[i] ≠ pasivo) es de 2^{n-1}-1 turnos. Ayuda: construye un escenario para 4 procesos (o más), que incluya los valores de las variables `flag[i]` y `turno` de los procesos durante su ejecución, que muestre una secuencia de ejecución del algoritmo que haga retrasarse a uno de los procesos el número de turnos previsto por la fórmula anterior.

3 **[0.25]** Demostrar que $\{r = 2^i\}$ es un invariante de $i := i+1;\quad r := r \times 2;$

4 Dado el código del siguiente monitor, que satisface el invariante: `{(0≤ s) ∧ ((s≥m ∧ p2(x) ∧ b)→ "s no disminuye" (p1(m) se bloquea)}`

a) **[0.5]** ¿Con qué tipos de señales (SX, SW, SU ó SC) sería *correcto* [1] el siguiente monitor?. Justificarlo utilizando las *reglas de detección de la equivalencia entre señales de monitores*.

b) **[0.25]** Suponiendo señales SU, ¿en qué orden se ejecutarían los procesos bloqueados en la cola de c cuando otro proceso, que ejecute `p2(...)`, les señale que se da la condición para que terminen de ejecutar `p1(...)`?

```
Monitor buffer
Const N0= (* valor inicial*)
  var s: integer;
      b: boolean;
      c: cond;
procedure p1(m:positive);
 begin
   while (m > s or b) do begin
         c.wait();
     if (m>s or b) then begin
           b:=true;
           c.signal();
     endif;
   enddo;
   s:= s-m;
   c.signal();
 end;
Procedure p2(x:positive);
 begin
  s:= s+x;  b:= false;
  c.signal();
end;

begin b:=false; s:= N0 end;
```

[1] se cumple el invariante al terminar los procedimientos y no hay "robo de señal" por los procesos que esperan entrar al monitor

5 **[0.5]** Con respecto a los axiomas de la operación **wait** de los monitores: explicar por qué el axioma de la operación **c.wait** de las señales con semántica *desplazante* ha de ser diferente del correspondiente para las señales *SC*.

el axioma para señales con semántica desplazante ha de asegurar que se cumple la condición de la lógica después del wait → asegurar que el proceso desbloqueado es el único que puede avanzar (no puede cambiar ningún invariante de la cola de un hilo)

6 **[0.5]** En las demostraciones de los procesos de un programa concurrente con monitores no podemos incluir ningún invariante como parte de los asertos en las demostraciones de los procesos. Explicar a qué es debido esto y escribir la regla de verificación que lo prohíbe.

lo prohíbe la regla de concurrencia para programas con monitores

[1.5] PROBLEMA: Monitor de implementación de una operación broadcast que asegure la vivacidad de los procesos que solicitan recibir un mensaje en una red (suponiendo que los mensajes se envían frecuentemente).

Suponer un número desconocido de procesos consumidores y productores de mensajes en una red de comunicaciones muy simple. Los productores llaman al procedimiento `broadcast(int m)` para enviar una copia de un mensaje a los procesos consumidores que lo hayan solicitado (se supone que se encuentran bloqueados desde que ejecutaron el procedimiento `fetch(x)`). También recibirán una copia del mensaje aquellos procesos que ejecuten `fetch(x)` mientras dure la difusión del mensaje a los consumidores que estaban bloqueados; los primeros no tienen que esperar, por tanto, para obtener la copia del mensaje, sino que la obtienen inmediatamente. Los consumidores llaman al procedimiento `fetch(int x)`, donde `x` es un argumento-resultado.

Se pide programar un monitor que, al menos, implemente los 2 procedimientos anteriores utilizando para ello señales SC y que cumpla las siguientes condiciones:

i) Un productor que ejecute `broadcast(int m)` no puede continuar con la siguiente instrucción de su programa hasta que al menos todos consumidores en espera tengan copia del mensaje.

i) Sólo se ha de almacenar 1 mensaje pendiente de ser recibido por todos los consumidores que lo soliciten; es decir, se ha de utilizar una variable `int` permanente del monitor para almacenar el mensaje hasta que sea recibido por los consumidores que hayan ejecutado el método `fetch()`.

ii) Si hay un mensaje pendiente de ser recibido por consumidores bloqueados, entonces cualquier llamada a `broadcast()` por parte de otro productor será retrasada hasta que los consumidores hayan recibido su copia del mensaje. Si por el contrario no se estuviera difundiendo ningún mensaje a los consumidores, nada impide que un productor llame a `broadcast()`, asignándole un valor ordinario a la variable permanente del monitor.

iii) Cuando el mensaje sea recibido por el último de los consumidores que lo esperan, la variable permanente del monitor tomará un valor singular que indique que no hay ningún mensaje que esté siendo difundido en ese momento.

Hacer la modificaciones necesarias en el código de los procedimientos del monitor anterior para que funcione con señales SU.

```
broadcast(m:int){
  while (mem ≠ -1) do
    c3.wait();
  mem := m;
  while (c1.queue()) do
    c1.signal();
  while (esperando > 0) do
    c2.wait();
  mem := -1;
  c3.signal();
}
```

```
fetch (var x:int){
  if mem ≠ -1 then { hay un mensaje difundiéndose }
    x := mem
  else begin { mem = -1; no hay mensaje }
    esperando++;
    c1.wait();
    x := mensaje;
    c2.signal();
    esperando--;
  end
}
```

```
begin_fetch (var x:int){
  bloqueados++;
  if (mem ≠ -1) then x := mem
  else begin { mem = -1 }
    c1.wait(); x := mem
  }
end_fetch (){
  bloqueados--;
  if (c1.queue()) then c1.signal()
  else if (c2.queue()) then c2.signal()
}
```

Apellidos:_____ **Nombre:** _____

SEGUNDA PARTE

1 **[1.0]** Para las expresiones CSP siguientes, indicar con (V/F), en cada caso. 2 respuestas erróneas anulan 1 correcta.

a. [F] El programa siguiente *se bloquea* siempre:

[in(i:1..N)::IN || mux::[(i:1..N) sigue;x:integer;in(i) ?x -> out ! (i, x) □ true-> sigue:= false;]].*

b. [F] Se produce interbloqueo:

$[P_1::*[P_2 ? x \rightarrow NULL\ ] \parallel P_2::P_1 ! (1,2)]$

c. [V] Ambos procesos terminan:

$[P_1::x:integer;*[P_2 ? x \rightarrow x:= x+5\ ] \parallel P_2::P_1 ! 0\ ]$

d. [F] El programa es correcto y termina:

$[P_1::*[x>0;P_2 ? x \rightarrow y:= x/3\ ] \parallel P_2::x:integer;\ P_1 ! 0]$

e. [F] El siguiente proceso P_1 termina y, después, termina P_2 :

$[P_1::y:integer;*[P_2 ? x \rightarrow y:= x+1;\ P_2 ! x] \parallel P_2::P_1 ! 0]$

f. [F] El programa termina:

$[P1 :: *[P_2 ? x \rightarrow x:= x+1; P_2 ! x; \square P3 ?\ y \rightarrow y:= y+1] \parallel P_2::\ P_1 ! 1 \parallel P_3::P_1 ! 6]$

↳ Bloquea

g. [F] El programa termina siempre que P_1 no elija la segunda alternativa:

$[P_1::*[NOT\ termina; P_3 ? x \rightarrow x:= x+5;\ termina:=true \square NOT\ termina;\ P_2 ? x \rightarrow x:= x+5;\ termina:= true]$
$\parallel P_2::P_1\ !\ (0,\ 1) \parallel P_3::P_1\ !\ 0;\ P_1 ?\ y]$

h. [V] El siguiente programa no termina

$[P_1::x:integer;[FALSE;P_2 ? x \rightarrow x:= x+5\ ] \parallel P_2::\ P_1\ !\ 0]$

i. [F] El siguiente programa se bloquea: $[P_1::*[sigue:boolean;P_2 ? x \rightarrow sigue:= false;\ ] \parallel P_2::P_1 ! x()]$

j. [V] Se bloquean ambos procesos:

$[P_1::\ *[\ var\ x:positive;P_2 ? x \rightarrow x:= x+5\ ] \parallel P_2::\ P_1 ! (-5)]$

2 **[1.0]** ¿Podemos afirmar que es condición necesaria para que una red de Petri sea repetitiva el que esta sea viva y cíclica? Justificarlo (demostrándolo si es cierto, o bien dando un contraejemplo si no lo es).

es condición suficiente, pero no necesaria (se pueden encontrar redes repetitivas que no son cíclicas)

3 **[1.5]** Demostrar que si un red marcada (R,M_0) es viva, entonces para cualquier transición de la red $t \in T$ (que ha de poder dispararse a partir de un marcado M alcanzable desde M_0 si la red es viva) se cumplirá la siguiente relación de acotación $M_0 \cdot Y = M \cdot Y \geq \sum_{p \in |Y|} Y(p) \cdot n(p,t) \geq 1$. Si la relación anterior se cumpliera para todas las componentes conservativas elementales de una red, ¿podríamos afirmar que satisface la propiedad de vivacidad?, ¿qué podríamos afirmar acerca de la vivacidad de la red si no se cumpliese dicha relación para alguna componente conservativa? Demuestra tus respuestas.

2, 4, 8, 9, 10

$Le \wedge Ve \Rightarrow R$

$(\overline{Le \wedge Ve}) \vee R \equiv \overline{Le} \vee \overline{Ve} \vee R \equiv \overline{Ve} \vee R \vee \overline{Le}$ | $Le \wedge Ve \rightarrow R \wedge C$

$\overline{Ve} \vee R \vee \overline{Le} \equiv (\overline{Ve \wedge \overline{R}}) \vee \overline{Le} \equiv Ve \wedge \overline{R} \Rightarrow \overline{Le}$ | $\overline{R} \rightarrow \overline{Le} \vee \overline{Ve}$

4 ⓟ [0.5] ¿Qué podemos suponer acerca de las diferentes propiedades de limitación de una red si sólo sabemos que es *estructuralmente viva*, y *no-repetitiva*? Demuestra tu contestación.

no puede ser estructuralmente limitada

$Le \Rightarrow L$

5 [0.5] Si se diera que tenemos una red cíclica que no cumple la propiedad de *repetitividad*, entonces ¿qué otras propiedades no deberían cumplirse y por qué no?

la limitación estructural es una condición suficiente pero no necesaria de limitación (Si una red no es limitada estructuralmente, puede ser limitada o no). Sabemos que la red $\overline{Le}$, luego no podemos afirmar nada acerca de su propiedad de limitación

6 [1.0] Qué propiedad de los *cerrojos y las trampas* permite determinar la propiedad de *vivacidad parcial* de una red. Enunciar la propiedad y demostrarla.

$\neg R \rightarrow \neg Le \vee \neg Ve$

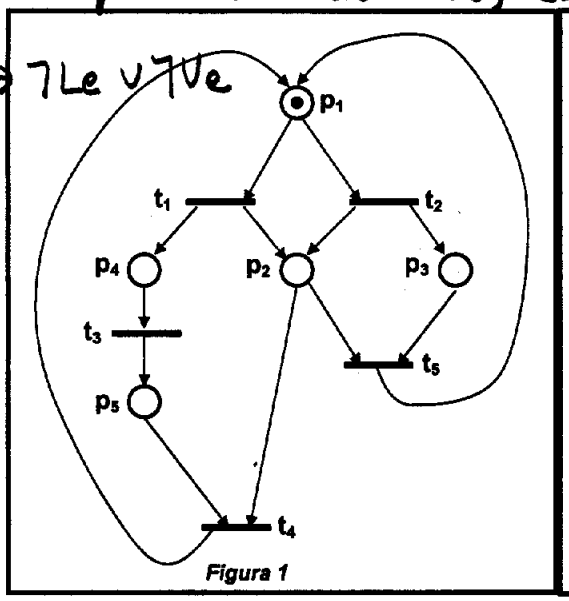


Figura 1

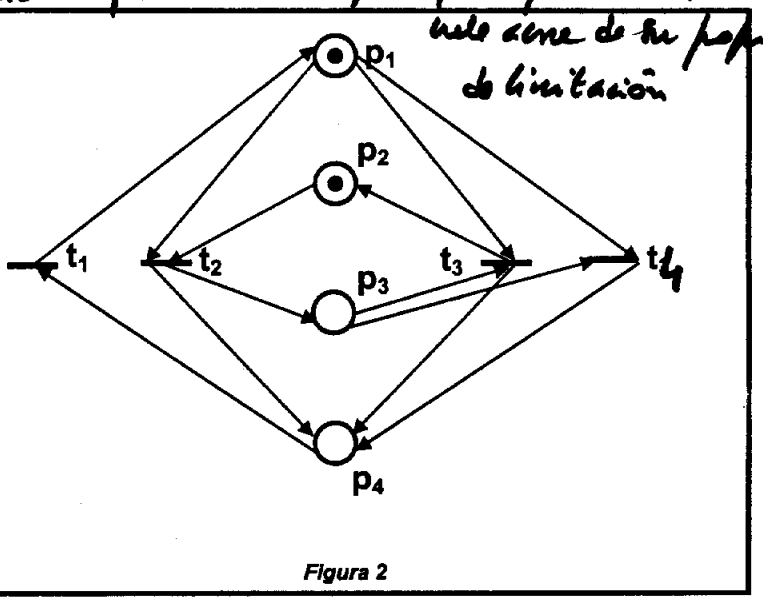


Figura 2

7 [1.0] Hallar los lugares implicantes de todos los lugares potencialmente implícitos de la red de la figura 1, utilizando para ello alguno de los métodos analíticos estudiados.

$M_0 = (1100)$

8 ⓐ[1.0]

a) Construir el *árbol de alcanzabilidad* de la red de la figura 2 y utilizar la información que contiene este grafo para saber si esta red cumple las propiedades de *vivacidad, repetitividad y ciclicidad*.

$\overline{V}$ $\overline{R}$ $\overline{C}$

b) ¿Qué podemos afirmar acerca de la propiedad de vivacidad (liveness) de la red de la figura 2? Podría determinarse un marcado inicial que haga que la red sea viva ¿Por qué? no vale porque el disparo sucesivo de t_4 hace que en algún marcado se disparen todos los lugares excepto p_4.

t_2 (0011) t_1 (1010) t_3 t_4 (0101) t_1 (1100) viejo (0001) t_1 (1000) terminal

9 ⓓ [1.0] Calcular el avance sincrónico de la transición t_1 respecto de la t_4 de la figura 3, así como la multiplicidad del lugar implícito respecto de estas transiciones y su marcado inicial

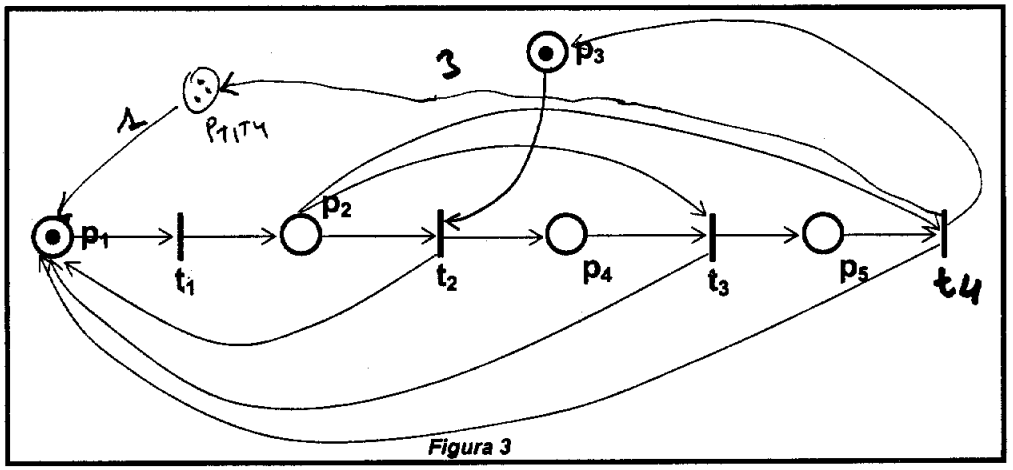


Figura 3

$M_0 = (01100)$ t_1 (01100) t_2 (10010) t_1 (01010) t_3 (10001) t_1 (01001) t_4 (10100) viejo

$D = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 1 & -1 & -1 & 1 & 0 \\ 1 & -1 & 0 & +1 & 1 \\ 1 & -1 & 1 & 0 & -1 \end{bmatrix}$ $r(D) = 3$

$t_1 t_2 t_1 t_3 t_1 t_4$ $\quad 3t_1/t_4$

$\begin{vmatrix} -1 & 0 & 0 & -\alpha \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 1 & 0 \\ 1 & 0 & -1 & \beta \end{vmatrix} = 0 \rightarrow +(-1)\cdot \begin{vmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & \beta \end{vmatrix} - (-\alpha)\cdot \begin{vmatrix} 1 & 1 & 0 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \end{vmatrix} = -\beta + 3\alpha$

10 ⓫ [1.5] Modelar un sistema de servicio de comidas de un pequeño restaurante utilizando para ello redes de Petri Coloreadas. La descripción del servicio de comidas a los clientes es como sigue:

Hay sólo 2 camareros en el restaurante: el que sirve platos de sopa, que llena el único plato encima de una mesa, que repone cuando el cliente sale de la habitación y se lo da al cliente que lo pide. El segundo camarero, que sirve los segundos platos, se encuentra en otra habitación del restaurante; también este sólo sirve 1 plato cada vez; pero, dispone de 2 platos llanos encima de la mesa. Los clientes acceden a las habitaciones donde están los camareros de 1 en 1. Las personas que entran al restaurante pueden tomar, o bien 1 plato de sopa y 1 segundo plato, o bien 2 segundos platos. La entrada al restaurante que utilizan los que van a tomar sopa y un segundo plato es diferente de la que utilizan los que van a tomar dos segundos platos, ya los segundos van a ser servidos por un solo camarero, mientras que los primeros serán servidos por los dos camareros, que se encuentran en diferentes habitaciones.

Hay 2 salas de espera, en las que aguardan los clientes antes de entrar a la habitación de cada camarero, una para ser servido por el camarero de la sopa (A) y otra para ser servido por el camarero de los segundos platos (B). Pero el acceso a (B) está restringida a los clientes que toman sopa, porque esta sala es pequeña y hay más gente que prefiere la opción de tomar dos "segundos" en lugar de sopa y "1 segundo". De tal forma que el camarero sólo servirá la siguiente sopa si no hay ya un cliente con sopa en la sala de espera (B). Además, un cliente con sopa no podrá entrar a la sala de espera (B) hasta que el camarero que sirve segundos platos esté ocupado! Sólo uno de los dos tipos clientes presentes en la sala (B) conseguirá entrar a la habitación donde está el siguiente camarero. Una vez que un cliente consiga entrar, sólo se le dará un segundo plato si ya tiene sopa, o bien dos segundos platos si no. Después de servir a un cliente, el camarero repone el(los) plato(s) que había(n) encima de la mesa, dependiendo del tipo de cliente al que haya servido. El resto de los clientes de la sala (B) permanecerán esperando hasta que la habitación del camarero de los segundos platos se quede libre de nuevo.

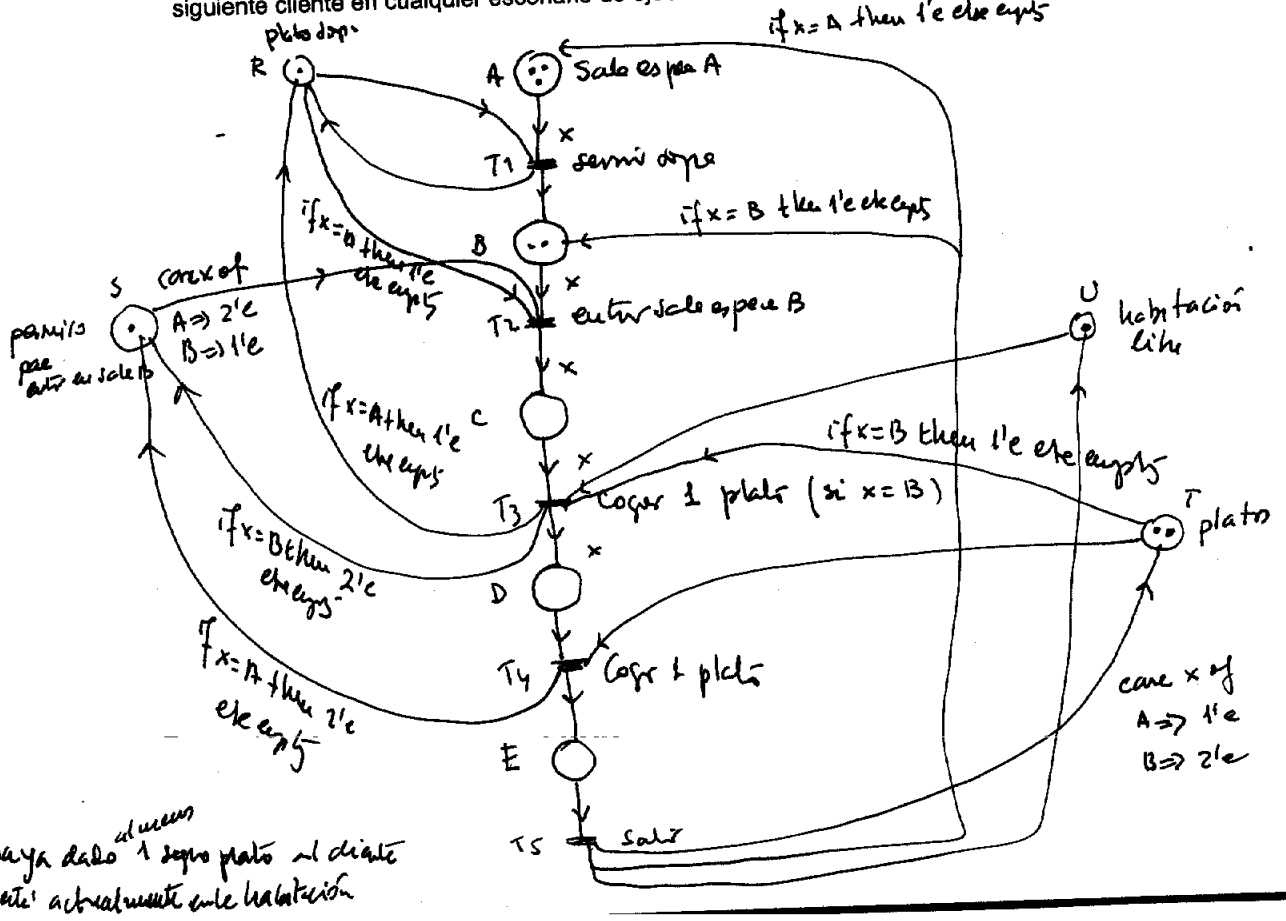
Se pide: modelar la actuación de los camareros, de los dos tipos de clientes, así como su competición por los recursos comunes (los platos) y responder a las siguientes preguntas:

a) ¿Cuántas personas como máximo pueden llegar a ser servidas (por camareros distintos) en un momento dado del funcionamiento del servicio que se ha descrito? Construir el *grafo de ocurrencia* de la red de Petri coloreada que hayas elaborado para demostrar tu respuesta.

b) Si siempre existieran en las salas de espera al menos 2 personas de cada tipo cuando se reponen los platos, ¿podría ocurrir en algún momento del funcionamiento del servicio que durante un tiempo alguna de las salas A o B se quedara vacía?

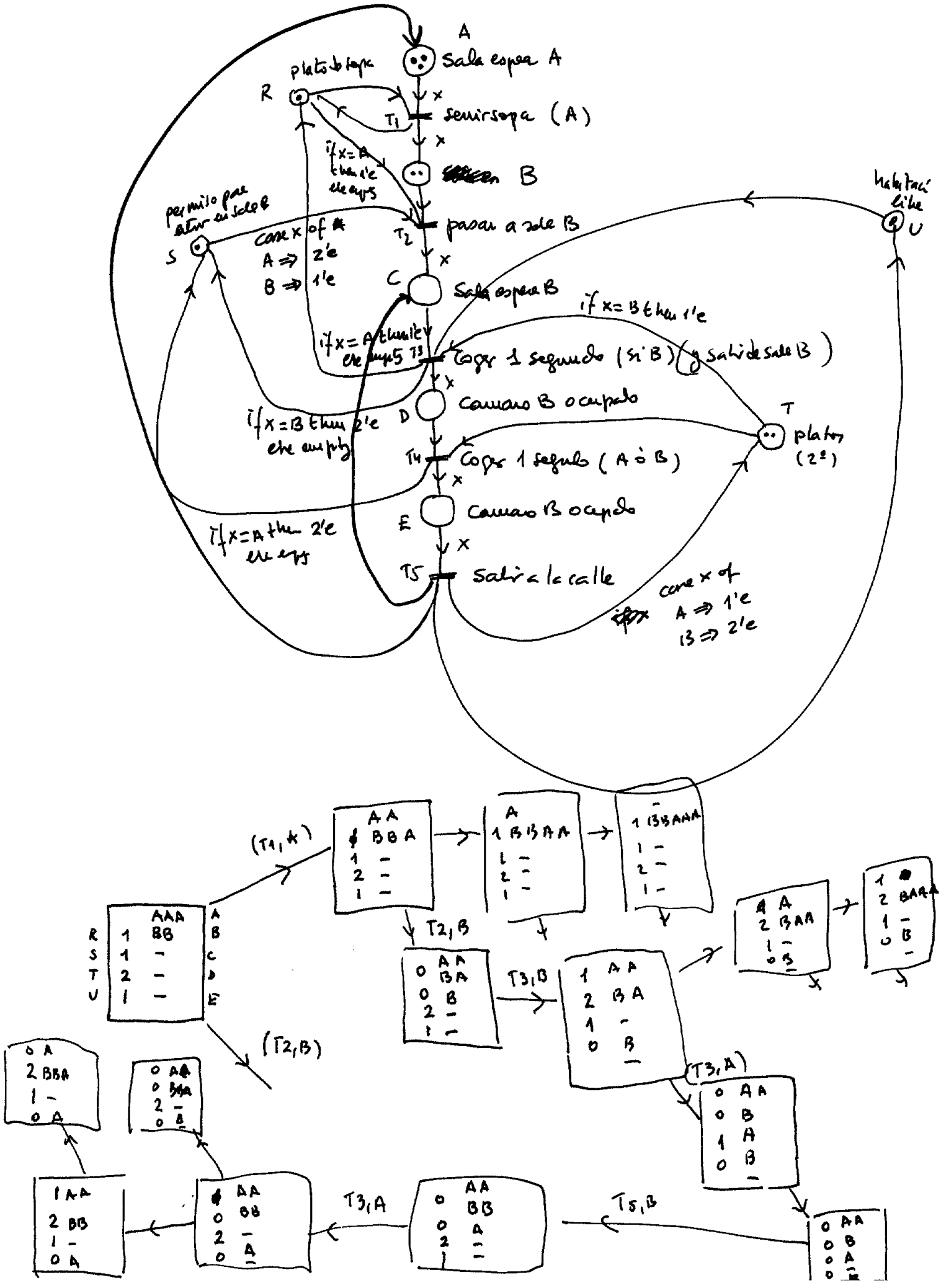
c) Sólo mirando el número de platos en la mesa del segundo camarero, cuando el cliente sale de la habitación, ¿se puede decir el tipo de cliente que acaba de ser servido?

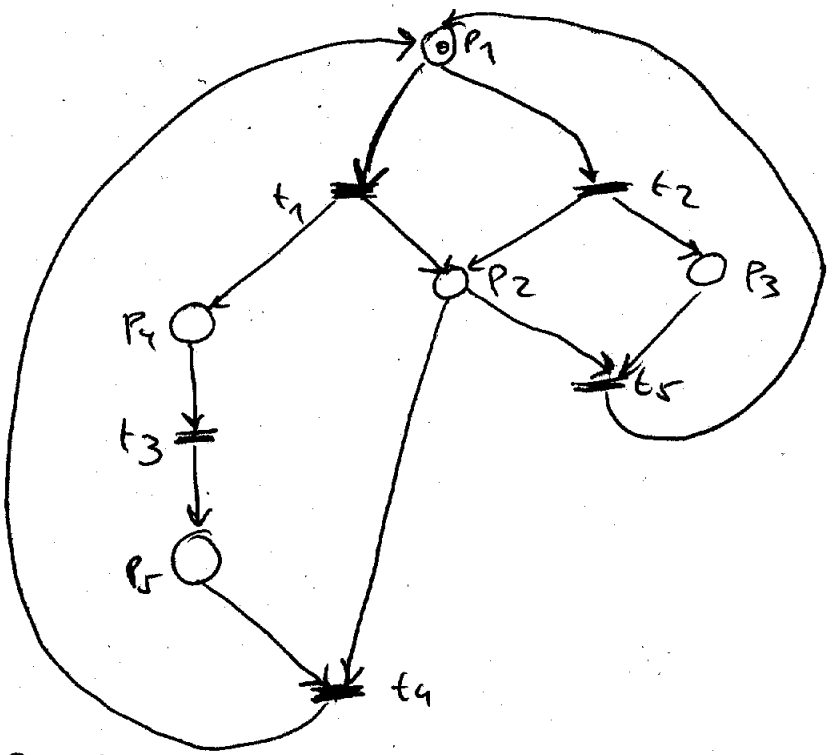
d) Demostrar que el sistema que has implementado cumple siempre la propiedad de *vivacidad;* por ejemplo, demostrar que los camareros siempre dispondrán de platos para servir la comida del siguiente cliente en cualquier escenario de ejecución del servicio.



Lema $R \Rightarrow G$ (prop. 2)

pregunta: ¿podría existir una red conservativa que no fuese viva?


A
Sala espera A
plato bajo
R
x
T1
semirsopa (A)
x
if x=B then 1'e else empty
B
permito pasar a sala B
case x of
A ⇒ 2'e
B ⇒ 1'e
S
T2
pasar a sala B
x
C
Sala espera B
habitación libre
U
if x=B then 1'e
if x=A then 1'e else empty T3
Coger 1 segundo (si B) (y salir de sala B)
x
D
Camara B ocupado
T
plato (2ª)
if x=B then 2'e else empty
T4
Coger 1 segundo (A ó B)
x
E
Camara B ocupado
if x=A then 2'e else empty
x
T5
Salir a la calle
case x of
A ⇒ 1'e
B ⇒ 2'e
(T1, A)
(T2, B)
T2, B
T3, B
(T3, A)
T3, A
T5, B



$D = \begin{matrix} p_1 \\ p_2 \\ p_3 \\ 1 \end{matrix}$

$$D = \begin{array}{c} \\ t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \end{array} \begin{array}{c} \begin{matrix} p_1 & p_2 & p_3 & p_4 & p_5 \end{matrix} \\ \left[\begin{array}{ccc|cc} -1 & 1 & 0 & 1 & 0 \\ -1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 \\ 1 & -1 & 0 & 0 & -1 \\ 1 & -1 & -1 & 0 & 0 \end{array}\right] \end{array}$$

c.l. (columnas)

c.l. (filas)

$r(D) = 3$

nº elems del conjt elem = $\dim(P) - r(D) = ?$

Components elemals

$$(C|A) = \left[\begin{array}{ccccc|ccccc} 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -1 & 0 \end{array}\right] \rightarrow \left[\begin{array}{ccccc|ccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & -1 \\ 1 & 0 & 0 & 1 & 0 & 0 & -1 & -1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -1 & 0 \end{array}\right] \rightarrow \left[\begin{array}{ccccc|ccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & -1 & 0 \end{array}\right]$$

$$\rightarrow \left[\begin{array}{ccccc|ccccc} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & \phi & 0 & 0 & 0 & 0 & 0 \end{array}\right] \qquad Y = \left\{ \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ \phi \end{pmatrix} \right\}$$

$Y = (21111)$ el única lug implicat és p_2

$$Y = \left\{ \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right\} \qquad Y = (21111)$$

$$k = \max_{j \neq 1} \left[\frac{Y(j)}{Y_2(j)} \right] = 2$$

$Y_2 = (10111)$

$$\| 2 \cdot Y_2 - Y \| = \begin{array}{r} 2\ 0\ 2\ 2\ 2 \\ -\ 2\ 1\ 1\ 1\ 1 \\ \hline 0\ \ 1\ 1\ 1 \end{array} = \{P_3, P_4, P_5\}$$

$\Pi_2 = \{P_3, P_4, P_5\}$

$Y_3 = (11000)$ $\qquad k' = \max_{j \neq 3} \left[\frac{Y(j)}{Y_3(j)} \right] = 2$

$$\| 2 \cdot Y_3 - Y \| = \begin{array}{r} 2\ 2\ 0\ 0\ 0 \\ -\ 2\ 1\ 1\ 1\ 1 \\ \hline \end{array} = \| Y_3 \| = \{P_2\} = \| Y_4 \| = \| Y_5 \|$$

$\overline{P_4} = (-10100)$

$k = 4$ $k = (2) \circ 3$

$$(C|A) = \left[\begin{array}{ccccc|ccccc} 1&0&0&0&0 & -1&-1&0&1&1 \\ 0&1&0&0&0 & 1&1&0&-1&-1 \\ 0&0&1&0&0 & 0&1&0&0&-1 \\ 0&0&0&1&0 & -1&0&1&0&0 \\ 0&0&0&0&1 & 0&0&1&-1&0 \end{array}\right] \begin{matrix}(1)\\(2)\\(3)\\(4)\\(5)\end{matrix} \Rightarrow \left[\begin{array}{ccccc|ccccc} 1&1&0&0&0 & 0&0&0&0&0 \\ 0&0&1&0&0 & 0&1&0&0&-1 \\ 0&1&0&1&0 & 0&1&1&-1&-1 \\ 0&0&0&0&1 & 0&0&1&-1&0 \end{array}\right] \begin{matrix}(1)+(2)\\(3)\\(2)+(4)\\(5)\end{matrix}$$