Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Redes de Petri para la Especificación Formal de Sistemas Concurrentes

Tesis profesional que para obtener el título de

Licenciado en Ciencias de la Computación

Presenta:

Pablo Arriaga Spezzia

Puebla, Pue., 1999

Contenido

Introducción

El desarrollo de sistemas ha tenido poco tiempo de llevarse a cabo, se han tenido experiencias de que los sistemas elaborados son hasta cierto punto confiables y la mayoría tiene fallas de distinta índole, esto se debe a que el sistema no es modelado formalmente.

La razón por la cual nosotros queremos modelar los sistemas, en particular los sistemas concurrentes, es precisamente por la necesidad de tener un análisis matemático que nos demuestre la confiabilidad de los sistemas. Existen modelos matemáticos que nos proporcionan fundamentos analíticos y una manera gráfica de visualizar su comportamiento. Una alternativa es utilizando grafos dirigidos (digrafos) para simular los procesos, pero poseen el problema de que no pueden modelar ciclos, ya que si los nodos del digrafo significan procesos y los arcos entre ellos significan precedencia en ejecución, el digrafo podría interpretarse como una secuencia de precedencias cicladas, que en la práctica no conduce a un análisis claro.

Para los sistemas concurrentes se requiere algo más que un digrafo para modelarlos, ya que la mayoría de estos sistemas contienen ciclos infinitos dentro de sus procesos (como los sistemas de producción, los sistemas operativos, etc.) y esto en un digrafo podría llegar a ser indeterminado, puesto que un ciclo en él podría significar un **deadlock** o simplemente un ciclo que requiere modelarse. Por tal motivo se introduce un modelo formal llamado **Red de Petri**, que nos permite modelar ciclos y representar mediante disparos de transiciones la precedencia entre procesos, de esta forma se puede verificar que todos los procesos y estados sean alcanzables y verificar la consistencia de los ciclos.

Por ejemplo, en la Figura 0.1, se muestra un digrafo [NuG89] con siete nodos con arcos que denotan la precedencia entre los procesos $p_{i,j}$, en el que se observa que si se requiere conectar el nodo $p_{1,4}$ al nodo $p_{1,1}$ con un arco hacia arriba, no parecería ser claro porque no se puede visualizar de manera simple a qué se refiere ese arco, ya sea que se necesite modelar un ciclo o que esté provocando un **deadlock**, puesto que no se puede iniciar el proceso $p_{1,1}$ hasta que no haya finalizado el proceso $p_{1,4}$ aunque después de una segunda iteración si podría.

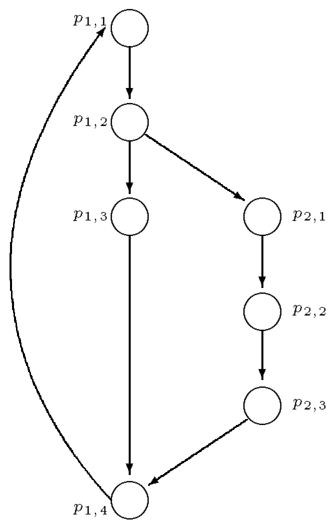

Figura 0.1: Modelos Gráficos, Grafo Dirigido

En la figura 0.2, se encuentra una Red de Petri que modela el mismo sistema en la que se puede ver más claramente su comportamiento y el ciclo mencionado ahora parece más natural, puesto que la precedencia entre estos procesos ahora está controlada por las transiciones.

El método formal de Redes de Petri permite modelar sistemas concurrentes en una forma sencilla gracias a su naturaleza gráfica, además que puede verificar la confiabilidad de este tipo de sistemas.

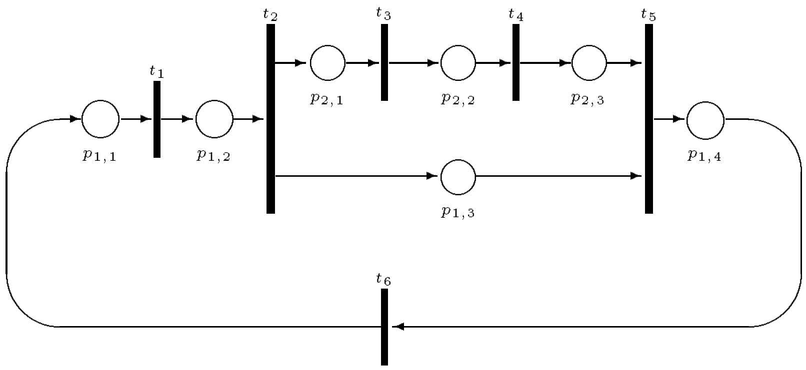

Figura 0.2: Modelos Gráficos, Red de Petri

Capítulo 1

Conceptos Básicos

Formalmente las Redes de Petri están definidas sobre bases matemáticas que permiten conocer y analizar su comportamiento. En este capítulo se darán a conocer definiciones formales de las Redes de Petri y algunas de sus extensiones más comunes, abarcando hasta las Redes de Petri Coloreadas con Intervalos de Tiempo.

1.1 Concepto de Red de Petri

La **Teoría de Redes de Petri** está enfocada para describir sistemas concurrentes en forma **gráfica** y **matemática**. Las Redes de Petri han sido usadas ampliamente para modelar y analizar sistemas concurrentes así como sistemas de tiempo real, no determinismo, comunicación y sincronización. La base de esta teoría es producto del trabajo realizado por **Carl Adam Petri** a principios de los 1960s.

Las Redes de Petri son un modelo independiente para estudiar la precedencia en sistemas operativos, programación paralela y arquitectura de computadoras [NuG89], las cuales proporcionan un panorama útil y alternativo de un sistema o cálculo. Existen muchos factores que contribuyen al éxito de las Redes de Petri: su naturaleza gráfica, la habilidad de modelar procesos paralelos y distribuidos de forma natural, la simplicidad del modelo y sus firmes fundamentos matemáticos.

Las Redes de Petri están basadas en una simple extensión de conceptos de estado y transición conocidos de los autómatas [EiU96]. En las Redes de Petri los estados y las transiciones son distribuidos sobre varios lugares, esto permite un distinción explícita entre concurrencia y secuencialidad.

Una Red de Petri simple es un digrafo con nodos que pueden ser lugares o transiciones los cuales se interconectan con arcos, es decir, una Red de Petri es un grafo dirigido compuesto de conjunto de nodos formados por la unión de lugares con transiciones. La representación de ciclos es permitida dentro de la Red de Petri. Los lugares son componentes pasivos, mientras que las transiciones son componentes activos, esto es dado que las transiciones pueden causar cambios de estado en la Red de Petri.

Como en el álgebra de procesos, el área principal de investigación consiste en la extensión de la teoría con nuevas propiedades que amplían la expresividad. Podemos ver que las Redes de Petri describiendo sistemas reales tienden a ser complejas y extremadamemte largas. A veces es casi imposible modelar el comportamiento del sistema minuciosamente. Para solucionar estos problemas, muchos autores proponen extensiones al modelo básico de las Redes de Petri. A todas estas extensiones juntas se les conoce como **Redes de Petri de Alto Nivel** [KuT+94]. Uno de los modelos más completos de Redes de Petri de alto nivel es el modelo de Redes de Petri Coloreadas con Intervalos de Tiempo, sobre el cual se basa esta tesis.

1.2 Redes de Petri Blanco y Negro

Existen muchas variedades de Redes de Petri, las Redes de Petri blanco y negro son conceptualmente simples y sencillas de analizar. En las Redes de Petri las **transiciones** son representadas por rectángulos o líneas y los **lugares** por círculos, los lugares y transiciones se encuentran interconectados por **arcos**. Los arcos pueden conectarse de lugares a transiciones (conocidos como arcos de entrada y los lugares correspondientes conocidos como de entrada) o de transiciones a lugares (conocidos como arcos de salida y los lugares correspondientes conocidos como de salida) [KhA91].

Un **token** se representa por un punto dentro de un lugar, la distribución de tokens en cada uno de los lugares de entrada a una transición se le denomina **marca**, esto significa que un recurso que se necesita para activar un proceso se hace disponible cuando tiene una marca. Así una transición puede ser habilitada si y sólo si tiene al menos un token en cada lugar de entrada de la transición. Una transición se puede **disparar** si todos los lugares de sus entradas están marcados, de esta forma un token es consumido (eliminado) de cada una de sus entradas y se producirán tokens en cada una de las salidas.

La distribución de tokens en la Red de Petri representa el estado de los procesos individuales que realizan el cálculo. Un conjunto de tokens que fluyen a través de la Red de Petri representa la ejecución del código correspondiente. Una Red de Petri es ejecutada estableciendo una **marca inicial**, entonces en cada ciclo subsecuente se toma un conjunto de transiciones elegibles para disparo [KhA91].

Notemos que la posibilidad para disparar una transición se determina solamente por condiciones locales, indicando la presencia de tokens en las entradas adyacentes. Esta característica de localidad es una cualidad deseable en el modelado de sistemas concurrentes.

Una Red de Petri blanco y negro es un sistema Lugar/Transición que se define formalmente de la siguiente manera [KuT+94]:

Definición 1.1 *Un sistema Lugar/Transición se define como un séxtuple* $RP = (P, T, N, A, K, M_0)$ *si y sólo si:*

- *(a)* P *es el conjunto de* ***lugares****,*
- *(b)* T *es el conjunto de* ***transiciones***
- *(c)* N *es el* ***flujo en la red*** *entre parejas de nodos de la forma* $N \subseteq (P \times T) \cup (T \times P)$*.*
- *(d)* $A : N \to I\!N^+$ *es la función de* ***peso****.*
- *(e)* $K : P \to I\!N^+ \cup \{\infty\}$ *es la función de* ***capacidad****.*
- *(f)* $M_0 : P \to I\!N$ *es una función de* ***marca inicial*** *que satisface que* $M_0(p) \leq K(p)$ *para todo* $p \in P$*.*

En (a) el conjunto P de lugares contiene todos los lugares en los que la Red de Petri puede tener un estado definido con tokens. El inciso (b) define el conjunto de transiciones T que contiene todas las transiciones que interconectan los lugares entre los cuales se realizan los disparos.

El flujo en la red N en (c) contiene parejas de nodos tales que el primer elemento es el nodo origen y el segundo es el nodo destino, esto indica que se pueden tener aristas dirigidas de un lugar a una transición o de una transición a un lugar, lo que significa entrar y salir de una transición. Estos tres primeros elementos conforman un sistema de red elemental.

La función de peso A en (d), nos proporciona información sobre la cantidad de aristas que interconectan un nodo origen con un nodo destino. El número de aristas entre un lugar y una transición indica la cantidad de tokens que serán consumidos al disparar la transición y el número de aristas entre transiciones y lugares indica la cantidad de tokens que se producirán con el disparo de la transición. De esta forma se puede controlar la cantidad de tokens que fluyen a través de la red.

En (e) la función de capacidad K mapea los lugares en el conjunto de los naturales, esto indica la cantidad de tokens que se pueden tener en un lugar determinado. Así se puede determinar el número de tokens que requiere un lugar para que pueda habilitar una transición.

La función M_0 de marca inicial definida en (f) nos indica el estado en el que la Red de Petri inicia, esto es el número de tokens con el que la red puede empezar. Una marca inicial determina el estado en el que la red comienza a 'ejecutarse'. El número de tokens iniciales debe ser menor o igual a los tokens que pueda tener el lugar en un instante determinado, ya que son los tokens mínimos con los que la Red de Petri puede empezar a trabajar. Esta información es útil cuando nos interesan las propiedades dinámicas de la Red de Petri, con estas propiedades podemos ver el comportamiento de la red.

Hay que notar que los lugares que posean al menos un token, indican que el recurso que ese lugar representa se hace disponible, esto es, se habilita su uso. Un recurso puede representar cualquier tipo de entidad, ya sea la memoria, el disco, algún proceso que se desee ejecutar, y aún más, un recurso puede representar personal, maquinaria, trabajo, dinero, etcétera.

1.2.1 Ejemplo de Aplicación

Para ver de una forma clara la aplicación de las Redes de Petri para modelar sistemas concurrentes, nos fijaremos en un problema en particular y veremos su representación con una Red de Petri. Dijkstra planteó y resolvió un problema de sincronización llamado el **problema de la cena de los filósofos**, el cual se enuncia de la siguiente manera [TaA91]:

"Cinco filósofos se sientan a la mesa. Cada uno de ellos tiene un plato de spaghetti. El spaghetti es tan escurridizo que un filósofo necesita dos tenedores para comerlo. Entre cada dos platos hay un tenedor. La vida de un filósofo consta de dos períodos alternados, comer y pensar".

Este problema plantea una situación concurrente en que podemos ver a un filósofo como un proceso y a los tenedores como los recursos que permiten o no a un filósofo pasar del estado de pensar al estado de comer. Cuando un filósofo tiene hambre, intenta obtener un tenedor para su mano izquierda y otro para su mano derecha, alzando uno a la vez y en cualquier orden. Si logra obtener los tenedores, come por un momento y después deja los tenedores para continuar pensando.

Los filósofos actúan aleatoriamente, así no se puede determinar cuales de ellos están comiendo y cuales pensando, se necesita llevar un registro para determinar la actividad de cada filósofo –es decir en que estado se encuentra– y usar semáforos para determinar cuando un recurso está disponible. El código en C para modelar el comportamiento de los filósofos se muestra en la figura 1.1 [TaA91].

El código que se muestra en la figura anterior es correcto y permite el máximo paralelismo para los filósofos. Utiliza un arreglo *estado* para llevar un registro de la actividad de cada filósofo: si está comiendo, pensando, o hambriento. Un filósofo puede comer si los vecinos izquierdo y derecho no lo están haciendo. Los vecinos del $i-$ésimo filósofo se definen con *IZQUIERDO* y *DERECHO*.

Esta solución utiliza un arreglo de semáforos, uno por cada tenedor, de manera que los filósofos hambrientos pueden detenerse mientras los tenedores (recursos) necesarios están ocupados. Antes de empezar a tomar los tenedores, un filósofo hace un *DOWN* en *mutex* y al volver a colocar los

```
#define N          5       //filosofos
#define IZQUIERDO  (i-1)%N  //vecino izquierdo
#define DERECHO    (i+1)%N  //vecino derecho
#define PENSANDO   0        //estado pensando
#define HAMBRIENTO 1        //estado tratando de tomar los tenedores
#define COMIENDO   2        //estado comiendo

typedef int semaforo        //se definen los semaforos como enteros

int estado[N];              //se llevan los estados independientemente
semaforo mutex = 1;         //para la exclusion mutua
semaforo s[N];              //semaforo para cada tenedor

void filosofo(int i)        //i es el numero de filosofo
{
   while(TRUE)
   {
      piensa();            //filosofo pensando
      toma_tenedores(i);   //toma dos tenedores o se bloquea
      come();              //el filosofo come
      coloca_tenedores(i); //coloca ambos tenedores de regreso
   }
}
```

Figura 1.1: Código en C para un filósofo

tenedores hace un *UP* en *mutex* para realizar una exclusión mutua en regiones críticas.

Se puede observar que cada proceso ejecuta el procedimiento *filosofo* como código principal, aunque los demás procedimientos (ver figura 1.2), *toma_tenedores*, *coloca_tenedores* y *prueba* son procedimientos ordinarios y no procesos separados.

Como se puede ver en el código presentado, no se visualiza el comportamiento de cada uno de los filósofos, ya que internamente los procesos hacen uso de estados y semáforos complicando su ejecución y no es posible analizar gráficamente la solución a este problema. Una manera de visualizar este problema para razonar el comportamiento de cada uno de los filósofos en un momento determinado, se hace utilizando una Red de Petri Blanco y Negro.

```
void toma_tenedores(int i) // i es el numero de filosofo
{
   down (&mutex);           //entra a la region critica
   estado[i] = HAMBRIENTO;  //el filosofo i tiene hambre
   prueba (i);              //intenta tomar dos tenedores
   up (&mutex);             //sale de la region critica
   down (&s[i]);            //se detiene si no consigue los tenedores
}

void coloca_tenedores(int i)// i es el numero de filosofo
{
   down (&mutex);           //entra a la region critica
   estado[i] = PENSANDO;    //el filosofo i termino de comer
   prueba (IZQUIERDO);      //ve si el vecino izquierdo puede comer
   prueba (DERECHO);        //ve si el vecino derecho puede comer
   up (&mutex);             //sale de la region critica
}

void prueba(int i)          // i es el numero de filosofo
{
   if (estado [i]         == HAMBRIENTO &&
       estado [IZQUIERDO] != COMIENDO   &&
       estado [DERECHO]   != COMIENDO)
   {
       estado [i] = COMIENDO;
       up (&s[i]);
   }
}
```

Figura 1.2: Código complementario para la solución del problema de la cena de los filósofos

En la Red de Petri de la figura 1.3 se muestran en la parte superior los tenedores, las dos transiciones expresan la posibilidad de que un filósofo empiece a comer o empiece a pensar y en la parte inferior se encuentran dos lugares indicando cuando un filósofo se encuentra comiendo o pensando [KhA91]. En el recuadro de la figura se puede ver que para que el filósofo 1 empiece a comer, requiere de la presencia de dos tokens, uno en el lugar *Tenedor 1* y otro en el lugar *Tenedor 2*, además se requiere que el filósofo actualmente se encuentre en el lugar *F. Pensando 1*. Si estos tokens están disponibles, la transición *Empieza a Comer* se puede disparar consumiendo los tokens de sus entradas.

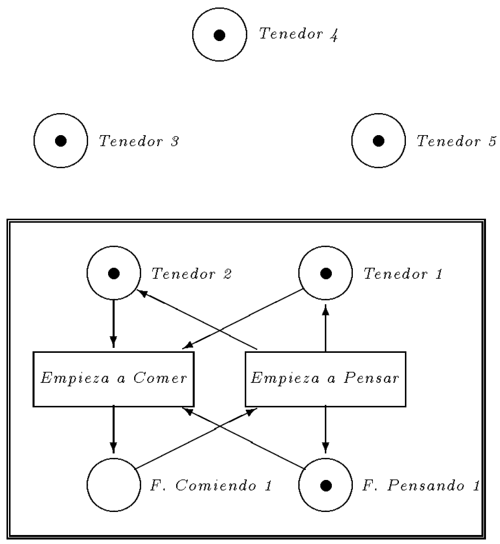

Figura 1.3: El problema de la cena de los filósofos modelado con una Red de Petri: El filósofo 1 puede *Empezar a Comer*

En la contraparte (figura 1.4), para que el filósofo pueda pensar, necesita que exista un token en el lugar *F. Comiendo 1*. Una vez que la transición *Empieza a Pensar* se dispare, se producirán otra vez los tokens en los lugares de los tenedores 1 y 2 indicando que se colocaron los tenedores de regreso y el filósofo pasará al lugar *F. Pensando 1*.

Según la definición (1.1), el segmento de Red de Petri mostrado en el recuadro de la figura 1.3 se puede denotar formalmente como sigue:

$RP = (P, T, N, A, K, M_0)$ donde:
$P = \{Tenedor1, Tenedor2, F.Comiendo1, F.Pensando1\}$,
$T = \{Empieza\ a\ Comer, Empieza\ a\ Pensar\}$,
$N = \{(Tenedor2, Empieza\ a\ Comer), (Tenedor1, Empieza\ a\ Comer),$
$(F.Pensando1, Empieza\ a\ Comer), (Empieza\ a\ Comer, F.Comiendo1),$
$(F.Comiendo1, Empieza\ a\ Pensar), (Empieza\ a\ Pensar, Tenedor2),$
$(Empieza\ a\ Pensar, Tenedor1), (Empieza\ a\ Pensar, F.Pensando1)\}$,
$\forall(x_1, x_2) \in N$: $A(x_1, x_2) = 1$,
$\forall p \in P$: $K(p) = 1$,
$M_0(Tenedor2) = 1$, $M_0(Tenedor1) = 1$, $M_0(F.Pensando1) = 1$

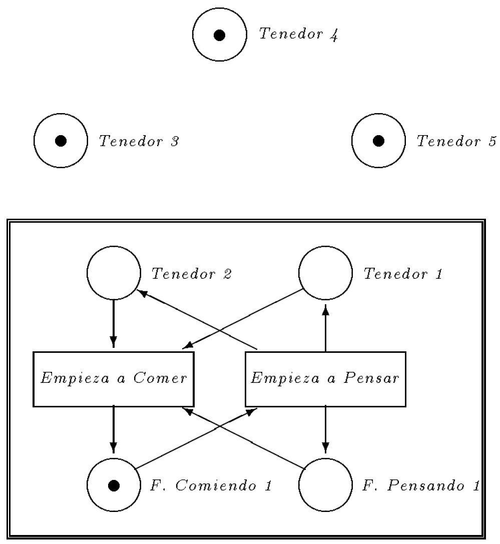

Figura 1.4: El problema de la cena de los filósofos: La transición *Empieza a Comer* se disparó

En esta Red de Petri, un filósofo sólo puede tomar los tenedores si ambos están disponibles, por lo tanto un deadlock no puede ocurrir a no ser que la red sea redibujada para modelar el alcance de un tenedor a la vez. De esta forma, este fragmento de Red de Petri visualiza el comportamiento de un filósofo y se puede entender más fácilmente cual es la actividad de un filósofo, pero aún existe el problema de que este fragmento se tiene que duplicar para cada filósofo y la Red crecería y se haría más compleja.

1.3 Redes de Petri Coloreadas

Las Redes de Petri blanco y negro son muy útiles pero limitadas en su poder de modelado. Una extensión común es conocida como **Redes de Petri Coloreadas** donde los tokens son diferenciados con colores que pueden tener valores de datos arbitrarios. El disparo de las transiciones es dependiente no sólo de la disponibilidad de un token en el lugar de entrada, sino de la existencia de un token coloreado apropiado. Similarmente la salida de una transición no es sólo un token sino específicamente un token coloreado. La notación de las Redes de Petri coloreadas es mucho más concisa que las Redes de Petri blanco y negro y por eso evita la duplicidad que es típica de estas últimas.

En una Red de Petri Coloreada se incluyen valores a los tokens que se encuentran en los lugares de la red, a estos valores se les llama colores, incluso las transiciones pueden contener condiciones bajo las cuales puedan dispararse. Esto se hace para lograr una mayor comprensión de la red y a la vez permite hacer un tipo de 'factorización' (en algunos casos) para resumir ciertos comportamientos similares.

Antes de definir formalmente las Redes de Petri Coloreadas, primero se introducirá la definición de multi-conjuntos, la terminología y notación que serán usadas más adelante.

1.3.1 Multi-conjuntos

Un **Multi-conjunto** es una colección de elementos sobre el mismo subconjunto de algún universo [vdA95]. Como un multi-conjunto puede tener múltiples ocurrencias de un elemento, a los multi-conjuntos se les denomina **bolsa**.

Definición 1.2 (Multi-conjunto) *Un Multi-conjunto m sobre el conjunto no vacío y finito A, es una función que mapea A en el conjunto de los números naturales. Si $a \in A$ entonces $m(a)$ es el número de ocurrencias de a en el multi-conjunto m. A_{MC} es el conjunto de todos los multi-conjuntos sobre A. El multi-conjunto vacío es denotado por $\emptyset_A$ (o por $\emptyset$). Un multi-conjunto $m \in A_{MC}$ es representado por la suma formal [vdA91]:*

$$\sum_{a \in A} m(a)a$$

Como ejemplo podemos considerar al conjunto $A = \{a, b, c, \ldots\}$, entonces los multi-conjuntos $3a, a+b+c+d$, $1a+2b+3c$ y $\emptyset_A$ son miembros del multi-conjunto A_{MC}. Como se puede ver usualmente omitimos los valores con coeficiente cero y también se omiten los coeficientes que sean iguales a uno.

Definición 1.3 *Muchos de los operadores de conjuntos pueden ser extendidos a multi-conjuntos de una manera más clara. Suponiendo A un conjunto, $m_1, m_2 \in A_{MC}$ y $q \in A$, se definen las siguientes operaciones entre multi-conjuntos:*

membresía:	$q \in b_1$ *ssi* $a_1(q) \geq 1$
inclusión:	$m_1 \leq m_2$ *ssi* $\forall_{a \in A} m_1(a) \leq m_2(a)$
igualdad:	$m_1 = m_2$ *ssi* $m_1 \leq m_2$ *y* $m_2 \leq m_1$
adición:	$m_1 + m_2 = \sum_{a \in A}(m_1(a) + m_2(a))a$
substracción:	$m_1 - m_2 = \begin{cases} \sum_{a \in A}(m_1(a) - m_2(a))a & \textit{si } m_2 \leq m_1 \\ 0 & \textit{en otro caso} \end{cases}$
cardinalidad:	$\# m_1 = \sum_{a \in A} m_1(a)$

Para las Redes de Petri Coloreadas ordinariamente se usan los términos **variables** y **expresiones**, en donde las variables son **atadas** a valores en lugar de ser asignadas. Conforme a [Jen90] para dar la definición de Redes de Petri Coloreadas es necesario introducir los siguientes conceptos:

- Los **tipos** son expresiones que denotan conjuntos en algún universo $\mathcal{U}$. Para un tipo $\mathcal{T}$ se usa este mismo nombre de tipo para denotar al **conjunto de elementos** en $\mathcal{T}$.
- El **tipo de una variable** v es denotado por $Tipo(v)$.
- El **tipo de una expresión** $expr$ es denotado por $Tipo(expr)$.
- El **conjunto de variables en una expresión** $expr$ es denotado por $Var(expr)$.
- Una **atadura de un conjunto de variables** V le asocia a cada variable $v \in V$ un elemento $b(v) \in Tipo(v)$.

- El **valor obtenido de evaluar una expresión** v **en una atadura** b es denotado por $expr\langle b\rangle$. $Var(expr)$ es requerido para ser un subconjunto de las variables de b, y la evaluación se hace sustituyendo para cada variable $v \in Var(expr)$ el valor $b(v) \in Tipo(v)$ determinado por la atadura.

Como un ejemplo para ilustrar esta notación, se puede tener:
Las variables x y y son del tipo S se denota por:
$Tipo(x) = Tipo(y) = S$
Las variables de la expresión $2 * (x + 3y)$ son x y y se denota como:
$Var(2 * (x + 3y)) = \{x, y\}$
El tipo de la expresión $2 * (x + 3y)$ es un multi-conjunto de S se denota:
$Tipo(2 * (x + 3y)) = S_{MC}$
Evaluar la expresión $2 * (x + 3y)$ con la atadura $\langle x = b, y = d\rangle$ produce el valor $2b + 6d$ se denota como:
$(2 * (x + 3y))\langle x = b, y = d\rangle = 2b + 6d$

Una expresión sin variables se dice que es una expresión **cerrada**. Se puede evaluar en todas las ataduras y todas las evaluaciones dan el mismo valor –que con frecuencia se denota por la misma expresión–. Esto significa que simplemente se escribe '$expr$' en lugar de '$expr\langle b\rangle$'.

Definición 1.4 *Una Red de Petri Coloreada [Jen94] se define formalmente como una tupla de la forma* $RPC = (\Sigma, P, T, A, N, C, G, E, I)$*, donde :*

(a) Σ *es un conjunto finito de* ***tipos*** *no vacíos, también llamados conjuntos de colores.*

(b) P *es un conjunto finito de* ***lugares****.*

(c) T *es un conjunto finito de* ***transiciones****.*

(d) A *es un conjunto finito de* ***arcos*** *tal que:*
$P \cap T = P \cap A = T \cap A = \emptyset$.

(e) N *es una función de* ***nodo****, definida como* $N : A \to (P \times T) \cup (T \times P)$.

(f) C *es una función de* ***coloreo****, definida como* $C : P \to \Sigma$.

(g) *G es una función de* ***guardia****. Está definida de T hacia expresiones booleanas tal que:*
$\forall t \in T : [Tipo(G(t)) = \mathbb{B} \wedge Tipo(Var(G(t))) \subseteq \Sigma]$. [1]

(h) *E es una función de* ***expresión de arco****. Está definida de A hacia expresiones tal que:*
$\forall a \in A : [Tipo(E(a)) = C(p)_{MC} \wedge Tipo(Var(E(a))) \subseteq \Sigma]$ [2]
donde p es el lugar de $N(a)$.

(i) *I es una función de* ***iniciación****. Está definida de P hacia expresiones cerradas tal que:*
$\forall p \in P : [Tipo(I(p)) = C(p)_{MC}]$.

El conjunto de **tipos** definido en (a) determina los valores de datos, operaciones y funciones que pueden ser usadas en las expresiones de la red (i.e. expresiones de arcos, guardias y expresiones de iniciación). Se asume que cada tipo tiene al menos un elemento.

Los incisos (b), (c) y (d) describen conjuntos finitos disjuntos de **lugares**, **transiciones** y **arcos**, como se definieron en la Red de Petri Blanco y negro. Requiriendo que los conjuntos de lugares, transiciones y arcos sean finitos, se evitan muchos problemas técnicos tales como la posibilidad de tener un número infinito de arcos entre dos nodos.

La función de **nodos** en (e) mapea cada arco en un par (x_1, x_2) donde x_1 es el nodo origen y x_2 es el nodo destino. Los dos nodos tienen que pertenecer a conjuntos distintos (i.e. si el primero es un lugar el segundo debe ser una transición y viceversa). En una Red de Petri Coloreada se permite tener varios arcos entre un par ordenado de nodos (por lo que se define a A como un conjunto separado y no como un subconjunto de $(P \times T) \cup (T \times P)$). Los arcos múltiples proporcionan comodidad para fines de modelado, en la teoría no agregan ni cambian propiedades.

(f) define la función de **coloreo** que mapea cada lugar p, a un tipo $C(p)$. Intuitivamente, esto significa que cada token en p debe tener un valor de datos que pertenezca a $C(p)$, es decir que corresponda a un tipo de Σ.

[1] El conjunto $\mathbb{B}$ denota al conjunto Booleano, $\mathbb{B} = \{Falso, Verdadero\}$
[2] El subíndice MC indica multi-conjunto.

La función de **guardia** (g) mapea cada transición t, en una expresión booleana donde todas las variables tienen tipos que pertenecen a Σ. La función guardia nos permite insertar precondiciones para las transiciones, las cuales pueden indicar los requisitos bajo los cuales una transición pueda ser disparada. Si se omiten las expresiones de guardia en el diseño de una Red de Petri, estas siempre se evaluarán a $Verdadero$.

(h) La función de **expresión de arco** E mapea cada arco a en una expresión del tipo $C(p)_{MC}$. Esto significa que cada expresión de arco debe evaluarse a *multi-conjuntos* sobre el tipo del lugar adyascente p, esto es el nodo lugar de la pareja de nodos $N(a)$. La evaluación de una expresión de arco indica el número de tokens coloreados que serán consumidos o producidos en un lugar p.

En (i) se define la función de iniciación I que mapea cada lugar p en expresiones cerradas que deben ser del tipo $C(p)_{MC}$. Con la evaluación de las expresiones de iniciación se obtiene la marca inicial, como se tiene de forma similar en las Redes de Petri blanco y negro. Cuando en el diseño de una Red de Petri Coloreada se omite la expresión de iniciación, esta se evalúa a $\emptyset$.

Nótese que la función de capacidad K, utilizada en la definición (1.1) no se emplea en esta definición, puesto que la evaluación de la expresión de arco proporciona mayor información utilizando tokens coloreados, es decir, ahora no interesa conocer 'cuántos' tokens puede haber en un lugar determinado –como en las Redes de Petri blanco y negro–, mas bien interesa conocer específicamente 'cuales' son los tokens que puede contener el lugar.

Se puede observar que para modelar una Red de Petri blanco y negro con una Red de Petri Coloreada, se requiere un sólo conjunto de tipos en Σ con dos elementos ($negro$ y $blanco$) indicando la presencia o ausencia de un token. De esta manera la función de coloreo C sólo proporcionará valores $negro$ o $blanco$ para cada uno de los lugares en la red. La función de guardia G para cada transición se deberá omitir para que se evalúe a *Verdadero*. A cada arco se le coloca una expresión de arco que entregue un token con valor $negro$ cada vez que se requiera un token en un lugar adyascente. Finalmente, para obtener la marca inicial, se evalúa la función de iniciación para obtener un color $negro$ sólo en los lugares donde se requiera un token inicial.

1.3.2 Comportamiento Dinámico

Para hablar del comportamiento de las Redes de Petri Coloreadas , hay que definir primero la siguiente notación [Jen94] para toda $t \in T$ y para todo par de nodos $(x_1, x_2) \in (P \times T) \cup (T \times P)$:

- $A_r(t) = \{a \in A | N(a) \in (P \times \{t\}) \cup (\{t\} \times P)\}$. Mapea cada transición t en el conjunto de arcos que la rodean.
- $Var(t) = \{v | v \in Var(G(t)) \vee \exists a \in A_r(t) : v \in Var(E(a))\}$. Se le llama el conjunto de variables de t.
- $A_c(x_1, x_2) = \{a \in A | N(a) = (x_1, x_2)\}$. Mapea cada par de nodos (x_1, x_2) en el conjunto de arcos conectados entre el nodo origen y el nodo destino.
- $E(x_1, x_2) = \sum_{a \in A_c(x_1, x_2)} E(a)$. Se le llama la expresión de la pareja de nodos (x_1, x_2).

La suma indica adición de expresiones (y está bien definida pues todas sus expresiones tienen un tipo de multi-conjunto en común). De los argumentos de E estará claro cuando nos referimos a la función $E : A \rightarrow Expr$ o a la función $E : (P \times T) \cup (T \times P) \rightarrow Expr$.

Después se definen las ataduras. Intuitivamente, la atadura de una transición t, es una sustitución que reemplaza cada variable de t con un color. Se exige que cada color es del tipo correcto y que la función guardia se evalúe a verdadero [Jen90]. Debe notarse que (b) implica que todas las ataduras satisfacen la función Guardia correspondiente. Como se define abajo, $G(t)\langle b \rangle$ denota la evaluación de la expresión guardia $G(t)$ en la atadura b:

Definición 1.5 *Una* ***atadura*** *de una transición t es una función b definida en $Var(t)$, tal que:*

(a) $\forall v \in Var(t) : b(v) \in Tipo(v)$.

(b) $G(t)\langle b \rangle$.

$B(t)$ denota el conjunto de todas las ataduras para t.

Con frecuencia se escriben las ataduras en la forma $\langle v_1 = c_1, v_2 = c_2, \ldots, v_n = c_n \rangle$ donde $Var(t) = \{v_1, v_2, \ldots, v_n\}$. El orden de las variables no tiene importancia. Para cada atadura se puede verificar cuando la transición –con esta atadura– está habilitada en la marca actual; esto se hace evaluando la función guardia y todas las expresiones en los arcos de entrada de la transición.

Definición 1.6 *Un **elemento token** es un par (p, c) donde $p \in P$ y $c \in C(p)$, mientras que un **elemento atadura** es un par (t, b) donde $t \in T$ y $b \in B(t)$. El conjunto de todos los elementos token es denotado por TE mientras que el conjunto de todos los elementos atadura se denota por BE.*

*Una **marca** es un multi-conjunto sobre TE mientras que un **paso** es un multi-conjunto no vacío y finito sobre BE. La **marca inicial** M_0 es la marca que se obtiene por la evaluación de las expresiones de iniciación:*

$$\forall (p, c) \in TE : M_0(p, c) = (I(p))(c).$$

Los conjuntos de todas las marcas y pasos están denotados por $\mathcal{M}$ y $\mathcal{Y}$, respectivamente.

Cada marca $M \in TE_{MC}$ determina una función única M^* definida en P tal que $M^*(p) \in C(p)_{MC}$:

$$\forall p \in P \ \forall c \in C(p) : (M^*(p))(c) = M(p, c).$$

Por otro lado, cada función M^* definida en P tal que $M^*(p) \in C(p)_{MC}$ para todo $p \in P$, determina una única marca M:

$$\forall (p, c) \in TE : M(p, c) = (M^*(p))(c).$$

Así frecuentemente se representan las marcas como funciones definidas en P (y se usará el mismo nombre para la función y para la representación de multi-conjunto de una marca).

Ahora, se puede dar la definición formal de habilitación y ocurrencia (en la cual se respresentan pasos por multi-conjuntos y se representan marcas como funciones).

Definición 1.7 *Un paso $Y \in \mathcal{Y}$ está **habilitado** en una marca $M \in \mathcal{M}$ ssi se satisface la siguiente propiedad:*

$$\forall p \in P : \sum_{(t,b) \in Y} E(p,t)\langle b \rangle \leq M(p)$$

Se dice entonces que (t,b) está habilitado y también t se encuentra habilitado, es decir, la transición t está lista para dispararse. Los elementos de Y están habilitados concurrentemente cuando $|Y| \geq 1$.

*Cuando un paso Y está habilitado en una marca M_1 ésta puede **ocurrir**, cambiando la marca M_1 a otra marca M_2, definida por:*

$$\forall p \in P : M_2(p) = (M_1(p) - \sum_{(t,b) \in Y} E(p,t)\langle b \rangle) + \sum_{(t,b) \in Y} E(t,p)\langle b \rangle$$

*M_2 es **directamente alcanzable** desde M_1. Esto se escribe como $M_1[Y\rangle M_2$*

La evaluación de la expresión $E(p,t)\langle b \rangle$ nos da los tokens que serán consumidos de p cuando ocurre t con la atadura b. Tomando la suma sobre todos los elementos atadura $(t,b) \in Y$ obtenemos todos los tokens que son consumidos de p cuando Y ocurre. Se requiere que este multi-conjunto sea menor o igual a la marca de p. Esto significa que cada elemento atadura $(t,b) \in Y$ debe de poder obtener los tokens especificados por $E(p,t)\langle b \rangle$, sin tener que compartir estos tokens con otros elementos atadura de Y. Se debe recordar que todas las ataduras de un paso, de acuerdo con la definición (1.5), automáticamente satisfacen sus correspondientes guardias. Más aún, se debe notar que las sumas en la definición (1.7) son sumas sobre un multiconjunto Y. Cuando un elemento atadura aparece más de una vez en Y, se obtiene una contribución a cada aparición.

La ocurrencia de un paso es un evento indivisible. Aunque la fórmula de arriba requiere de que se lleve a cabo una substracción antes que una suma, **no** se reconoce la existencia de una marca intermedia, donde los tokens en la primera suma hayan sido eliminados mientras aquellos en la segunda no hayan sido aún añadidos. También debe notarse que un paso **no** necesita ser máximo. Cuando un número de elementos atadura están concurrentemente habilitados, es posible tener un paso ocurriendo que sólo contenga algunos de estos elementos.

Definición 1.8 *Una **secuencia finita de ocurrencias** es una secuencia de marcas y pasos:*

$$M_1[Y_1\rangle M_2[Y_2\rangle M_3 \ldots M_n[Y_n\rangle M_{n+1}$$

*tal que $n \in I\!N$, y $M_i[Y_i\rangle M_{i+1}$ para todo $i \in \{1, 2, \ldots, n\}$ M_1 es la **marca inicial**, M_{n+1} es la **marca final** y n es la **longitud**.*

*Análogamente, una **secuencia infinita de ocurrencias** es una secuencia de marcas y pasos:*

$$M_1[Y_1\rangle M_2[Y_2\rangle M_3 \ldots$$

Tal que $M_i[Y_i\rangle M_{i+1}$ para todo $i \geq 1$.

*Una marca M'' es **alcanzable desde** una marca M' ssi existe una secuencia finita de ocurrencias iniciando en M' y finalizando en M''. El conjunto de marcas las cuales son alcanzables desde M' es denotado por $[M'\rangle$. Una marca es entonces alcanzable si pertenece a $[M_0\rangle$.*

1.3.3 Ejemplo de Aplicación

Regresando al problema de la cena de los filósofos, una Red de Petri coloreada se muestra en la figura 1.5 (cf. [KhA91]), en donde se visualiza la actividad de los filósofos utilizando los lugares de: los *tenedores*, los *filósofos pensando* y los *filósofos comiendo*. Así podemos colocar los 5 tenedores como tokens en su lugar respectivo y en lugar de los filósofos pensando, inicialmente se colocan los tokens de los 5 filósofos. Se incluyen dos transiciones, como en el caso anterior, para pasar al estado de comiendo o pensando.

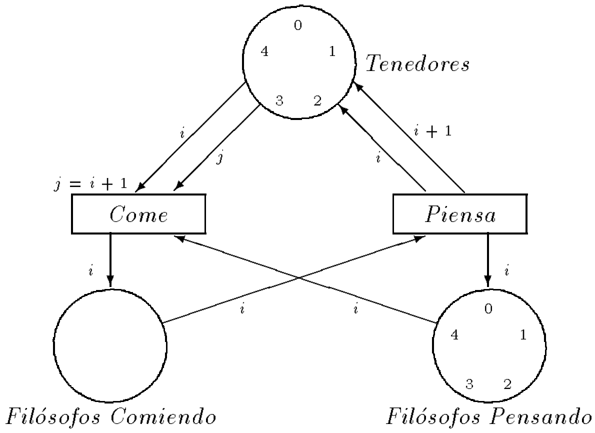

Figura 1.5: El problema de la cena de los filósofos modelado con una Red de Petri Coloreada

Para que la transición Come se dispare, se requiere de un $i-$ésimo filósofo que esté en el lugar de Pensando y sus respectivos tenedores i e $i+1$, una vez que se tienen estos tokens coloreados, la transición Come se puede disparar y el $i-$ésimo filósofo pasa al lugar *Filósofos Comiendo*, consumiendo los tokens respectivos.

En la figura 1.6 se muestra el disparo de la transición *Come* tomando al filósofo 3 y a los tenedores 3 y 4, produciendo al filósofo 3 en el lugar *Filósofo Comiendo*

La transición **Piensa** únicamente requiere el token de un filósofo que se encuentre en el lugar Comiendo. En el momento en que se dispare la tran-

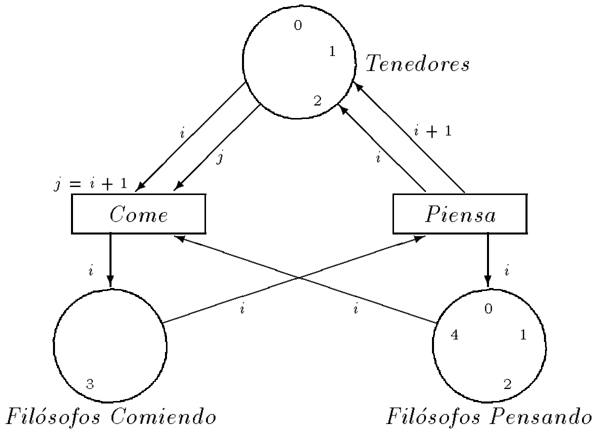

Figura 1.6: El problema de la cena de los filósofos: La transición *Come* se ha disparado

sición **Piensa**, produce los tokens correspondientes a los tenedores colocados en su lugar y el token correspondiente al filósofo que está ahora Pensando.

Formalmente la Red de Petri de la figura 1.5 se puede denotar de la siguiente manera de acuerdo con la definición (1.4).

$RPC = (\Sigma, P, T, A, N, C, G, E, I)$, donde :
$\Sigma = \{Tenedor, Filosofo\}$,
$Tenedor = \{0, 1, 2, 3, 4\}$, $Filosofo = \{0, 1, 2, 3, 4\}$,
$P = \{Tenedores, Comiendo, Pensando\}$,
$T = \{Come, Piensa\}$,
$A = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$,
$N(a_1) = (Tenedores, Come)$, $N(a_2) = (Tenedores, Come)$,
$N(a_3) = (Pensando, Come)$, $N(a_4) = (Come, Comiendo)$,
$N(a_5) = (Comiendo, Piensa)$, $N(a_6) = (Piensa, Tenedores)$,
$N(a_7) = (Piensa, Tenedores)$, $N(a_8) = (Piensa, Pensando)$,
$C(Tenedores) = Tenedor$, $C(Comiendo) = Filosofo$,
$C(Pensando) = Filosofo$,
$G(Come) = (j{=}i + 1)$, $G(Piensa) = (True)$,
$E(a_1) = i$, $E(a_2) = j$, $E(a_3) = i$, $E(a_4) = i$,
$E(a_5) = i$, $E(a_6) = i$, $E(a_7) = i + 1$, $E(a_8) = i$,
$I(Tenedores) = [0, 1, 2, 3, 4]$, $I(Pensando) = [0, 1, 2, 3, 4]$.

Capítulo 2

Análisis de las Redes de Petri

Para un estudio más completo de las Redes de Petri se requiere analizar su comportamiento dinámico, esto es con la finalidad de mostrar la conducta de una Red de Petri que se encuentra en 'ejecución'. Este capítulo se enfoca a la definición formal y al estudio del comportamiento dinámico de las Redes de Petri Coloreadas con Intervalos de Tiempo, que son más poderosas en cuanto a su potencia de modelado. Cuando se requiere de un control temporalizado de los disparos de las tranisiciones, es preciso hacer una extensión al modelo de Red de Petri Coloreada agregando el concepto de intervalos de tiempo. El motivo de esta extensión es debido a que generalmente para la ejecución de un proceso se requiere consumir un tiempo determinado.

2.1 Redes de Petri Coloreadas con Intervalos de Tiempo (RPCIT)

El modelo de **Redes de Petri Coloreadas con Intervalos de Tiempo** (RPCIT) se propone como una extensión al modelo de Redes de Petri Coloreadas para permitir la manipulación del tiempo, esto es por la razón de que existen ciertos procesos los cuales involucran un lapso de tiempo durante su ejecución, por ejemplo el tiempo que se tarda en producirse una reacción química o el tiempo que se demora un obrero en ejecutar su trabajo.

El modelo RPCIT utiliza un mecanismo de tiempo novedoso donde el tiempo es asociado a los tokens. El concepto de tiempo ha sido adoptado de Van Hee [vaH94]. A cada transición se le debe asignar un retardo de disparo. Se podría dar a cada transición un retardo fijo, pero en ocasiones el retardo de una transición puede depender de influencias externas. Por lo que los retardos se especifican con intervalos de tiempo. Entonces una transición en este modelo recibe un intervalo y el retardo de disparo de cada transición debe estar dentro de su intervalo.

En este modelo se le asigna una **etiqueta de tiempo** a cada token. Esta etiqueta de tiempo indica el tiempo en que cada token se hace disponible. El hecho de asociar el tiempo con los tokens, nos proporciona una forma natural de utilizar Redes de Petri de alto nivel además de que el **color** es asociado a ellos. El **tiempo de habilitación** de una transición es la etiqueta de tiempo máxima de los tokens a ser consumidos con el disparo de la transición. Como las transiciones se disparan tan pronto como sea posible, la transición con el menor tiempo de habilitación se disparará primero.

Disparar una transición significa consumir todos los tokens de las entradas y producir tokens en las salidas. Si en cualquier momento más de una transición es habilitada, habría muchas posibilidades de que alguna de las transiciones pueda ser 'la siguiente' a ser disparada. Esto hace una decisión no determinística si existen transiciones con el mismo tiempo de disparo.

El disparo es una acción atómica, por ello se producen tokens con etiqueta de tiempo de al menos el tiempo de disparo. La marca de tiempo indica el tiempo en el cual lo nuevos tokens estarán disponibles para otras transiciones. La diferencia entre el tiempo de disparo y la etiqueta de tiempo de un token producido es llamado **retardo de disparo**. El retardo (de disparo) de un token producido es especificado por un límite superior y un límite inferior de tiempo, esto es, un **intervalo**.

Usaremos un ejemplo para introducir la noción de Redes coloreadas de Petri con intervalos de tiempo [vdA95]. La figura 2.1 muestra una RPCIT compuesta de cuatro **lugares** ($p_{entrada}, p_{ocupado}, p_{libre}$ y p_{salida}) y dos **transiciones** (t_1 y t_2). En cualquier momento un lugar contiene cero o más **tokens**, dibujados como puntos. En el modelo RPCIT, un token tiene tres atributos: una posición, un valor y un tiempo, i.e. podemos usar la tupla

$\langle\langle p,v\rangle,x\rangle$ para denotar un token en el lugar p con valor v y un tiempo x. El valor de un token es regularmente referido como **token coloreado**. Cada lugar tiene un **conjunto de color** ligado a él, que especifica el conjunto de valores permitidos, i.e. cada token residiendo en el lugar p debe tener un color (valor) que es miembro del conjunto color de p.

La RPCIT de la figura 2.1 representa un sistema de computadora abstracto, los trabajos llegan a través del lugar $p_{entrada}$ y salen del sistema por p_{salida}. La computadora está compuesta por un número de procesadores paralelos. Cada procesador está representado por un token que puede estar en el lugar p_{libre} o en el lugar $p_{ocupado}$. El inicio y terminación de la ejecución de un trabajo en un procesador está representado por las transiciones t_1 y t_2 respectivamente. Hay tres conjuntos de color $\mathcal{P} = \{P1, P2, \ldots\}$ y $\mathcal{J} = \{J1, J2, \ldots\}$ y $\mathcal{P} \times \mathcal{J}$. El conjunto de color $\mathcal{J}$ (tipos de trabajo) está ligado a los lugares $p_{entrada}$ y p_{salida}, el conjunto de color $\mathcal{P}$ (tipos de procesador) está ligado al conjunto p_{libre}. El conjunto $\mathcal{P} \times \mathcal{J}$ está ligado al lugar $p_{ocupado}$.

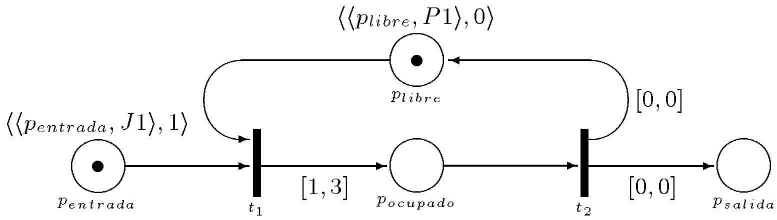

Figura 2.1: Una Red de Petri Coloreada con Intervalos de Tiempo

Los lugares y transiciones están interconectados por **arcos**. Cada arco conecta un lugar y una transición en una dirección precisamente. La transición t_1 tiene dos lugares de entrada ($p_{entrada}$ y p_{libre}) y un lugar de salida ($p_{ocupado}$). La transición t_2 tiene un lugar de entrada ($p_{ocupado}$) y dos lugares de salida (p_{libre} y p_{salida}).

El número de tokens producidos por el disparo de una transición puede depender de los valores de los tokens consumidos. Además los valores y los retardos de los tokens producidos dependerán también del valor de los tokens producidos. La **función de transición** describe la relación entre el **multi-conjunto** de tokens consumidos y el **multi-conjunto** de tokens consumidos. La Función $F(t_1)$ especifica la transición t_1 en la red de la figura 2.1:

$dom(F(t_1)) = \{\langle p_{entrada}, j\rangle + \langle p_{libre}, p\rangle | j \in \mathcal{J} \text{ y } p \in \mathcal{P}\ \}$
Para $j \in \mathcal{J}$ y $p \in \mathcal{P}$, tenemos:
$F(t_1)(\langle p_{entrada}, j\rangle + \langle p_{libre}, p\rangle) = \langle\langle p_{ocupado}, \langle p, j\rangle\rangle, [1,3]\rangle$

El dominio de $F(t_1)$ describe la condición para que la transición t_1 es habilitada, i.e. t_1 se habilita sólo cuando tiene un token (al menos) en cada uno de sus lugares de entrada. Esto significa que la transición t_1 puede disparar si hay un trabajo esperando y alguno de los procesadores está libre. Hay que notar en este caso que la habilitación de una transición no depende de los valores de los tokens consumidos, depende de las etiquetas de tiempo de los tokens a ser consumidos.

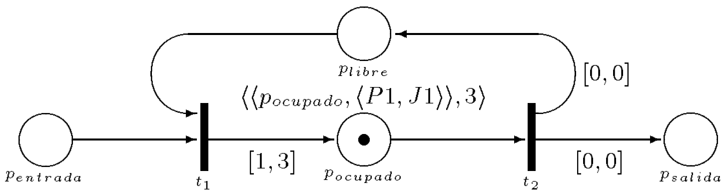

Figura 2.2: Disparo de la transición t_1

Si t_1 ocurre, consume un token de cada una de sus entradas y produce un token en su salida. El color del token producido $p_{ocupado}$ es el par $\langle p, j\rangle$ donde p representa el procesador y j el trabajo. El retardo de este token es un valor arbitrario entre 1 y 3. El resultado del disparo de t_1 se muestra en la figura 2.2. La transición t_2 se especifica como:

$dom(F(t_2)) = \{\langle p_{ocupado}, \langle p, j\rangle\rangle | j \in \mathcal{J} \text{ y } p \in \mathcal{P}\ \}$
Para $j \in \mathcal{J}$ y $p \in \mathcal{P}$, tenemos:
$F(t_2)(\langle p_{ocupado}, \langle p, j\rangle\rangle) = \langle\langle p_{libre}, p\rangle, [0,0]\rangle + \langle\langle p_{salida}, j\rangle, [0,0]\rangle$

La transición representa la terminación de un trabajo. Si t_2 ocurre, consume un token de su entrada y produce dos tokens en su salida, ambos con un retardo igual a cero. Si t_2 ocurre en el estado mostrado en la figura 2.2, entonces el estado resultante contiene dos tokens: $\langle\langle p_{salida}, J1\rangle, 3\rangle$ y $\langle\langle p_{libre}, P1\rangle, 3\rangle$.

2.2 Definición Formal

En esta sección se definirá una Red de Petri Coloreada con Intervalos de Tiempo en términos matemáticos. Así como funciones y relaciones.

2.2.1 Definición de RPCIT

Antes de introducir la definición formal de las Redes de Petri Coloreadas con Intervalos de Tiempo hay que notar que casi todas utilizan un dominio de tiempo continuo, ya que una etiqueta de tiempo indica en qué tiempo el token se hará disponible para una transición. Definimos entonces el conjunto de tiempos sobre los que se toman los intervalos:

Definición 2.1 *TS es el **conjunto de tiempos**, $TS = \{x \in \mathbb{R} | x \geq 0\}$, i.e. el conjunto de todos los reales no negativos, $INT = \{[y, z] \in TS \times TS | y \leq z\}$, representa el conjunto de todos los intervalos cerrados. Si $x \in TS$ y $[y, z] \in INT$ entonces $x \in [y, z]$ si y sólo si $y \leq x \leq z$.*

Formalmente una Red de Petri coloreada con intervalos de tiempo se define de la siguiente forma (cf [vdA91]):

Definición 2.2 (RPCIT) *Se define una **Red de Petri Coloreada con Intervalos de Tiempo** como una tupla $RPCIT = (\Sigma, P, T, C, CT, G, F)$ que satisface los siguientes requerimientos:*

*(a) Σ es un conjunto finito de tipos, llamados **conjuntos de colores**.*

*(b) P es un conjunto finito de **lugares**.*

*(c) T es un conjunto finito de **transiciones**.*

*(d) C es una **función de coloreo**. Es definida de P a Σ, i.e. $C : P \to \Sigma$.*

*(e) $CT = \{(p, v) | p \in P \wedge v \in C(p)\}$ es el conjunto de todos los posibles **tokens coloreados**.*

(f) *G es una función de* ***guardia****. Está definida de T hacia expresiones booleanas. Si* $t \in T$ *se tiene:* [1]
$Tipo(G(t)) = I\!B \wedge Tipo(Var(G(t))) \subseteq \Sigma$.

(g) *F es la* ***Función de Transición****. Es definida de T en funciones. Si* $t \in T$ *entonces:* [2]
$F(t) : CT_{MC} \not\rightarrow (CT \times INT)_{MC}$

(a) Σ es un conjunto de tipos. Cada tipo es un conjunto de colores que pueden estar asignados a uno de los lugares.

(b) y (c) Los lugares y transiciones son descritos como dos conjuntos disjuntos i.e. $P \cap T = \emptyset$.

(d) Cada lugar $p \in P$ tiene un conjunto de colores ligado a él y esto significa que cada token residiendo en p debe tener un valor v, el cual es un elemento de este conjunto, i.e. $v \in C(p)$.

(e) CT es el conjunto de todos los token coloreados, i.e. $\langle p, v \rangle$ donde p es la posición del token y v es el color del token.

La función de Guardia (f) tiene la misma utilización que en las Redes de Petri coloreadas, donde $G(t)$ mapea cada transición t, en una expresión booleana donde todas las variables tienen tipos que pertenecen a Σ.

(g) La función de transición especifica cada transición en la RPCIT. Para una transición t, $F(t)$ especifica la relación entre el multi-conjunto de tokens consumidos y el multi-conjunto de tokens producidos. El domino de $F(t)$ describe la condición en la que la transición t es habilitada. Hay que notar que el token producido tiene un retardo especificado por un intervalo. Se necesita que los multi-conjuntos de tokens consumidos y los multi-conjuntos de tokens producidos contengan un número finito de elementos.

En la definición se puede notar la ausencia de Arcos, Nodos, y Expresiones de arco que se encontraban definidos en las Redes de Petri coloreadas; el motivo de la ausencia es por lo siguiente:

[1] El conjunto $I\!B$ denota al conjunto Booleano, $I\!B = \{Falso, Verdadero\}$
[2] $A \not\rightarrow B$ denota los conjuntos de funciones parciales de A a B

- Con los arcos y nodos se determina la relación entre los lugares y transiciones; los arcos determinan de dónde tomar los tokens y hacia dónde producirlos entre parejas de nodos lugar-transición y transición-lugar, respectivamente. La función de transición define por su misma naturaleza la relación entre los lugares de entrada a una transición y los lugares de salida de la misma, de esta manera se determina la relación entre el multi-conjunto de los tokens consumidos y el multi-conjunto de tokens producidos.

- El dominio de la función de transición describe las condiciones con las cuales una transición se puede habilitar. En el caso de las Redes de Petri coloreadas, la expresión en un arco de entrada a una transición determina las condiciones para que se pueda habilitar y las expresiones en un arco de salida, determinan los tokens que se producen. La definición de la función de transición puede verse como una expresión, en donde las variables son los lugares y los valores que se sustituyen en éstas son los mismos colores. El dominio de esta función determina cuántos y cuales tokens son requeridos para habilitar la transición.

Debe notarse también, como se indicó en el inciso (b) de la definición (1.5), que una función Guardia de una transición debe ser satisfecha por todas las ataduras con los colores apropiados, por lo que se requiere evaluar a verdadero para que la transición se pueda habilitar. Se puede observar en la definición (1.7) que no se requiere de la función Guardia para determinar la habilitación de un paso ni para el disparo de una transición. De igual forma sucede con las RPCIT's a las cuales se les agrega el manejo de precondiciones para obtener un modelo más robusto que abarca colores, expresiones e intervalos de tiempo.

Para modelar una Red de Petri coloreada con una Red de Petri coloreada con intervalos de tiempo, únicamente se requiere eliminar todo cálculo de tiempo, definiendo la función de transición con un intervalo de tiempo nulo, es decir $[0, 0]$ colocando tokens en cada uno de los lugares llevando una etiqueta de tiempo con valor cero. Con esto se puede indicar una ausencia de tiempo en la Red de Petri.

2.3 Comportamiento Dinámico de RPCIT

La tupla $(\Sigma, P, T, C, CT, G, F)$ especifica una estructura estática de una RPCIT. La semántica del modelo RPCIT define un comportamiento de una Red coloreada de Petri con intervalos de tiempo.

Definición 2.3 *Un **estado** es definido como un multi-conjunto de tokens coloreados llevando una etiqueta de tiempo. S es el **espacio de estados**, i.e. el conjunto de todos los posibles estados:*
$S = (CT \times TS)_{MC}$
*La **marca** de una RPCIT en un estado $s \in S$ es la distribución de tokens 'no temporalizada': $M(s) \in CT_{MC}$ y*

$$M(s) = \sum_{\langle\langle p,v\rangle,x\rangle \in CT \times TS} s(\langle\langle p,v\rangle,x\rangle)\langle p,v\rangle$$

Un estado de una RPCIT es un multi-conjunto de tokens coloreados llevando una etiqueta de tiempo, i.e. un multi-conjunto de tuplas $\langle\langle p,v\rangle,x\rangle$ $(p \in P, v \in C(p)$ y $x \in TS)$. El estado mostrado en la figura 2.1 es $\langle\langle p_{entrda}, J1\rangle, 1\rangle + \langle\langle p_{libre}, M1\rangle, 0\rangle$, que es el estado con un token en $p_{entrada}$ con un valor $J1$ y un token en p_{libre} con un valor $M1$. El token en $p_{entrada}$ lleva una etiqueta de tiempo 1 y el token en p_{libre} lleva una etiqueta de tiempo 0.

Definición 2.4 *Un **evento** es un triple $\langle t, d_e, d_s\rangle$, el cual representa el posible diparo de una transición t mientras se eliminan los tokens especificados por el multi-conjunto d_e y agregando los tokens especificados por el multi-conjunto d_s. E es el **conjunto de eventos:***
$E = T \times (CT \times TS)_{MC} \times (CT \times TS)_{MC}$

Un evento $e = \langle t, d_e, d_s\rangle$ representa el disparo de t mientras consume tokens especificados por d_e y produciendo tokens especificados por d_s. Si $\langle\langle p,v\rangle,x\rangle \in d_e$, entonces e consume un token de p con valor v y una etiqueta de tiempo x. Si $\langle\langle p',v'\rangle,x'\rangle \in d_e$, entonces e produce un token para p' con valor v' y un **retardo** x'. Hay que notar que x' es relativo al tiempo de disparo y x' es un miembro de uno de los intervalos de retardo especificados por $F(t)(d_e)$. Para seleccionar miembros arbitrarios de esos intervalos de retardo, se necesita el concepto de especialización.

Definición 2.5 (Especialización) *Para relacionar multi-conjuntos de tokens llevando etiquetas de tiempo con multi-conjuntos de tokens, llevando intervalos (de tiempo), se define la* ***relación de especialización***, $\lhd \subseteq (CT \times TS)_{MC} \times (CT \times INT)_{MC}$. *Para* $d \in (CT \times TS)_{MC}$ *y* $\overline{d} \in (CT \times INT)_{MC}$, $d \lhd \overline{d}$ *si y sólo si cada token en d corresponde a un token exactamente en* $\overline{d}$, *tal que estén en el mismo lugar, tengan el mismo valor y la etiqueta de tiempo del token en d esté en el intervalo (de tiempo) del token en* $\overline{d}$.
Más formalmente: $d \lhd \overline{d}$ *si y sólo si* ($d = \emptyset$ *y* $\overline{d} = \emptyset$) *en otro caso*

$\exists_{\langle\langle p,v\rangle,x\rangle \in d} \exists_{\langle\langle p,v\rangle,[y,z]\rangle \in \overline{d}} : (x \in [y,z])$ *y* $(d - \langle\langle p,v\rangle,x\rangle) \lhd (\overline{d} - \langle\langle p,v\rangle,[y,z]\rangle)$

Como ejemplo podemos considerar:
$\emptyset \lhd \emptyset$
$\langle\langle p_{entrada}, J1\rangle, 1\rangle \lhd \langle\langle p_{entrada}, J1\rangle, [0.5, 1.5]\rangle$
$\langle\langle p_{entrada}, J1\rangle, 1\rangle + 2\langle\langle p_{libre}, P1\rangle, 0\rangle \lhd \langle\langle p_{entrada}, J1\rangle, [0.5, 1.5]\rangle + 2\langle\langle p_{libre}, P1\rangle, [0, 1]\rangle$

Si $d \lhd \overline{d}$, entonces existe una biyección entre los tokens en d y los tokens en $\overline{d}$ tal que cada token en d corresponde exactamente a un token en $\overline{d}$ el cual está en el mismo lugar, tiene el mismo valor y corresponden en el intervalo de tiempo.

Definición 2.6 *Un evento* $\langle t, d_e, d_s\rangle \in E$ *es* ***habilitado*** *en el estado* $s \in S$ *si y sólo si:*

(a) $d_e \leq s$

(b) $M(d_e) \in dom(F(t))$

(c) $d_s \lhd F(t)(M(d_e))$

Un evento es habilitado si y sólo si:
(a) Los token a ser consumidos están presentes en el estado actual.

(b) Una transición es habilitada si hay 'suficientes' tokens en cada uno de sus lugares de entrada, esto es especificado por el dominio de $F(t)$. Hay que notar que la habilitación puede depender de los valores de los tokens a ser consumidos, pero no de sus etiquetas de tiempo.

(c) El número y valores de los tokens a ser producidos están determinados por el multi-conjunto $F(t)(M(d_e))$. Este multi-conjunto también especifica las cotas superior e inferior de los retardos de estos tokens.

Definición 2.7 *El **tiempo de habilitación** de un evento $\langle t, d_e, d_s \rangle \in E$ es el máximo de todas las etiquetas de tiempo de los tokens a ser consumidos, i.e.*

$$ET(\langle t, d_e, d_s \rangle) = \max_{\langle\langle p,v\rangle,x\rangle \in d_e} x$$

*Un evento habilitado es **habilitado por tiempo** si y sólo si no hay otros eventos habilitados que tengan menor tiempo de habilitación.*

Si un evento es habilitado por tiempo entonces puede ocurrir, es decir la transición podría dispararse. De hecho, una transición se dispara tan pronto como sea posible (las transiciones están 'deseosas' de disparar). Aunque el dominio del tiempo es continuo ($TS = \{x \in \mathbb{R} \,|x \geq 0\}$), el tiempo avanza discontinuamente.

Definición 2.8 *El **tiempo modelo** de un estado $s \in S$ es el mínimo de los tiempos de habilitación, esto es*

$$MT(s) = min\{ET(e)|e \in E \text{ y } e \text{ es habilitado en el estado } s\}$$

Nótese que un evento habilitado e es habilitado por tiempo en el estado s si y sólo si $ET(e) = MT(s)$.

Si $\langle t, d_e, d_s \rangle \in E$ es un evento habilitado, entonces una etiqueta de tiempo en d_s representa el retardo del token en lugar de una etiqueta de tiempo absoluta. Por lo tanto necesitaremos una función para 'escalar' las etiquetas de tiempo.

Definición 2.9 *La función* $SC : ((CT \times TS)_{MC} \times TS) \rightarrow (CT \times TS)_{MC}$ *escala las etiquetas de tiempo en un multi-conjunto de tokens coloreados de tiempo. Para* $d \in (CT \times TS)_{MC}$ *y* $y \in TS$*, tenemos:*

$$SC(d, y) = \sum_{\langle\langle p,v\rangle,x\rangle \in CT \times TS} d(\langle\langle p, v\rangle, x\rangle)\langle\langle p, v\rangle, x + y\rangle$$

Un evento habilitado por tiempo e en el estado s puede ocurrir en el tiempo $ET(e) = MT(s)$.

Definición 2.10 *Si un evento habilitado* $\langle t, d_e, d_s\rangle$ *es habilitado por tiempo en el estado* s_1 *puede entonces* ***ocurrir****, i.e. la transición* t *se* ***dispara*** *mientras elimina los tokens especificados por* d_e *y añade tokens especificados por* d_s*.*
Si $\langle t, d_e, d_s\rangle$ *ocurre en el estado* s_1*, entonces la red cambia al estado* s_2*, definido por:*
$s_2 = (s_1 - d_e) + SC(d_s, ET(\langle t, d_e, d_s\rangle))$
Al estado s_2 *se le denomina* ***directamente alcanzable*** *desde* s_1 *por la ocurrencia del evento* $e = \langle t, d_e, d_s\rangle$*, esto es denotado también por:*
$s_1 \xrightarrow{e} s_2$

Más aún, $s_1 \longrightarrow s_2$ significa que existe un evento habilitado por tiempo e tal que $s_1 \xrightarrow{e} s_2$. Las transiciones se disparan tan pronto como sea posible, i.e. si un evento ocurre, entonces ocurre en su tiempo de habilitación. Hay que notar que el estado mostrado en la figura 2.2 es directamente alcanzable del estado mostrado en la figura 2.1.

Definición 2.11 *Una* ***secuencia de disparo*** *es una secuencia de estados y eventos:*
$s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} \cdots$

*El estado s_n es **alcanzable** desde s_1 si y sólo si existe una secuencia de disparos de longitud finita empezando desde s_1 y terminando en s_n:*
$s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} \cdots \xrightarrow{e_{n-1}} s_n$

Una propiedad importante del modelo RPCIT es la monotonía en que transcurre el tiempo, esto significa, que el tiempo sólo se puede mover hacia adelante.

2.4 Método de Análisis

Existen técnicas utilizadas para analizar el comportamiento dinámico de las Redes de Petri con tiempo y coloreadas. Un método para verificar ciertas propiedades y calcular cotas para todas las medidas de tiempo está basado en un método de análisis llamado **Técnica de Reducción de Sistemas con Transiciones Modificadas** (TRSTM), el cual genera un **grafo de alcanzabilidad reducido**. El método TRSTM puede ser aplicado a RPCIT arbitrarias [vdA91].

La técnica de análisis de **alcanzabilidad** consiste en construir un grafo de alcanzabilidad (en ocasiones árbol de alcanzabilidad o grafo de ocurrencia). Un grafo de alcanzabilidad contiene un nodo por cada posible estado y un arco para cada posible cambio de estado. El análisis de alcanzabilidad es un método muy poderoso en el sentido de que puede ser usado para probar todo tipo de propiedades.

En el modelo RPCIT, un retardo es descrito por un intervalo mas que con un valor fijo o alguna distribución de retardos. Por un lado, los intervalos de retardo permiten modelar retardos variables, y por otro lado, no es necesario determinar alguna distribución artificial de retardos. En lugar de eso únicamente se tienen que especificar cotas inferior y superior para especificar y verificar restricciones de tiempo. Esto es importante cuando se modelan sistemas de tiempo real con 'líneas muertas'. Por lo tanto, nos interesa estudiar las técnicas que verifican estas lineas muertas y calcular las cotas superior e inferior para todos los tipos de criterios de desempeño.

2.4.1 Grafos de Alcanzabilidad

En la sección (2.3) se definió el comportamiento dinámico de una RPCIT. Estas definiciones pueden ser usadas para construir un **grafo de alcanzabilidad**. La idea básica de un grafo de alcanzabilidad es organizar todas las marcas alcanzables en un grafo, donde cada nodo representa un estado y cada arco representa un evento transformando un estado en otro estado [vdA95].

Por ejemplo se considera el grafo de alcanzabilidad mostrado en la figura 2.3. Suponer que s_1 es el estado inicial de la RPCIT en cuestión. Este estado es conectado a un número de estados $s_{11}, s_{12}, s_{13}, \ldots$ alcanzables por s_1 dado el disparo de alguna transición $(s_1 \longrightarrow s_{1i})$. Estos estados son llamados los 'sucesores' de s_1.

Repitiendo este proceso se produce la representación del grafo de alcanzabilidad mostrado en la figura 2.3. Se puede ver que un grafo de alcanzabilidad contiene toda la información acerca del comportamiento dinámico del sistema [vdA91].

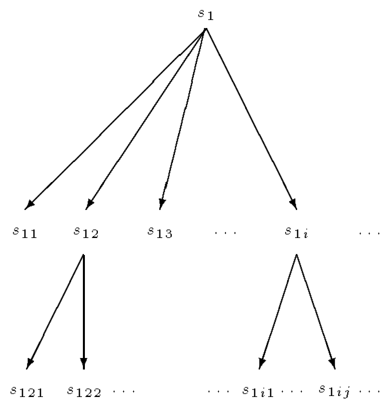

Figura 2.3: Un grafo de alcanzabilidad

Si se tiene la posibilidad para generar este grafo de alcanzabilidad, se podría responder 'cualquier' tipo de pregunta acerca del comportamiento dinámico del sistema. Obviamente tal grafo puede llegar a ser muy grande e incluso infinito, aún para redes pequeñas.

En general el número de estados alcanzables de una RPCIT (dado un estado inicial) es infinito. La causa principal de esto es el hecho de que se utilizan intervalos de tiempo. Al considerar una transición habilitada, generalmente hay un número infinito de retardos de disparo permitidos y todos resultan en diferentes estados.

Considerando por ejemlo la RPCIT de la figura 2.4, si se dispara la transición t, entonces consume el token de p_1 y produce un token para p_2 y un token para p_3. con intervalos de tiempo $[1, 2]$ y $[3, 4]$. Supongamos que el estado inicial es tal que hay un token en el lugar p_1 con etiqueta de tiempo 0. El número de sucesores de este estado es infinito, porque todos los estados con un token en p_2 teniendo una etiqueta de tiempo $x \in [1, 2]$ y un token en p_3 teniendo una etiqueta de tiempo $y \in [3, 4]$ son alcanzables.

Puede sonar ilógico que este simple ejemplo corresponda a un grafo de alcanzabilidad con un número infinito de estados. Esta es la razón por la que se desarrolla la técnica de Reducción de Sistemas con Transiciones Modificadas, que genera el grafo de alcanzabilidad y utiliza sólo por motivos computacionales, definiciones alternativas para el comportamiento dinámico de una RPCIT.

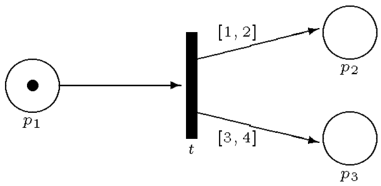

Figura 2.4: Ejemplo de RPCIT

2.4.2 Reducción del Grafo de Alcanzabilidad de una RPCIT

La Técnica de Reducción de Sistemas con Transiciones Modificadas, utiliza definiciones alternativas para el comportamiento dinámico de una RPCIT, esto es utiliza una semántica alternativa. La diferencia principal entre estas definiciones y las originales es el hecho de que se liga un **intervalo de tiempo** a cada uno de los tokens en lugar de una etiqueta de tiempo, esto es, $\overline{S} = (CT \times INT)_{MC}$. Se podrá ver que usando esta semántica, es posible calcular el conjunto de todos los estados alcanzables, o al menos un subconjunto relevante. El grafo de alcanzabilidad de la RPCIT utilizando este tipo de semántica es mucho más pequeño que el grafo original, lo llamaremos ahora un **grafo de alcanzabilidad reducido** [vdA91].

Cada estado en el grafo de alcanzabilidad reducido corresponde a un número (infinito) de estados en el grafo de alcanzabilidad original. Estos estados se pueden pensar como una **equivalencia** o **clases de estados**. Una clase de estado $\overline{s} \in \overline{S}$ corresponde al conjunto de todos los estados que son una especialización de s, esto es $\{s \in S | s \lhd \overline{s}\}$. Informalmente hablando, las clases de estado están definidas como la unión de estados 'similares' teniendo la misma distribución de tokens (marcas) pero diferentes etiquetas de tiempo dentro de ciertas cotas.

Podremos ver que las definiciones alternativas (tomadas de [vdA91]) que se darán pueden ser usadas para responder preguntas acerca del comportamiento dinámico de la RPCIT.

Definición 2.12 *Una clase de estados es definida como un multi-conjunto de tokens coloreados, cada uno llevando un intervalo de tiempo. $\overline{S}$ es el* ***espacio de estados:*** [3]

$$\overline{S} = (CT \times INT)_{MC} \tag{2.1}$$

Un ***evento*** *es un triple $\langle t, d_e, d_s \rangle$, que representa el posible disparo de la transición t mientras elimina los tokens especificados por el multi-conjunto d_e y agregando tokens especificados por el multi-conjunto d_s. $\overline{E}$ es el* ***conjunto de eventos:***

$$\overline{E} = T \times (CT \times INT)_{MC} \times (CT \times INT)_{MC} \tag{2.2}$$

---

[3] Los símbolos con una línea encima están asociados a la semántica alternativa.

En la semántica alternativa hay que notar que los tokens llevan un intervalo de tiempo en lugar de etiquetas de tiempo como se tenía en la semántica original.

Definición 2.13 *Un evento* $\langle t, d_e, d_s \rangle \in \overline{E}$ *es* ***habilitado*** *en la clase de estado* $s \in \overline{S}$ *si y sólo si:*

(a) $d_e \leq s$

(b) $M(d_e) \in dom(F(t))$

(c) $d_s = F(t)(M(d_e))$

El momento exacto en que un token se hace disponible es especificado por un intervalo de tiempo, por lo tanto es imposible especificar el tiempo de habilitación de un evento $e \in \overline{E}$. Sin embargo, es posible determinar una cota superior y una cota inferior para el tiempo de habilitación del evento e en la semántica alternativa.

Definición 2.14 *El* ***tiempo de habilitación mínimo*** *y* ***el tiempo de habilitación máximo*** *de un evento* $\langle t, d_e, d_s \rangle \in \overline{E}$ *están definidos como sigue:*

$$ET_{min}(\langle t, d_e, d_s \rangle) = \max_{\langle\langle p,v \rangle,[y,z]\rangle \in d_e} y$$

$$ET_{max}(\langle t, d_e, d_s \rangle) = \max_{\langle\langle p,v \rangle,[y,z]\rangle \in d_e} z$$

Un evento habilitado e es ***habilitado por tiempo*** *si y sólo si el tiempo de habilitación mínimo de e es más pequeño que el tiempo de habilitación máximo de cualquier otro evento habilitado.*

Si $\langle t, d_e, d_s \rangle \in \overline{E}$ es un evento habilitado, entonces un intervalo de tiempo en d_s representa el intervalo de retardo del token en lugar de un intervalo de tiempo. Por lo tanto se necesita una función para escalar los intervalos de tiempo.

Definición 2.15 *La función* $\overline{SC} \in ((CT \times INT)_{MC} \times INT) \rightarrow (CT \times INT)_{MC}$ *escala los intervalos de tiempo en un multi-conjunto de tokens coloreados con tiempo. Para* $d \in (CT \times INT)_{MC}$ *y* $[y,z] \in INT$*, tenemos:*

$$\overline{SC}(d,[y,z]) = \sum_{\langle\langle p,v\rangle,[y',z']\rangle \in CT \times INT} d(\langle\langle p,v\rangle,[y',z']\rangle)\langle\langle p,v\rangle,[x'+x,y'+y]\rangle$$

Un evento e habilitado por tiempo en la clase de estado s puede ocurrir en un tiempo entre $ET_{min}(e)$ y $MT_{max}(s) = min\{ET_{max}(e)|e \in \overline{E}$ y e es habilitado en $s\}$.

Definición 2.16 *Cuando un evento habilitado* $\langle t,d_e,d_s\rangle$ *es habilitado por tiempo en la clase del estado* $s_1 \in \overline{S}$*, el evento puede* ***ocurrir****, esto es, la transición* t *se* ***dispara*** *eliminando los tokens especificados por* d_e *y agregando tokens especificados por* d_s*.*

Si $\langle t,d_e,d_s\rangle$ *ocurre en la clase de estado* s_1*, entonces la red cambia a la clase de estado* $s_2 \in \overline{S}$*, definido por:*

$$s_2 = (s_1 - d_e) + \overline{SC}(d_s,[ET_{min}(\langle t,d_e,d_s\rangle), MT_{max}(s_1)])$$

A la clase de estado s_2 se le denomina **directamente alcanzable** desde s_1 por la ocurrencia del evento $e = \langle t,d_e,d_s\rangle$. Esto es denotado por $s_1 \stackrel{e}{\Longrightarrow} s2$ en la semántica alternativa.

Hay que notar que las flechas dobles se utilizan para denotar posibles cambios de clase de estado dados en la semántica alternativa de una RPCIT. Si se utilizan las definiciones anteriores para la generación del grafo de alcanzabilidad, entonces podemos obtener el **grafo de alcanzabilidad reducido**.

Comparando estas definiciones con las definiciones de la sección (2.3), se puede ver que todas las diferencias parten del hecho de que la semántica alternativa asocia intervalos de tiempo en lugar de etiquetas de tiempo a cada token. Como resultado de estos intervalos, el tiempo de habilitación de un evento y el tiempo modelo de una clase de estado están caracterizados por una cota superior y una inferior.

En la semántica alternativa una clase de estado $\overline{s}$ es un multiconjunto de tuples $\langle\langle p, v\rangle, [y, z]\rangle$, cada tupla corresponde a un token en la red. Una clase de estados corresponde a un conjunto de estados, la clase de estados $\overline{s}$ corresponde al estado s si y sólo si existe una biyección entre los tokens en $\overline{s}$ y s tal que $\langle\langle p, v\rangle, [y, z]\rangle$ sea mapeado en $\langle\langle p, v\rangle, x\rangle$ con $x \in [y, z]$ (i.e. un token residiendo en el mismo lugar, teniendo el mismo valor y una etiqueta de tiempo que está dentro del intervalo de tiempo $[y, z]$. Se pueden pensar a las clases de estado como un tipo de clases de equivalencia.

Para ilustrar esta semántica alternativa, usaremos el ejemplo que se muestra en la figura 2.1. El estado inicial corresponde a la clase del estado $\langle\langle p_{entrada}, J1\rangle, [1, 1]\rangle + \langle\langle p_{libre}, M1\rangle, [0, 0]\rangle$. Hay que notar que sólo hay un estado el cual es una especialización de esta clase. El tiempo de habilitación mínimo (ET_{min}) del evento que corresponde el disparo de la transición t_1 es igual a 1. El tiempo modelo máximo (MT_{max}) es también 1. El único estado directamente alcanzable desde $\langle\langle p_{entrada}, J1\rangle, [1, 1]\rangle + \langle\langle p_{libre}, M1\rangle, [0, 0]\rangle$ es la clase de estado $\langle\langle p_{ocupado}, \langle M1, J1\rangle, [2, 4]\rangle\rangle$. Para esta clase de estado, el tiempo de habilitación mínimo (ET_{min}) del evento que corresponde al disparo de la transición t_2 es igual a 2. El tiempo modelo máximo (MT_{max}) es 4. La transición t_2 podrá disparar entre 2 y 4. Consecuentemente, la única clase de estado directamente alcanzable desde $\langle\langle p_{ocupado}, \langle M1, J1\rangle, [2, 4]\rangle\rangle$ es $\langle\langle p_{salida}, J1\rangle, [2, 4]\rangle + \langle\langle p_{libre}, M1\rangle, [2, 4]\rangle$.

2.4.3 Validez

Las definiciones alternativas de la sección (2.4.2) han sido proporcionadas por razones computacionales. Aunque, el cálculo del grafo de alcanzabilidad reducido sólo tiene sentido si este grafo puede ser usado para deducir propiedades del grafo de alcanzabilidad que representa el comportamiento dinámico de la RPCIT. Por lo tanto, se investigan las relaciones entre los dos grafos de alcanzabilidad.

Es sencillo ver que dos grafos de alcanzabilidad no son equivalentes. Aunque, un estado $s \in S$ y una clase de estado $\overline{s} \in \overline{S}$ parezca estar 'relacionado' si s es una especialización de $\overline{s}$. Recordando que $s \lhd \overline{s}$ si y sólo si cada token en s corresponde a exactamente un token en $\overline{s}$, tales que están en el mismo lugar, teniendo el mismo valor y la etiqueta de tiempo del token en s está en el intervalo de tiempo del token en $\overline{s}$.

Podemos ver que una RPCIT con $s_1, s_2 \in S$ tales que $s_1 \longrightarrow s_2$. Si $\overline{s}_1 \in \overline{S}$ y $s_1 \lhd \overline{s}_1$, entonces existe una clase de estado $s_2 \in \overline{S}$ tales que $\overline{s}_1 \Longrightarrow \overline{s}_2$ y $s_2 \lhd \overline{s}_2$.

Lo anterior nos dice que si un evento ocurre con cambios de estado s_1 a s_2, entonces existe un evento correspondiente el cual cambia de cualquier clase de estado $\overline{s}_1$ que 'cubre' a s_1, en una clase de estado $\overline{s}_2$ que 'cubre' a s_2 (ver figura 2.5). Por lo que la semántica alternativa es **válida**.

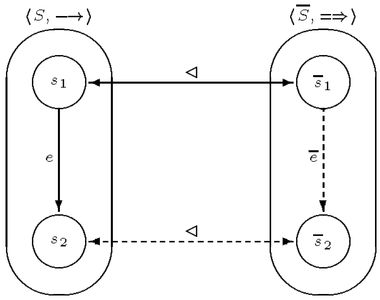

Figura 2.5: La propiedad de validez

Una implicación de lo mencionado es el hecho de que si $s_1 \lhd \overline{s}_1$ y existe una secuencia de disparos $s_1 \xrightarrow{e_1} s_2 \xrightarrow{e_2} s_3 \xrightarrow{e_3} \cdots \xrightarrow{e_{n-1}} s_n$, entonces existe una secuencia de disparos, $\overline{s}_1 \xrightarrow{\overline{e}_1} \overline{s}_2 \xrightarrow{\overline{e}_2} \overline{s}_3 \xrightarrow{\overline{e}_3} \cdots \xrightarrow{\overline{e}_{n-1}} \overline{s}_n$ tal que $s_n \lhd \overline{s}_n$. Para cualquier estado alcanzable desde s_1 hay una clase de estado relacionada, alcanzable desde $\overline{s}_1$.

Si comparamos el grafo de alcanzabilidad con el grafo de alcanzabilidad reducido de una RPCIT arbitraria, entonces podemos ver que todos los posibles cambios de estado en el grafo de alcanzabilidad son también posibles en en grafo de alcanzabilidad reducido. Hay que notar que lo contrario no sucede, dado que las 'dependencias' entre los tokens no son tomadas en cuenta.

Considerando por ejemplo el grafo de la figura 2.4, supongase que hay un token en el lugar p_1 con intervalo de tiempo $[0, 1]$ y los otros lugares vacíos. En este caso, la transición t se dispara entre el tiempo 0 y el tiempo 1, esto es, entre $ET_{min}(e)$ y $MT_{max}(s)$. Utilizando las definiciones alternativas, el siguiente estado de la RPCIT será con un token en p_2 (con intervalo $[1, 3]$) y un token en p_3 (con intervalo $[3, 5]$). Esto sugiere que es posible tener un token en p_2 con etiqueta de tiempo 1 y un token en p_3 con etiqueta de tiempo 5. Aunque esto no es posible, porque las etiquetas de tiempo están relacionadas (esto es, se producen al mismo tiempo). Entonces se dice que la semántica alternativa no es **completa**.

A pesar de la incompletez, la propiedad de validez nos permite resolver varias preguntas. Podemos **probar** que un sistema tiene un cierto conjunto de propiedades probándolo para el sistema de transiciones modificadas. Por ejemplo en ocasiones podemos usar el grafo de alcanzabilidad reducido para probar los acotamientos, la ausencia de trampas y deadlocks, etc.

El grafo de alcanzabiliad reducido también puede utilizarse para analizar el **rendimiento** del sistema modelado por una RPCIT. Nos referimos con rendimiento a las características, tales como: tiempos de respuesta, proporciones de ocupación, proporciones de transferencia, etc. El método TRSTM puede ser usado para calcular las cotas para medir el rendimiento. Aunque estas cotas son válidas (es decir, seguras) no tienen que ser tan estrechas como sea posible, esto es debido a que existen dependencias entre los tokens (incompletez).

Capítulo 3

Implantación del Prototipo

Para el desarrollo de esta tesis, se implantó un prototipo en SWI-Prolog que realiza Especificación Formal utilizando el modelo formal de Redes de Petri, el cual efectúa un análisis de sistemas modelados usando Redes de Petri Coloreadas con Intervalos de Tiempo (RPCIT).

3.1 Diseño del prototipo

Para especifar formalmente sistemas concurrentes empleando RPCIT's, se requiere modelar el sistema identificando los lugares, transiciones, intervalos de tiempo, valores, condiciones en las que se disparen las transiciones, etc. De esta manera se obtiene la Red de Petri que modela al sistema en base a sus características, esto se logra siguiendo la definición de RPCIT dada en (2.2).

Una vez identificados los elementos de la Red de Petri para el sistema modelado, se describen los nombres de los lugares, detallando los posibles valores que se puedan tener para cada uno de éstos; se describen también las transiciones, indicando sus lugares de entrada, de salida, los intervalos de tiempo y precisando las precondiciones –si las hay– para las cuales se disparan cada una de las transiciones. De la misma manera se requiere indicar los tokens iniciales para la Red de Petri, es decir, los recursos iniciales con los que cuenta el sistema modelado; de esta forma se consigue precisar el estado inicial en el que se encuentra el sistema y realizar el análisis respectivo.

Las descripciones anteriormente mencionadas deben hacerse con declaraciones de cláusulas de Prolog basadas en la sintáxis del Prolog de Edimburgo, esto con el fin de que se puedan detectar inconsistencias durante el análisis y conseguir resultados correctos. En Prolog estas definiciones se hacen utilizando cláusulas en forma de hechos, puesto que no se necesitan condiciones qué cumplir. Un ejemplo diseñado utilizando la definición (2.2) con base en la figura 3.1 es el siguiente:

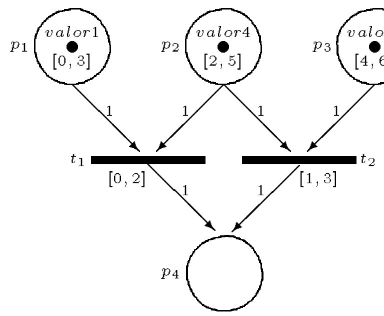

Figura 3.1: Ejemplo de una RPCIT

Tomando como referencia el inciso (a) de la definición (2.2) tenemos:

(a) Σ es un conjunto finito de tipos, llamados **conjuntos de colores**.

El conjunto de posibles valores (colores) que se le pueden asignar a cada uno de los lugares se define con el predicado **sigma** y como sus argumentos, el nombre del conjunto de color y entre paréntesis cuadrados una lista con los elementos del conjunto de colores. En el ejemplo de abajo tenemos los conjuntos $C1 = \{valor1, valor4\}$, $C2 = \{valor5\}$ y $C3 = \{valor2, valor3\}$.

```
sigma(conj1, [valor1, valor4]).
sigma(conj2, [valor5]).
sigma(conj3, [valor2, valor3]).
```

En seguida del inciso (b) de la misma definición se tiene:

(b) P es un conjunto finito de **lugares**.

Se definen los Lugares empleando el predicado **places** con una lista de los nombres de los lugares como argumento.

```
places([p1,p2,p3,p4]).
```

Para el siguiente elemento de la definición nos basamos en el inciso:

(c) T es un conjunto finito de **transiciones**.

Las Transiciones se detallan en forma de una lista utilizando: **trans** con una lista con los nombres de la transiciones.

```
trans([t1, t2]).
```

A continuación, con referencia a el inciso (d) se determina la relación entre el conjunto de lugares con los valores de sigma. Para simplificar el modelado del sistema no es necesario especificar concretamente el conjunto del inciso (e) puesto que la evaluación de la función de coloreo nos determina esta relación.

(d) C es una **función de coloreo**. Es definida de P a Σ, i.e. $C : P \rightarrow \Sigma$.

(e) $CT = \{(p,v)|p \in P \wedge v \in C(p)\}$ es el conjunto de todos los posibles **tokens coloreados**.

La relación de **color** entre el conjunto de lugares (**places**) y el de tipos (**sigma**), se forma haciendo que cada lugar tenga un conjunto de valores permitidos que están ligados a él.

```
color(p1, conj1).
color(p2, conj1).
color(p3, conj2).
color(p4, conj3).
```

$suma$	valor1	valor2	valor3	valor4	valor5
valor1	$valor3$	–	–	$valor2$	$valor2$
valor2	–	–	–	–	–
valor3	–	–	–	–	–
valor4	$valor2$	–	–	$valor2$	$valor3$
valor5	$valor2$	–	–	$valor3$	–

Figura 3.2: Tabla de valores para el operador *suma*

(f) G es una función de **guardia**. Está definida de T hacia expresiones booleanas. Si $t \in T$ se tiene:
$Tipo(G(t)) = I\!B \wedge Tipo(Var(G(t))) \subseteq \Sigma$.

Puesto que la función de transición nos determina una expresión donde las variables son los lugares y los valores que se sustituyen en éstas son los mismos colores, se necesitan declarar cláusulas que nos auxilien para determinar la evaluación de la función de transición tratándola como una expresión. Debido también a la versatilidad de las definiciones del prototipo para la especificación de sistemas, es necesario precisar cómo se están relacionando los valores involucrados dentro de los tipos definidos por **sigma** puesto que estos valores pueden especificar cualquier cosa en particular. Es por esta razón que se requiere utilizar operadores aditivos y multiplicativos para determinar la forma en que se relacionan estos valores.

Así, para evaluar la función de transición se definen las operaciones que se utilizarán entre los tokens para producir unos nuevos en el disparo de las transiciones utilizando los predicados **opadit** y **opmult** teniendo éstos como argumentos el símbolo del operador y su nombre.

Una vez realizado esto, hay que determinar la relación entre los operandos al aplicar cada uno de los operadores, es decir, hay que definir la **tabla** de cada uno de las operaciones. Por ejemplo, de la tabla en la figura 3.2 se puede tener: **suma** de $valor1$ con $valor4$ resulta $valor2$. La tabla puede formularse también de forma general para cualquier tipo de valores X, por ejemplo para obtener un duplicado de algún factor: se tiene **imagen** de X resulta X.

```
opadit(+, suma).
opmult(*, duplica).

tabla(suma, valor1, valor1, valor3).
tabla(suma, valor1, valor4, valor2).
tabla(suma, valor1, valor5, valor2).
tabla(suma, valor4, valor4, valor2).
tabla(suma, valor4, valor5, valor3).
tabla(imagen, X, X).
```

Por último el inciso (g) de la definición se utiliza como sigue:

(g) F es la **Función de Transición**. Es definida de T en funciones. Si $t \in T$ entonces:
$F(t) : CT_{MC} \not\rightarrow (CT \times INT)_{MC}$

Para definir los arcos de entrada y de salida de cada transición se coloca la función de transición **func** y como argumentos: el nombre de la transición, en seguida entre paréntesis cuadrados los lugares de entrada antecedidos por el número de tokens necesarios para la transición y precedidos por una variable de SWI-prolog (cualquier identificador que comience con mayúscula o con subraya), separados por un signo de + (formando los tokens coloreados de entrada), la variable que se coloca indica el posible color que puede llegar en el lugar. Este formato se utiliza de igual forma para los lugares de salida que se colocan en seguida (para formar los tokens coloreados de salida) y finalmente se coloca el intervalo de tiempo de la transición entre paréntesis cuadrados.

```
func(t1, [1, p1, X, +, 1, p2, _], [1, p4, X], [0, 2]).
func(t2, [1, p2, X, +, 1, p3, _], [1, p4, X], [1, 3]).
```

El formato de esta definición, auxiliado por las definiciones de operadores, permiten formular expresiones de forma implícita, puesto que este tipo de definiciones proporciona una expresión para los lugares de entrada a una transición y para los lugares de salida, los cuales se evalúan mientras se realiza el análisis de la RPCIT.

Como ejemplo, podemos tomar la siguiente meta en SWI-Prolog para indicar las marcas iniciales de la Redes de Petri colocando los tokens indicados en la figura 3.1 como sigue:

```
?- inicio([[1, p1, valor1, [0, 3]], [1, p2, valor4, [2, 5]],
[1, p3, valor5, [4, 6]]]).
```

Una vez introducida esta meta, el prototipo comenzará el análisis de la RPCIT utilizando los tokens iniciales y entregará los resultados en la pantalla al finalizar su estudio. En la siguiente sección se presenta la implantación del análisis de RPCIT's utilizando cláusulas de Prolog como auxiliares para la solución.

3.2 Implantación del Análisis de RPCIT

La implantación del prototipo se realizó siguiendo las definiciones del comportamiento dinámico de una RPCIT basados en la **Técnica de Reducción de Sistemas con Transiciones Modificadas** (TRSTM) para Reducir el grafo de alcanzabilidad y definiendo cada una con cláusulas de Prolog. Se incluyen algunas cláusulas adicionales a estas definiciones para el análisis de las precondiciones, con el fin de realizar un estudio amplio de la Red de Petri. Gracias a la naturaleza declarativa de Prolog se definen sencillamente estas cláusulas tomando la información de las definiciones de la Red de Petri arriba citadas.

Así utilizando la definición (2.1) se crean las cláusulas del Conjunto de Tiempos y de Intervalo como sigue:

*TS es el **conjunto de tiempos**, $TS = \{x \in \mathbb{R} | x \geq 0\}$, i.e. el conjunto de todos los reales no negativos, $INT = \{[y, z] \in TS \times TS | y \leq z\}$, representa el conjunto de todos los intervalos cerrados.*

```
conjunto_tiempo(Tiempo) :-
  Tiempo >= 0.

intervalo([Ci,Cd]) :-
  conjunto_tiempo(Ci),
```

```
  conjunto_tiempo(Cd),
  Ci =< Cd.
```

La cláusula **conjunto_tiempo** verifica que el $Tiempo$ sea mayor o igual a 0, para que se encuentre dentro del conjunto de tiempos. De esta manera, la cláusula **intervalo** verifica que se cumpla que el intervalo de tiempo esté dentro del conjunto de tiempos comprobando que la cota izquierda Ci sea menor que la cota derecha Cd para que cumpla con la definición.

De la definición (2.12) con la fórmula (2.1) es posible definir la cláusula **clases_estado** para verificar que los tokens especificados estén dentro del espacio de estados.

*Una clase de estados es definida como un multi-conjunto de tokens coloreados, cada uno llevando un intervalo de tiempo. $\overline{S}$ es el **espacio de estados:***

$$\overline{S} = (CT \times INT)_{MC}$$

```
clases_estado([[Coef, Lugar, Color, Interv]|Resto]) :-
  c_token(Lugar,Color),
  intervalo(Interv),
  Coef >= 1,
  clases_estado(Resto).
clases_estado([]).
```

Esta cláusula verifica que los Valores de los tokens así como sus intervalos de tiempo sean los apropiados para el multi-conjunto de tokens que le sea especificado.

Tomando de la definición (2.12) la fórmula (2.2) se tiene la implantación de los **eventos**, que en Prolog se define como la cláusula:

*Un **evento** es un triple $\langle t, d_e, d_s \rangle$, que representa el posible disparo de la transición t mientras elimina los tokens especificados por el multi-conjunto d_e y agregando tokens especificados por el multi-conjunto d_s. $\overline{E}$ es el **conjunto de eventos:***

$$\overline{E} = T \times (CT \times INT)_{MC} \times (CT \times INT)_{MC}$$

```
eventos([Tran|Resto]) :-
  func(Tran, Forma_E, _, _),
  tok_entrada(Forma_E,Entradas),
  clases_estado(Entradas),
  assert(evento(Tran,Entradas,_)),
  eventos(Resto).
eventos([_|Resto]) :-
  eventos(Resto).
eventos([]).
```

La cláusula **eventos** se encarga de armar un conjunto de eventos en cláusulas denominadas **evento** tomando la lista de transiciones, de forma que **Tran** sea una transición, **Entradas** contenga los tokens de sus entradas y en en el tercer argumento se formen los tokens de sus salidas. Hasta este nivel estos últimos tokens no se encuentran determinados puesto que la transición **Tran** no se ha disparado aún. Para formar los eventos se toma de la cláusula **func** la forma de los tokens de entrada para la transición en **Forma_E**. Los tokens de entrada en **Entradas** se arman verificando cada uno de los lugares de entrada de la transición **Tran** utilizando la cláusula **tok_entrada** y enseguida se comprueba que cada uno de esos tokens correspondan a una clase de estado con la cláusula **clases_estado**.

En dado caso que la transición no tenga suficientes tokens de entrada, la cláusula **clases_estado** fallará, por lo que no se le toma en cuenta y continúa con el resto de las transiciones.

Una vez armado el evento se agrega la cláusula formada. La cláusula **eventos** examina recursivamente todos los eventos posibles en la RPCIT tomando en cuenta cada una de las transiciones con los tokens que se encuentren en sus entradas. El conjunto de cláusulas formadas por **eventos** serán utilizadas posteriormente para determinar los eventos habilitados.

Con la definición (2.13) se buscan los eventos que estén habilitados, esta cláusula se define en Prolog como sigue:

Un evento $\langle t, d_e, d_s \rangle \in \overline{E}$ es ***habilitado*** *en la clase de estado $s \in \overline{S}$ si y sólo si:*

(a) $d_e \leq s$

(b) $M(d_e) \in dom(F(t))$

(c) $d_s = F(t)(M(d_e))$

```
e_habilitados :-
  retract(evento(Tran,[[Coef,Lugar,Valor,Interv]|Resto],
  Salidas)),
  func(Tran,Forma_E,Forma_S,Interv_S),
  estado([Coef1,Lugar,Valor,Interv]),
  Coef =< Coef1, !,
  marca([[Coef,Lugar,Valor,Interv]|Resto],Marca,Expresion),
  pert_dom(Marca,Forma_E),
  eval([Expresion, =, Valores]),
  ftrans(Salidas,Forma_S,Interv_S,Valores),
  agrega(event_habilitados,
  [Tran,[[Coef,Lugar,Valor,Interv]|Resto],Salidas]),
  e_habilitados.
e_habilitados.
```

Esta cláusula toma el conjunto de eventos y analiza cuales de ellos cumplen con la definición de estar habilitados. Primero se toma la información de la forma de los tokens de entrada y salida, así como su intervalo de tiempo de la cláusula **func**; se verifica que los tokens de entrada que se encuentran en la lista $[[Coef, Lugar, Valor, Interv]|Resto]$ se hallen presentes en estado actual. La transición se habilita cuando tiene suficientes tokens en sus entradas, esto se logra comparando las marcas de los tokens de entrada con el dominio de la transición. En este momento se evalúa la expresión formada de la transición **Tran**, utilizando la cláusula **eval**, para determinar los valores que llevarán los tokens de salida. Finalmente se producen los tokens de salida en **Salidas** que es determinado por la función de transición con la cláusula **ftrans**.

Recursivamente se forma una lista con los eventos habilitados listos para ser disparados. Cabe aclarar que las expresiones se arman en el momento de crear la **marca** de los tokens de entrada; esto proporciona una forma de determinar los valores para los tokens de salida.

Una vez teniendo la lista de eventos habilitados, se efectúa el disparo de las transiciones que contengan estos eventos. Esto se logra siguiendo la definición (2.16) de disparo, la cual se define de la siguiente manera:

Si $\langle t, d_e, d_s\rangle$ ocurre en la clase de estado s_1, entonces la red cambia a la clase de estado $s_2 \in \overline{S}$, definido por:

$$s_2 = (s_1 - d_e) + \overline{SC}(d_s, [ET_{min}(\langle t, d_e, d_s\rangle), MT_{max}(s_1)])$$

```
disparo(Eventos, [[Tran, Tokens_Entrada, Tokens_Salida]|
  Resto]) :-
  et_min([Tran, Tokens_Entrada, Tokens_Salida], Ci),
  mt_max(Eventos, Cd),
  escala(Tokens_Salida, [Ci, Cd], Tokens_Nuevos),
  cambia(Tokens_Entrada,Tokens_Nuevos),
  disparo(Eventos,Resto).
disparo(_, []).
```

La cláusula **disparo** realiza las operaciones con los intervalos de tiempo de los tokens, para determinar los intervalos de tiempo que llevarán los tokens de salida en **Tokens_Nuevos**. Lo primero es calcular el Tiempo de Habilitación Mínimo con **et_min**, en seguida se calcula el Tiempo Modelo Máximo empleando **mt_max** y se realiza un escalamiento para conseguir el intervalo de tiempo para los tokens en **Tokens_Nuevos** utilizando **escala**. Una vez formados los tokens de salida, se consumen –si es posible– cada uno de los tokens de las entradas en **Tokens_Entrada** de las transiciones habilitadas y se producen los **Tokens_Nuevos** en sus lugares de salida. Puede suceder que algunos tokens de entrada de las transiciones en los eventos habilitados no sean consumidos, puesto que puede haber eventos habilitados concurrentemente que utilicen tokens en común; en este caso la primera transición que se analice, será la que pueda tomar los tokens de sus entradas.

Estas tres últimas cláusulas principales se ejecutan constantemente para analizar el comportamiento dinámico de la RPCIT definida, utilizando la información de los tokens proporcionados al inicio. Las cláusulas auxiliares están definidas de igual manera siguiendo las definiciones tomadas de [vdA91] y [Jen94].

Se puede observar que para realizar el disparo de transiciones en los eventos habilitados, se requiere de calcular el tiempo de habilitación mínimo y el tiempo modelo máximo. La cláusula **et_min** calcula el máximo de las cotas izquierdas que llevan todos los tokens de entrada de un evento habilitado. La cláusula **et_max** calcula el máximo de las cotas derechas que llevan los tokens de entrada en un evento habilitado. La cláusula **mt_max** calcula el mínimo de los tiempos de habilitación máximo para un evento habilitado.

Tomando como referencia la definición (2.14), estas cláusulas se definen de la siguiente forma:

*El **tiempo de habilitación mínimo y máximo** de un evento* $\langle t, d_e, d_s\rangle \in \overline{E}$ *están definidos como sigue:*

$$ET_{min}(\langle t, d_e, d_s\rangle) = \max_{\langle\langle p,v\rangle,[y,z]\rangle \in d_e} y$$

$$ET_{max}(\langle t, d_e, d_s\rangle) = \max_{\langle\langle p,v\rangle,[y,z]\rangle \in d_e} z$$

```
et_min([_, Tokens_Entrada, _], Res) :-
        maxCI(Tokens_Entrada, 0, Res).

et_max([_, Tokens_Entrada, _], Res) :-
        maxCD(Tokens_Entrada, 0, Res).
```

De la misma forma se obtiene el tiempo modelo máximo para aquellos eventos habilitados en el estado s.

$$MT_{max}(s) = min\{ET_{max}(e) | e \in \overline{E} \text{ y } e \text{ es habilitado en } s\}.$$

```
mt_max(Eventos, Res) :-
        minET(Eventos, 999999, Res).
```

La función que escala los intervalos de tiempo en un multi-conjunto de tokens coloreados con intervalos de tiempo se define como una cláusula de Prolog según la definición (2.15), de la siguiente forma.

La función $\overline{SC} \in ((CT \times INT)_{MC} \times INT) \rightarrow (CT \times INT)_{MC}$ *escala los intervalos de tiempo en un multi-conjunto de tokens coloreados con tiempo. Para* $d \in (CT \times INT)_{MC}$ *y* $[y, z] \in INT$*, tenemos:*

$$\overline{SC}(d, [y, z]) = \sum_{\langle\langle p,v\rangle,[y',z']\rangle \in CT \times INT} d(\langle\langle p, v\rangle, [y', z']\rangle)\langle\langle p, v\rangle, [x' + x, y' + y]\rangle$$

```
escala([Coef, Lugar, Color, [Cia,Cda], +|Resto1], [Ci,Cd],
  [Coef, Lugar, Color, [Cin,Cdn], +|Resto2]) :-
  Cin is Ci + Cia,
  Cdn is Cd + Cda,
  escala(Resto1,[Ci,Cd],Resto2).
escala([Coef, Lugar, Color, [Cia,Cda]], [Ci,Cd],
[Coef, Lugar, Color, [Cin,Cdn]]) :-
  Cin is Ci + Cia,
  Cdn is Cd + Cda.
```

Considerando como ejemplo de análisis a red en la figura 3.1, al inicio hay un token en p_1 con un intervalo del $[0,3]$, un token en p_2 con un intervalo de $[2,5]$ y un token en p_3 con un intervalo de $[4,6]$. Se puede ver que esta clase de estado $\overline{s}$ corresponde a un número infinito de estados en el grafo de alcanzabilidad original, para una instancia del estado con un token en p_1 con una etiqueta de tiempo de 2.4, un token en p_2 con una etiqueta de tiempo π y un token en p_3 con una etiqueta de tiempo de 31/6. Las dos transiciones están habilitadas, si la transición t_1 se dispara primero los tokens en p_1 y p_2 son consumidos, si la transición t_2 se dispara primero los tokens en p_2 y p_3 son consumidos. El tiempo de habilitación de t_1 está entre 2 ($ET_{min}(t_1)$) y 5 ($ET_{max}(t_1)$), el tiempo de habilitación de t_2 está entre 4 ($ET_{min}(t_2)$) y 6 ($ET_{max}(t_2)$). La transición con el menor tiempo de habilitación se disparará primero. Como los intervalos asociados a los tiempos de habilitación de las transiciones (i.e. $[2,5]$ y $[4,6]$) están empalmados, no está determinado cuando t_1 o t_2 dispara primero. Sin embargo, la cota superior del tiempo de transición ($MT_{max}(\overline{s})$) es igual a 5, i.e. una transición disparará en un tiempo de 5 o antes. Si t_1 se dispara, será entre 2 ($ET_{min}(t_1)$) y 5 ($MT_{max}(\overline{s})$). Si t_2 se dispara, será entre 4 ($ET_{min}(t_2)$) y 5 ($MT_{max}(\overline{s})$). En ambos casos un token es producido en el lugar p_4. Existen dos posibles estados terminales: uno con un token en p_3 y en p_4 y otro con un token en p_1 y en p_4. En el primer caso el intervalo de tiempo del token en p_4 es $[2,7]$, puesto que el intervalo de tiempo de un token producido por t_1 es $[0,2]$. En el segundo caso el intervalo de tiempo del token en p_4 es $[5,8]$.

Al final del análisis que realiza el prototipo, podemos conocer los estados finales en que se encuentre el sistema, de esta manera se pueden detectar **deadlocks**, lugares inalcanzables y transiciones que no se hayan disparado.

3.3 Código Fuente del Prototipo

El prototipo en SWI-Prolog que efectúa el análisis de una Red de Petri Coloreada con Intervalos de Tiempo, utilizando la Técnica de Reducción de Sistemas con Transiciones Modificadas se enlista en esta sección.

```
/* Definiciones auxiliares */

membermc([Elemento | Resto], Conjunto) :-
member(Elemento, Conjunto),
membermc(Resto, Conjunto).
membermc([], _).
membermc(Elemento, Conjunto) :-
member(Elemento, Conjunto).

memberc(Elemento, Nombre) :-
conj(Nombre, Conjunto),
member(Elemento, Conjunto).

multiconjunto([[Coef, _, _, _] | _], Coef).

agrega([Coef, Lugar, Color, Interv]) :-
retract(estado([Anterior, Lugar, Color, Interv])),
Nuevo is Coef + Anterior,
assert(estado([Nuevo, Lugar, Color, Interv])).
agrega(Estado)  :-
assert(estado(Estado)).

agrega(event_habilitados, Evento) :-
retract(event_habilitados([Lista | Anterior])),
append([Lista | Anterior], [Evento], Nuevo),
assert(event_habilitados(Nuevo)).
agrega(event_habilitados, Evento) :-
assert(event_habilitados([Evento])).

agrega(estado, [Coef, Lugar, Color, Interv, + | Resto]) :-
agrega([Coef, Lugar, Color, Interv]),
agrega(estado, Resto).
agrega(estado, Estado) :-
agrega(Estado).

consume([[Coef_1, Lugar, Color, Interv] | Resto]) :-
clause(estado([Coef_2, Lugar, Color, Interv]), _),
Coef_2 > 1,
Coef_n is Coef_2 - Coef_1,
retract(estado([Coef_2, Lugar, Color, Interv])),
agrega([Coef_n, Lugar, Color, Interv]),
```

```
consume(Resto).
consume([[1, Lugar, Color, Interv] | Resto]) :-
retract(estado([1, Lugar, Color, Interv])),
consume(Resto).
consume([]).

cambia(Anterior, Nuevo) :-
consume(Anterior),
agrega(estado, Nuevo).
cambia(_, _).

maxCI([[_, _, _, [Ci, _]] | Resto], Cuanto, Maxima) :-
Max is max(Ci,Cuanto),
maxCI(Resto,Max,Maxima).
maxCI([],Maxima,Maxima).

maxCD([[_, _, _, [_, Cd]] | Resto], Cuanto, Maxima) :-
Max is max(Cd,Cuanto),
maxCD(Resto,Max,Maxima).
maxCD([],Maxima,Maxima).

tok_entrada([Coef, Lugar, _, + | Resto],
        [[Coef, Lugar, ValorE, Interv]|RestoE]) :-
clause(estado([_, Lugar, ValorE, Interv]), _),
tok_entrada(Resto,RestoE).
tok_entrada([Coef, Lugar, _], [[Coef, Lugar, ValorE, Interv]]) :-
clause(estado([Coef1, Lugar, ValorE, Interv]), _),
        Coef =< Coef1.
tok_entrada([_, Lugar, _, + | _], ['no token']) :-
not clause(estado([_, Lugar, _, _]), _).
tok_entrada([_, Lugar, _], ['no token']) :-
not clause(estado([_, Lugar, _, _]), _).

minET([Evento|Resto], Cuanto, Minima) :-
et_max(Evento,ETmax),
Min is min(ETmax,Cuanto),
        minET(Resto,Min,Minima).
minET([],Minima,Minima).

/* Definiciones para precondiciones */

eval([A ,=, Res]) :- !,
expr(A,Res).

eval([A ,==, B]) :-
expr(A, Res), !,
expr(B, Res).
```

```
eval([A ,<>, B]) :- !,
not eval([A,==,B]).

eval([A ,<, B]) :-
expr(A,Res_A),
expr(B,Res_B), !,
Res_A < Res_B.

eval([A ,>, B]) :-
expr(A,Res_A),
expr(B,Res_B), !,
Res_A > Res_B.

eval([A ,=<, B]) :-
expr(A,Res_A),
expr(B,Res_B), !,
Res_A =< Res_B.

eval([A ,>=, B]) :-
expr(A,Res_A),
expr(B,Res_B), !,
Res_A >= Res_B.

eval([A ,and, B]) :-
eval(A), !,
eval(B).

eval([A ,or, _]) :-
eval(A).
eval([_ ,or, B]) :-
eval(B).

eval([not, A]) :-
not eval(A).

eval([]).

eval([true]).

expr([A, Operador, B], Res) :- !,
        opadit(Operador,Funcion),
        ter(A,Valor_A), !,
expr(B,Valor_B), !,
tabla(Funcion,Valor_A,Valor_B,Res).
expr(A, Res) :-
ter(A,Res).
```

```
ter([A, Operador, B],Res) :-
opmult(Operador,Funcion),
fac(A,Valor_A),
ter(B,Valor_B),
tabla(Funcion,Valor_A,Valor_B,Res).
ter(A,Res) :-
fac(A,Res).


fac([_,A], Res) :-
opmult(_,Funcion),
tabla(Funcion,A,Res).
fac([_,Res], Res).

fac([[_, A]], Res) :-
opmult(_,Funcion),
tabla(Funcion,A,Res).
fac([[_, Res]], Res).

fac(A,Res) :-
expr(A,Res).
fac(A,Res) :-
ter(A,Res).

sustituye(Variable, Valor, [[Variable ,=, Valor]| _]).
sustituye(Variable, Valor, [[_ ,=, _]|Resto]) :-
sustituye(Variable,Valor,Resto).
sustituye(Valor,Valor,[]).

/* Definiciones para las RPCIT's */

conjunto_tiempo(Tiempo) :-
Tiempo >= 0.

intervalo([Ci,Cd]) :-
conjunto_tiempo(Ci),
conjunto_tiempo(Cd),
Ci =< Cd.

pert_dom([Coef1, Lugar, Color, +|Resto1],
        [Coef2, Lugar, Color, +|Resto2]) :-
c_token(Lugar,Color),
Coef2 =< Coef1,
pert_dom(Resto1, [Coef2, Lugar, Color, +|Resto2]).
pert_dom([Coef1, Lugar, Color, +|Resto1], [_, _, _, +|Resto2]) :-
pert_dom([Coef1, Lugar, Color, +|Resto1], Resto2),
```

```
pert_dom(Resto1,[Coef1, Lugar, Color, +|Resto2]).
pert_dom([Coef1, Lugar, Color], [_, _, _, +|Resto2]) :-
pert_dom([Coef1, Lugar, Color], Resto2).
pert_dom([Coef1, Lugar, Color], [Coef2, Lugar, Color]) :-
c_token(Lugar,Color),
Coef2 =< Coef1.

c_token(Lugar, Valor) :-
        places(Lugares),
        member(Lugar,Lugares),
color(Lugar,Color),
sigma(Color,Conjunto),
membermc(Valor,Conjunto).

ftrans([Coef, Lugar, Valor, [Ci, Cd], +|Resto1],
       [Coef, Lugar, _, +|Resto2], [Ci, Cd], [Valor]) :-
        ftrans(Resto1,Resto2,[Ci,Cd],Valor).
ftrans([Coef, Lugar, Valor, [Ci, Cd], +|Resto1],
       [Coef, Lugar, _, +|Resto2], [Ci, Cd], [Valor|Valores]) :-
ftrans(Resto1,Resto2,[Ci,Cd],Valores).
ftrans([Coef, Lugar, Valor, [Ci, Cd], +|Resto1],
       [Coef, Lugar, _, +|Resto2], [Ci, Cd], Valor) :-
ftrans(Resto1,Resto2,[Ci,Cd], Valor).
ftrans([Coef, Lugar, Valor, [Ci, Cd]], [Coef, Lugar, _],
       [Ci, Cd], [Valor]).
ftrans([Coef, Lugar, Valor, [Ci, Cd]], [Coef, Lugar, _],
       [Ci, Cd], Valor).

/* Definiciones para el comportamiento dinamico de la RPCIT */

inicio(Marca) :-
        abolish(estado,1),
marca_inicial(Marca),
seq_disparo.

marca_inicial([[Coef, Lugar, Color, Interv]|Resto]) :-
c_token(Lugar,Color),
intervalo(Interv),
Coef >= 1,
agrega([Coef, Lugar, Color, Interv]),
marca_inicial(Resto).
marca_inicial([]).
marca_inicial(_) :-
error('Componentes de marca no validos').

marca([[Coef1, Lugar, Color, Interv]], [Coef, Lugar, Color],
       [[Lugar, Color]]) :-
```

```
clause(estado([Coef,Lugar,Color,Interv]),_),
c_token(Lugar,Color),
Coef1 >= 1.
marca([[Coef1, Lugar, Color, Interv]|Resto1],
      [Coef, Lugar, Color, +|Resto2],
      [[Lugar, Color],+|[Resto3]]) :-
clause(estado([Coef,Lugar,Color,Interv]),_),
c_token(Lugar,Color),
Coef1 >= 1,
marca(Resto1,Resto2,Resto3).

clases_estado([[Coef, Lugar, Color, Interv]|Resto]) :-
c_token(Lugar,Color),
intervalo(Interv),
Coef >= 1,
clases_estado(Resto).
clases_estado([]).

eventos([Tran|Resto]) :-
func(Tran, Forma_E, _, _),
tok_entrada(Forma_E,Entradas),
clases_estado(Entradas),
assert(evento(Tran,Entradas,_)),
eventos(Resto).
eventos([_|Resto]) :-
eventos(Resto).
eventos([]).

e_habilitados :-
        retract(evento(Tran,[[Coef,Lugar,Valor,Interv]
        |Resto],Salidas)),
func(Tran,Forma_E,Forma_S,Interv_S),
estado([Coef1,Lugar,Valor,Interv]),
Coef =< Coef1, !,
        marca([[Coef,Lugar,Valor,Interv]|Resto],Marca,
        Expresion),
pert_dom(Marca,Forma_E),
        eval([Expresion, =, Valores]),
        ftrans(Salidas,Forma_S,Interv_S,Valores),
        agrega(event_habilitados,[Tran,
        [[Coef,Lugar,Valor,Interv]|Resto],Salidas]),
e_habilitados.
e_habilitados.

et_min([_, Tokens_Entrada, _], Res) :-
maxCI(Tokens_Entrada, 0, Res).
```

```
et_max([_, Tokens_Entrada, _], Res) :-
maxCD(Tokens_Entrada, 0, Res).

mt_max(Eventos, Res) :-
        minET(Eventos, 999999, Res).

escala([Coef, Lugar, Color, [Cia,Cda], +|Resto1], [Ci,Cd],
       [Coef, Lugar, Color, [Cin,Cdn], +|Resto2]) :-
Cin is Ci + Cia,
Cdn is Cd + Cda,
escala(Resto1,[Ci,Cd],Resto2).
escala([Coef, Lugar, Color, [Cia,Cda]], [Ci,Cd],
       [Coef, Lugar, Color, [Cin,Cdn]]) :-
Cin is Ci + Cia,
Cdn is Cd + Cda.

seq_disparo :-
listing(estado),
trans(Transiciones),
eventos(Transiciones),
listing(evento),
e_habilitados,
listing(event_habilitados),
retract(event_habilitados(Eventos)),
disparo(Eventos, Eventos),
        abolish(evento,3),
seq_disparo.
seq_disparo.

disparo(Eventos, [[Tran, Tokens_Entrada, Tokens_Salida]|Resto]) :-
et_min([Tran, Tokens_Entrada, Tokens_Salida], Ci),
mt_max(Eventos, Cd),
escala(Tokens_Salida, [Ci, Cd], Tokens_Nuevos),
cambia(Tokens_Entrada,Tokens_Nuevos),
disparo(Eventos,Resto).
disparo(_, []).

/* disparo(Eventos, [Tran, Tokens_Entrada, Tokens_Salida]) :-
et_min([Tran, Tokens_Entrada, Tokens_Salida], Ci),
mt_max(Eventos, Cd),
escala(Tokens_Salida, [Ci, Cd], Tokens_Nuevos),
cambia(Tokens_Entrada,Tokens_Nuevos). */
```

Capítulo 4

Aplicaciones Prácticas

En este capítulo se presentarán algunos ejemplos de Redes de Petri con los que se probó el prototipo, así como el análisis que se realiza durante la ejecución del mismo al inspeccionar el grafo de alcanzabilidad reducido para las RPCIT's.

4.1 Ejemplo 1: Una RPCIT simple

Para mostrar el análisis completo que realiza el prototipo, regresamos al ejemplo planteado en la figura 3.1 que fue definido en el capítulo anterior. Para comenzar el análisis de una Red de Petri ya definida tomando en cuenta las consideraciones del capítulo anterior, se requiere colocar los tokens iniciales con los que la red empezará su 'ejecución'; esto se hace introduciendo una meta en SWI-Prolog donde se especifiquen precisamente estos tokens iniciales, por ejemplo:

```
?- inicio([[1, p1, valor1, [0, 3]], [1, p2, valor4, [2, 5]],
[1, p3, valor5, [4, 6]]]).
```

Esta meta hace un empatamiento con la cláusula **inicio** del prototipo, la cual elimina primero los estados que tenga para prevenir que no se trate de un análisis anterior, puesto que lo primero que hace es generar los primeros estados para determinar las marcas de los tokens para ser analizados, esto se logra con la cláusula **marca_inicial**.

De esta manera se toman recursivamente cada uno de los tokens de la forma [$Coef$, $Lugar$, $Color$, $Interv$], en este caso se tiene una lista con estos tokens de la forma [[1, $p1$, $valor1$, [0, 3]], [1, $p2$, $valor4$, [2, 5]], [1, $p3$, $valor5$, [4, 6]]]. La cáusula **marca_inicial** verifica que el par $(Lugar, Color)$ forme un token coloreado, que $Interv$ sea un intervalo de tiempo y que al menos haya un token con estas características. Si cumple con esto, entonces es agregado el nuevo estado. Para el ejemplo, **marca_inicial** agrega los siguientes estados:

```
estado([1, p1, valor1, [0, 3]]).
estado([1, p2, valor4, [2, 5]]).
estado([1, p3, valor5, [4, 6]]).
```

El siguiente paso es buscar eventos que estén habilitados para ir formando las secuencias de disparo hasta que no existan más transiciones que puedan ser disparadas. Para realizar este análisis primero se arman los eventos que pueden existir en un momento dado, de ellos, se busca cuáles de ellos están habilitados y se realiza el disparo de las transiciones en los eventos habilitados.

Para buscar los eventos de la red de Petri en 'ejecución', se analiza cada una de las transiciones requiriendo que tengan los tokens necesarios en sus entradas, esto se hace revisando el conjunto de estados que está presente en un momento determinado. De esta forma la cláusula **eventos** recorre la lista de las transiciones y se buscan los tokens de entrada para la transición de acuerdo a los estados, que proporcionan información sobre la distribución de los tokens en la red. Antes de agregar el evento, se verifica que los tokens de sus entradas sean una clase de estados, comprobando la autenticidad de los colores del token y su intervalo de tiempo.

La red de Petri del ejemplo en cuestión tiene las transiciones t_1 y t_2 con lo que la cláusula **Eventos** se comporta de la siguiente manera: Al principio empata con la primera transición (t_1), se toma el formato de los tokens de entrada usando el hecho en **func** para la transición. La cláusula **tok_entrada** trata de hacer empatamientos con los estados a manera de encontrar los tokens de entrada en los lugares especificados por $Forma\_E$. Y la cláusula **clases_estado** verifica que sea una clase de estados permitida para después agregar el nuevo evento, después continúa recursivamente con la transición t_2 para continuar con observaciones similares sobre sus tokens de entrada. Una vez concluido este análisis, el nuevo evento es agregado. Debe notarse que los tokens para las salidas no han sido aún determinados puesto que la transición no ha disparado. Para el ejemplo la cláusula **eventos** encuentra los siguientes eventos para las transiciones t_1 y t_2:

```
evento(t1, [[1, p1, valor1, [0, 3]], [1, p2, valor4, [2, 5]]], A).
evento(t2, [[1, p2, valor4, [2, 5]], [1, p3, valor5, [4, 6]]], A).
```

Una vez encontrados los eventos, habrá que determinar cuáles de ellos están habilitados, para esto la cláusula **e_habilitados** se encarga del análisis respectivo. Como primer paso, se toma el evento de una transición $Tran$ y se ve si está dentro de el estado correspondiente; se toma la marca de los tokens de entrada con la cláusula **marca** para ver si pertenece al dominio de la transición $Tran$, utilizando para ello **pert_dom**. La cláusula **marca**, también se encarga de armar la expresión del arco entre los lugares de entrada y la transición. A continuación se evalúa la expresión de arco con **eval** que nos proporciona los colores de los tokens de salida en $Valores$ y se evalúa la función de transición **ftrans** que obtiene finalmente los tokens de salida para $Tran$ con su intervalo de tiempo respectivo. Una vez precisado que el evento si está habilitado, se agrega a la lista. Para el ejemplo, la **e_habilitados** encuentra los siguientes eventos habilitados en la cláusula siguiente:

```
event_habilitados(
[[t1, [[1,p1,valor1,[0,3]], [1,p2,valor4,[2,5]]], [1,p4,valor2,[0,2]]],
 [t2, [[1,p2,valor4,[2,5]], [1,p3,valor5,[4,6]]], [1,p4,valor3,[1,3]]]]).
```

En este momento ya se puede hacer el **disparo** de las transiciones que estén habilitadas, para ello, se requiere primero calcular el mínimo de las transiciones habilitadas, usando **et_min** y el tiempo modelo máximo del estado con **mt_max**, que son las cotas $[Ci, Cd]$ con las que se van a escalar los intervalos de los tokens de salida. En este caso Ci es 2 y Cd es 5. De esta forma, antes de efectuar el disparo de una transición t_1, se tiene en $Tokens\_Salida$ los tokens de salida para la transición, esto es [1, $p4$, $valor2$, [0, 2]] que se escalan con 2 y 5 para formar los tokens que estarán en los lugares de salida en $Tokens\_Nuevos$, esto es [1, $p4$, $valor2$, [2, 7]].

El siguiente paso es consumir -si es posible- los $Tokens\_Entrada$ de la transición y producir los $Tokens\_Nuevos$ en sus salidas. Cuando se tienen transiciones habilitadas concurrentemente, se disparará primero, la transición que se haya evaluado primero, y así sucesivamente. En este caso, las transiciones t_1 y t_2 están habilitadas concurrentemente, pero no pueden dispararse ambas, puesto que el token en p_2 será consumido primero por t_1 y la transición t_2 no podrá tomarlo, por lo que ya no puede dispararse. De esta forma los estados finales después del disparo de t_1 son los siguientes:

```
estado([1, p3, valor5, [4, 6]]).
estado([1, p4, valor2, [2, 7]]).
```

Una vez efectuado el disparo el prototipo buscará los siguientes eventos tomando en cuenta los nuevos estados que acaba de generar; como se puede notar, este estado no proporciona los tokens suficientes en las entradas de las transiciones t_1 y t_2 para hacerlas elegibles para disparo, por lo que ya no podrá encontrar eventos habilitados y el análisis será detenido.

4.2 Ejemplo 2: Sistema simple de computadora

Otro ejemplo práctico se trata de un sistema simple de computadora compuesto de varios CPU's y un número de discos; cada trabajo que procesa este sistema, visita un CPU y un disco. Inicialmente hay tres procesadores (CPU's) paralelos y tres discos. Cada procesador tiene un tiempo de servicio entre 50 y 60 milisegundos. El tiempo de servicio de entrada y salida de el disco 1 está entre 95 y 100 milisegundos. El disco 2 tiene un tiempo de servicio entre 100 y 115 milisegundos y el disco 3 tiene un tiempo de servicio entre 95 y 110 milisegundos. Para cada trabajo se tiene que alojar algo de memoria. Para cambiar un trabajo hacia la memoria se requiere de un tiempo (entre 20 y 25 milisegundos). Los trabajos que son introducidos se llaman 'activos'. Los trabajos activos visitan un CPU y un disco, entonces ellos removidos para permitir la entrada a otros trabajos. El tiempo

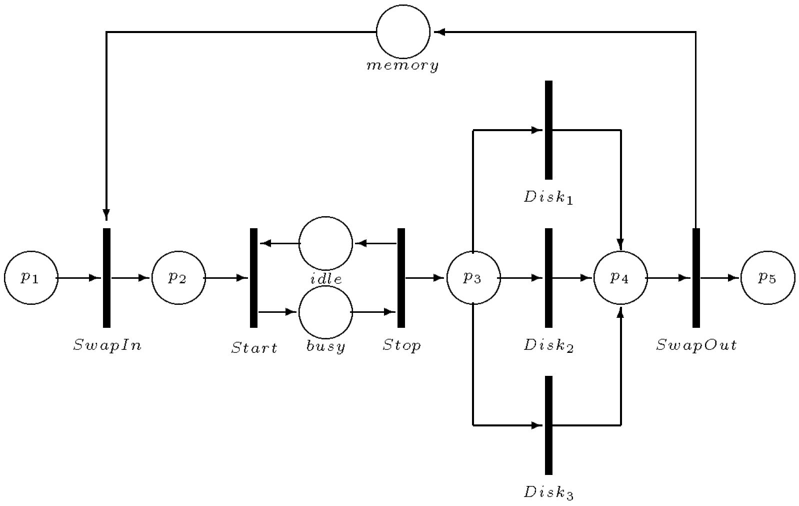

Figura 4.1: Sistema simple de computadora.

requierido para desalojar la memoria (tiempo de salida) está entre 40 y 45 milisegundos. El número máximo de trabajos activos es 10.

La figura 4.1 muestra la RPCIT que modela este sistema. Los trabajos llegan al sistema a través del lugar **p1** y salen por el lugar **p5**. Las transiciones **SwapIn** y **SwapOut** se encargan del manejo de la memoria. Hay tres discos modelados por las transiciones **Disk1**, **Disk2** y **Disk3**. Inicialmente hay 10 tokens en el lugar **memory** (El número máximo de trabajos es 10) y el lugar **idle** contiene 3 tokens iniciales, cada uno representando un procesador. Se puede notar que la transición **Start** se habilita si un trabajo está esperando en el lugar **p2** y un procesador está libre (i.e. hay un token en **idle**). La transición **Stop** representa la terminación de la ejecución de un trabajo en un procesador, i.e. la transición **Stop** desaloja un procesador y manda el trabajo a alguno de los tres discos.

Este sistema de computadora simple se define para el prototipo con las siguientes cláusulas en forma de hechos:

```
sigma(ausente, [v]).

places([p1,p2,idle,busy,p3,p4,p5,memory]).

trans([swapIn, start, stop, disk1, disk2, disk3, swapOut]).

color(p1,     ausente).
color(p2,     ausente).
color(idle,   ausente).
```

```
color(busy,   ausente).
color(p3,     ausente).
color(p4,     ausente).
color(p5,     ausente).
color(memory, ausente).

oparit(+, suma).
opmult(*, mult).

tabla(suma, X, X, X).
tabla(mult, X, X).

func(swapIn, [1, p1, X, +, 1, memory, X], [1, p2, X], [20, 25]).
func(start,  [1, p2, X, +, 1, idle, X], [1, busy, X], [50, 60]).
func(stop,   [1, busy, X], [1, idle, X, +, 1, p3, X], [50, 60]).
func(disk1,  [1, p3, X], [1, p4, X],                  [95, 100]).
func(disk2,  [1, p3, X], [1, p4, X],                  [100,115]).
func(disk3,  [1, p3, X], [1, p4, X],                  [95, 110]).
func(swapOut,[1, p4, X], [1, memory, X, +, 1, p5, X], [40, 45]).
```

De forma similar al ejemplo anterior, para colocar los tokens iniciales en **memory**, en **idle** y la carga inicial en **p1** se introduce la siguiente meta en SWI-Prolog para que comience con el análisis respectivo:

```
?- inicio([[3,p1,v,[0,0]], [3,idle,v,[50,60]], [10,memory,v,[0,160]]]).
```

Tomando en cuenta que los trabajos llegan al sistema a través de **p1**, y salen por **p5**, la carga inicial de los 3 trabajos produce los siguientes estados finales después de recorrer el correspondiente grafo de alcanzabilidad reducido:

```
estado([7, memory, v, [0, 160]]).
estado([1, idle, v, [150, 220]]).
estado([1, idle, v, [150, 245]]).
estado([1, idle, v, [150, 305]]).
estado([1, memory, v, [285, 290]]).
estado([1, p5, v, [285, 290]]).
estado([1, memory, v, [285, 350]]).
estado([1, p5, v, [285, 350]]).
estado([1, memory, v, [285, 450]]).
estado([1, p5, v, [285, 450]]).
```

Como se puede ver en el estado final como resultado del análisis de este sistema, quedan consumidos los tres tokens de los trabajos en **p1** y se produjeron tres tokens en el lugar de salida **p5**, para lo cual se utilizaron tres veces las unidades de memoria para almacenar los trabajos y los CPU's quedan nuevamente disponibles en el lugar **idle**.

4.3 Ejemplo 3: Sistema Logístico de Empaque

A continuación se presenta un ejemplo de un diseño de red de Petri que puede no ser viable de acuerdo a ciertas circunstancias. Se trata de un sistema logístico para empaque de dulces que utiliza contenedores en cada una de las etapas del proceso de empaque. Para el ejemplo se utilizan 4 elementos los cuales son: dulces, envolturas, bolsas y listones. La producción se hace en 3 etapas –que se modelan con transiciones– que son: envolver, guardar y cerrar. Hay que hacer mención de que para envolver se requiere un elemento dulce y un elemento envoltura, para guardar se requieren dulces envueltos y una bolsa y para cerrar se requiere una bolsa llena y un listón. La figura 4.2 muestra una red de Petri que modela este proceso.

Figura 4.2: Sistema logístico de empaque.

La definición de este ejemplo de sistema logístico se realiza con hechos de la siguiente forma:

```
sigma(dulces, [dulce]).
sigma(envolturas, [envoltura]).
sigma(dulcesEnvueltos, [dulenv]).
sigma(bolsas, [bolsa]).
sigma(bolsaDeDulces, [boldul]).
sigma(listones, [liston]).
sigma(productoTerminado, [prodterm]).

places([d,e,b,de,l,bd,pt]).

trans([envolver, guardar, cerrar]).

color(d, dulces).
color(e, envolturas).
color(b,  bolsas).
color(de, dulcesEnvueltos).
color(l,  listones).
color(bd, bolsaDeDulces).
color(pt, productoTerminado).

oparit(+, procesa).
opmult(*, mult).
```

```
tabla(procesa, dulce,  envoltura, dulenv).
tabla(procesa, bolsa,  dulenv,    boldul).
tabla(procesa, liston, boldul,    prodterm).

tabla(mult, X, X).

func(envolver, [1, d, _, +, 1,  e,  _], [1, de, _], [0, 0]).
func(guardar,  [1, b, _, +, 10, de, _], [1, bd, _], [0, 0]).
func(cerrar,   [1, l, _, +, 1,  bd, _], [1, pt, _], [0, 0]).
```

De manera sencilla se le puede solicitar a Prolog que analice este modelo colocando tokens en los lugares de entrada a las transiciones. Por ejemplo, Para obtener un solo producto terminado se debe tener como mínimo 10 dulces, 10 envolturas, 1 bolsa y 1 listón, esto es:

```
?- inicio([[10, d, dulce, [0, 0]], [10, e, envoltura, [0, 0]],
[1, b, bolsa, [0, 0]], [2, l, liston, [0, 0]]]).
```

Como la figura sugiere 4.2, para obtener 10 dulces envueltos, se tienen que tomar un dulce de **d** y una envoltura de **e**, de forma que se van produciendo tokens en **de** (dulces envueltos) de uno en uno hasta producir 10 (con el mismo número de disparos de la transición **envolver**), con lo que se llega al siguiente estado:

```
estado([1, b, bolsa, [0, 0]]).
estado([1, l, liston, [0, 0]]).
estado([10, de, dulenv, [0, 0]]).
```

En este momento se habilita la transición **guardar**, la cual toma los 10 tokens en **de** y el token en **b** (bolsas) produciendo 1 token en **bd** (bolsa de dulce), de esta forma se obtiene el estado siguiente:

```
estado([1, l, liston, [0, 0]]).
estado([0, de, dulenv, [0, 0]]).
estado([1, bd, boldul, [0, 0]]).
```

Al finalizar, después de disparar la transición **cerrar**, se consume el token en **l** y el token en **bd** y se genera el producto terminado en **pt**, llegando al estado final con:

```
estado([0, de, dulenv, [0, 0]]).
estado([1, pt, prodterm, [0, 0]]).
```

En caso de que este sistema de producción sea mal alimentado, éste puede llegar a detenerse por completo si alguno de los recusos se agota, por ejemplo si se omite en el estado inicial un token en el lugar **b**, se tendría una consulta para Prolog de la siguiente forma:

```
?- inicio([[10, d, dulce, [0, 0]], [10, e, envoltura, [0, 0]],
[2, l, liston, [0, 0]]]).
```

Después de tomar los 10 tokens de los dulces y los de sus respectivas envolutras la Red de Petri se encontraría con el siguiente estado:

```
estado([1, l, liston, [0, 0]]).
estado([10, de, dulenv, [0, 0]]).
```

Se puede observar que no existen los tokens suficientes para habilitar la transición **guardar** por lo que el sistema no puede continuar con el análisis y en este punto terminaría su análisis puesto que no se puede disparar esta transición. En este caso, los lugares **BD** y **PT** son inalcanzables puesto que nunca contendrían token alguno e igualmente la tansición **cerrar** no sería disparada. De esta forma se puede notar que este es un sistema logístico no confiable puesto que requiere de un cuidado especial para su correcta ejecución.

4.4 Ejemplo 4: El Problema de la Cena de los Filósofos

En la sección (1.2.1) se presentó el problema de la cena de los filósofos, el cual ya ha sido analizado y visualizado gráficamente el comportamiento de los filósofos utilizando redes de Petri. Ahora se utiliza esta situación concurrente para plantearla en términos del prototipo de esta tesis. Para propósitos de análisis se muestran las siguientes cláusulas en forma de hechos que serán explicados con referencia en la figura 4.3.

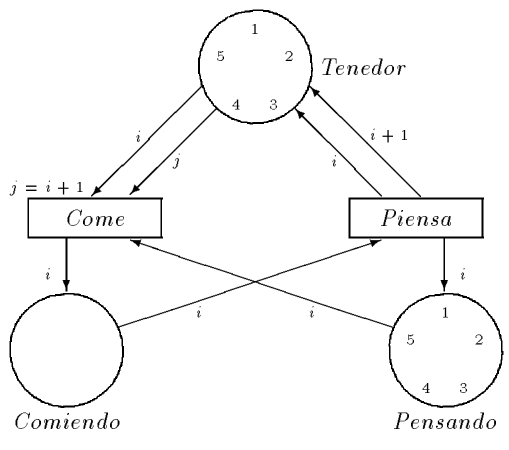

Figura 4.3: El problema de la cena de los filósofos

Los conjuntos de colores para este problema son los cinco filósofos y los cinco tenedores:

```
sigma(filosofos, [f1, f2, f3, f4, f5]).
sigma(tenedores, [t1, t2, t3, t4, t5]).
```

El conjunto de lugares está formado por los filósofos que están **pensando**, **comiendo** y el lugar **tenedor**. Las transiciones están formadas por el conjunto **come** y **piensa**:

```
places([pensando,comiendo,tenedor]).

trans([come,piensa]).
```

Con el conjunto color se determina que para el lugar pensando y para el lugar comiendo sólo pueden hallar **filósofos** y para el lugar tenedor se requieren los **tenedores**:

```
color(pensando, filosofos).
color(comiendo, filosofos).
color(tenedor,  tenedores).
```

Los siguientes operadores determinan cómo manipular los tenedores con los filósofos, por ejemplo: al sumar un tenedor t_i con un tenedor t_j obtenemos ambos tenedores juntos t_{ij}; al sumar $[f1, t1, f2]$ con los tenedores juntos $t12$ indica la presencia del filósofo $f1$, de esta forma se tiene la siguiente tabla para el operador aditivo **suma**:

```
opadit(+, suma).

tabla(suma, t1, t2, t12).
tabla(suma, t2, t3, t23).
tabla(suma, t3, t4, t34).
tabla(suma, t4, t5, t45).
tabla(suma, t5, t1, t51).

tabla(suma, [f1, t1, t2], t12, f1).
tabla(suma, [f2, t2, t3], t23, f2).
tabla(suma, [f3, t3, t4], t34, f3).
tabla(suma, [f4, t4, t5], t45, f4).
tabla(suma, [f5, t5, t1], t51, f5).
```

La tabla siguiente indica que cada filósofo tomará únicamente a sus tenedores contiguos al momento de tomarlos y colocarlos:

```
opmult(*, filosofo).

tabla(filosofo, f1, [f1, t1, t2]).
tabla(filosofo, f2, [f2, t2, t3]).
tabla(filosofo, f3, [f3, t3, t4]).
tabla(filosofo, f4, [f4, t4, t5]).
tabla(filosofo, f5, [f5, t5, t1]).
```

Las funciones de transición indican las necesidades de los filósofos para comer y pensar, la función **come** indica que el filósofo X requiere de dos tenedores, que serán explícitamente los tenedores contiguos según indican las tablas anteriores y le toma un tiempo por ejemplo entre 0 y 2 unidades de tiempo para realizar la actividad. La función **piensa** requiere únicamente de un filósofo en el lugar comiendo y produce al mismo filósofo en el lugar pensando al tiempo de regresar los tenedores a su lugar, necesitando para ello un tiempo entre 1 y 3.

```
func(come,   [1, pensando, X, +, 1, tenedor, _, +, 1, tenedor, _],
             [1, comiendo, X], [0, 2]).
func(piensa, [1, comiendo, X], [1, pensando, X, +, 1, tenedor, _, +,
              1, tenedor, _],  [1, 3]).
```

La siguiente meta para SWI-Prolog le indica al prototipo colocar los cinco filósofos en el lugar pensando y los cinco tenedores en su lugar para comenzar con el análisis de la red de Petri:

```
?- inicio([[1, pensando, f1, [0, 0]], [1, pensando, f2, [0, 0]],
         [1, pensando, f3, [0, 0]], [1, pensando, f4, [0, 0]],
         [1, pensando, f5, [0, 0]], [1, tenedor, t1, [0, 0]],
         [1, tenedor, t2, [0, 0]],  [1, tenedor, t3, [0, 0]],
         [1, tenedor, t4, [0, 0]],  [1, tenedor, t5, [0, 0]]]).
```

Inicialmente todos los filósofos están pensando, asi que el prototipo determina que el filósofo $f1$ es el candidato para habilitar la transición **come**, de forma que al producirse el disparo de esta transición se llega al siguiente estado:

```
estado([1, pensando, f2, [0, 0]]).
estado([1, pensando, f3, [0, 0]]).
estado([1, pensando, f4, [0, 0]]).
estado([1, pensando, f5, [0, 0]]).
estado([1, tenedor, t3, [0, 0]]).
estado([1, tenedor, t4, [0, 0]]).
estado([1, tenedor, t5, [0, 0]]).
estado([1, comiendo, f1, [0, 2]]).
```

Analizando la respuesta del prototipo se consigue visualizar al filósofo en el lugar **comiendo** habiendo transcurrido el tiempo entre 0 y 2, también se puede notar la ausencia de los tenedores $t1$ y $t2$ puesto que están siendo utilizados por este filósofo. En este momento se habilita la transición **piensa** y produce el siguiente estado al ocurrir el disparo:

```
estado([1, pensando, f2, [0, 0]]).
estado([1, pensando, f3, [0, 0]]).
estado([1, pensando, f4, [0, 0]]).
estado([1, pensando, f5, [0, 0]]).
estado([1, tenedor, t3, [0, 0]]).
estado([1, tenedor, t4, [0, 0]]).
```

```
estado([1, tenedor, t5, [0, 0]]).
estado([1, pensando, f1, [1, 5]]).
estado([1, tenedor, t1, [1, 5]]).
estado([1, tenedor, t2, [1, 5]]).
```

Con este estado se habilita nuevamente la transición **come** pero ahora con el filósofo $f2$ consumiendo los tenedores $t2$ y $t3$ y obteniendo un estado similar al del primer filósofo. El prototipo realiza sucesivamente un análisis semejante para cada uno de los filósofos permitiendo el paso al lugar **comiendo**, tomando los tenedores respectivos y regresando al lugar **pensando** colocando los tenedores en su lugar.

Este comportamiento sucesivo conduce a una ejecución infinita, puesto que no existe condición alguna bajo la cual el prototipo pueda detener el análisis. Esto es correcto si recordamos que este problema no tiene fin por la forma de su planteamiento. Cabe aclarar que el prototipo llega a detener la ejecución después de un tiempo, debido a que se le agota la memoria de la computadora indicando que se termina la pila local. Esta situación puede variar de un momento a otro dependiendo de el equipo que lo ejecute y a la memoria disponible.

Conclusión

En este trabajo de tesis se ha planteado la necesidad de obtener una herramienta de análisis para los sistemas concurrentes, que es realmente indispensable para poder determinar la confiabilidad de dichos sistemas. Nos encontramos en una época en que el desarrollo de sistemas se ha venido enfrentando a diversos problemas críticos, sobre todo en su desempeño y esto es debido a que no se realiza un planteamiento adecuado previo al desarrollo; una forma para evitar los problemas en los sistemas es precisamente utilizando la especificación formal.

Dada esta necesidad de tener una herramienta de especificación formal que permita analizar sistemas concurrentes, utilizamos a las Redes de Petri como fundamento para modelar de forma gráfica y matemática estos sistemas.

En esta tesis se ha conceptualizado a las redes de Petri y sus extensiones de alto nivel, centrándonos principalmente en las Redes de Petri Coloreadas con Intervalos de Tiempo para modelar sistemas y permitir su análisis. En este caso el lenguaje Prolog, fue el instrumento adecuado para el desarrollo del prototipo que permite este análisis.

Es evidente que para estudiar un sistema en particular, se requiere de una investigación previa para identificar sus partes y plantearlo en términos de este prototipo; de esta manera podemos hacer que el prototipo nos entregue el análisis respectivo de acuerdo a la forma en que se alimente al sistema y sea posible interpretar adecuadamente los resultados que nos proporcione.

Se utilizó la Técnica de Reducción de Sistemas con Transiciones Modificadas para realizar el análisis de RPCIT, obteniendo resultados satisfactorios al inspeccionar el grafo de alcanzabilidad reducido sobre sistemas modelados con redes de Petri. De esta forma se puede encontrar en el prototipo de esta tesis una herramienta de análisis de sistemas concurrentes utilizando la especificación formal.

Bibliografía

[NuG89] Nutt Gary J. *Centralized and Distributed Operating Systems.* Prentice Hall, 1989.

[TaA91] Tanenbaum Andrew S. *Sistemas Operativos.* Prentice Hall, 1991.

[KhA91] Khodabandeh Arash, *Introduction to Petri Nets*, $>$ 1991?.

[EiU96] Eindhoven University of Technology. *Petri Nets Notation*, 1996.

[Jen94] Jensen Kurt. *An Introduction to the Theoretical Aspects of Coloured Petri Nets*, DAIMI PB - 338, Agosto 1994.

[Jen90] Jensen Kurt. *Coloured Petri Nets: A High Level Languaje for System Design an Analysis*, DAIMI PB - 476, Noviembre 1990.

[vdA91] Aalst W.M.P van der. *Interval Timed Coloured Petri Nets and their analysis*, 1991.

[vdA95] Aalst W.M.P van der. *Using Interval Timed Coloured Petri Nets to calculate Performance Bounds*, 1995.

[KuT+94] Kuippers Tobias y Lutti, Bas. *ExSpect Executable Specification Tool*, Noviembre 1994.

[vaH94] K.M. Van Hee. *Information System Engineering: a Formal Approach*, 1994.