PEP: Una herramienta para traducir y analizar especificaciones de PBC con redes de Petri *

Mere Macià Valentín Valero

Escuela Politécnica Superior de Albacete
Universidad de Castilla-La Mancha
Campus Universitario s/n
02071, Albacete. Spain

Resumen El sistema PEP (**P**rogramming **E**nviroment based on **P**etri Nets) soporta las más importantes tareas de una buena herramienta de redes de Petri, incluyendo la edición de redes y una fácil simulación; todo esto dentro de unas sofisticadas componentes de programación y verificación. El componente de programación permite al usuario diseñar algoritmos concurrentes en un lenguaje de programación concurrente fácil de usar, y el sistema PEP entonces genera redes de Petri a partir de dichos programas.

1 Origen y Motivación

La herramienta PEP es el resultado de un proyecto del mismo nombre, llevado a cabo conjuntamente por 'Universität Hildshein', 'Carl-von Ossietzkg Universität Oldenburg' y 'Humboldt-Universität zu Berlin', durante los años 93/98 y que ha estado financiado por el DFG (Fundación de Investigación Alemana). Los principales artífices han sido Eike Best y Brend Grahlmann. Existen versiones FTP-disponibles de la herramienta en:

http://theorica.informatik.uni-oldenburg.de/~pep

Y, en esa misma dirección, se pueden encontrar numerosas publicaciones relacionadas con ella. En particular, en este trabajo, para realizar esta breve descripción del PEP, nos hemos basado principalmente en [1], [11], [12], [13], [14], [15] y [16].

En el origen, el principal objetivo del proyecto PEP fue el de diseñar e implementar una herramienta que, combinase las ventajas de dos de las más aceptadas teorías sobre programas concurrentes (las álgebras de procesos y las redes de Petri). La integración de esos dos métodos se realiza a través de un lenguaje de programación concurrente ($B(PN)^2$, [5]). En la idea original, se partía de un programa escrito en $B(PN)^2$, luego se trasladaba al álgebra de procesos del

* Este trabajo ha estado parcialmente financiado por el proyecto CICYT "Evaluación de Prestaciones de Sistemas Distribuidos, Ref. TIC2000-0701-C02-02"

PBC ([2], [3] y [6]) y, de aquí, a una clase especial de redes de Petri (cajas de Petri), (ver Figura 1), para así poder aplicar la teoría de redes de Petri en la verificación de programas. A continuación pasamos a describir brevemente el PBC y, posteriormente se dará una pequeña introducción del $B(PN)^2$.

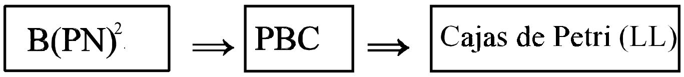

Figura1. Dependencia original entre los objetos del sistema PEP

1.1 El PBC

El PBC (**P**etri **B**ox **C**alculus) se originó como puente entre los dos modelos más estudiados en el campo de los sistemas concurrentes: las álgebras de procesos y las redes de Petri. De esta forma, se consiguen las ventajas propias de los modelos algebraicos y las estructurales de las redes de Petri.

El PBC proporcionará, para una misma sintaxis, dos semánticas equivalentes: una operacional y, la otra, denotacional. Esta última definida composicionalmente y basada en en una clase especial de redes de Petri etiquetadas llamadas *cajas*.

El PBC está inspirado en el álgebra de procesos del CCS [20], extendida y modificada. Su sintaxis viene dada de la forma descrita a continuación. Una *expresión* del PBC se define de acuerdo a la siguiente notación BNF:

$$E ::= \ \alpha \mid X \mid E; E \mid E \,\square\, E \mid E \,\|\, E \mid E[f] \mid E\, sy\, a \mid E\, rs\, a \mid [a : E] \mid [E * E * E]$$

donde, brevemente, la descripción de estos operadores sería:

- $\alpha \in \mathcal{L}$ es una multiacción ($\mathcal{L} \in \mathcal{B}(A)$, donde A es el conjunto de los nombres de acciones);
- X la utilizaremos en la definición de ecuaciones recursivas;
- $E; F$ representará la secuencia de procesos;
- $E \,\square\, F$ será la elección entre dos procesos;
- $E \,\|\, F$ se utilizará para la composición en paralelo. Cada uno de los procesos podrá evolucionar de forma independiente;
- $E[f]$ representará al operador de reetiquetación, donde f es una función de reetiquetación que conserva los conjugados;
- $E\, sy\, a$ será la sincronización sobre acciones conjugadas ($a, \widehat{a} \in A$): los pares $(a, \widehat{a})$ se unirán dando lugar a $\emptyset$;
- $[a : E]$ es un operador derivado, $[a : E] = (E\, sy\, a)\, rs\, a$;
- $[E_1 * E_2 * E_3]$ representará al operador iteración: se empezará realizando E_1, pasando luego a realizar un número de veces (o ninguna) E_2, y terminando con la realización de E_3.

La sintaxis del PBC se puede ampliar en lo que hace referencia a la declaración de variables y de canales, como se sigue de [3] y de [5]. Así en la Tabla 1 se tiene esa extensión :

D(q,V) |	declaración de la variable q con dominio de valores en V
D(q,V,i) |	declaración de la variable q con dominio de valores en V e inicialización i
C(c,k,V) |	declaración del canal c, con capacidad k y dominio de valores en V

Tabla1. Una extensión de la sintaxis del PBC

A continuación pasamos a definir los objetos semánticos de nuestra semántica denotacional: las cajas planas, que no son más que una clase especial de redes de Petri etiquetadas.

Definición 1 *Una red de Petri etiquetada plana es Σ=(S,T,W,λ) donde (S,T,W) es una Red de Petri ($S \cap T = \emptyset$, $W : (S \times T) \cup (T \times S) \to \mathbb{N}$) y λ es una función llamada la etiquetación de Σ tal que: $\lambda : S \to \{e, i, x\}$ y $\lambda : T \to \mathcal{L}$.*
Los lugares s con $\lambda(s) = e$ se llaman lugares de entrada, siendo

$$^\bullet\Sigma = \{s \in S \mid \lambda(s) = e\}$$

Los lugares s con $\lambda(s) = x$ se llaman lugares de salida, siendo

$$\Sigma^\bullet = \{s \in S \mid \lambda(s) = x\}$$

El resto de lugares se llaman internos.

Definición 2 *Una caja de Petri plana es una clase de red de Petri etiquetada plana Σ=(S,T,W,λ) que cumple:*

(i) $^\bullet\Sigma \neq \emptyset$
(ii) $\Sigma^\bullet \neq \emptyset$
(iii) $\forall s \in {}^\bullet\Sigma\ \forall t \in T :$ *W(t,s)=0*
(iv) $\forall s \in \Sigma^\bullet\ \forall t \in T :$ *W(s,t)=0*

Llamaremos M_e al marcado de una caja plana que sólo tiene un token en cada lugar de entrada, y ningún token en los otros lugares. Será el marcado inicial.

Llamaremos M_x al marcado de una caja plana que sólo tiene un token en cada lugar de salida, y ningún token en los otros lugares. Representará el marcado final.

Una caja plana es limpia si cuando se alcanza un marcado con tokens en todos los lugares de salida, entonces ese marcado coincide con M_x.

A través de unas cajas operador (se tendrá una caja operador Ω_{OP} por cada una de las operaciones (OP) de la sintaxis), se define el álgebra de cajas. De esta forma se obtiene el homomorfismo:

$$Box \,:\, Expresiones\ PBC \longrightarrow\ Cajas\ de\ Petri\ (LL)$$

$$Box(OP(E_1,\ldots,E_n)) = \Omega_{OP}(Box(E_1),\ldots,Box(E_n))$$

que proporciona siempre redes de Petri 1-seguras y limpias.

1.2 El $B(PN)^2$

El $B(PN)^2$ (a **B**asic **P**etri **N**et **P**rogramming **N**otation) tiene por objetivos: tener una semántica de RdP composicional (vía el álgebra de procesos del PBC) y ser un lenguaje de programación concurrente sencillo y flexible.

En la Tabla 2 describe la sintaxis del $B(PN)^2$, donde las palabras reservadas están en negrita.

program	::=	block
block	::=	**begin** scope **end**
scope	::=	com | decl ; scope
decl	::=	**var** var-name : type | decl ; decl
type	::=	set | **chan** b **of** set (b$\in$ $\{0,1,2,...\} \cup \{w\}$)
com	::=	<expr> | com ; com | com $\parallel$ com | block | **do** alt - set **od**
expr	::=	$'w \mid w \mid w' \mid c? \mid c! \mid$ const | expr *op* expr | *op* expr
alt-set	::=	com ; **repeat** | com ; **exit** | alt - set $\Box$ alt - set
const	::=	**F** | **T** | z ($z \in \mathbb{Z}$)

Tabla2. Sintaxis del $B(PN)^2$

Dicho lenguaje incorpora características habituales de los lenguajes de programación tradicionales: identificadores (nombres de variable), w (para variables planas) o c (para canales); composición secuencial (;) y composición paralela ($\|$); constantes (enteras y Booleanas); operaciones *op* unarias/ binarias; y declaraciones de variables. Por ejemplo, la cláusula **var** w:{**F**,**T**} declara una variable Booleana w, y la cláusula **var** w:$\mathbb{Z}$ declara una variable entera w. (Un programa completo, como de costumbre, es un bloque para el cual todas las variables han sido declaradas). Las construcciones más novedosas son:

1. La cláusula **var** c : **chan** b **of** s declara un canal (buffer FIFO) c de capacidad b que puede llevar valores en s. Esto puede ser: de capacidad no acotada (w); de capacidad cero y en este caso se da la sincronización como en CSP ([18]); o de cualquier capacidad finita mayor que cero.
2. El comando **do ... od** combina el comando tradicional **if** (introduciendo la cláusula **exit**) con el comando tradicional **while** (con la cláusula **repeat**). Puede haber cero o más iteraciones, siendo cada una la ejecución de una cláusula repeat, seguida por la ejecución de una cláusula exit.
3. El comando <expr> denota la ejecución atómica de la expresión expr. Una variable w puede aparecer com $'w$ y como w' denotando, respectivamente, su valor justo antes y después de la ejecución del comando; y (redundantemente, pero convenientemente) sólo como w, denotando ambos e implícitamente $'w = w'$. La comunicación, sobre un canal c, también se incorpora con $c!$ como una comunicación 'de salida', y con $c?$ como una comunicación 'de entrada'; en ambos casos la expresión proporciona el valor comunicado.

2 La Estructura del PEP

Más tarde, se fueron incorporando nuevas mejoras, como una semántica alternativa para los programas $B(PN)^2$ a través de las M-redes ([4]) (una clase especial de redes de Petri de alto nivel, HL); un nuevo interfaz de entrada PFA (Parallel Finite Automata), con acciones $B(PN)^2$ en las anotaciones de los arcos y compilado en un programa $B(PN)^2$ ([17]). Luego les siguieron varias más, entre otras, la posibilidad de introducir las especificaciones de entrada bajo el formalismo SDL (Specification Description Language) y su compilador a HL-net, ([10]); una ampliación del $B(PN)^2$ ([9]); ó un compilador de $B(PN)^2$ a código C ([19]).

En la actualidad la herramienta consta, principalmente, de los tres componentes siguientes (ver Figura 2): El componente de edición y simulación, el componente fórmula y el componente de verificación.

2.1 El componente de edición y simulación

Contiene editores y simuladores para sistemas concurrentes, desde los cuales las redes de Petri correspondientes pueden generarse automáticamente, utilizando los diferentes compiladores integrados en este componente. Como ya hemos comentado con anterioridad, ha estado creciendo continuamente, correspondiendo con la intención principal de la herramienta - esto es, utilizar la teoría de redes de Petri para la verificación de programas - siendo un editor de programa y un editor de redes el punto de partida. Así el componente de edición y simulación del PEP contiene:

1. Editores para SDL, PFA, programas en $B(PN)^2$, términos PBC y redes HL (M-redes) y LL (cajas de Petri).

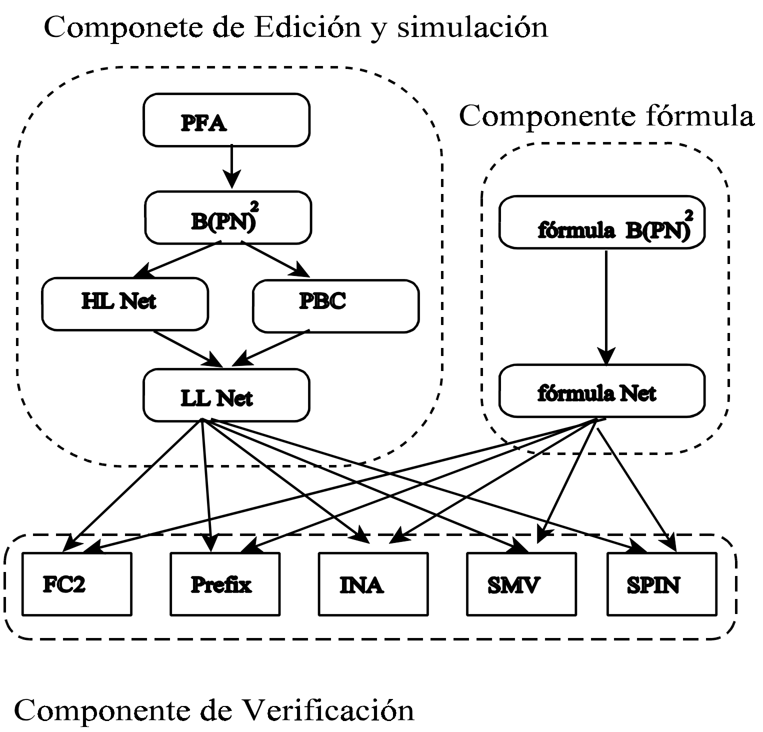

Figura2. Las diferentes componentes del sistema PEP

2. Simuladores para programas $B(PN)^2$, redes HL y LL.
3. Compiladores de la forma, entre otros: $B(PN)^2 \Rightarrow$ PBC, PBC $\Rightarrow$ LL, PFA $\Rightarrow B(PN)^2$

y los que se indican con las flechas de la Figura 3, que representa la ventana principal de un proyecto PEP; o bien a través del menú de **Execute**.

2.2 El componente fórmula

PEP soporta la definición de un conjunto de fórmulas en una lógica temporal, con lógica proposicional sobre los nombres de los lugares, aumentada con $\Box$ para "siempre"y $\Diamond$ para "posiblemente en algún momento futuro".

Así se permite la entrada de propiedades para ser probadas por alguno de los comprobadores de propiedades (model checker) integrados. Se dispone de tres modos, según la lógica soportada: Modo PEP (para el checker prefijo), SPIN y SMV.

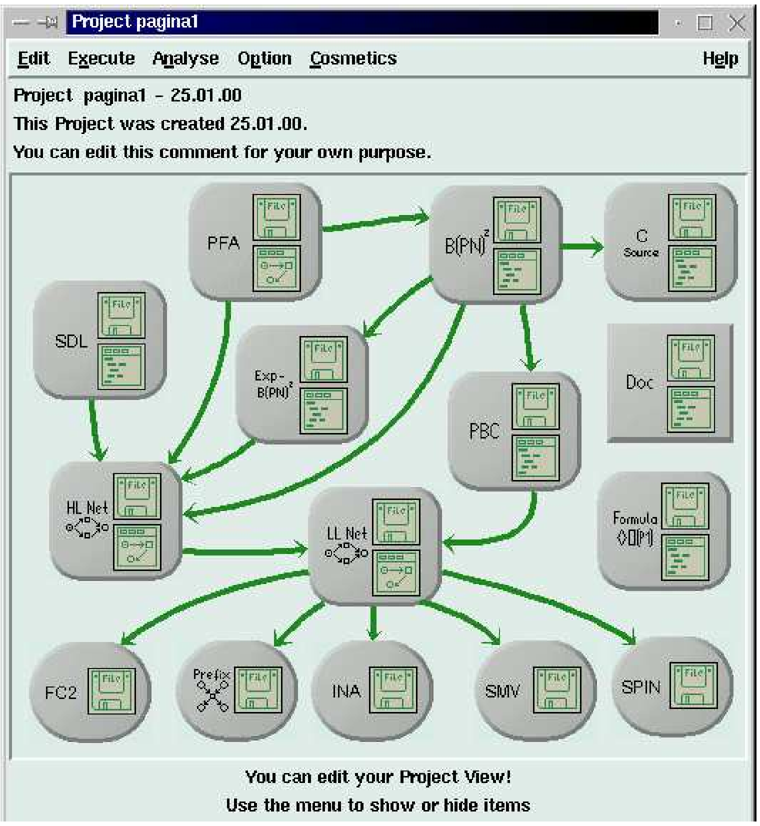

Figura3. Ventana principal de un proyecto PEP

El sistema ofrece la posibilidad de que la entrada sea una fórmula que puede referirse a un programa concurrente o a una red. De hecho, las fórmulas de programa se transforman en fórmulas para redes ([14]).

2.3 El componente de verificación

Además del análisis de propiedades estándar, se obtuvo la implementación de un algoritmo de model checker ([8]), para permitir así que una fórmula en una lógica temporal pudiera probarse.

También se añadieron interfaces a otros paquetes de verificación como 'INA' ([23]) ó 'SMV' ([7]), para ofrecer así distintos métodos de verificación y poder soportar lógicas más potentes. Estas posibilidades pueden observarse en la Figura 4, donde viene reflejado el menú de **Analyse**.

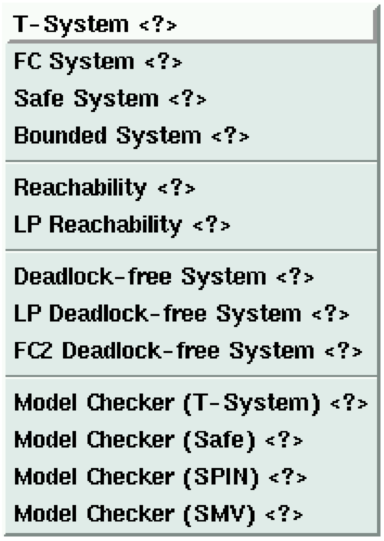

Figura4. Menú del Analyse

3 Ejemplos

Al utilizar la herramienta, seguiremos habitualmente las cinco etapas mostradas en la Figura 5.

Fase 1 Normalmente el usuario comenzaría modelando un sistema concurrente, para lo cual dispone de los editores de PFA, $B(PN)^2$, PBC ó, incluso directamente, redes de Petri.

Fase 2 La simulación, el análisis y la verificación son sólo posibles para las redes de Petri. Por ello, dependiendo de la forma de introducir la especificación, según la Fase 1, se emplearía el compilador correspondiente.

Fase 3 Una simulación permite al usuario familiarizarse con el sistema concurrente diseñado. Esto puede darnos valiosos indicios, pero no prueba la corrección.

Fase 4 Alguna información específica puede obtenerse con la aplicación de algunos algoritmos de análisis estándar. Especialmente, en el caso de redes de Petri ordinarias, puede ser de gran interés si es viva o 1-segura.

Fase 5 Finalmente, un punto importante es la fase de verificación. En comparación con otras herramientas, el componente de verificación ofrece más posibilidades. Los algoritmos de comprobación de propiedades sobre el modelo pueden determinar cuando una red de Petri satisface una propiedad, dada en términos de una lógica temporal. La transparencia, y así la completa funcionalidad, del componente de verificación, se obtiene por un traductor que transforma una fórmula abstracta, (tal como "si un proceso termina"), en una fórmula referente a su red de Petri correspondiente.

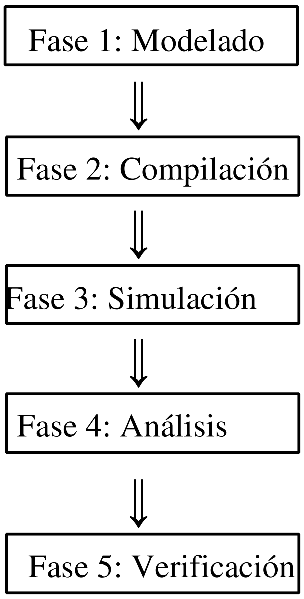

Figura5. Ciclo típico en el uso del PEP

A continuación pasamos a estudiar dos ejemplos populares en el campo de la concurrencia: El algoritmo de Peterson y el problema del productor/consumidor.

3.1 El Algoritmo de Peterson

La descripción del problema de la sección crítica para dos procesos puede encontrarse en [21].

Supongamos que se tiene un sistema de dos procesos en cooperación P_1 y P_2. Cada proceso tiene un segmento de código (sección crítica), en el que el proceso puede estar leyendo o escribiendo sobre las variables comunes, actualizando una tabla, escribiendo sobre un fichero, etc. La característica relevante del sistema es que cuando un proceso está ejecutando su sección crítica, el otro proceso no está autorizado a ejecutar su sección crítica. Así, la ejecución de secciones críticas por procesos es mutuamente excluyente en el tiempo. El problema de la sección *crítica es diseñar un protocolo que puedan utilizar los procesos para cooperar y respete la restricción anterior. Cada proceso tiene que solicitar permiso para entrar en su sección crítica.*

Fase 1 Este algoritmo se puede modelar, por ejemplo, en PFA. (Ver Figuras 6 y 7).

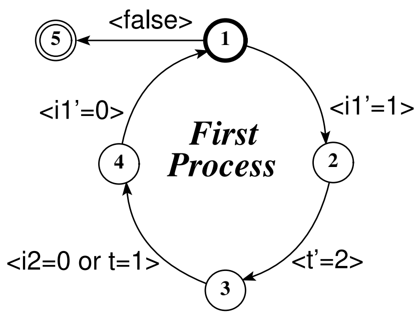

Figura6. El algoritmo de Peterson modelado con Autómatas Finitos (Primer Proceso)

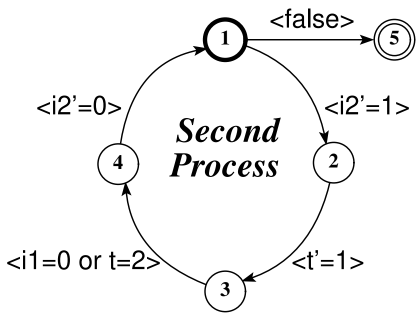

Figura7. El algoritmo de Peterson modelado con Autómatas Finitos (Segundo Proceso)

Fase 2 Podemos utilizar el compilador que transforma el autómata en un programa $B(PN)^2$:

$$PFA \Rightarrow B(PN)^2$$

obteniendo de esa forma el programa escrito en $B(PN)^2$ de la Figura 8.

```
begin
    var i1,i2:{0..1} init 0;
    var t:{1..2} init 1;
    begin
        do <true> enter
            <i2'=1>;
            <t'=1>;
            <i1=0 or t=2>;
            <i2'=0>; repeat
         [] <false>; exit
        od
    end
||
    begin
        do <true> enter
            <i1'=1>;
            <t'=2>;
            <i2=0 or t=1>;
            <i1'=0>; repeat
         [] <false>; exit
        od
    end
end
```

Figura8. El algoritmo de Peterson en $B(PN)^2$

Después haremos lo propio para transformar el programa $B(PN)^2$, en el correspondiente programa en el álgebra de procesos PBC:

$$B(PN)^2 \;\Rightarrow\; PBC$$

y así obtendríamos el algoritmo expresado en PBC de la Figura 9.

Y, por último, utilizar el traductor a redes 1-seguras:

$$PBC \;\Rightarrow\; LL$$

obteniendo así la caja de Petri de la Figura 10 (imagen parcial de la misma).

Fase 3 Una simulación sobre la red anterior parece indicarnos que está libre de bloqueos.

```
[{i1,i1_0_0,i1_0_1,i1_1_0,i1_1_1,i2,i2_0_0,i2_0_1,i2_1_0,i2_1_1,
t,t_1_1,t_1_2,t_2_1,t_2_2}:
 D_BOX(i1,0,{0..1}) || D_BOX(i2,0,{0..1}) || D_BOX(t,1,{1..2}) || (

( [(({})) * (({i2_0_1} # {i2_1_1}) ; ({t_1_1} # {t_2_1}) ;
({i1_0_0, t_1_1} # {i1_0_0, t_2_2} # {i1_1_1, t_2_2}) ; ({i2_0_0}
# {i2_1_0})) * ((STOP))] )
 ||

( [(({})) * (({i1_0_1} # {i1_1_1}) ; ({t_1_2} # {t_2_2}) ;
({i2_0_0, t_1_1} # {i2_1_1, t_1_1} # {i2_0_0, t_2_2}) ; ({i1_0_0}
# {i1_1_0})) * ((STOP))] )

);i1;i2;t]
```

Figura9. El algoritmo de Peterson en PBC

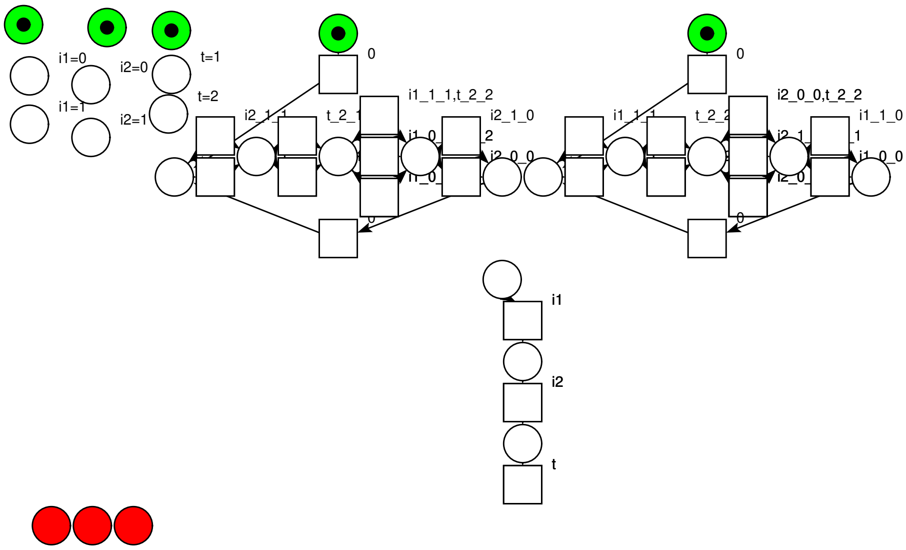

Figura10. La caja de Petri asociada al algoritmo de Peterson

Fase 4 Algunos análisis estándar nos proporcionan los siguientes resultados:

- No es un T-sistema.
- No es un FC-sistema.
- Es un sistema seguro.

Fase 5 Primero verificamos que está libre de bloqueos.

Luego probamos que se da la propiedad de exclusión mutua con la fórmula siguiente:

$$\Diamond((('i1 = 1' * \ 't = 1') + ('i1 = 1' * \ 'i2 = 0')) *$$
$$(('i2 = 1' * \ 't = 2') + ('i2 = 1' * \ 'i1 = 0')))$$

Y también probamos que, por ejemplo, siempre será posible que el Proceso primero entre en su Sección Crítica:

$$\Box\Diamond(('i1 = 1' * \ 't = 1') + ('i1 = 1' * \ 'i2 = 0'))$$

3.2 El problema del Productor/Consumidor

La descripción de este clásico problema de la programación concurrente puede encontrarse en [22] y sería:

Una unidad de producción produce cierta clase de objetos, que deposita en un almacén. Este almacén puede tener una capacidad máxima. Una unidad de consumo retira los objetos del almacén para su posterior consumo. De esta forma se pretende describir un sistema que coordine la producción y el consumo.

Fase 1 En nuestra formulación como un programa en $B(PN)^2$ hemos considerado:

- Un productor que puede producir elementos del conjunto $\{0, 1\}$, sobre la variable x.
- Un consumidor que consumirá los productos del consumidor, sobre la variable y.
- Un canal (c), de capacidad 2, que hará el papel del almacén actuando como un buffer con estrategia FIFO.

De esta forma, en nuestro caso, el problema del productor/consumidor descrito en la notación $B(PN)^2$ sería el mostrado en Figura 11). Obviamente, se puede conseguir la extensión y modificación de los elementos producidos/consumidos, y la capacidad del almacén.

begin

var c: **chan** 2 **of** {0,1};
var x: {0,1} ;
var y: {0,1} ;

do < **true** > **enter**
    < x'=0 **or** x'=1 >; <c!=x>; **repeat**
od

||

do < **true** > **enter**
    < y'=c? >; **repeat**

od

end

Figura11. El problema del productor/consumidor en $B(PN)^2$

Fase 2 Podemos utilizar el traductor al álgebra:

$$B(PN)^2 \Rightarrow PBC$$

y así tendríamos el problema expresado en PBC de la Figura 12.

```
[{c,c!!,c??,c?0,c!0,c?1,c!1,x,x_0_0,x_0_1,x_1_0,x_1_1,
y,y_0_0,y_0_1,y_1_0,y_1_1} : C_BOX(c,2,{0..1}) ||
D_BOX(x,0,{0..1}) || D_BOX(y,0,{0..1}) ||
  ( [(({})) * (({x_0_0} # {x_0_1} # {x_1_0} # {x_1_1}) ;
 ({c!0, x_0_0} # {c!1, x_1_1})) * STOP] ||
  [(({})) * (({c?0, y_0_0} # {c?0,y_1_0} #
   {c?1, y_0_1} #{c?1, y_1_1})) * STOP] );
  [{}*(c?0[]c?1)*c];x;y]
```

Figura12. El problema del productor/consumidor en PBC

Y después, utilizar el traductor a redes 1-seguras: $PBC \Rightarrow LL$ obteniendo por tanto la caja de Petri de la Figura 13 (de nuevo una imagen parcial).

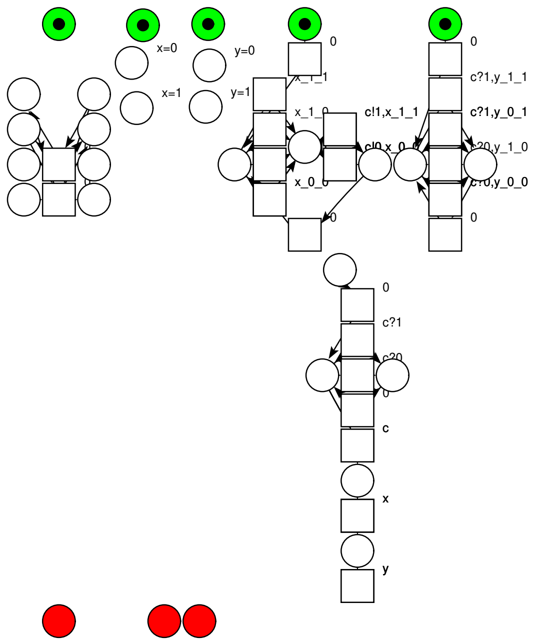

Figura13. La caja asociada al problema del productor/consumidor

Fase 3 Una simulación sobre la red anterior parece indicarnos que está libre de bloqueos.

Fase 4 Algunos análisis estándar nos dan los siguientes resultados:

- No es un T-sistema.
- No es un FC-sistema.
- Es un sistema seguro.

Fase 5 Primero verificamos que está libre de bloqueos. Y luego que se cumplen las siguientes propiedades:

$\Box\Diamond\, 'y = 0'$ (siempre es posible que el consumidor consuma el elemento 0) ó

$\Box\Diamond\, ('x = 1' * 'y = 0')$ (siempre es posible que el productor produzca el elemento 1 y el consumidor consuma el 0).

Referencias

1. E. Best. *Partial Order Verification with PEP . Proc. of POMIV'96*, 1996.
2. E. Best. *Petri Nets, Process Algebras and Programming Languages.* Draft Tutorial prepared for the 3rd Advanced Course on Petri Nets, 1996.
3. E. Best, R. Devilllers, and J. Hall. *The Box Calculus : A New Causal Algebra with Multi-label Communication. Advances in Petri Nets. Springer, LNCS*, 609:21–69, 1992.
4. E. Best, H. Fleischhack, W. Fraczak, R.P. Hopkins, H. Klaudel, and E. Pelz. *An M-net Semantics of B(PN)^2. Proc. of STRICT'95 (Berlin), Springer*, 1995.
5. E. Best and P. Hopkins. *$B(PN)^2$ - a Basic Petri Net Programming Notation. PARLE 93. Springer, LNCS*, 694:379–390, 1993.
6. E. Best and M. Koutny. *A Refined View of the Box Algebra. Proc. Petri Net Conference'95. Springer, LNCS*, 935:1–20, 1995.
7. E. Clarke, K. McMillan, and S. Campos. *Symbolic Model Checking. Proc. of CAV'96 (New Brunswick). Springer, LNCS*, 1102, 1996.
8. J. Esparza, S. Römer, and W. Vogler. *An Improvement of McMillan's Unfolding Algorith. Proc. of TACAS'96 (Oldenburg). Springer, LNCS*, 1055, 1995.
9. H. Fleischhack and B. Grahlmann. *A Petri Net Semantics for $B(PN)^2$ with Prosedures. Proc. of PDSE'97 - Boston*, 1997.
10. H. Fleischhack and B. Grahlmann. *A Compositional Petri Net Semantics for SDL. Proc. of ATPN'98 - Lisboa*, 1998.
11. B. Grahlmann. *PEP: A Programming Enviorment based on Petri Nets. Application and Theory of Petri Nets'95 - Tool Presentation (Torino)*, 1995.
12. B. Grahlmann. *The PEP Tool. CAV'97 (Haifa)*, 1997.
13. B. Grahlmann. *The PEP Tool. ATPN'97 (Toulouse)*, 1997.
14. B. Grahlmann. *The Reference Component of PEP. Proc. of TACAS'97 (Enschede). Springer, LNCS*, 1217, 1997.
15. B. Grahlmann. *The state of PEP. AMAST'98 (Algebraaic Methodology amd Software Technology, Manaus Brazil)*, 1998.
16. B. Grahlmann and E. Best. *PEP - More than a Petri Net Tool. Proc. of TACAS'96 (Passau)*, 1996.
17. B. Grahlmann, M. Moeller, and U. Anhalt. *A new Interface for the PEP tool - Parallel Finite Automata. Proc. of 2. AWPN'95 (Oldenburg)*, 1995.
18. C.A.R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International, 1985.
19. S. Melzer. *Design and Implementation of a C-Code Generator for $B(PN)^2$. Proc. of Concurrency, Specification and Programming - Berlin*, 1994.
20. R. Milner. *Communication and Concurrency.* Prentice-Hall International, 1989.
21. J.L. Peterson and A. Silberschatz. *Sistemas Operativos. Conceptos Fundamentales.* Reverté, 1989.
22. M. Silva. *Las Redes de Petri: en la Automática y la Informática.* AC, 1988.
23. P.H. Startke. *INA: Integrated Net Analyzer. (Handbuch)* , 1992.