Universidad Nacional de Córdoba

Facultad de Ciencias Exactas Físicas y Naturales

FONDO MONETARIO INTERNACIONAL SOFISTICADO

PROGRAMACIÓN CONCURRENTE

Redes de Petri

TRABAJO PRÁCTICO N°2

Autores:

- ❏ Daniele, Francisco || francisco.daniele@mi.unc.edu.ar
- ❏ Endrek, Marco || mendrek@mi.unc.edu.ar
- ❏ Palacios Lucero, Matías Iván || ivan.palacios@mi.unc.edu.ar

Introducción

En este informe se dejará constancia del desarrollo de la solución diseñada para el proyecto propuesto por la cátedra Programación Concurrente de la UNC - FCEFyN como trabajo final del cursado en el año 2021. El mismo es el **enunciado B** y consiste en analizar una red de petri, *mejorarla* y diseñar un monitor de concurrencia para la misma.

Herramientas

Para desarrollar el proyecto se utilizaron las siguientes herramientas:

- PIPE para creación y análisis de Redes de Petri,
- Visual Studio Code y/o IntelliJ como editor de texto,
- Java como lenguaje para modelar el proyecto
- Python como lenguaje para analizar regex

Trabajo Realizado

Red propuesta

La primera parte de este trabajo consiste en analizar la siguiente Red de Petri propuesta, justificando sus propiedades, determinando sus invariantes y realizando una breve descripción de lo que representa. Además debemos, si es que lo tiene, eliminar el deadlock tratando de obtener el mayor paralelismo posible en la red.

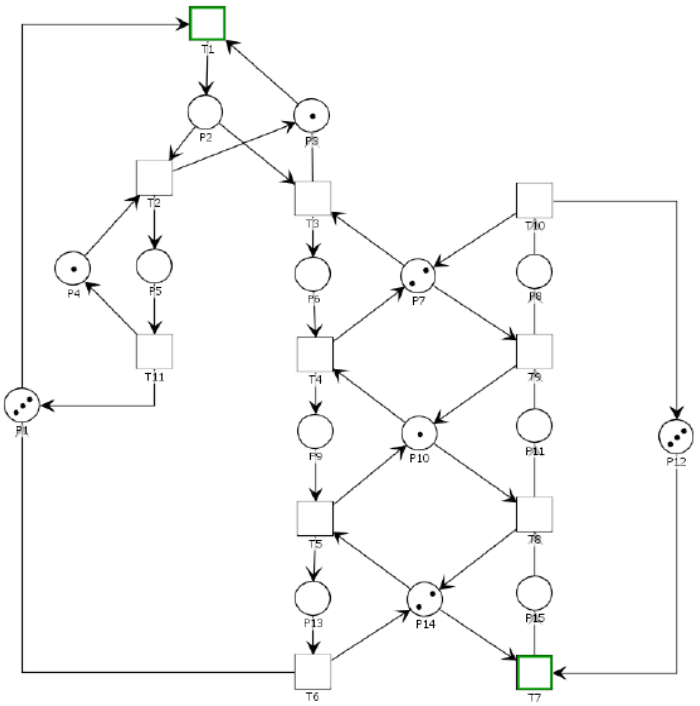


Figura 1

Analizandola con el software PIPE llegamos a lo siguiente:

Propiedades

La RdP no está libre de deadlock. Al tener deadlock, la RdP no es viva.
La RdP está limitada ya que está cubierta por invariantes de plaza positivos, nunca va a existir un lugar que posea más de 3 tokens.
La RdP no es segura, nos damos cuenta viendo la marca inicial m_0 dónde ya se observan lugares con más de 1 token.
La red es consistente ya que existe al menos una secuencia de disparos que contenga todas las transiciones y que comenzando desde el marcado inicial, vuelva al mismo marcado luego de dispararse todas las transiciones de dicha secuencia.

Petri net state space analysis results

Bounded	true
Safe	false
Deadlock	true

Shortest path to deadlock: T1 T2 T1 T3 T1 T11 T3 T1 T2 T4 T11 T1 T3 T5 T4 T5 T4 T6 T1 T2 T5 T11 T1 T6 T2 T1 T3 T11 T1 T7 T2 T11 T4 T1 T2 T6 T1 T11 T3 T1 T2 T7 T11 T1 T3

Invariantes

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.

Figura 2

P-Invariants

P1	P10	P11	P12	P13	P14	P15	P2	P3	P4	P5	P6	P7	P8	P9
1	0	0	0	1	0	0	1	0	0	1	1	0	0	1
0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	1	0	0	1	0	0	0	0	0	0	1	0
0	0	0	0	1	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	1	1	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

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

Figura 3

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

Figura 4

Conclusiones

Según el análisis realizado anteriormente podemos realizar la siguiente descripción sobre la RdP:

- Las plazas {P3, P4, P7, P10, P14} representan recursos compartidos en el sistema. Serían por ejemplo, máquinas de una fábrica que se pueden utilizar para un proceso a la vez. Estas limitan la cantidad de hilos que pueden estar realizando las tareas que representan las plazas que están encerradas entre las transiciones que quitan y ponen tokens a las mismas.

- Las plazas {P1, P12} son plazas idle, que corresponden a buffers del sistema. Cuando un token (hilo) está allí, se encuentra esperando poder disparar la primera transición de un invariante para comenzar a realizar tareas.
- El resto de las plazas son plazas donde se realizan actividades relacionadas con el proceso.
- Analizando los invariantes de plaza (fig. 3):
    - El primero determina que entre la plaza idle (1) y {P2, P5, P6, P9, P13}, las cuáles representan la realización de procesos que harán los hilos de P1, siempre habrá 3 hilos.
    - El segundo nos dice que cómo P10 representa un único recurso compartido, las tareas que representan P9 y P11 se realizan en exclusión mutua.
    - El tercero nos dice que como hay 3 tokens en la plaza idle P12, entre esta y {P8, P11, P15} donde los hilos realizan tareas siempre estarán esos 3 tokens.
    - El cuarto y el último son similares, nos indican que se poseen 2 tokens de recurso compartido por lo que las únicas opciones son: o qué ningún recurso esté siendo utilizado, que uno sólo sea utilizado por una plaza de alguno de los dos caminos, o que los 2 recursos sean utilizados por la plaza de un camino o uno usado por un camino y el otro por el restante.
    - El quinto determina que sólo habrá un hilo a la vez “decidiendo” qué conjunto de tareas realizará (conflicto en P2). Si las del invariante T1T2T11 o del invariante T1T3T4T5T6.
    - El sexto determina que al P4 representar un recurso, sólo podrá haber 1 hilo realizando la tarea de P5 a la vez (sección crítica).
- Observamos que tiene 3 invariantes de transición (secuencias de disparo que se pueden repetir y te llevan al estado inicial) que representan una secuencia de tareas que realizaría el sistema que modela la Red.
    - Invariante T1T2T11: este no conlleva problemas ya que no comparte recursos con otro invariante del sistema.
    - Invariantes T1T3T4T5T6 y T7T8T9T10: estos invariantes comparten recursos y realizan una tarea en exclusión mutua. Aquí observamos que mientras haya tokens en uno sólo de los caminos, no hay problema. Pero cuándo en la plaza 6 (o 15) hay 2 tokens y en la 11 (o 9) hay uno realizando la tarea, se interbloquean estos 2 invariantes y no se pueden disparar más. A partir de ahí, dependiendo del interleaving, puede quedarse disparando infinitamente el invariante T1T2T11 o si todos los tokens de P1 pasaron a estar en las plazas 6, 9 o 13 se llega al deadlock.

Eliminación del deadlock

Para poder seguir con el trabajo había que averiguar el origen del deadlock y encontrar una manera eficiente de eliminarlo. Para esto fuimos probando diferentes cosas, siempre agregando plazas de control en distintos puntos para mantener lo más simple y entendible para nuestro criterio (evitamos uso de brazos inhibidores para no agregar elementos y complejidad a la red), empezando muy restrictivamente hasta poder llegar a una RdP con el mayor paralelismo que pudimos.
A continuación describimos cómo se fue llegando a la red final:

Lo primero que hicimos fue agregar P16 con 2 tokens limitando así que entre las plazas que encierran los invariantes conflictivos solo pueda haber 2 hilos trabajando. Esto, si bien soluciona el deadlock, no es eficiente ya que secuencializa mucho el trabajo.

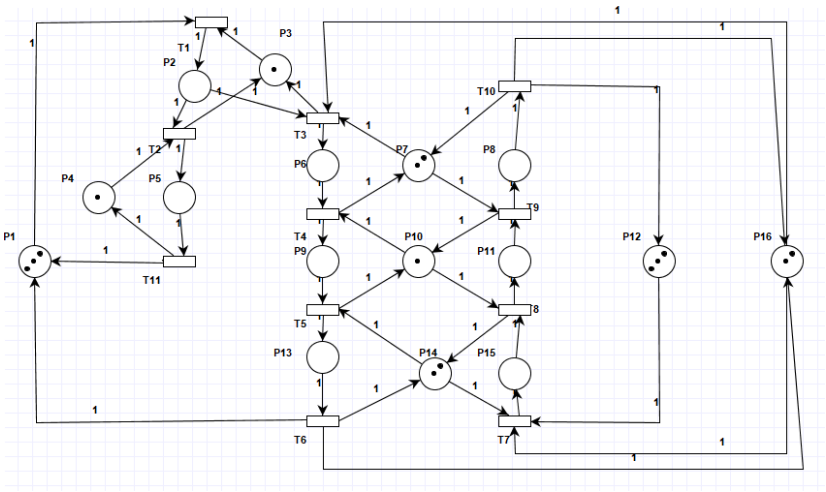


Figura 5

Luego de seguir probando, llegamos a las siguientes 2 redes similares, las cuales no limitan la cantidad de hilos trabajando en {P6, P9, P13, P8, P11, P15} (están sólo limitados por los recursos iniciales, tal cómo la red inicial de la consigna). Tienen una ligera diferencia, ya que una limita a los lugares de “entrada” P6 y P15 a un hilo a la vez y la otra sólo hace esto con P15.

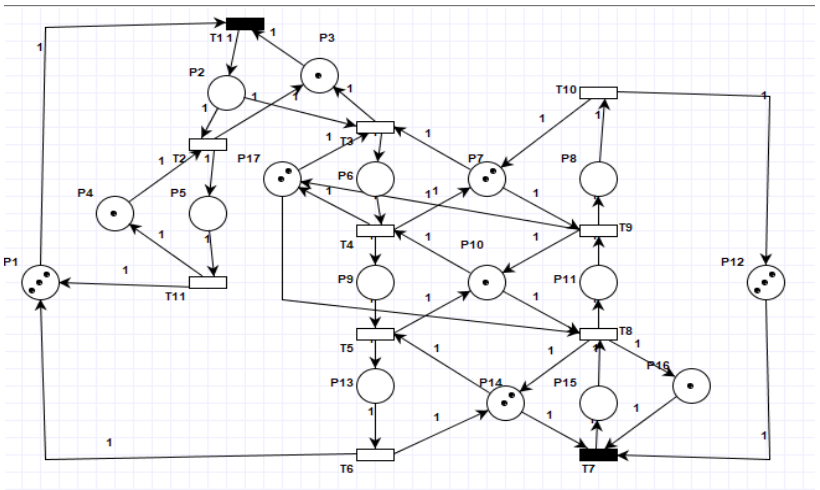


Figura 6

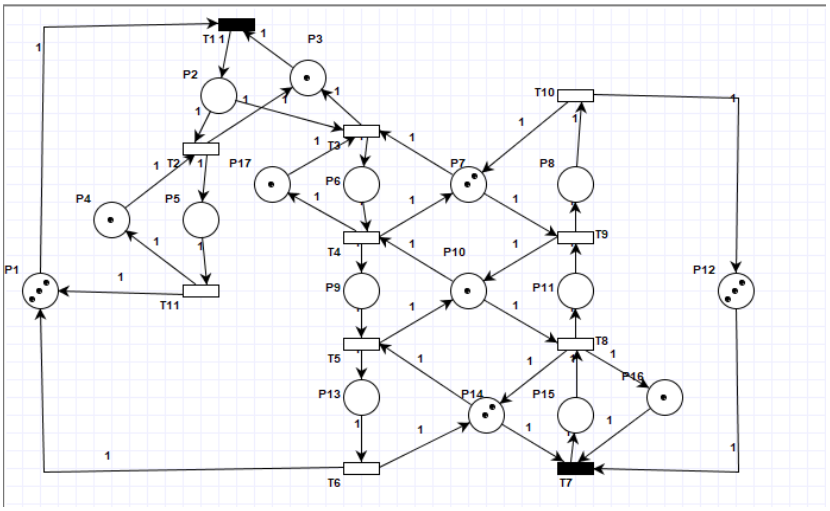


Figura 7

Elección de la red

Luego realizamos un análisis mediante PIPE comparando estas dos redes para determinar cuál permitía más paralelismo y en base a esto tomar la decisión de con qué red continuar el trabajo:

	Red fig. 6		Red fig. 7	
Marcados		58		54
Promedio de tokens por plaza de interés según los diferentes marcados				
P1		0,444		0,567
P12		1,549		1,327
P2		0,582		0,697
P5		0,446		0,536
P6		0,886		0,695
P8		0,236		0,363
P9		0,331		0,250
P11		0,272		0,391
P13		0,311		0,255
P15		0,943		0,919
Suma plazas IDLE		1,993		1,894
Suma plazas de tareas		4,007		4,106

Tabla 1

A partir de la tabla anterior concluimos que la Red de Petri de la figura 7 es ligeramente “mejor” ya que aporta mayor paralelismo que la otra. Esto se observa al realizar el análisis del promedio de tokens por plaza por marcado, observamos que la suma de tokens promedio en las plazas idle (dónde no “trabajan”) es menor en dicha red, lo que implica (y se observa en la tabla) que la de las plazas no marcadas que no son recursos, y por lo tanto representan tareas, es mayor.

En base a este análisis, decidimos elegir la Red de Petri de la figura 7 para realizar la siguiente parte del trabajo.

Red solución

A partir del análisis de la red de la figura 7 mediante PIPE, observamos lo siguiente:

Propiedades

La RdP ahora está libre de deadlock y por lo tanto es viva.
La RdP sigue siendo limitada ya que está cubierta por invariantes de plaza positivos, nunca va a existir un lugar que posea más de 3 tokens, por esto también se mantiene no segura, con lugares con más de 1 token.
La red mantiene la propiedad de consistencia.

Invariantes

Como se pide en la consigna, esta nueva red mantiene sus invariantes de transición.

P-Invariants

P1	P10	P11	P12	P13	P14	P15	P16	P17	P2	P3	P4	P5	P6	P7	P8	P9
1	0	0	0	1	0	0	0	0	1	0	0	1	1	0	0	1
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	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	0	1	0	0	0	0	1	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	0	0	1	1	1	0

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(P15) + M(P16) = 1
M(P17) + M(P6) = 1
M(P2) + M(P3) = 1
M(P4) + M(P5) = 1
M(P6) + M(P7) + M(P8) = 2

Figura 8

Clasificación

La clasificación de esta RdP sigue siendo la misma que la anterior, observada en fig. 4.

Conclusiones

Como hicimos con la red original, a partir del análisis realizado por software de la nueva RdP, procedemos a una breve descripción de la misma:

- Las plazas {P3, P4, P7, P10, P14} siguen representando recursos compartidos en el sistema y limitan la cantidad de hilos que pueden estar realizando las tareas que representan las plazas que utilizarían los tokens de estas plazas.
- Las plazas {P1, P12} son plazas idle, que corresponden a buffers del sistema.
- Las plazas {P16, P17} son plazas de control, utilizadas para eliminar el deadlock original, las cuáles limitan a 1 los tokens en las plazas {P6, P15}.
- El resto de las plazas son plazas donde se realizan actividades relacionadas con el proceso.
- Analizando los invariantes de plaza (fig. 8) vemos que se mantienen los mismos 7 invariantes de plaza de la red original y además aparecen 2 nuevos de similar significado (capacidad finita):
    - M(P16) + M(P15) = 1. Determina que en P15 sólo podrá haber un hilo realizando la tarea.
    - M(P17) + M(P6) = 1. En P6 sólo podrá haber un hilo realizando la tarea.
- Se mantienen los mismos 3 invariantes de transición, pero esta vez entre ellos no hay problemas de concurrencia que originen un deadlock.

Criterio de hilos

En la figura 9 se determinan los hilos necesarios para la ejecución del sistema con el mayor paralelismo posible. Las flechas indican hilos. A continuación se indica el por qué de la decisión:

- Hilo amarillo: P3 me limita el disparo de T1 de a un hilo por vez, por lo que sólo necesitaría uno que se encargue del disparo de esa transición. Cómo en P2 hay un conflicto, debemos dejar la tarea de ejecutar las siguientes transiciones a otros hilos.
- Hilo naranja: P4 limita a 1 la cantidad de hilos que pueden disparar T2 y T11, por lo que habrá uno solo encargado de realizar la tarea de P5 y disparar dichas transiciones.
- Hilos rojos: la red puede tener hasta 3 hilos realizando simultáneamente tareas en P6, P9 y P13 por lo que encargamos a 3 hilos distintos la tarea de disparar las transiciones T3, T4, T5 y T6.
- Hilos azules: es el mismo caso que el anterior, puede haber hasta 3 hilos realizando tareas en P8, P11 y P15 por lo que encargamos a 3 hilos distintos el disparo de T7, T8, T9 y T10.

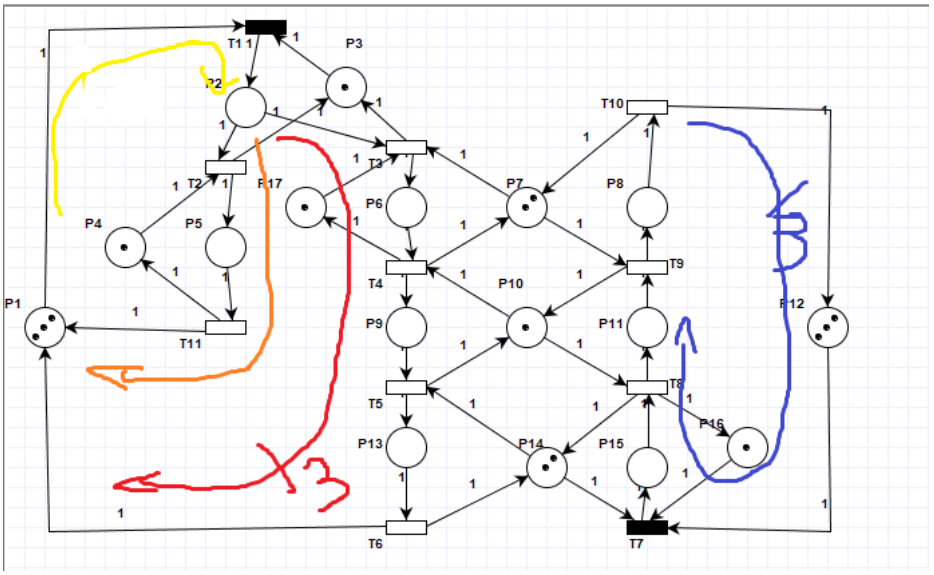


Figura 9

Tabla de estados

Aquí se muestran las plazas que componen la red, indicando el estado que representa un token en cada plaza.

P1	Actividades disponibles
P2	Trabajador decidiendo hacer actividad de pasante u operario
P3	Sección crítica disponible
P4	Sección crítica disponible
P5	Pasante realizando su actividad
P6	Operario (izquierda) realizando parte de su actividad
P7	Recurso disponible
P8	Operario (derecha) realizando parte de su actividad
P9	Operario (izquierda) realizando parte de su actividad
P10	Recurso disponible
P11	Operario (derecha) realizando parte de su actividad
P12	Actividades disponibles
P13	Operario (izquierda) realizando parte de su actividad
P14	Recurso disponible
P15	Operario (derecha) realizando parte de su actividad
P16	Sección crítica disponible

P17	Sección crítica disponible

Tabla 2

Tabla de eventos

En esta tabla a continuación, se enumeran las distintas transiciones de la red y el evento que supone el disparo de cada una.

T1	Capataz ordena comenzar una actividad del flujo izquierdo de trabajo
T2	Pasante comienza actividad
T3	Operario (izquierda) comienza actividad
T4	Operario (izquierda) completa parte de su actividad
T5	Operario (izquierda) completa parte de su actividad
T6	Operario (izquierda) completa su actividad
T7	Operario (derecha) comienza actividad
T8	Operario (derecha) completa parte de su actividad
T9	Operario (derecha) completa parte de su actividad
T10	Operario (derecha) completa su actividad
T11	Pasante completa su actividad

Tabla 3

Monitor

Para manejar la concurrencia que trae aparejado el avance del marcado de la red mediante los disparos de las transiciones, y como parte de los requerimientos del trabajo, se diseñó un monitor de concurrencia.

El mismo cuenta con su constructor, las variables locales que almacenan el estado del recurso que maneja (la RdP), procedimientos internos que manejan las variables permanentes, por ej. las colas de condición, y procedimientos exportados a los que acceden los hilos activos (cómo el método que actualiza el marcado de la red).

Para su implementación decidimos aplicar la política *“Signal and Exit”* la cuál consiste en que el proceso que señaliza sale del monitor y el despertado toma el mutex para ejecutar el método.

El monitor dispone de semáforos para controlar la concurrencia. Uno controla la entrada al método crítico del monitor, y los demás (en este caso 11) se utilizan para las colas de condición de las transiciones.
El mutex principal del monitor controla la entrada al método *disparar* el cuál es el que realiza la actividad principal de los procesos: intentar cambiar el estado de la Red de Petri controlada por el monitor. Este semáforo se inicializa en 1, indicando un solo recurso. Cuándo un proceso quiere disparar una transición, trata de adquirir dicho semáforo y si la sección crítica está ocupada se pone en la cola. Caso contrario entra en la sección crítica, intenta disparar, despierta a otro proceso durmiendo en las colas de condición y finalmente sale de la sección crítica.
Los semáforos de las colas de condición se inicializan en 0, para así hacer entrar a la cola a los procesos que necesiten esperar a que se cumpla su condición para disparar una transición. Estos semáforos incorporan hilos a su cola en el método *dormir*, al cuál ingresan los hilos que no pudieron cambiar el estado de la red. Para liberar a los mismos, en el método *signal,* al cuál ingresan todos los hilos que están en la sección crítica del monitor, se fijan si hay alguien esperando su condición y por medio de la política de prioridades se despierta a un hilo de una cola.

Política

Este apartado refiere a la política de “despertado de hilos en las colas de condición”. Con esta se busca una manera de asignar prioridades sobre que hilo despertar cuándo hay más de uno esperando y así poder cumplir con la consigna de que la cantidad de invariantes completados sea pareja.
Así es que para implementar la política diseñamos dos algoritmos distintos, uno más simple y otro un poco más complejo.
El primero un algoritmo que consiste en revisar quiénes pueden ser despertados y dar prioridades en este orden:

1. T1 que representa la entrada al conflicto {P2, T2, T3}.
2. T3, T4, T5 y T6 que corresponden al invariante de los operarios de la izquierda.
3. T2 y T11, transiciones del invariante del pasante.
4. T7, T8, T9 y T10, correspondientes al último invariante.

Esto es así ya que al haber 3 tokens para los invariantes {T1, T3, T4, T5, T6} y {T1, T2, T11} y otros 3 para {T7, T8, T9, T10} se observa que el 50% aprox. de los invariantes completados correspondían al último y el 50% restante se dividían entre los 2 primeros con mayor parte de {T1, T2, T11}.
El otro algoritmo selecciona el hilo a despertar teniendo en cuenta el invariante al que pertenece esa transición y la cuenta de invariantes completados de cada tipo hasta el momento. El hilo que se despierta será el que quiera disparar una transición del invariante menos completado hasta el momento. Este muestra resultados más precisos por lo que es el seleccionado para manejar la política del monitor.

Tiempo

Luego de tener funcionando el monitor de concurrencia, procedimos a implementar el sistema de temporización de las transiciones.
Por esto creamos una clase llamada *Tiempo* la cual utiliza 5 vectores para controlar qué transiciones son temporizadas y su correspondiente tiempo:

- *transicionesTemporizadas* indica que transiciones son temporizadas
- *alfas* para el tiempo mínimo y *betas* el tiempo máximo de disparo de cada transición.
- El vector *tiempoDeSensibilizado* almacena el momento en que se sensibilizó c/ transición.
- Y el último es el vector *q* de las redes de petri, que almacena en cada instante cuánto tiempo pasó desde que la transición está sensibilizada y permite conocer si se encuentra en su ventana temporal para dispararse o no.

Para su funcionamiento cuenta con distintos métodos:

- *resetTiempo:* resetea el tiempo correspondiente a una transición en los vectores *q* y *tiempoDeSensibilizado*
- *actualizarQ*: actualiza el valor del vector *q* de una transición
- *sensibilizarTiempo*: marca el momento en que se sensibilizó una transición
- *enVentana:* indica si el valor de *q* de la transición está dentro de su correspondiente valor en *alfas* y *betas*, es decir puede dispararse
- *calcularTiempoRestante:* retorna cuánto le falta a la transición alcanzar su valor en *alfas* para así indicarle al hilo que la quiera disparar que espere el tiempo necesario.

Con esto incorporado al sistema, cada vez que un hilo quiere disparar una transición se evalúa si esta es temporizada y si es así, si se encuentra en su ventana temporal. Luego si no se pudo disparar por estar sensibilizada pero no dentro de su ventana, el hilo se va a dormir el tiempo necesario hasta llegar a su ventana. Para poder conocer estas cuestiones, cada vez que se le pide a la red que calcule sus transiciones sensibilizadas esta se fija si hubo cambio de estado en la transición:

- Si la transición no estaba sensibilizada y ahora si, se marca el momento en que sucedió.
- En el caso contrario al anterior, se dispara el método *resetTiempo.*
- Por último, si la transición estaba sensibilizada y sigue de la misma manera, se actualiza su valor en el vector *q.*

Resultados

A continuación se adjuntan imágenes correspondientes al resultado de distintas ejecuciones del programa, diferenciando ejecuciones con transiciones con y sin semántica temporal.
Se observa cómo se completan los 1000 invariantes la mayor parte del tiempo, con algunas excepciones en que algunos hilos quedan esperando en la cola de condición de alguna transición que nunca más se sensibilizará debido a que los demás hilos completaron su ejecución. También se puede ver cómo en todos los casos hay un balance equilibrado entre la cantidad de veces que se dispara cada invariante, lográndose la consigna propuesta para el trabajo.

Red sin temporizar

```
----------------INVARIANTES-----------------------
Invariante que termina en T6: 333 (33,33%)
Invariante que termina en T10: 333 (33,33%)
Invariante que termina en T11: 333 (33,33%)
------------------MAIN END-----------------------

El fin de la jornada laboral encontró a TRABAJADOR 3 durmiendo.
```

```
----------------INVARIANTES-----------------------
Invariante que termina en T6: 333 (33,30%)
Invariante que termina en T10: 334 (33,40%)
Invariante que termina en T11: 333 (33,30%)
------------------MAIN END-----------------------
```

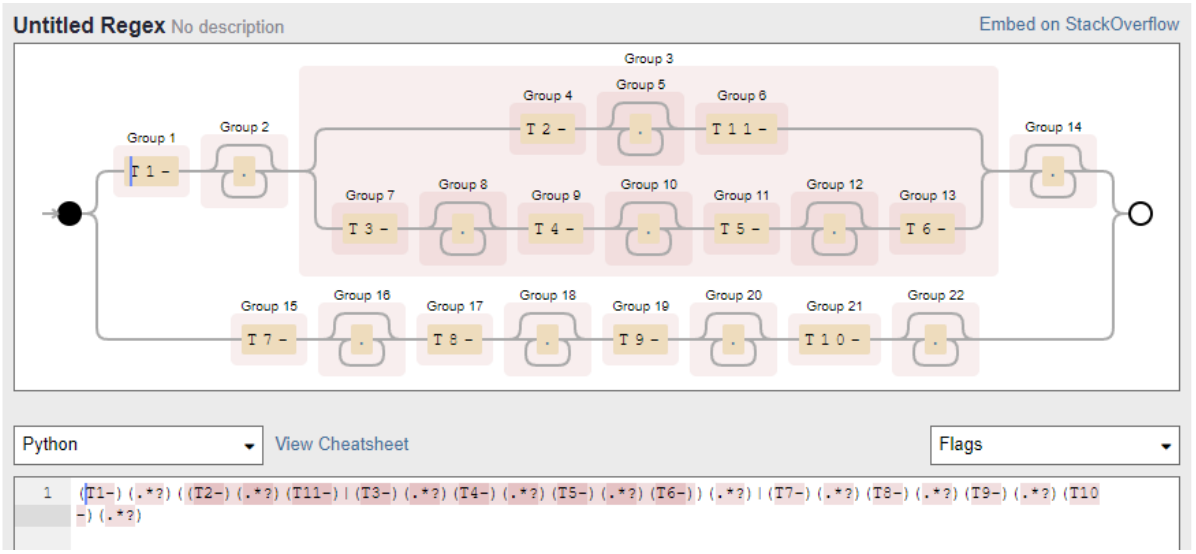
Red temporizada

```
----------------INVARIANTES-----------------------
Invariante que termina en T6: 332 (33,30%)
Invariante que termina en T10: 332 (33,30%)
Invariante que termina en T11: 333 (33,40%)
------------------MAIN END-----------------------

El fin de la jornada laboral encontró a TRABAJADOR 1 durmiendo.

El fin de la jornada laboral encontró a TRABAJADOR 4 durmiendo.
```

Regex

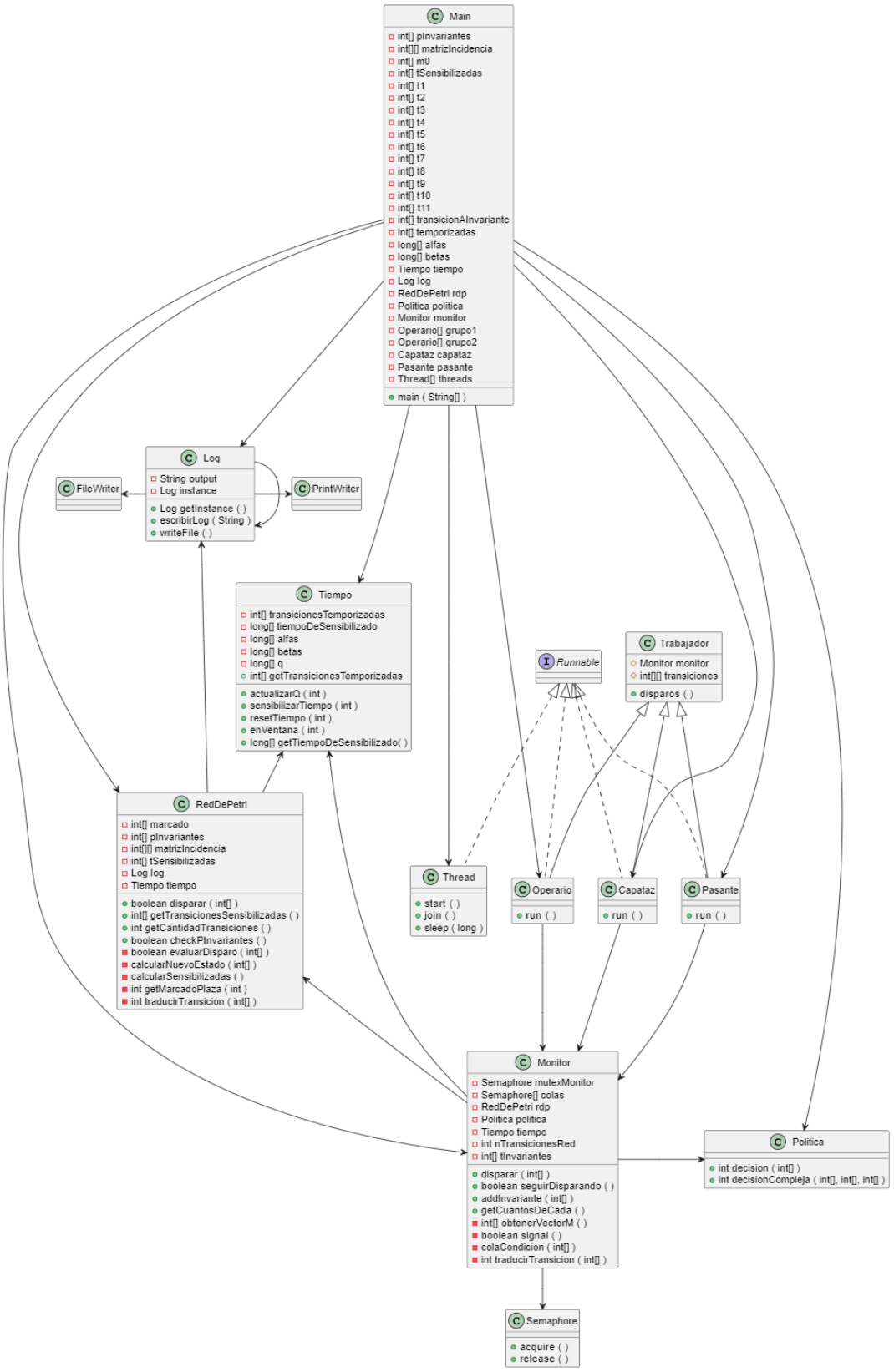
Para verificar el correcto funcionamiento del sistema, se pedía que el programa cree un *log* donde se impriman las transiciones en el orden que se van disparando y luego analizarlo mediante expresiones regulares para verificar que se cumplieron los invariantes de transiciones.

Para esto se desarrolló un script en *python* el cuál lee el archivo log, generado por el sistema como output de una ejecución, y luego mediante una expresión regular que detecta los 3 posibles invariantes de transición de la RdP los va “sacando” hasta llegar a analizar todo el texto. Si al final sólo quedan secuencias de disparos pertenecientes a alguno de los invariantes o nada, quiere decir que el sistema funciona correctamente.


Untitled Regex No description
Embed on StackOverflow
Group 1
T 1 -
Group 2
.
Group 3
Group 4
T 2 -
Group 5
.
Group 6
T 1 1 -
Group 7
T 3 -
Group 8
.
Group 9
T 4 -
Group 10
.
Group 11
T 5 -
Group 12
.
Group 13
T 6 -
Group 14
.
Group 15
T 7 -
Group 16
.
Group 17
T 8 -
Group 18
.
Group 19
T 9 -
Group 20
.
Group 21
T 1 0 -
Group 22
.
Python
View Cheatsheet
Flags
1
(T1-)(.*?)((T2-)(.*?)(T11-)|(T3-)(.*?)(T4-)(.*?)(T5-)(.*?)(T6-))(.*?)|(T7-)(.*?)(T8-)(.*?)(T9-)(.*?)(T10-)(.*?)

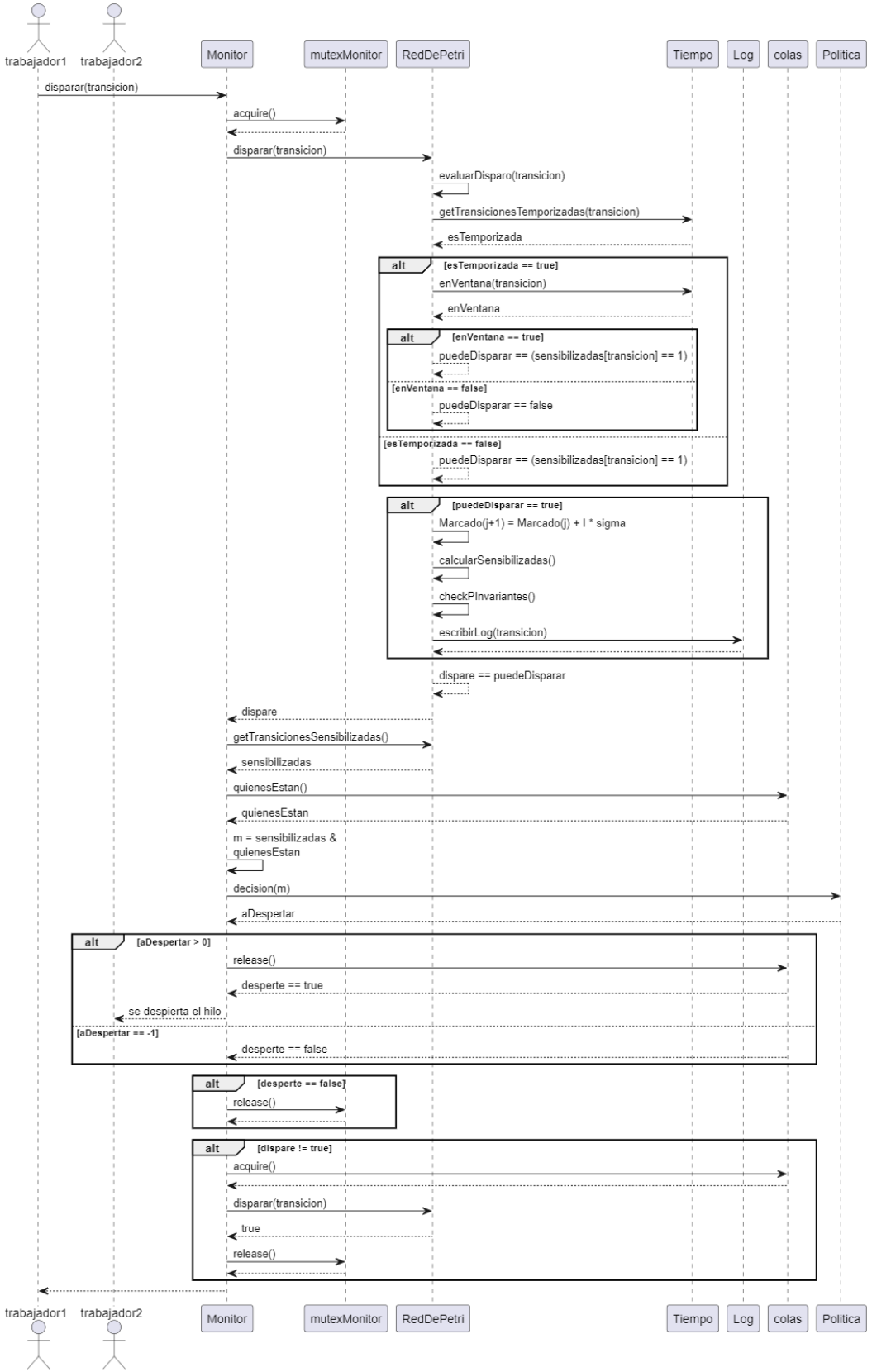
Class Diagram

A continuación se adjunta una imágen del diagrama UML de clases del sistema generado para la solución del problema propuesto. Allí se observan todas las clases con sus variables y métodos, y las relaciones entre las mismas.



Sequence Diagram

La imagen que se muestra a continuación es el diagrama de secuencia del disparo de una transición, donde se ve cómo interactúan las distintas partes del sistema en dicha actividad.



Referencias

- Micolini, Orlando & Ventre, Luis & Cebollada, Marcelo & Eschoyez, Maximiliano. (2016). Ecuación de estado generalizada para redes de Petri no autónomas y con distintos tipos de arcos.
- https://docs.oracle.com/en/java/javase/19/
- https://www.debuggex.com/
- https://regex101.com/
- https://plantuml.com/es/