

BUGUI SOFT

Integrantes:

Juan Manuel Gil

Francisco Ciordia Cantarella

Julieta Prieto

Gina Commisso

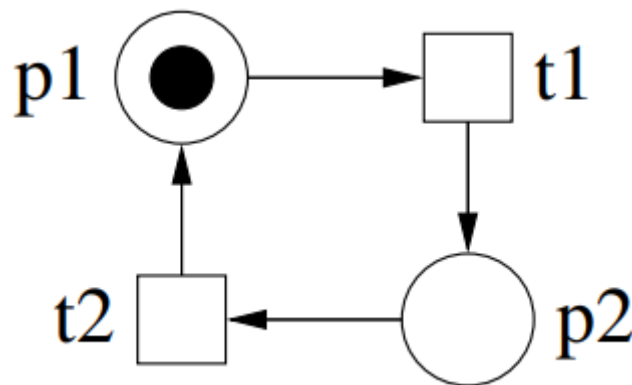
ÍNDICE

NOMENCLATURA	3
PRESENTACIÓN DE LA RED	4
Gráfico	4
CLASIFICACIÓN	5
ANÁLISIS DE LOS INVARIANTES	6
ANÁLISIS DE CONFLICTOS Y DEADLOCK	9
ANÁLISIS DE CONCURRENCIA	10
IDENTIFICACIÓN DEL PROBLEMA	11
PROPUESTAS DE SOLUCIONES	11
SOLUCIÓN 1	12
SOLUCIÓN 2	16
SOLUCIÓN 3	21
TABLAS DE ESTADOS Y EVENTOS	24
TABLA DE ESTADOS DE LA RED ORIGINAL	24
TABLA DE EVENTOS DE LA RED ORIGINAL	25
CAMBIOS A LAS TABLAS SEGÚN LAS SOLUCIONES PROPUESTAS	25
GRÁFICOS DE HILOS	27
Política	28
Monitor	29
Implementación en Java	29
Semántica Temporal	31
Consideraciones	34
Conclusiones	37
Update#1	37
Referencias	40

NOMENCLATURA

A lo largo del trabajo, haremos referencia a las siguientes definiciones de elementos[6]:

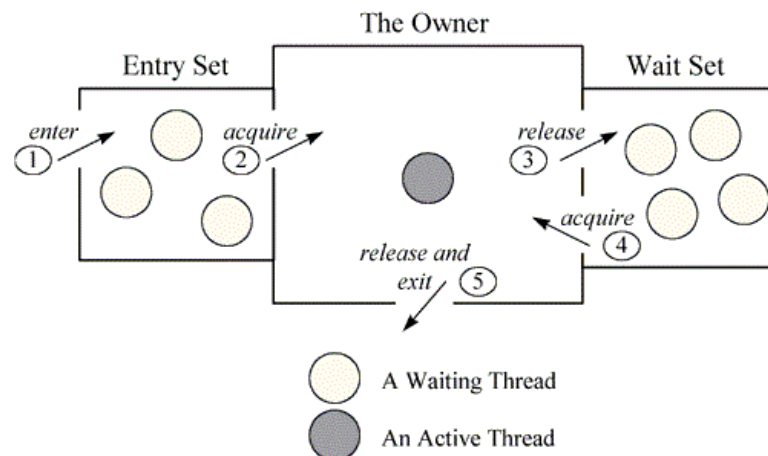
- T-invariantes: es un vector conformado por números enteros asociados a una secuencia de disparos. Son útiles para determinar propiedades estructurales de una PN en forma analítica. Sea N la red de Petri, C la matriz de incidencia, una solución natural $C \cdot u = 0$ es llamada invariante de transición o T-invariante de N . Por ejemplo, en la siguiente imagen $u = (1, 1)^T$ es un T-invariante



Un T-invariante indica un posible loop en la red, una secuencia de transiciones cuyo efecto en la red es nulo ya que vuelve al marcado en donde comenzó.

- Sección crítica: partes del programa donde se accede a un recurso compartido que está protegido de manera que se evite el acceso concurrente.
- Exclusión mutua: Es el requisito de que un hilo en ejecución nunca entre en su sección crítica al mismo tiempo que otro hilo la esté ejecutando.
- Deadlock: Un punto muerto es un estado en el que cada miembro de un grupo de procesos/hilos está esperando que otro miembro, incluido él mismo, tome medidas, como enviar un mensaje o, más comúnmente, liberar un bloqueo

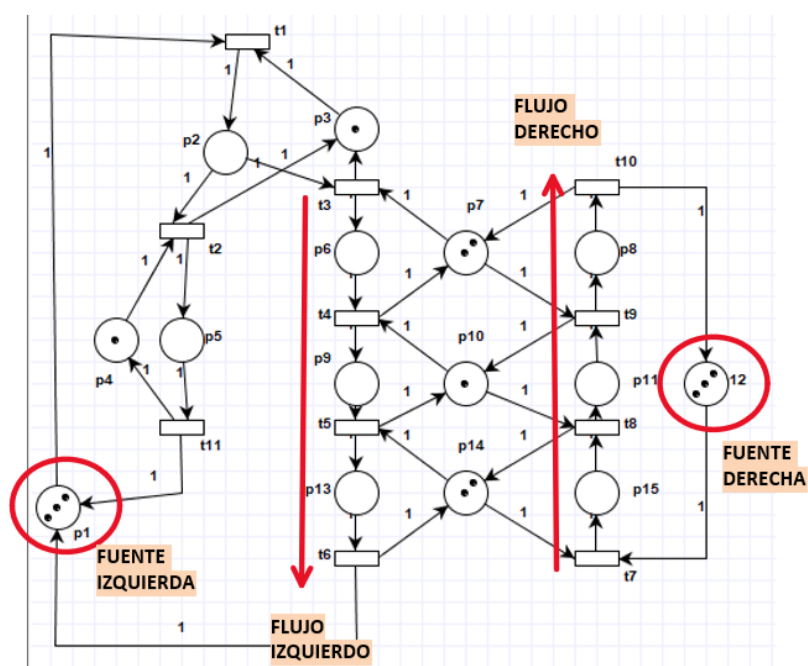
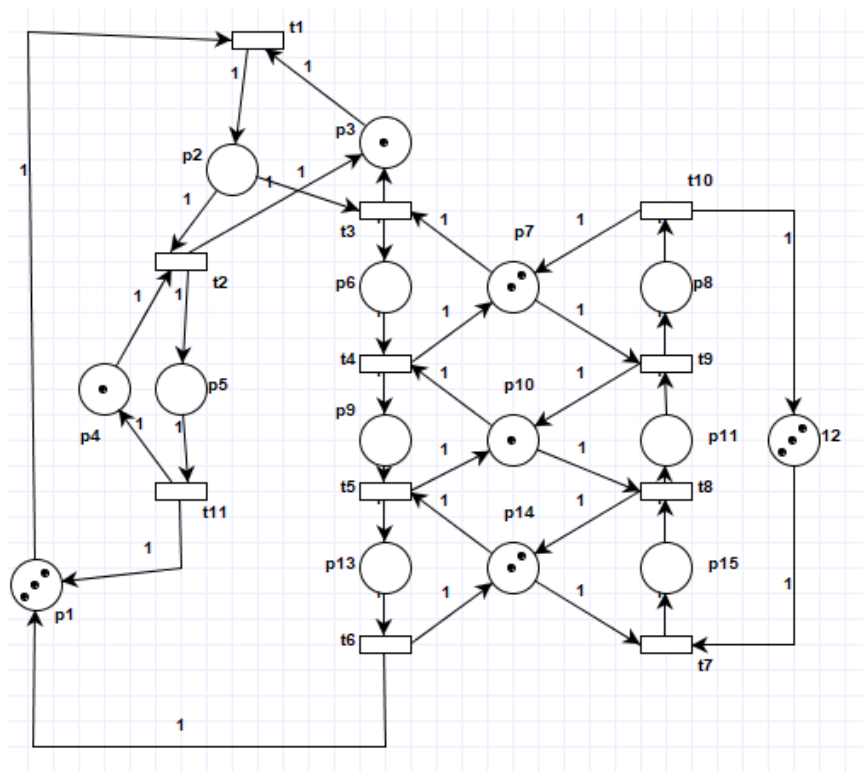
- **Monitor:** Es un mecanismo de software (de alto nivel) para control de concurrencia que contiene los datos y los procedimientos necesarios para realizar la asignación de un determinado recurso o grupo de recursos compartidos reutilizables en serie. Son de gran utilidad para la comunicación entre procesos y localización de recursos en una sección crítica.



- **Política:** Es un conjunto de reglas que define el comportamiento del monitor. En el cuál le otorgará más o menos prioridad a los distintos tipos de colas del monitor.

PRESENTACIÓN DE LA RED

Gráfico



Clasificación

Petri net classification results

State Machine	false
Marked Graph	false
Free Choice Net	false
Extended Free Choice Net	false
Simple Net	false
Extended Simple Net	false

Análisis de los Invariantes

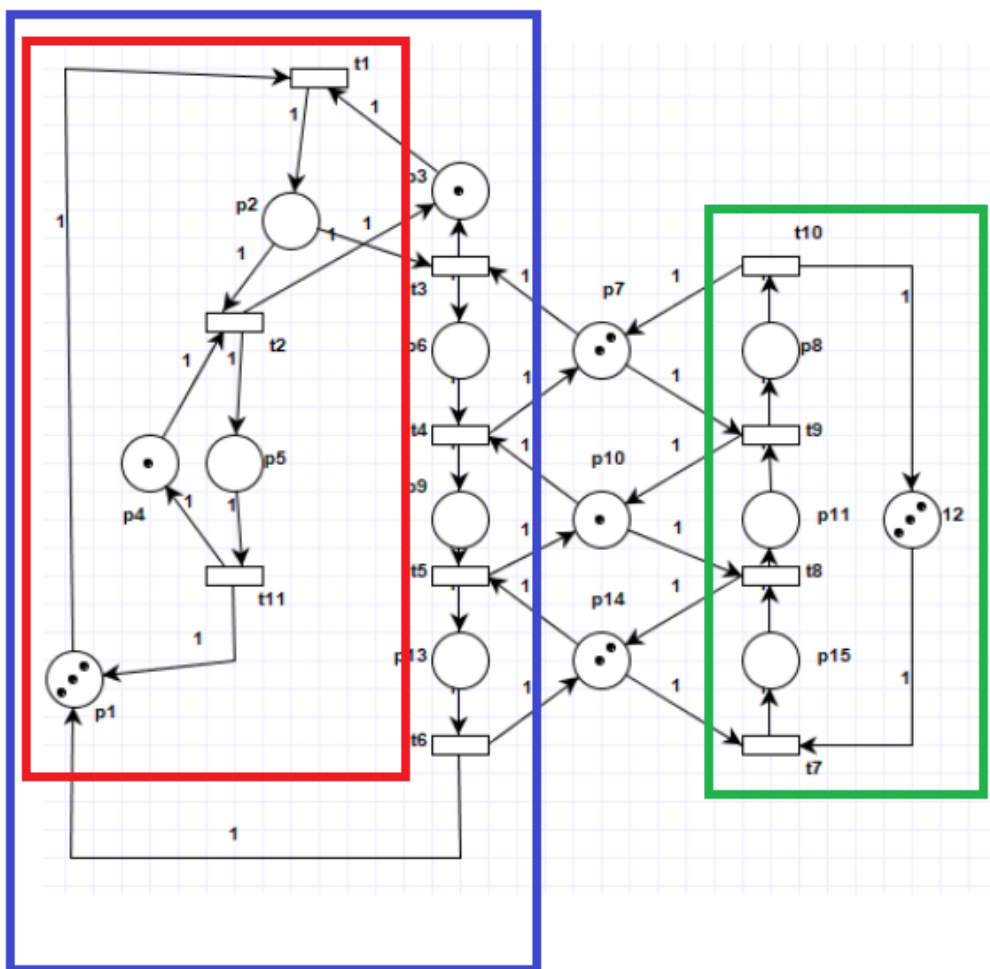
Petri net invariant analysis results

T-Invariants

t1	t3	t4	t5	t6	t2	t11	t10	t9	t8	t7
1	1	1	1	1	0	0	0	0	0	0
1	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1

The net is covered by positive T-Invariants, therefore it might be bounded and live.

Como la red tiene T-invariantes positivos, **es probable** que sea acotada y viva. Viva quiere decir que sea cual sea la evolución del marcado, todas las transiciones se van a poder volver a disparar. Es decir, está la posibilidad de que no haya interbloqueo, pero aun no lo comprobamos. Acotadas hace referencia a que las plazas están delimitadas por un marcado inicial y por un entero que limita la cantidad de tokens que pueden tener.



Ciclos/bucles del sistema [IMG 2]

P-Invariants

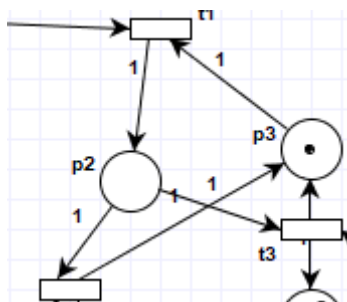
p3	p6	p9	p13	p7	p10	p14	p12	p2	p5	p4	p8	p11	p15	p1
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	1	0	0	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1	0
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
0	1	1	1	0	0	0	0	1	1	0	0	0	0	1

The net is covered by positive P-Invariants, therefore it is bounded.

P-Invariant equations

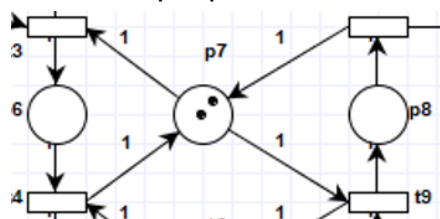
- 1 $M(p3) + M(p2) = 1$
- 2 $M(p6) + M(p7) + M(p8) = 2$
- 3 $M(p9) + M(p10) + M(p11) = 1$
- 4 $M(p13) + M(p14) + M(p15) = 2$
- 5 $M(p12) + M(p8) + M(p11) + M(p15) = 3$
- 6 $M(p5) + M(p4) = 1$
- 7 $M(p6) + M(p9) + M(p13) + M(p2) + M(p5) + M(p1) = 3$

1. Sección crítica



2. Capacidad finita (LIMITADOR-2)

El máximo número de tokens que puede almacenar p6, p7 y p8 es 2.



3. Exclusión mutua

Recurso en p10 que puede ser utilizado por t4 o t8 pero no de forma simultánea.

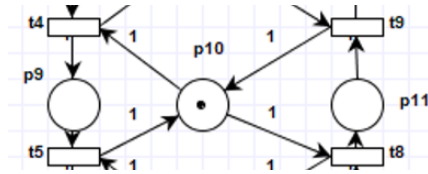


UNC

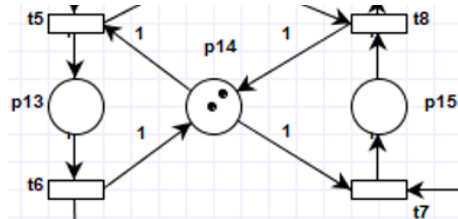
Universidad
Nacional
de Córdoba



FCEFN
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS y NATURALES

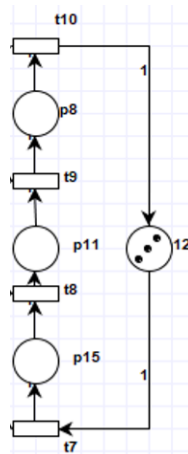


4. Capacidad finita (LIMITADOR-2)

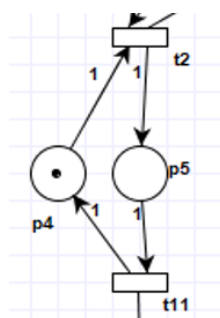


5. Secuencia

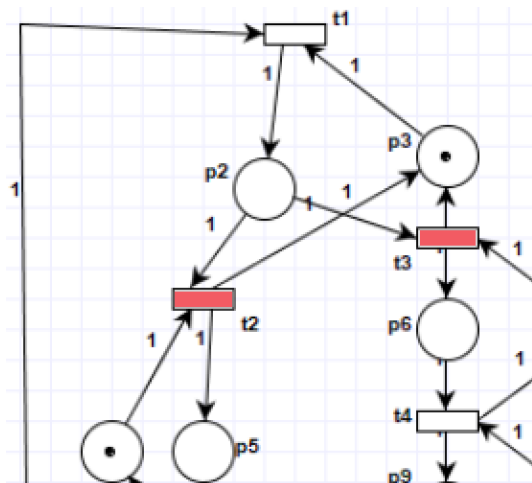
Varios disparos en forma ordenada.



6. Sección crítica



7. Conflicto en P2.



ANÁLISIS DE CONFLICTOS Y DEADLOCK

Análisis de los espacios de estados

Petri net state space analysis results

Bounded	true
Safe	false
Deadlock	true

Shortest path to deadlock: t1 t3 t1 t3 t1 t4 t3 t5 t4 t5 t4 t6 t1 t5 t6 t3 t1 t2 t4 t5 t6 t1 t6 t3 t1 t3 t4 t11 t7 t1
 t3 t5 t6 t1 t2 t4 t5 t4 t6 t1 t5 t6 t3 t1 t11 t4 t5 t3 t1 t8 t6 t2 t1 t9 t4 t3 t5 t4 t5 t6 t1 t6 t3 t4 t11 t1 t3 t1 t5 t4 t3
 t5 t4 t6 t5 t1 t6 t3 t6 t1 t7 t4 t3 t1 t5 t4 t2 t6 t1 t3 t5 t6 t1 t8 t11 t2 t11 t10 t7 t1 t2 t1 t7 t11 t2 t11 t9 t1 t4 t3
 t1 t2 t10 t11 t1 t3



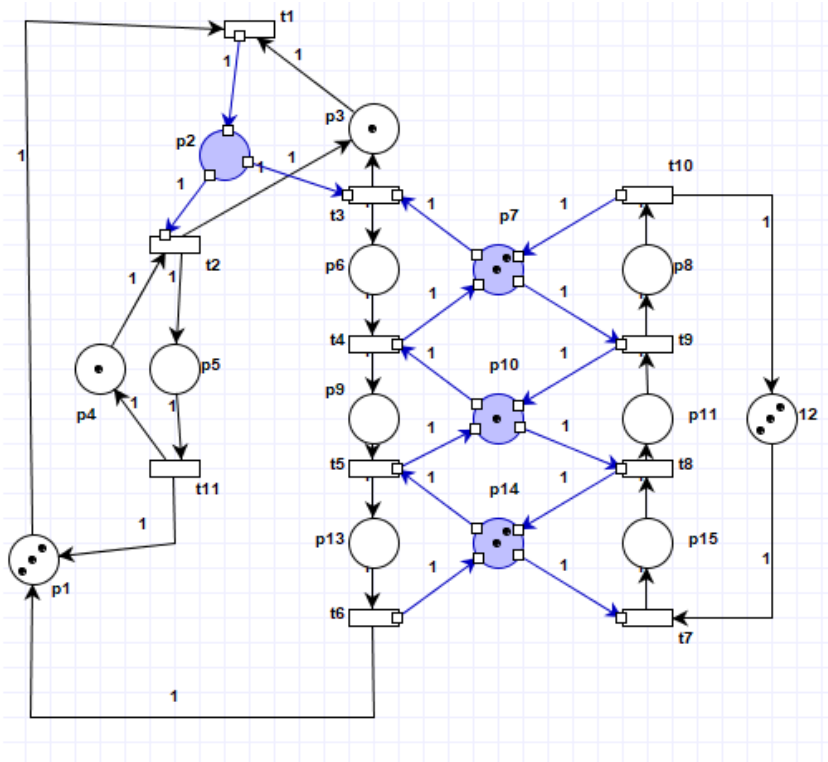
UNC

Universidad
Nacional
de Córdoba

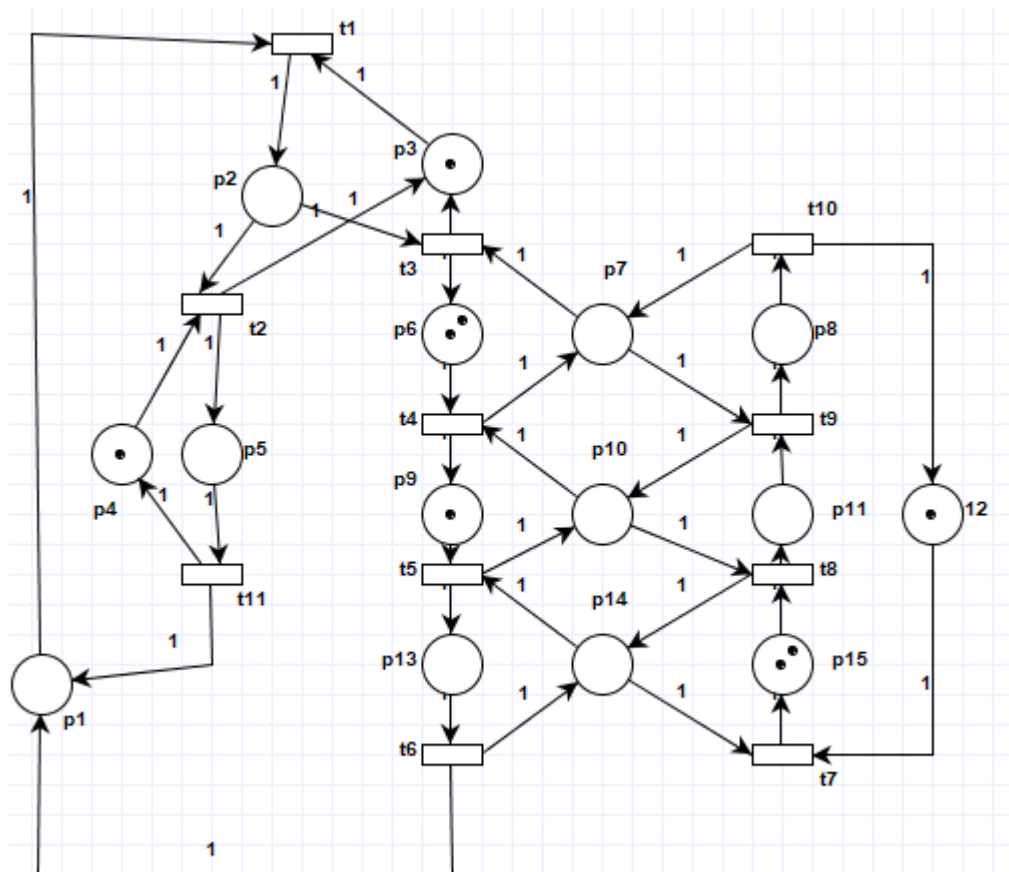


FCEFyN
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS y NATURALES

Conflictos estructurales



DEADLOCK



Este es uno de los casos en que se puede dar deadlock ya que de manera similar ocurre cuando hay un token en la plaza p11 en vez en lugar de la p9.

ANÁLISIS DE CONCURRENCIA

Para ello veremos el promedio de tokens marcados en el color oscuro ya que hacen referencia a los hilos en procesos.

Promedio	0,958	0,425	0,333	1,368	0,414	0,862	0,724	0,421	0,579	0,579	0,421	0,544	0,881	0,575	0,241
MIN	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MAX	3	1	1	3	2	2	2	1	1	1	1	2	2	2	1
	p1	p10	p11	p12	p13	p14	p15	p2	p3	p4	p5	p6	p7	p8	p9
	suma total			Promedio T	Suma max	prom. Max									
	2,831			0,472	10,000	1,667									

SUMA TOTAL= 2,831

PROMEDIO GENERAL=0,472

SUMA MAX= 10

PROM DEL MÁX= 1.667

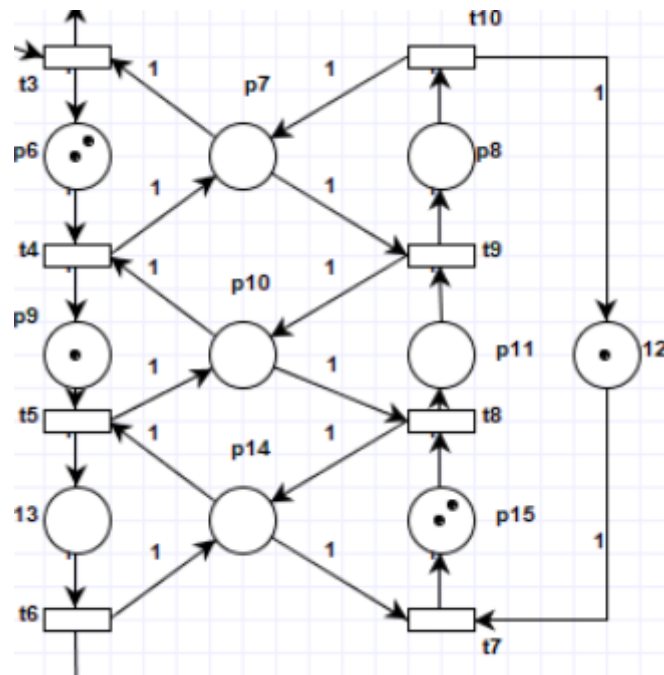
[Análisis hecho en Excel](#)

IDENTIFICACIÓN DEL PROBLEMA

El problema de deadlock se da cuando hay un hilo accediendo a un recurso en exclusión mutua, y cuando los recursos necesarios para continuar están tomados por los hilos de la fuente contraria.

En la imagen se aprecia cuando HAY 2 HILOS derechos QUE SE ENCUENTRAN ENTRE LAS PLAZAS DE ENTRADA AL FLUJO derecho (P15) Y AL MISMO TIEMPO HAY UN HILO perteneciente a la fuente izquierda EN EXCLUSIÓN MUTUA.

Como el problema radica en un sector de la red, solo nos enfocaremos en este bloque para proponer las soluciones.



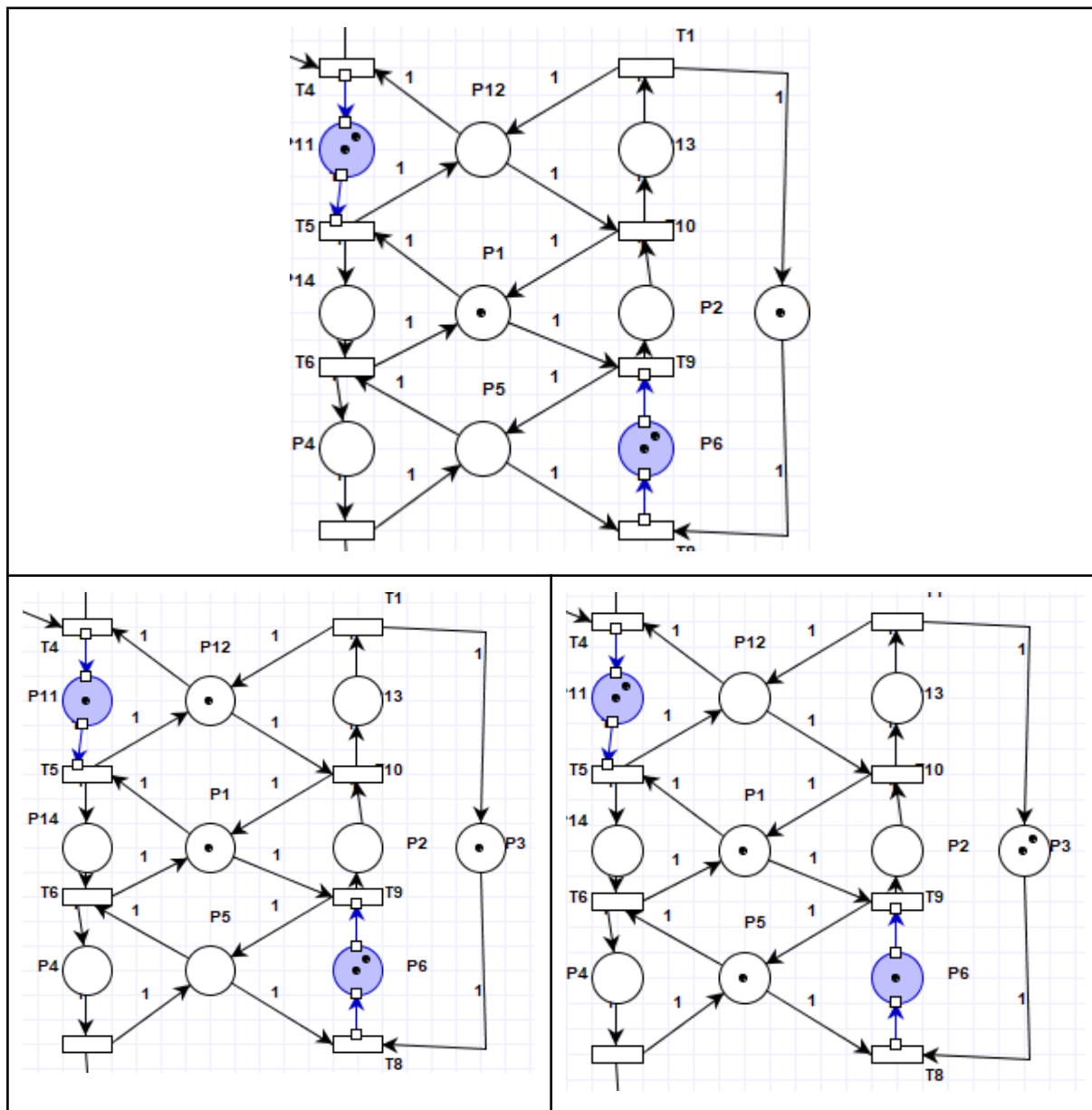
Bloque Deadlock

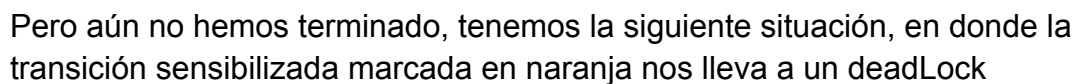
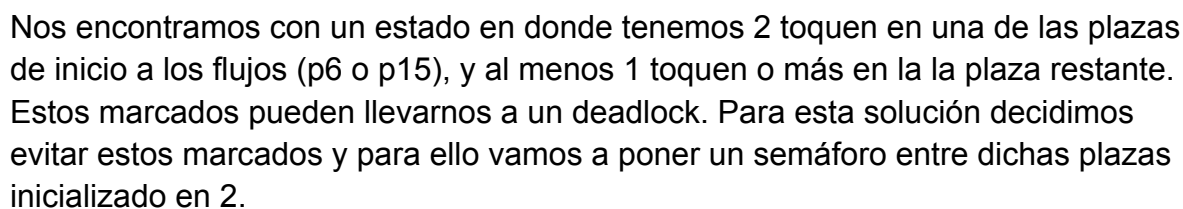
PROPUESTAS DE SOLUCIONES

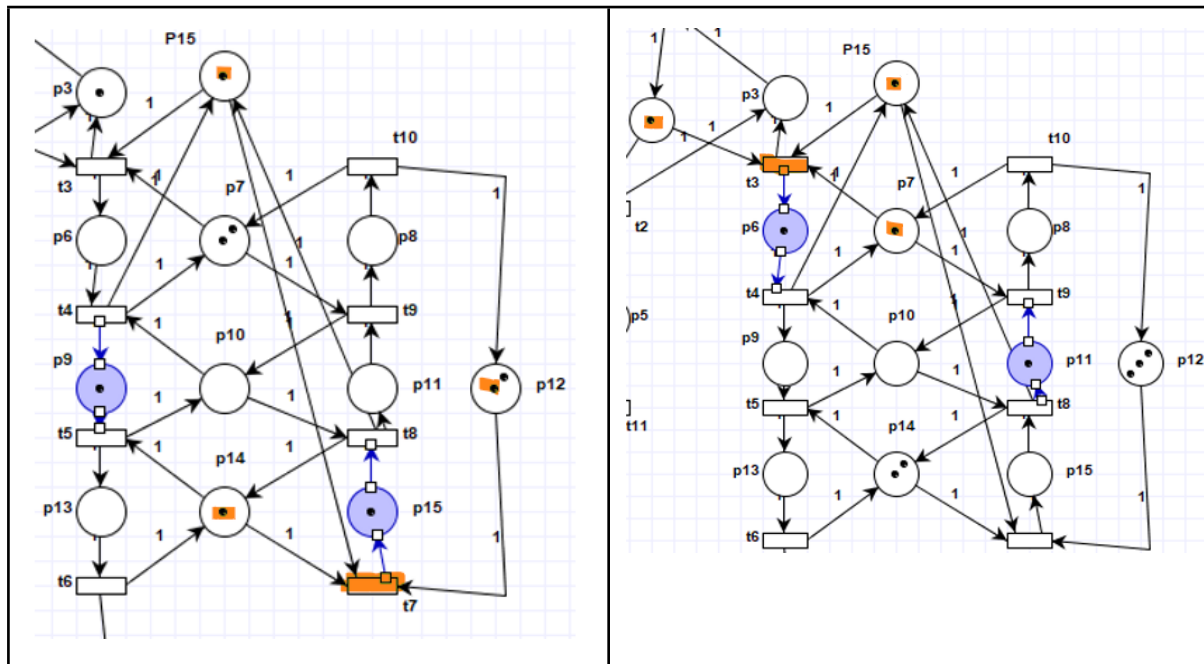
A continuación se presentarán las diferentes soluciones aplicables para este problema y la elección de nuestra solución final.

SOLUCIÓN 1

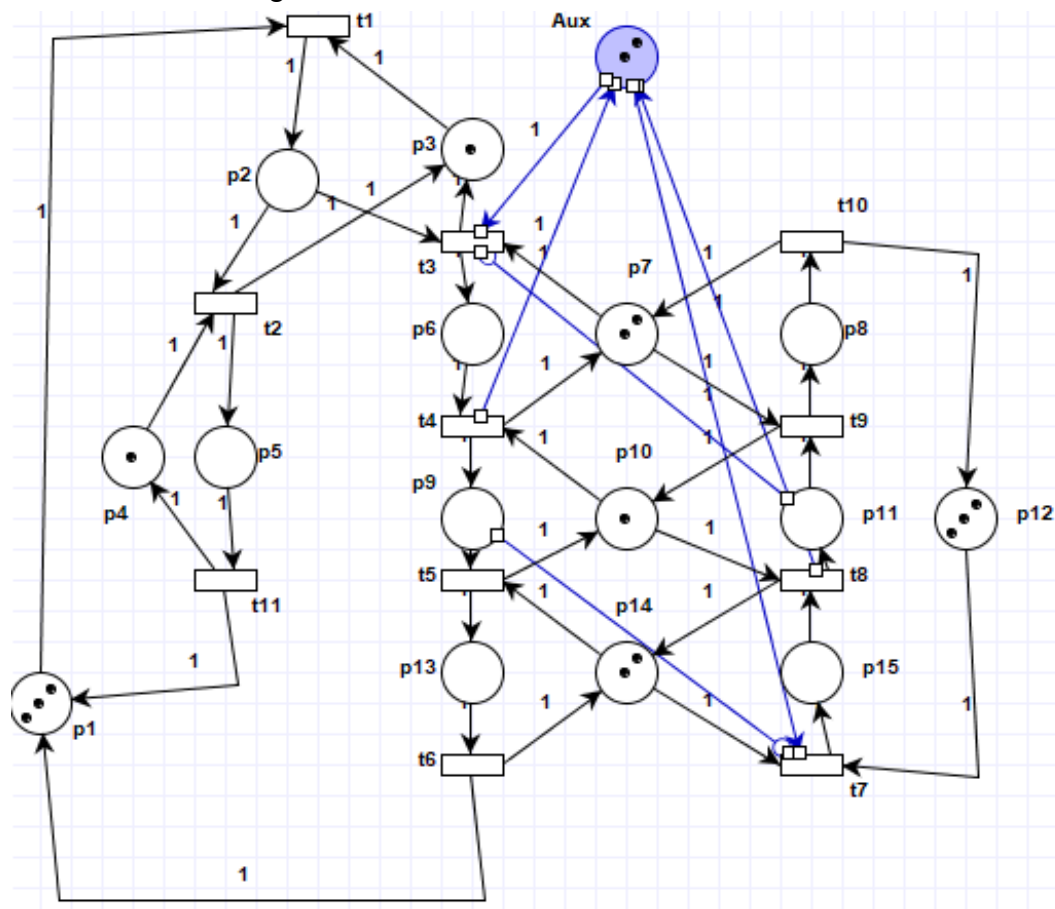
Para evitar la situación en donde un hilo ejecutando un recurso en exclusión mutua, se encuentre con que los recursos necesarios para continuar estén tomados por los hilos de la fuente contraria, hay que tener en cuenta los Marcados previos que nos llevan al deadLock:







Es por esto que colocaremos arcos inhibidores que impidan la sensibilización de las transiciones en relación a la plaza activa en exclusión mutua (p11 y p9), cómo se visualiza en la imagen:



- Análisis de los Resultados de la Red:

Petri net state space analysis results

Bounded	true
Safe	false
Deadlock	false

- ANALISIS DE LOS INVARIANTES:
Rectificamos que los invariantes se mantienen

Petri net invariant analysis results

T-Invariants

t1	t10	t11	t2	t3	t4	t5	t6	t7	t8	t9
1	0	1	1	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	0	0	0
0	1	0	0	0	0	0	0	1	1	1

The net is covered by positive T-Invariants, therefore it might be bounded and live.

P-Invariants

p1	p10	p11	p12	p13	p14	p15	p2	p3	p4	p5	p6	p7	p8	p9	P15
1	0	0	0	1	0	0	1	0	0	1	1	0	0	1	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	1	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1

The net is covered by positive P-Invariants, therefore it is bounded.

P-Invariant equations

$$M(p1) + M(p13) + M(p2) + M(p5) + M(p6) + M(p9) = 3$$

$$M(p10) + M(p11) + M(p9) = 1$$

$$M(p11) + M(p12) + M(p15) + M(p8) = 3$$

$$M(p13) + M(p14) + M(p15) = 2$$

$$M(p2) + M(p3) = 1$$

$$M(p4) + M(p5) = 1$$

$$M(p6) + M(p7) + M(p8) = 2$$

$$M(p15) + M(p6) + M(P15) = 2$$

- ANÁLISIS DE CONCURRENCIA:

	Marking															
	p1	p10	p11	p12	p13	p14	p15	p2	p3	p4	p5	p6	p7	p8	p9	AUX
Initial	3	1	0	3	0	2	0	0	1	1	0	0	2	0	0	0
Current	3	1	0	3	0	2	0	0	1	1	0	0	2	0	0	2

Respecto a al análisis, se consideró que el agregado del limitador en las plazas p6 y p15 mantienen el paralelismo o concurrencia de las tareas, sacrificando la libertad de usabilidad total de los recursos representados en la plazas p7 y p14 ya que con las restricciones propuestas deberán tener minimamente 2 de estos recursos inactivos.

Para ello veremos el promedio de las casillas de tono más oscuro que hacen referencia a los hilos en procesos.

[illegible]

SUMA TOTAL= 2,609

PROMEDIO GENERAL=0,435

SUMA MAX=10

PROMEDIO DEL MAX= 1.667

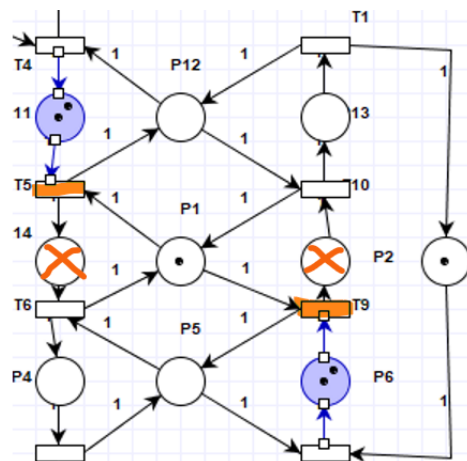
Verificamos que efectivamente disminuyó el **promedio** de las marcas que reflejan la cantidad de hilos y por ende la **conurrencia** en la plazas p6 y p15, ya que las estamos limitando como solución al deadlock

Análisis hecho en Excel

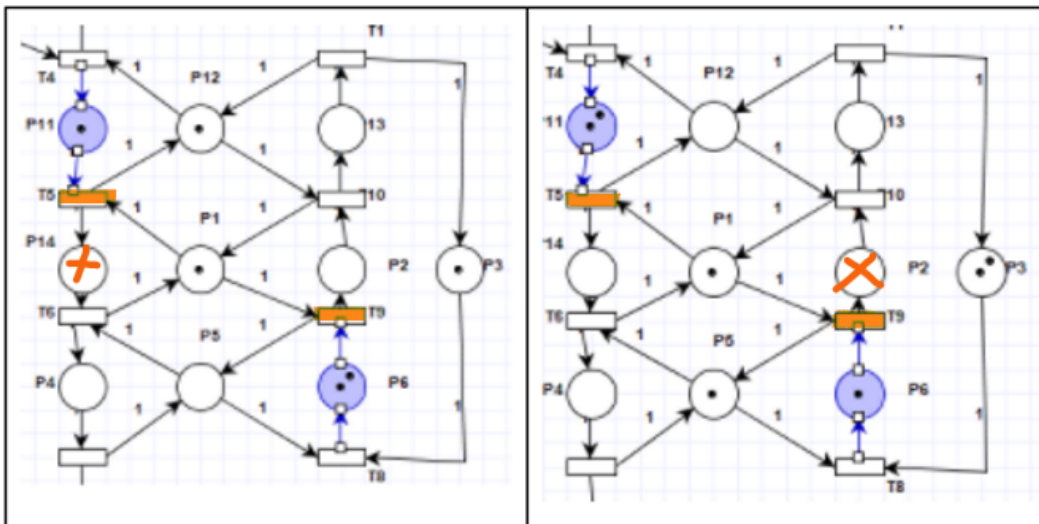
SOLUCIÓN 2

Para esta solución, nos enfocamos en la ACCIÓN previa al deadLock, en el preMarcado al deadlock.

Anteriormente lo que hacíamos era evitar este marcado, en cambio ahora solo evitaremos los pre marcados en los cuales es inevitable llegar al deadlock, y en caso contrario solo las transiciones que con dicho marcado nos lleven a un deadlock.

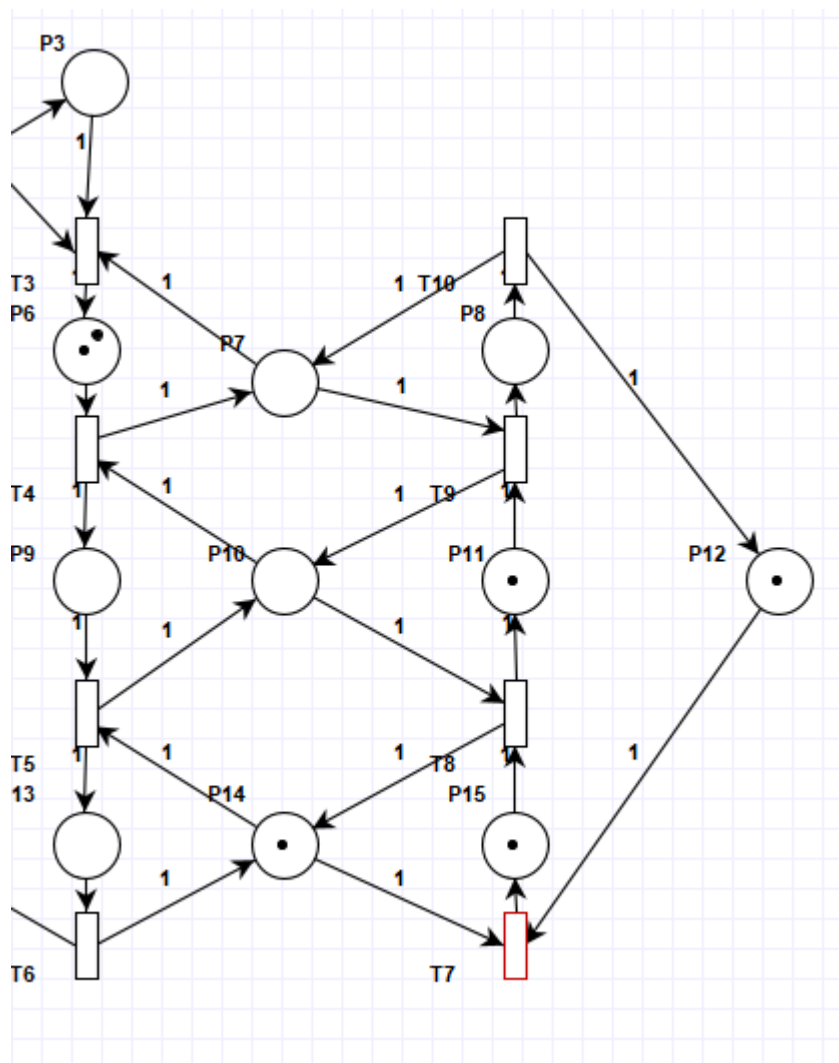


En esta situación, vemos que si sacamos el token de P1, nos quedamos con todas las transiciones de los laterales desensibilizadas.



En el primer caso, sacó el token de P1. Quedándose sin tokens en p1 y p5, se produce un deadlock.

En el segundo caso



otra situación previa al deadlock. Si se dispara t_7 se produce.

En la imagen se encuentran en naranja las transiciones sensibilizadas en los marcados previos al deadlock. Y con una cruz la acción que nos lleva al deadlock, el cual queremos evitar.

Como es de esperar evitaremos el marcado superior (con las dos cruces naranjas) mediante un semáforo iniciado en 3 tokens.

Para los marcados restantes utilizaremos semáforos inicializados en 2 para evitar el sensibilizado de las transiciones que nos llevan al deadlock, las cuales corresponden a las transiciones naranjas cuando se encuentran 2 tokens en la plaza azul correspondiente al flujo contrario.

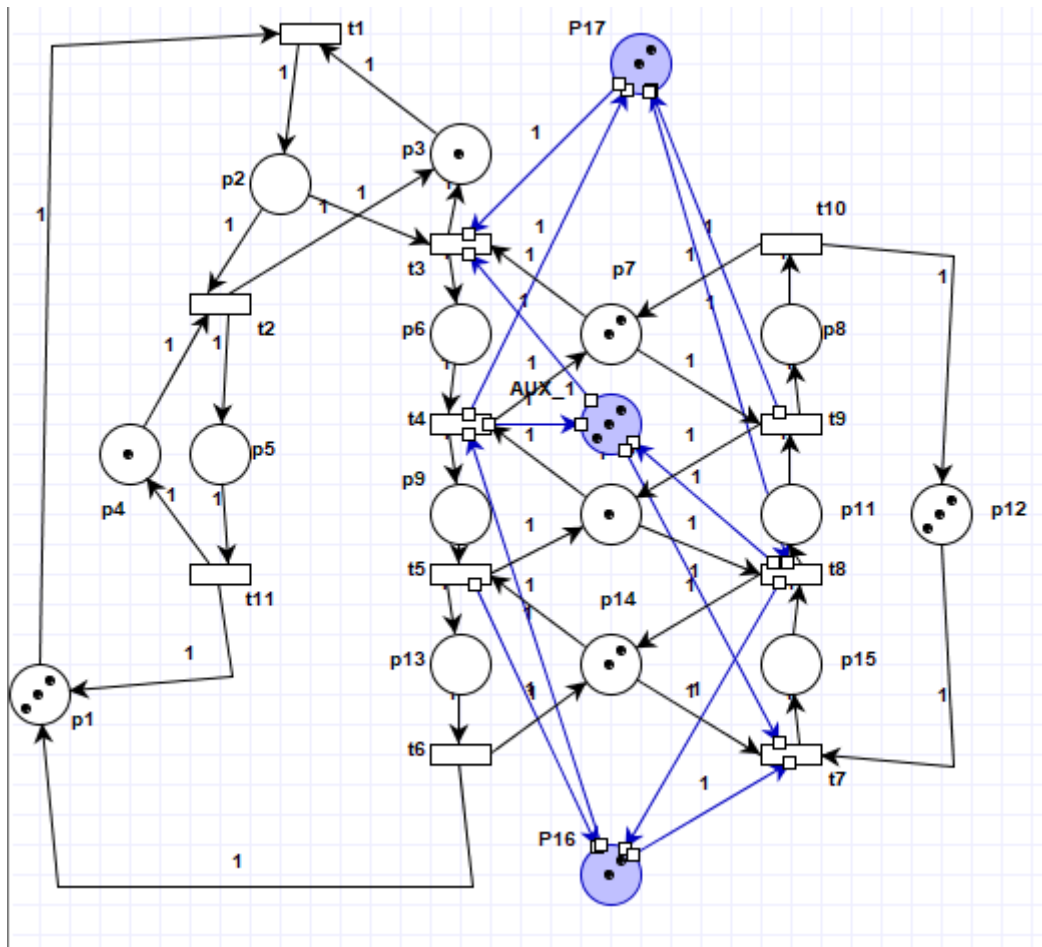


UNC

Universidad
Nacional
de Córdoba



FCEFyN
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES



- Análisis de los Resultados de la Red:

Petri net state space analysis results

Bounded	true
Safe	false
Deadlock	false

- ANALISIS DE LOS INVARIANTES:

En la siguiente imagen se puede ver que se mantienen



UNC

Universidad
Nacional
de Córdoba

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

P-Invariants

p1	p10	p11	p12	p13	p14	p15	p2	p3	p4	p5	p6	p7	p8	p9	AUX_1	P16	P17
1	0	0	0	1	0	0	1	0	0	1	1	0	0	1	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	1	1	0	0	1	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0
0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

The net is covered by positive P-Invariants, therefore it is bounded.

P-Invariant equations

$$M(p1) + M(p13) + M(p2) + M(p5) + M(p6) + M(p9) = 3$$

$$M(p10) + M(p11) + M(p9) = 1$$

$$M(p11) + M(p12) + M(p15) + M(p8) = 3$$

$$M(p13) + M(p14) + M(p15) = 2$$

$$M(p2) + M(p3) = 1$$

$$M(p4) + M(p5) = 1$$

$$M(p6) + M(p7) + M(p8) = 2$$

$$M(p15) + M(p6) + M(AUX_1) = 3$$

$$M(p15) + M(p9) + M(P16) = 2$$

$$M(p11) + M(p6) + M(P17) = 2$$

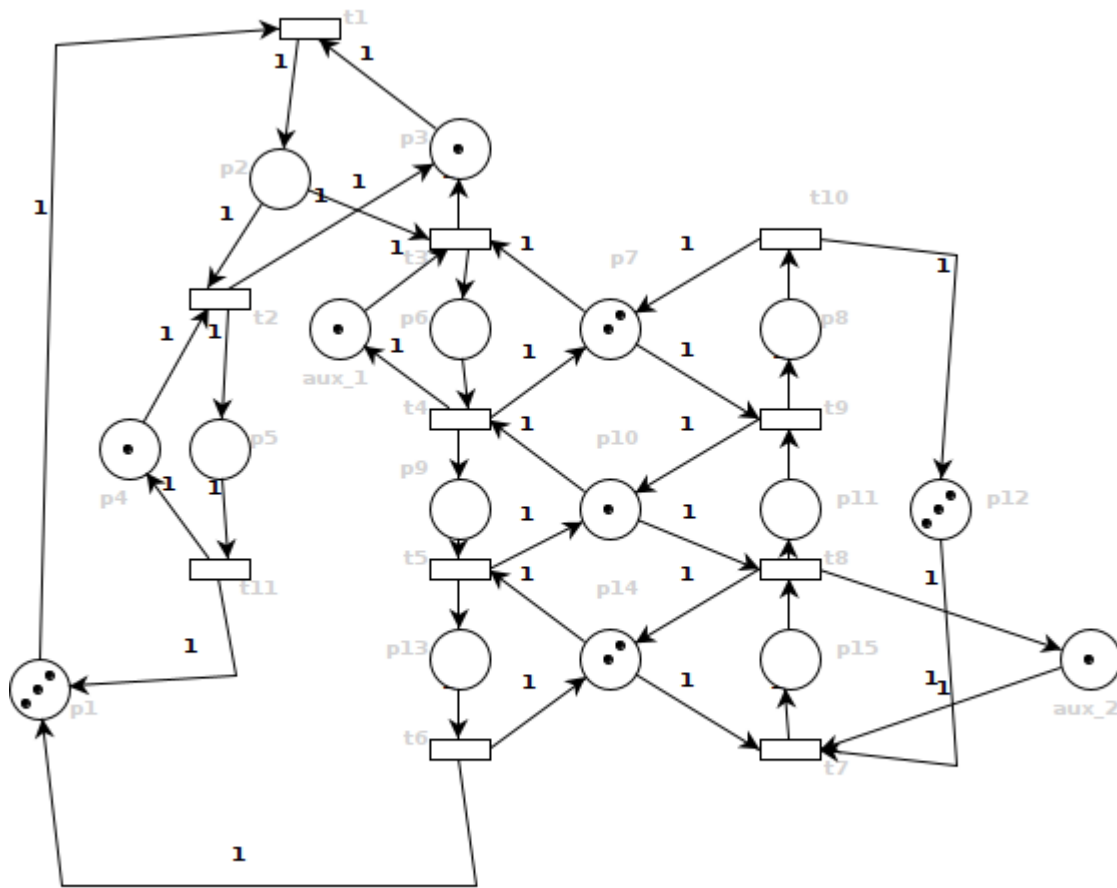
Cabe mencionar que las últimas ecuaciones corresponden a las plazas añadidas sin afectar a las ecuaciones originales. Además es necesario distinguir que AUX_1, P16 y P17 corresponden a las plazas añadidas para evitar el deadlock.

- ANÁLISIS DE CONCURRENCIA:

Marking

	p1	p10	p11	p12	p13	p14	p15	p2	p3	p4	p5	p6	p7	p8	p9	AUX_1	P17	P16
Initial	3	1	0	3	0	2	0	0	1	1	0	0	2	0	0	0	0	0
Current	3	1	0	3	0	2	0	0	1	1	0	0	2	0	0	3	2	2

Para la siguiente solución pensamos que era suficiente eliminar el deadlock limitando la cantidad de tokens de las plazas p6 y p15 al valor de 1 (uno).



- ANÁLISIS DE LOS RESULTADOS DE LA RED

Petri net state space analysis results

Bounded	true
Safe	false
Deadlock	false

- ANÁLISIS DE LOS INVARIANTES:

En la siguiente imagen se puede ver que se mantienen

T-Invariants

t1	t10	t11	t2	t3	t4	t5	t6	t7	t8	t9
1	0	1	1	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	0	0	0
0	1	0	0	0	0	0	0	1	1	1

The net is covered by positive T-Invariants, therefore it might be bounded and live.

P-Invariants

aux_1	aux_2	p1	p10	p11	p12	p13	p14	p15	p2	p3	p4	p5	p6	p7	p8	p9
1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	1	0	0	1	1	0	0	1
0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0

The net is covered by positive P-Invariants, therefore it is bounded.

P-Invariant equations

$$\begin{aligned}
 M(\text{aux_1}) + M(\text{p6}) &= 1 \\
 M(\text{aux_2}) + M(\text{p15}) &= 1 \\
 M(\text{p1}) + M(\text{p13}) + M(\text{p2}) + M(\text{p5}) + M(\text{p6}) + M(\text{p9}) &= 3 \\
 M(\text{p10}) + M(\text{p11}) + M(\text{p9}) &= 1 \\
 M(\text{p11}) + M(\text{p12}) + M(\text{p15}) + M(\text{p8}) &= 3 \\
 M(\text{p13}) + M(\text{p14}) + M(\text{p15}) &= 2 \\
 M(\text{p2}) + M(\text{p3}) &= 1 \\
 M(\text{p4}) + M(\text{p5}) &= 1 \\
 M(\text{p6}) + M(\text{p7}) + M(\text{p8}) &= 2
 \end{aligned}$$

Las primeras dos ecuaciones demuestran que los cambios realizados sobre la red y el efecto que produce a la concurrencia de la misma.

• ANÁLISIS DE CONCURRENCIA:

	aux_1	aux_2	p1	p10	p11	p12	p13	p14	p15	p2	p3	p4	p5	p6	p7	p8	p9
Initial	1	1	3	1	0	3	0	2	0	0	1	1	0	0	2	0	0
Current	1	1	3	1	0	3	0	2	0	0	1	1	0	0	2	0	0

Para hacer este análisis observaremos la capacidad de dar lugar a distintos marcados observando la suma de tokens en las plazas de procesos afectados por la salvada del deadlock (p6,p8,p9,p11,p13,p15) y además cuantos tokens pueden acumularse en el mismo proceso.

En este caso vemos que la suma total de tokens de las plazas afectadas es de 2, con la posibilidad de tener únicamente 1 token por plaza. Por ende esta solución es considerada el peor de los casos.

Aunque el promedio del máximo, es decir el promedio de la cantidad máxima de tokens que puede haber en cada lugar de interés, disminuye al resto de las soluciones.

Para la realización de la siguiente tabla supondremos que esta red está simulando el proceso de una fábrica de café.[1]

TABLA DE ESTADOS DE LA RED ORIGINAL

Lugar	Estado
p1	Personal de producción disponible.
p2	Granos clasificados.
p3	Sifones disponibles.
p4	Máquina incineradora disponible.
p5	Salida de la planta incineradora.
p6	Cereza despulpada.
p7	Máquina despulpadora disponible.
p8	Máquina despulpadora revisada.
p9	Grano fermentado.
p10	Cámara de fermentación disponible.
p11	Cámara de fermentación revisada.
p12	Personal de mantenimiento disponible
p13	Grano lavado.
p14	Máquina de lavado disponible.
p15	Máquina de lavado revisada.

TABLA DE EVENTOS DE LA RED ORIGINAL

Transición	Evento
t1	Clasificar las cerezas de café.
t2	Incinerar
t3	Despulpado
t4	Fermentado
t5	Lavado
t6	Despachado del producto. Regreso del

	personal
t7	Mantenimiento de la máquina de lavado.
t8	Mantenimiento de la máquina de fermentado.
t9	Mantenimiento máquina despulpado.
t10	Fin del proceso de mantenimiento y regreso del personal.
t11	Personal regresando de la planta incineradora.

CAMBIOS A LAS TABLAS SEGÚN LAS SOLUCIONES PROPUESTAS

a continuación se encuentran los cambios realizados a las tablas según corresponda.

Se corresponde con la primera solución al deadlock de la red de petri.

cambios	Estado/ Evento
t7 (transición)	mantenimiento de la máquina de lavado siempre y cuando no hayan granos fermentados.
t3 (transición)	Despulpado siempre y cuando no haya máquinas de fermentación en proceso de mantenimiento.
AUX_1 (lugar)	Accesos disponibles a los procesos de despulpado y mantenimiento de la maquina de lavado

Se corresponde con la segunda solución al deadlock de la red de petri.

Lugar	Estado
p16	Accesos disponibles a los procesos de despulpamiento y mantenimiento de la cámara de fermentación.
p17	Accesos disponibles a los procesos de

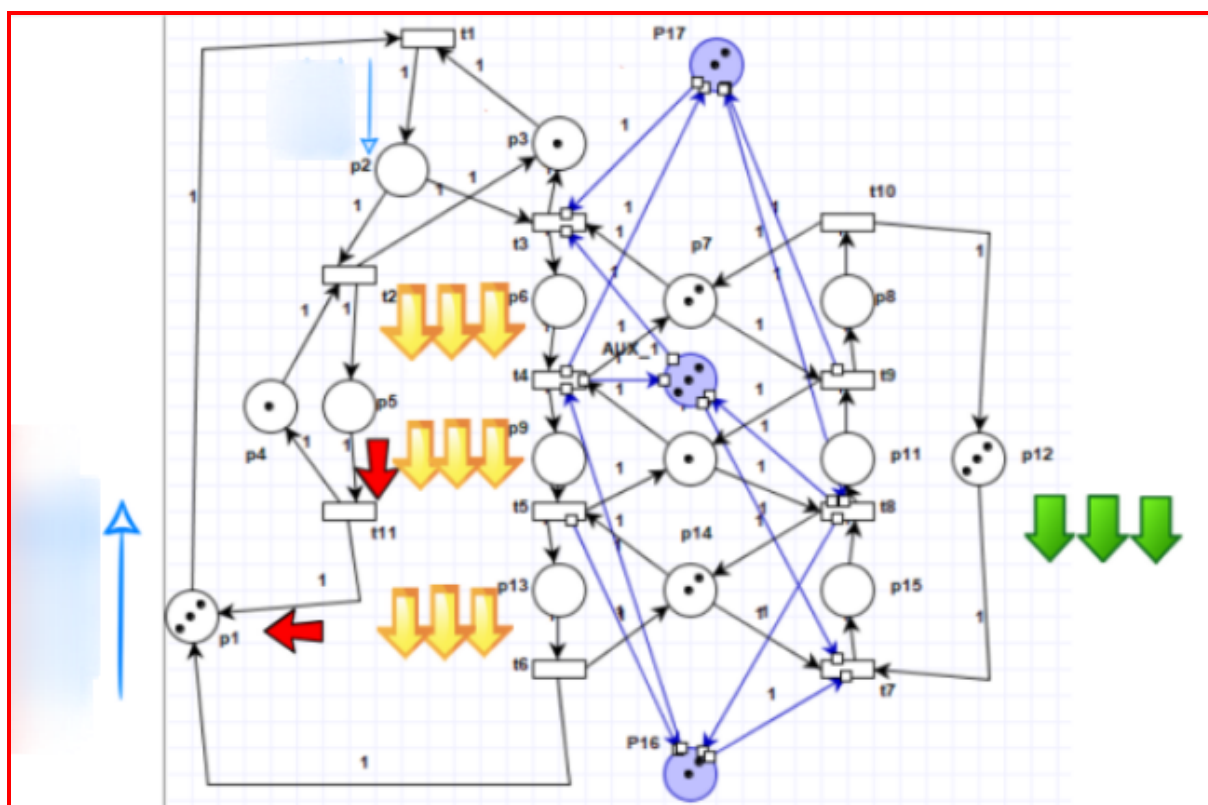
	fermentación y mantenimiento de la máquina de lavado.
AUX_1	Accesos disponibles a los procesos de despulpamiento y mantenimiento de la máquina de lavado.

Se corresponde con la tercera solución al deadlock de la red de petri.

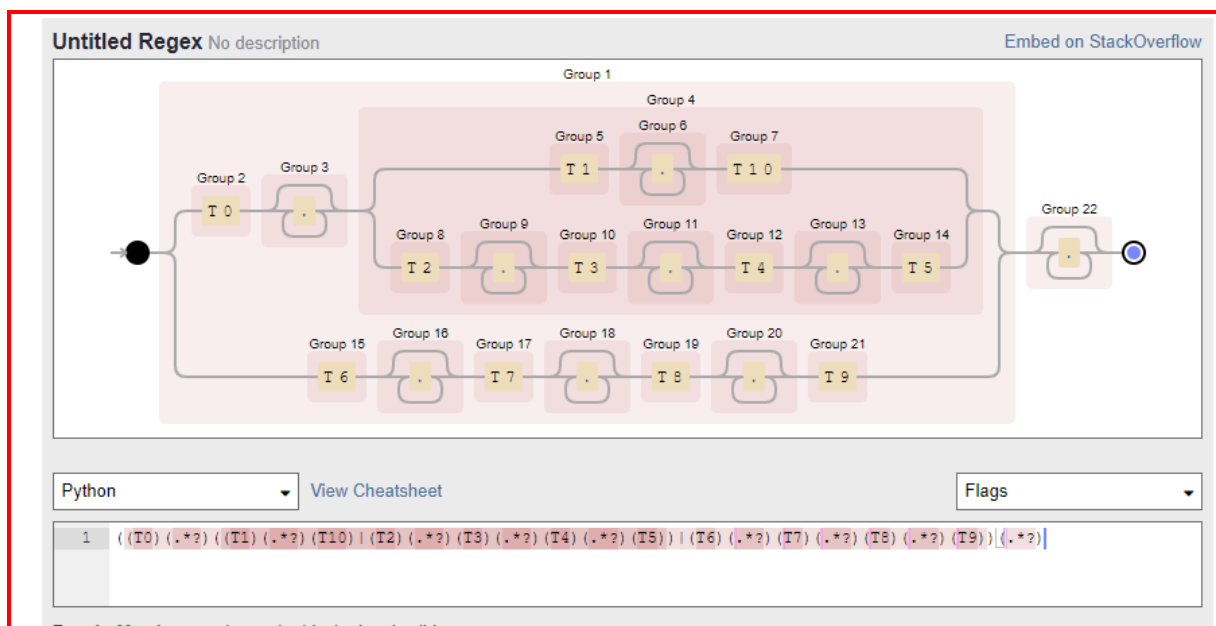
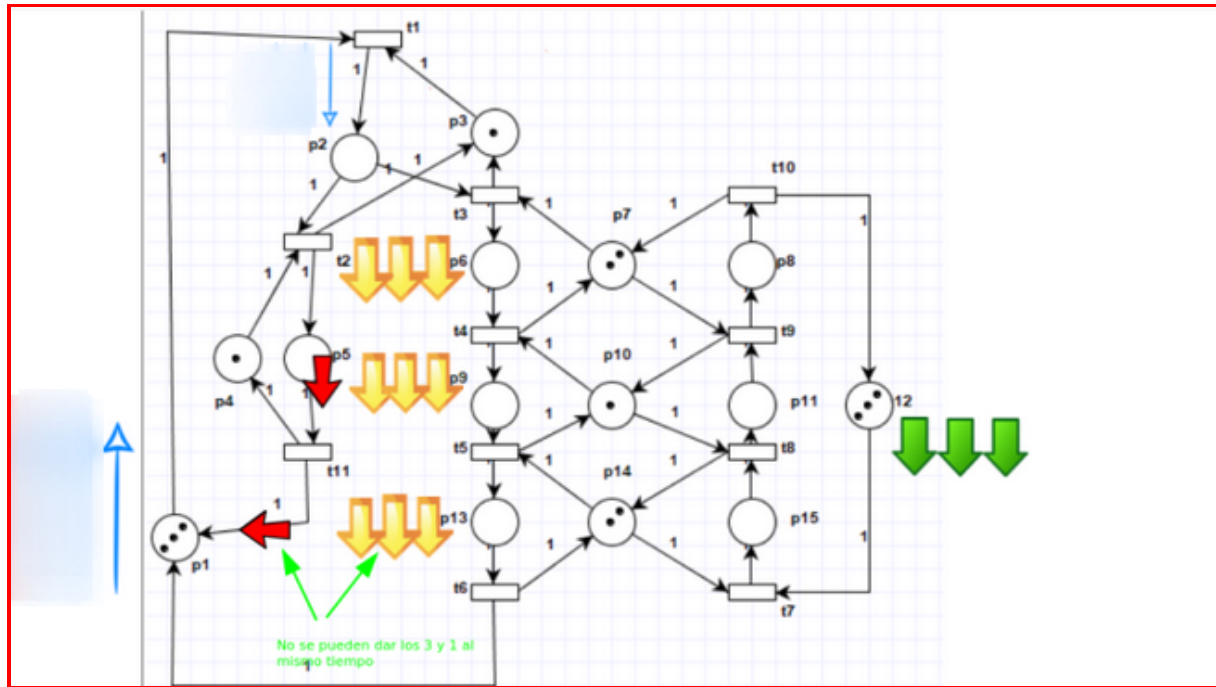
Lugar	Estado
Aux_1	Acceso disponibles al proceso de despulpamiento
Aux_2	Acceso disponibles al mantenimiento de la maquina de lavado

GRÁFICOS DE HILOS

Red sin deadlock (solución final)



Red original



Política

La política es el arte de engañar - Nicolas Maquiavelo

El único requerimiento que se pidió para la política es que al momento de consultarle, esta sea capaz de decidir a una sola opción, es decir, mantener el programa funcionando sin que

intenten ejecutar 2 acciones o más al mismo tiempo, pero al mismo tiempo ninguna. Esto en primera instancia se implementó con una función random, es decir, observaba las trans con la capacidad de disparar y de manera azarosa, devuelve una transición, a la larga esto debía mantener los invariantes sin diferencias entre sí, este debería era lo que a algunos miembros del grupo no les gusto.

¿Qué se hizo entonces? Un sistema de registro donde la política tiene en cuenta cuantas veces se disparó cada invariante, y en base a esto tomar sus propias decisiones, de esta manera nos aseguramos que por criterio de la política, los invariantes se mantengan ejecutándose de manera equitativa.

Monitor

No puedes controlar el viento, pero si la dirección que pones la vela - Randy Gage

El monitor[2] es un área de código, donde la concurrencia se da en este sector del código de manera modularizada. De esta manera se tuvo en cuenta que el monitor (Como clase), debía tener control sobre los elementos que van a ser manejados en sección crítica. En este caso, esto es la red de petri.

Los hilos ingresan al monitor, ven si pueden disparar, y de poder hacerlo, lo intentan, para esto deben estar sensibilizados (no vamos a hablar del tiempo aun). Si es posible, se dispara, sino, se va a una cola de condición de esa transición.

El señalizado de las condiciones se hizo con la política de “Signal and Exit”, de tal forma que cada hilo al terminar de ejecutarse verifica si hay alguna transición para despertar. Todo esto se ejecuta en sección crítica, es decir, de manera atómica, a fines de evitar que más de un hilo modifique la red y genere un problema de concurrencia.

Implementación en Java

El código respeta las convenciones estándares dadas por oracle para la programación en java [3]. Para no extendernos en cuestiones inherentes al código, vamos a hacer hincapié en cuestiones de diseño que se consideren importantes.

Para la implementación en java se aprovechó las virtudes de la orientación a objetos, creando módulos para cada tarea, los cuales se ven y se relacionan en el [diagrama de clases](#). Se aprovecharon las características de abstracción y polimorfismos para implementar objetos dados por java y patrones de diseño, de esta manera minimizar la cohesión y el acoplamiento de clases.

Además se implementaron patrones de diseño:

Factory: Con el fin de abstraer la creación de los “obreros” y los hilos, se implementaron

patrones factory los cuales permiten obtener las instancias deseadas sin la necesidad de hacer uso de “new” de esta forma, generamos una capa de abstracción entre el programa y el main.

Singleton: Dentro del programa hay clases las cuales requieren que solamente exista una instancia del mismo, con este objetivo, implementamos el patrón Singleton, el cual sustituye la llamada a “new” por un “getInstanceOf”. [4]

Observer: A decir verdad, no se usa el patrón en sí. En momentos tempranos del código, se implementó y funcionaba, sin embargo en clase se utilizó una clase llamada “Exchanger” [5], la cual mediante una variable de condición, brindaba un intercambio de mensajes más provechoso y generando una espera activa entre los hilos. En etapas finales del código se decidió agregar una GUI para ver la carga de los invariantes de la ejecución, usando este mismo principio. Quizás en este punto se podría haber aplicado algo más parecido al patrón para priorizar la encapsulación de las clases, sin embargo el uso de exchanger sigue dando un mejor rendimiento en el intercambio de mensajes y de que las salidas del programa no estén trabajando innecesariamente. Quizás si el proyecto necesitara escalar más en este aspecto, sería necesaria una refactorización.

Además de las clases esperables, se crearon 2 clases extras:

Constantes: Donde se colocan los parámetros que se utilizan en el programa, para poder cambiar de manera sencilla si se quieren probar cosas nuevas.

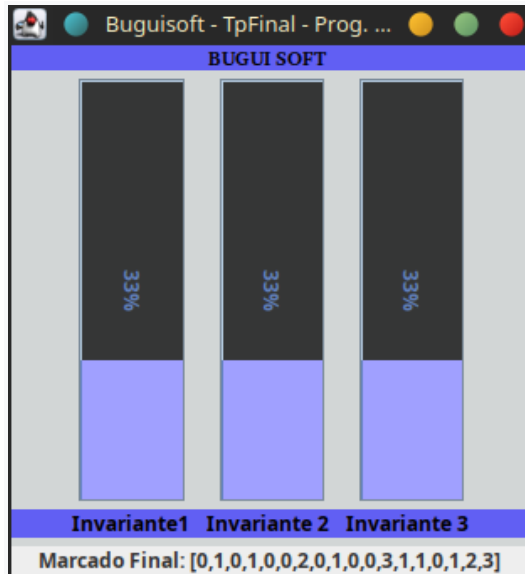
Utilidades: clase de funciones estáticas que no pertenecen a ninguna clase, como productos vectoriales para la ecuación fundamental de la red de petri entre otras funcionalidades. (Viene siendo una forma de implementar la sobrecarga de los operadores de C++, de la cual java carece)

Ya en este punto es posible verificar el funcionamiento del programa, de tal manera que ejecutamos 1000 transiciones sin semántica temporal.

```
{'T0': 671, 'T1': 334, 'T2': 336, 'T3': 336, 'T4': 335, 'T5': 335, 'T6': 335, 'T7': 334, 'T8': 334, 'T9': 334, 'T10': 333}

invariante 1: 333
invariante 2: 335
invariante 3: 334
```

[Análisis de los invariantes hecho con Python]



[GUI del programa en java]

Se ve que la política se mantiene equitativa en su totalidad, el resto de pruebas arrojan resultados similares.

A modo de conclusión, dejo un [diagrama de secuencia](#) conceptual del conjunto de tareas que se lleva a cabo para que un hilo haga un disparo de una transición que se da por hecho que está sensibilizada.

Semántica Temporal

Bien, esta sección fue la que más tiempo llevo. Si bien la idea estaba clara y teníamos los diagramas vistos en clases para guiarnos por donde debían ir los tantos.

En el proyecto se utilizó una semántica temporal de tiempo débil, por lo tanto existe una ventana de tiempo donde la transición está sensibilizada, a partir de un tiempo Alpha hasta un tiempo Beta.

En primera instancia se agregaron alfa y betas al azar a todas las transiciones por igual. Y funcionaban ya que serializabamos la espera de la transición. Con el grupo consideramos que esto era un error ya que debería poder darse la oportunidad de que otra transición intente disparar si su alfa se cumple primero, así que hicimos que se libere el mutex cada vez que una trans esperaba su ventana temporal.

Esto generó otro problema, varios hilos se quedaban esperando por la ventana temporal de la misma transición. Esto se soluciono creando un arreglo que identificaba si había alguien esperando ya por esa transición.

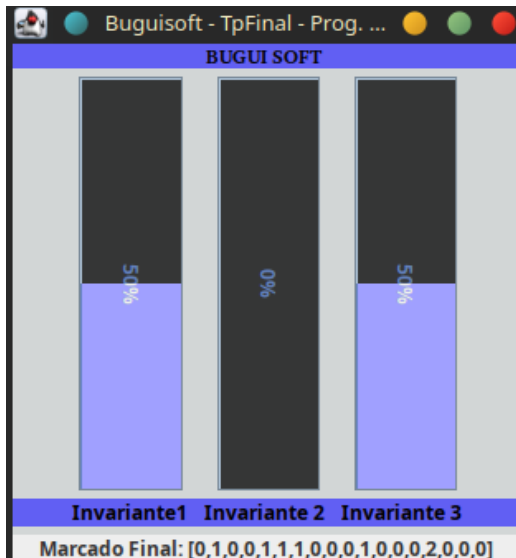
Pero al momento de hacer las pruebas algo andaba mal, obtenemos estos resultados:

```
{'T0': 503, 'T1': 501, 'T2': 1, 'T3': 0, 'T4': 0, 'T5': 0, 'T6': 504, 'T7': 502, 'T8': 501, 'T9': 501, 'T10': 500}
```

```
invariante 1: 500
```

```
invariante 2: 0
```

```
invariante 3: 501
```

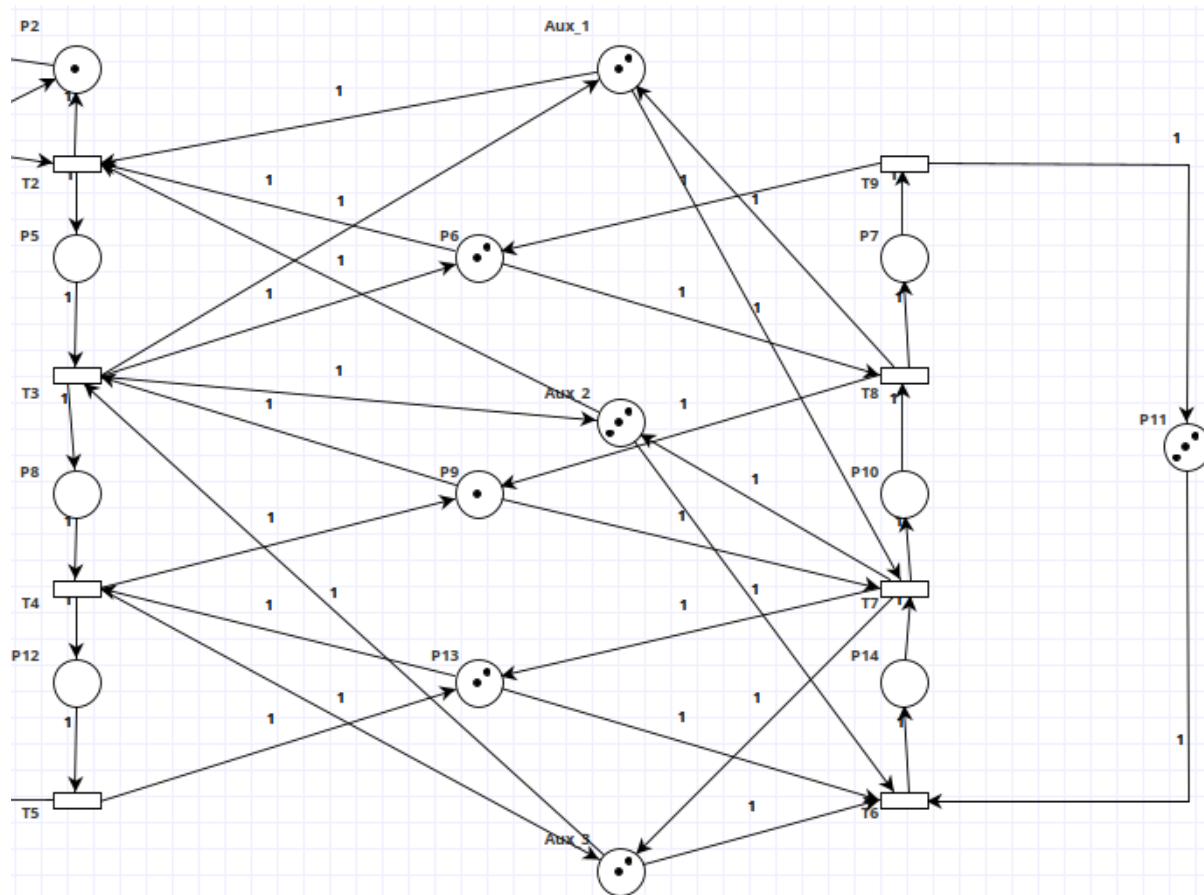


Siendo este el peor resultado posible, había casos donde el invariante 2 se ejecutaba un par de veces, sin embargo, sin ser suficiente para considerar que funcionaba bien.

En ese momento se pensó que estaba habiendo problemas de concurrencia y se implementaron distintos mecanismos para asegurarla, los cuales se especificarán más adelante. De todas formas, la concurrencia funcionaba bien, o eso se podía asegurar más o menos con los resultados que se veían. En ese momento se percató de un detalle; La transición número 3 no se ejecutaba nunca, pero la 2, si. Esto me permitió identificar el problema.

Entonces se reformuló la clase política, podría tratarse que por alguna razón la política no estuviera andando bien, lo cual fue un disparo al aire, la política estaba desacoplada de la semántica temporal y el problema aparece cuando incluía la anterior.

Siguiendo las transiciones 1 a 1 en Pipe, se podía identificar que las transiciones se turnaban de tal manera que no permitían que la trans número 3 se ejecutará, también se pudo ver que siempre se ejecutaba T6 con demasiada frecuencia. Se re analizó la red:



En ese momento se percató de lo siguiente, T6 y T3 compiten recursos, y además T6 no esta temporizada pero T3 sí, concepto que se ignoró, pero que se volvió fundamental después de debuggear mucho tiempo, probando, se temporizo T6 y el programa funcionaba mejor. Acá surge un concepto que en su momento se ignoraba pero se volvería determinante.

“La política decide, pero la semántica temporal va a ser al final quien decida quién disparar”

Esto llevó a distintas pruebas, al principio se creía que mientras más lento sea el invariante 3 y más rápido el 2, intuitivamente mientras más tarda el 3 en sensibilizarse, más tiempo tiene el invariante 2 para ejecutarse. Sin embargo teníamos un funcionamiento del 60%, entre ejecuciones que cumplan el criterio y otra que no. Pero finalmente la idea más determinante fue la siguiente; El problema se da cuando el invariante 3 mantiene mucho tiempo los tokens y el 2 muy poco, entonces la solución es al revés de lo que se planteó antes. Y así fue.

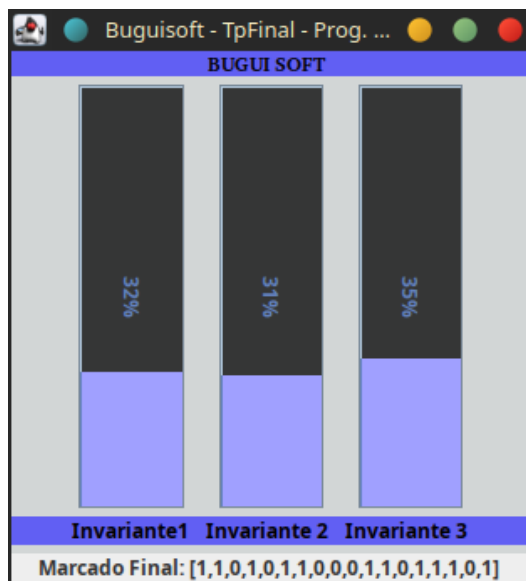
En la pc que se testeo en una pc de 2 cores, en la cual funcionó bien, pero a medida que aumentamos los cores el programa volvió a fallar. Finalmente testeando todo en una pc de 12 cores, pudimos dar con el umbral de los alfas para que el programa ande bien sin importar la cantidad de cores de la pc donde se ejecute, (O eso se intuye).

Para asignar los alphas, ponderamos el tiempo máximo que se le asigna al invariante 2 (Siendo este el más damnificado por el actuar de la semántica temporal). Notamos que el

valor umbral que se le puede asignar, es de 8 milisegundos, a partir de ahí deja de funcionar con la mayoría de las veces. (Notar que esto está más relacionado a la ponderación al asignar los tiempos, y que se podrían hacer otras asignaciones, cambiando este umbral). Al hacer las pruebas de disparo obtuvimos los siguientes resultados:

```
{'T0': 621, 'T1': 312, 'T2': 309, 'T3': 307, 'T4': 306, 'T5': 306,
'T6': 384, 'T7': 383, 'T8': 383, 'T9': 383, 'T10': 312}

invariante 1: 312
invariante 2: 306
invariante 3: 383
```



Viéndose una tendencia al invariante 3 ya que tiene una ventana temporal menor y la Transición 6 que no está temporizada. Pero es un resultado excelente que en contraste con la anterior muestra cómo la semántica temporal afecta a la decisión final de la política que a pesar de intentar mantener los disparos simétricos, no lo logra.

Respecto al Beta, se seleccionó uno de tal manera que no afecte nuestra ejecución y de un riesgo de deadlock, sin embargo, se aseguró que cada tanto se de el evento de que una ventana de tiempo se supere. En ese caso salta el siguiente mensaje:

```
La transición T5 se pasó la ventana de tiempo
```

Y posteriormente la transición se desensibiliza y se coloca en la cola de condición de la transición que se intentó disparar.

Consideraciones

Previamente hablamos de cosas que se agregaron para asegurarnos que la concurrencia funcione.

Se implementó una función que verifique en cada disparo que no se violen los invariantes de plaza, para encontrarnos que es bastante difícil romperlo hasta a propósito.

También se agregó un cortafuego en caso que el mutex se rompa.

Además, al finalizar la ejecución, viendo las transiciones que sobraron, podemos ejecutarlas manualmente en el Pipe a partir del home state y comparar los marcados finales, si estos se cumplen es porque se mantuvieron los invariantes de transición.

También como se vio en las imágenes, se implementó una interfaz más amigable con el usuario (Respecto a la CLI que teníamos hasta el momento de la decisión), que va mostrando la distribución de la carga de los invariantes de manera simple.

Conclusiones

A través de los mecanismos de concurrencia dados a lo largo de la materia podemos asegurar que la concurrencia funciona correctamente.

También cabe nombrar la situación en la cual nos percatamos en una etapa muy avanzada del proyecto que cometimos un error conceptual al asignar las tareas de los hilos, pero que este error era justamente el que hacía que los test pasaran, cuando corregimos ese error, el programa dejaba de funcionar correctamente (Consideramos que fue una falta de comunicación entre los integrantes del grupo lo que hizo que 2 ideas distintas avancen y no se corrijan juntas, dejando una parte del código en mal estado hasta las últimas revisiones). Para corregirlo fue necesario refactorizar varias áreas del código, lo cual dejó bien en claro lo importante que fue crear el código con el menor acoplamiento posible, ya que a pesar de haber sido un cambio importante en muchas líneas, gran parte del proyecto se mantuvo ya que su funcionamiento era independiente de las otras clases.

Otra cosa a tener en cuenta es que el programa se testeó en computadoras de distintas cantidades de núcleos y sistemas operativos, resultó muy ilustrador de cómo cada sistema gestiona de manera diferente sus recursos.

Update #1

Luego de la primera exposición del trabajo se detallaron un conjunto de problemas que algunos eran mínimos y otros se volvían determinantes para verificar el correcto

Código:

El primer problema era la política de señalizado. La cual daba prioridad a la cola de entrada en lugar de a la cola de condición, esta fue una decisión de diseño adrede pero en palabras de los profes, no se puede asegurar el cumplimiento de la política; ya que esta decidía quien se liberaba, pero no se aseguraba que sea la próxima en pasar la cola de entrada.

Otra corrección fueron partes en el código donde el profe no le gustaba la forma que se implementó o sectores que considero que era preferible otro tipo de implementación. Las mismas se analizaron y se decidió mejores alternativas.

Otras correcciones eran cosas que se prefería cambiar y que nos percatamos de ello en los procesos previos a la primera corrección, y se aprovechó esta instancia para ello. Es posible verlos en el [repositorio](#) analizando los commits posteriores al “af88fad74c968a8713bd8f45b6d95ee92fbb48fb”.

Regex y Script:

El script de python dio una respuesta errónea durante la exposición. El error se aisló y se visualiza mejor en la siguiente imagen.

```
> python3.8 data/expresiones_regulares.py

T0 T2 T3 T0 T1 T4 T10

{'T0': 2, 'T1': 1, 'T2': 1, 'T3': 1, 'T4': 1, 'T5': 0, 'T6': 0, 'T7': 0, 'T8': 0, 'T9': 0, 'T10': 1}

invariante 1: 1
invariante 2: 0
invariante 3: 0

Sobranter:
T2 T3 T0 T4
```

El conjunto de transiciones son perfectamente válidas, lo cual indica que el monitor no es culpable del problema. Pero si se ven los sobrantes, se detecta que no es posible esa secuencia ordenada de transiciones ya que previamente a T2 siempre se hay un T0, el cual en la imagen se ve en disparos posteriores. Entonces probablemente había un problema en la Regex. el error se daba de la siguiente manera:

En la red existen 2 invariantes que **comparten la misma transición (T0)**. De modo que los invariantes son T0, T1, T10 conocido como **invariante 1** y T0, T2, T3, T4, T5, conocido como **invariante 2**.

Lo que sucedía en el video es que el último invariante 2, quedó sin terminar, y además se ejecutó un invariante 1 completo, quedando un log de la siguiente forma.

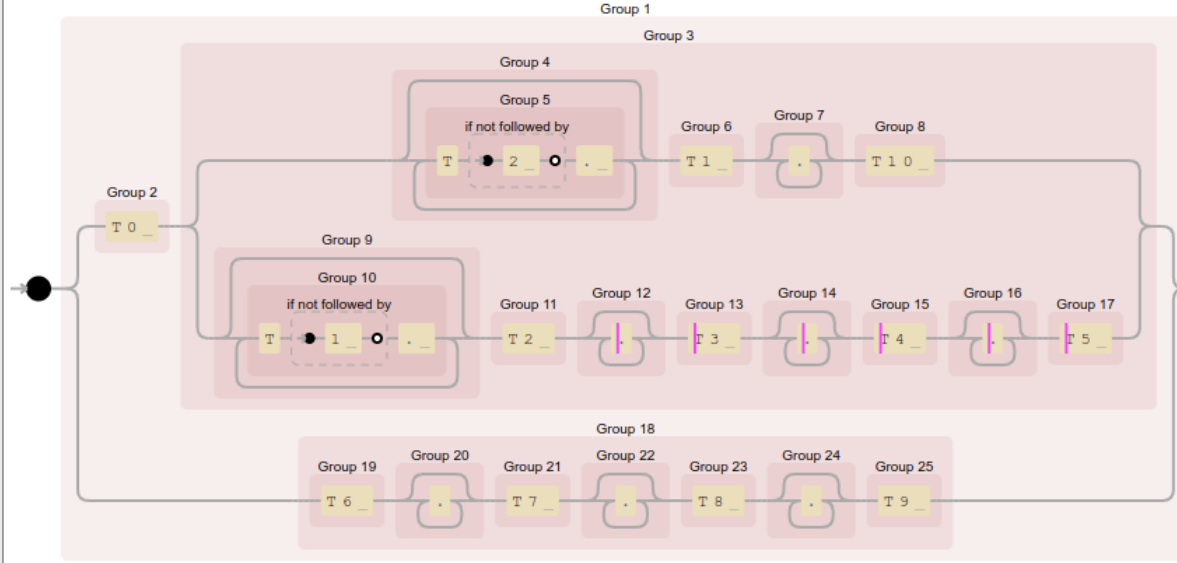
T0 T2 T3 T0 T1 T4 T10

Lo que se ve que sucede, es que al faltar el último T5, la regex interpreta el primer T0 como parte del invariante 1. De tal forma:

T0 T2 T3 T0 T1 T4 T10 -> T2 T3 T0 T4 como sobrantes. (Siendo en ese orden, un conjunto de transiciones invalida)

Entonces, lo que había que revisar era que de alguna manera, el regex identificara que el T0 previo perteneciera a un T1 o a un T2. Eso se logró implementando una herramienta de las regex llamada Lookaround [7]. Siendo la nueva regex, la siguiente.

Group 1



Python ▼
[View Cheatsheet](#)
Flags ▼

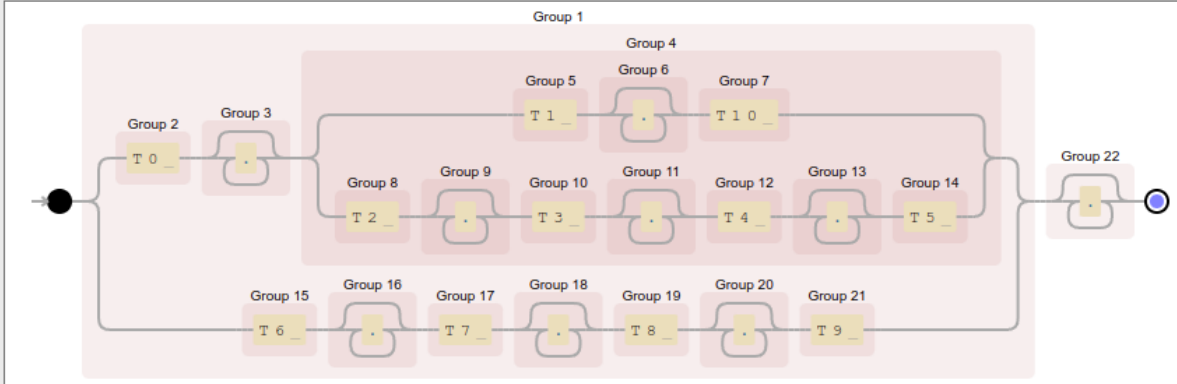
```
1 ((T0 ) ((T(?!2 ) . ) *?) (T1 ) (.*) (T10 ) | ((T(?!1 ) . ) *?) (T2 ) (.*) (T3 ) (.*) (T4 ) (.*) (T5 ) ) | ((T6 ) (.*) (T7 ) (.*) (T8 ) (.*) (T9 ) ) ) (.*)
```

Result: Does not match starting at the black triangle slider

```
1 T0 T2 T3 T0 T1 T4 T10
```

Si comparamos con la regex anterior vemos que:

Group 1



Python ▼
[View Cheatsheet](#)
Flags ▼

```
1 ((T0 ) (.*) (T1 ) (.*) (T10 ) | (T2 ) (.*) (T3 ) (.*) (T4 ) (.*) (T5 ) ) | (T6 ) (.*) (T7 ) (.*) (T8 ) (.*) (T9 ) ) (.*)
```

Result: Matches starting at the black triangle slider

```
1 T0 T2 T3 T0 T1 T4 T10
```

Comparando directamente, se observa que el el script nuevo, posterior a T0 se verifica que no haya nada que corresponda a el otro invariante, de esta forma nos aseguramos que


cada T0 corresponda únicamente a la transición que se le ejecute después. También se ve que el problema se soluciona, comparando los Matches de cada regex.

Diagramas:

Se pidió que se mejore estéticamente el diagrama de clase y se rehaga el diagrama de secuencia, ya que la perspectiva de alto nivel no era la que correspondía a las intenciones de explicar el trabajo.

Se pueden comparar los diagramas rápidamente y ver que se han agregado clases y comportamientos que en el diagrama anterior se consideraban tácitos.

Diagrama anterior:  diagrama_secuencia-Page-1.png

Diagrama Nuevo:  DiagramaDeSecuencia_v3.0.jpg

Referencias

[1] *¿Cómo se produce el café?* (2019, November 4). Universidad del Claustro de Sor

Juana. Retrieved December 29, 2021, from

<https://www.elclauastro.edu.mx/claustronomia/index.php/mundo-foodie/item/400-como-se-produce-el-cafe>

[2] Méndez, J. T. (2001). Cap 6 - Monitor. Programación *Concurrente*. Thomson.

[3] Oracle. (1996, October 4). *Java Code Conventions*. Oracle. Retrieved December 29, 2021, from <https://www.oracle.com/technetwork/java/codeconventions-150003.pdf>

[4] *5 formas de implementar el patrón Singleton en Java*. (2020, November 21). Blog Bitix.

Retrieved December 29, 2021, from

<https://picodotdev.github.io/blog-bitix/2020/11/5-formas-de-implementar-el-patron-singleton-en-java/>

[5] Oracle. (n.d.). *Exchanger (Java Platform SE 7)*. Oracle Help Center. Retrieved

December 29, 2021, from

<https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Exchanger.html>

[6] Petri nets: Structural analysis from ,

<http://www.lsv.fr/~schwoon/enseignement/verification/ws0910/nets2>

[7] *lookaround - Regex lookahead, lookbehind and atomic groups*. (2010, June 4). Stack

Overflow. Retrieved February 27, 2022, from

<https://stackoverflow.com/questions/2973436/regex-lookahead-lookbehind-and-atomic-groups>