**Walkers of the city**

Programacion concurrente

**Integrantes:**

* Altman Quaranta Augusto.
* Fretes Ricardo.
* Diaz Marcos.

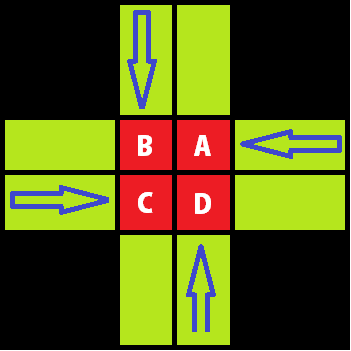
Córdoba, 23 de Noviembre del 2012

***Sistema***

**Introducción**

Se quiere representar una esquina (un cruce) transitada por autos que no deben chocar entra ellos. Los autos se comportan independientemente uno de otro, viajando a diferentes velocidades y realizando diferentes acciones sobre el cruce, como ser seguir derecho, doblar a la izquierda, derecha, etc.

Se plantea entonces un sistema como solución a lo planteado, en donde cada auto es un hilo de ejecución diferente, que viajan a velocidades aleatorias y ejecutan acciones aleatorias sobre el cruce. Este último además es un monitor que gestiona 4 recursos que componen la esquina, posibilitando la exclusión mutua entre los hilos y evitando los interbloqueos y la inanición.

Definimos el cruce como sigue:

En donde los cuadrados rojos A, B, C y D son coordenadas específicas en el cruce y son recursos que comparten todos los autos que deben ser gestionados por el monitor para obtener exclusión mutua. Se desea, para aumentar el nivel de paralelismo que los autos, vayan liberando recursos a medida que los utilizan y no los devuelvan todos juntos al final de la ejecución de la acción que estén llevando a cabo.

Los autos llegarán por cualquiera de las direcciones marcadas con flechas azules, y dependiendo de la misma van a entrar al cruce por un recurso diferente. Al entrar al cruce los autos pueden ejecutar una de las siguientes acciones:

* Seguir derecho
* Doblar a la derecha
* Doblar a la izquierda

A su vez dependiendo por donde entren los autos y que acción ejecuten van a tomar diferentes recursos formando diferentes secuencias de uso y liberación de los mismos.

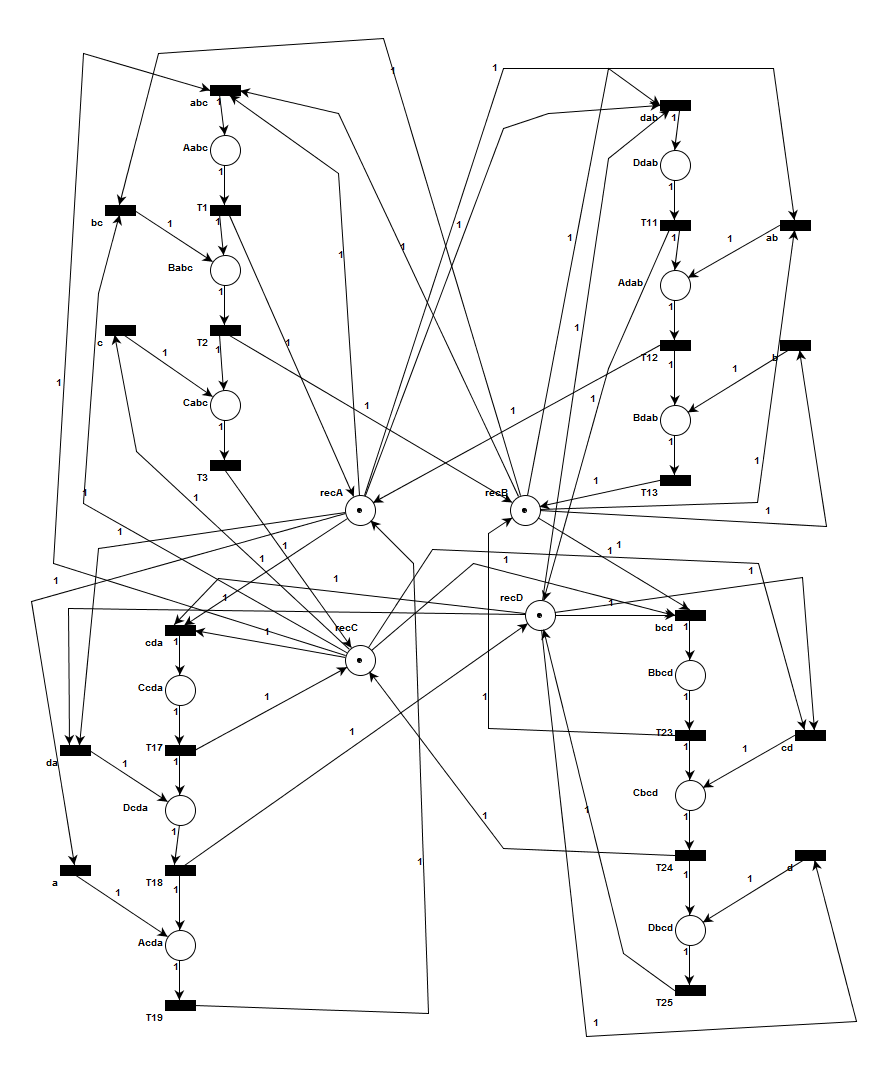
En la siguiente tabla se resumen las secuencias de uso y liberación de recursos según el recurso de entrada y la acción a ejecutar.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Doblar a la derecha | Seguir derecho | Doblar a la izquierda |
| A | A | A, B | A, B, C |
| B | B | B, C | B, C, D |
| C | C | C, D | C, D, A |
| D | D | D, A | D, A, B |

De la tabla anterior se observan que hay 12 secuencia posibles. El monitor entonces deberá tener 12 colas que contengan hilos que estén esperando por llevar a cabo alguna de las secuencias anteriores. Se le llamaran a estas colas “colas de acción”. Además poseerá una cola de entrada en donde estarán todos los autos esperando por entrar al monitor.

Para evitar interbloqueos el monitor deberá asignarle todos los recursos que el auto requiera al mismo tiempo antes de comenzar a ejecutar la acción y los mismos deberán ser liberados uno a uno a medida que se van utilizando, por esto último y como se requirió más arriba se aumenta el nivel de paralelismo del sistema. La inanición se evitara aplicando una política de gestión de recursos que se explicara más adelante.

**Modelo del sistema**

Red de Petri

Para modelar el sistema se confecciono la siguiente red de Petri.

Descripción de la red

Se puede ver en el centro 4 plazas que representan los 4 recursos del cruce bajo los nombres de “rec<recurso>”, pudiendo ser <recurso> = A, B, C o D.

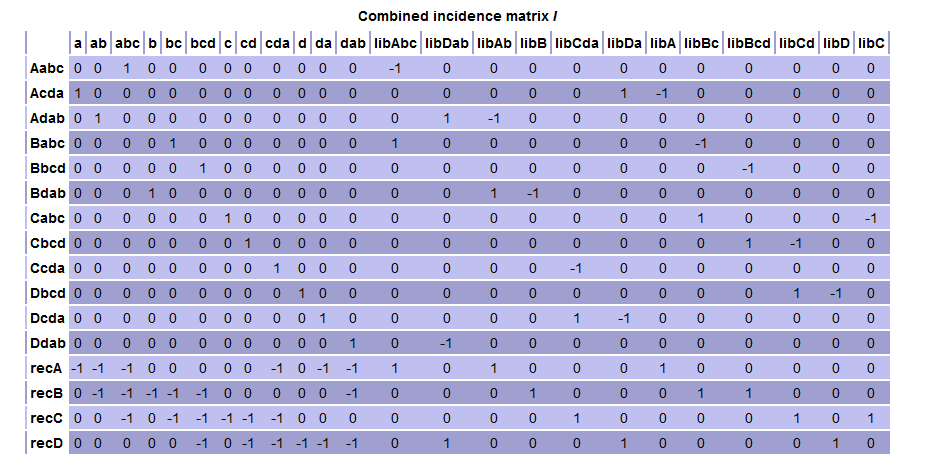
Además se observan otras 12 plazas alrededor de estas 4 plazas centrales, que se organizan en grupos de 3, formando 4 grupos a los que llamaremos “cadenas”. Cada plaza dentro de una cadena representa un recurso usado durante la misma. Mediante estas cadenas es posible representar las 12 secuencias posibles de uso y liberación de recursos que puede ejecutar un auto cualquiera. Esto se logra ingresando a las cadenas en diferentes puntos. De esta forma dada una secuencia determinada se entrara (mediante una transición) a la cadena que termine con esta secuencia desde el punto correspondiente. Por ejemplo para la secuencia BC se usara la cadena ABC y se ingresara por la transición que se conecta con la plaza babc. La notación utilizada en los nombres de las plazas que componen las cadenas es “<recurso><cadena>” pudiendo ser <recurso> = A, B, C o D y <cadena> = ABC, BCD, CDA, DAB. La parte de <recurso> representa el recurso que está siendo utilizado dentro de la <cadena> correspondiente.

Cada vez que se usa el recurso se libera y es por ello que se puede encontrar una transición conectada a estas plazas que colocan un token en la plaza del recurso correspondiente.

Las transiciones de entrada son las que ejecutan los autos (hilos) al llegar al cruce para realizar una acción dada desde un punto de ingreso al mismo. Por ende las transiciones de entrada representan una secuencia de uso y liberación. Como se puede observar estas solo se disparan si los recursos para ejecutar dicha secuencia están disponibles, proveyendo exclusión mutua entre autos. Además al disparar una transición de entrada se toman los tokens de todas las plazas de los recursos necesarios para llevar a cabo dicha secuencia. De esta manera se evitan los interbloqueos.

Datos de la red y utilización de los mismos

De la red anterior se obtuvo la siguiente matriz de incidencia.



La matriz de incidencia es una representación matricial de la red. Cada columna describe los tokens que cambian (pierde si es negativo o gana si es positivo) en cada plaza.

Esta matriz será el núcleo de nuestro monitor. De hecho todo el modelado con redes de Petri descrito anteriormente se realiza con el único objetivo de obtener esta matriz. Usando la teoría de las redes de Petri nuestro monitor será capaz de proveer todas las propiedades que se observaban en la red (exclusión mutua, no interbloqueos, liberar los recursos a medida que se van utilizando) con un simple calculo matemático, evitando un montón programación adicional para conseguir la lógica que provea esas propiedades.

Se multiplica la matriz de incidencia por un vector que representa el disparo de una transición y luego se suma el resultado de lo anterior al vector que representa el estado de la red de Petri (ese vector es la marcación de las plazas), obteniendo, si todos los valores del vector resultado son positivos, un nuevo estado (una nueva marcación). Si alguno de los valores del resultado es negativo entonces esa transición no se puede disparar en ese estado (estado no alcanzable).

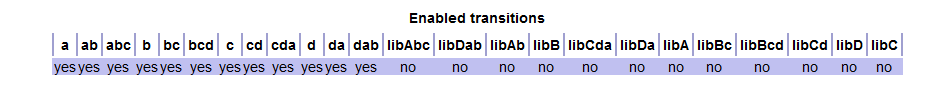
N = E + M x T

Siendo “N” nuevo estado, “E” estado actual, “M” matriz de incidencia y “T” vector de disparo.

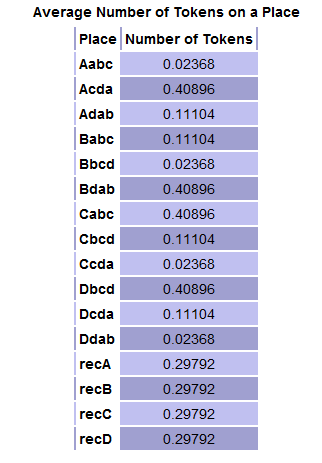
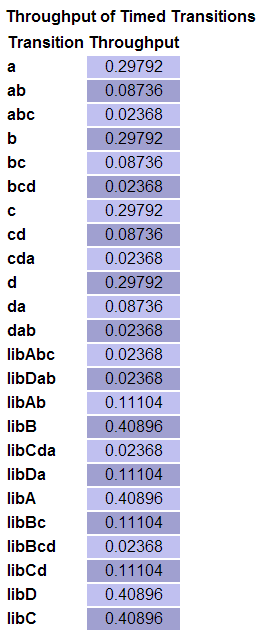
Análisis del modelo

De la red de Petri se extraen datos secundarios, tablas y estadísticas que nos permiten analizar el funcionamiento de la misma y verificar si es correcto.

A continuación se muestra una tabla con las transiciones están habilitadas al inicio:



Podemos ver que al inicio se permite el movimiento de todos los posibles caminos, esto sucede naturalmente al estar disponibles los cuatro recursos.

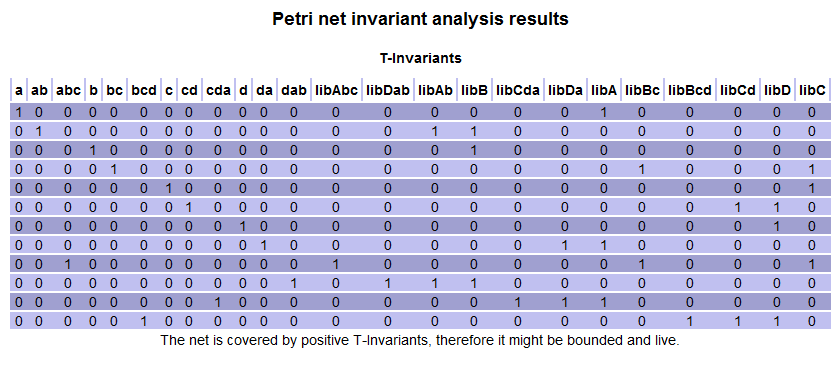
A partir de las siguientes tablas hacemos un análisis de las probabilidades de encontrar un token en una plaza o de que se ejecute una transición (asignando mismas probabilidades a todas las transiciones):

Por un lado, al no haber plazas que no tengan probabilidad de ser ocupadas, ni transiciones que no tengan probabilidad de ser ejecutadas, nos muestra que no hay interbloqueo, y que todas las plazas pueden llegar a ser tomadas de alguna forma.

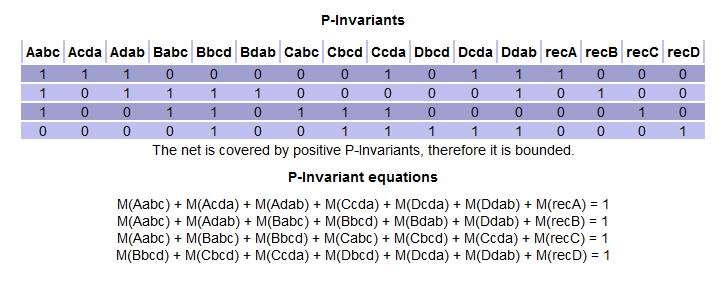
Además podemos ver tanto en las plazas como en las transiciones, que todos los autos que quieran tomar la misma cantidad de recursos, tienen la misma probabilidad de que puedan tomarlos, es decir que dadas dos secuencias de uso y liberación de recursos que requieran de la misma cantidad de recursos no hay preferencias de una sobre la otra. Por ejemplo, se puede ver que la probabilidad de que ocurra la transición a, b, c o d es la misma, de esta forma todos los autos que quieran tomar un solo recurso tienen la misma probabilidad de ejecutarse. Lo mismo sucede por ejemplo con ab, bc, cd y da. Esto nos muestra que la red es simétrica en cuanto a cada recurso.

También vemos que hay mucha más probabilidad de que se pueda ejecutar una transición que pida un recurso que una que pida dos o que pida tres. Esto tiene mucho sentido, ya que es más fácil que se encuentre un solo recurso disponible a que haya más. Esto lo vamos a tener que tener en cuenta a la hora de asignar prioridades en nuestro monitor. Una política adecuada podría ser asignarle más prioridad a las acciones que requieran de más recursos, de forma que en caso que haya recursos disponibles para ejecutar más de una transición se seleccione aquella que consuma más. Esta política tiende a disminuir la inanición que podría sufrir un auto que quiere ejecutar una acción larga (dada la menor probabilidad de ejecución de estas), pero no soluciona del todo el problema (puede haber inanición de todas formas).

Ahora haremos un análisis de los invariantes de nuestra red:



Esta tabla nos muestra cada conjunto de transiciones que al ejecutarse nos lleven a un mismo estado de marcación. Podemos ver que son 12 los t-invariantes lo que nos muestra los 12 posibles caminos que se pueden tomar en una esquina, es decir, el conjunto de todos los movimientos posibles de un auto en una esquina.



Esto nos muestra que de los miembros de cada ecuación de los p-invariantes solo uno puede tener el token en determinado momento, con esto podemos determinar que no se multiplican los tokens (garantizamos que el recurso sea tomado una sola vez), ni se destruyen (no desaparece el recurso); naturalmente tenemos una ecuación por cada posible recurso (A, B, C o D). Vemos entonces que se cumple la condición de exclusión mutua.

**Política del sistema**

En la planificación del monitor, a través de la red de Petri, hemos decido implementar el uso de una política para solventar una posible inanición y manejo de acumulación en las colas de acción.

Manejo de acumulación en colas

Por lo observado de la probabilidad de disparar determinadas transiciones, tenemos que la probabilidad disminuye a medida que necesitamos más recursos simultáneos para esa transición (una secuencia de uso y liberación más larga). Esto quiere decir que son más probables las que necesiten menos recursos, por ende se van a disparar con mayor frecuencia produciendo una acumulación menor en sus respectivas colas (se considera que la distribución de entrada en las colas de acción es aleatoria con distribución probabilística normal). Por otro lado la acumulación en las colas de acciones más largas va a ser mayor, puesto que al haber menos probabilidad de disparo de las transiciones que ejecutan dichas secuencias la frecuencia de disparo va a ser menor (los autos van a salir más lentos de esa cola).

Para contrarrestar esto se plantea la siguiente solución:

Decidimos que en frente al caso en donde se encuentren recursos para poder disparar acciones de diferentes cantidad de tokens (teniendo en cuenta solo las acciones mutuamente excluyentes entre ellas), se le dará prioridad a la acción que necesite más recursos. Estableciendo así, tres niveles de prioridades, uno cada tipo de acción: (acción de 1 token, 2 tokens, y 3 tokens).

|  |  |  |
| --- | --- | --- |
| Cantidad de tokens necesarios por acción | Acciones | Prioridad |
| 1 token | A, b, c, d | 1 |
| 2 tokens | Ab, bc, cd, da | 2 |
| 3 tokens | Abc, bcd, cda, dab | 3 |

**A mayor número de prioridad la prioridad es mayor.**

Para un mejor entendimiento se da el siguiente ejemplo:

Dado un momento determinado, un auto ingresa al monitor y devuelve el token A y el token B siendo que no hay ningún otro recurso disponible. En este caso se podrían disparar las acciones:

* A
* B
* AB

Frente a esta situación la acción con prioridad a ejecutarse será la acción AB, ya que posee una prioridad de 2, frente a la prioridad 1 de las otras dos acciones.

Inanición

La inanición se puede dar frente a la situación en la cual lleguen continuamente autos que deseen utilizar acciones cortas (que requieran poca cantidad de recursos) y nunca se pueda juntar la combinación de recursos necesarios para una acción larga (que necesitan mayor cantidad) habiendo autos en la cola de espera de esa acción. Esto ocurre ya que cuando está disponible un recurso, alguno de los otros que necesita no lo está y de esta forma no se consigue nunca la combinación necesaria. La política de prioridad presentada anteriormente no resuelve este problema ya que el uso de esa prioridad se da en el caso que se encuentre la posibilidad de ejecutar en ese momento acciones mutuamente excluyentes entre sí.

Para un mejor entendimiento se da el siguiente ejemplo:

Llegan continuamente autos para realizar la acción a, y acción b. Debido a esto se produce inanición para efectuar las acciones ab, abc ya que nunca consigo la combinación de estos recursos.

Para contrarrestar esto se plantea la siguiente solución:

Definimos que un hilo cae en una posible inanición cuando DESDE su ingreso al monitor, ya han entrado K cantidad de hilos luego de él, y todavía no ha conseguido los recursos que necesita para ejecutar la acción deseada.

Una vez que un hilo entra en la categoría de posible inanición, se producen las siguientes modificaciones al flujo normal del monitor:

Cuando un hilo ingresa con la intención de solicitar recursos, verifica si el hilo más viejo en estado de posible inanición requiere alguno de los recursos que esté está solicitando, si es así, no retira ningún recurso y se ubica en la cola de acción correspondiente.

Cuando un hilo ingresa para devolver recursos, verifica cuales son todas las acciones posibles con los recursos disponibles previamente junto lo que él devuelve (el estado del monitor luego de devolver el recurso), si hay inactivos en ellas, se despierta al más antiguo, si no hay, continua con el flujo normal.

**Diseño del sistema**