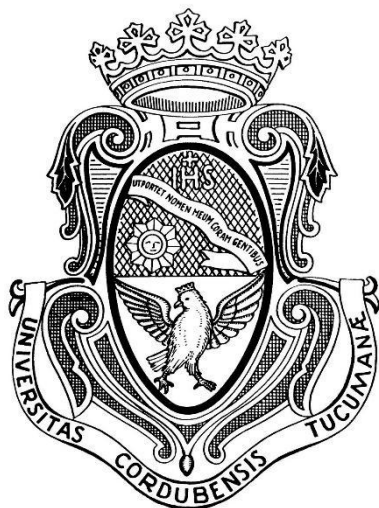


UNIVERSIDAD NACIONAL DE CÓRDOBA  
FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES



## Trabajo Práctico N° 3

Programación Concurrente

Año 2022

Profesores:

- Micolini, Orlando
- Ventre, Luis Orlando
- Ludemann, Mauricio

Alumnos	Matrícula
García, Bruno Emilio	39449179
Losano Quintana, Juan Cruz	39621464

<b>Enunciado</b>	<b>3</b>
Problema	3
Consignas	3
<b>Modelo y propiedades de la red.</b>	<b>4</b>
Propiedades de la red	5
<b>Política</b>	<b>6</b>
<b>Invariantes</b>	<b>7</b>
Invariantes de plaza	7
Invariantes de transición	8
<b>Cantidad de hilos</b>	<b>8</b>
<b>Cumplimiento de restricciones</b>	<b>10</b>
P-invariantes	10
T-invariantes	11
<b>Ejecuciones</b>	<b>13</b>
Núcleo 2 = Núcleo 1	13
Núcleo 2 con doble tiempo de servicio	14
Núcleo 2 con triple tiempo de servicio	14
Análisis de tiempos	15
<b>Conclusiones</b>	<b>15</b>
Cantidad de hilos	15
Uso de ventana de tiempo	16
Expresión regular	16
Implementación de la política	16

# Enunciado

## Problema

Se implementará un simulador de un procesador de dos núcleos en el cual, a partir de la red de Petri de la figura 1 propuesta para un procesador mono núcleo, se extenderá para el caso ya mencionado.

Además, se usarán políticas para resolver los conflictos que se generan con las transiciones que alimentarán a los buffers de los núcleos.

Las transiciones “Arrival\_Rate” y “Service\_rate” son temporizadas y representan el tiempo de arribo de procesos y el tiempo de servicio que tendrá para concluir una tarea.

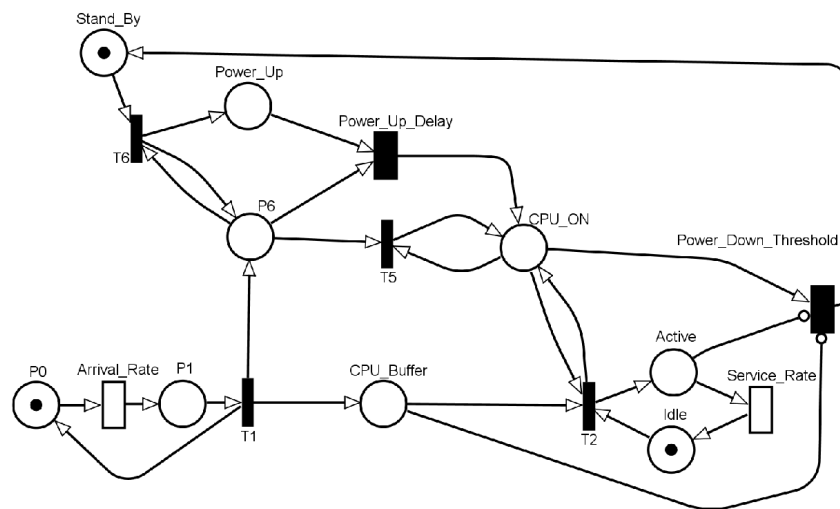


Figura 1. Modelo de Red de Petri de una CPU de 1 núcleo.

## Consignas

- Modelar el sistema de un procesador con dos núcleos, extendiendo la RdP de la *Figura 1* y verificar todas sus propiedades haciendo uso de la herramienta “*Petrinator*”.
- Hacer un diagrama de clases que modele el sistema.
- Implementar un objeto “Política” que resuelva el conflicto entre las transiciones sensibilizadas que alimentan los buffers de los núcleos, manteniendo la carga de la CPU equitativa.
- Hacer un diagrama de secuencia que muestre el disparo exitoso de una de las transiciones que alimenta a uno de los buffers de la CPU, mostrando el uso de la política.
- Indicar la cantidad de hilos necesarios para la ejecución y justificar.
- Modelar el sistema con objetos en Java.

- ## Modelo y propiedades de la red.

Las transiciones T0, T6 y T13 son con tiempo.

## Propiedades de la red

Types of Petri net	
State Machine	false
Marked Graph	false
Free Choice Net	false
Extended FCN	false
Simple Net	false
Extended SN	false

Mathematical properties	
Bounded	false
Safe	false
Deadlock	false

Figura 3. Propiedades de la red.

- **State Machine:** falso, no se corresponde con una RdP del tipo State Machine porque los arcos de entrada a las transiciones son mayor a 1 para algunas, y los arcos de salida de las transiciones son mayor a 1 para algunas transiciones.
- **Marked Graph:** falso, no se corresponde con una RdP del tipo Grafo de Marcado, debido a que los arcos de entrada a algunas plazas son mayor a 1 y a su vez los arcos de salida de algunas plazas son mayor a 1.
- **Free Choice Net:** falso, no se corresponde con una RdP del tipo Free Choice porque hay conflictos en los que tengo otros lugares de entrada. Por ejemplo el conflicto  $P6\{T4, T5\}$   $T4$  posee además el lugar de entrada  $P3$  y  $T5$  posee además el lugar  $P2$  y  $P7$ .
- **Extended Free Choice:** falso, no se corresponde con una RdP del tipo Free Choice extendida porque para cada conflicto de transiciones, estas no tienen los mismos lugares de entrada. Se puede ver en  $P6\{T4, T5...\}$   $P3$  tiene un arco de salida hacia  $T4$  pero no hacia  $T5$  y  $P2$  tiene un arco de salida hacia  $T5$  pero no hacia  $T4$ .

- **Simple Net:** falso, no se corresponde con una RdP del tipo Simple porque hay transiciones que se ven afectadas por más de 1 conflicto. Por ejemplo,  $P6\{T4, T5...\}$  la transición T4 también se encuentra en el conflicto  $P3\{T3, T4...\}$
- **Extended Simple Net:** falso, no se corresponde con una RdP del tipo Simple extendida porque donde hay más de 1 conflicto la transición no incluye a todo el conjunto de plazas del otro conflicto.
- **Bounded:** falso, no se corresponde con una RdP acotada porque hay plazas que pueden llegar a tener un número de tokens infinito, es el caso de P2 y P9
- **Safe:** falso, no se corresponde con una RdP segura porque hay momentos en los que el número de tokens de algunas plazas es mayor a 1.
- **Deadlock:** falso, no se corresponde con una RdP con Deadlock porque el grafo de alcanzabilidad no me da una posibilidad en la cual no se pueda disparar ninguna transición. A cualquier marcado que se llegue siempre se me sensibilizara alguna transición y podrá ser disparada.

## Política

La política que se utilizo es una tal que sirva para distribuir la carga de los núcleos equitativamente, es decir a medida que se van llenando los buffers de cada núcleo se señala al hilo que controla T1 para pasar la tarea al núcleo 1, o al que controla T8 para pasar la tarea al núcleo 2. Cuando el núcleo 2 tenga más tareas que el núcleo 1 se señala T1, cuando ambos tengan igual cantidad de tareas se señala T8. Además los buffers de los núcleos tienen un máximo de almacenamiento de 10 tareas, si el buffer del núcleo 1 se llena se desensibiliza T1, si el buffer del núcleo 2 se llena se desensibiliza T8. Siempre se le dará prioridad a los hilos que controlan a T1 y T8 por sobre los demás hilos.

El método que decide el señalizado del hilo es el siguiente:

```

public int cualDespierto(int[][] vectorEncolados, int[][] vectorEx
                        , int[][] vectorM) {

    if (vectorEncolados[T1][0] == 1 && vectorEncolados[T8][0] == 1
        && vectorEx[T1][0] == 1) {
        if (vectorM[P9][0] > vectorM[P2][0])
            return T1;
        else
            return T8;
    }

    for (int i = 0; i < CANT_TRANSICIONES; i++) {
        if (vectorEncolados[i][0] == 1 && vectorEx[i][0] == 1)
            return i;
    }
    return -1;
}

```

## Invariantes

### Invariantes de plaza

Los invariantes de plaza son los siguientes:

#### **P-Invariant equations**

$$\begin{aligned}
 M(p_0) + M(p_1) &= 1 \\
 M(p_{11}) + M(p_{12}) + M(p_{13}) &= 1 \\
 M(p_{14}) + M(p_{15}) &= 1 \\
 M(p_4) + M(p_5) + M(p_6) &= 1 \\
 M(p_7) + M(p_8) &= 1
 \end{aligned}$$

Figura 4. P-invariantes

Estos se corresponde a que cuando ocurra cualquier disparo, cada una de estas ecuaciones se va a cumplir, Por ejemplo si se dispara la transición T0, ocurre que la marca de P0 sumado a la marca de P1, va a dar un resultado de 1, siempre, y así para las demás ecuaciones.

## Invariantes de transición

Los invariantes de transición son los siguientes:

T-Invariants														
t9	t11	t10	t12	t14	t2	t3	t4	t5	t7	t1	t8	t0	t6	t13
0	1	0	1	0	0	0	0	0	0	0	1	1	0	1
1	0	1	1	1	0	0	0	0	0	0	1	1	0	1
0	0	0	0	0	1	1	0	1	1	1	0	1	1	0
0	0	0	0	0	0	0	1	1	0	1	0	1	1	0

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

Figura 5. T-invariantes

Así los T-invariantes son:

T0 T1 T2 T3 T5 T6 T7

T0 T1 T4 T5 T6

T0 T8 T9 T10 T12 T13 T14

T0 T8 T11 T12 T13

Cada una de estas secuencias de disparo cumple con la propiedad de, al dispararse, quedar con el mismo marcado de antes de haber disparado la secuencia y así poder repetirse siempre.

## Cantidad de hilos

Se utilizan 9 hilos para esta Red de Petri, donde cada hilo controla 1 o más transiciones:

- Thread-0 : T0
- Thread-1 : T1
- Thread-2 : T8
- Thread-3 : T2, T3, T7
- Thread-4 : T4
- Thread-5 : T5, T6
- Thread-7 : T9, T10, T14



- Thread-8 : T11
- Thread-9 : T12, T13

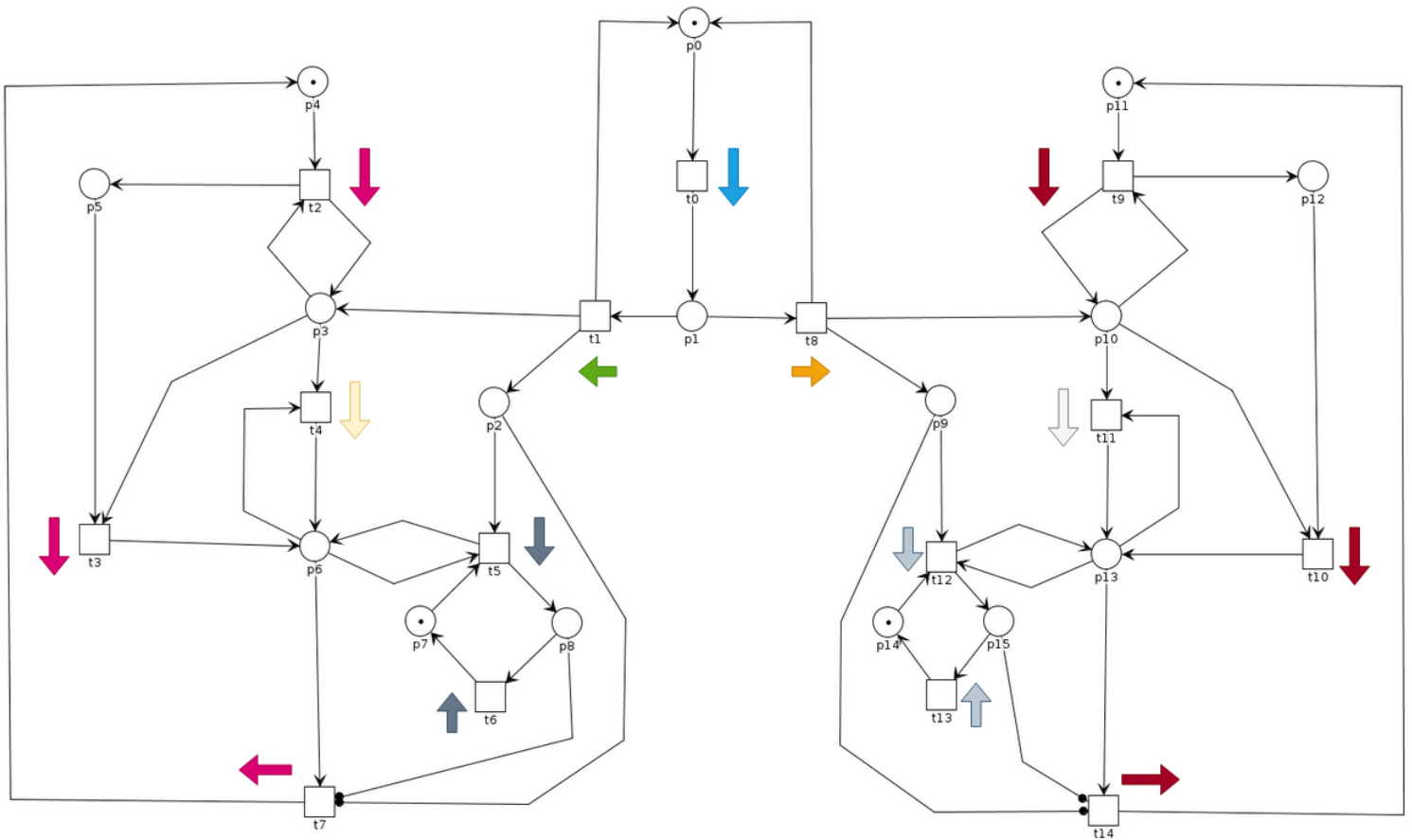


Figura 6. Hilos por transición

La elección de los hilos se dio así debido a los invariantes de transición y al conflicto que presentan estos invariantes.

Se puede ver que el invariante  $\{T0\ T1\ T2\ T3\ T5\ T6\ T7\}$  se encuentra en conflicto con  $\{T0\ T8\ T9\ T10\ T12\ T13\ T14\}$  luego de disparar  $T0$ , por eso  $T0$  es controlado por un hilo ya que no presenta conflicto, pero  $T1$  y  $T8$  si presentan conflicto así que se le da 1 hilo distinto a cada transición.

El invariante  $\{T0\ T1\ T2\ T3\ T5\ T6\ T7\}$  presenta conflicto con  $\{T0\ T1\ T4\ T5\ T6\}$

Luego de disparar T1, T4 pasa a controlarse por un hilo distinto al de T2 T3 T7, y parecería que T5 y T6 deberían ser controladas por el mismo hilo de T4 pero esto no puede ser así, porque luego de disparar T2 T3 o T4 hay una unión entre los dos invariantes donde se procede a disparar T5 T6 en ambos. Entonces se utiliza 1 hilo para T5 y T6 debido a que tengo 1 solo token como máximo en la plaza donde se unen ambos invariantes. T7 es controlada por el mismo hilo de T2 y T3 porque es parte del proceso del apagado y encendido del núcleo.

Esto mismo anterior sucede para el conflicto entre {T0 T8 T9 T10 T12 T13 T14} y {T0 T8 T11 T12 T13}

## Cumplimiento de restricciones

### P-invariantes

Se utilizó el siguiente método en el programa para verificar que se cumplan los p-invariantes:

```
private boolean checkInvariantePlaza() {
    vectorM = rdp.getVectorM();
    short invariante = 1;

    if (vectorM[0][0] + vectorM[15][0] != invariante)
        return false;
    if (vectorM[1][0] + vectorM[3][0] + vectorM[4][0] !=
invariante)
        return false;
    if (vectorM[5][0] + vectorM[7][0] != invariante)
        return false;
    if (vectorM[8][0] + vectorM[10][0] + vectorM[14][0] !=
invariante)
        return false;
    if (vectorM[12][0] + vectorM[13][0] != invariante)
        return false;

    return true;
}
```

## T-invariantes

Se utilizó un script de python que usa una expresión regular para verificar que se cumplan los T-invariantes. El script de python es el siguiente:

```
import re
import io

from more_itertools import rstrip

txt = ''

for i in open("t_invariante.txt", "r"):
    txt = txt + i

def invariantes(txt):
    line = [txt, 0]
    while(True) :

        line =
re.subn('T0(.*)?(?:Tu(.*)?(?:T5(.*)?(?:T6(.*)T4|T4(.*)T6)|T4(.*)T5(.*)T6
|(?:(?:T2(.*)T3(.*)T5(.*)T6(.*)|T5(.*)T6(.*)T2(.*)T3(.*)T7)|(?:(?:T8(.*)
(?:T12(.*)T13(.*)T11|T11(.*)T13)|T11(.*)T12(.*)T13|(?:(?:T9(.*)T10(.*)
)T12(.*)T13(.*)|T12(.*)T13(.*)T9(.*)T10(.*)T14)))',

'\g<1>\g<2>\g<3>\g<4>\g<5>\g<6>\g<7>\g<8>\g<9>\g<10>\g<11>\g<12>\g<13>\g<14>
\g<15>\g<16>\g<17>\g<18>\g<19>\g<20>\g<21>\g<22>\g<23>\g<24>\g<25>\g<26>\g<2
7>\g<28>\g<29>', line[0].rstrip())

        if(line[1] == 0):
            break

        if(line[0] == ""):
            print("Test de invariantes exitoso")
        else:
            print("Error de invarianteT")
            print(line[0])

invariantes(txt)
```

La expresión regular usada fue la siguiente:

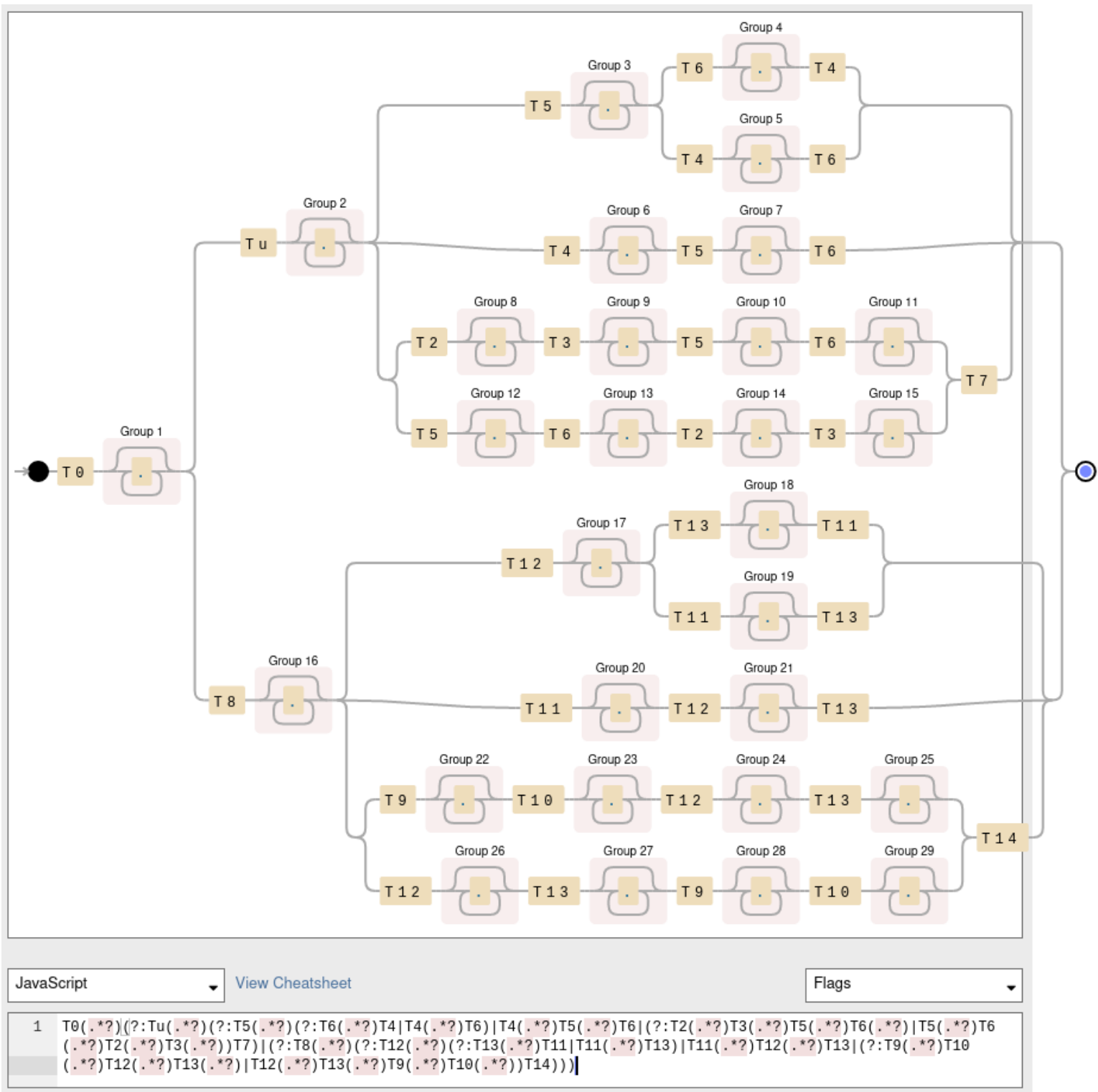


Figura 7. Expresión regular

El script de python se encarga de encontrar invariantes utilizando la expresión regular mencionada, cuando encuentra un invariante lo elimina y se queda con la captura de los grupos que no pertenecen a este invariante, así vuelve a correr la verificación y trata de encontrar otro invariante. Si cuando termina la ejecución, el script se queda

con un residuo, entonces no se cumple la verificación de t-invariantes, en cambio si no queda ningún residuo si se cumple, debido a que todos los disparos que se han hecho pertenecen a algún invariante.

## Ejecuciones

La Red de Petri tiene las transiciones T0, T6 y T13 con tiempo, estas transiciones se verán sensibilizadas por token y además por tiempo. Si se va a disparar alguna de estas transiciones debe tener los token necesarios y además encontrarse dentro de la ventana de tiempo para poder dispararse. Las ventanas de tiempo elegidas fueron las siguientes:

T0 : ALPHA\_ARRIBO = 20 [ms]

T0 : BETA\_ARRIBO = 2000 [ms]

T6 : ALPHA\_SERVICIO1 = 60 [ms]

T6 : BETA\_SERVICIO1 = 6000 [ms]

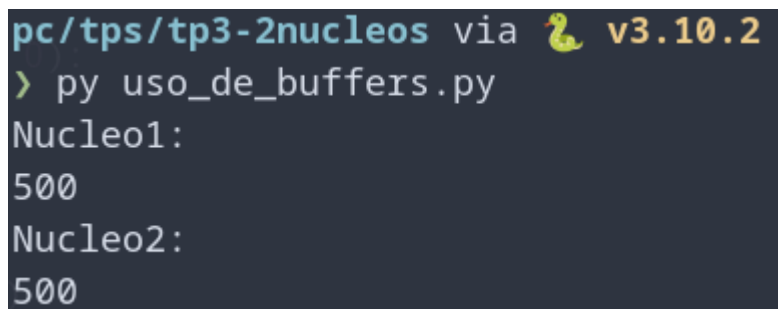
T13 : ALPHA\_SERVICIO2 = 60[ms]

T13 : BETA\_SERVICIO2 = 6000[ms]

Se realizaron 3 ejecuciones donde la primera ejecución se le dio el mismo tiempo a ambos núcleos, la segunda ejecución se le dio el doble de tiempo al núcleo 2, y la tercera ejecución se le dio el triple de tiempo al núcleo 2.

Núcleo 2 = Núcleo 1

Tiempo de ejecución: 30251 [ms]



```
pc/tps/tp3-2nucleos via 🐍 v3.10.2
> py uso_de_buffers.py
Nucleo1:
500
Nucleo2:
500
```

Figura 8. Uso de núcleos (núcleo 1 = núcleo 2)

## Núcleo 2 con doble tiempo de servicio

Tiempo de ejecución: 40603 [ms]

```
pc/tps/tp3-2nucleos via 🐍 v3.10.2
> py uso_de_buffers.py
Nucleo1:
663
Nucleo2:
337
```

Figura 9. Uso de núcleos (2 x núcleo 1 = núcleo 2)

## Núcleo 2 con triple tiempo de servicio

Tiempo de ejecución: 46219 [ms]

```
pc/tps/tp3-2nucleos via 🐍 v3.10.2
> py uso_de_buffers.py
Nucleo1:
744
Nucleo2:
256
```

Figura 10. Uso de núcleos (3 x núcleo 1 = núcleo 2)

Se puede concluir que a medida que el tiempo de servicio de uno de los núcleos aumenta, la cantidad de tareas que tratara ese núcleo sera menor, aumentando así también el tiempo total de ejecución del total de las tareas.

## Análisis de tiempos

Analizando los tiempos anteriores vemos que en el primer caso dónde ambos núcleos demoran 60 [ms] en procesar, el tiempo promedio del programa en procesar 1.000 eventos es de 30 [seg] y esto tiene sentido ya que cada núcleo procesa 500 eventos  $\times 60 \text{ [ms]} = 30.000 \text{ [ms]}$ . En el caso de que un núcleo demore el doble del tiempo pero la política para utilización de los núcleos sea por disponibilidad vemos que  $663 \text{ eventos} \times 60 \text{ [ms]} \sim 40.000 \text{ [ms]} \sim 337 \text{ eventos} \times 120 \text{ [ms]}$ , procesando el núcleo que demora menos aproximadamente  $2/3$  de los eventos y el otro  $1/3$ , esta misma lógica aplica para el caso de que un núcleo demore el triple del tiempo.

## Conclusiones

Al realizar el proyecto nos encontramos con distintos inconvenientes para poder resolverlo, a continuación vamos a detallar alguno de estos inconvenientes.

### Cantidad de hilos

Al ser de un año anterior a uno de los años donde se explicó la relación entre hilos e invariantes, tuvimos la dificultad de no saber cuántos hilos deberíamos usar, en un principio creíamos que lo óptimo eran 5 hilos, pero pudimos sacar la cuenta de los hilos al ver lo siguiente:

- **Determinar la cantidad de hilos necesarios para la ejecución del sistema con el mayor paralelismo posible:**
  - Caso 1: si el invariante de transición no tiene conflictos, un hilo debe estar encargado de la ejecución de las transiciones de ese invariante.
  - Caso 2: si el invariante de transición tiene un conflicto, con otro invariante, debe haber un hilo encargado de la ejecución de la/s transición/es anterior/es al conflicto y luego un hilo por invariante.
  - Caso 3: si el invariante de transición presenta un join, con otro invariante de transición, luego del join debe haber tantos hilos, como token simultáneos en la plaza, encargados de las transiciones restantes dado que hay un solo camino.

Figura 11. cantidad de hilos a utilizar

## Uso de ventana de tiempo

Tuvimos problemas para entender el diagrama de secuencia de un monitor que controla una red de petri con transiciones temporizadas, debido a que las clases del teórico presenciales al no poderse re-ver no quedaban muy claros algunos conceptos, por lo contrario con el video donde explica las redes de petri con tiempo y el uso del monitor con tiempo, quedo mucho mas claro.

Tuvimos problemas en la clase CdT (control de tiempo) con el uso de las ventanas de tiempo, donde se nos confundian las ventanas y la de T0 pasaba a ser la de T6 o T13 y el programa empezaba a funcionar mal.

## Expresión regular

Tuvimos problemas con el armado de la expresión regular del testeo de T-invariantes debido a que no funcionaba nunca el testeo de T-invariantes porque nos faltaba un posible camino que no teníamos en cuenta. Por ejemplo había veces que el núcleo se prendia, trataba la tarea y le llegaba otra tarea y no hacía uso del garbage collector(T4, T11) entonces trataba la otra tarea y para sacar el token del garbage collector se volvía a prender y a apagar, este camino no lo teníamos en cuenta.

## Implementación de la política

Podemos decir que con la política no tuvimos muchos inconvenientes porque era algo sencillo, solo mandar las tareas a un núcleo o al otro viendo que tan cargado está cada uno. Pero como tuvimos problemas con la expresión regular y el script de python en un momento tuvimos la idea de asignar distintas prioridades dentro de la política y preguntar por más hilos específicamente, no solo los que controlan T1 y T8. Esto no se dio así porque pudimos solucionar nuestros inconvenientes.