UNIVERSIDAD NACIONAL DE CÓRDOBA

Facultad de Ciencias Exactas, Físicas y Naturales



Programación Concurrente

Trabajo Práctico Final

**Grupo: Game of Threads**

Integrantes:

* Patiño, Jonathan Armando DNI: 35.636.684

Índice

[I) Sistema modelado por la Red de Petri 3](#_Toc89819391)

[Propiedades de la red original 3](#_Toc89819392)

[Análisis de invariantes 4](#_Toc89819393)

[Invariantes de plaza 6](#_Toc89819394)

[Desbloqueo de la Red de Petri 7](#_Toc89819395)

[Comparación de las soluciones 11](#_Toc89819396)

[Conclusión 15](#_Toc89819397)

[II) Implementación en Java 15](#_Toc89819398)

[Descripción de las clases 15](#_Toc89819399)

[Clase Hilo 15](#_Toc89819400)

[Clase Matriz 16](#_Toc89819401)

[Clase RDP 16](#_Toc89819402)

[Clase Cola 16](#_Toc89819403)

[Clase Política 16](#_Toc89819404)

[Clase Monitor 16](#_Toc89819405)

[Clase Main 16](#_Toc89819406)

[Clase Log 16](#_Toc89819407)

[Clase Mutex 16](#_Toc89819408)

[III) Dinámica de ejecución 17](#_Toc89819409)

[Mecanismos de control de la finalización del programa 17](#_Toc89819410)

[Mecanismos de control de la concurrencia de los hilos 17](#_Toc89819411)

[IV) Implementación de la política 18](#_Toc89819412)

[V) Diagrama de clase 20](#_Toc89819413)

[VI) Diagrama de secuencia 21](#_Toc89819414)

[VII) Determinación de la cantidad de hilos necesarios 21](#_Toc89819415)

[VIII) Análisis con 1000 Disparos 23](#_Toc89819416)

[IX) Registro de resultados 24](#_Toc89819417)

[X) Interpretación de los invariantes 25](#_Toc89819418)

[XI) Verificación de los invariantes de plaza 26](#_Toc89819419)

[XII) Expresiones regulares 26](#_Toc89819420)

[XIII) Tabla de eventos 27](#_Toc89819421)

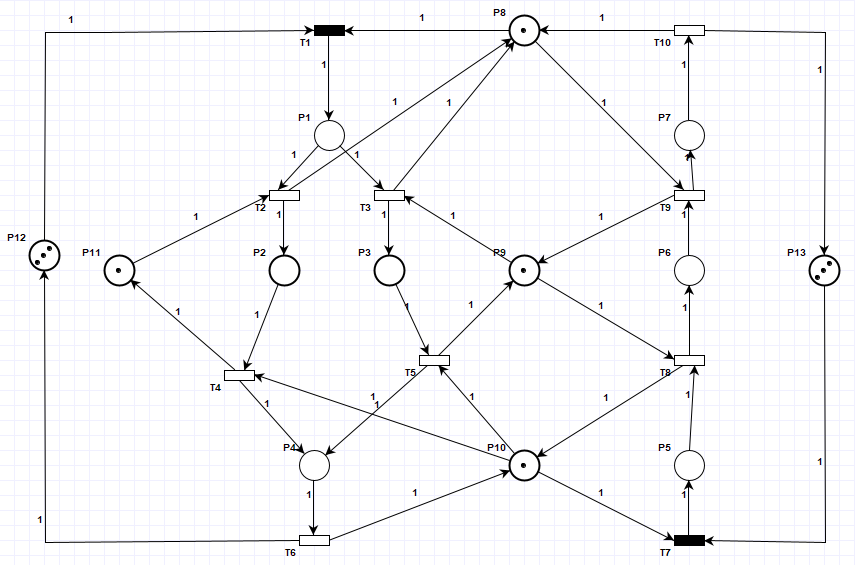
[XIV) Tabla de estados 28](#_Toc89819422)

[XV) Conclusión 28](#_Toc89819423)

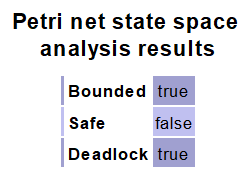
# I) Sistema modelado por la Red de Petri

## Propiedades de la red original

Se partió de la Red de Petri del enunciado y se utilizó la herramienta Pipe para analizar sus propiedades.



Se ejecutó la clasificación de la red y el “State Space Analysis” que ofrece la herramienta, obteniendo los siguientes resultados:

****

* La red es acotada (bounded = true): esto significa que la cantidad de tokens en la red se mantiene constante cuando la red evoluciona. Tiene sentido, considerando que la red tiene 2 circuitos cerrados que comienzan y terminan en la misma plaza. Uno de ellos es el indicado por las transiciones T1, T2T4 ó T3T5, T6, que utiliza como buffer a la plaza P12, inicialmente con 3 tokens. El otro es el camino indicado por las transiciones T7, T8, T9 y T10, con P13 como buffer.

Que sean circuitos cerrados implica que todo lo que sale de P12 vuelve en igual cantidad. Lo mismo sucede con P13. El token que sale del buffer va avanzando en algún proceso y debe liberar el recurso que está usando para que pueda ingresar otro token. En ningún momento hay una transición que consuma o genere más de un token, y todo recurso compartido que se usa, se devuelve dentro del mismo circuito cerrado. Las plazas en sí no están limitadas a una cantidad de tokens, pero la forma en que está diseñado el modelo ocasiona que nunca haya más de 3 tokens entre las plazas de tareas de cada circuito cerrado (P1, P2, P3, P4, P5, P6 y P7) y los buffers (P12 y P13), y que nunca haya más de un token en las plazas correspondientes a los recursos compartidos (P8, P9, P10 y P11). La red sería no acotada si, por ejemplo, se agregara una plaza de control que cuente la cantidad de veces que se disparó T7.

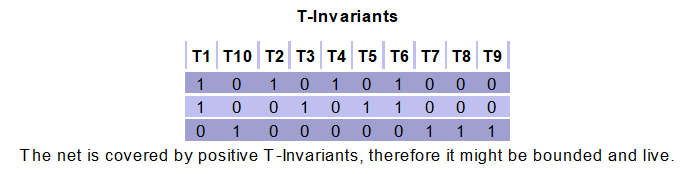
* La red no es segura (safe = false): una plaza de la red es k-limitada para un marcado inicial M0 si, para cualquier otra marca, el número de tokens en p es menor o igual a k. Una red es k-limitada para un marcado inicial M0 si cada plaza es k-limitada. Una RdP es segura si todas sus plazas, y por lo tanto la red en sí, son 1-limitadas. Esto no se cumple, porque desde el inicio hay 3 tokens en cada uno de los buffers.
* La red tiene deadlock (deadlock = true), por lo tanto, la red no está viva: como se ve en el modelo, las transiciones T3, T4, T5, T7, T8 y T9 utilizan recursos que son compartidos con otras transiciones y, a priori, no hay una relación de orden respecto a qué transición toma el token. Tampoco hay una liberación del recurso sin antes tomar el siguiente recurso compartido. Se terminan generando interbloqueos porque puede que haya dos transiciones que necesiten intercambiar recursos y que ninguna suelte el suyo hasta no obtener el siguiente. Por ejemplo, si se dispara T3, va a estar utilizando el recurso de P9, y para disparar T4 va a necesitar el token de P10. Si luego de T2 se disparó T7, va a tomar el token de P10 y no lo va a liberar hasta no tener acceso al token de P9, que ya lo tiene tomado T3. Es necesario así imponer restricciones para que se inhiban ciertas transiciones si ocurrieron otras previamente, como se verá a continuación.
* La red no es persistente: una red es persistente si todas sus transiciones cumplen con que una vez habilitada la transición, ésta sólo puede deshabilitarse mediante su disparo. Esto no se cumple, precisamente porque hay transiciones que comparten recursos. Luego, una transición que estaba sensibilizada puede perder su estado si otra transición utiliza primero el recurso en común.
* La red no es reversible: una RdP es reversible para una marca inicial M0 si M0 es un Home State, es decir, si M0 es una marca alcanzable desde cualquier otra marca. Si bien, la marca M0 de esta red pareciera ser un home state, esto no se cumple porque la red tiene deadlock, luego hay marcas en las que no va a haber ninguna transición sensibilizada y, por lo tanto, no se podrá volver al estado inicial.
* Conflictos: esta red presenta conflictos del tipo estructural y del tipo efectivo, porque hay transiciones que tienen un lugar de entrada común. No se consideran conflictos generales porque el grado de habilitación de las transiciones es siempre 1.
  + Las transiciones T2 y T3: el conflicto es estructural, porque tienen a P1 en común, y también efectivo porque en ningún momento puede haber más de un token en P1 permitiendo que se disparen las dos transiciones a la vez (para que haya un token en P1 se debe disparar T1 que requiere un token de P8, el cual solo se repone al dispararse T2 o T3).
  + Las transiciones T4, T5 y T7: las tres transiciones comparten el token de P10. Similar al caso anterior, el conflicto es tanto estructural como efectivo, ya que en ningún momento P10 puede tener más de un token.
  + Las transiciones T3 y T8: comparten a P9, conflicto estructural y efectivo.
  + Las transiciones T1 y T9: comparten a P8, conflicto estructural y efectivo.
* Exclusión mutua: todas las transiciones que presenten conflicto con otras, se deberán realizar en exclusión mutua, lo que se ve representado con los recursos compartidos que tienen un solo token y son consumidos por una transición a la vez.

## Análisis de invariantes

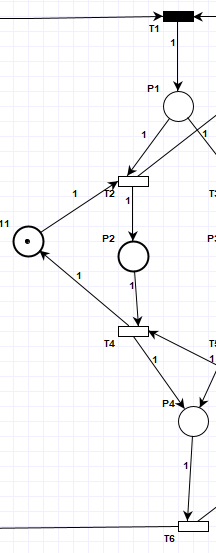
Implica elementos del sistema que no varían, independientemente del orden con el que se disparen las transiciones.

#### Invariantes de transición

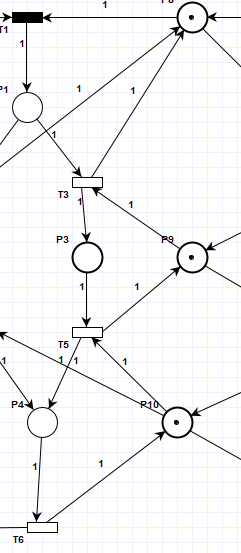
Representan los circuitos cerrados de la red, es decir, una secuencia de transiciones que luego de dispararse vuelven a la red a su estado original. Son componentes repetitivos. Esta red presenta 3 invariantes de transición:



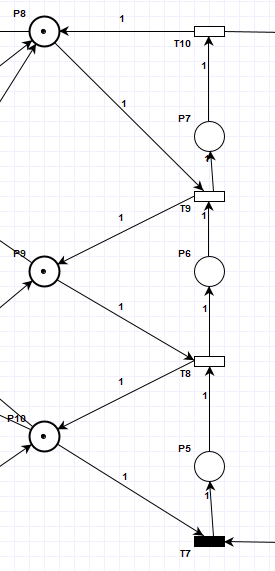
1. T1T2T4T6



1. T1T3T5T6



1. T7T8T910



## Invariantes de plaza

Representan relaciones constantes entre la cantidad de tokens en las plazas, que pueden expresarse como relaciones que definen el comportamiento de la red y las acciones que deben realizarse en exclusión mutua. Esta red tiene 6 invariantes de plaza:

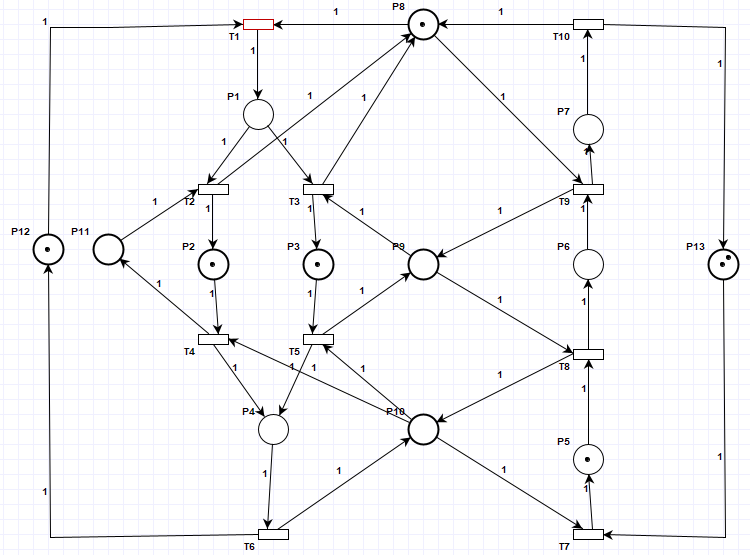
#### 

1. Entre las plazas P10, P4 y P5 sólo puede haber un token a la vez. Esto es, el token que inicialmente se encuentra en P10 representa un recurso que puede estar desocupado, si se encuentra en P10, o está siendo utilizado por P4 o P5, pero no por los dos a la vez. Por lo tanto, es necesaria la exclusión mutua entre P4 y P5 en cuanto al uso del recurso de P10.
2. El recurso de P11 puede estar desocupado o en uso en P2, pero no hay necesidad de exclusión mutua porque P2 es la única tarea que hace uso de este token.
3. La rama de la izquierda es un circuito cerrado que tiene siempre 3 tokens repartidos entre sus plazas (P1, P2, P3, P4 y P12). Esto coincide con el primer y segundo invariante de transición, que indican el camino para que los tokens vuelvan a su estado inicial.
4. La rama de la derecha es un circuito cerrado que tiene siempre 3 tokens repartidos entre sus plazas (P5, P6, P7 y P13). Esto coincide con el tercer invariante de transición.
5. Entre las plazas P8, P1 y P7 solo puede haber un token a la vez. El token que inicialmente está en P8 representa un recurso compartido que puede estar desocupado, si está en la plaza P8, o en uso por P1 o P7, pero no los dos a la vez. Es necesaria la exclusión mutua entre P1 y P7.
6. Entre las plazas P9, P3 y P6 sólo puede haber un token a la vez. De la misma forma que se explicó en los casos anteriores, esto indica que es necesaria la exclusión mutua entre P3 y P6.

## Desbloqueo de la Red de Petri

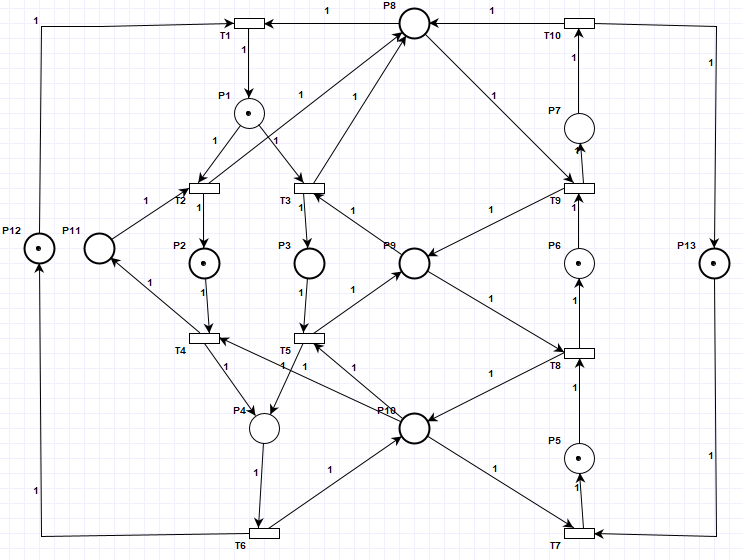
Se analizaron los distintos casos en los que la red se bloqueaba para entender cómo evitar que dos o más transiciones se bloqueen mutuamente. En particular, se determinó que para que la red no se bloquee se debe cumplir que:

1. Caso 1 de interbloqueo: hay token en P2 y P3 y se dispara T7. T3 toma el recurso de P9 y no lo libera hasta que se dispara T5, para ello requiere del token de P10 que está en P5. Por su parte, T8 no puede disparar y liberar el recurso de P10 porque necesita del recurso de P9 que está en P3.



* + Solución: si se disparó T2 o T3, ya no se debería poder disparar T7 hasta que no se dispare T6, porque si no T7 consumiría el token de P10 y los tokens de P2 y P3 quedarían atrapados hasta que se dispare T8. Por la misma razón, si se disparó primero T7 ya no se debería poder disparar T3 hasta que no se dispare T9. Notar que no es necesario restringir la transición T2 si se disparó T7 porque T2 no utiliza recursos compartidos, como sí lo hace T3.

1. Caso 2 de interbloqueo: hay un token en P2 y se dispara T7 T8 T1 T7, o bien, T1 T7 T8. Para que el token de P6 pueda continuar necesita del token de P8 que, como se disparó T1, está en P1. T2 no se puede disparar porque ya hay un token en P2 que no puede avanzar porque T4 necesita el recurso de P10 que tomó T7. T3 tampoco puede disparar porque necesita el token de P9 que tomó T8.



* + Solución: no dejar disparar T1 si previamente se disparó T8. Así el token de P6 no se queda atrapado.

Para implementar las soluciones, se consideraron varias alternativas que se explican a continuación. Se puede comprobar que con cualquiera de las soluciones no se alteran los invariantes de la red original.

1. **Utilizar arcos inhibidores:** se agregaron 4 brazos inhibidores:

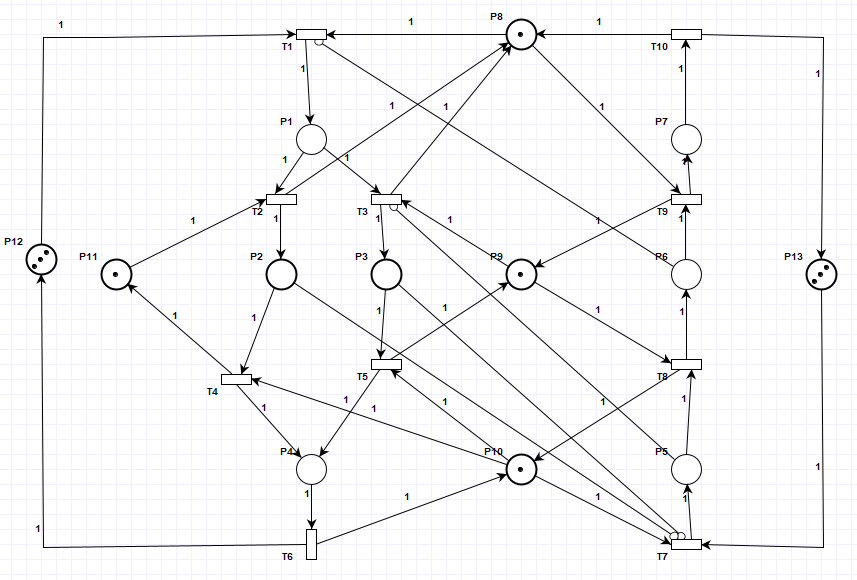


Figura 1: Red de Petri N°1.

* 1. Entre P3 y T7 para que T7 no se dispare si previamente se disparó T3 y así el token de P3 no queda encerrado.
  2. Entre P2 y T7 para que T7 no se dispare si previamente se disparó T2 y así el token de P2 no queda encerrado.
  3. Entre P5 y T3 para que no se dispare T3 si previamente se disparó T7 y el token de P5 no quede encerrado.
  4. Entre P6 y T1 para que no se dispare T1 si previamente se disparó T8 y así el token de P6 no quede encerrado.

1. **Utilizar arcos lectores:** se agregan 3 arcos lectores.

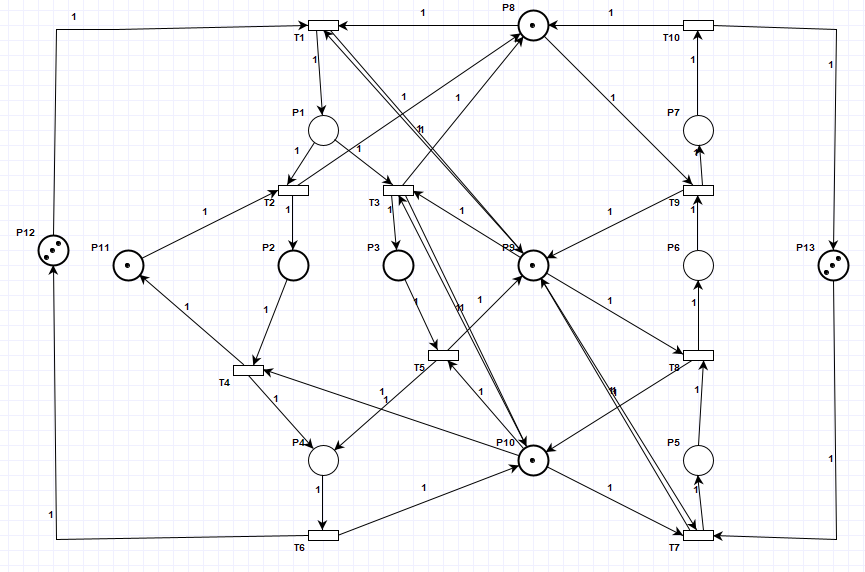


Figura 2: Red de Petri N°2.

* 1. Entre T3 y P10, de forma que T3 sólo se dispare si T7 no consumió el recurso compartido de P10.
  2. Entre T7 y P9, de forma que T7 solo dispare si T3 no consumió primero el recurso compartido de P9.
  3. Entre T1 y P9, para que T1 no pueda disparar si ya disparó T8 y así el token de P6 no queda encerrado.

1. **Utilizar plazas:** se agregan 2 plazas.

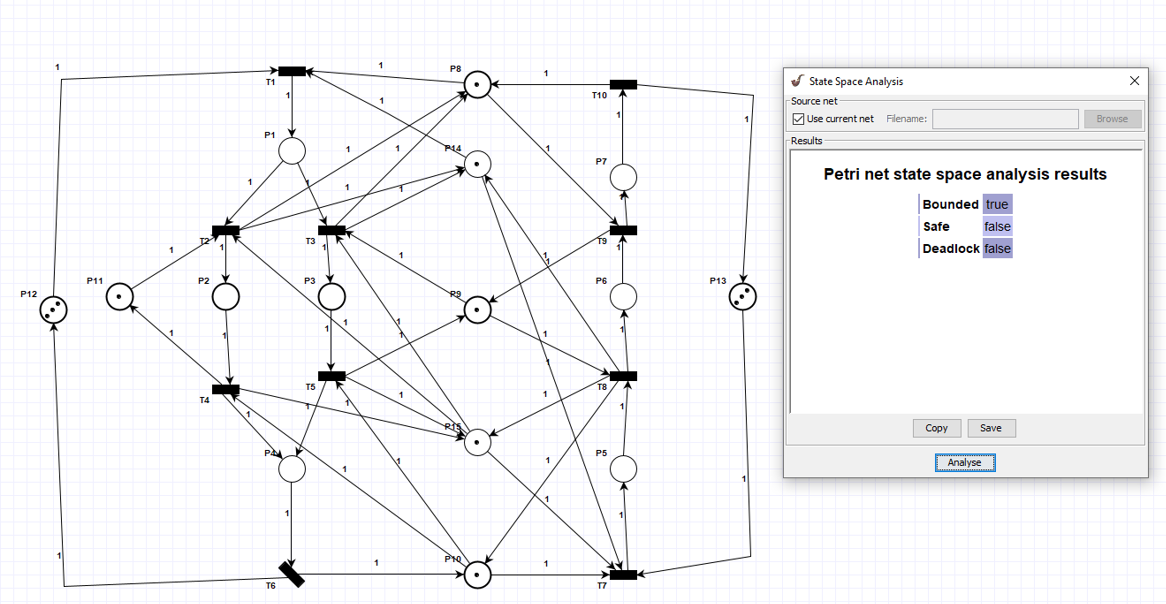


Figura 3: Red de Petri N°3.

* Se agregaron P14 y P15 de tal forma que si se dispara T1 ya no se pueda disparar T7 o en caso contrario si se dispara T7 ya no pueda dispararse T1.

1. **Utilizar plazas:** se agrega 1 plaza.

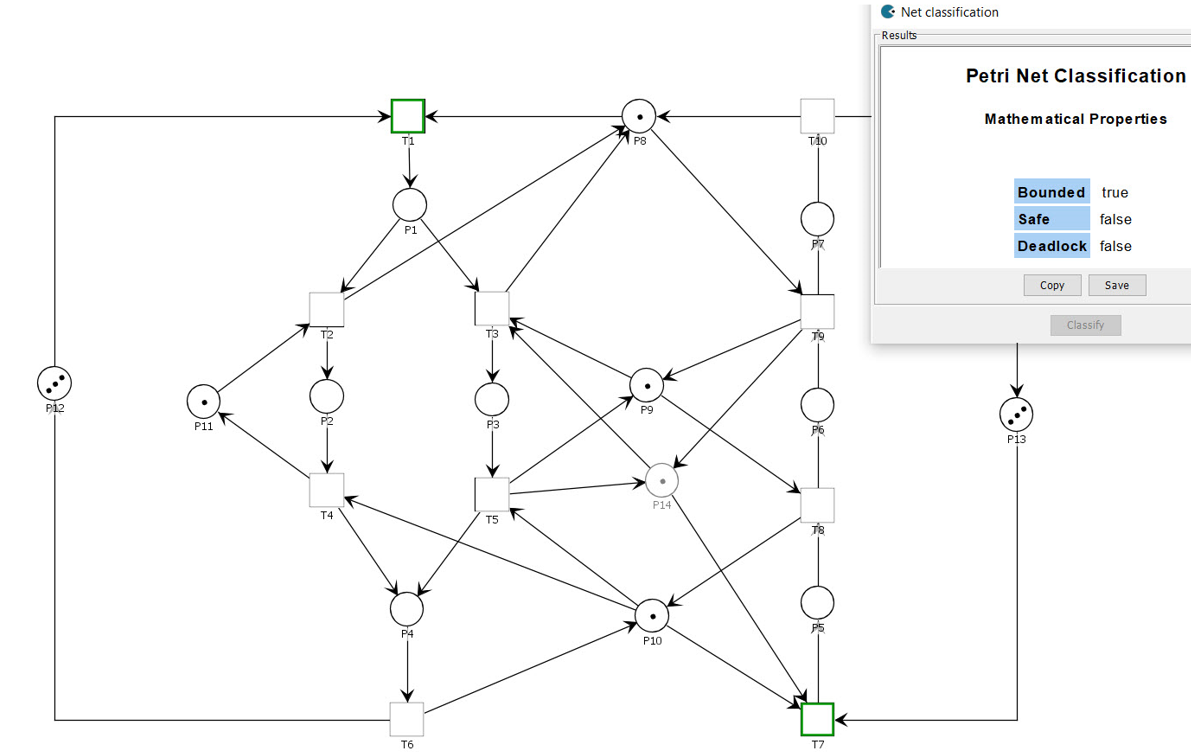


Figura 4: Red de Petri N°4.

* En este caso se implementa de manera tal que si se dispara T7 no pueda dispararse T3 o viceversa.

1. **Utilizar plazas:** se agrega 1 plaza, pero con diferente distribución de los arcos.

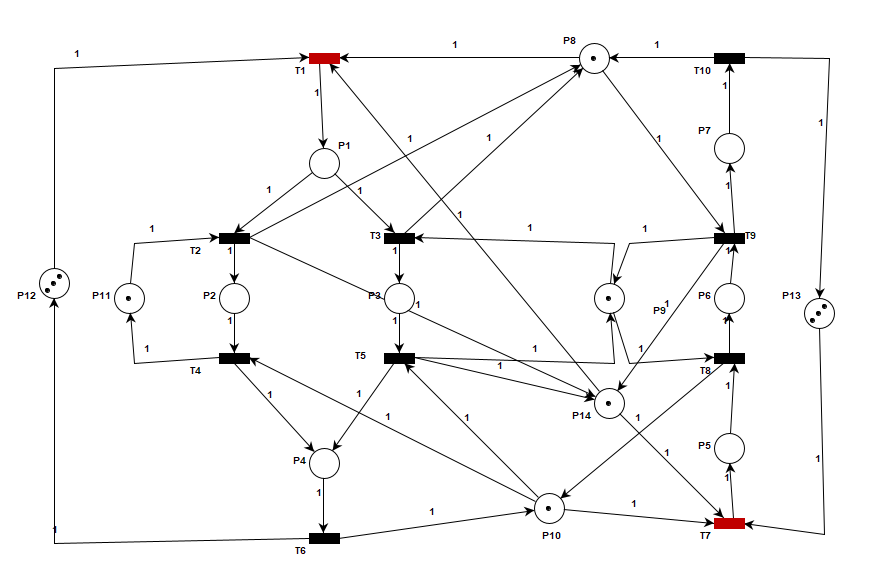


Figura 5: Red de Petri N°5.

* En este caso, la competencia por el avance del hilo a través de los invariantes, se realiza a través de la restricción que tiene entre T1 y T7 este tipo de red es un poco más restrictiva, es decir, tiene menos paralelismo que la anterior cuando se trata sin tiempo, pero es una mejor opción cuando se le agrega el tiempo, ya que la decisión de ir por un invariante o por otro lo toman transiciones no temporales. En términos de ejecución, en la práctica son muy similares.

## Comparación de las soluciones

En principio, ambas soluciones son viables porque eliminan el deadlock y no alteran los invariantes de la red original. Para poder elegir la mejor alternativa, es necesario cuantificar en alguna medida el nivel de paralelismo que permiten estas opciones.

Procedimiento utilizado:

1. Se ejecutó en PIPE el GSPN Analysis de cada red. Éste devuelve, entre otras cosas, todos los posibles marcados de la red y la distribución de probabilidad de cada uno de los estados marcados posibles.
2. Cada uno de estos resultados se copiaron en una hoja de cálculo. Se armó una tabla con todos los marcados posibles en la que se eliminaron las columnas no relevantes. Solo se dejaron las columnas correspondientes a P1, P2, P3, P4, P5, P6 y P7 que son las plazas que indican que se está realizando una tarea. Se agregó una columna con la distribución asociada a cada estado.
3. Se suma para cada estado la cantidad de procesos que hay activos al mismo tiempo. Se obtiene un mínimo de 0 para el marcado inicial M0, y un máximo de 3 en varios estados. Se representan con distintos colores para indicar mayor o menor concurrencia. Claramente, cuantos más procesos activos tenga el estado, mayor concurrencia habrá.
4. Se multiplica el número de procesos activos en ese estado por la probabilidad de que la red esté en ese estado (columna de distribución).
5. Por último, se suman los valores obtenidos en el punto anterior y se obtiene un número promedio de la cantidad de procesos activos simultáneamente. Este número da una medida cuantificable del paralelismo que permite esta red. Cuanto más por encima esté de 1, mayor paralelismo habrá. Esto es así porque daría 1 en el caso de que en todos los estados haya 1 tarea realizándose (en realidad daría un poco menos que 1 porque en M0 no hay tareas activas y es el estado con mayor distribución de probabilidad). Para que dé mayor que 1 es necesario que en algún estado haya más de una tarea ejecutándose en paralelo.
   * Otra forma de calcular este número es multiplicar cada celda por la distribución correspondiente a su estado y luego sumar los valores de las columnas. Se obtiene un número por plaza que indica una especie de porcentaje en que la plaza, y por lo tanto la tarea que representa, estaría activa en la evolución de la red. Esto permite hacer un análisis comparativo entre las plazas para determinar si todos los invariantes se están ejecutando aproximadamente la misma cantidad de veces. La suma de los valores de cada plaza da, nuevamente, el valor obtenido en el punto anterior.
   * Otro análisis que se hizo: se simularon 1000 transiciones de la red y luego se sacó el porcentaje en el que se disparó cada transición. Se agruparon por invariantes y se vio la cantidad de veces que la red fue por cada rama.
   * Otra forma de obtener los mismos datos es mediante Petrinator, utilizando la opción reachibility/coverability, que permite obtener todos los marcados posibles.

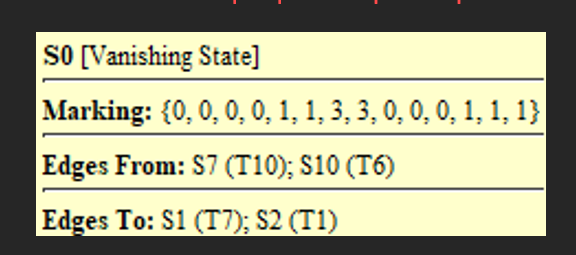


Figura 6: figura obtenida de Petrinator donde se observa uno de los marcados.

Extrayendo la información provista por Petrinator y luego llevada a una hoja de cálculo (i.e, Excel) en este caso se suman los tokens que se encuentran en las plazas que representan tareas realizándose, y luego se procede a sacar un promedio de todos los estados.

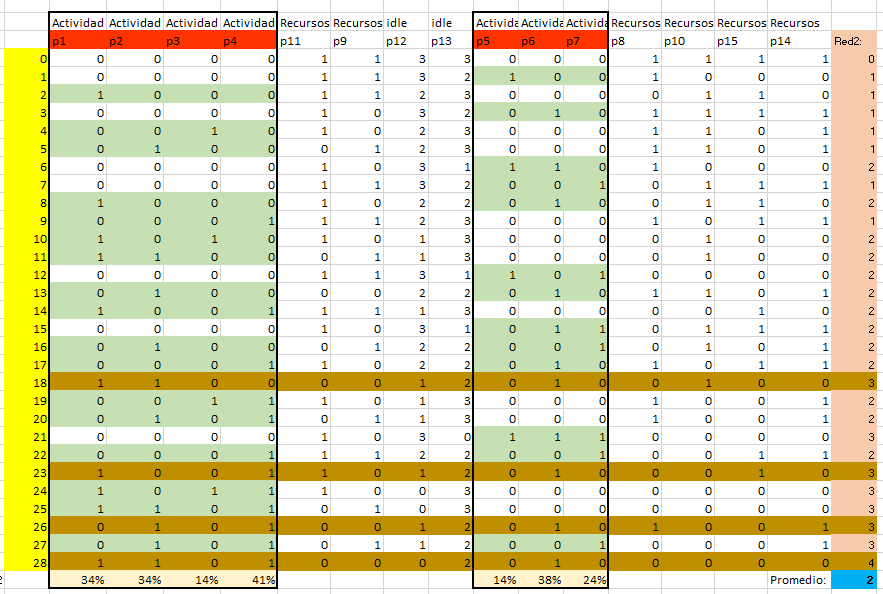
Resultados obtenidos con la solución que implementa arcos inhibidores

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **∑** | **Distribución** |  |
| **M0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3,26% | 0,00 |
| **M1** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 3,72% | 0,04 |
| **M2** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2,22% | 0,02 |
| **M3** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 3,65% | 0,04 |
| **M4** | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 2,97% | 0,06 |
| **M5** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2,96% | 0,03 |
| **M6** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 3,70% | 0,04 |
| **M7** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2,51% | 0,03 |
| **M8** | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 2 | 5,87% | 0,12 |
| **M9** | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 1,98% | 0,04 |
| **M10** | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 2 | 1,98% | 0,04 |
| **M11** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 4,01% | 0,04 |
| **M12** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 2,90% | 0,06 |
| **M13** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 3,30% | 0,07 |
| **M14** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 4,19% | 0,08 |
| **M15** | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 3 | 1,98% | 0,06 |
| **M16** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 2 | 2,48% | 0,05 |
| **M17** | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 3 | 1,98% | 0,06 |
| **M18** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 3,40% | 0,07 |
| **M19** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 2 | 3,98% | 0,08 |
| **M20** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2,22% | 0,04 |
| **M21** | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 3 | 1,98% | 0,06 |
| **M22** | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 1,36% | 0,03 |
| **M23** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 2 | 1,36% | 0,03 |
| **M24** | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 3 | 1,98% | 0,06 |
| **M25** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 2 | 3,69% | 0,07 |
| **M26** | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 2 | 3,82% | 0,08 |
| **M27** | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 3 | 1,99% | 0,06 |
| **M28** | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 3 | 2,22% | 0,07 |
| **M29** | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 3 | 0,99% | 0,03 |
| **M30** | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 1,36% | 0,03 |
| **M31** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | 0,99% | 0,03 |
| **M32** | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 3 | 2,84% | 0,09 |
| **M33** | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 3 | 2,90% | 0,09 |
| **M34** | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 3 | 0,50% | 0,01 |
| **M35** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 3 | 0,50% | 0,01 |
| **M36** | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 3 | 5,75% | 0,17 |
| **M37** | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 3 | 0,25% | 0,01 |
| **M38** | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 3 | 0,25% | 0,01 |
|  |  |  |  |  |  |  |  |  |  |  |
|  | **0,31** | **0,39** | **0,24** | **0,31** | **0,28** | **0,28** | **0,16** | Promedio de procesos activos simultáneamente | | **1,98** |

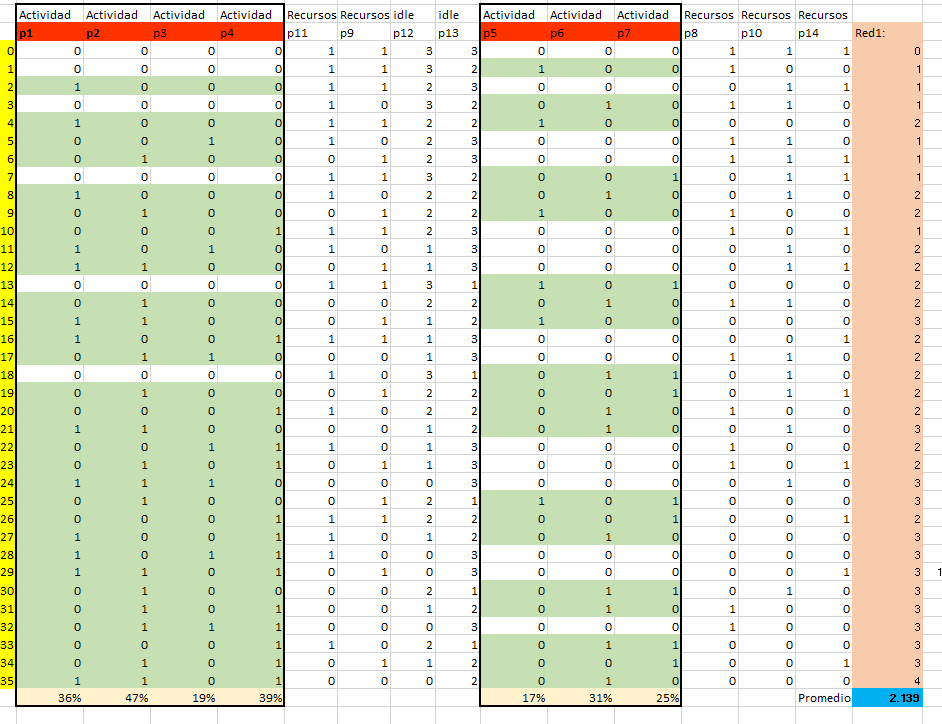
Resultados obtenidos con la solución que implementa arcos lectores

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **P1** | **P2** | **P3** | **P4** | **P5** | **P6** | **P7** | **∑** | **Distribución** |  |
| **M0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 5,94% | 0 |
| **M1** | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 4,34% | 0,04338 |
| **M2** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 3,78% | 0,03781 |
| **M3** | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 8,98% | 0,08976 |
| **M4** | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 4,06% | 0,08118 |
| **M5** | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 5,98% | 0,05981 |
| **M6** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 4,87% | 0,0487 |
| **M7** | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 5,47% | 0,05471 |
| **M8** | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 5,07% | 0,10148 |
| **M9** | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 2 | 2,03% | 0,0406 |
| **M10** | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 6,41% | 0,06408 |
| **M11** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 4,40% | 0,08798 |
| **M12** | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 2 | 2,74% | 0,05472 |
| **M13** | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 2 | 3,81% | 0,07612 |
| **M14** | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 3 | 2,03% | 0,0609 |
| **M15** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 5,40% | 0,10808 |
| **M16** | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 2 | 2,20% | 0,044 |
| **M17** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 | 2,74% | 0,05472 |
| **M18** | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 2 | 2,03% | 0,0406 |
| **M19** | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 2 | 1,90% | 0,03806 |
| **M20** | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 3 | 2,03% | 0,0609 |
| **M21** | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 2 | 3,93% | 0,07858 |
| **M22** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 2 | 2,20% | 0,044 |
| **M23** | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 2 | 1,97% | 0,03932 |
| **M24** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 3 | 1,02% | 0,03045 |
| **M25** | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 3 | 3,93% | 0,11787 |
| **M26** | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 3 | 0,51% | 0,01521 |
| **M27** | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 3 | 0,25% | 0,00762 |
|  |  |  |  |  |  |  |  |  |  |  |
|  | **0,32** | **0,32** | **0,10** | **0,28** | **0,15** | **0,26** | **0,15** | Promedio de procesos activos simultáneamente | | **1,58** |

Resultados obtenidos con la solución que implementa 2 plazas:

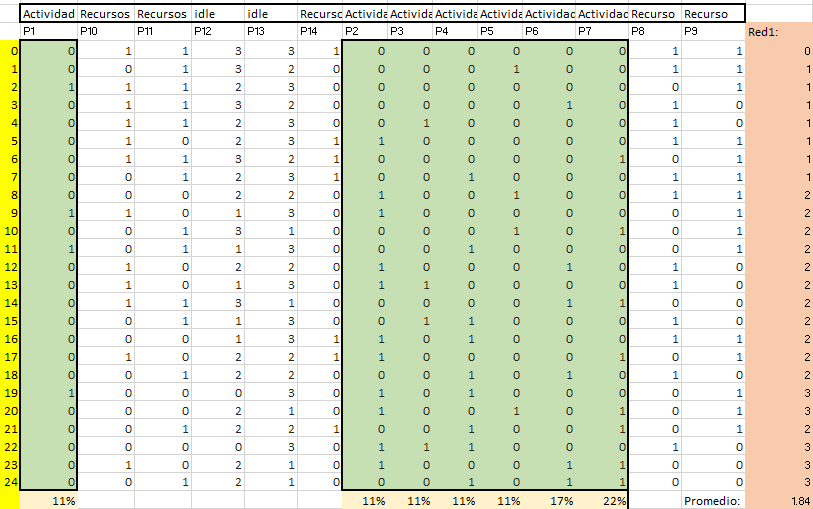


Resultados obtenidos con la solución que implementa 1 plaza:



Resultados obtenidos con la solución que implementa 1 plaza:

Se diferencia de la anterior en la distribución de los arcos.



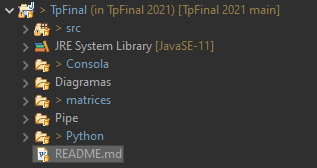
## Conclusión

En base a los resultados obtenidos, se optó por usar la solución con una sola plaza con un promedio de paralelismo de 1.84. Esto se debe a que, si bien no es la que permite un mayor paralelismo (teniendo en cuenta el promedio de los procesos activos, la cual es de 2.139), ofrece la posibilidad de utilizar intervalos de tiempo mayores. Es decir, en el caso de la de mayor paralelismo se debe hacer prácticamente instantánea la transición T3, ya que esta transición compite con T7 -la cual no es temporal-. En cambio, la red que se consideró para la presentación fue una en donde compiten transiciones no temporales como lo son T1 y T7.

Cabe aclarar que el hecho de elegir esta red, no quita que se hayan tenido en cuenta las demás.

# II) Implementación en Java

Para la implantación en Eclipse se necesita una versión de la máquina virtual de Java, mayor o igual que 11.



## Descripción de las clases

### Clase Hilo

Esta clase recibe como parámetro una secuencia a disparar. En su método *run()*, llama al método *dispararTransicion()* de la clase Monitor y dispara secuencialmente las transiciones que le fueron asignadas dentro de un bucle *while*. Continúa haciéndolo hasta que desde el hilo del *main* se llama al método *setFin()* que pone en *false* la variable *continuar*, permitiendo que el hilo finalice la ejecución de su método *run()*.

### Clase Matriz

Es una clase auxiliar que contiene todos los métodos necesarios para definir y operar con las matrices de la Red de Petri, como generar la matriz a partir de los datos de un archivo, escribir o leer un dato determinado, multiplicar o hacer un AND con otra matriz, etc.

### Clase RDP

Contiene todos los métodos necesarios para implementar una Red de Petri a partir de archivos de texto con:

* Los invariantes de transición
* La matriz de incidencia
* La matriz de entrada
* La matriz de inhibición (opcionalmente)
* El vector de marcado inicial
* Los vectores de intervalos de tiempo

Esta clase, además de los métodos para cargar los datos de las matrices en estructuras de datos adecuadas, también contiene métodos para poder describir el estado de la red en cualquier momento.

### Clase Cola

Modela las colas donde van a esperar los hilos para acceder al monitor y contiene los métodos que agregan y sacan transiciones de la cola. Genera un arreglo de semáforos, con tantos semáforos como transiciones tenga la Red de Petri. Todos son inicializados con 0 permisos disponibles. Esto es porque sólo se quiere usar al semáforo para registrar el orden en el que los hilos van llegando a la cola, y así poder darles acceso al monitor cuando se habilite su transición.

El control de los *acquire* y *release* de los semáforos los tiene el monitor cuando llama a los métodos *ponerEnCola()* y *sacarDeCola()*, respectivamente.

### Clase Política

Posee como método más relevante el método *cual(Matriz m)* que permite la elección entre dos transiciones, las cuales deben estar sensibilizadas y también deben estar en la cola de espera en el monitor. La elección se lleva a cabo de acuerdo a cuál transición se disparó menos en el transcurro del tiempo. La clase política también registra cada disparo exitoso de una transición.

### Clase Monitor

Se encuentra lo necesario para controlar la sección critica. A esta clase sólo accede el hilo que tomó el token del semáforo binario, el cual se encuentra en la clase *Mutex()* y puede hacer uso de los recursos que necesite para avanzar o esperar hasta que tenga los recursos disponibles.

### Clase Main

Es la clase principal para un programa realizado en Eclipse. Todo programa debe tener una clase *main*. Esta clase está controlada por el proceso principal, y es la encargada de crear los hilos y, luego, finalizarlos.

### Clase Log

Se registran el cambio de marcado y el vector de sensibilizado extendido, el cual, luego, es evaluado para saber si se tomaron las decisiones correctas.

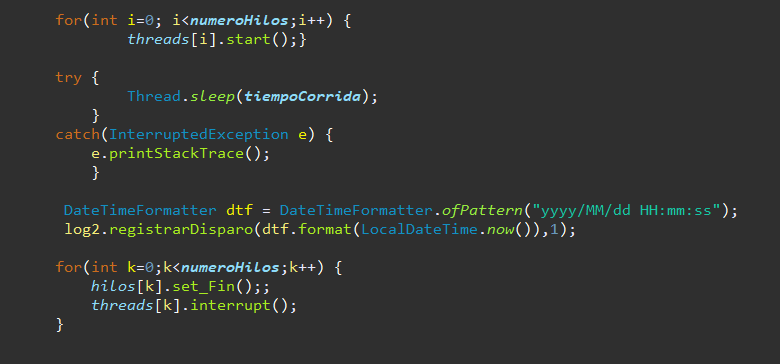
### Clase Mutex

Permite poder controlar la exclusión mutua, es decir, debido a esta clase es que los hilos pueden acceder de manera ordenada, y sólo de a uno, al monitor para realizar las tareas necesarias. Posee métodos *\_acquire()* y *\_release(),* lo cual, mediante el uso de semáforos binarios, permite tener acceso de a uno al monitor. También es usada por la clase RDP, ya que se puede liberar el monitor desde la RDP o desde el propio monitor.

# III) Dinámica de ejecución

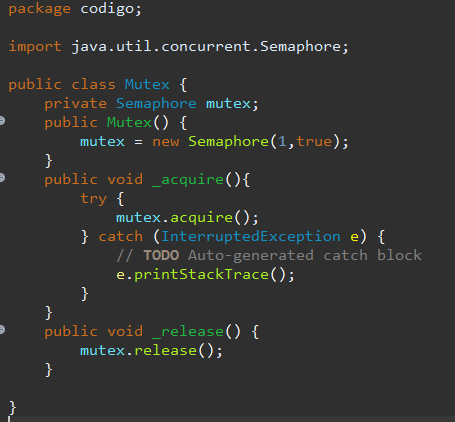
## Mecanismos de control de la finalización del programa

Los hilos comienzan en el proceso principal mediante el método *start()*, luego, el proceso principal, el cual mantiene conexión con los hilos, se duerme en el tiempo de corrida del programa establecido, luego despierta y llama al método *set\_Fin()* de cada hilo, y también al método *interrupt()* el cual interrumpe al hilo.

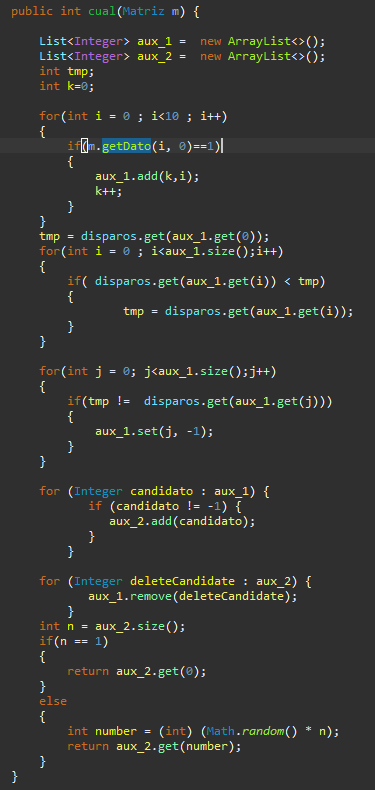


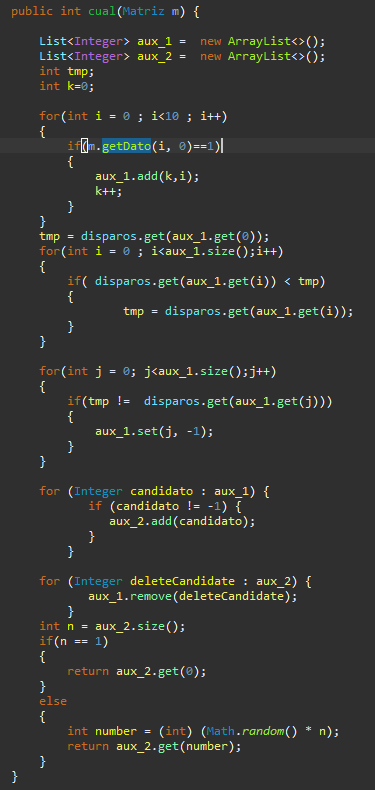
## Mecanismos de control de la concurrencia de los hilos

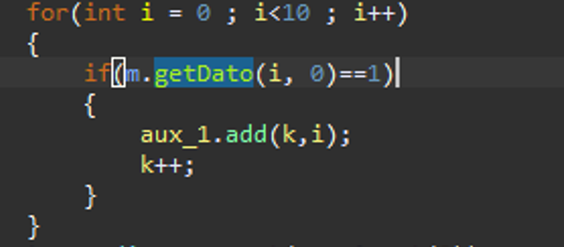
Para el control de la concurrencia se utiliza un semáforo binario, es decir, de dos estados (libre u ocupado). Luego, con los métodos *acquire* y *release* se puede utilizar la sección critica de forma ordenada y justa (justa se refiere a que el primero en llegar sea el primero en tomar la sección de código controlada por el semáforo).



# IV) Implementación de la política



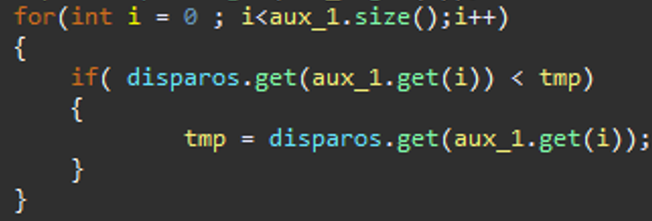




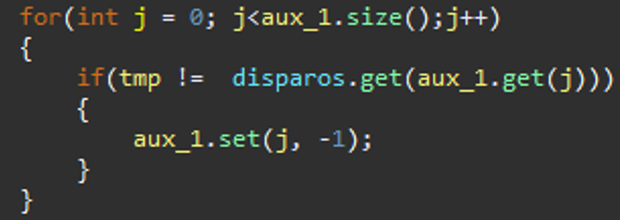
**aux\_1** contiene las transiciones que están esperando en el monitor (es decir, que están en *m*), o sea, que en *m* tiene un 1 en la posición de esa transición.



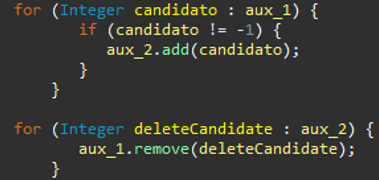
Contiene la cantidad de disparos de la primera transición que está en **aux\_1**.



Se obtiene el menor número de disparos que están en **aux\_1**



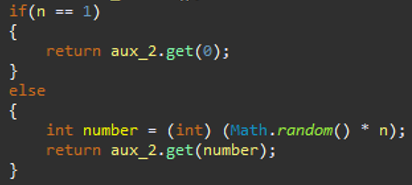
En el **for** anterior se coloca -1 a todas las transiciones que no tienen el mismo valor de **tmp**, es decir, las que no tienen la menor cantidad de disparos.



Se agrega en aux\_2 a los que no tienen -1, es decir, queden las transiciones que tienen menor cantidad de disparos.



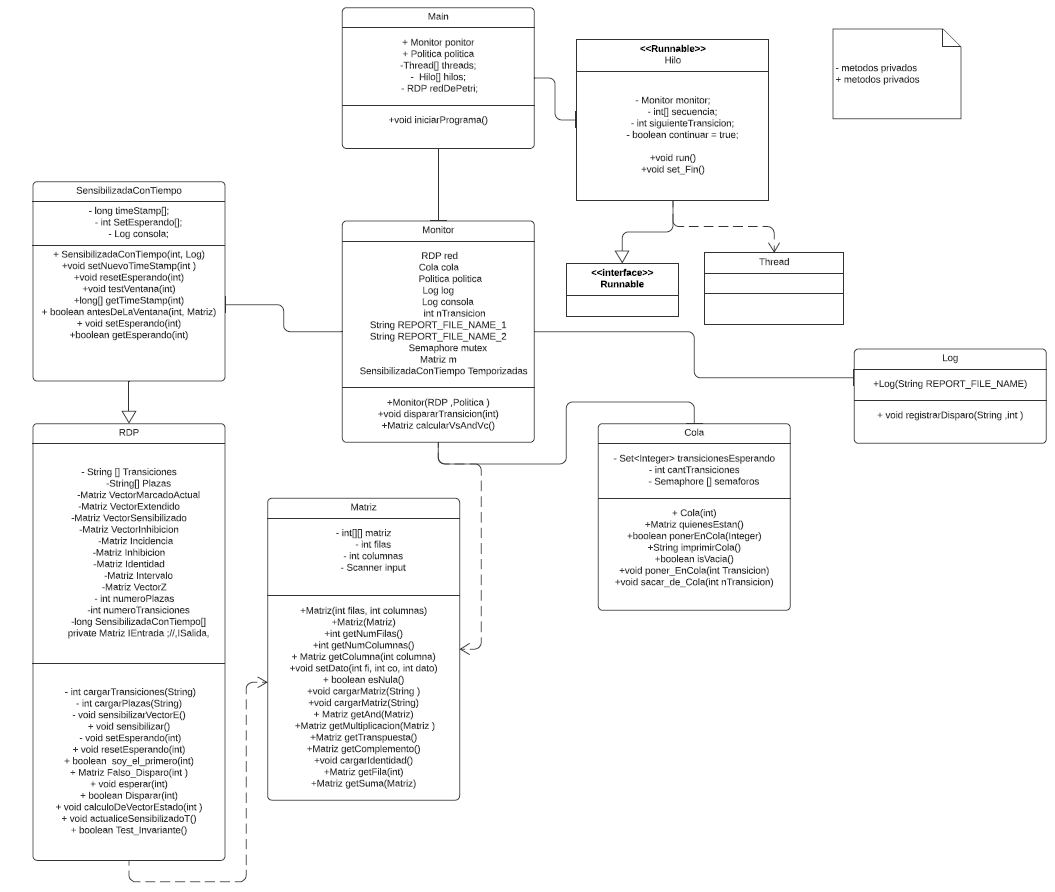
**n** contiene la cantidad de transiciones que fueron las que menos se dispararon. Luego se escoge un valor aleatoriamente (*random*) entre la cantidad n.



Si n == 1 solo quedó una transición, entonces, se retorna.

En caso contrario, de la cantidad de transiciones que quedaron se obtiene una, de manera aleatoria.

# V) Diagrama de clase



# VI) Diagrama de secuencia

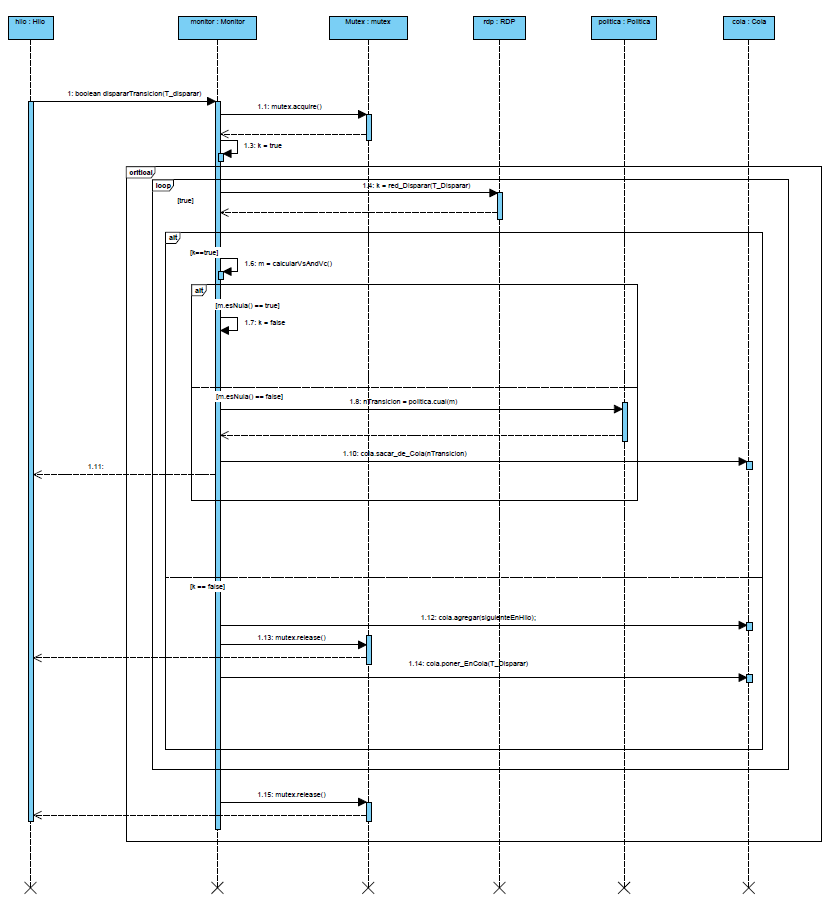


Figura 7: Monitor

# VII) Determinación de la cantidad de hilos necesarios

**Análisis para la Red de Petri N°5**

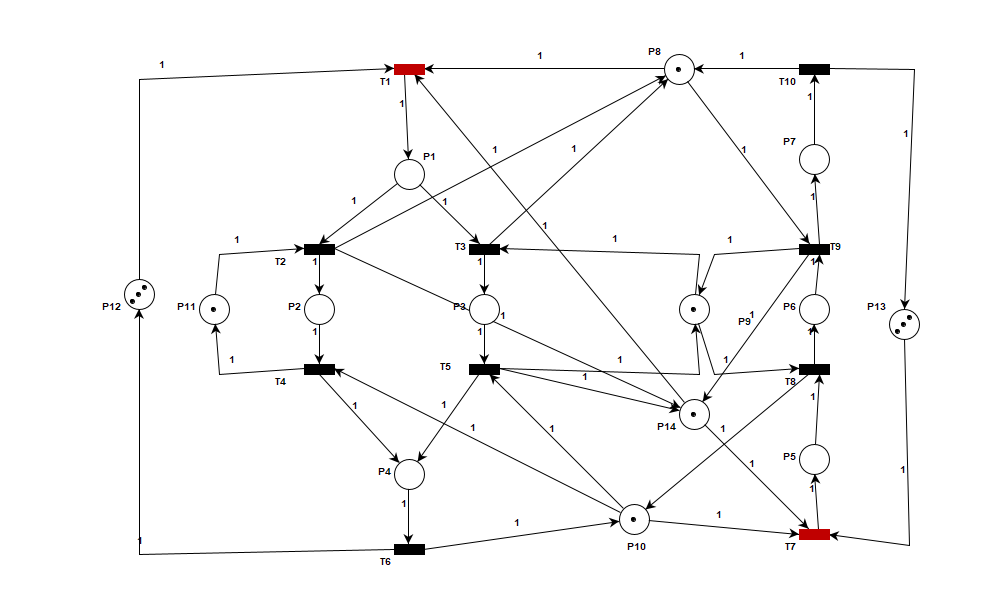
Los invariantes de la red representan tareas que están relacionadas, por lo tanto, mientras no haya conflictos en el camino del invariante, se utiliza un hilo para que las ejecute.

Este es el caso de las transiciones del invariante III (T7T8T9T10) que pueden ser ejecutadas en bucle por un hilo sin problemas.

En el caso de los otros dos invariantes, es diferente, porque hay conflicto en el medio. Si se dejara todo el invariante a cargo de un solo hilo, al llegar al conflicto no sabría qué camino tomar y habría posibilidad de que un camino no se ejecute nunca. La elección del camino, por lo tanto, se deja a cargo del monitor, el cual a su vez se basará en la política implementada. Habrá un hilo a cargo de la ejecución de la transición previa al conflicto, un hilo por cada rama en la que se divida, y luego el hilo a cargo de la transición previa se hará cargo, luego del *join*. Ambos invariantes comparten las transiciones T1 y T6, por lo que son necesarios 3 hilos distintos: uno que dispare T1, otro que dispare T2T4, otro T3T5 y un hilo que dispara T1 y otro que dispare T6. Habrá un solo hilo para cada secuencia. Esto es así porque, como T4 y T5 comparten el recurso de P10, en ningún caso puede haber más de un token en P4.

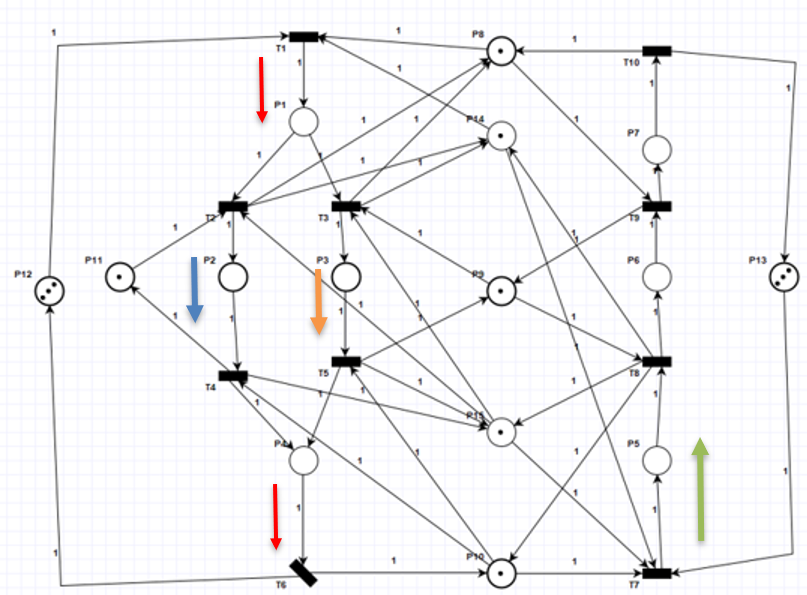
En síntesis, el programa utiliza 5 hilos, que se representan con distintos colores en el gráfico de abajo:

* 1 hilo dispara las transiciones T7T8T9T10
* 1 hilo dispara T1
* 1 hilo dispara T6
* 1 hilo dispara T2T4
* 1 hilo dispara T3T5



**Análisis para la Red de Petri N°3**

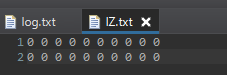
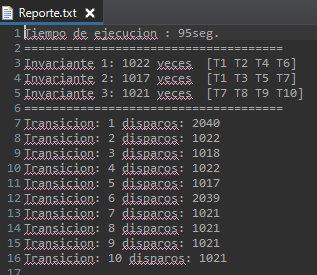
En este caso, se utilizan 4 hilos. Compartiendo hilos entre T1 y T6, es decir, se dispara T1 y luego el hilo hace el disparo de T6.

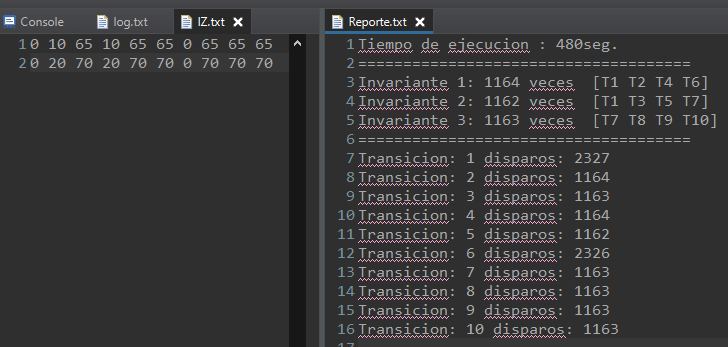


# VIII) Análisis con 1000 Disparos

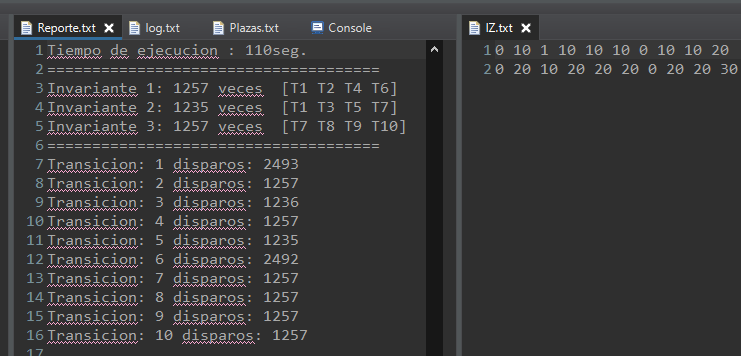
El programa devuelve dos archivos luego de la ejecución: uno llamado “Reporte”, el cual contiene la cantidad de veces que se dispararon las transiciones y la cantidad de veces que se cumplieron los invariantes; y otro, llamado “log”, el cual contiene la evolución de los eventos de la red y el vector sensibilizado extendido.

* 1000 disparos aprox. para la Red de Petri N°3:



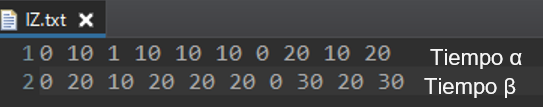


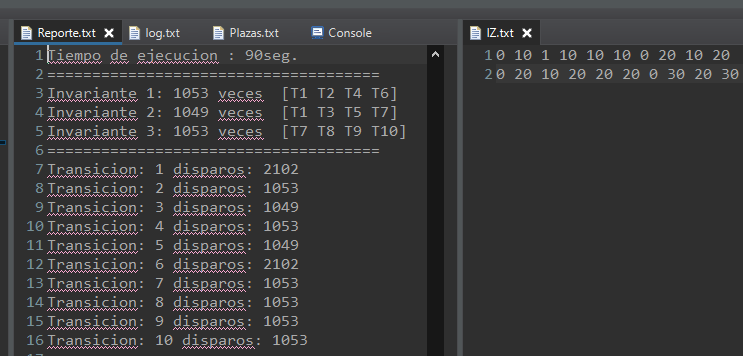
* 1000 disparos aprox. para la Red de Petri N°4:



De acuerdo con el tiempo de ejecución de cada invariante, se recurrió a observar el tiempo que demora cada invariante, sumar los tres tiempos y dividirlos por la cantidad de tiempo que tomó la ejecución. De este análisis se obtuvo un porcentaje aproximado de la cantidad de veces que se ejecutó cada invariante. A continuación, se observó el archivo Reporte.txt, y se constató que los resultados se asemejan. Puede haber algún retraso debido a la realización de otras instrucciones, las cuales consumen una fracción de tiempo.

La equidad de la política que se puede observar fue hallada a través de los tiempos que se fijaron en el disparo de cada transición que complementa al invariante. Los tiempos en milisegundos que lograron el equilibrio de forma más equitativa fueron los siguientes:





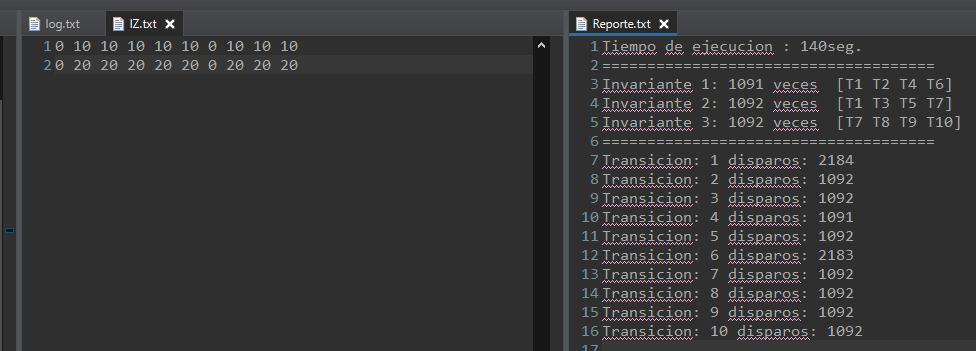
Tiempo mínimo consumido por transiciones en cada hilo:

* T1 + T2 + T4 = 20
* T1 + T3 + T5 = 11
* T7 + T8 + T9 + T10 = 50
* T6 = 10

Sumando, resulta = 91ms. (considerando el tiempo de ventana mínimo)

Resultado promedio por invariante = 900000/91 = 989

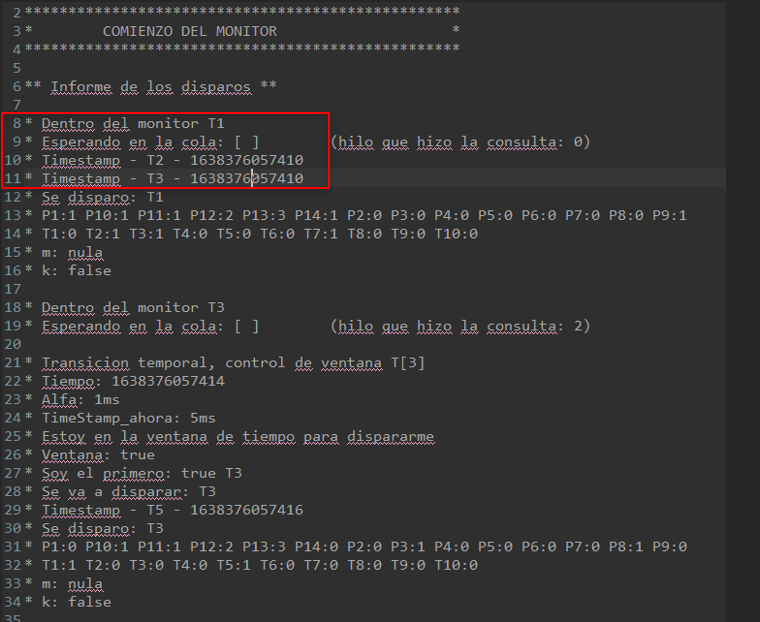
* 1000 disparos aprox. para la Red de Petri N°5:



Considerando las 3 redes que se analizaron, se observa que distintos intervalos de tiempo llevan al equilibrio de los invariantes. Los datos de las ejecuciones fueron los que llevaron a un intervalo de tiempo en el cual se equilibraban los invariantes en 1000 ejecuciones, no obstante, no necesariamente son los únicos tiempos que llevan a ese equilibrio.

# IX) Registro de resultados

En la siguiente imagen se muestran el avance del hilo y cómo cada hilo va disparando la transición y consumiendo los recursos o tokens que hay en las plazas. También se pueden observar los datos de las transiciones con tiempo. Por ejemplo, T1 setea en un vector el tiempo en el cual habilitó a T2 y T3, se puede observar en la figura.

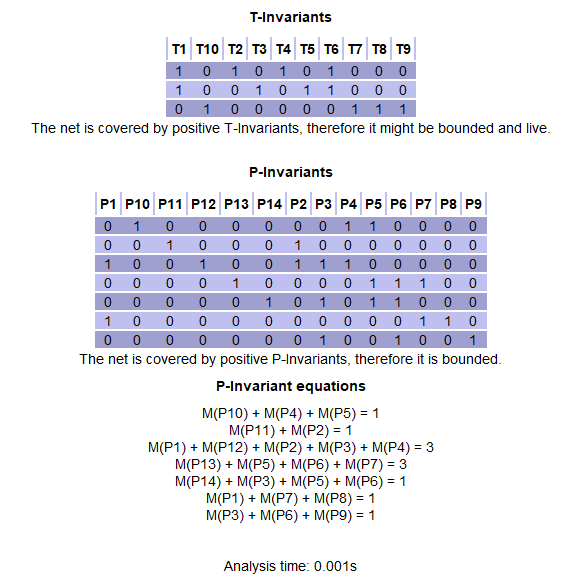


# X) Interpretación de los invariantes

A partir de una marca inicial, el marcado de una RdP puede evolucionar mediante el disparo de transiciones (si no hay deadlock, dicho número es ilimitado). Sin embargo, no se puede alcanzar cualquier marca, y todas las marcas alcanzables tienen algunas propiedades en común. Se dice que una propiedad que no varía cuando se activan las transiciones, es invariante.

Si la matriz de incidencia por el vector de disparo da cero, entonces el vector de disparo se llama invariante de transición el vector de disparo hace que la red vuelva al estado inicial.

Los invariantes de plaza indican que el número de tokens en todas las marcas alcanzables satisface algún invariante lineal.

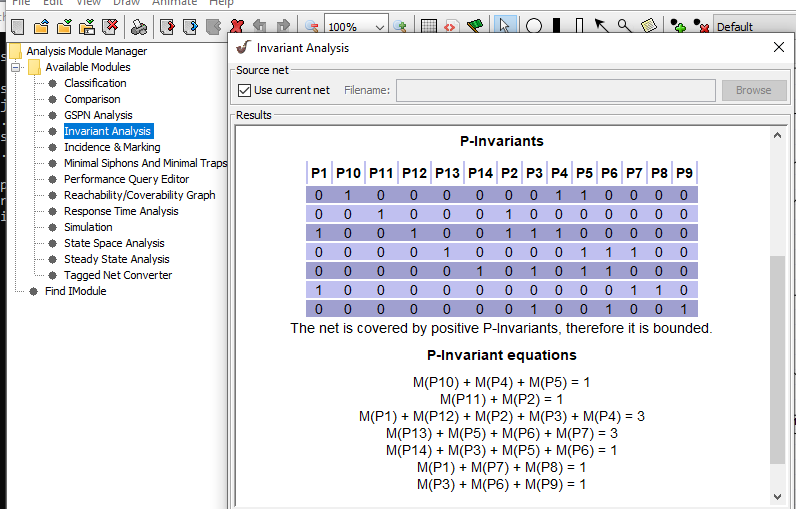


# XI) Verificación de los invariantes de plaza

Para verificar los invariantes de plaza:

* Se dispara una transición.
* Se calcula el nuevo marcado.
* Finalmente se verifica si las ecuaciones de invariante de plaza se cumplen.

Los invariantes se obtienen utilizando **Pipe**, como se observa en la siguiente figura:



Por ejemplo:



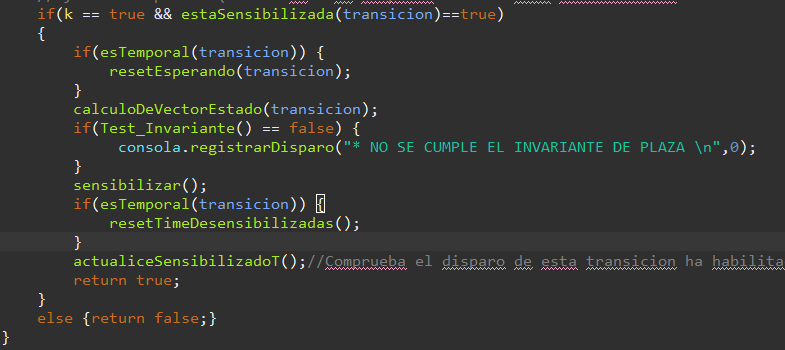
Para el siguiente invariante:



Se comprueba:

 +  +  = 1

De esta forma se procede sucesivamente con todos los invariantes de plaza.



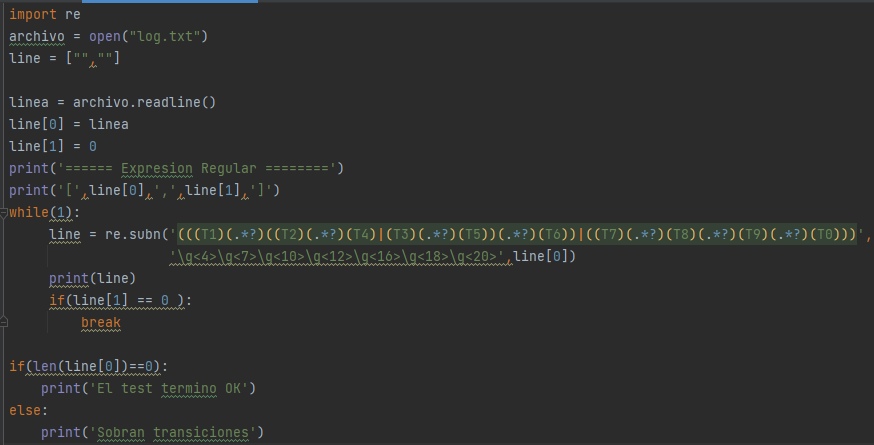
# XII) Expresiones regulares

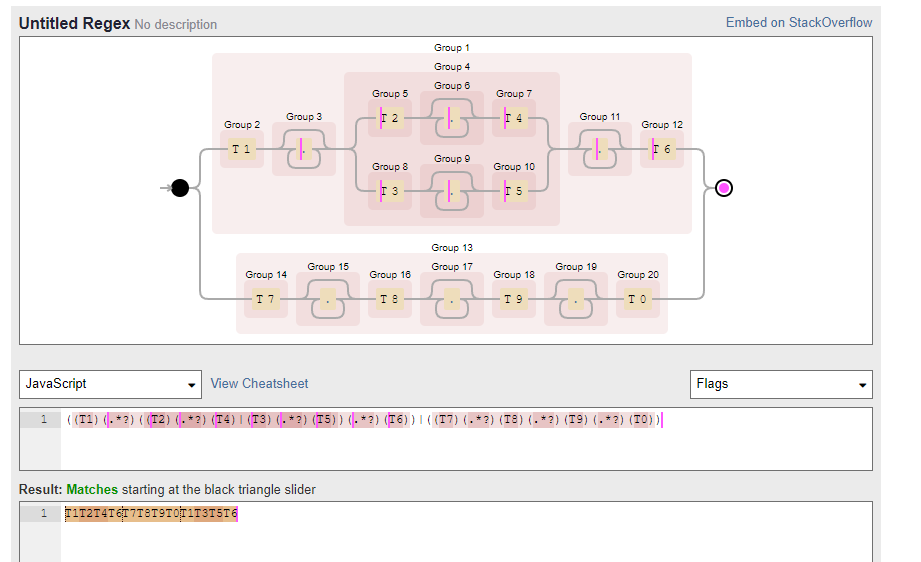
El desarrollo de verificación de la expresión regular resultante de los disparos se realiza mediante Python, utilizando la librería **re**,y formando los caminos posibles mediante la página [www.debuggex.com](http://www.debuggex.com)

Un cambio que se debió hacer para poder llegar a la expresión regular necesaria fue cambiar el nombre de una transición de la siguiente manera:



ya que para T10 la expresión regular considera T1 a T10 entonces se procedió a considerar T0 como T10. Dicha modificación no afecta el funcionamiento del programa.





# XIII) Tabla de eventos

|  |  |
| --- | --- |
| Tabla de eventos | |
| Transiciones | Eventos |
| T1 | Ingreso a la primera actividad del invariante 1, o invariante 2 |
| T2 | Token ingresando a la segunda actividad del Invariante 1 |
| T3 | Token ingresando a la segunda actividad del invariante 2 |
| T4 | Token saliendo de la segunda actividad del invariante 1 |
| T5 | Token saliendo de la segunda actividad del invariante 2 |
| T6 | Token saliendo del invariante 1 o del invariante 2 |
| T7 | Token ingresando a la primera actividad invariante 3 |
| T8 | Token ingresando a la segunda actividad invariante 3 |
| T9 | Token ingresando a la tercera actividad invariante 3 |
| T10 | Token saliendo del invariante 3 |

# XIV) Tabla de estados

|  |  |
| --- | --- |
| Tabla de estados o actividades | |
| Plazas | Estados |
| P1 | Primera actividad antes de proseguir por el invariante de transición 1, o el invariante de transición 2 |
| P2 | Segunda actividad realizándose en el invariante de transición 1 |
| P3 | Segunda actividad realizándose en el invariante de transición 2 |
| P4 | Tercera actividad realizándose ya sea del invariante de transición 1, o del 2 |
| P5 | Primera actividad del invariante 3 |
| P6 | Segunda actividad del invariante 3 |
| P7 | Tercera actividad del invariante 3 |
| P8 | Token disponible para el avance hacia el invariante 1, o el invariante 2 |
| P9 | Token disponible para el avance hacia la segunda actividad del invariante de transición 2, o la segunda actividad del invariante de transición 3 |
| P10 | Token disponible para el avance hacia la tercera actividad del invariante de transición 1, o 2, o la primera actividad del invariante de transición 3 |
| P11 | Token disponible para el avance del invariante 2 hacia la segunda actividad |
| P12 | Tokens disponibles para el avance por el invariante 1, o el invariante 2 hacia la primera actividad |
| P13 | Tokens disponibles para el avance por el invariante 3 hacia la primera actividad |

# XV) Conclusión

Se observó que el uso de un monitor y una red de Petri permite controlar la exclusión mutua, de manera tal, que todo se encuentre en un solo lugar. Tener el manejo de la sección critica (algo que anteriormente se realizaba en varias partes en el código mediante el uso de semáforos, lock, synchronized, etcétera), ahora se encuentra en el monitor. La red permite obtener una expresión matemática de la simulación, en base a matrices y vectores. Los problemas encontrados fueron:

* hallar la mejor red, es decir, la que paralelice la mayor cantidad de procesos posibles
* elegir la cantidad de hilos, los cuales, en algunos casos, fueron 4 o 5
* el tiempo en el cual debe suceder un evento (disparar la transición), manteniendo la política.

Los resultados finales fueron los esperados. Por un lado, se tienen redes de menor paralelismo, pero en ellas se pueden manejar de manera más sencilla los tiempos y el balance de los invariantes; en cambio, cuanto más se desea que la red sea paralela, más dificultoso se vuelve poder encontrar tiempo para que resulte equitativo el promedio de disparos de las transiciones de los invariantes en el tiempo. Por ejemplo, para el caso de 1000 ejecuciones.

Observacion:

Pasos para agregar una nueva red creada en pipe:

Primero obtener el archivo HTML desde pipe en la opción “Incidence & Marking” luego ejecute el programa de Python que permite obtener las matrices ordenadas para cargar en el programa principal en java luego modifique el archivo P\_invariantes con las nuevas ecuaciones las cuales se obtiene desde pipe en la opción “Invariant Analysis”, haciendo estos pasos se podrá ejecutar la nueva red.

Repositorio:

https://github.com/Jonathan684/Programacion-Concurrente.git