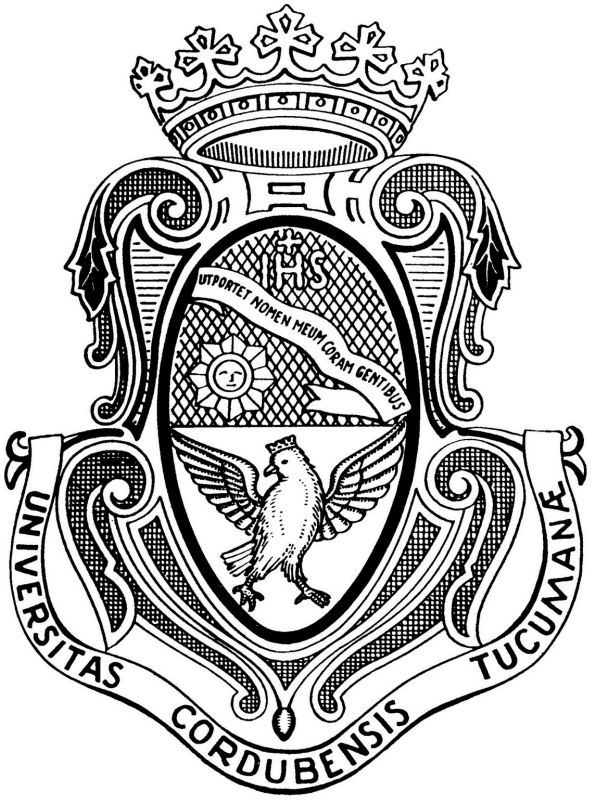
Universidad Nacional de Córdoba
Facultad de Ciencias Exactas, Físicas y Naturales



Trabajo Práctico Integrador

Programación Concurrente

Docentes:

Dr. Ing. Orlando Micolini

Ing. Luis Orlando Ventre

Alumnos:

Navarro, Sebastián

navarrosebastian95@gmail.com

Piñero, Tomás Santiago

tom-300@hotmail.com

Año 2019

Índice

1. Enunciado

En este práctico se debe resolver el control de acceso a una playa de estacionamiento con 3 entradas (calles) diferentes. En esta playa hay 2 pisos, y en cada piso pueden estacionar 30 autos. La playa cuenta con 2 salidas diferentes y una única estación de pago (caja). En los accesos a la playa y en los egresos existen barreras que deben modelarse.

La playa cuenta con lugares (3) donde los vehículos se detienen cuando quieren entrar (barrera), una vez que ingresaron se les indica un piso y estacionan (puede ser piso 1 o piso 2). Se debe cuidar que no se permita el ingreso (superar barrera) a más vehículos de los espacios disponibles totales.

Los autos que se retiran de la playa deben liberar un espacio del piso en que se encontraban (diferenciar estacionamiento en cada piso). Cuando un vehículo se va a retirar puede optar por salida a calle 1 o salida a calle 2. Luego debe abonar la estadía. El cobro de la estadía le lleva a un empleado promedio al menos 3 minutos. (Existe una sola caja)

En caso de que la playa esté llena, se debe encender un cartel luminoso externo que indica tal situación.

El sistema controlador debe estar conformado por distintos hilos, los cuales deben ser asignados a cada conjunto de responsabilidades afines en particular. Por ejemplo: Ingreso de vehículos, manejo de barreras, etcétera.

Debe realizar:

1. La red de Petri que modela el sistema.
2. Agregar las restricciones necesarias para evitar interbloqueos ni accesos cuando no hay lugar, mostrarlo con la herramienta elegida y justificarlo.
3. Simular la solución en un proyecto desarrollado con la herramienta adecuada (explique porque eligió la herramienta usada).
4. Colocar tiempo en las estación de pago caja (en la/s transición/es correspondiente/s).
5. Hacer la tabla de eventos.
6. Hacer la tabla de estados o actividades.
7. Determinar la cantidad de hilos necesarios (justificarlo)
8. Implementar dos casos de Políticas para:
   - Prioridad llenar de vehículos planta baja (piso 1) y luego habilitar el piso superior. Prioridad salida indistinta (caja).
   - Prioridad llenado indistinta. Prioridad salida a calle 2.
9. Hacer el diagrama de clases.
10. Hacer los diagramas de secuencias.
11. Hacer el código.
12. Hacer el *testing*.

2. Desarrollo

2.1. Tabla de estados

Numero	Plaza	Estado
P0	Limitador clientes calle 1	Buffer que limita la cantidad de clientes a ingresar por la primer calle
P1	Limitador clientes calle 2	Buffer que limita la cantidad de clientes a ingresar por la segunda calle
P2	Limitador clientes calle 3	Buffer que limita la cantidad de clientes a ingresar por la tercer calle
P3	Autos esperando ingresar calle 1	Cola de autos esperando para sacar el ticket en la primer calle
P4	Autos esperando ingresar calle 2	Cola de autos esperando para sacar el ticket en la segunda calle
P5	Autos esperando ingresar calle 3	Cola de autos esperando para sacar el ticket en la tercer calle
P6	Auto pasando barrera 1	El cliente sacó el ticket y la barrera se levanta
P7	Auto pasando barrera 2	El cliente sacó el ticket y la barrera se levanta
P8	Auto pasando barrera 3	El cliente sacó el ticket y la barrera se levanta
P9	Barrera de calle 1	Mutex para el uso de la barrera de la primer calle
P10	Barrera de calle 2	Mutex para el uso de la barrera de la segunda calle
P11	Barrera de calle 3	Mutex para el uso de la barrera de la tercer calle
P12	Limitador de autos	Buffer que limita la cantidad de autos a utilizar la rampa
P13	Autos por tomar rampa	Autos utilizando la rampa
P14	Auto buscando lugar arriba	Cliente buscando lugar para estacionar el auto en planta alta
P15	Auto buscando lugar abajo	Cliente buscando lugar para estacionar el auto en subsuelo
P16	Rampa	Mutex de uso de la rampa
P17	Auto estacionado arriba	Auto estacionado en la planta de arriba
P18	Auto estacionado abajo	Auto estacionado en el subsuelo
P19	Lugares disponibles arriba	Cantidad de lugares disponibles en la planta alta
P20	Lugares disponibles abajo	Cantidad de lugares disponibles en el subsuelo
P21	Cliente por bajar	Cliente sale de la planta alta y se dirije a la caja
P22	Cliente por subir	Cliente sale del subsuelo y se dirije a la caja
P23	Auto en rampa	Cliente esta bajando o subiendo por la rampa
P24	Cliente por pagar	Cliente esperando para pagar el ticket
P25	Cliente pagando	Cliente pagando el ticket

P26	Cajero	Cajero
P27	Auto por salir	Auto por abonar el ticket
P28	Limitador de playa	Buffer que limita la cantidad de clientes que pueden dirigirse a la caja
P29	Cartel encendido	No hay mas lugar en la playa
P30	Cartel apagado	Hay lugar en la playa

Cuadro 1: Tabla de estados.

2.2. Tabla de eventos

Numero	Plaza	Estado
T0	Entrar calle 1	Autos ingresando por la primer calle
T1	Entrar calle 2	Autos ingresando por la segunda calle
T2	Entrar calle 3	Autos ingresando por la tercer calle
T3	Sacar ticket 1	El cliente saca el ticket para levantar la barrera
v4	Sacar ticket 2	El cliente saca el ticket para levantar la barrera
T5	Sacar ticket 3 3	El cliente saca el ticket para levantar la barrera
T6	Levantar barrera 1	Una vez sacado el ticket, se levanta la barrera para el ingreso a la playa
T7	Levantar barrera 2	Una vez sacado el ticket, se levanta la barrera para el ingreso a la playa
T8	Levantar barrera 3	Una vez sacado el ticket, se levanta la barrera para el ingreso a la playa
T9	Subir por rampa	El cliente sube a la planta alta del estacionamiento
T10	Bajar por rampa	El cliente va a la planta baja del estacionamiento
T11	Buscar lugar arriba	El cliente busca lugar para estacionar en la planta alta
T12	Buscar lugar abajo	El cliente busca lugar para estacionar en la planta baja
T13	Aguardar salida arriba	El cliente dejó el auto estacionado en la planta alta
T14	Aguardar salida abajo	El cliente dejó el auto estacionado en la planta baja
T15	Bajar	El cliente de la planta alta se dirige a la salida
T16	Subir	El cliente de la planta baja se dirige a la salida
T17	Esperar caja	El cliente se dirige a la caja
T18	Pagar	El cliente paga el ticket en la caja
T19	Devolver cajero	El cliente elige la salida
T20	Salir por calle 1	El cliente sale por la calle 1
T21	Salir por calle 2	El cliente sale por la calle 2
T22	Encender el cartel 2	Se llenó la playa, se enciende el cartel
T23	Apagar cartel 2	Hay lugar en la playa, se apaga el cartel

Cuadro 2: Tabla de eventos.

2.3. Hilos

Los hilos son aquellos que realizan un conjunto de acciones, por lo que para determinar la cantidad de hilos necesarios, se realizó una modularización de la red en actividades, por lo tanto, la cantidad de hilos utilizados son 9:

1. Entrada calle 1: T0 - T3.
2. Entrada calle 2: T1 - T4.
3. Entrada calle 3: T2 - T5.
4. Acceder a la playa: T6 - T7 - T8.
5. Planta Alta: T9 - T11 - T13 - T15.
6. Planta Baja: T10 - T12 - T14 - T16.
7. Cobrar tickets: T17 - T18 - T19.
8. Salida: T20 - T21.
9. Cartel: T22 - T23.

Hilos 1, 2 y 3

Los 3 hilos iniciales en las entradas son de existencia trivial, donde cada calle (o entrada) es independiente una de la otra, y cada una cuenta con sus propios recursos. Estos hilos representan a los autos pasando las barreras.

Hilo 4

La segunda etapa de la red, el tramo que abarca el acceso a la playa. Este hilo resulta indispensable para la fluidez de la red. Al ser un conjunto de transiciones de mucha demanda, resulta necesario contar con un hilo que tenga la mayor parte del tiempo alguna de sus transiciones sensibilizadas, de modo que éste se encole la menor cantidad de veces posible y en consecuencia, los alrededores de la red no conduzcan a un interbloqueo.

Hilos 5, 6 y 7

Se independizan esos 3 conjuntos de transiciones de la red para modularizarla lo más posible y así elevar su concurrencia, por lo que se asigna un hilo a cada piso (hilos 5 y 6, respectivamente) y otro que maneja la caja (hilo 7). Además, esta distribución conserva la linealidad de la red cuando se la compara con la realidad, resulta intuitivo pensar que la linea de Planta Alta, Planta Baja y Caja es administrado de manera completa por un hilo cada una.

Hilo 8

Se asigna un hilo a la dupla de transiciones T20 y T21 para resolver el último conflicto de la red, de modo que se puede implementar más fácilmente una política que las administre, y que además asegure un comportamiento consistente.

Hilo 9

Se encarga del manejo del cartel de la playa.

Discusiones

Aca hay que poner todo el tema de la discusion de los hilos, escribir que asignamos de una forma primero, y que antes de nada se utlizo una politica fifo, donde se producia el çuelgue", despues que se cambiaron a otra forma los hilos y se mejoro la concurrencia, pero con bloqueaos igual, luego de varias pruebas, se descubrio que el problema no era la modularizacion y asignacion de hilos, sino la politica fifo que no permitia la ejecucion correcta de la red.

2.4. Red de Petri

Al tratarse de una red que está acotada, se tiene una secuencia de disparos infinita, por lo tanto la red es monótona.

Para las transiciones temporizadas se utiliza una clase dedicada a cronometrarlas: 'Tiempos'. Una transición con tiempo se cronometra desde el inicio de su sensibilización, y en el momento en el que se la quiere disparar, se debe verificar que el tiempo dicho esté dentro de su intervalo de tiempo $[a_i, b_i]$. Si el disparo se quiere realizar antes del límite inferior (a_i), el hilo debe dormir $a_i - x$ segundos, donde x es el tiempo transcurrido desde que se sensibilizó la transición.

Semántica utilizada para las transiciones temporizadas

Se tomo la primera de las semánticas especificadas en *"Redes de Petri Temporales 2017.pdf" – Hoja 11*, por lo tanto el disparo se realiza en dos etapas:

- Transcurre un determinado tiempo desde que una transición se sensibiliza.
- Se retiran y colocan las marcas de forma atómica.

Cabe mencionar que esto se usa para ambos tipos de transiciones: inmediatas y temporales, ya que en el caso de la inmediata, el paso 1 implica un intervalo de tiempo nulo.

Al tener un arco lector, no se utiliza la matriz de incidencia combinada para realizar el cálculo del estado siguiente de la red, ya que este tipo de arco puede definirse como un arco común que entra y sale de una misma transición, por lo que se verá reflejado como un 0 en la matriz de incidencia combinada. Debido a esto se realiza el cálculo del estado siguiente en dos etapas: primero se realiza la toma de tokens utilizando la matriz de incidencia negativa y luego se realiza la inserción de tokens utilizando la matriz de incidencia positiva.

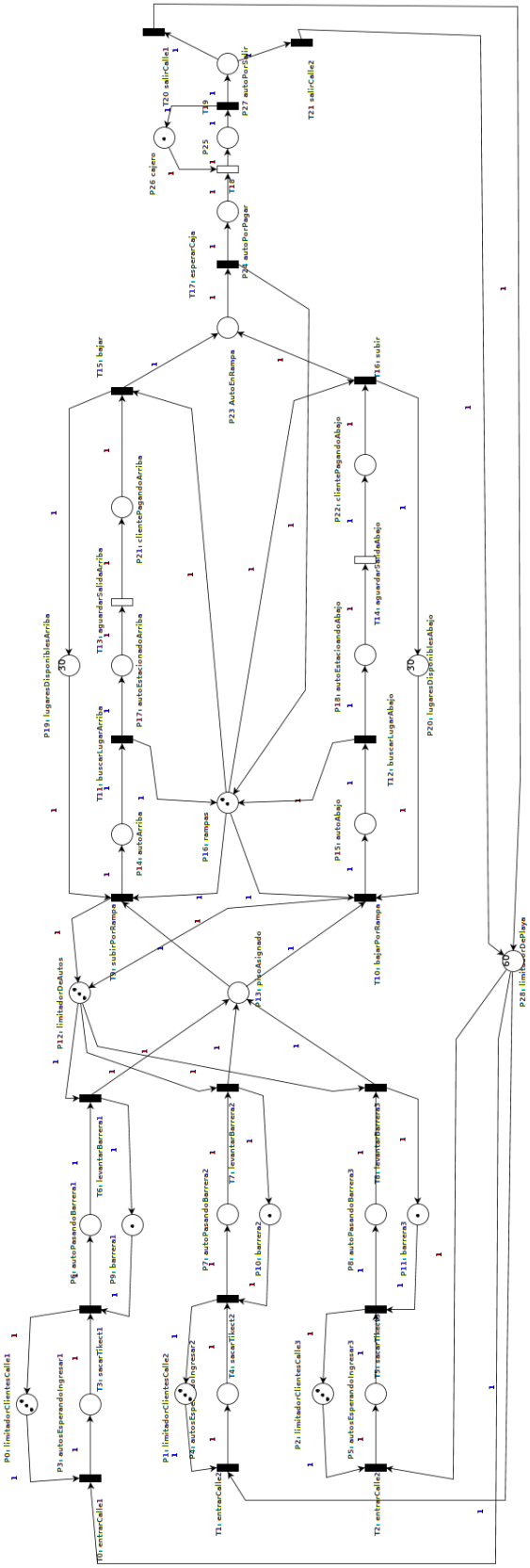


Figura 1: Red de Petri que modela el sistema.

2.5. Políticas

Aquel conjunto de transiciones pertenecientes a un hilo en particular, que no requieran el tratado de políticas específicas, se dispararan preferentemente en el orden dispuesto en el archivo de texto 'hilos.txt', siendo este un posible orden de prioridades, para garantizar el orden y la fluidez de la red. Ejemplificando, suponga un hilo que haya conseguido acceso al monitor y que lleva parametrizadas *[T0 T3 T6]*; la transición *T0* será la primera en ser evaluada en función de su estado de sensibilizada, de poder dispararse, se procede, caso contrario, se intentará con T3. Si finalmente, ninguna de las 3 transiciones del hilo puede ser disparada en ese instante, éste se encolará dentro del monitor con el primer parámetro de la lista, en este caso *T0*.

Por otro lado, las transiciones que requieren tratado de políticas especificas, son administradas por la clase 'Politicas' del proyecto y obedecen al enunciado.

Cabe mencionar que se interpretó por 'prioridad' a un piso o una salida como la opción más probable que puede llegar a suceder, por lo que para la primer política, el 70 % de los autos irá a planta baja, mientras que el 30 % restante irá a planta alta. Estos mismos porcentajes se utilizan en la segunda política para la elección de la calle de salida (70 % para la salida 2 y 30 % para la salida 1).

La elección de esta política se realiza por medio del constructor de la clase 'Politicas', donde se pasa como parámetro el número de política que se desea utilizar: 1 para la primer política ó 2 para la segunda.

2.6. Análisis de Invariantes

El análisis de invariantes es utilizado para la realización del *testing* del código. Para que éste sea exitoso ambos invariantes (de plaza y transiciones) deben cumplirse, ya que el hecho de que no se cumpla alguno implica que el sistema está funcionando de manera incorrecta. Si no se cumplen los invariantes de plaza significa que no se está respetando la cantidad de recursos existentes en la red, mientras que si no se cumplen los invariantes de transiciones se verifica que los bucles posibles que posee la red no se están realizando, por lo que los disparos están ocurriendo en manera incorrecta.

2.6.1. P-Invariantes

Un invariante P indica que el número de tokens en todas las marcas alcanzables satisface una invariante lineal. Un invariante algebraico es una cierta combinación de las componentes de una matriz cuyo valor numérico no queda alterado al hacer un cambio de base. Las invariantes la red generadas por PIPE son las siguientes:

1. Limitador de autos en la primer entrada.

   ```
   M(P0) + M(P3) = 3
   ```

2. Limitador de autos en la segunda entrada.

   ```
   M(P1) + M(P4) = 3
   ```

3. Limitador de autos en la tercer entrada.

`M(P2) + M(P5) = 3`

4. Barrera de la primer entrada.

   `M(P6) + M(P9) = 1`

5. Barrera de la segunda entrada.

   `M(P10) + M(P7) = 1`

6. Barrera de la tercer entrada.

   `M(P11) + M(P8) = 1`

7. Regulador de autos para acceso a la rampa.

   `M(P12) + M(P13) = 3`

8. Limitador de autos en planta alta.

   `M(P14) + M(P17) + M(P19) + M(P21) = 30`

9. Limitador de autos en planta baja.

   `M(P15) + M(P18) + M(P20) + M(P22) = 30`

10. Regulador de autos para acceso y egreso a las plantas de la playa.

    `M(P14) + M(P15) + M(P16) + M(P23) = 2`

11. Regulador de autos en cola para pagar.

    `M(P21) + M(P22) + M(P23) + M(P24) + M(P31) = 3`

12. Empleado de caja.

    `10- M(P25) + M(P26) = 1`

13. Cartel de la playa.

    `M(P29) + M(P30) = 1`

14. Limitador de clientes totales que pueden encontrarse en el sistema.

    `M(P13) + M(P14) + M(P15) + M(P17) + M(P18) + M(P21) + M(P22) + M(P23)`
    `+ M(P24) + M(P25) + M(P27) + M(P28) + M(P3) + M(P4) + M(P5) + M(P6)`
    `+ M(P7) + M(P8) = 60`

Luego de un disparo, y en consecuencia, luego de cada evolución de la red, se evalúa cada una de las 14 condiciones de P-Invariantes arriba listadas. Una bandera booleana inicializada en *true* permanece en ese estado siempre que el resultado del análisis sea satisfactorio, pero en caso de haber una inconsistencia en la red que viole el análisis, la bandera pasará al estado *false* y permanecerá en ese estado hasta el fin de la ejecución, indicando que como mínimo una P-Invariante no se cumplió.

2.6.2. T-Invariantes

Estas indican posibles bucles (*loops*) en la red. Es decir, una secuencia de disparos que tenga asociado un invariante de transición, vuelve al mismo marcado desde el que partió.

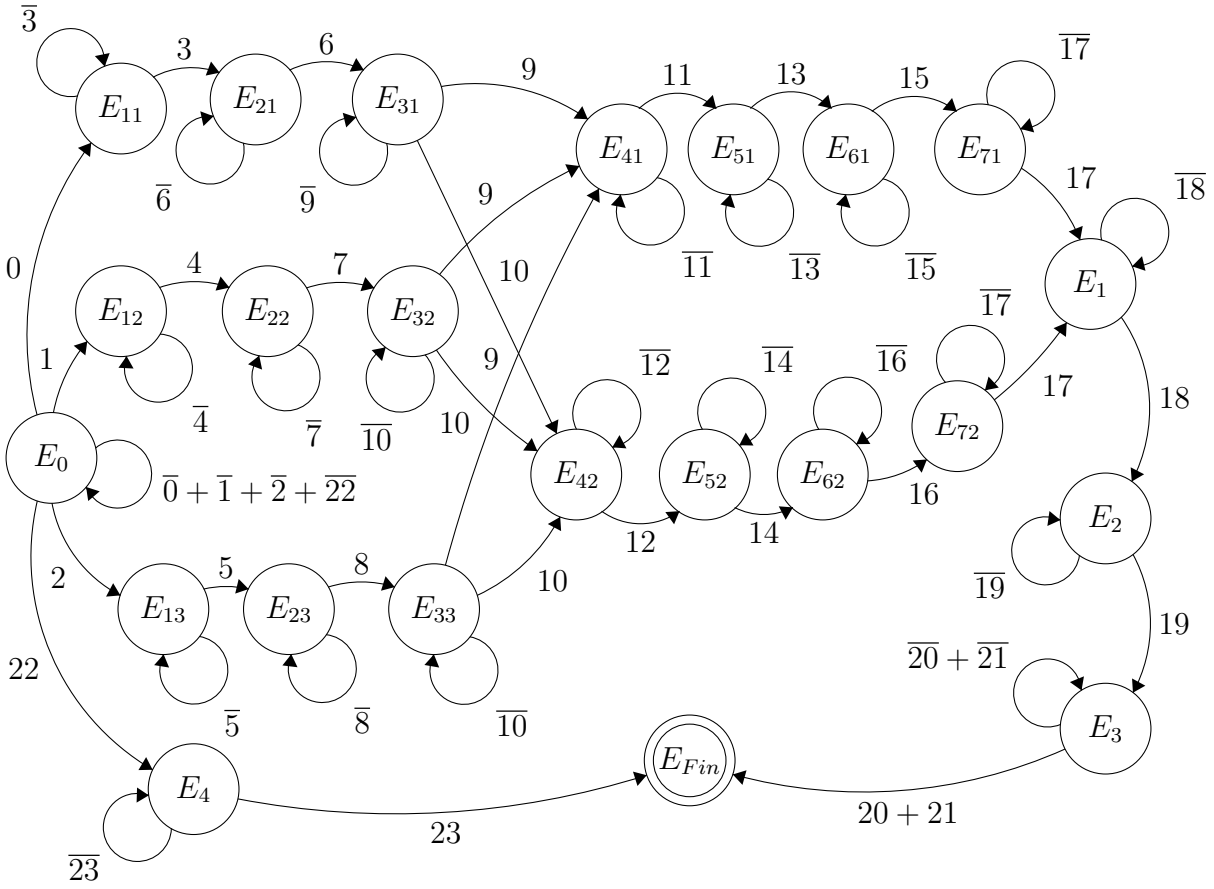
1. Entra por la primer calle, estaciona en planta alta, y sale por calle 1.
   `[T0, T3, T6, T9, T11, T13, T15, T17, T18, T19, T20]`
2. Entra por la primer calle, estaciona en planta alta, y sale por calle 2.
   `[T0, T3, T6, T9, T11, T13, T15, T17, T18, T19, T21]`
3. Entra por la primer calle, estaciona en planta baja, y sale por calle 1.
   `[T0, T3, T6, T10, T12, T14, T16, T17, T18, T19, T20]`
4. Entra por la primer calle, estaciona en planta baja, y sale por calle 2.
   `[T0, T3, T6, T10, T12, T14, T16, T17, T18, T19, T21]`
5. Entra por la segunda calle, estaciona en planta alta, y sale por calle 1.
   `[T1, T4, T7, T9, T11, T13, T15, T17, T18, T19, T20]`
6. Entra por la segunda calle, estaciona en planta alta, y sale por calle 2.
   `[T1, T4, T7, T9, T11, T13, T15, T17, T18, T19, T21]`
7. Entra por la segunda calle, estaciona en planta baja, y sale por calle 1.
   `[T1, T4, T7, T10, T12, T14, T16, T17, T18, T19, T20]`
8. Entra por la segunda calle, estaciona en planta baja, y sale por calle 2.
   `[T1, T4, T7, T10, T12, T14, T16, T17, T18, T19, T21]`
9. Entra por la tercer calle, estaciona en planta alta, y sale por calle 1.
   `[T2, T5, T8, T9, T11, T13, T15, T17, T18, T19, T20]`
10. Entra por la tercer calle, estaciona en planta alta, y sale por calle 2.
    `[T2, T5, T8, T9, T11, T13, T15, T17, T18, T19, T21]`
11. Entra por la tercer calle, estaciona en planta baja, y sale por calle 1.
    `[T2, T5, T8, T10, T12, T14, T16, T17, T18, T19, T20]`
12. Entra por la tercer calle, estaciona en planta baja, y sale por calle 2.
    `[T2, T5, T8, T10, T12, T14, T16, T17, T18, T19, T21]`
13. Encender y apagar el cartel.
    `[T22, T23]`

Se almacena en un arreglo la secuencia de disparos durante la ejecución, para luego hacer uso de las T-Invariantes y verificar que se cumplen los ciclos de disparos de la red.

En el caso ideal, si la ejecución finalizase en el mismo estado de transiciones sensibilizadas que al inicio de la ejecución (es decir, todos los ciclos de T-Invariante son completados), la intersección entre el conjunto total de disparos y cada vector de secuencia (completo) de T-Invariante arriba listado, resultaría en un remanente de 0 disparos, lo cual indicaría un perfecto funcionamiento de la red.

No obstante, como la ejecución tiene una cantidad de disparos predefinida, dicha intersección dejará remanentes de disparos que no han llegado a completar su ciclo, pero esto no implica una inconsistencia en la red o en el monitor, ya que al estar limitada en cantidad de disparos, dichos remanentes serán secuencias parciales de los invariantes.

2.6.3. Maquina de Estados de la Red



Este autómata finito parte de las T-Invariantes de la red descriptas en la *subsección* anterior, que añade mayor simplicidad a la hora de reconocer los ciclos que puede realizar la Red.

A grandes rasgos, pueden identificarse los 3 conjuntos de entradas al estacionamiento, ambas plantas, el cajero y las instancias de salida a la calle. Y además se visualiza el corto ciclo que recorre el autómata para encender el cartel.

Esta maquina de estados finito fue implementada utilizando una clase de tipo *Enum*, la cual tiene 23 enumeraciones (i.e. cantidad de estados), cada una con un método que recibe

el valor de la variable como parámetro, y devuelve al llamado el próximo estado (Enum) en caso de cumplir con la secuencia del autómata, de modo que evoluciona hasta eventualmente llegar al estado *EFin*.

Resulta una herramienta muy útil a la hora de verificar las Invariantes de Transiciones. Barriendo el registro con la secuencia de disparos que realizó la red, se identifica mas fácilmente aquellos ciclos que fueron completados como también aquellos ciclos parciales que no llegaron a finalizarse en la ejecución.

Implementacion en Java

Para no entorpecer el documento, se incluye una porción simplificada de código capaz de dar idea a la implementación general.

```
public enum MaqEstado {
E0{
    @Override
    public MaqEstado siguienteEstado(int variable) {
        switch(variable){
            case 0:
                registro.add(variable);
                return E11;
            case 1:
                registro.add(variable);
                return E12;
            case 2:
                registro.add(variable);
                return E13;
            case 22:
                registro.add(variable);
                return E4;
            default:
                return E0;
        }
    }
},
E11{
    @Override
    public MaqEstado siguienteEstado(int variable) {
        if (variable == 3){registro.add(variable); return E21;}
        else return E11;
    }
};

public abstract MaqEstado siguienteEstado(int variable);
static ArrayList<Integer> registro = new ArrayList<>();

/**
 * reinicia el registro de transiciones disparadas.
 */
static public void reiniciarRegistro(){
```

```
        registro.clear();
    }

    /**
     * getter del registro
     * @return ArrayList<Integer> registro
     */
    static public ArrayList<Integer> getRegistro(){
        return registro;
    }
}
```

Véase que en cada evolución del autómata, se añade el valor de la variable a un *ArrayList* llamado 'registro' para conocer la secuencia de la evolución.

Cada método *'siguienteEstado(int variable)'* devuelve el *Enum* próximo si corresponde, o el mismo *Enum* para permanecer en ese estado.

2.7. Diagrama de clases

INSERTAR DIAGRAMA DE CLASES ACA.

2.8. Diagramas de secuencia

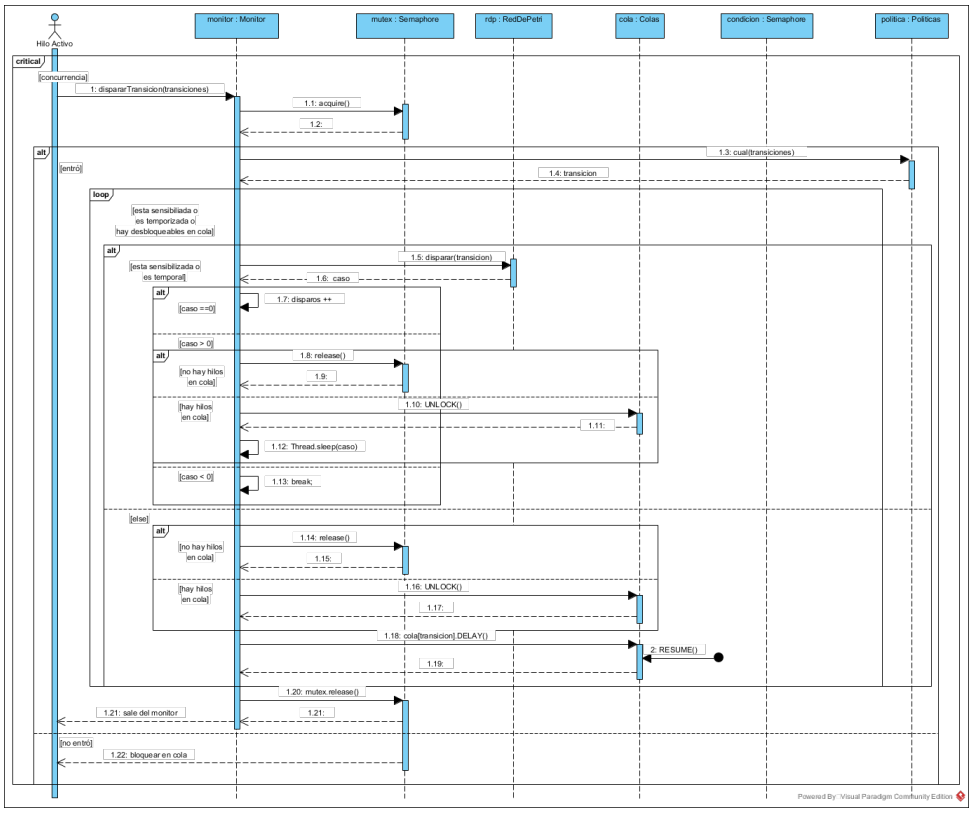


Figura 2: Diagrama del método 'disparar Transicion' de la clase 'Monitor'.

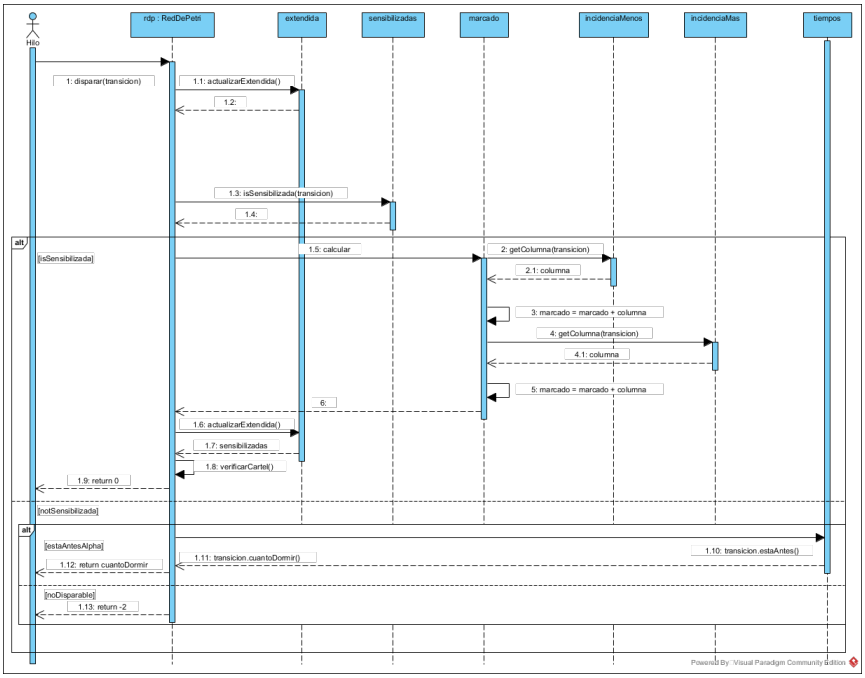


Figura 3: Diagrama del método 'disparar' de la clase 'RedDePetri'.

3. Conclusiones

Una vez realizados los *testings*, se pudo verificar que ambos invariantes se cumplen.

En cuanto a los hilos, la política influía en estos, por lo que los hilos se bloqueaban por culpa de ella.

En cuanto a los T-invariantes, se cumplen correctamente, quedando secuencias parciales de ellos una vez finalizada la ejecución.

Se agregaron dos invariantes de transiciones más que el PIPE no las establecía, siendo éstos los invariantes correspondientes a la tercer entrada. Se cree que tiene una limitación en su cantidad de muestras, con esto mejoró significativamente la verificación de los disparos realizados.

ALGUNA OTRA CONCLUSION MAS, DESARROLLAR MEJOR ESTO.