Redes de Petri móviles para la especificación y verificación de propiedades de seguridad en sistemas ubicuos

TESIS DOCTORAL

Fernando Rosa Velardo
Departamento de Sistemas Informáticos y Computación
Facultad de Ciencias Matemáticas
Septiembre 2007

Trabajo dirigido por el Doctor David de Frutos Escrig

Redes de Petri móviles para la especificación y verificación de propiedades de seguridad en sistemas ubicuos

Memoria presentada para obtener el grado de
Doctor en Ciencias Matemáticas

Fernando Rosa Velardo

Dirigida por el Doctor

David de Frutos Escrig

Departamento de Sistemas Informáticos y Computación
Facultad de Ciencias Matemáticas
Septiembre 2007

A mis padres
A mis hermanas

Agradecimientos

Quiero empezar este trabajo agradeciendo a todas las personas que han hecho que este haya sido posible o más llevadero. En primer lugar le agradezco a mi director, David de Frutos, su disponibilidad, sus sugerencias y sus correcciones, y sobre todo por contagiarme su entusiasmo al enfrentarme a nuevos problemas, resueltos muchos de ellos en productivas sesiones de "lluvia de ideas" frente a la pizarra del despacho, casi siempre a avanzadas horas de la tarde. La satisfacción con la que esos días dejaba la facultad ha mantenido siempre presentes los motivos por los que decidí emprender esta tarea.

La versión definitiva de esta tesis tampoco sería lo que es sin el filtro que han supuesto las correcciones de los informantes. Por ello quiero agradecer a Javier Esparza y a Narciso Martí tanto sus comentarios elogiosos como sus constructivas críticas, que han contribuido decisivamente a que esta tesis acabe siendo un poquito mejor.

En el departamento SIP/SIC, y más concretamente en la sección, me he sentido como uno más desde el comienzo, es decir, desde los primeros días como becario, tan ninguneados en tantos sitios. Por eso, y por echarme una mano siempre que lo he pedido, gracias a mis compañeros, sobre todo a aquellos con los que he trabajado más codo con codo, Clara, Alberto y Olga. En especial, tengo que agradecer la paciencia de José Miguel, con el que he compartido despacho durante la mayoría de este tiempo, por no perderla ni siquiera en la n-ésima distracción en una de esas tardes menos centradas.

También quiero nombrar (aunque no uno a uno por razones obvias) a mi gran familia, especialmente a la "sección madrileña", y a Nando, por hacer que Badajoz no haya cambiado demasiado desde que me fui. Además, está esa otra pequeña familia, Ester, Ana, Stef y Nuria, gracias a la cual Madrid ha sido un hogar en estos últimos años.

Y por último, a mis padres y a mis hermanas, muchísimas gracias por todo.

Resumen

Las redes de Petri son un formalismo presentado por primera vez en la tesis de Carl Adam Petri para el estudio de sistemas concurrentes. Desde su aparición hace ya más de cuatro décadas han aparecido numerosos trabajos que han desarrollado una estable teoría para la especificación y verificación de sistemas concurrentes mediante redes de Petri. Factores como su representación gráfica y los numerosos resultados de decidibilidad para problemas indecidibles en otros formalismos más expresivos son sin duda parte de los motivos que han contribuido al éxito del que han gozado las redes de Petri.

Sin embargo, la teoría clásica de redes se enfrenta al desafío de adecuarse a las nuevas necesidades de los sistemas informáticos que los ingenieros diseñan en la actualidad. Un claro ejemplo de estas nuevas necesidades son los sistemas distribuidos y móviles, en los que conceptos como localidad y conectividad cobran gran importancia. Un paso más allá están los sistemas ubicuos, definidos de manera un tanto difusa, que comparten la particularidad de estar formados por componentes con poca capacidad de cálculo que se comunican a través de redes establecidas específicamente para cada comunicación, lo que dificulta el diseño de sistemas ubicuos seguros.

En esta tesis nos centraremos en dos aspectos de vital importancia en los sistemas ubicuos, como son la coordinación y la seguridad. El primer objetivo de la tesis es el de extender el modelo clásico de redes de Petri con características que nos permitan hablar de términos propios de los sistemas ubicuos, como pueden ser componentes, localidad, movilidad, comunicación, coordinación o seguridad. Presentaremos progresivamente distintos modelos que vayan incorporando nuevas características, comparando las distintas extensiones entre sí fundamentalmente a través del estudio de las propiedades de alcanzabilidad, recubrimiento y acotación.

Por su especial relevancia desde el punto de vista teórico, prestaremos especial interés a los resultados obtenidos para dos clases de redes, que llamamos ν-APNs y sistemas RUN, extensiones de redes de Petri con nombres puros y replicación de componentes, respectivamente. Veremos que estas dos clases son equivalentes con una noción de equivalencia que preserva, al menos, la alcanzabilidad y el recubrimiento, y que ambas extienden estrictamente a las redes de Petri sin llegar a la potencia de cómputo de las máquinas de Turing.

Para el estudio de propiedades de seguridad introducimos una semántica que tiene en cuenta la presencia del entorno incierto en el que se interactúa y se prueba

que no por ello perdemos los resultados de decidibilidad que habíamos obtenido. Aunque no era el objetivo primero de esta tesis, presentamos un prototipo de herramienta implementada en el sistema Maude, basada en los resultados obtenidos en la tesis para la especificación y verificación de sistemas ubicuos. Por último, mostramos cómo modelar usando los modelos presentados la asociación transitoria segura, una forma de autenticación débil en sistemas ubicuos.

Índice general

1. Introducción **1**
1.1. Computación ubicua 2
1.2. Seguridad 4
1.3. Redes de Petri y extensiones 6
1.4. Álgebras de procesos: nombres y replicación 8
1.5. Objetivos y organización de la tesis 11

2. Preliminares **15**
2.1. Preórdenes y multiconjuntos 15
2.2. Sistemas de transiciones 16
2.3. Redes de Petri 18
2.4. Lógica de reescritura y Maude 19
2.5. Máquinas de dos contadores (TCM) 21
2.6. Nociones de simulación 21

3. Computación ubicua y redes de Petri **27**
3.1. Redes ubicuas 27
3.2. De los sistemas ubicuos a las redes P/T 30
3.3. Un ejemplo de aplicación 34
3.4. Extensiones del modelo básico 37

4. Redes de Petri móviles **41**
4.1. Sistemas MSPN 42
4.2. Ejemplo 45
4.3. De los sistemas MSPN a los sistemas ubicuos 49
4.4. Autenticación 53
4.5. Ejemplo: protocolo de autenticación distribuido 57
4.6. Generación de nombres abstractos 59

5. Modelos simplificados **63**
5.1. Redes de Petri con contadores 63
5.2. ν-APNs 70

6. Expresividad de las MSPN **79**
6.1. Resultados de alcanzabilidad 79
6.2. Resultados de recubrimiento 89
6.3. Resultados de acotación 104
6.4. Desigualdades 117
6.5. Arcos de transferencia y reset 119

7. Redes ubicuas replicadas **125**
7.1. Redes ubicuas replicadas 126
7.2. Recolección de basura en sistemas RUN 129
7.3. Sistemas RUN centralizados 130
7.4. Verificación de sistemas RUN 133
7.5. Creación de nombres y replicación 153

8. Sistemas etiquetados **159**
8.1. Sistemas de transiciones etiquetados 160
8.2. Caracterización alternativa 164
8.3. Derivación de lenguajes etiquetados 171
8.4. Lenguaje $D\pi$ etiquetado 173
8.5. Aplicaciones de la estructura algebraica 178

9. Verificación de sistemas abiertos **183**
9.1. Ejemplo: control de acceso a recursos 184
9.2. Sistemas abiertos 186
9.3. Propiedades de seguridad 190
9.4. Semántica simbólica de sistemas MSPN 193
9.5. Verificación de sistemas abiertos 201

10. Implementación de las MSPN **207**
10.1. ν-APNs en lógica de reescritura 207
10.2. ν-APNs en Maude 216
10.3. De las MSPNs a las ν-APNs 218
10.4. Algoritmo de recubrimiento 222
10.5. Arquitectura de la implementación 223

11. Asociación segura transitoria **229**
11.1. La política del pato que resucita 230
11.2. Sistemas TSA 231
11.3. Un ejemplo: Termómetros electrónicos 234
11.4. Verificación de sistemas TSA 237

12. Conclusiones **251**

A. Código del prototipo **255**

Índice de figuras

2.1. Red de Petri que modela un sistema productor-consumidor 21

3.1. Ejemplo de red ubicua 32
3.2. Traducción del ejemplo de la figura 3.1 a redes P/T 33
3.3. Traducción de sistema ubicuo a red P/T 35
3.4. Otro ejemplo de sistema ubicuo y su simulación mediante una red P/T 35
3.5. Sistema ubicuo modelado por redes de Petri 36

4.1. Enrutador 46
4.2. Servidores 46
4.3. Cliente honesto 47
4.4. Marcajes de una traza con un cliente honesto 47
4.5. Cliente malintencionado 49
4.6. Contrabandista 49
4.7. Representación de sistemas ubicuos con sistemas MSPN 50
4.8. Transiciones autónomas (izquierda) y de sincronización (derecha) . 55
4.9. Enrutador con autenticación 56
4.10. Servidores con autenticación 57
4.11. Cliente honesto 57
4.12. Marcajes de una traza con un cliente honesto usando autenticación 58
4.13. Ataque DoS 58
4.14. Servidor de tickets (arriba) y agente (abajo) 60
4.15. Servidor crítico 61
4.16. Clientes 61

5.1. Forma general de transiciones sucesor en redes con contadores . . . 67
5.2. Simulación de CsPNs con MSPNs 67
5.3. Red con conjuntos en arcos 71
5.4. Simulación de la red en la figura 5.3 sin conjuntos en los arcos . . . 72
5.5. Simple red con contador 76
5.6. Simulación de red con contador con ν-APNs 77

6.1. Desdoblamiento de una red con un contador respecto de $M(c)=\{2\}$ y $M(p)=\emptyset$ . . . 81
6.2. Simulación de $Inc(i,s,t)$ . . . 83
6.3. Simulación de $Dec(i,s,t,u)$ (eligiendo $c_i>0$) . . . 84
6.4. Simulación de $D = Dec(i,s,t,u)$ (eligiendo $c_i=0$) . . . 85
6.5. Simulación de $D = Dec(i,s,t,u)$ con identificadores naturales (eligiendo $c_i=0$) . . . 89
6.6. Una ν-APN . . . 93
6.7. Cómputo de $Pre_{t(\sigma)}(\mathcal{C}(M))$ . . . 98
6.8. Un ejemplo sencillo . . . 99
6.9. De nuevo el ejemplo sencillo . . . 100
6.10. Red con un contador . . . 101
6.11. Simulación de la red de la figura 6.10 mediante una ν-APN . . . 102
6.12. Algoritmo de decisión de la acotación . . . 107
6.13. ν-APN acotada en anchura pero no en profundidad . . . 108
6.14. ν-APN acotada en profundidad pero no en anchura . . . 109
6.15. Algoritmo de decisión de la acotación en anchura . . . 110
6.16. Árbol de Karp-Miller de la ν-APN de la figura 6.13 . . . 113
6.17. APN que simula la ν-APN de la figura 6.13 . . . 115
6.18. Otra ν-APN acotada en anchura . . . 115
6.19. Árbol de Karp-Miller de la ν-APN de la figura 6.18 . . . 115
6.20. APN que simula la ν-APN de la figura 6.18 . . . 116
6.21. Resumen de resultados de decidibilidad . . . 117
6.22. Simulación de una ν-APN (izquierda) por una $\nu_{\neq}$-APN (derecha) . 118
6.23. Red con arcos de transferencia . . . 120
6.24. Simulación débil de la red con arcos de transferencia . . . 120
6.25. Red con arcos reset . . . 123
6.26. Simulación débil de la red reset . . . 123

7.1. Sistema ubicuo modelado con RUNs . . . 128
7.2. De los sistemas MSPN extendidos a los sistemas MSPN . . . 135
7.3. Transiciones no activadas . . . 138
7.4. Simulación de transiciones de replicación . . . 139
7.5. Simulación de transiciones de replicación con recogida de basura . 142
7.6. Monitor . . . 146
7.7. APN sin creación de nombres . . . 149
7.8. Simulación de la APN de la figura 7.7 por medio de sistemas RUN con recogida de basura . . . 150
7.9. ν-APN y la RUN que la simula . . . 152
7.10. Celda de la cinta de una máquina de Turing . . . 155
7.11. Máquina de Turing que computa 0101... . . . 155
7.12. El orden natural de los sistemas ν-RN no es un wqo . . . 156

8.1. Árboles de sintaxis abstracta de $1 + (0 \times 1)$ y $1 \oplus (0 \boxtimes 1)$ . . . 161

8.2. Ejemplo de sistema de transiciones 5 161
8.3. Sistemas etiquetados del Ejemplo 5 162
8.4. Más sistemas etiquetados del Ejemplo 5 163
8.5. Sistema de transiciones (izquierda) y sistemas etiquetados (derecha) del ejemplo 6 164
8.6. Diagrama conmutativo inducido por un sistema etiquetado 167
8.7. Grafo de transiciones (izquierda) y árbol de alcanzabilidad (derecha) del ejemplo 7 168
8.8. Dos sistemas cociente del ejemplo 7 169
8.9. Sistemas cociente isomorfos del ejemplo 8 169
8.10. Sintaxis del $D\pi$ 174
8.11. Equivalencia estructural del lenguaje $D\pi$ 174
8.12. Semántica del lenguaje $D\pi$ 175
8.13. Semántica del $D\pi$ etiquetado 176
8.14. Operadores O_1, O_2 y $\oplus$ 179
8.15. Mezcla de sistemas etiquetados 181
8.16. Retículo extendido de sistemas cociente 182

9.1. Ventanilla de billetes y agente 185
9.2. Teatro 185
9.3. Posible $(\{k,l\},\mathcal{N})$-entorno para el ejemplo de la sección 9.1 190
9.4. Posible cliente 192
9.5. Restricción de $\mathcal{P}$ a lugares de $P_\bullet$ 196
9.6. Red del ejemplo 11 198
9.7. $(\emptyset,\mathcal{S})$-entorno $\mathcal{P}$ para la traza del ejemplo 11 199
9.8. Sistema MSPN y $\{a\}$-entorno que implementa su semántica simbólica 203

10.1. Ejemplo de APN 208
10.2. Una ν-APN 215
10.3. Ejemplo de ν-APN 217
10.4. Invariante de `cover` 223
10.5. CPN Tools 224
10.6. Sistema MSPN 225
10.7. ν-APN que simula al sistema MSPN de la figura 10.6 225

11.1. La política del pato que resucita 230
11.2. Enfermera 235
11.3. Controlador de termómetros 236
11.4. Pacientes 236
11.5. Árboles de dependencia 237
11.6. Ejemplo simple de sistema TSA 238
11.7. Traducción del sistema TSA de la figura 11.6 239
11.8. Sistema TSA 242
11.9. Implementación de sincronizaciones 242

11.10 Implementación de impresiones 243
11.11 Implementación de transiciones *kill* 243
11.12 Implementación del sistema TSA de la figura 11.8 248
11.13 Entornos para el sistema TSA de la figura 11.8 248
11.14 Implementación del sistema TSA de la figura 11.8 249

12.1. Diagrama de resultados 254

Capítulo 1

Introducción

Las nuevas tecnologías están cambiando muy rápidamente la manera como usamos los ordenadores. Antiguamente muchos usuarios compartían unos pocos ordenadores muy caros. Unos años después aparecieron los primeros ordenadores personales, de modo que los ordenadores estaban a total disposición de sus dueños. La aparición de la primera *red de área local* (LAN) hizo posible la interconexión de distintos ordenadores, haciendo por tanto posible el acceso de los usuarios a muchos ordenadores, aunque comprometiendo la seguridad y la disponibilidad de estos. Al mismo tiempo, para explotar las posibilidades ofrecidas por la creciente cantidad de recursos disponibles, aparecieron nuevas formas de programar. Así, a la programación *secuencial* le siguieron la programación *concurrente* y la *distribuida*.

Las principales características de las redes locales se pueden resumir en las siguientes: los ordenadores tienen aproximadamente la misma potencia, las conexiones entre los distintos nodos tienen aproximadamente el mismo ancho de banda, los retrasos en las comunicaciones están acotados (los nodos se pueden detectar haciendo "ping") y están administradas y protegidas por un cortafuegos.

Sin embargo, cuando se habla de una *red de área amplia* (WAN), como la *Web*, las propiedades anteriores han venido a llamarse las *falacias de la computación distribuida*. Las WAN tienen, por tanto, propiedades distintas: no están administradas de manera centralizada, los ordenadores difieren mucho tanto en potencia como en disponibilidad, las distancias físicas tienen efectos visibles, la topología de la red es dinámica y los ordenadores se vuelven intrínsecamente móviles [Car99].

En particular, al referirnos en concreto a la Web, los siguientes aspectos de las redes se vuelven observables:

- **Localidades virtuales**. No se puede abstraer de los cortafuegos ni de otro tipo de protecciones cuando queremos atravesarlos, por lo que las localidades virtuales han de ser visibles.

- **Localidades físicas**. En un contexto de computación global, la velocidad de la luz impone una cota inferior absoluta en los retrasos de las comunica-

ciones, lo que induce inevitablemente una noción de localidad física.

- **Fluctuaciones en el ancho de banda.** La congestión de la red produce grandes diferencias en el rango de los retrasos de las comunicaciones. Los programas deberían poder observar y reaccionar ante estas fluctuaciones.

- **No se pueden observar los fallos.** Los retrasos en la Web pueden ser arbitrariamente largos y, por tanto, no podemos concluir que un servidor no existe por el hecho de que no responda. Ello nos podría llevar, por ejemplo, a intentar acceder a él por medio de un camino alternativo.

Para programar en la web, necesitamos tener en cuenta todas estas características. La *computación móvil* (*mobile computation* en inglés) es una técnica surgida para hacerles frente. Se basa en la idea de que los programas no tienen por qué residir en una localidad fija para ejecutarse (software móvil). La computación móvil consigue lidiar con las características anteriores, permitiendo que los programas puedan cruzar barreras, convertir llamadas remotas a procedimientos en llamadas locales, tener en cuenta las fluctuaciones en el ancho de banda y reaccionar ante posibles fallos.

Pero existe una segunda noción de movilidad, conocida en inglés como *mobile computing*, que tiene que ver con la movilidad física, es decir, con el hardware móvil.

1.1. Computación ubicua

Un paso más allá es el de la *computación ubicua* (*ubiquitous computing* en inglés), también llamada *computación ubicua global*, o por el acrónimo *Ubicomp*. La idea de la computación ubicua se la debemos a Mark Weiser, investigador de *Palo Alto Research Center* (PARC), de Xerox, en el que ha venido a considerarse el artículo seminal de la computación ubicua, *The computer for the 21st century* [Wei91]. En él se sostiene que toda tecnología alcanza el grado de madurez a partir del momento en el que se vuelve invisible, es decir, cuando sus usuarios dejan de verla como tecnología y pasan a integrarla de manera natural como herramienta en su vida diaria. Así ocurrió, por ejemplo, con la primera tecnología de la información: la escritura. Cuando aparecieron los primeros sistemas de representación simbólica, sólo los grandes contratos comerciales o los tratados jurídicos eran grabados, ya fuese en trozos de arcilla, conchas de tortuga o, más tarde, en papiro. Sin embargo, hoy en día, la escritura es una tecnología ubicua, que nos rodea allá donde quiera que vayamos, de la que nos aprovechamos constantemente y que pasa desapercibida (en ese sentido, ha desaparecido). En su artículo, Weiser afirma que el futuro de la informática también está en su "desaparición", previendo ya en los ochenta un futuro en el que nos rodearían un gran número de pequeños ordenadores invisibles empotrados en nuestro ambiente, interaccionando con usuarios que se mueven en él. Los usuarios experimentarían este entorno

por medio de una gran variedad de dispositivos, que interaccionarían con sensores inteligentes para componer un entorno de computación móvil y ubicua, que nos ayudaría en nuestras actividades. Entonces, será muy importante el uso de comunicaciones inalámbricas para sostener esas interacciones entre dispositivos móviles, aunque el entorno también podrá tener acceso a una WAN cableada, de manera que los conceptos que surgirían en ellas también se aplicarían aquí.

Características específicas de los sistemas ubicuos son:

- **Unidades con poca capacidad de cómputo:** la computación ubicua supone que estaremos rodeados de unidades de cómputo empotradas en pequeños dispositivos. Por tanto, no podemos suponer que estas unidades tengan gran capacidad de cómputo, no sólo por su tamaño, sino por la inviabilidad de acoplar una batería suficientemente duradera en ellos.

- **Redes ad hoc:** en la computación ubicua no es realista suponer la existencia de una infraestructura de red en cualquier parte. Por tanto, en este contexto son de gran importancia las redes ad hoc móviles (MANET), es decir, redes cuyos nodos son únicamente los propios dispositivos que se quieren aprovechar de esa red, con una conectividad altamente variable, de manera que la decisión de a quién mandar un paquete se ha de tomar dinámicamente.

- **Descubrimiento de servicios:** una tecnología clave en la computación ubicua es la del descubrimiento de servicios, que permite que una componente pueda describir, publicitar y descubrir servicios. Los mecanismos de descubrimiento de servicios más habituales, como UDDI, Jini o UPnP, están basados en una fuerte estandardización [Ric00].

La visión de Weiser está haciéndose poco a poco una realidad, sobre todo por los increíbles avances que se han hecho en los últimos años en el campo de la microelectrónica, que han permitido que se creen dispositivos empotrados cada vez más pequeños. Sin embargo, el estado de esta ciencia no está tan avanzado como se esperaba hace quince años [Wei93]. Una de las razones de ello puede ser que no existan modelos formales ampliamente aceptados, necesarios en distintos niveles de abstracción para comprender y ayudar a diseñar lo que, según Milner, *será sin duda el artefacto ingenieril más grande de la historia de la humanidad* [Mil04].

Milner hace un repaso en su artículo de los distintos problemas teóricos que se plantean cuando se especifican y analizan sistemas ubicuos. Estos problemas se pueden englobar dentro de las siguientes categorías:

- Computación móvil y distribuida,
- Seguridad y privacidad,
- Límites y confianza,
- Semánticas de juegos,

- Sistemas híbridos y estocásticos,
- Model checking.

Muchos de estos temas han sido extensamente estudiados durante la última década, y contamos con aproximaciones formales para tratarlas aisladamente. Además, los sistemas ubicuos presentan dificultades extra respecto a su coordinación. En primer lugar, el cambio continuo del estado de las distintas componentes que forman un sistema es inherente a la computación ubicua. Este hecho, junto con la naturaleza poco fiable de las redes ad hoc y la autonomía de las componentes hace de la orquestación una aproximación poco realista y robusta, de manera que la aproximación coreográfica es mucho más apropiada.

1.2. Seguridad

Existen muchos trabajos, artículos y libros sobre seguridad informática. El trabajo [RV04] constituye una fácil introducción a la seguridad informática, y contiene un gran número de bibliografía sobre el estado del arte, centrándose en la seguridad para sistemas con movilidad, punto de partida del tema que nos ocupa, y en distintos formalismos para su estudio. Una referencia mucho más exhaustiva es [And01].

Al igual que la seguridad en la vida real, la seguridad informática trata de garantizar la calidad de los programas y datos (integridad) y de proteger propiedades e intereses de robos o intrusiones (confidencialidad). Para conseguirlo en un entorno hostil, necesitamos restringir el acceso a nuestros activos, ofreciéndoselos únicamente a algunos, de manera que tenemos que saber en quién podemos confiar y verificar los credenciales (autenticar) de aquellos que nos visitan.

Así que la seguridad está basada en las siguientes propiedades:

- **Confidencialidad:** la capacidad de mantener algo en privado. Puede tratarse de cosas de muy distinta índole: datos y código, pero también la identidad del dueño o relaciones entre estas entidades, lo que a veces se denomina **anonimato**.
- **Confianza:** en quién, o en los datos de quién, podemos confiar. Al final, todos los problemas de seguridad se pueden reducir a un problema de confianza.
- **Autenticidad:** el problema de decidir si efectivamente estamos hablando con la persona/el ordenador con el que creemos que estamos hablando. La autenticación tiene dos caras [Aba98]: crédito (que tiene que ver con "quién mandó el mensaje") y autoridad (que tiene que ver con "quién es responsable de la emisión del mensaje").

- **Integridad:** algunas cosas no se pueden corromper. En la mayoría de los casos, la integridad y la autenticidad van de la mano: la integridad es un prerrequisito para la autenticidad (para comprobar la autenticidad de un mensaje debemos de estar seguros de que no ha sido modificado) y la integridad sin autenticidad no tiene sentido (normalmente sólo tenemos interés en comprobar que un mensaje no ha sido modificado cuando conocemos su origen).

Como decíamos, todos los problemas de seguridad dependen de en quién confiamos. Normalmente se introduce alguna tecnología para mover la confianza de un sitio arriesgado a otro más seguro. Por ejemplo, si un usuario se instala un antivirus es porque confía en la empresa que lo ha desarrollado, o si confiamos en un certificado es porque nos fiamos de quien lo ha emitido, como la Fábrica Nacional de Moneda y Timbre o Verisign. Este es uno de los aspectos de la seguridad que hacen que se convierta en un problema social.

Desgraciadamente, las propiedades de seguridad son muy resbaladizas y no existe un consenso general en sus definiciones. Como se puede ver con la autenticidad, los diferentes aspectos que concurren en las propiedades de seguridad, hacen que las descripciones informales no sean suficientes para su especificación, no ya precisa, sino simplemente inteligible.

Los entornos de un sistema informático pueden ser hostiles por muy diversos motivos: amenazas físicas como desastres naturales o apagones, amenazas humanas como espionaje o sabotajes, y amenazas de software como virus y troyanos. Además, es preciso tener en cuenta que la seguridad aumenta la complejidad de los sistemas, y los hace inevitablemente menos eficientes y menos funcionales (más difíciles de usar correctamente).

En la computación ubicua la seguridad es un tema trascendental, como muy bien refleja el cuento corto de Bruce Sterling titulado *Maneki Neko*, que aparece en la recopilación *A good old-fashioned future* [Ste99]. En él un virus consigue hacerle la vida imposible al protagonista, en un mundo donde la tecnología es ubicua y silenciosa, o debería serlo.

La computación ubicua tiene una serie de características que hacen que los problemas de seguridad sean distintos a los de los sistemas tradicionales. Las características que más influyen en este sentido son las comunicaciones predominantemente inalámbricas, la poca capacidad de cómputo de muchos microdispositivos y la escasa disponibilidad de energía de estos dispositivos, que generalmente cuentan con baterías limitadas. Las comunicaciones inalámbricas hacen a estos sistemas más sensibles a la intercepción pasiva de las comunicaciones, sin necesidad de violentar físicamente las conexiones entre los participantes. Por el mismo motivo, estos sistemas son también más sensibles a ataques contra la integridad de las comunicaciones.

No podemos suponer que todos los dispositivos que forman parte de nuestro entorno estén dotados de una potente CPU, capaz de realizar rápidamente costosas operaciones. Los algoritmos de cifrado de clave pública, como RSA [MVO96]

requieren complejas operaciones, inasumibles para estos aparatos. Por ello, tendremos que conformarnos en general con una seguridad basada en criptografía simétrica. Además, no podemos suponer la existencia de una infraestructura de clave pública o de terceras partes de confianza que estén siempre accesibles, debido a la escasa fiabilidad de las redes ad hoc. Por todo ello, el problema de la autenticidad no se puede abordar en la forma tradicional.

Otra característica de la computación ubicua con influencia en su seguridad son los dispositivos desatendidos (*unattended devices*). La visión de la computación ubicua de que existan aparatos desatendidos, que puede usar cualquiera que acceda a ellos, abre la puerta a ataques de integridad contra esos aparatos o a ataques de denegación de servicio (por ejemplo, si se permite que acceda a ellos código móvil que los bloquee).

Por último, debido a que la computación ubicua prevé que la tecnología abarcará todos los aspectos de nuestra vida, cobra especial importancia el asunto del anonimato, que se preocupa no de que el contenido de las comunicaciones permanezca secreto, sino de que lo hagan los propios interlocutores de estas comunicaciones. De lo contrario, la computación ubicua podría derivar en un represivo Gran Hermano ubicuo.

1.3. Redes de Petri y extensiones

Las redes de Petri aparecieron por primera vez en 1962, en la tesis doctoral de Carl Adam Petri [Pet62]. Una red de Petri es básicamente una generalización distribuida de los autómatas finitos, con una intuitiva y sencilla representación gráfica. En una red de Petri el estado global se especifica por medio de la combinación de una serie de estados locales, lo que las hace muy adecuadas para la representación de sistemas distribuidos. En los llamados *sistemas elementales* cada uno de los estados locales, llamados *lugares*, puede estar en uno de dos estados. Por tanto, n lugares en un sistema elemental pueden representar 2^n configuraciones distintas. Gráficamente, los lugares se representan mediante un círculo. Estos círculos pueden contener un punto (un *token*) o no, obteniéndose así los dos posibles estados en los que se pueden encontrar.

Los cambios de estado en las redes de Petri se conocen como *transiciones*. Las transiciones cambian la configuración global modificando solamente algunos estados locales, es decir, el estado de algunos lugares, quitando el token de algunos lugares con token y poniendo un token en algunos lugares que no lo tienen. Intuitivamente, se puede entender que los lugares de los que se quita un token representan condiciones que se tienen que satisfacer para que pueda ocurrir la transición. Por ello, a esos lugares se les llama *precondiciones* de la transición. Análogamente, los lugares donde se pone el token se les llama *postcondiciones* de la transición. Además, varias de estas transiciones pueden ocurrir (o, en terminología de redes de Petri, se pueden disparar) de manera concurrente, siempre que afecten a conjuntos disjuntos de lugares. En general, los disparos son no de-

terministas, lo que las hace apropiadas para el estudio de sistemas concurrentes. En particular, el disparo de una transición con varias precondiciones se puede ver intuitivamente como una sincronización entre varios autómatas.

Gráficamente, las transiciones se representan mediante rectángulos. Un lugar será precondición de una transición si hay una flecha del lugar a la transición, y será una postcondición si la flecha va de la transición al lugar. Por tanto, una red de Petri es en realidad un grafo bipartito dirigido.

Los sistemas elementales comentados no son una extensión semántica de los autómatas finitos. Sin embargo, éstos son fácilmente generalizables sin más que considerar que los lugares no puedan estar simplemente en uno de dos estados, sino en un número arbitrario de estados, representados por el número de tokens que contienen. A esta clase de redes de Petri se les llama redes Lugar/Transición, o simplemente P/T por *Place/Transition*. Evidentemente, las redes P/T pueden llegar a representar sistemas con infinitos estados. Sin embargo, las redes P/T satisfacen una propiedad de monotonía, que se puede resumir en que *con más tokens siempre se pueden hacer más cosas*. Este hecho, así como la imposibilidad de implementar mecanismos generales de elección `IF THEN ELSE`, hace que existan para ellas un gran número de resultados positivos de decidibilidad, que no se dan en otros modelos de cómputo.

Existen un gran número de extensiones de las redes de Petri. Algunas extensiones van precisamente en la línea de flexibilizar el concepto de activación y disparo de transiciones. Por ejemplo, las redes *con arcos inhibidores* pueden chequear si un lugar está vacío, lo que les da la potencia de las máquinas de Turing (una vez contemos con al menos dos de dichos arcos). Una extensión similar es la que viene dada por los *arcos de transferencia*, que mueven todos los tokens que se encuentren en un lugar a otro, o los *arcos reset* o de puesta a cero, que eliminan todos los tokens que haya en un lugar.

Otras generalizaciones extienden la naturaleza de los tokens que pueden contener los lugares. La más conocida es la de las *redes de Petri coloreadas* [Jen96]. En ellas los tokens dejan de ser indistinguibles, portando un valor de un cierto tipo, de manera que las transiciones no sólo eliminan los tokens de las precondiciones y los crean en las postcondiciones, sino que transforman su valor. Es fácil ver que las redes de Petri coloreadas son Turing completas, ya que una transición podría transformar la entrada de un token precondición según cualquier función recursiva, para producir un token con el correspondiente valor en su postcondición.

En los últimos años han aparecido algunos modelos basados en redes de Petri para tratar problemas de movilidad y de programación orientada a objetos o a agentes. Probablemente el primero de ellos se encuentra en [AB96], donde los autores definen las *redes de Petri móviles*, muy inspiradas en el cálculo *Join* [FG96, FGL$^+$96]. Se introduce una conectividad dinámica en sus redes por medio de colores. Más específicamente, los valores de los tokens son tuplas de nombres de lugares, de manera que los tokens en las precondiciones especifican su propio destino por medio de su valor. De esta manera, cualquier lugar es postcondición potencial de cualquier transición, lo que por un lado hace que las Redes de Petri

móviles sean imposibles de dibujar y, lo que es más importante, hace que se pierda el sentido local de las transiciones.

De entre los trabajos basados en redes para el estudio de la movilidad cabe destacar los *sistemas elementales de objetos* [Val98, Val01]. Los sistemas elementales de objetos se componen de una red sistema y varias redes objeto, que se mueven por la red sistema como si fuesen tokens normales. Estas redes token pueden cambiar su marcaje, pero no su estructura, tanto cuando están en un lugar de manera autónoma o cuando se mueven por el disparo de una transición de la red sistema.

Las *redes de Petri anidadas* [LS99, Lom01a, Lom01b, Lom02] también son un modelo jerárquico, con redes token. En este caso, las redes token se pueden crear, copiar y eliminar durante la ejecución del sistema, como se indica en las etiquetas de los arcos. El número de estos tokens, así como el nivel de anidamiento, es arbitrario, obteniéndose sistemas anidados multinivel.

Otra extensión multinivel de los sistemas elementales de objetos son las *redes de Petri ambientales* [FEA03, FEMA03], que permiten el anidamiento arbitrario del cálculo de ambientes [CG98], del que hablaremos con más detalle en la próxima sección. Como consecuencia, es posible encontrar como tokens de un lugar tanto tokens ordinarios como tokens de alto nivel. Estos segundos se mueven por la red según las capacidades obtenidas del conocimiento de nombres.

En [Lak05] se presenta un modelo para la movilidad basado en redes de Petri, basándose en la fusión de lugares y transiciones. Sin embargo, no hemos encontrado trabajos que estudien las propiedades y la capacidad expresiva de este modelo. En [BBPP04] los autores se apartan de la movilidad basada en nombres, como ocurre en [AB96], y sigue la línea del paradigma de *redes dentro de redes* [Val03]. Sin embargo, asumen que todas las redes sistema son *máquinas de estados*, para mantener la identidad de las redes token. Por lo tanto, el espacio de estados de una red sistema es finito, por lo que su capacidad expresiva es limitada. En [HR04] se presentan las redes PEPA, donde PEPA son las siglas de *Performance Evaluation Process Algebra* [GHKR04], un álgebra de procesos estocástica para la evaluación de rendimientos. Este álgebra de procesos se usa como lenguaje de inscripciones de los arcos de redes de Petri estocásticas.

1.4. Álgebras de procesos: nombres y replicación

El mundo de la concurrencia ha estado tradicionalmente dividido (en el buen sentido) en dos comunidades, la de redes de Petri y la de álgebras de proceso. Las álgebras de procesos son lenguajes formales para el estudio fundacional de los lenguajes de programación, con el foco puesto en los problemas asociados a la concurrencia como pueden ser el no-determinismo y los interbloqueos. Las primeras álgebras de procesos, como CSP [Hoa85] o CCS [Mil82], consideran únicamente conectividad estática, de modo que el conjunto de canales que cada proceso puede usar para comunicarse con los demás procesos no cambia durante su ejecución.

El cálculo π [MPW92] es una extensión de CCS en la que los propios canales (sus nombres) son susceptibles de ser comunicados, así que los procesos pueden adquirir nuevos canales, que pueden usar en futuras comunicaciones. El cálculo π se ha configurado como el modelo canónico en la computación concurrente, del mismo modo que el λ-cálculo lo hizo en la programación funcional. En él cobra vital importancia el concepto de *nombre puro*, que según Needham "no es nada más que un patrón de bits usado como identificador (...) para compararlo con otros" [Nee93]. Quizás la parte más importante de la definición sea la de *nada más*, sugiriendo que un nombre puro no puede tener una estructura significativa susceptible de ser reconocida, como podría ser una tupla, la dirección absoluta de un fichero o el resultado de cifrar un mensaje con una clave.

En el cálculo π los nombres de los canales son nombres puros en el sentido anterior. También se pueden crear nuevos nombres por medio del operador de restricción $(\nu\eta)P$, que crea el nuevo nombre η con ámbito P. Aquí el nombre η actúa en realidad como un "agujero" en el que se puede colocar cualquier nombre pues, intuitivamente, lo único que nos importa del nuevo nombre es que sea, efectivamente, nuevo. Esto se formaliza asumiendo una noción de equivalencia que permite el renombramiento (consistente) de los nombres que aparecen bajo el operador ν. Esto permite además obtener la composicionalidad que normalmente persiguen las álgebras de procesos mediante la *regla de extrusión* (como si los nombres fueran fluidos maleables), que afirma esencialmente que los procesos siempre se pueden componer paralelamente, siempre que los nombres creados sigan siendo nuevos, lo que se consigue forzando que los agujeros que hay en cada uno de ellos sean distintos. La regla de extrusión queda formalizada por la siguiente equivalencia entre procesos:

$$P \mid (\nu\eta)Q \equiv (\nu\eta)(P \mid Q)$$

si $\eta \notin fn(P)$, donde $fn(P)$ es el conjunto de variables libres de P. Dos procesos concurrentes pueden comunicarse siempre que uno de ellos esté dispuesto a enviar un nombre por un canal y el otro esté dispuesto a recibir un nombre a través del mismo canal:

$$a?(x).P \mid a!\langle b\rangle.Q \rightarrow P\{x := b\} \mid Q$$

Esta regla implementa una disciplina de paso de mensajes síncrona, ya que la guarda del proceso que envía el mensaje está bloqueada hasta que no se produce la comunicación. En [Bou92] se prueba que esta primitiva de comunicación síncrona puede ser implementada en una versión asíncrona del lenguaje, en la que no se permiten guardas detrás de los mensajes.

Tras el cálculo π la literatura se ha llenado de ejemplos de *cálculos nominales*, como se llama en [Gor00] a las álgebras de procesos en las que los nombres puros tienen un papel relevante. En general, estas álgebras son variantes o extensiones del cálculo π en las que se modifica o se sustituye el principio de que el nombre de los canales es primitivo. Así, en el lenguaje $D\pi$ [HR02b], o π cálculo distribuido, los canales son locales a ciertas localidades, cuyo nombramiento es también primitivo.

Así, un canal a puede aparecer en dos localidades k_1 y k_2 y ser, al efecto, distintos, pues dos procesos han de compartir canal y localidad para poder comunicarse. Así, la regla principal del cálculo viene dada por

$$k[\![a?(x).P]\!] \mid k[\![a!\langle b\rangle.Q]\!] \rightarrow k[\![P\{x := b\}]\!] \mid k[\![Q]\!]$$

Otra extensión del cálculo π es el cálculo de ambientes [AG97]. En él los únicos nombres primitivos son los de los ambientes, que se pueden interpretar en sentido amplio como espacios acotados de cómputo. La diferencia esencial entre las localidades del lenguaje $D\pi$ y los ambientes es que los últimos definen una topología de red jerárquica y dinámica. A cambio, se abandona el concepto de canal, y se asume que dos procesos en el mismo ambiente siempre pueden comunicarse.

En general, las primeras versiones de estos cálculos son Turing completos, así que el esfuerzo se ha enfocado en definir análisis estáticos de propiedades indecidibles [CGG02, NHN03] y en la restricción de los formalismos originales [BZ05, Zim00, Zim03] para conseguir lenguajes más manejables, con interesantes propiedades decidibles.

El estado del arte está lleno de intentos de acercar el mundo de las redes de Petri con el de las álgebras de procesos. En [DKK04] los autores definen una semántica de redes de Petri para el cálculo π y, por tanto, tienen que tratar con nombres locales, aunque sólo en el caso finito. Se centran en el tratamiento de nombres locales de manera composicional, aunque su solución usa de una manera un tanto ad hoc un único lugar *tag-place* para representar al entorno global.

Otra traducción del cálculo π a las redes de Petri se puede encontrar en [BG95], donde usan redes de Petri con arcos inhibidores. Sin embargo, la composicionalidad se consigue de manera que la traducción puede resultar en una red con infinitos lugares y transiciones. Esta infinitud era también el problema de muchas de las primeras semánticas basadas en redes de Petri para álgebras de procesos, como [Tau89]. Para evitar esta característica indeseable, E. Best, R. Devillers, M. Koutny et al. desarrollaron el *cálculo de cajas*, o *Petri Box Calculus* (PBC) [BDK01, KEB94], donde el operador paralelo clásico se separa en un operador de intercalamiento o *interleaving* y un operador de sincronización unario, lo que permite evitar el problema mencionado. En [FEA03] los autores extienden este modelo obteniendo APBC, que incorpora para PBC el concepto de ambiente.

Por otro lado, se ha dedicado mucho esfuerzo en los últimos años al estudio de los programas multihebra [Rin01]. En general, una vez que se permite sincronización entre hebras la alcanzabilidad se vuelve indecidible [Ram00], de manera que el estudio se ha dirigido a calcular sobreaproximaciones del conjunto de marcajes alcanzables. Por ejemplo, [BCR01] estudia el caso en el que las hebras son de finitos estados y [BET03] considera un marco para analizar estáticamente sistemas multihebra con un número fijo de hebras, posiblemente de infinitos estados, que sincronizan entre sí.

Un formalismo que cuenta tanto con generación dinámica de nombres y replicación es TDL [Del], que es una extensión con nombres de [BCR01]. En TDL hay

primitivas tanto para la creación de nombres como para la creación de hebras. Sin embargo, no se considera ningún mecanismo de recogida de basura, ni para nombres ni para hebras. Consecuentemente, y de modo un poco sorprendente, la alcanzabilidad para TDL resulta ser decidible, mientras el recubrimiento (o alcanzabilidad de control) es indecidible.

1.5. Objetivos y organización de la tesis

El objetivo de este trabajo es desarrollar un modelo que nos permita estudiar algunas de las características de la computación ubicua. Nos centraremos en los problemas derivados de la coordinación y la seguridad, dedicando especial atención al carácter abierto de los sistemas ubicuos. Dicho modelo estará basado en redes de Petri. Esto quiere decir que queremos mantener tres de las características que hacen a nuestro juicio de las redes de Petri un formalismo tan útil: localidad y monotonía de las acciones, representaciones gráficas y la falta de Turing completitud.

Es decir, queremos definir un modelo que permita tratar problemas de coordinación y seguridad, fiel al estilo de las redes de Petri, en el sentido de que el disparo de una transición sólo tenga consecuencias en el conjunto de lugares adyacentes, y que puedan ser representados cómodamente de manera gráfica. El mayor desafío es conseguir mantener el carácter no Turing completo del formalismo obtenido, lo que nos proporcionaría un gran número de resultados positivos en cuanto a la decidibilidad de propiedades interesantes, lo que a su vez permitiría abordar la verificación de los sistemas especificados en él.

Precisamente para evitar la Turing completitud hemos decidido que el concepto de localidad que incorporaremos a nuestros formalismos esté más aproximado al que aparece en el cálculo π distribuido que al que aparece en el cálculo de ambientes, ya que la estructura jerárquica de los ambientes hace que, incluso aunque el nombre de los ambientes sea puro, la estructura de la jerarquía no lo sea, lo que conduce inmediatamente a la Turing completitud [AG97].

Hemos optado por una presentación incremental, lo que clarifica la presentación de nuevos conceptos, pero en ocasiones nos obligará a repetir algunas definiciones, ya que la mera modificación puntual o la simple extensión de algún detalle podría llevarnos en ocasiones a que la definición obtenida fuera incorrecta o incompleta, si bien haremos especial hincapié en los nuevos aspectos que aparezcan. Alternativamente, podríamos haber definido un modelo general desde un principio, de manera que resultara sencillo ir incorporando las distintas características, pero ello complicaría irremediablemente el modelo inicial y, por consiguiente, su comprensión. Además, la opción elegida tiene la ventaja de hacer más autocontenidos cada uno de los capítulos.

A lo largo de la tesis iremos definiendo los distintos modelos, presentando los conceptos con la ayuda de pequeños ejemplos. Prestaremos especial atención a las propiedades de los formalismos introducidos, sobre todo aquellas que sean de

interés para la seguridad de los sistemas, así como a la relación existente entre ellos y a la transferencia de propiedades de unos a otros. Si bien el aspecto teórico será el fundamental del trabajo, no perdemos de vista la aplicabilidad práctica con la presentación de ejemplos más elaborados y con la implementación de una herramienta para la ejecución y análisis de algunos de los modelos presentados.

El trabajo está organizado en base a los siguientes capítulos. En el capítulo 2 introduciremos brevemente las nociones fundamentales que se utilizarán a lo largo de la tesis, así como las notaciones que usaremos en ella.

En el capítulo 3 definimos el modelo básico para el estudio de sistemas ubicuos, que nos servirá como punto de partida. Presentaremos un ejemplo de aplicación del modelo y probaremos el resultado que establece la equivalencia entre el modelo definido y las redes P/T. Concluiremos el capítulo con algunas posibles extensiones del modelo que, sin embargo, no introducen nueva capacidad expresiva. El contenido de este capítulo se ha presentado en [FERVMA05].

El capítulo 4 extiende de manera no trivial el modelo. En una primera extensión, se introducen colores para establecer un mecanismo dinámico de movilidad. Tras probar que esta extensión permanece todavía equivalente a las redes P/T, se añaden primitivas de autenticación, que permiten la creación de identificadores nuevos y la restricción de las interacciones entre componentes en base a estos identificadores. Esta última extensión se realiza de dos maneras ortogonales. Primero mediante la introducción de tokens naturales y lugares "contadores", que pueden ser manipulados por medio de unas transiciones especiales. La manipulación de los tokens naturales estará muy restringida. Las transiciones ordinarias sólo podrán mover o eliminar estos tokens, sin modificar su estructura. Las transiciones especiales de las que hablábamos sólo manipularán el lugar contador, y más concretamente sólo podrán sustituir el token que contengan por su sucesor. Esto es suficiente para el propósito que nos ocupa, que no es otro que usar estos tokens como identificadores que permitan la autenticación. Además, de esta manera evitamos la posibilidad de simular contadores (en el sentido de las máquinas de contadores [Min67]), lo que nos llevaría irremediablemente a la Turing completitud.

Después extenderemos el modelo básico mediante identificadores arbitrarios y transiciones que generan nombres frescos, es decir, que no aparecen en ningún otro lugar del marcaje. Este mecanismo está más cerca del de los cálculos nominales discutidos en la sección anterior que el que usa naturales. Sin embargo, la falta de una primitiva de creación de ámbitos en nuestro modelo provoca importantes diferencias respecto a lo que ocurría allí, sobre todo en lo relativo a la noción de α-equivalencia inducida, así como en lo relativo a la composicionalidad. Este capítulo se corresponde con lo presentado en [RVMAFE06].

El breve capítulo 5 allana el camino para el estudio de la expresividad de los modelos con creación de nombres, eliminando de ellos todas las características no esenciales, que no afecten a su expresividad.

El capítulo 6 es uno de los capítulos centrales de este trabajo, donde se presentan los resultados publicados en [RVFEA07] y [RVFE07]. En él se establecen una

serie de resultados sobre la decidibilidad de varias propiedades para las dos versiones de creación de nombres. En particular, se prueba que tanto el recubrimiento como la alcanzabilidad son decidibles para el caso de identificadores naturales. En el caso de identificadores arbitrarios, se pierde la decidibilidad de la alcanzabilidad, pero se conserva la del recubrimiento. Al introducir los nombres se crea una nueva dimensión de infinitud en el modelo, ya que los tokens no sólo pueden aparecer una cantidad no acotada de veces, sino que puede haber una cantidad no acotada de tokens distintos. Este hecho origina distintas nociones de acotación, que serán estudiados también en este capítulo.

En el siguiente capítulo se vuelve a extender el modelo básico con una primitiva para la replicación de componentes. Se trata por tanto de un modelo con creación dinámica y no acotada de componentes (con un estado inicial fijo), en el que cada componente, que puede interactuar con las demás, puede tener infinitos estados. Se prueba la decidibilidad, tanto del problema de alcanzabilidad como la del recubrimiento. Después se añade un mecanismo de recolección de basura, que elimina de los marcajes las componentes sin tokens, de modo que también tenemos destrucción dinámica de componentes. Se prueba que el modelo con recogida de basura es equivalente al modelo sin replicación pero con creación de nombres abstractos. Por tanto, la alcanzabilidad para él es indecidible, mientras que el recubrimiento se mantiene decidible. Por último, se consideran simultáneamente ambas primitivas, de replicación y creación de nombres abstractos. En este caso se prueba que, aunque ambas eran equivalentes entre sí, cuando se consideran juntas se alcanza la Turing completitud, con lo que el recubrimiento se vuelve también indecidible para ellas. El contenido de este capítulo se puede encontrar en [RVFEMA06] y en [RVFE07].

El capítulo 8 introduce un tecnicismo, presentado en [RVSFE05], que será de utilidad en el capítulo siguiente. Se trata del concepto de sistema etiquetado derivado de un sistema de transiciones dado, que se puede usar para el estudio de propiedades del sistema original que dependen de la historia. Se define el concepto de sistema etiquetado y se prueba un resultado semántico, que afirma que los sistemas etiquetados son cocientes del árbol de alcanzabilidad generado por el sistema original, lo que justifica la corrección del concepto introducido.

En el capítulo 9 se considera el estudio de nuestros sistemas en presencia de un entorno arbitrario. Para ello se define una semántica abierta, que no es más que la semántica original cuando se considera al sistema en presencia de un entorno arbitrario. Para hacer más manejable esta definición, se define una semántica simbólica, que considera al entorno de manera simbólica, etiquetando los marcajes con la información aprendida por este. La corrección de la semántica simbólica se prueba gracias a los resultados del capítulo anterior, es decir, comprobando que la semántica simbólica es un etiquetado de la semántica abierta. Por último, se prueba que también se tiene la decidibilidad del recubrimiento para la semántica abierta. El contenido de este capítulo se presentó en [RVFE06].

El capítulo 10 aborda la implementación de un prototipo para la especificación y verificación de nuestros sistemas, tal y como se presentó en [RV07a]. La

implementación se hace en lógica de reescritura [MOM02] y Maude [CDE+02]. La primera parte del capítulo trata de la traducción de nuestros sistemas a teorías de reescritura. La segunda parte es una descripción de varias decisiones de implementación, así como del interfaz con el usuario elegido.

En el capítulo 11 se presenta una aplicación para el estudio de sistemas que utilizan la *asociación transitoria segura*, más específicamente a través de la *política del pato que resucita*. Para ello se modifica ligeramente el modelo básico con primitivas de esta política. Tras probar que estas especificaciones son equivalentes a las redes P/T, se prueba que se pueden implementar mediante el mecanismo de creación de nombres presentado en el capítulo 4, tanto para sistemas cerrados como abiertos. En cualquier caso, gracias a los resultados vistos a lo largo de la tesis, se tiene la decidibilidad del recubrimiento. Este capítulo se presentó en [RV07b].

Por último, como no podía ser de otra manera, en el capítulo 12 resumimos las conclusiones más relevantes extraídas de la tesis.

Capítulo 2

Preliminares

En esta sección presentaremos diversos conceptos básicos que usaremos a lo largo de esta tesis, ya asentados de sobra en la literatura científica sobre estos temas, y fijaremos algunas notaciones.

2.1. Preórdenes y multiconjuntos

Dado un conjunto A, un *preorden* o *cuasiorden* sobre A es una relación binaria reflexiva y transitiva entre elementos de A. Llamaremos núcleo de un preorden $\leq$ al conjunto de pares (a, b) tales que $a \leq b$ y $b \leq a$. Escribiremos $a < b$ si $a \leq b$ y $b \not\leq a$. Un preorden es un orden parcial si es antisimétrico.

Si $(A, \leq)$ es un preorden y $B \subseteq A$, diremos que un elemento $a \in B$ es minimal en B si $b \in B$ con $b \leq a$ implica $a \leq b$. Denotaremos mediante $min(B)$ al conjunto de elementos minimales en B. Un ideal es un conjunto I cerrado hacia arriba, es decir, tal que si $a \in I$ y $a \leq b$, entonces $b \in I$. Todo conjunto J induce un ideal $\mathcal{C}(J) = \{b \in A \mid a \in J,\ a \leq b\}$.

Llamaremos multiconjunto de elementos de A a cualquier función $\mathcal{A} : A \to \mathbb{N}$. Escribiremos $\mathcal{MS}(A)$ para denotar al conjunto de multiconjuntos de elementos de A. Llamaremos soporte de $\mathcal{A}$ al conjunto $S(\mathcal{A}) = \{a \in A \mid \mathcal{A}(a) > 0\}$. Diremos que un elemento $a \in A$ tiene *multiplicidad* $\mathcal{A}(a)$. Un multiconjunto $\mathcal{A}$ es finito si $S(\mathcal{A})$ es finito, en cuyo caso definimos su *cardinal* como $|\mathcal{A}| = \sum\limits_{a \in S(\mathcal{A})} \mathcal{A}(a)$. Dados dos multiconjuntos de A, $\mathcal{A}_1$ y $\mathcal{A}_2$, definimos su unión $\mathcal{A}_1 + \mathcal{A}_2 : A \to \mathbb{N}$ como el multiconjunto definido por $(\mathcal{A}_1 + \mathcal{A}_2)(a) = \mathcal{A}_1(a) + \mathcal{A}_2(a)$, $A_1 \sqcup A_2 : A \to \mathbb{N}$ como $(A_1 \sqcup A_2)(a) = max(A_1(a), A_2(a))$ y diremos que $\mathcal{A}_1 \subseteq \mathcal{A}_2$ si $\mathcal{A}_1(a) \leq \mathcal{A}_2(a)$ para todo $a \in S(\mathcal{A}_1)$. Si $\mathcal{A}_1 \subseteq \mathcal{A}_2$ se define $\mathcal{A}_2 - \mathcal{A}_1$ mediante $(\mathcal{A}_2 - \mathcal{A}_1)(a) = \mathcal{A}_2(a) - \mathcal{A}_1(a)$. Todo subconjunto $B \subseteq A$ se puede ver como miembro de $\mathcal{MS}(A)$, sin más que identificar a B con su función característica $i_B(a) = \begin{cases} 1 \text{ si } a \in B \\ 0 \text{ si } a \notin B \end{cases}$.

El siguiente es el orden natural entre multiconjuntos de elementos ordenados.

Definición 2.1.1 *Sea $(A, \leq)$ un cuasiorden y $\mathcal{A}, \mathcal{A}' \in \mathcal{MS}(A)$. Definimos el cuasiorden $\sqsubseteq$ en $\mathcal{MS}(A)$ mediante $\mathcal{A} \sqsubseteq \mathcal{A}'$ sii existe una familia de multiconjuntos $\{\mathcal{A}'_a \mid a \in S(\mathcal{A})\}$ tales que:*

- $\sum_{a \in S(\mathcal{A})} \mathcal{A}'_a \subseteq \mathcal{A}'$,
- $\mathcal{A}(a) = |\mathcal{A}'_a|$ *para todo* $a \in S(\mathcal{A})$,
- $a \leq b$ *para todo* $a \in S(\mathcal{A})$ *y* $b \in S(\mathcal{A}'_a)$.

2.2. Sistemas de transiciones

Definición 2.2.1 *Un* sistema de transiciones etiquetadas *S es una tupla $S = (E, V, \rightarrow, e_0)$, donde E es un conjunto de estados, V es el conjunto de etiquetas de transiciones, $e_0 \in E$ es el estado inicial y $\rightarrow \subseteq E \times V \times E$ es la relación de transición. Si $(e_1, a, e_2) \in \rightarrow$ escribiremos $e_1 \xrightarrow{a} e_2$.*

Escribiremos $e \rightarrow^* e'$ si existen $a_1, \ldots, a_n \in V$ y $e_1, \ldots, e_n \in E$ tales que $e \xrightarrow{a_1} e_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} e_n = e'$. Diremos que una secuencia como la anterior es una *traza* del sistema de transiciones y denotaremos por $\mathcal{C}_S$ (o simplemente $\mathcal{C}$) al conjunto de trazas. Diremos que e es alcanzable si $e_0 \rightarrow^* e$ y denotaremos mediante $Reach(S)$ al conjunto de estados alcanzables de S.

La siguiente es la definición de isomorfismo entre sistemas de transiciones que usaremos, que difiere de la definición usual entre árboles en que sólo nos fijamos en los estados alcanzables desde el estado inicial.

Definición 2.2.2 *Sean $S = (E, V, \rightarrow, e_0)$ y $T = (F, V, \mapsto, f_0)$ dos sistemas de transiciones etiquetadas. Diremos que S y T son isomorfos, en cuyo caso escribiremos $S \approx T$, si existe una correspondencia $h : Reach(S) \rightarrow Reach(T)$ tal que:*

1. *h es una biyección entre $Reach(S)$ y $Reach(T)$,*
2. *$h(e_0) = f_0$,*
3. *$e \xrightarrow{a} e' \Leftrightarrow h(e) \overset{a}{\mapsto} h(e')$, para todo $e \in Reach(S)$.*

Diremos que la función h anterior es un isomorfismo de sistemas de transiciones etiquetadas.

Dado $a \in V$, el conjunto de a-predecesores de $e \in E$ se define como $Pre_a(e) = \{e' \in E \mid e' \xrightarrow{a} e\}$ y se extiende puntualmente a conjuntos de estados.

Un preorden $\leq$ definido sobre E induce el siguiente problema de accesibilidad: dado $e \in E$, ¿existe algún e' alcanzable tal que $e \leq e'$?. A este problema se le conoce como el problema de accesibilidad de estados de control, o *recubrimiento*.

Sistemas bien estructurados

En muchas de las demostraciones del capítulo 6 jugarán un importante papel los conceptos de buen cuasiorden y sistema bien estructurado, que definimos a continuación.

Definición 2.2.3 *Diremos que un cuasiorden $\leq$ es un buen cuasiorden si para toda secuencia $(a_i)_{i=1}^{\infty}$ existen índices $i < j$ tales que $a_i \leq a_j$.*

En lugar de *buen cuasiorden* escribiremos wqo, por las siglas en inglés de *well quasiorder*. Necesitaremos los siguientes resultados, cuyas demostraciones se pueden encontrar en [ACJT00].

Lema 2.2.4 *Si A es un conjunto finito y $\leq$ es un preorden en A, entonces $(A, \leq)$ es un wqo.*

Lema 2.2.5 *Si $\{(A_i, \leq_i)\}_{i=1}^{n}$ es una familia de wqos, entonces $(A_1 \times \ldots \times A_n, \leq)$ es un wqo, donde*

$$(a_1, \ldots, a_n) \leq (b_1, \ldots, b_n) \Leftrightarrow a_i \leq_i b_i \quad \forall i \in \{1, \ldots, n\}.$$

Lema 2.2.6 *Si $(A, \leq)$ es un wqo, entonces $(\mathcal{MS}(A), \sqsubseteq)$, tal y como se ha definido en la definición 2.1.1, es un wqo.*

Un sistema de transiciones $S = (E, V, \rightarrow, e_0)$ se dice monótono respecto de un preorden $\leq$ si siempre que $e_1 \xrightarrow{a} e_2$ y $e_1 \leq e_1'$ existe e_2' tal que $e_1' \xrightarrow{a} e_2'$ y $e_2 \leq e_2'$.

Definición 2.2.7 *Sea $\leq$ un preorden definido sobre el conjunto de estados de un sistema de transiciones $S = (E, V, \rightarrow, e_0)$. Diremos que S es un sistema bien estructurado si se cumplen las siguientes condiciones:*

- *$\leq$ es un wqo,*
- *S es monótono respecto de $\leq$,*
- *el conjunto $min(pre_a(\mathcal{C}(J)))$ es computable para todo $J \subseteq E$ y $a \in V$.*

Además, diremos que es estrictamente bien estructurado si $\leq$ es un orden parcial y se cumple la siguiente propiedad de monotonía estricta: Si $s_1 \rightarrow s_2$ y $s_1 < s_1'$ entonces existe s_2' tal que $s_2 < s_2'$ y $s_1' \rightarrow s_2'$.

En [FS01] se prueba que para los sistemas bien estructurados el problema de recubrimiento asociado es decidible. Además, también se prueba que si el sistema es estrictamente bien estructurado entonces el problema de decidir si el sistema es acotado, es decir, el conjunto de estados alcanzables es finito, también es decidible.

2.3. Redes de Petri

Presentamos ahora las definiciones formales para los conceptos relativos a las redes P/T descritos intuitivamente en el capítulo anterior.

Definición 2.3.1 *Una red P/T es una tupla $N = (P, T, F)$, donde P y T son dos conjuntos disjuntos, $F \subseteq (P \times T) \cup (T \times P)$.*

Dada una red P/T $N = (P, T, F)$, llamaremos *lugares* a los elementos de P, *transiciones* a los elementos de T y arcos a los elementos de F. Un *marcaje* es cualquier elemento de $\mathcal{MS}(P)$. Así que especificaremos el número de tokens que hay en un lugar p mediante $M(p)$. Si $t \in T$ definimos ${}^\bullet t = \{p \in P \mid (p, t) \in F\}$ y $t^\bullet = \{p \in P \mid (t, p) \in F\}$.

Definición 2.3.2 *Sea $N = (P, T, F)$, M un marcaje de N y $t \in T$. Diremos que la transición t está activada en M si ${}^\bullet t \subseteq M$, en cuyo caso t se puede disparar, obteniéndose tras su disparo el marcaje M' definido por*

$$M' = M - {}^\bullet t + t^\bullet$$

Si M' es el resultado de disparar t sobre el marcaje M, escribiremos $M[t\rangle M'$ o $M \xrightarrow{t} M'$. Diremos que un marcaje M es alcanzable desde un marcaje inicial M_0 si existe una cadena de marcajes y transiciones tal que

$$M_0[t_1\rangle M_1[t_2\rangle \ldots M_{n-1}[t_n\rangle M$$

con $n \geq 0$. El problema de decidir si un marcaje dado es alcanzable es el llamado *problema de alcanzabilidad*, que es decidible para las redes P/T [Reu90]. Diremos que un marcaje M se puede cubrir en N si existe un marcaje alcanzable M' tal que $M \subseteq M'$. El problema de decidir si un marcaje dado se puede cubrir es el *problema del recubrimiento*, que también es decidible para redes ordinarias. Este último es el problema de accesibilidad de estado de control, tal y como está definido en la sección 2.2, inducido por el orden natural de marcajes en redes P/T, $M_1 \sqsubseteq M_2 \Leftrightarrow M_1(p) \leq M_2(p)$ para todo lugar p.

Árbol de recubrimiento

Una red de Petri induce un *grafo de accesibilidad*, cuyos nodos son el conjunto de marcajes alcanzables, con un arco entre dos marcajes si en el primero se puede disparar una transición para alcanzar el segundo. Una red se dice *acotada* cuando su grafo de alcanzabilidad es finito. Para la verificación de una red P/T no acotada es de gran utilidad una representación finita de su árbol de alcanzabilidad, llamado *árbol de recubrimiento*. En él los nodos no son marcajes, sino ω-marcajes.

Se llama *ω-marcaje* de una red P/T $N = (P, T, F)$ a cualquier función $\overline{M} : P \to \mathbb{N} \cup \{\infty\}$, es decir, a los marcajes en los que algunos lugares pueden aparecer

un número infinito de veces. Las definiciones de activación y disparo de transiciones anteriores se generalizan a ω-marcajes sin más que extender la aritmética en $\mathbb{N}$ a $\mathbb{N} \cup \{\infty\}$ en la forma natural (nótese que el caso $\infty - \infty$, que no está definido, no aparece).

La idea para construir el árbol de recubrimiento radica en el hecho de que si $M \rightarrow^* M'$ y M' cubre a M, entonces la secuencia de transiciones disparadas desde M para llegar a M' está de nuevo activada. Si, además, existe un lugar p tal que $M(p) < M'(p)$ entonces el número de tokens en p puede llegar a ser arbitrariamente grande. Por tanto, cuando se detecta una secuencia así, el marcaje M' se puede sustituir de manera segura por el ω-marcaje $\overline{M}'$ que coincide con M', excepto que $\overline{M}'(p) = \infty$. Este es el mecanismo para construir el árbol de recubrimiento, que tiene la ventaja de ser siempre finito, tanto el árbol en sí como el proceso que lo construye, independientemente del carácter acotado o no de la red.

En particular, mediante inspección del árbol de recubrimiento se puede saber si la red es acotada (si todos los ω-marcajes son, de hecho, marcajes ordinarios) y si un marcaje dado se puede cubrir (si existe un ω-marcaje en el árbol que lo cubre).

Para ver más detalles y otras cuestiones técnicas sobre redes P/T puede consultarse, por ejemplo, [DR98].

2.4. Lógica de reescritura y Maude

Como hemos dicho, hemos implementado nuestro prototipo en el lenguaje Maude [CDE+02], que es una implementación eficiente de la lógica de reescritura [MOM02]. En la lógica de reescritura y Maude, tanto los datos como el estado del sistema se especifican formalmente como un tipo de datos algebraico, por medio de una especificación ecuacional. Maude usa una versión muy expresiva de la lógica ecuacional, la *lógica ecuacional de pertenencia* [BJM00]. En estas especificaciones podemos definir nuevos tipos (por medio de la palabra reservada `sort(s)`); relaciones de subtipado (entendidas como la relación de inclusión) entre tipos (`subsort`); operadores (`op`) para construir nuevos valores de esos tipos a partir de valores del tipo de sus argumentos, y que pueden tener atributos como la asociatividad (`assoc`) o la conmutatividad (`comm`), por ejemplo; ecuaciones (`eq`) que identifican términos construidos con esos operadores; y pertenencias (`mb`) $t : s$, que afirman que el término t tiene tipo s. Tanto las pertenencias como las ecuaciones pueden ser condicionales, lo que se indica utilizando las palabras reservadas `cmb` y `ceq`, respectivamente. Las condiciones se construyen como conjunciones (escritas `/\`) de ecuaciones y pertenencias.

El siguiente módulo *funcional* (con sintaxis `fmod...endfm`) de Maude define la sintaxis de un sistema productor-consumidor donde los productores pueden estar o bien inactivos o preparados para un envío, los consumidores pueden estar o bien esperando un producto o preparados para su consumo y los productos pueden

estar o bien preparados para ser enviados, almacenados, o aceptados para ser consumidos.

```
fmod CONSUMER-PRODUCER-SIGNATURE is
  sorts ProducerState ConsumerState ItemState State .
  subsorts ProducerState ConsumerState ItemState < State .
  op __ : State -> State [assoc comm id: null] .
  op null : -> State .
  ops idleProducer readyToSendProducer : -> ProducerState .
  ops waitingConsumer readyToConsume : -> ConsumerState .
  ops itemReady itemStored itemAccepted : -> ItemState .
endfm
```

Suponemos que las ecuaciones son confluentes y terminantes, es decir, podemos usar las ecuaciones de izquierda a derecha para reducir un término t a una única forma canónica t' (módulo los atributos de los operadores, como la asociatividad, la conmutatividad o el elemento neutro) que es equivalente a t (representan el mismo valor).

El comportamiento *dinámico* de un sistema se especifica por medio de reglas de reescritura de la forma

$$t \longrightarrow t' \;\; si \;\; (\bigwedge_i u_i = v_i) \wedge (\bigwedge_j w_j : s_j) \wedge (\bigwedge_k p_k \longrightarrow q_k)$$

que describen transiciones locales y concurrentes del sistema. Es decir, cuando alguna parte del sistema encaja con el patrón t y se cumplen las condiciones, entonces esa parte puede transformarse en la correspondiente instancia del patrón t'. Las reglas de reescritura se incluyen en los módulos *sistema* (con sintaxis `mod...endm`).

El siguiente módulo especifica el comportamiento del sistema productor-consumidor.

```
mod PRODUCER-CONSUMER is
  inc PRODUCER-CONSUMER-SIGNATURE .
  rl [produce] : idleProducer => itemReady readyToSendProducer .
  rl [send] : itemReady readyToSendProducer => idleProducer itemStored .
  rl [accept] : itemStored waitingConsumer => itemAccepted readyConsumer .
  rl [consume] : itemAccepted readyConsumer => waitingConsumer .
endm
```

De hecho, este módulo se puede ver como la semántica de reescritura, escrita en Maude, de la red de Petri que se muestra en la figura 2.1, como se afirma en [SMÖ01a], donde los autores continuaron el trabajo comenzado en [MM90]. La idea es que un marcaje se puede representar como un término que especifique multiconjuntos de lugares. Entonces, una red de Petri se puede ver como un módulo de sistema que indica cómo se pueden reescribir esos multiconjuntos, a través de una regla de reescritura (incondicional) por cada transición de la red, que reescribe precondiciones de la transición en postcondiciones.

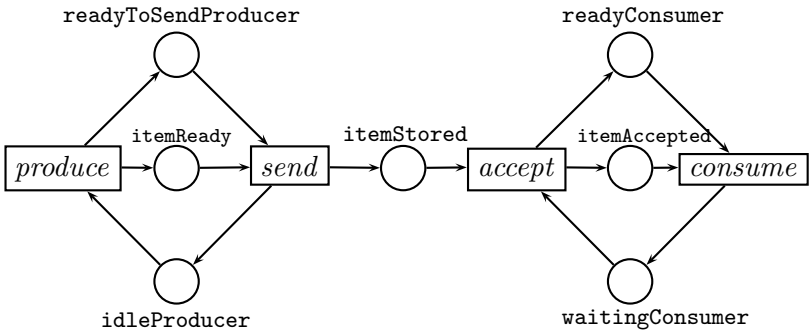

Figura 2.1: Red de Petri que modela un sistema productor-consumidor

La idea básica es que se puede representar cada marcaje de una red P/T mediante el multiconjunto de lugares en el que hay tokens, de manera que el disparo de una transición se corresponde con una reescritura de esos multiconjuntos.

2.5. Máquinas de dos contadores (TCM)

Una máquina de contadores, o máquina de Minsky [Min67], es una máquina de finitos estados que opera sobre un conjunto de contadores sobre los que puede realizar las siguientes operaciones: chequear si el valor de un contador es cero, incrementar el contador en una unidad o decrementarlo en una unidad (y si es cero dejarlo como está).

Nosotros nos centraremos en el caso en el que contemos únicamente con dos contadores. Formalmente, una *máquina de dos contadores* (TCM por las siglas de *Two Counter Machines*) consiste en un conjunto finito de estados $\mathcal{S} = \{s_0, \ldots, s_k\}$ y un conjunto de instrucciones $\mathcal{I} = \mathit{Inc} \cup \mathit{Dec}$: $\mathit{Inc}(i, s, t) \in \mathit{Inc}$ incrementa el contador c_i en una unidad cuando la máquina está en el estado s, pasando a estar en el estado t; $\mathit{Dec}(i, s, t, u) \in \mathit{Dec}$ decrementa el contador c_i en una unidad cuando está en el estado s, pasando a estar en el estado t, si $c_i > 0$, y simplemente pasa al estado u, en caso de que $c_i = 0$. El estado inicial de la máquina es $\langle s_0, c_1 = 0, c_2 = 0\rangle$.

Las TCMs son Turing-completas. En particular, la alcanzabilidad es indecidible para las TCMs. También es indecidible el problema de decidir si una configuración de la forma $\langle s, 0, 0\rangle$ es alcanzable [Min67, HU79].

2.6. Nociones de simulación

A lo largo de esta tesis iremos definiendo una serie de modelos para el estudio de propiedades de seguridad y coordinación de creciente capacidad expresiva. Conforme vayamos introduciendo estos modelos los iremos comparando con los demás. Para ello usaremos fundamentalmente la noción de simulación monótona,

definida a continuación, que preserva las propiedades en las que estamos interesado, como probamos en esta sección.

Primero veamos una versión débil de esa noción de simulación, que podemos llamar simulación débil, con la que nos tendremos que conformar en algunas ocasiones.

Definición 2.6.1 *Diremos que un sistema de transiciones* $S_1 = (E_1, V_1, \rightarrow_1, e_1)$ *es* simulado débilmente *por* $S_2 = (E_2, V_2, \rightarrow_2, e_2)$ *por medio de* $h : E_1 \rightarrow E_2$ *si* h *es inyectiva,* $h(e_1) = e_2$ *y* $e_1 \rightarrow_1^* e_2 \Leftrightarrow h(e_1) \rightarrow_2^* h(e_2)$*. Diremos que un estado* $e \in E_2$ *es estable si está en la imagen de* h*.*

Por lo tanto, la simulación débil es completa respecto del sistema original, pero sólo es correcta en cómputos que lleven a los estados que hemos llamado estables, es decir, tiene cómputos que no se corresponden con simulaciones de cómputos del sistema original, pero estos cómputos acaban en estados que no están en la imagen de h. Cuando probemos que una transformación es una simulación débil usaremos una técnica de demostración que podemos llamar de *detección estática de basura*, ya usada en [FERA99, VRGFE99]. La técnica consiste en definir un conjunto de *estados con basura* y probar que son subconjunto de los estados inestables primero y que el sistema es incapaz de tirar la basura después, es decir, que a partir estados con basura no se pueden alcanzar estados que no la tengan. De esta forma podemos suponer que todos los estados por los que ha pasado un cómputo que llega a un estado estable son estables y sin basura, lo que nos proporciona mayor control sobre la naturaleza de los posibles cómputos. Usaremos esta técnica en las demostraciones de la proposiciones 5.2.5 y 6.1.8.

Antes de definir la simulación monótona veamos un paso intermedio, que llamaremos simplemente simulación.

Definición 2.6.2 *Sean* $S_1 = (E_1, V_1, \rightarrow_1, e_1)$ *y* $S_2 = (E_2, V_2, \rightarrow_2, e_2)$ *dos sistemas de transiciones tales que* S_2 *simula débilmente a* S_1 *por medio de* h*. Diremos que* S_2 simula *a* S_1 *si* $h(e_1) \rightarrow_2 e'_1 \rightarrow_2 \ldots \rightarrow_2 e'_n \rightarrow e'$ *con* e'_i *inestable para* $i \in \{1, \ldots, n\}$ *implica:*

- *si* $e' = h(e_2)$ *para algún* e_2 *entonces* $e_1 \rightarrow_1^* e_2$*,*
- *si* e' *es inestable entonces* $e' \rightarrow_2 e''_1 \rightarrow_2 \ldots \rightarrow_2 e''_m \rightarrow_2 h(e_2)$ *con* e''_i *inestables y* $e_1 \rightarrow_1^* e_2$*.*

La anterior definición añade a las condiciones de simulación débil la corrección de todos los cómputos del sistema original. Intuitivamente, una transición en el sistema original es simulado por una secuencia de transiciones. La definición anterior establece que cualquier cómputo de la simulación o bien es un estado estable, en cuyo caso ese cómputo se corresponde con uno del sistema simulado, o es un estado inestable, en cuyo caso basta dejar ejecutarse al sistema que simula para que eventualmente alcance un estado estable, de manera que su cómputo también represente un cómputo del sistema original.

En nuestro caso necesitamos algo más. Generalmente trabajaremos con modelos en los que la noción de orden es de vital importancia. El caso más notable es la definición de recubrimiento que, como hemos visto, depende del orden considerado. Necesitamos entonces simulaciones que también preserven ese orden.

Definición 2.6.3 *Sean $S_1 = (E_1, V_1, \rightarrow_1, e_1)$ y $S_2 = (E_2, V_2, \rightarrow_2, e_2)$ dos sistemas de transiciones tales que S_2 simula a S_1 por medio de h. Supongamos que $(E_1, \sqsubseteq_1)$ y $(E_2, \sqsubseteq_2)$ son dos cuasiórdenes. Diremos que $(S_2, \sqsubseteq_2)$ simula monótonamente a $(S_1, \sqsubseteq_1)$ si h y h^{-1} son monótonas respecto de $\sqsubseteq_1$ y $\sqsubseteq_2$ y para todo $e_1 \in E_1$ y $e_2 \in E_2 \setminus h(E_1)$ alcanzable se tiene que $h(e_1) \not\sqsubseteq_2 e_2$.*

Levantemos las anteriores definiciones a conjuntos de sistemas de transiciones.

Definición 2.6.4 *Diremos que un conjunto de sistemas de transiciones $\mathbf{M}'$ simula (débilmente/monótonamente) a otro $\mathbf{M}$ si para todo $N \in \mathbf{M}$ existe $F(N) = N' \in \mathbf{M}$, con F computable, tal que $F(N)$ simula (débilmente/monótonamente) a N.*

La simulación débil preserva la alcanzabilidad:

Proposición 2.6.5 *Si $\mathbf{M}'$ simula débilmente a $\mathbf{M}$ y la alcanzabilidad es decidible en $\mathbf{M}'$ entonces la alcanzabilidad es decidible en $\mathbf{M}$.*

Demostración. Es inmediato ver que por la definición de simulación débil, para cada $N = (E, V, \rightarrow, e_0) \in \mathbf{M}$ y $e \in E$, e es alcanzable en N si y sólo si $h(e)$ es alcanzable en $F(N)$, por lo que basta decidir la alcanzabilidad de este último.

□

Veamos ahora que la simulación monótona preserva el recubrimiento.

Proposición 2.6.6 *Si $\mathbf{M}'$ simula monótonamente a $\mathbf{M}$ y el recubrimiento es decidible en $\mathbf{M}'$ entonces el recubrimiento es decidible en $\mathbf{M}$.*

Demostración. Veamos que M se puede cubrir en N si y sólo si $h(M)$ se puede cubrir en $F(N)$. Si M se puede cubrir, entonces $M_0 \rightarrow^* \overline{M} \sqsupseteq_1 M$. Entonces se tiene que $h(M_0) \rightarrow^* h(\overline{M})$ y por la monotonía de h, $h(M) \sqsubseteq h(\overline{M})$ y por tanto $h(M)$ se puede cubrir en $F(N)$. Veamos el recíproco. Si $h(M)$ se puede cubrir es porque existe $\overline{M}'$ con $h(M_0) \rightarrow^* \overline{M}' \sqsupseteq h(M)$. Como la simulación es monótona y $\overline{M}' \sqsupseteq h(M)$ ha de existir $\overline{M}$ con $\overline{M}' = h(\overline{M})$. Entonces tenemos que $M_0 \rightarrow^* \overline{M}$ y por monotonía de h^{-1} se sigue que $\overline{M} \sqsupseteq M$ y M se puede cubrir en N.

□

Cuando la simulación lo es paso a paso, es decir, cuando toda transición es simulada por una única transición, la simulación débil implica simulación, como vemos en el próximo resultado.

Proposición 2.6.7 *Sean $S_1 = (E_1, V_1, \rightarrow_1, e_1)$ y $S_2 = (E_2, V_2, \rightarrow_2, e_2)$ dos sistemas de transiciones y $h : E_1 \rightarrow E_2$ inyectiva tal que $h(e_1) = e_2$. Si $e_1 \rightarrow e_2 \Leftrightarrow h(e_1) \rightarrow h(e_2)$, todos los estados alcanzables de S_2 son estables y tanto h como h^{-1} son monótonas entonces S_2 simula monótonamente a S_1.*

Demostración. Claramente $e_1 \rightarrow e_2 \Leftrightarrow h(e_1) \rightarrow h(e_2)$ implica que S_2 simula débilmente a S_1. Veamos que esa simulación no es estrictamente débil. Para ello supongamos que $h(e_1) \rightarrow_2 e'_1 \rightarrow_2 \ldots \rightarrow_2 e'_n \rightarrow e'$ con e'_i inestables. Como todos los alcanzables en S_2 son estables se cumple que $n = 0$ y $e' = h(e_2)$, es decir, $h(e_1) \rightarrow_2 h(e_2)$ lo que, por hipótesis, implica que $e_1 \rightarrow_1 e_2$ y por tanto S_2 simula a S_1. El resto de las hipótesis nos dicen que la simulación es monótona y la tesis se sigue.

□

Como corolario inmediato del resultado anterior podemos concluir que el concepto de isomorfismo de la sección 2.2 es más fuerte que el de simulación.

Corolario 2.6.8 *Si h es un isomorfismo entre S y T y tanto h como su inversa son monótonas entonces S y T se simulan mutuamente.*

Definición 2.6.9 *Diremos que dos conjuntos de sistemas de transiciones $\mathbf{M}$ y $\mathbf{M}'$ son equivalentes si se simulan monótonamente mutuamente.*

Así que diremos que dos modelos son equivalentes cuando se pueden simular mutuamente, es decir, cuando están en el núcleo de la relación de orden que define la simulación monótona. El siguiente resultado nos dice que, efectivamente, la simulación monótona define tal noción de orden.

Proposición 2.6.10 *La simulación monótona es un cuasiorden.*

Demostración. La reflexividad es trivial tomando F como la identidad. Veamos que es transitiva. Si $\mathbf{M}''$ simula a $\mathbf{M}'$ a través de F' y $\mathbf{M}'$ simula a $\mathbf{M}$ a través de F, veamos que entonces $\mathbf{M}''$ simula a $\mathbf{M}$ a través de $F \circ F'$. Como $\mathbf{M}'$ simula a $\mathbf{M}$ existe $F(N)$ que simula a $N \in \mathbf{M}$ por medio de h. Como $\mathbf{M}''$ simula a $\mathbf{M}'$ existe $F'(F(N))$ que simula a $F(N)$ por medio de h'. Veamos que $F'(F(N))$ simula a N por medio de $h' \circ h$:

- h es inyectiva por serlo h y h'.
- $(h' \circ h)(M_0) = h'(M'_0) = M''_0$.
- $e_1 \rightarrow^* e_2 \Leftrightarrow h(e_1) \rightarrow^* h(e_2) \Leftrightarrow h'(h(e_1)) \rightarrow^* h'(h(e_2))$.
- Supongamos que $(h' \circ h)(e_1) \rightarrow^* e''_1 \rightarrow \ldots \rightarrow e''_n \rightarrow e''$ con e''_i inestables.

  Si $e'' = h'(h(e_2))$ para algún e_2 por ser h' simulación débil se tiene que $h(e_1) \rightarrow^* h(e_2)$, donde todos los estados intermedios que aparecen en ese

cómputo son inestables. Entonces, por ser h simulación se tiene que $e_1 \to e_2$, como queríamos probar.

Supongamos ahora que e'' no es estable para $h' \circ h$ pero que sí lo es para h', es decir, existe e' tal que $e'' = h'(e')$ con e' inestable para h. De nuevo, por ser h' simulación débil tenemos que $h(e_1) \to^* e'$ con todos los estados excepto $h(e_1)$ inestables. Por ser h simulación existe e_2 tal que $e' \to^* h(e_2)$ (todos inestables excepto $h(e_2)$) y $e_1 \to e_2$. De nuevo por ser h' simulación débil concluimos que $e'' = h'(e') \to^* h'(h(e_2))$, como queríamos ver en este caso.

Supongamos por último que e'' tampoco es estable para h'. Sea $\bar{e}''$ el estado anterior a e'' más cercano a él en ese cómputo que sea estable para h', es decir, $\bar{e}'' = h'(\bar{e}')$. Por ser h' simulación existe e' tal que $\bar{e}' \to e'$ y $e'' \to^* h'(e')$. Además, h' es simulación débil y $h'(h(e_1)) \to^* h'(\bar{e}')$, por lo que $h(e_1) \to^* \bar{e}'$ y todos los estados intermedios son inestables para h. Ahora, como h es simulación ha de existir e_2 tal que $e_1 \to e_2$ y $\bar{e}' \to^* h(e_2)$. Pues bien, $e'' = h'(\bar{e}') \to^* h'(h(e_2))$, como queríamos demostrar.

- La composición de funciones monótonas es monótona. Veamos para concluir que no existen inestables mayores que estables. Sea e'' un inestable cualquiera según $h' \circ h$ y supongamos que $h'(h(e)) \sqsubseteq e''$. En particular, e'' es inestable para h' y, por tanto, para $e' = h(e)$ se cumple que $h'(e') \sqsubseteq e''$, lo que nos lleva a contradicción por ser h' simulación monótona.

□

Capítulo 3

Computación ubicua y redes de Petri

En este capítulo vamos a introducir un primer modelo para el estudio de los sistemas ubicuos. Vamos a realizar una presentación incremental a lo largo de la tesis, añadiendo nuevas características en los capítulos sucesivos. En esta aproximación nos vamos a fijar únicamente en la coordinación, obviando cualquier referencia a la seguridad. Así, se puede ver el modelo que definiremos a continuación como punto de partida de esta tesis.

3.1. Redes ubicuas

Vamos a definir un modelo que llamaremos sistemas de redes ubicuas. Estos sistemas serán de dos niveles. Por un lado tendremos procesadores, que se interpretan como dispositivos inmóviles, y por otro lado tenemos procesos, o agentes, que se mueven entre los distintos procesadores. Tanto los procesos como los procesadores están representados por redes de Petri con transiciones especiales, que llamamos de sincronización, usadas para la interacción entre agentes y procesadores.

En lo sucesivo usaremos un conjunto $\mathcal{A}$ para referirnos a acciones autónomas y un conjunto S para referirnos a nombres de servicios, comunes a todos los sistemas. Estos nombres se pueden ver intuitivamente como las primitivas del estándar de coordinación o del estándar de descubrimiento de servicios usado, y que, por tanto, son conocidas por todo el mundo. Este conjunto da lugar a dos conjuntos disjuntos de etiquetas de sincronización, $S? = \{s? \mid s \in S\}$ y $S! = \{s! \mid s \in S\}$. Además, para identificar a las componentes proceso, usamos un conjunto $\mathcal{N}_r$ de nombres de procesadores.

Definición 3.1.1 *Una* red procesador *es una red de Petri etiquetada* $L = (P, T, F, \lambda)$ *donde:*

- *P y T son conjuntos finitos y disjuntos de lugares y transiciones, respectivamente,*

- $F \subseteq (P \times T) \cup (T \times P)$ *es el conjunto de arcos de la red,*
- $\lambda : T \to \mathcal{A} \cup S!$.

Definición 3.1.2 *Una* red proceso *es una red de Petri etiquetada* $A = (P, T, F, \lambda)$ *donde:*

- P *y* T *son conjuntos finitos y disjuntos de lugares y transiciones, respectivamente,*
- $F \subseteq (P \times T) \cup (T \times P)$ *es el conjunto de arcos de la red,*
- $\lambda : T \to \mathcal{A} \cup S? \cup \mathcal{M}_s$, *donde* $\mathcal{M}_s = \{go\ l \mid l \in \mathcal{N}_r\}$.

Los sistemas ubicuos estarán compuestos por una colección de redes procesador que proveen de servicios a una colección de redes proceso. Por tanto, un procesador tiene dos tipos de transiciones: *transiciones autónomas* (aquellas con $\lambda(t) \in \mathcal{A}$) y *transiciones proveedoras* (aquéllas con $\lambda(t) \in S!$). Las redes proceso, que pueden migrar entre los distintos procesadores debido a la ejecución de *go*, solicitan estos servicios. Por tanto, una red proceso tiene tres tipos de transiciones: *transiciones autónomas* (aquellas con $\lambda(t) \in \mathcal{A}$), *transiciones de solicitud* (aquellas con $\lambda(t) \in S?$), y *transiciones de movimiento* (aquellas con $\lambda(t) \in \mathcal{M}_s$).

Definición 3.1.3 *Un* sistema ubicuo *es un par de la forma* $\mathcal{N} = \langle \mathcal{N}_P, \mathcal{N}_A \rangle$ *donde* $\mathcal{N}_P = \{l_1 : L_1, \ldots, l_m : L_m\}$ *es una colección finita de redes procesador (o localidades) y* $\mathcal{N}_A = \{A_1, \ldots, A_n\}$ *es una colección finita de redes proceso (o agentes), con* $m > 0$, $n \geq 0$ *y* $l_i \in \mathcal{N}_r$ *para* $i = 1, \ldots, m$ *distintas dos a dos.*

$\mathcal{N}_r(\mathcal{N}) = \{l_1, \ldots, l_m\}$ denotará al conjunto de nombres de procesadores en $\mathcal{N}$. La naturaleza estática de las redes procesador, cuya localización está fija, subraya el carácter estático de algunos dispositivos o sistemas como los sistemas operativos. Por el otro lado, la naturaleza dinámica de las redes proceso, que pueden moverse de un procesador a otro para obtener los servicios que solicita, refleja el comportamiento de procesos en ambientes distribuidos.

Dado un sistema ubicuo denotaremos simplemente por P al conjunto de lugares de todas sus componentes, tanto procesos como procesadores. Del mismo modo, denotaremos simplemente por T al conjunto de transiciones de todo el sistema y análogamente con los arcos. Por último, simplemente denotaremos como λ a la función que etiqueta todas las transiciones en T según las funciones que etiquetan las transiciones de cada procesador o proceso.

Describimos la localidad en la que se encuentran los procesos en cada momento por medio de una función de localización *loc*, que le asigna a cada proceso el procesador en el que se encuentra.

Definición 3.1.4 *Dado un sistema ubicuo* $\mathcal{N}$, *definimos una* función de localización *para* $\mathcal{N}$ *como* $loc : \mathcal{N}_A \to \mathcal{N}_r(\mathcal{N})$.

Para completar la representación del estado del sistema, definimos los marcajes de los sistemas ubicuos como marcajes ordinarios de sus redes, que pueden ser modificados siguiendo reglas análogas a las reglas de disparo de las redes P/T.

Definición 3.1.5 *Un* marcaje *de un sistema ubicuo* $\mathcal{N}$ *es un par* (M, loc)*, donde* $M \in \mathcal{MS}(P)$ *y loc es una función de localización para* $\mathcal{N}$*.*

Como hemos dicho antes, tanto los procesadores como los procesos pueden realizar acciones autónomas, modeladas por las transiciones autónomas, cuya ejecución no depende del contexto en el que se encuentre la red. La regla correspondiente para estas transiciones es la misma que para las redes de Petri ordinarias, y en particular la localización de la red no cambia. Una red también puede moverse de su localización actual a otra por medio de la ejecución de una transición de movimiento, etiquetada por *go* l con $l \in \mathcal{N}_r$, que especifica el destino l del movimiento. En este modelo básico no consideramos problemas de seguridad, de manera que cualquier proceso podría desplazarse a cualquier procesador del sistema. Finalmente, modelamos la concesión de un servicio s a través del disparo sincronizado de dos transiciones, $t_1 \in T$ y $t_2 \in T$ con $\lambda(t_1) = s?$ y $\lambda(t_2) = s!$, que se corresponden respectivamente con la petición del servicio por parte de una red proceso A_k y su oferta por parte de una red procesador $L_{k'}$. Estas transiciones sólo se pueden disparar simultáneamente, cada una de ellas siguiendo la regla habitual de disparo en redes de Petri, siempre que el proceso A_k se encuentre en la localidad $L_{k'}$.

Definición 3.1.6 *Sea* $\mathcal{N}$ *un sistema ubicuo y* (M, loc) *un marcaje suyo. Una transición* autónoma $t \in T$ *con* $\lambda(t) \in \mathcal{A}$ *está activada en el marcaje* M *si* ${}^\bullet t \subseteq M$*. El marcaje de* $\mathcal{N}$ *alcanzado después del disparo de* t *es* (M', loc)*, donde* $M' = M - {}^\bullet t + t^\bullet$*.*

Definición 3.1.7 *Sea* $\mathcal{N}$ *un sistema ubicuo y* (M, loc) *un marcaje suyo. Una transición de* movimiento*,* $t \in T$ *con* $\lambda(t) = go\ l \in \mathcal{M}_s$*, está activada para el marcaje* M *si* ${}^\bullet t \subseteq M$*. El marcaje alcanzado por* $\mathcal{N}$ *tras el disparo de* t *es* (M', loc')*, donde:*

- $M' = M - {}^\bullet t + t^\bullet$,
- loc' *es la función de localización para* $\mathcal{N}$ *que coincide con loc salvo para la red proceso* A *que dispara* t*, para la que se tiene* $loc(A) = l$*.*

Definición 3.1.8 *Sea* $\mathcal{N}$ *un sistema ubicuo y* (M, loc) *un marcaje suyo. Un par de transiciones de* oferta/petición de servicio*,* $(t_1, t_2) \in T^2$ *con* t_1 *transición de* $A \in \mathcal{N}_A$*,* t_2 *transición de* $l : L \in \mathcal{N}_P$*,* $\lambda(t_1) = s?$*,* $\lambda(t_2) = s!$ *y* $loc(A) = l$*, está activado para* (M, loc) *si* ${}^\bullet t_1 + {}^\bullet t_2 \subseteq M$*. El estado alcanzado por* $\mathcal{N}$ *tras el disparo del par* (t_1, t_2) *es* (M', loc)*, donde*

$$M' = M - {}^\bullet t_1 - {}^\bullet t_2 + t_1^\bullet + t_2^\bullet.$$

Por lo tanto, de una manera similar a como se hace en [CH94], representamos la conectividad entre las distintas componentes, aunque de manera dinámica, ya que también estamos considerando las distintas localidades.

Aunque los servicios se ofrecen concurrentemente, la definición anterior garantiza que su uso se hace en exclusión mutua, considerándolos servicios incompartibles. Cuando un proceso pide un servicio s a un procesador, ambas redes sincronizan el disparo de sus correspondientes transiciones. En este momento, el servicio s deja de estar disponible, en principio, para cualquier otro proceso que esté pidiendo acceso al mismo. Por otra parte, si hubiese varios procesos solicitando el mismo servicio, la elección entre ellos se haría de manera no determinista.

En cualquiera de los casos anteriores escribiremos $\mathcal{M}[u\rangle\mathcal{M}'$ o $\mathcal{M} \xrightarrow{u} \mathcal{M}'$ si $\mathcal{M}'$ es el resultado de disparar la transición o par de transiciones u desde $\mathcal{M}$. Es más, si λ es inyectiva y no hay confusión en ocasiones escribiremos $\mathcal{M}[\lambda(u)\rangle\mathcal{M}'$ o $\mathcal{M} \xrightarrow{\lambda(u)} \mathcal{M}'$ si u es una transición y análogamente si es un par de transiciones.

Definición 3.1.9 *Sea $\mathcal{N}$ un sistema ubicuo con marcaje inicial (M_0, loc_0). Diremos que un marcaje (M, loc) es* alcanzable *si existe una secuencia de marcajes $(M_1, loc_1), \ldots, (M_n, loc_n) = (M, loc)$ tal que $(M_{i-1}, loc_{i-1})[t_i\rangle(M_i, loc_i)$ para todo $i = 1, \ldots, n$. El* problema de alcanzabilidad *consiste en decidir si un marcaje dado es alcanzable.*

En cuanto al análogo a la propiedad de recubrimiento de redes de Petri, para tener una definición más general permitiremos funciones de localidad parciales, definidas sólo para algunas componentes. Diremos que (M, loc) es un marcaje parcial si loc es una función parcial.

Definición 3.1.10 *Sea $\mathcal{N}$ un sistema ubicuo con marcaje inicial (M_0, loc_0). Diremos que un marcaje parcial (M, loc) se puede* cubrir *si existe un marcaje alcanzable (M', loc') tal que:*

- *$M \subseteq M'$,*
- *para todo $A \in Dom(loc)$, $loc(A) = loc'(A)$.*

El problema del recubrimiento *consiste en decidir si un marcaje parcial dado se puede cubrir.*

Es decir, estamos considerando el orden entre marcajes definido por $(M, loc) \sqsubseteq (M', loc) \Leftrightarrow M(p) \subseteq M'(p)$ para todo lugar p.

3.2. De los sistemas ubicuos a las redes P/T

En esta sección veremos que, por el momento, en realidad no hemos definido más que un mecanismo de azúcar sintáctico para redes de Petri ordinarias, que

facilitan su uso para el modelado de sistemas basados en componentes y con movilidad. Una red P/T puede verse como un sistema ubicuo degenerado con una sola componente procesador que no se mueve y no pretende sincronizar con ninguna otra componente. Veremos que el resultado recíproco también es cierto, en el sentido de que el comportamiento de cualquier sistema ubicuo puede ser simulado monótonamente por una red P/T ordinaria. La simulación, que más abajo describiremos con detalle, se basa en el hecho de que cada red puede únicamente moverse a un número finito de localidades (las especificadas por sus transiciones de movimiento) y puede sincronizar con un número finito de componentes.

Para simular las localidades, usaremos un conjunto de lugares del tipo (N, k), de manera que un lugar de esa forma marcado representa la situación en la que la red N se encuentra en la localidad k. Las transiciones de movimiento deben cambiar el marcaje de manera adecuada, de manera que si (N, k) está marcado (y por tanto, para cualquier l tal que $l \neq k$ el lugar (N, l) no está marcado) entonces el disparo de la transición $t_{go}(k, l)$ mueve el token de (N, k) a (N, l).

La sincronización de dos transiciones $t \in T_i$ y $t' \in T_j$ se puede simular en una red de Petri ordinaria, fusionándolas en una sola transición para cada localidad, de modo que el disparo de una transición (t, t', k) simule la sincronización de t y t' cuando ambas se encuentran en la localidad k. Para ello, dicha transición tendrá como precondiciones los lugares (N, k) y (N', k) (si t es una transición de N y t' lo es de N'), para garantizar que solamente se sincronizan transiciones de redes que están en la misma localidad.

Veamos la construcción de manera formal.

Definición 3.2.1 *Sea $\mathcal{N}$ un sistema ubicuo. Definamos entonces la siguiente red P/T, $\mathcal{N}^* = (P^*, T^*, F^*)$:*

1. $P^* = P \cup P_{@}$, *con* $P_{@} = \{(A, k) \mid A \in \mathcal{N}_A,\ k \in \mathcal{N}_r(\mathcal{N})\}$

2. $T^* = \{t \mid \lambda(t) \in \mathcal{A}\} \cup \{t_{go}(k, l) \mid t \in T,\ \lambda(t) = go\ l,\ k \in \mathcal{N}_r(\mathcal{N})\} \cup$
   $\{(t_1, t_2, k) \mid \lambda(t_1) = s!, \lambda(t_2) = s?,\ t_1$ *transición de* $k\}$.

3. F^* *es el menor conjunto que satisface:*

   a) *Si* $(t, p) \in F$ *entonces (y análogamente para* $(p, t) \in F$*):*
      - $\lambda(t) \in \mathcal{A} \Rightarrow (t, p) \in F^*$.
      - $\lambda(t) = go\ l \Rightarrow (t_{go}(k, l), p) \in F^*$ *para todo* $k \in \mathcal{N}_r(\mathcal{N})$.
      - $\lambda(t) = s! \Rightarrow ((t, t', k), p) \in F^*$ *para toda* $(t, t', k) \in T^*$.
      - $\lambda(t) = s? \Rightarrow ((t', t, k), p) \in F^*$ *para toda* $(t', t, k) \in T^*$.

   b) $(t, t', k) \in T^*$ *y* t' *transición de* $A \in \mathcal{N}_A$ *entonces* $((A, k), (t, t', k)) \in F^*$ *y* $(t, t', k), (A, k) \in F^*$.

   c) t *transición de* $A \in \mathcal{N}_A$ *y* $\lambda(t) = go\ k$ *entonces* $((A, l), t_{go}(l, k)) \in F^*$ *y* $(t_{go}(l, k), (A, k)) \in F^*$ *para todo* $l \in \mathcal{N}_r(\mathcal{N})$.

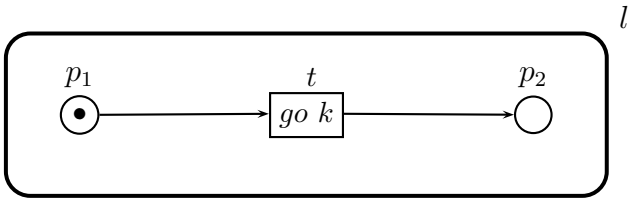

Figura 3.1: Ejemplo de red ubicua

Además, los marcajes del sistema ubicuo y su simulación están en correspondencia biunívoca, de la siguiente manera:

Definición 3.2.2 *Dado un sistema ubicuo $\mathcal{N}$ y un marcaje (M, loc) de $\mathcal{N}$, definimos el marcaje $(M, loc)^* : P^* \to \mathbb{N}$ de $\mathcal{N}^*$ de la siguiente manera:*

- $(M, loc)^*(p) = M(p)$ *para todo* $p \in P \subseteq P^*$,
- $(M, loc)^*((A, k)) = \begin{cases} 1 \text{ si } loc(A) = k \\ 0 \text{ en otro caso} \end{cases}$.

Probemos ahora la corrección de la simulación anterior.

Proposición 3.2.3 *Dado un sistema ubicuo $\mathcal{N}$, para la red P/T asociada $\mathcal{N}^*$ se tiene la siguiente relación:*

$$(M, loc)\ [u\rangle\ (M', loc') \Longleftrightarrow (M, loc)^*\ [v\rangle\ (M', loc')^*,$$

donde

- $\lambda(u) \in \mathcal{A} \Leftrightarrow u = v$,
- $u = (t_1, t_2)$, t_1 *transición de* $A \in \mathcal{N}_A$ *y* $loc(A) = k \Leftrightarrow v = (t_1, t_2, k)$,
- u *transición de* $A \in \mathcal{N}_A$, $loc(A) = k$ *y* $\lambda(u) = go\ l \Leftrightarrow v = u_{go}(k, l)$.

Demostración. En primer lugar, nótese que para cualquier marcaje (M, loc) se cumple que $(M, loc)^*|_P = M$ y que $(M, loc)^*|_{P_@} = (M', loc)^*|_{P_@}$ para M y M' cualesquiera.

- Por el punto 1 del apartado 3a de la definición 3.2.1 se cumple que ${}^\bullet u$ vista u como transición de $\mathcal{N}$ coincide con ${}^\bullet u$ cuando se ve u como transición de $\mathcal{N}^*$, al igual que ocurre con $u^\bullet$. Además, ${}^\bullet u \subseteq P$ y $u^\bullet \subseteq P$, por lo que la tesis se sigue inmediatamente.
- Por el tercer y el cuarto punto del apartado 3a y por el apartado 3b de la definición 3.2.1 se cumple que ${}^\bullet(t_1, t_2, k) = {}^\bullet t_1 + {}^\bullet t_2 + \{(A, k)\}$ y $(t_1, t_2, k)^\bullet = t_1^\bullet + t_2^\bullet + \{(A, k)\}$. Veamos la implicación a la derecha. Por hipótesis ${}^\bullet t_1 + {}^\bullet t_2 \subseteq M \subseteq (M, loc)^*$. Además por hipótesis sabemos que $loc(A) = k$, por lo

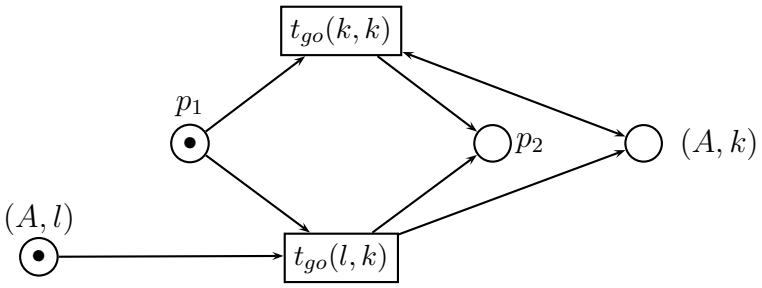

Figura 3.2: Traducción del ejemplo de la figura 3.1 a redes P/T

que según la definición 3.2.2 se tiene que $(M, loc)^*((A,k)) = 1$ y la transición está activada. Entonces $(M, loc)^* - {}^\bullet(t_1, t_2, k) + (t_1, t_2, k)^\bullet = M + (M, loc)^*|_{P_@} - {}^\bullet t_1 - {}^\bullet t_2 - \{(A,k)\} + t_1^\bullet + t_2^\bullet + \{(A,k)\} = M' + (M, loc)^*|_{P_@} = (M', loc)^*$.

Para la implicación a la izquierda, basta tener en cuenta que ${}^\bullet t_1 \cup {}^\bullet t_2 \cup \{(A,k)\} \subseteq (M, loc)^*$ implica que ${}^\bullet t_1 + {}^\bullet t_2 \subseteq M$ y $loc(A) = k$. Además, como $(M', loc)^* = (M, loc)^* - {}^\bullet t_1 - {}^\bullet t_2 - \{(A,k)\} + t_1^\bullet + t_2^\bullet + \{(A,k)\}$ se sigue, que $M' = (M', loc)|_P = M - {}^\bullet t_1 - {}^\bullet t_2 + t_1^\bullet + t_2^\bullet$.

- En este caso ${}^\bullet u_{go}(k,l) = {}^\bullet u \cup \{(A,k)\}$ y $u_{go}(k,l)^\bullet = u^\bullet \cup \{(A,l)\}$. Para la implicación a la derecha, la transición está activada siguiendo el razonamiento análogo al punto anterior. Además, $(M, loc)^* - {}^\bullet u_{go}(k,l) + u_{go}(k,l)^\bullet = M + (M, loc)^*|_{P_@} - {}^\bullet u - \{(A,k)\} + u^\bullet + \{(A,l)\} = M' + (M, loc')^*|_{P_@} = (M', loc')^*$, donde la penúltima igualdad se debe al segundo punto de la definición 3.1.7 y al segundo punto de la definición 3.2.2.

  A la izquierda, por hipótesis ${}^\bullet u \cup \{(A,k)\} \subseteq (M, loc)^*$, lo que implica que ${}^\bullet u \subseteq M$ y $loc(A) = k$ y la transición de movimiento u está activada. Por lo tanto, $(M', loc')^* = (M, loc)^* - {}^\bullet u - \{(A,k)\} + u^\bullet + \{(A,l)\}$ y, proyectando sobre P, $M' = (M', loc')|_P = M - {}^\bullet u + u^\bullet$. Por último, si proyectamos sobre $P_@$ y por el último comentario del párrafo anterior, se tiene que la función de localización obtenida según la definición 3.1.7 es loc'.

□

Proposición 3.2.4 *Las redes P/T simulan monótonamente a las redes ubicuas.*

Demostración. Veamos que estamos en las hipótesis de la proposición 2.6.7. Por la proposición anterior las redes P/T simulan paso a paso a las redes ubicuas. Veamos que todo marcaje alcanzable es estable. En efecto, si M es alcanzable entonces existe un único k para cada A tal que $M((A,k)) = 1$ y $M((A,l)) = 0$ para cada $l \neq k$. Esto es así porque el estado inicial lo cumple y porque las únicas

transiciones que modifican el marcaje en los $P_@$ son las transiciones de la forma $t_{go}(k,l)$. Si M cumple lo anterior y dispara $t_{go}(k,l)$ entonces $M((A,k)) = 0$ y $M((a,l)) = 1$ tras el disparo.

Entonces, si $\overline{M}$ es alcanzable con $\overline{M}((A,k_A)) =$ para cada A definimos (M, loc) tal que $M(p) = \overline{M}(p)$ para $p \in P$ y $loc(A) = k_A$, que verifica que $(M, loc)^* = \overline{M}$, de modo que $\overline{M}$ es estable.

Por último, veamos que tanto $(\ )^*$ como su inversa son monótonas. Pues bien, teniendo en cuenta una vez más que los marcajes alcanzables en la simulación cumplen la anterior condición, se tiene que $(M_1, loc_1) \sqsubseteq (M_2, loc_2) \Leftrightarrow M_1(p) \leq M_2(p)$ para todo p y $loc_1(A) = loc_2(A)$ para todo $A \Leftrightarrow (M_1, loc_1)^*(q) \leq (M_2, loc_2)^*(q)$ para todo $q \in P^* \Leftrightarrow (M_1, loc_1)^* \sqsubseteq (M_2, loc_2)^*$.

□

En particular, la simulación reduce tanto la alcanzabilidad como el recubrimiento de un marcaje (parcial) (M, loc) a la alcanzabilidad y al recubrimiento, respectivamente, de $(M, loc)^*$, de manera que se tiene el siguiente resultado.

Corolario 3.2.5 *Los problemas de alcanzabilidad y recubrimiento en sistemas ubicuos son decidibles.*

Es más, la anterior simulación también preserva el recubrimiento que considera marcajes parciales.

Ejemplo 1 Consideremos el sistema ubicuo de la figura 3.1, con una sola red (sin transiciones de sincronización) situada en l. La correspondiente traducción a redes de Petri es la red de la figura 3.2. Por otro lado, a la izquierda de la figura 3.3 se muestra un sistema con dos redes, una red proceso y una red procesador, y esta vez con transiciones de sincronización, aunque sin transiciones de movimiento. A la derecha de la misma figura está su traducción a red P/T. Consideramos por último un sistema en el que se combinan las distintas características, como el de la izquierda de la figura 3.4, en el que simplemente hemos sustituido la transición autónoma del sistema ubicuo de la figura 3.3 por una transición de movimiento.

3.3. Un ejemplo de aplicación

Para ilustrar el comportamiento de los sistemas ubicuos, presentamos en esta sección un ejemplo que modela un sistema compuesto por tres redes procesador, L_1, L_2 y L_3, y una red proceso, A, situada inicialmente en L_3 (figura 3.5). Podemos interpretar el procesador L_1 como un sistema electrónico de notas [FC03, CDF$^+$03] que requiere autenticación para acceder a sus contenidos (acción identificada mediante el servicio s_2), y podemos entender L_2 como un termómetro electrónico [Wan04], en el que la acción de consultar la temperatura se corresponde con el servicio s_3. Además, ambos procesadores pueden facilitar la

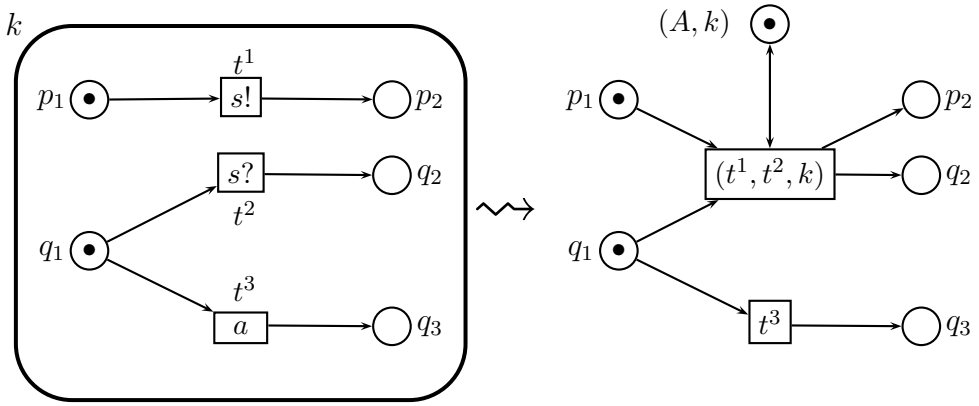

Figura 3.3: Traducción de sistema ubicuo a red P/T

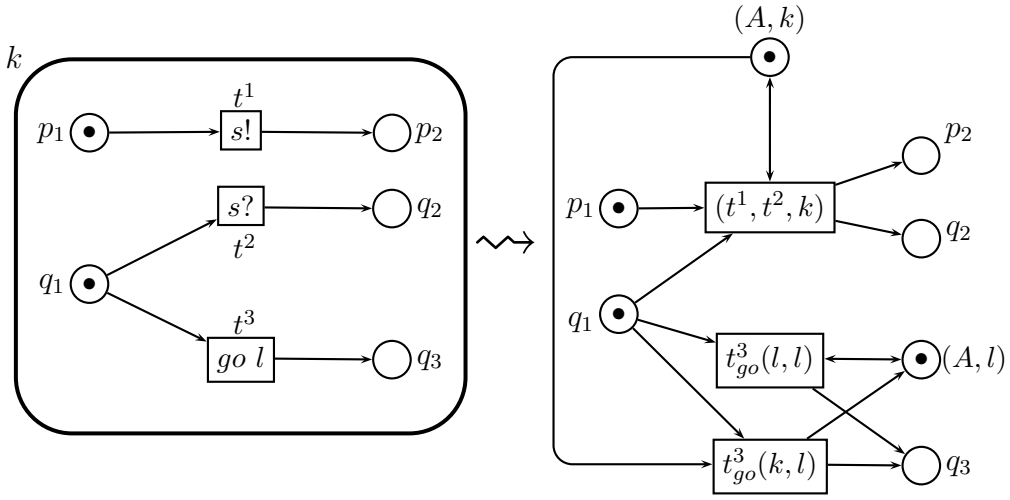

Figura 3.4: Otro ejemplo de sistema ubicuo y su simulación mediante una red P/T

hora local, acción facilitada por el servicio s_1. El procesador L_3 es simplemente un contenedor que puede albergar a las redes agente.

El proceso A puede moverse tanto al procesador L_1 como a L_2, donde puede pedir los servicios s_2 y s_3 (es decir, intentar ver los contenidos del sistema de notas o preguntar la temperatura, respectivamente). Antes de ofrecer el servicio s_2, el procesador L_1 requiere que se efectúe un *login*, y en caso de que se haga satisfactoriamente entonces se hace un *commit*. Si el login no es satisfactorio entonces se aborta el proceso. El procesador L_2 no requiere ninguna autenticación previa a la concesión de sus servicios. Por otro lado, después del disparo de la transición de movimiento, el proceso A tiene cuatro transiciones activadas, las etiquetadas por $c?$, $l?$, $s_2?$ y $s_3?$. Por tanto, está dispuesto a recibir los servicios s_2 o s_3, incluso cuando s_2 requiere autenticación. Es más, puede intentar hacer el commit sin un login previo, intentando forzar así el protocolo de comunicación.

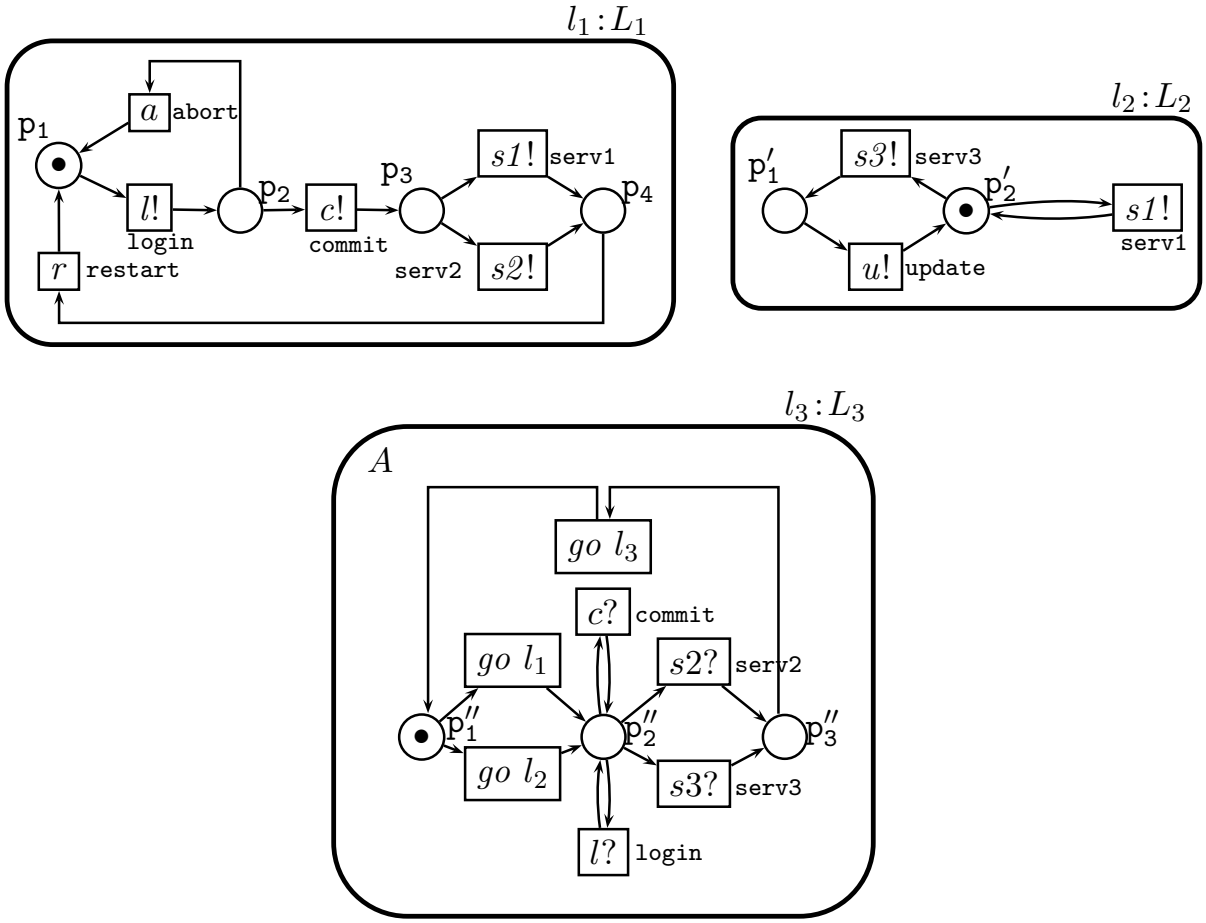

Figura 3.5: Sistema ubicuo modelado por redes de Petri

En este contexto, está claro que después del disparo de la transición *go* l_2, el proceso A obtiene el servicio s_3 de su nueva localidad L_2, la cual tiene que actualizarse (disparando la transición *update*) antes de ofrecer otra vez sus servicios (por ejemplo, si tiene que medir de nuevo la temperatura). El proceso A puede disparar *go* l_3 para volver a su estado inicial. Esta vez podría elegir disparar la transición *go* l_1. Como consecuencia del disparo de esta transición, el proceso A se mueve a L_1, donde se le pide hacer login antes de commit.

La siguiente traza se corresponde con la segunda de las situaciones, en la que el proceso quiere recibir el servicio $s2$.

$$\mathcal{M}_0[go\ l_1\rangle\mathcal{M}_1[(l?, l!)\rangle\mathcal{M}_2[(c?, c!)\rangle\mathcal{M}_3[(s2?, s2!)\rangle\mathcal{M}_4[go\ l_3\rangle\mathcal{M}_5[r\rangle\mathcal{M}_0$$

donde si para $i = 0, \ldots, 5$ escribimos $\mathcal{M}_i = (M_i, loc_i)$ entonces:

$$M_0 = \{p_1, p'_2, p''_1\} \quad loc_0(A) = l_3$$
$$M_1 = \{p_1, p'_2, p''_2\} \quad loc_1(A) = l_1$$
$$M_2 = \{p_2, p'_2, p''_2\} \quad loc_2(A) = l_1$$
$$M_3 = \{p_3, p'_2, p''_2\} \quad loc_3(A) = l_1$$
$$M_4 = \{p_4, p'_2, p''_3\} \quad loc_4(A) = l_1$$
$$M_5 = \{p_4, p'_2, p''_1\} \quad loc_5(A) = l_3$$

En el caso en el que el proceso decide buscar el servicio $s3$ la traza resultante es mucho más corta, ya que ese servicio no necesita autenticación.

$$\mathcal{M}_0[go\ l_2\rangle\mathcal{M}'_1[(s3?, s3!)\rangle\mathcal{M}'_2[go\ l_1\rangle\mathcal{M}'_3$$

donde si para $i = 1, 2, 3$ escribimos $\mathcal{M}'_i = (M'_i, loc'_i)$ entonces:

$$M'_1 = \{p_1, p'_2, p''_2\} \quad loc'_1(A) = l_2$$
$$M'_2 = \{p_1, p'_1, p''_3\} \quad loc'_2(A) = l_2$$
$$M'_3 = \{p_1, p'_1, p''_1\} \quad loc'_3(A) = l_1$$

Nótese que tras la ejecución de la primera traza se alcanzaba de nuevo el marcaje inicial, cosa que no ocurre en esta. Ello se debe a que el procesador L_2, después de ofrecer el servicio $s3$, necesita disparar su transición de sincronización $u!$ para poder volver a ofrecerlo, cosa que en este ejemplo no puede hacer sencillamente porque ningún proceso tiene la correspondiente $u?$.

3.4. Extensiones del modelo básico

Las redes ubicuas nos permiten definir sistemas de dos niveles, centrándonos en la oferta de servicios por parte de procesadores y su petición por parte de procesos móviles. Sin embargo, su simplicidad también tiene inconvenientes, que se pueden mitigar en parte con la introducción de algunas extensiones del modelo básico.

En primer lugar, en general los sistemas reales restringen tanto la movilidad de los procesos como el acceso a sus recursos. En particular, los procesos no emigran únicamente por sí solos, sino que son los procesadores los que los mueven. Por tanto, es razonable limitar el acceso a los procesadores dependiendo de su disponibilidad para recibir procesos. Es más, no sólo necesitarían el permiso del procesador donde quieren emigrar, sino que también necesitarían el permiso del procesador en el que se encuentran para moverse a cualquier otro.

En definitiva, resulta habitual tener que llevar a cabo una sincronización a tres bandas, en la que intervienen tanto el proceso que se quiere mover, como los procesadores de origen y de destino. Para ello, introducimos las etiquetas para transiciones *lgo l* y *lin*. Cuando en un procesador se puede disparar una transición etiquetada con *lgo l* se permite que un proceso que pueda disparar la transición

go l le abandone. Este movimiento sólo se puede llevar a cabo cuando el procesador destino, l, puede ejecutar la transición de admisión *lin*.

Definición 3.4.1 *Una* red go-procesador *es una red de Petri* $L = (P, T, F, \lambda)$, *donde:*

- P *y* T *son conjuntos finitos y disjuntos de lugares y transiciones, respectivamente,*
- $F \subseteq (P \times T) \cup (T \times P)$ *es el conjunto de arcos de la red,*
- λ *es una función de* T *a* $\mathcal{A} \cup S! \cup \mathcal{M}_r$, *donde* $\mathcal{M}_r = \{lin\} \cup \{lgo\ l \mid l \in \mathcal{N}_r\}$.

Definición 3.4.2 *Sea* N *un sistema ubicuo con redes go-procesador y sea* (M, loc) *un marcaje suyo. Una tupla de transiciones de movimiento,* $(t_1, t_2, t_3) \in T^3$ *con* $t_1 \in T_{l_1}$, $t_2 \in T_k$ *y* $t_3 \in T_{l_2}$, $\lambda(t_1) = lgo\ l_2$, $\lambda(t_2) = go\ l_2$ *y* $\lambda(t_3) = lin$, *y* $loc(A_k) = l_1$, *está activada para el marcaje si* ${}^\bullet t_1 + {}^\bullet t_2 + {}^\bullet t_3 \subseteq M$. *El marcaje de* $\mathcal{N}$ *alcanzado tras el disparo de* (t_1, t_2, t_3) *es* (M', loc'), *donde*

- $M' = M - \sum_{v \in \{t_1, t_2, t_3\}} {}^\bullet v + \sum_{v \in \{t_1, t_2, t_3\}} v^\bullet$,
- loc' *es la función de localización para* $\mathcal{N}$ *que coincide con* loc *salvo para la red proceso* A_k, *para la que se tiene* $loc(A_k) = l_2$.

En este modelo extendido se podrían añadir fácilmente condiciones de autenticación: podríamos sustituir la etiqueta de transiciones *lin* por la colección de etiquetas lin_a, donde a recorre los nombres de los procesos, es decir, el nombre del proceso que está dispuesto a recibir el procesador. De esta manera, para limitar el acceso a algunos servicios la red procesador podría discriminar entre los distintos procesos del sistema, eso sí, de manera estática. En el próximo capítulo introducimos un mecanismo más flexible para abordar este problema.

Es fácil ver que las extensiones de esta sección también son reducibles a redes P/T mediante construcciones similares a las de la definición 3.2.1, de modo que no introducen expresividad extra, por lo que no las consideraremos a partir de ahora. Para ello bastaría con modificar la construcción en la demostración de la proposición 3.2.3, añadiendo en la simulación transiciones (t_1, t_2, t_3) como las que aparecen en la anterior definición a T^*, en lugar de las $t_{go}(k, l)$ que teníamos en la definición 3.2.1.

Como hemos visto, en este modelo los procesos deben incluir en su código información concreta acerca de sus movimientos, aunque en ocasiones estos se pueden producir de manera no determinista (cuando existe una elección entre distintas transiciones de movimiento). Es más, los procesos deben tener un conocimiento estático de los nombres de los procesos cuya entrada permiten. Esta suposición es muy restrictiva, comparable a a situación que se tienen en álgebras de procesos de conectividad dinámica como CCS o CSP. Además, como comentábamos arriba, de momento estamos desentendiéndonos totalmente de la seguridad. Los servicios

se ofrecen de manera anónima, de manera que cualquiera que esté dispuesto a recibirlo puede hacerlo, reaccionando el proceso que lo ofrece igual ante todos ellos. En la próxima sección extenderemos el modelo presentado en este capítulo para superar estos dos problemas.

Capítulo 4

Redes de Petri móviles

En el capítulo anterior hablábamos de procesadores y procesos. Los procesos podían migrar entre procesadores y sincronizar con estos. Sin embargo, procesos y procesadores eran esencialmente lo mismo: redes de Petri capaces de sincronizar con otras, algunas de ellas móviles. En el refinamiento del modelo que vamos a llevar a cabo en este apartado para tratar con problemas de seguridad, vamos a empezar por eliminar esa diferencia entre procesos y procesadores. Por tanto, hablaremos sencillamente de componentes, abstrayendo de su naturaleza. Estas componentes podrán o no moverse, en la medida en la que tengan o no transiciones de movimiento. Es más, en lugar de encontrarse en un procesador, diremos que se encuentran en una localidad (que podrá o no contener componentes inmóviles), para lo que consideraremos un conjunto $\mathcal{L}$ de nombres de localidades. Abstraemos del límite en el que son posibles las comunicaciones entre componentes, y suponemos que éstas son posibles siempre que las dos componentes se encuentren en la misma localidad. Por tanto, desde el punto de vista intuitivo, esta generalización supone un acercamiento a la realidad ubicua, en la que distintos dispositivos físicos se comunican entre sí por medios inalámbricos y en la que los dispositivos físicos también pueden ser móviles. Además, de esta manera se prescinde de la necesidad de tener que asignar un nombre único y global para cada procesador, como teníamos en el capítulo anterior, lo que de hecho era poco realista en el campo de la computación ubicua.

Con la intención de realizar una presentación progresiva de los modelos que aparecerán en esta tesis, hemos presentado en el capítulo anterior un modelo en el que se obvia totalmente el tema de la seguridad. Los servicios se ofrecen de manera totalmente uniforme, sin posibilidad de restringir su acceso de manera selectiva. En otras palabras, los servicios se ofrecen o bien a todo el mundo o bien a nadie. Asimismo, los servicios se reciben de manera anónima, de modo que no se puede, una vez ofrecido el servicio, saber quién fue su beneficiario. En este capítulo vamos a introducir un mecanismo que nos permitirá realizar todas estas acciones. Este mecanismo consiste en añadirles colores a nuestras redes, es decir, considerar tokens distinguibles, que actuarán como identificadores de las distintas

componentes.

Así que la seguridad se basa en el conocimiento de ciertos nombres, en la forma de tokens en las precondiciones. De esta manera se separa la seguridad de la coordinación, que dependía de las etiquetas de sincronización, que se podían ver como primitivas del estándar de coordinación usado, que son de dominio público.

Antes de introducir ese mecanismo, y para hacer una presentación incremental, vamos a extender el modelo de redes ubicuas con colores, pero no para la seguridad, sino para tener unas primitivas para la movilidad más dinámicas. En el capítulo anterior los destinos de los movimientos estaban especificados estáticamente en las etiquetas de las transiciones de movimiento, lo que indudablemente era demasiado restrictivo. Para eliminar este handicap, vamos a considerar también colores para las localidades, de manera que el destino de un movimiento se especifique dinámicamente como uno de los tokens localidad que se toman de las precondiciones.

4.1. Sistemas MSPN

Al igual que en el capítulo anterior, usaremos un conjunto $\mathcal{A}$ de etiquetas para transiciones autónomas, pero ahora consideraremos que además tenemos una etiqueta especial $go \in \mathcal{A}$. Además, usaremos como tokens los elementos del conjunto de localidades $\mathcal{L}$, de modo que $Tokens = \mathcal{L} \cup \{\bullet\}$. Denotaremos mediante $Sync$ al conjunto de etiquetas de sincronización $S? \cup S!$. Por último, usaremos un conjunto de variables Var, con una variable distinguida $\varepsilon \in Var$.

Definición 4.1.1 *Una* red de Petri móvil *es una red de Petri coloreada etiquetada* $N = (P, T, F, \lambda, C)$*, donde* P *y* T *son los conjuntos disjuntos y finitos de lugares y transiciones, respectivamente,* $F : (P \times T) \cup (T \times P) \to Var$ *es una función parcial,* $\lambda : T \to \mathcal{A} \cup Sync$ *es la función de etiquetado de las transiciones y* $C : P \to \{\bullet, \mathcal{L}\}$ *es la función de coloreado de lugares, tales que:*

1. *Para cada* p *y* t *tales que* $(t, p) \in Dom(F)$ *(resp.* $(p, t) \in Dom(F)$*),*

$$C(p) = \bullet \Leftrightarrow F(t, p) = \varepsilon \ (resp.\ F(p, t) = \varepsilon)$$

2. *Para cada* $t \in T$ *y* $p \in P$ *tales que* $\lambda(t) \in \mathcal{A}$*,* $C(p) = \mathcal{L}$ *y* $(t, p) \in Dom(F)$ *existe* p' *tal que* $C(p') = \mathcal{L}$ *y* $F(t, p) = F(p', t)$*.*

3. *Para cada* $t \in T$ *con* $\lambda(t) = go$*, existe un solo* p *con* $C(p) = \mathcal{L}$ *tal que* $(p, t) \in F$*.*

Escribiremos simplemente MSPN para referirnos a las redes de Petri móviles (por las siglas de su nombre en inglés, *mobile synchronizing Petri nets*). Por homogeneidad en la definición, estamos suponiendo que todos los arcos, es decir, todos los pares en el dominio de F, están etiquetados con una variable. Sin embargo, las variables en los arcos sólo son necesarias para especificar el flujo de tokens

localidad de las precondiciones a las postcondiciones. Por ello, hemos introducido la variable especial ε, que etiqueta todos los arcos adyacentes a lugares que contienen tokens ordinarios, como se dice en la primera condición. La segunda condición impone que toda variable (excepto ε) que aparezca en un arco saliente de una transición autónoma también debe aparecer en algún arco entrante en dicha transición, de manera que las transiciones autónomas sólo puedan mover, eliminar o incluso replicar los nombres de localidades, pero no crearlos. La última condición establece la existencia de un lugar precondición de tipo localidad distinguido para las transiciones de movimiento, usado, como veremos enseguida, para especificar el destino de los movimientos.

A las notaciones que usábamos en el capítulo anterior, les añadimos las siguientes. Escribiremos $pl(p,t) = pl(t,p) = p$, $tr(p,t) = tr(t,p) = t$, $P_{\mathcal{T}} = \{p \mid C(p) = \mathcal{T}\}$ para $\mathcal{T} \in \{\bullet, \mathcal{L}\}$, ${}^{\bullet}t_{\mathcal{T}} = {}^{\bullet}t \cap P_{\mathcal{T}}$, $t^{\bullet}_{\mathcal{T}} = t^{\bullet} \cap P_{\mathcal{T}}$, $post(t) = \{F(t,p) \mid p \in t^{\bullet}_{\mathcal{L}}\}$, $pre(t) = \{F(p,t) \mid p \in {}^{\bullet}t_{\mathcal{L}}\}$ y $Var(t) = pre(t) \cup post(t)$. Con estas notaciones, las tres condiciones de la definición anterior se pueden sintetizar de la siguiente manera:

1. Para todo $a \in Dom(F)$, $C(pl(a)) = \bullet \Leftrightarrow F(a) = \varepsilon$.
2. Para todo $t \in T$ tal que $\lambda(t) \in \mathcal{A}$, $post(t) \setminus \{\varepsilon\} \subseteq pre(t)$.
3. Para cada $t \in T$ con $\lambda(t) = go$, $|{}^{\bullet}t_{\mathcal{L}}| = 1$.

Definición 4.1.2 *Un* marcaje M *de una MSPN* $N = (P, T, F, \lambda, C)$ *es una función* $M : P \to \mathcal{MS}(Tokens)$ *tal que* $M(P_{\mathcal{T}}) \in \mathcal{MS}(\mathcal{T})$ *para* $\mathcal{T} \in \{\mathcal{L}, \bullet\}$.

Un marcaje de un sistema MSPN no es sólo una tupla de marcajes, una por cada red, pues las localidades en las que se encuentra cada componente también han de formar parte del estado.

Definición 4.1.3 *Un* sistema MSPN *es un conjunto* $\mathcal{N}$ *de MSPNs disjuntas. Un marcaje* $\mathcal{M}$ *de un sistema MSPN es un par* (M, loc)*, donde* M *es un conjunto de marcajes, uno por cada componente de* $\mathcal{N}$ *y* $loc : \mathcal{N} \to \mathcal{L}$ *es una función que asigna a cada componente su localización.*

Cuando no haya confusión escribiremos $P = \bigcup_{i=1}^{n} P_i$, $M(p)$ para denotar $M_i(p)$, cuando $p \in P_i$; $F(a)$ para $F_i(a)$, si $a \in F_i$; y $\lambda(t)$ para $\lambda_i(t)$, cuando $t \in T_i$. Con estas notaciones los marcajes serían pares (M, loc) con $M : P \to \mathcal{MS}(Tokens)$.

Como ocurre en cualquier tipo de red de Petri de alto nivel, las transiciones se pueden disparar en distintos *modos*. Un modo especifica qué tokens toma la transición de cada una de sus precondiciones y qué otros añade en cada una de las postcondiciones. Para nosotros serán simplemente funciones del conjunto de variables que rodean una transición al conjunto de tokens. Por homogeneidad, hemos etiquetado también a los arcos adyacentes a lugares de tipo $\bullet$ con la variable

especial ε. Por tanto, será esa variable, y sólo ella, la que pueda instanciarse al token $\bullet$. En los dibujos supondremos que los arcos no etiquetados lo están con la variable ε.

Definición 4.1.4 *Dado un sistema MSPN, llamaremos* modo *de una transición $t \in T$ a cualquier función $\sigma : Var(t) \to Tokens$ tal que*

$$\sigma(x) = \bullet \Leftrightarrow x = \varepsilon.$$

Definamos ahora cómo se disparan los distintos tipos de transiciones que hemos ido viendo. En lo sucesivo, supondremos para simplificar la notación que $\sigma(F(a)) = \emptyset$ cuando $a \notin Dom(F)$.

Definición 4.1.5 *Sea $\mathcal{N}$ un sistema MSPN, $t \in T_i$ con $\lambda(t) \in \mathcal{A}$ y $\mathcal{M} = (M, loc)$ un marcaje de $\mathcal{N}$. La transición t está activada en modo σ si para todo $p \in {}^\bullet t$, $\sigma(F(p,t)) \in M(p)$. Entonces decimos que t se puede disparar, y el marcaje de $\mathcal{N}$ alcanzado tras el disparo de t en modo σ es $\mathcal{M}' = (M', loc')$, donde:*

1. *Para todo $p \in P$, $M'(p) = M(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$.*

2. *Si $\lambda(t) \neq go$ entonces $loc' = loc$. En otro caso, $loc'(N_j) = loc(N_j)$ para todo $j \neq i$ y $loc'(N_i) = \sigma(F(p,t))$, donde $p \in {}^\bullet t_{\mathcal{L}}$.*

Dadas un par de transiciones de sincronización t_1 y t_2, escribiremos $pre(t_1, t_2) = pre(t_1) \cup pre(t_2)$, $post(t_1, t_2) = post(t_1) \cup post(t_2)$ y $Var(t_1, t_2) = pre(t_1, t_2) \cup post(t_1, t_2)$. Veamos entonces cuándo dos transiciones de sincronización son compatibles. Las condiciones de compatibilidad se pueden ver intuitivamente como el protocolo de comunicaciones o de coordinación que se está usando, por lo que son condiciones meramente sintácticas: por un lado, las etiquetas de las transiciones han de ser conjugadas, es decir, una debe de ofrecer un servicio y la otra solicitar el mismo servicio; por otro lado, las dos transiciones deben cumplir conjuntamente la misma condición que se impuso a las transiciones autónomas, es decir, toda variable que aparezca en algún arco saliente de cualquiera de las transiciones debe aparecer también en alguno de los arcos entrantes de una de las dos.

Definición 4.1.6 *Dadas dos transiciones t_1 y t_2 de distintas componentes de un sistema MSPN, diremos que son* compatibles *si:*

1. *$\lambda(t_1) = s?$ y $\lambda(t_2) = s!$ para algún $s \in S$,*

2. *$post(t_1, t_2) \setminus \{\varepsilon\} \subseteq pre(t_1, t_2)$.*

La definición de modo para un par de transiciones de sincronización es análoga a la definición 4.1.4:

Definición 4.1.7 *Sea* (t_1, t_2) *un par de transiciones compatibles de un sistema MSPN. Llamaremos* modo *de* (t_1, t_2) *a cualquier función* $\sigma : Var(t_1, t_2) \to Tokens$ *tal que*

$$\sigma(x) = \bullet \Leftrightarrow x = \varepsilon.$$

Definamos ahora cuándo un par de transiciones de sincronización está activado en algún modo y qué marcaje se alcanza tras su disparo:

Definición 4.1.8 *Sea* $\mathcal{N}$ *un sistema MSPN,* t_i *y* t_j *dos transiciones compatibles y* $\mathcal{M} = (M, loc)$ *un marcaje de* $\mathcal{N}$*. Diremos que el par* (t_i, t_j) *está activado en modo* σ *si para todo* $p \in {}^\bullet t_i \cup {}^\bullet t_u$, $\sum_{h \in \{i,j\}} \{\sigma(F(p, t_h))\} \subseteq M(p)$*. El estado de* $\mathcal{N}$ *alcanzado tras el disparo de* (t_i, t_j) *en modo* σ *es el marcaje* $\mathcal{M}' = (M', loc)$ *donde para cada* $p \in P$,

$$M'(p) = M(p) - \sum_{h \in \{i,j\}} \{\sigma(F(p, t_h))\} + \sum_{h \in \{i,j\}} \{\sigma(F(t_h, p))\}$$

Tanto en el caso del disparo de una transición autónoma como en el de un par de transiciones de sincronización, escribiremos $\mathcal{N}(\mathcal{M})[u(\sigma)\rangle\mathcal{N}(\mathcal{M}')$ (omitiendo $\mathcal{N}$ cuando no exista confusión) para indicar que desde el marcaje $\mathcal{M}$ se ha obtenido $\mathcal{M}'$ después del disparo de la transición $u = t$ o del par $u = (t_1, t_2)$, en modo σ. Diremos que un marcaje $\mathcal{M}$ es alcanzable desde el estado inicial $\mathcal{M}_0$ si existe una cadena

$$\mathcal{N}(\mathcal{M}_0)[u_1(\sigma_1)\rangle\mathcal{N}(\mathcal{M}_1)[u_2(\sigma_2)\rangle \ldots \mathcal{N}(\mathcal{M}_n)[u_n(\sigma_n)\rangle\mathcal{N}(\mathcal{M})$$

Llamaremos traza de $\mathcal{N}$ a cada secuencia como la anterior y denotaremos por $[\![\mathcal{N}]\!]$ al conjunto de trazas de $\mathcal{N}$.

Por último, consideramos el siguiente orden entre marcajes:

$$(M, loc) \sqsubseteq (M', loc) \Leftrightarrow M(p) \subseteq M'(p) \quad \forall p \in P$$

Un caso especial de sistemas MSPN son aquellos divididos en dos conjuntos disjuntos de redes, como teníamos en el capítulo anterior, es decir, un conjunto de redes procesador $\mathcal{N}_P = \{N_1, \ldots, N_k\}$ y un conjunto de redes proceso $\mathcal{N}_A = \{N_{k+1}, \ldots, N_n\}$, de modo que $\mathcal{L} = \mathcal{N}_P$, $\lambda(t) \in \mathcal{A} \setminus \{go\} \cup S!$ para toda transición t de $\mathcal{N}_P$ y $\lambda(t) \in \mathcal{A} \cup S?$ para cada t de $\mathcal{N}_A$.

4.2. Ejemplo

En esta sección presentaremos un ejemplo que ilustra los conceptos introducidos en las anteriores definiciones. El proceso de la figura 4.1, situado en una localidad llamada h, es un enrutador que envía a *cualquier* otro proceso que lo solicite a una de las localidades k de un conjunto K de localidades donde se ofrece cierto servicio. Más concretamente, el enrutador comunica una de las localidades en K cuando se sincroniza con él por medio de la transición *ask*.

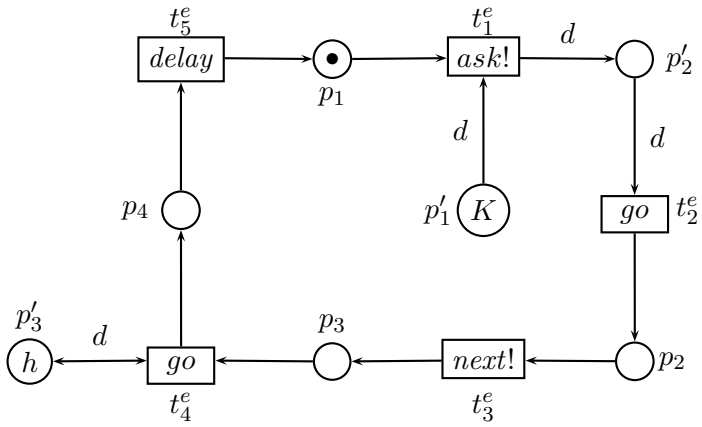

Figura 4.1: Enrutador

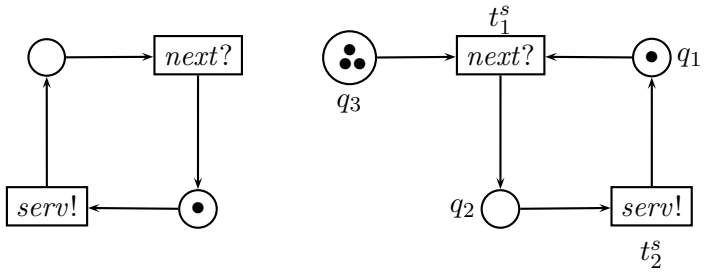

Figura 4.2: Servidores

Cada vez que el enrutador asigna a un proceso solicitante una de estas localidades, él mismo se desplaza a ella (en un ejemplo más elaborado podría enviar a un agente en lugar de desplazarse él mismo) para informar mediante la transición de sincronización etiquetada por *next* de que en un futuro próximo un nuevo proceso puede llegar demandando el servicio. Los procesos que ofrecen el servicio *serv*, situados en localidades distintas $k \in K$, pueden ser como el de la figura 4.2. El de la derecha sería una versión restringida del de la izquierda, que puede ofrecer el servicio solamente un determinado número de veces (tres en este caso).

Sólo tras la ejecución del protocolo anterior se ofrecerá finalmente el servicio *serv*. Nos preguntamos ahora qué propiedades tiene el servicio descrito en presencia de los clientes. Consideremos en primer lugar el cliente descrito en la figura 4.3, que interactuaría con el sistema de la manera deseada y esperada. Vamos a ilustrar este comportamiento a través de un cómputo generado por el enrutador, el servidor (suponemos que trabajamos con el de la derecha de la figura 4.2) y ese cliente. Vamos a escribir los marcajes como tuplas (M^e, M^s, M^c, loc) donde M^e es un marcaje del enrutador, M^s es un marcaje del servidor y M^c lo es del cliente. Además, escribimos $K' = K \setminus \{k\}$ y cada M^a con $a \in \{e, s, c\}$ lo representaremos mediante una secuencia $(M^a|_{P_\bullet},\ p_1 : M^a(p_1), \ldots, p_n : M^a(p_n))$ para todo p_i tal que $M^a(p_i) \neq \emptyset$. Escribiremos $u(a)$ para denotar el disparo de u en modo

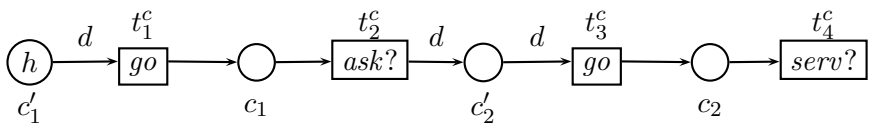

Figura 4.3: Cliente honesto

M_i^a	$a = e$	$a = s$	$a = c$
$i = 0$	$(\{p_1\}, p_1' : K, p_3' : \{h\})$	$\{q_1, q_3, q_3, q_3\}$	$(c_1' : \{h\})$
$i = 1$	$(p_1' : K', p_2' : \{k\}, p_3' : \{h\})$	$\{q_2, q_3, q_3\}$	$\{c_1\}$
$i = 2$	$(\{p_2\}, p_1' : K', p_3' : \{h\})$	$\{q_1, q_3, q_3\}$	$(c_2' : \{k\})$
$i = 3$	$(\{p_3\}, p_1' : K', p_3' : \{h\})$		$\{c_2\}$
$i = 4$	$(\{p_4\}, p_1' : K', p_3' : \{h\})$		$\emptyset$
$i = 5$	$(\{p_1\}, p_1' : K', p_3' : \{h\})$		

Figura 4.4: Marcajes de una traza con un cliente honesto

$[\varepsilon \to \bullet, d \to a]$ y simplemente u para el disparo de u en modo $[\varepsilon \to \bullet]$, pues esas son las únicas variables que aparecen en todas las transiciones y pares de transiciones compatibles del ejemplo. Por último, como el servidor está siempre en k representaremos la función de localización simplemente con un par $\langle l_1, l_2 \rangle$, de modo que l_1 sea la localidad del enrutador y l_2 la del cliente. Entonces la siguiente es la traza en la que el cliente de la figura 4.3 obtiene el servicio. Las definiciones de cada marcaje se pueden ver en la figura 4.4.

$$(M_0^e, M_0^s, M_0^c, \langle h, l \rangle) \xrightarrow{t_1^c(h)} (M_0^e, M_0^s, M_1^c, \langle h, h \rangle) \xrightarrow{(t_2^c, t_1^e)(k)} (M_1^e, M_0^s, M_2^c, \langle h, h \rangle) \xrightarrow{t_3^c(k)}$$

$$(M_1^e, M_0^s, M_3^c, \langle h, k \rangle) \xrightarrow{t_2^e(k)} (M_2^e, M_0^s, M_3^c, \langle k, k \rangle) \xrightarrow{(t_1^s, t_3^e)} (M_3^e, M_1^s, M_3^c, \langle k, k \rangle) \xrightarrow{(t_4^c, t_2^s)}$$

$$(M_3^e, M_2^s, M_4^c, \langle k, k \rangle) \xrightarrow{t_4^e(h)} (M_4^e, M_2^s, M_4^c, \langle h, k \rangle) \xrightarrow{t_5^e} (M_5^e, M_2^s, M_4^c, \langle h, k \rangle)$$

Podríamos además tener clientes como el descrito en la figura 4.5, que intentan usar su *permiso* más de una vez. Después de sincronizar en *serv* pueden moverse a otra localidad, hacer *algo* y volver a la localidad del servidor que conoce sin tener que volver a ejecutar *ask*. Claramente, sólo podrán sincronizar de nuevo para obtener el servicio si esa localidad estaba en K más de una vez y el enrutador la selecciona de nuevo. En ese caso se avisaría al servidor de que puede arribar un nuevo cliente, momento que podría aprovechar el cliente malicioso para robarle el servicio.

Formalmente, si escribimos M^c para referirnos a los marcajes del cliente de la figura 4.5 y $M^{c'}$ para hacer los propio con el cliente de la figura 4.3 entonces

se podría ejecutar la traza anterior (usando $t_i'^c$ cada vez que allí se usaba t_i^c), pero obteniendo un marcaje $M_4^c = (\{c_2\}, c_2' : \{k\})$, que podría convertirse, tras el disparo autónomo de *go* y *algo* en $M_5^c = (\{c_1\}, c_2' : \{k\})$, localizada en algún $l' \in L$. El marcaje M_5^e coincide con M_0^e excepto por el hecho de que $p_1' : K'$ en el primero y $p_1' : K$ en el segundo. También coinciden M_0^s y M_2^s excepto por el hecho de que el segundo tiene un token menos en q_3. Si denotamos $M_i'^e$ al marcaje M_i^e con $K \setminus \{k\}$ en p_1' y $M_i'^s$ a M_i^s con un token menos en q_3 entonces, suponiendo que $k \in K'$, se podría volver a disparar la secuencia anterior, pero esta vez *primada*, con $M_0^{c'}$ y en presencia (de momento inadvertida) de M_5^c.

$$(M_0'^e, M_0'^s, M_0^{c'}, M_5^c, \langle h, l, l' \rangle) \xrightarrow{t_1^c(h)} (M_0'^e, M_0'^s, M_1^{c'}, M_5^c, \langle h, h, l' \rangle) \xrightarrow{(t_2^c, t_1^e)(k)}$$

$$(M_1'^e, M_0'^s, M_2^{c'}, M_5^c, \langle h, h, l' \rangle) \xrightarrow{t_3^c(k)} (M_1'^e, M_0'^s, M_3^{c'}, M_5^c, \langle h, k, l' \rangle) \xrightarrow{t_2^e(k)}$$

$$(M_2'^e, M_0'^s, M_3^{c'}, M_5^c, \langle k, k, l' \rangle) \xrightarrow{(t_1^s, t_3^e)} (M_3'^e, M_1'^s, M_3^{c'}, M_5^c, \langle k, k, l' \rangle)$$

donde añadimos una tercera componente a las funciones de localización representando la posición del cliente malintencionado. Llegado este punto podrían ocurrir dos cosas. Podrían sincronizar $M_3^{c'}$ y $M_1'^s$ como hizo antes M_3^c. Sin embargo, también podría intervenir el cliente malintencionado, aprovechando el momento para moverse a la localidad k y robarle el servicio:

$$(M_3'^e, M_1'^s, M_3^{c'}, M_5^c, \langle k, k, l' \rangle) \xrightarrow{t_3'^c(k)} (M_3'^e, M_1'^s, M_3^{c'}, M_6^c, \langle k, k, k \rangle) \xrightarrow{t_4'^c}$$

$$(M_3'^e, M_2'^s, M_3^{c'}, M_4^c, \langle k, k, l' \rangle)$$

donde $M_c^6 = (\{c_2\}, c_2' : \{k\})$ e, intuitivamente, el cliente malintencionado le ha robado el servicio al cliente honesto. Nótese que, además, los estados del enrutador y del servidor vuelven a ser los análogos al estado inicial (una localidad k menos en p_1' y un token menos en q_3), que serían también los estados alcanzados si el cliente honesto hubiese obtenido su servicio, lo que se puede interpretar como que el enrutador y el servidor no son conscientes de que se ha producido un ataque aunque ciertamente este no pasa inadvertido para el cliente honesto, quien ya difícilmente puede recibir el servicio.

Podrían aparecer también clientes como el de la figura 4.6, que ni siquiera intentan obtener el servicio, sino que se desplazan a una cierta localidad en L, donde están dispuestos a *vender* su permiso. Entonces pueden seguir estafando, vendiendo el mismo permiso, hasta que decidan escaparse mediante el disparo de la transición *escape* e ir a otra localidad de L para volver a venderlo allí.

Obviamente, el sistema es vulnerable ante cualquiera de los ataques que hemos visto, correspondientes a la existencia de clientes con un comportamiento distinto del esperado en principio. Todos ellos representan un uso ilegítimo de participantes legítimos, en el sentido de que conocen inicialmente la localidad h donde se puede

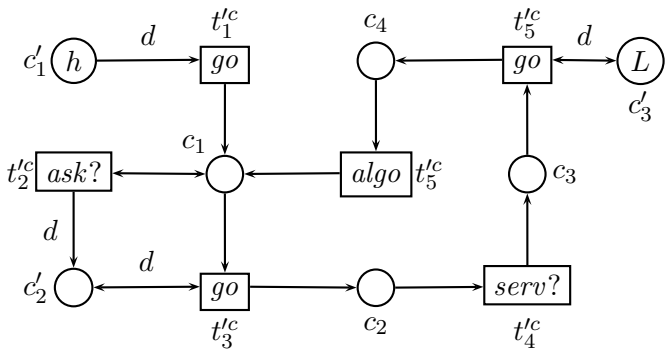

Figura 4.5: Cliente malintencionado

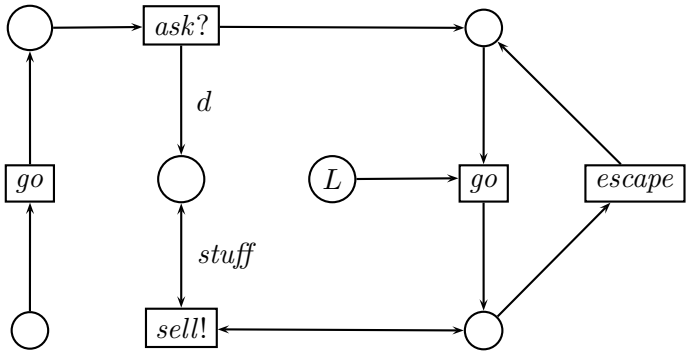

Figura 4.6: Contrabandista

pedir el servicio. Podemos interpretar la situación como el ofrecimiento de uso de un cierto servicio a una comunidad o grupo de usuarios, haciendo pública al efecto la localidad h. El uso adecuado del servicio es una responsabilidad compartida de todo el grupo. Los ejemplos anteriores serían ejemplos de usuarios del servicio que hacen un uso ilegítimo, no solidario del mismo.

Por tanto, solamente podemos asegurar que cualquier concesión del servicio *serv* se corresponde con una ejecución de *ask*, pero no, como hemos visto, que las partes que sincronizan en ambos casos sean las mismas. De hecho, en este modelo es imposible dar semejante garantía, ya que no existen mecanismos para discriminar a los distintos procesos. Discutiremos una forma de conseguirlo en la sección 4.4.

4.3. De los sistemas MSPN a los sistemas ubicuos

En el presente capítulo hemos extendido el modelo del capítulo anterior para hacer dinámico el conocimiento de localidades y permitir la especificación del destino de los movimientos también de manera dinámica. Para ver que los sistemas MSPN en efecto extienden a los sistemas ubicuos basta tener en cuenta que todo sistema ubicuo puede verse como un sistema MSPN. En efecto, si $\mathcal{N} = \langle \mathcal{N}_P, \mathcal{N}_A \rangle$ es

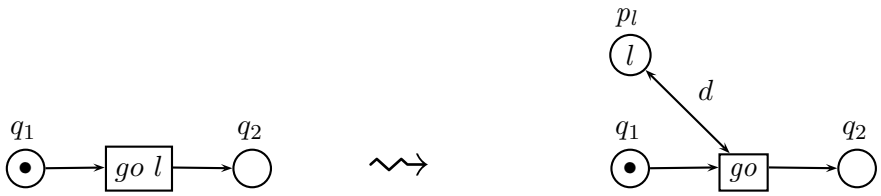

Figura 4.7: Representación de sistemas ubicuos con sistemas MSPN

un sistema ubicuo con $\mathcal{N}_P = \{l_1 : L_1, \ldots, l_m : L_m\}$ y $\mathcal{N}_A = \{A_1, \ldots, A_n\}$ entonces basta considerar el sistema MSPN $\mathcal{N}^* = \{L_1, \ldots, L_m, A_1^*, \ldots, A_n^*\}$ donde cada A_i^* tiene un lugar extra p_l para cada transición etiquetada por *go* l que contiene al token localidad l, unida sólo a esta a través de dos arcos etiquetados por la misma variable nueva d (véase la figura 4.7).

Además, como veremos a continuación, la flexibilización que supone el modelo presentado en este capítulo no añade poder expresivo a nuestro modelo, debido a que el conjunto de tokens localidad que pueden aparecer en los marcajes está fijado desde el principio.

Vamos a simular los sistemas MSPN usando sistemas ubicuos. Más concretamente, usaremos una variación de los sistemas ubicuos, que podemos llamar *sistemas ubicuos planos*, en la que no existe la distinción entre procesos y procesadores, de manera que todas las redes pueden sincronizar entre sí (no sólo procesos con procesadores), y las componentes pueden moverse o no solamente en la medida en la que tengan transiciones de movimiento o no. Por lo tanto, un sistema ubicuo plano es sencillamente un conjunto de redes componentes, donde cada red componente es una red P/T etiquetada con etiquetas de elementos en $\mathcal{A} \cup \mathit{Sync}$ donde *go* $k \in \mathcal{A}$ para todo $k \in \mathcal{L}$, y su comportamiento es análogo al de los sistemas ubicuos del anterior capítulo.

Proposición 4.3.1 *Las redes P/T pueden simular monótonamente a los sistemas ubicuos planos.*

Demostración. Una construcción idéntica a la vista en la definición 3.2.1 nos da una red P/T a partir de un sistema ubicuo plano. En aquel caso, se añadía una serie de lugares para cada agente, uno por cada procesador. En este caso hay que añadir un lugar por cada red y cada nombre de localidad que aparece en el marcaje inicial. Además, las transiciones $(t_1, t_2, k) \in T^*$ cumplían que t_1 era una transición de un proceso A y t_2 de un procesador llamado k, por lo que bastaba añadir la condición de que (A, k) fuese precondición y postcondición de (t_1, t_2, k). En esta caso, t_1 y t_2 pueden ser dos transiciones cualesquiera, y por lo tanto, habría que añadir arcos para que tanto la red que dispara t_1 como la que dispara t_2 estén en la localidad k. Entonces la tesis se sigue siguiendo las demostraciones de las proposiciones 3.2.3 y 3.2.4.

□

De ahora en adelante, siempre que hablemos de sistemas ubicuos nos referiremos a sistemas ubicuos planos.

Veamos entonces que los sistemas ubicuos simulan los sistemas MSPN. Como no hay mecanismo de creación de nuevas localidades, el conjunto K de tokens localidad que pueden aparecer en cualquier marcaje alcanzable es precisamente el conjunto de localidades que aparecen en el marcaje inicial, que es finito. Por tanto, podemos simular la aparición de un token $k \in K$ en el lugar p mediante un token negro en un lugar específico para k, p_k. Además, tendremos una transición distinta, que denotaremos $t(\sigma)$, para simular el disparo de t en modo σ, que tenga a p_k como precondición o postcondición, sólo si σ instancia a la variable que se encuentre en el arco correspondiente a k. Veámoslo formalmente.

Definición 4.3.2 *Sea $\mathcal{N} = \{N_1, \ldots, N_n\}$ un sistema MSPN con $N_i = (P_i, T_i, F_i, \lambda_i, C_i)$ y (M, loc) el marcaje inicial de $\mathcal{N}$. Si K es el conjunto de todos los elementos de $\mathcal{L}$ que aparecen en (M, loc) (que es finito) definimos el sistema ubicuo $\mathcal{N}^* = \{N_1^*, \ldots, N_n^*\}$ con $N_i^* = (P_i^*, T_i^*, F_i^*, \lambda_i^*)$ de la siguiente manera:*

1. $P_i^* = \{p_k \mid p \in P_i, C_i(p) = \mathcal{L}, k \in K\} \cup \{p \in P_i \mid C_i(p) = \bullet\}$.

2. $T_i^* = \{t(\sigma) \mid t \in T_i,\ \lambda(t) \in \mathcal{A},\ \sigma \text{ modo de } t\} \cup$
$\{(t,t')(\sigma) \mid t \in T_i,\ t' \in T,\ t \text{ y } t' \text{ compatibles},\ \sigma \text{ modo de } (t,t')\}$

3. $$\lambda_i^*(t(\sigma)) = \begin{cases} \lambda_i(t) \text{ si } \lambda_i(t) \neq go. \\ go\ k \text{ si } \lambda_i(t) = go \text{ y } \sigma({}^\bullet t_{\mathcal{L}}) = \{k\}. \end{cases}$$
$$\lambda_i^*((t,t')(\sigma)) = \begin{cases} s(t,t')! \text{ si } \lambda_i(t) = s!. \\ s(t',t)? \text{ si } \lambda_i(t) = s?. \end{cases}$$

4. F_i^* *es el menor conjunto que, para cada $(p,t) \in F_i$ (y análogamente para $(t,p) \in F_i$), satisface las condiciones siguientes:*

   a) $\lambda_i(t) \in \mathcal{A}$ y $C_i(p) = \bullet \Rightarrow (p, t(\sigma)) \in F_i^*$, *para todo* $t(\sigma) \in T_i^*$.
   b) $\lambda_i(t) \in \mathcal{A}$ y $C_i(p) = \mathcal{L} \Rightarrow \{(p_k, t(\sigma)) \mid \sigma(F_i(p,t)) = k\} \subseteq F_i^*$.
   c) $\lambda_i(t) \notin \mathcal{A}$ y $C_i(p) = \bullet \Rightarrow (p, (t,t')(\sigma)) \in F_i^*$, *para todo* $(t,t')(\sigma) \in T_i^*$.
   d) $\lambda_i(t) \notin \mathcal{A}$ y $C_i(p) = \mathcal{L} \Rightarrow \{(p_k, (t,t')(\sigma)) \mid \sigma(F_i(p,t)) = k\} \subseteq F_i^*$.

Los marcajes de la red original y los de su simulación están relacionados de la siguiente manera.

Definición 4.3.3 *Dado un sistema MSPN $\mathcal{N}$ y un marcaje (M, loc) de $\mathcal{N}$, definimos $(M, loc)^*$ como el marcaje de $\mathcal{N}^*$ dado por (M^*, loc) tal que:*

- $M^*(p) = M(p)$ *para cada* $p \in P_\bullet$.
- $M^*(p_k) = M(p)(k)$ *para todo* $p_k \in P^*$.

Proposición 4.3.4 *Dado un sistema MSPN $\mathcal{N}$ y un marcaje (M, loc) de $\mathcal{N}$, para el sistema ubicuo asociado $\mathcal{N}^*$ se tiene*

$$\mathcal{N}(M_1, loc_1)\ [u(\sigma)\rangle\ \mathcal{N}(M_2, loc_2) \Longleftrightarrow \mathcal{N}^*(M_1, loc_1)^*\ [v\rangle\ \mathcal{N}^*(M_2, loc_2)^*$$

Además:

- $\lambda(u) \in \mathcal{A} \Leftrightarrow v = u(\sigma)$.
- $u = (t_1, t_2) \Leftrightarrow v = ((t_1, t_2)(\sigma), (t_2, t_1)(\sigma))$.
- $\lambda(u) = go$, $\sigma({}^\bullet u_{\mathcal{L}}) = \{k\} \Leftrightarrow v = u(\sigma)$ *y* $\lambda(v) = go\ k$.

Demostración. En primer lugar, téngase en cuenta que por definición de multiconjunto tenemos que $\{\sigma(F(p,t))\}(k) = \begin{cases} 1 \text{ si } \sigma(F(p,t)) = k \\ 0 \text{ en otro caso} \end{cases}$

- Por hipótesis $\sigma(F(p,u)) \in M_1(p)$ para todo $p \in P$ y $M_2(p) = M_1(p) - \{\sigma(F(p,u))\} + \{\sigma(F(u,p))\}$. Veamos en primer lugar que la transición $u(\sigma)$ está activada. Por la definición 4.3.2 tenemos que ${}^\bullet u(\sigma) = \{p \in P \mid p \in {}^\bullet u\} \cup \{p_k \mid p \in {}^\bullet u,\ \sigma(F(p,u)) = k\}$ y $u(\sigma)^\bullet = \{p \in P \mid p \in u^\bullet\} \cup \{p_k \mid p \in u^\bullet,\ \sigma(F(u,p)) = k\}$. Sea $p \in {}^\bullet u(\sigma)$. Si $p \in P$ entonces $p \in {}^\bullet u$ y $M_1^*(p) = M_1(p) > 0$. Si $p = q_k$ entonces $q \in {}^\bullet u$ y $k = \sigma(F(q,u)) \in M_1(q)$, con lo que $M_1^*(p) = M_1(q)(k) > 0$ y se tiene que $u(\sigma)$ está activada.

  Veamos que los cómputos se preservan. Para ello, nótese que por el punto 4a de la definición 4.3.2, ${}^\bullet u(\sigma)(p_k) = \{\sigma(F(p,u))\}(k)$ y $u(\sigma)^\bullet(p_k) = \{\sigma(F(u,p))\}(k)$. Entonces $M_1^*(p_k) - {}^\bullet u(\sigma)(p_k) + u(\sigma)^\bullet(p_k) = M_1(p)(k) - \{\sigma(F(p,u))\}(k) + \{\sigma(F(u,p))\} = M_2(p)(k) = M_2(p_k)$. Análogamente, por el punto 4b, se tiene también para todo $p \in P$, de modo que $M_2^* = M_1^* - {}^\bullet u(\sigma) + u(\sigma)^\bullet$ y la tesis se sigue. Para el recíproco basta seguir el razonamiento inverso.

- En el caso de que las transiciones de sincronización, se cumple que t_1 y t_2 son compatibles si y sólo si $(t_1, t_2)(\sigma)$ y $(t_2, t_1)(\sigma)$ lo son. En efecto, por la definición de λ^* en 4.3.2 tenemos que si $\lambda(t_1) = s?$ y $\lambda(t_2) = s!$ entonces $\lambda^*((t_1, t_2)(\sigma)) = s(t_2, t_1)?$ y $\lambda^*((t_2, t_1)(\sigma)) = s(t_2, t_1)!$, y recíprocamente. La condición de activación de las transiciones y su disparo se obtiene de manera análoga al punto anterior.

- De nuevo, la activación y disparo de la transición son similares a los puntos anteriores, de modo que nos centramos en los movimientos. Por definición de λ^* se tiene que $\lambda^*(u(\sigma)) = go\ k$ si y sólo si $\sigma({}^\bullet u_{\mathcal{L}}) = \{k\}$, por lo que en ambos casos, según las definiciones 4.1.5 y 3.1.7, la función de localización resultante es la que coincide con loc_1 en todo su dominio excepto en la componente que dispara la transición, donde toma el valor k.

□

En el teorema anterior estamos sobrecargando la notación, ya que a la izquierda de la equivalencia $u(\sigma)$ representa "el disparo de u en modo σ", mientras que a la derecha $v(\sigma)$ quiere decir "el disparo de la transición $v(\sigma)$".

Proposición 4.3.5 *Los sistemas ubicuos simulan monótonamente a los sistemas MSPN.*

Demostración. Veamos que estamos en las hipótesis de la proposición 2.6.7. Por el resultado anterior los simulan paso a paso.

Dado un marcaje $(\overline{M}, loc)$ de $\mathcal{N}^*$ definimos el marcaje (M, loc) de $\mathcal{N}$ dado por $M(p)(a) = M(p_a)$, que verifica que $(M, loc)^* = (M^*, loc) = (\overline{M}, loc)$, así que todos los marcajes son estables y, por tanto, todos los alcanzables también. Por último, $M_1(p) \subseteq M_2(p) \Leftrightarrow M_1(p)(a) \leq M_2(p)(a)$ para todo $a \Leftrightarrow M_1^*(p_a) \leq M_2^*(p_a)$ y por lo tanto $(M_1, loc_1) \sqsubseteq (M_2, loc_2) \Leftrightarrow (M_1^*, loc_1) \sqsubseteq (M_2^*, loc_2) \Leftrightarrow (M_1, loc_1)^* \sqsubseteq (M_2, loc_2)^*$ y tanto $(\ )^*$ como su inversa son monótonas.

□

Corolario 4.3.6 *La alcanzabilidad y el recubrimiento son decidibles para los sistemas MSPN.*

Demostración. Esas propiedades son decidibles para los sistemas ubicuos, como probamos en el corolario 3.2.5, y la construcción anterior las preserva por ser una simulación monótona.

□

4.4. Autenticación

El mecanismo que hemos visto en las secciones anteriores no nos permite tratar propiedades de seguridad que requieran autenticación. Por ejemplo, en el modelo anterior es imposible controlar el número de veces que un proceso obtiene un servicio o el orden en el que los servicios han de recibirse. Así que es necesario un modelo más robusto respecto de los temas de autenticación.

Proponemos un mecanismo de identificación local que es suficiente en algunas aplicaciones y no tan costoso en lo que respecta a las comunicaciones como una infraestructura de autenticación global clásica, que no siempre se puede permitir.

Tomaremos simplemente $\mathbb{N}$ como nuestro dominio de identificadores. Sin embargo, no queremos que los nombres producidos puedan ser falsificados por los demás. Para evitar esta posibilidad consideraremos para cada proceso un conjunto disjunto con los demás, de modo que el dominio de identificadores será $Id = \bigoplus_{i \in I} \mathbb{N}$. Así que en definitiva los identificadores tendrán la forma (i, m) con $i \in I$ y $m \in \mathbb{N}$. Entonces, para formalizar que los nombres no puedan ser falsificados, supondremos que cada $i \in I$ pertenece a un sólo proceso y que cada proceso sólo tiene constructores de pares (i, m) para los índices $i \in I$ que le pertenecen. Estos constructores

son simplemente transiciones especiales, etiquetadas con $succ \in \mathcal{A}$, que toman un token (i, m) de un lugar precondición distinguido y devuelven un token $(i, m+1)$ en algunos lugares postcondición. Por simplicidad, supondremos que hay una copia de los naturales por cada red, es decir, que $I = \{1, \ldots, |\mathcal{N}|\}$ y los identificadores de la forma (i, m) pertenecen a N_i.

Una vez definido el modelo con primitivas de autenticación, rara vez nos referiremos al anterior modelo, que no contaba con esas primitivas. Por ello, llamaremos sistemas MSPN al modelo con esas primitivas, y diremos sencillamente cuando sea necesario que el sistema no tiene creación de nombres o que no tiene identificadores.

Usaremos las siguientes nuevas notaciones: un conjunto de variables $Var = Var_\bullet \cup Var_{\mathcal{L}} \cup Var_{Id}$ con $Var_\bullet = \{\varepsilon\}$, $\tau^+, \tau^- \in Var_{Id}$ y un conjunto $Var_{Auth} \subset Var_{Id}$. A las variables de Var_{Auth} las llamaremos variables de autenticación. Tomamos $Tokens = \mathcal{L} \cup Id \cup \{\bullet\}$ y $CTokens = \mathcal{L} \cup Id$. Usando estas notaciones y las que vimos en la sección anterior tenemos las siguientes definiciones:

Definición 4.4.1 *Una* MSPN *es una red de Petri etiquetada $N = (P, T, F, \lambda, C)$ donde P es el conjunto finito de lugares, T es el conjunto finito de transiciones, disjunto del anterior, $F : (P \times T) \cup (T \times P) \to Var$ es una función parcial del conjunto de arcos a las variables, $C : P \to \{\mathcal{L}, Id, \bullet\}$ es una función que colorea los lugares y $\lambda : T \to \mathcal{A} \cup Sync$ es una función que etiqueta las transiciones, de modo que:*

1. *Para todo $a \in Dom(F)$, $C(pl(a)) = \mathcal{T} \Leftrightarrow F(a) \in Var_{\mathcal{T}}$,*
2. *Existe a lo sumo un $t \in T$ con $\lambda(t) = succ$. Si $\lambda(t) = succ$ entonces existe $c_t \in P$ con $C(c_t) = Id$ tal que:*
   - *$\tau^+, \tau^- \notin Var(t')$ para $t' \neq t$,*
   - *$F(t, c_t) = \tau^+$ y $F(t, p) \neq \tau^-$ para todo $p \in P$,*
   - *$F(p, t) = \tau^- \Leftrightarrow p = c_t$,*
3. *Para todo $t \in T$, $post(t) \setminus \{\varepsilon, \tau^+\} \subseteq pre(t)$,*
4. *Para todo $t \in T$ con $\lambda(t) = go$, $|{}^\bullet t_{\mathcal{L}}| = 1$.*

Así que una MSPN como las del capítulo anterior, sin creación de nombres, no es más que una MSPN según la definición anterior tal que $C(p) \neq Id$ para todo $p \in P$. De hecho, incluso si existen lugares de tipo identificador, podríamos seguir considerándolas como en el capítulo anterior cuando $\lambda(t) \neq succ$ para todo $t \in T$.

Definición 4.4.2 *Un* marcaje *M de una MSPN $N = (P, T, F, \lambda, C)$ es una función $M : P \to \mathcal{MS}(Tokens)$ tal que $M(P_{\mathcal{T}}) \subseteq \mathcal{MS}(\mathcal{T})$ para $\mathcal{T} \in \{\mathcal{L}, Id, \bullet\}$.*

Los sistemas MSPN se definen de modo similar a los de las secciones anteriores:

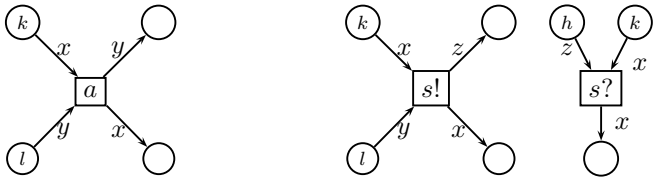

Figura 4.8: Transiciones autónomas (izquierda) y de sincronización (derecha)

Definición 4.4.3 *Un* sistema MSPN *es un conjunto* $\mathcal{N} = \{N_i\}_{i=1}^n$ *de MSPNs disjuntas. Un marcaje* $\mathcal{M}$ *de un sistema MSPN es un par* (M, loc) *donde* $M = (M_1, \ldots, M_n)$ *es tal que cada* M_i *es un marcaje de la MSPN* N_i *y* $loc : \mathcal{N} \to \mathcal{L}$ *es una función que lleva cada MSPN a una localidad.*

Para definir correctamente el comportamiento de la nueva transición especial *succ* tenemos que asegurar que se cumplen dos condiciones: por un lado, los modos de transiciones *succ* sólo pueden consumir tokens (i, m) que pertenezcan a esa red; por otro lado, el valor que toma τ^+ ha de ser el sucesor del valor tomado por τ^-.

Definición 4.4.4 *Con las notaciones anteriores, dada una transición* $t \in T_i$ *con* $\lambda(t) \in \mathcal{A}$*, un* modo *de* t *es una función* $\sigma : Var(t) \to Tokens$ *tal que*

1. $\sigma(x) \in \mathcal{T} \Leftrightarrow x \in Var_{\mathcal{T}}$ *para* $\mathcal{T} \in \{\bullet, \mathcal{L}, Id\}$.
2. *Si* $\tau^+, \tau^- \in Var(t)$ *entonces* $\sigma(\tau^-) = (i, m)$ *y* $\sigma(\tau^+) = (i, m+1)$ *para algún* $m \in \mathbb{N}$.

Al definir el modo de un par de transiciones podemos ignorar las nuevas variables, ya que éstas no aparecen en las sincronizaciones. De modo que la definición de modo de un par de transiciones síncronas será similar a la presentada en las secciones previas, excepto por las variables de autenticación.

Definición 4.4.5 *Dadas dos transiciones de sincronización* t_1 *y* t_2*, diremos que son* compatibles *si:*

1. $\lambda(t_1) = s?$ *y* $\lambda(t_2) = s!$ *para algún* $s \in S$,
2. $post(t_1, t_2) \setminus \{\varepsilon\} \subseteq pre(t_1, t_2)$,
3. *Si* $z \in pre(t_i) \cap Var_{Auth}$ *entonces* $z \in pre(t_j)$ *para* $i, j \in \{1, 2\}$, $i \neq j$.

Por medio de la sincronización obtenemos tanto la comunicación de identificadores como la restricción de esas comunicaciones. La comunicación se obtiene usando una variable en $post(t_1) \setminus pre(t_1)$ y en $pre(t_2)$ (o viceversa), como la variable z a la derecha de la figura 4.8. Se trata por tanto de una primitiva de comunicación síncrona, a través de un canal público. La restricción de las comunicaciones se obtiene gracias a las variables de autenticación, usando una variable en $pre(t_1) \cap Var_{Auth}$. Por definición de compatibilidad de transiciones de

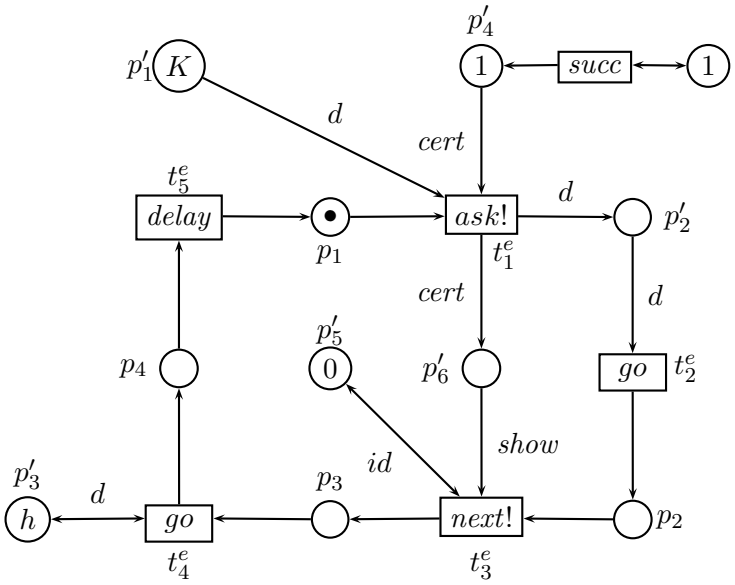

Figura 4.9: Enrutador con autenticación

sincronización, ha de ocurrir que esa variable también esté en $pre(t_2)$ (como la variable x a la derecha de la figura 4.8), lo que fuerza que la sincronización sólo tenga lugar cuando esa variable (la x del ejemplo) se pueda instanciar al mismo valor en las dos componentes, es decir, cuando ambas conocen dicho valor. Siguiendo la analogía con el cálculo π, si además de esa restricción en la interacción se establece una comunicación mediante el mecanismo antes mencionado, entonces se trataría de una comunicación a través del canal protegido que se llama como el valor al que se instancia la variable de autenticación.

Nótese que si para una transición t de sincronización existe una variable $x \in Var_{Auth}$ que etiqueta arcos postcondición, pero ninguno precondición, entonces ningún par (t, t') puede cumplir las condiciones de la anterior definición. Si $x \notin pre(t')$ entonces no se puede cumplir el segundo punto, y si $x \in pre(t')$ entonces no se puede cumplir el punto tercero. Por lo tanto, esa variable t nunca estará activada, así que de ahora en adelante podemos suponer que nuestros sistemas no tienen transiciones que cumplan esas condiciones.

Definición 4.4.6 *Dado un par de transiciones compatibles* (t, t')*, llamaremos* modo *suyo a una función* $\sigma : Var(t, t') \to Tokens$ *tal que* $\sigma(x) \in \mathcal{T} \Leftrightarrow x \in Var_{\mathcal{T}}$ *para* $\mathcal{T} \in \{\bullet, \mathcal{L}, Id\}$.

Con estas notaciones y definiciones, las reglas de disparo de transiciones autónomas y síncronas son idénticas a las expuestas en las secciones anteriores. Además, el hecho de haber introducido generación de identificadores no modifica la noción de orden considerado.

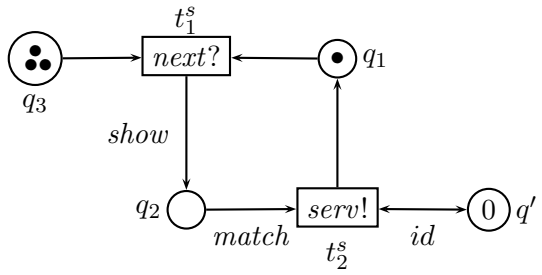

Figura 4.10: Servidores con autenticación

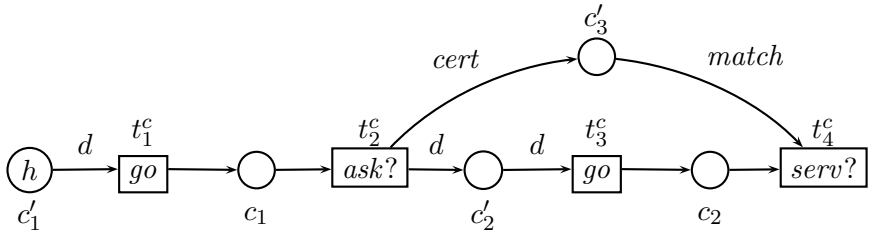

Figura 4.11: Cliente honesto

4.5. Ejemplo: protocolo de autenticación distribuido

El siguiente ejemplo es una extensión del visto en la sección 4.2, ilustrando los mecanismos de autenticación que acabamos de introducir. Ahora cuando un proceso le demanda el servicio al enrutador (véase la figura 4.9) este le devuelve no sólo la localidad donde el proceso debe ir, sino también un identificador único por medio de la variable *cert*, que servirá para garantizar que el único proceso que pueda acceder al servicio como consecuencia de esta solicitud sea el que enseñe el identificador facilitado. Asimismo, cuando el enrutador y el servidor sincronizan en las transiciones etiquetadas por *next* el enrutador le da al servidor el identificador que le proporcionó al cliente. Además, esta sincronización se asegura por medio de la variable de autenticación *id*, de manera que el enrutador solamente le puede dar el permiso correspondiente al servidor adecuado.

Para garantizar que el cliente que accede al servicio es el que lo demandó, el servidor debe comprobar que el proceso dispuesto a sincronizar con él dispone del ticket oportuno (ver figura 4.10), a través de la variable de autenticación *match*.

Por supuesto, habría que modificar también los clientes como corresponde. La figura 4.11 muestra al cliente honesto de la figura 4.3, modificado para que pueda seguir el nuevo protocolo. La única diferencia es el nuevo lugar c_3' donde almacena el identificador tras la transición *ask* y que utiliza para mostrarlo en el disparo de *serv*.

La primera traza de las vistas en el ejemplo de la sección 4.2 lo sigue siendo

M_i^a	$a = e$	$a = s$	$a = c$
$i = 0$	$(\{p_1\}, p'_1 : K, p'_3 : \{h\}, p'_4 : \{1\})$	$(\{q_1, q_3, q_3, q_3\}, q' : 0)$	$(c'_1 : \{h\})$
$i = 1$	$(p'_1 : K', p'_2 : \{k\}, p'_3 : \{h\}, p'_6 : \{1\})$	$(\{q_3, q_3\}, q_2 : \{1\}, q' : 0)$	$\{c_1\}$
$i = 2$	$(\{p_2\}, p'_1 : K', p'_3 : \{h\}, p'_6 : \{1\})$	$(\{q_1, q_3, q_3\}, q' : 0)$	$(c'_2 : \{k\}, c'_3 : \{1\})$
$i = 3$	$(\{p_3\}, p'_1 : K', p'_3 : \{h\})$		$(\{c_2\}, c'_3 : \{1\})$
$i = 4$	$(\{p_4\}, p'_1 : K', p'_3 : \{h\})$		$\emptyset$
$i = 5$	$(\{p_1\}, p'_1 : K', p'_3 : \{h\})$		

Figura 4.12: Marcajes de una traza con un cliente honesto usando autenticación

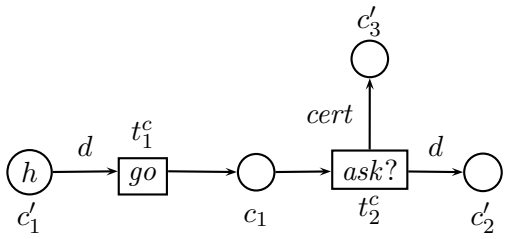

Figura 4.13: Ataque DoS

con las modificaciones vistas, si consideramos los marcajes dados por la figura 4.12 (omitiendo $p'_5 : \{0\}$, que aparece en M_i^e para todo i).

Se puede ver que ahora sí se evitarían los ataques que vimos en la sección 4.2. Efectivamente, como los identificadores no se pueden falsificar, cada uno de ellos se corresponde con una oferta del servicio. Por consiguiente, no tiene sentido intentar acceder al servicio más de una vez. Es cierto que un cliente podría ceder su permiso para acceder al servicio a otro proceso, pero en este caso perdería la posibilidad de acceder él mismo al servicio, a menos que "engañara" al correspondiente arrendatario quedándose con una copia del permiso y precipitándose a hacer uso del mismo. De modo que el sistema sería seguro si suponemos que no nos importa quién accede finalmente al servicio, a sabiendas de que en cualquier caso se trata o bien de quien recibió el permiso, o de algún delegado suyo.

Sin embargo, un ataque como el de la figura 4.13 aún podría darse. Este proceso sigue el protocolo hasta el momento en el que tiene que sincronizar con el servicio. En este punto se bloquea, de manera que no le puede dar servicio a más clientes, provocando así un ataque de denegación de servicio, (o DoS por sus siglas en inglés, *Denial of Service*). Otro ataque de este estilo podría consistir en recibir primero el servicio y después intentar recibirlo de nuevo, causando un bloqueo en el proceso que lo ofrece. Estas situaciones se pueden evitar añadiendo a dicho proceso una transición autónoma *abort* que reinicialice el sistema en ese momento.

4.6. Generación de nombres abstractos

En la sección anterior hemos introducido un mecanismo de generación de nombres basado en el manejo de naturales. Además, hemos restringido su uso para que sólo se puedan usar para implementar mecanismos de autenticación. En particular, esas restricciones evitan que se puedan usar los identificadores para simular contadores.

Hemos obtenido en efecto un mecanismo para implementar la autenticación. Sin embargo, el uso de números naturales como identificadores tiene varias consecuencias no deseadas: cuando se examina un marcaje desde fuera, se puede saber el orden en el que los identificadores han sido generados; además, se puede saber cuántos identificadores se han creado y puede constatarse el hecho de que algunos han existido temporalmente, aunque no se encuentren ya en el presente marcaje. Para evitar estas consecuencias podríamos tener un mecanismo de *recogida de basura*, que nos permitiera reutilizar identificadores, a la manera como se hace en [Del] para un modelo distinto.

Una aproximación alternativa, que elimina automáticamente del modelo estos problemas sin requerir recogida de basura explícita consiste en utilizar como dominio de identificadores un conjunto infinito arbitrario. La única diferencia entre el modelo anterior y el nuevo es que la transición sucesor que teníamos se sustituye por una transición *new*, que tiene algunos arcos que van a postcondiciones de tipo identificador etiquetados con una variable especial ν, que se instanciará a un identificador nuevo, no presente en el marcaje actual. Dado un marcaje (M, loc) definimos $S(M) = \bigcup_{p \in P} S(M(p))$. Entonces, el disparo de esta nueva transición queda definido como se espera:

Definición 4.6.1 *Sea $\mathcal{N}$ un sistema MSPN, $t \in T$ con $\lambda(t) = new$, y (M, loc) un marcaje de $\mathcal{N}$. Diremos que t está activada en modo σ si $\sigma(\nu) \notin S(M)$ y para cada $p \in {}^\bullet t$, $\sigma(F(p,t)) \in M(p)$. El marcaje de $\mathcal{N}$ alcanzado después del disparo de t en modo σ es (M', loc), donde para todo $p \in P$, $M'(p) = M(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$.*

Por tanto, en todas las postcondiciones de la transición a través de un arco etiquetado por ν aparece tras su disparo un token identificador que no aparece en ningún otro lugar del marcaje, el mismo en todos esos lugares. La recogida automática de basura que comentábamos anteriormente viene dada por la desaparición de identificadores del marcaje. Los identificadores que desaparecen sabemos con certeza que no van a volver a usarse a lo largo del cómputo, con lo que podrían ser reutilizados al crear nuevos identificadores, sin interferir con el comportamiento del sistema. En efecto, al no aparecer dicho identificador en el marcaje, la variable ν puede volver a instanciarse a él, con lo cual este puede volver a generarse.

Nótese que si bien la primitiva anterior está inspirada en la creación de nombres frescos del cálculo π, su implementación es sustancialmente distinta al no

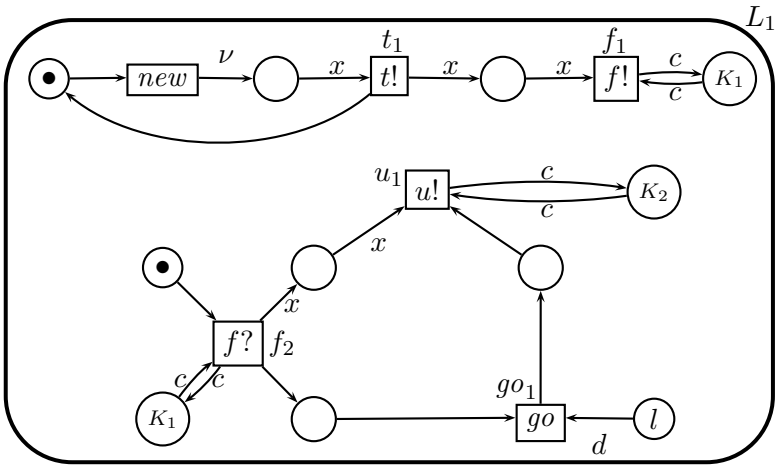

Figura 4.14: Servidor de tickets (arriba) y agente (abajo)

contar nuestro formalismo con un mecanismo explícito de definición de ámbitos, lo que lleva a que, en principio, para asegurarse de que un nombre no está en un marcaje sería inevitable inspeccionar el marcaje completo. Aparentemente, esto puede interpretarse como una violación del principio de localidad (en el sentido clásico de redes de Petri), ya que el disparo de una transición depende no sólo del marcaje en sus lugares adyacentes, sino de todo el marcaje. Sin embargo, las relaciones de causalidad y conflicto [DR98] no se ven afectadas, y sólo el modo en el que se dispara la transición (más concretamente, el valor al que se instancia la variable ν) depende de todo el marcaje. Además, el valor al que se instancia esa variable se toma de un conjunto infinito e intuitivamente no importa qué elección se tome (como formalizaremos en el capítulo 6). En este sentido, dicha elección no supone la violación al principio de localidad que mencionábamos. En otras palabras, y siguiendo una metáfora sacada de la criptografía, se puede considerar que estamos en el análogo a la generación de claves criptográficas donde, sin compararlas con el resto de las claves que han sido antes creadas, podemos suponer que es de hecho única. La falta de un operador de declaración de ámbitos tiene la consecuencia de que el modelo no es composicional en el sentido clásico de las álgebras de procesos, o al menos no inmediatamente, ya que no se puede derivar el comportamiento de un sistema de manera directa en base al comportamiento de sus componentes. Esto es así simplemente porque un nombre podría ser creado por separado en dos redes de dos sistemas distintos, cosa que no podría ocurrir si las redes fuesen del mismo sistema.

Ejemplo 2 (Un protocolo de exclusión mutua) Este ejemplo modeliza un protocolo de exclusión mutua en un entorno distribuido. El sistema lo componen cuatro participantes: un servidor de tickets, un agente, el servidor crítico y los

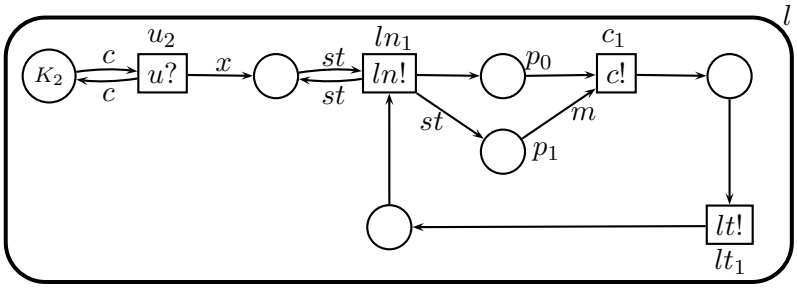

Figura 4.15: Servidor crítico

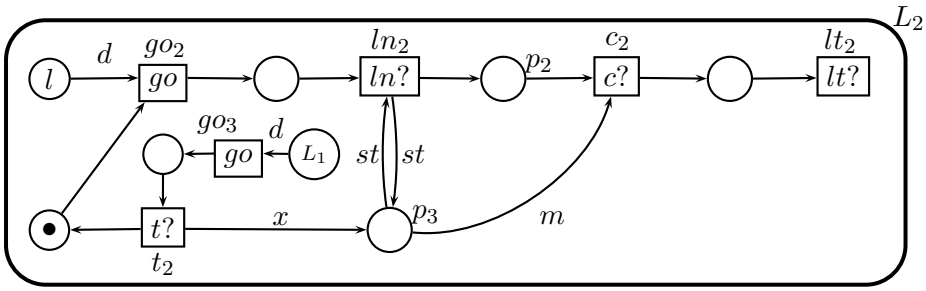

Figura 4.16: Clientes

clientes. El servidor de tickets, en la figura 4.14, genera los tickets que deben exhibir los clientes para acceder a la sección crítica del servidor crítico. Éste ofrece los tickets a los clientes y le dice al agente, ilustrado también en la figura 4.14, que actualice al servidor crítico (figura 4.15), de manera que el ticket comunicado sea uno de los válidos. El servidor de tickets y el agente se comunican usando la clave compartida K_1, mientras que el servidor crítico y el agente se comunican usando la clave K_2. Se garantiza que el uso de esas claves es seguro tomando a la variable c como variable de autenticación. Los clientes, descritos en la figura 4.16, después de recibir su ticket pueden ir a la localidad donde se encuentra el servidor crítico y hacer login. En ese momento, y después de mostrar de nuevo su ticket, pueden acceder a la sección crítica, representada por la transición `critical`. Solamente después de que el cliente presente haga logout podrá hacer login un nuevo cliente.

En efecto, comenzando en el marcaje representado en las figuras 4.14, 4.15 y 4.16, se puede disparar la secuencia de disparos

$$new(\eta)\ ,\ go_2(l_1)\ ,\ (t_1,t_2)(\eta)\ ,\ (f_1,f_2)(x:\eta,c:K_1)\ ,\ go_1(l)$$

$$go_2(l)\ ,\ (u_1,u_2)(x:\eta,c:K_2)\ ,\ (ln_1,ln_2)(\eta)\ ,\ (c_1,c_2)(\eta)\ ,\ (lt_1,lt_2)$$

Se entiende que el sistema es seguro si dos clientes no pueden acceder nunca a la sección crítica al mismo tiempo, es decir, si el siguiente marcaje no puede ser cubierto: $M(p_0) = M(p_2) = 1$, $M(p_1) = M(p_3) = \eta$ y $M(q) = \emptyset$ para el resto de

lugares. En el próximo capítulo demostraremos la decidibilidad del recubrimiento para sistemas MSPN, con lo que quedaría probado que propiedades como la anterior pueden ser comprobadas automáticamente, al menos en teoría.

Con la introducción del mecanismo de autenticación, y más concretamente con la capacidad de crear nuevos identificadores, hemos añadido en el modelo una nueva dimensión de infinitud. Ahora no sólo puede haber un número arbitrario de tokens en un lugar, sino puede existir un número arbitrario de tokens distintos en un marcaje alcanzable. Esto hace que la complejidad del modelo se incremente. En los sucesivos capítulos nos dedicaremos a formalizar esta complejidad añadida, estudiando la capacidad de cómputo de los modelos con creación de identificadores.

Capítulo 5

Modelos simplificados

Después de haber definido varias versiones de distintos modelos para sistemas ubicuos basados en redes de Petri, nos disponemos a comparar sus potencias expresivas mediante el estudio de la decidibilidad de varias de sus propiedades. Sin embargo, para llevar a cabo este estudio, es conveniente soltar lastre, es decir, librar a los distintos modelos de todas aquellas características que no influyan en sus capacidades expresivas, para poder realizar ese estudio en un marco simplificado, pero equivalente. En este capítulo probaremos que los modelos simplificados que introduciremos son equivalentes a los modelos sin simplificar en el sentido de la definición 2.6.9. En general, los modelos resultantes serán muy parecidos a los originales, así como las notaciones utilizadas, de modo que no presentaremos exhaustivamente cada una de estas notaciones, entendiéndose que se refieren a lo mismo que se referían las notaciones análogas de los modelos sin simplificar.

5.1. Redes de Petri con contadores

Las redes de Petri con contadores se corresponderán con los sistemas MSPNs con identificadores naturales. Esencialmente, son redes de Petri coloreadas con un solo color, el de $Id = \mathbb{N} \times \mathbb{N}$, capaces de realizar unas operaciones muy limitadas: incrementar el valor de una serie de contadores.

Definición 5.1.1 *Una* red de Petri con n contadores (n-CsPN) *es una tupla* $N = (P, T, F)$*, donde* P *es un conjunto de lugares,* T *es un conjunto disjunto del anterior de transiciones y* $F : (P \times T) \cup (T \times P) \to Var$ *es una función parcial, tales que:*

1. *Existen* $P_C \subseteq P$ *y* $T_C = \{t_c \mid c \in P_C\}$ *tales que* $|P_C| = |T_C| = n$.
2. *Para todo* $c \in P_C$, $F(c, t_c) = \tau^-$ *y* $F(t_c, c) = \tau^+$.
3. *Para todo* $t \notin T_C$, $\tau^-, \tau^+ \notin Vars(t)$.
4. *Para todo* $t \in T$, $post(t) \setminus \{\tau^+\} \subseteq pre(t)$.

Diremos que P_C es el conjunto de contadores de N y T_C el conjunto de transiciones sucesor de N.

Denotaremos por $NatId(n)$ al conjunto $\{(i,m) \mid i = 1,\ldots,n, \quad m \in \mathbb{N}\}$

Definición 5.1.2 *Sea $N = (P,T,F)$ una n-CsPN con conjunto de contadores $P_C = \{c_1,\ldots,c_n\}$. Un* marcaje *de N es una función $M : P \to \mathcal{MS}(NatId(n))$ tal que para todo $i \in \{1,\ldots,n\}, M(c_i) = \{(i,m_i)\}$ para algún $m_i \in \mathbb{N}$ y si $(i,k) \in S(M)$ entonces $k \leq m_i$.*

Definición 5.1.3 *Dada una n-CsPN N y una transición t de N, llamaremos* modo *de t a una función $\sigma : Var(t) \to NatId$ tal que si $\sigma(\tau^-) = (i,m)$ entonces $\sigma(\tau^+) = (i,m+1)$. Diremos que una transición t está activada en modo σ si para todo $p \in {}^\bullet t, \quad \sigma(F(p,t)) \in M(p)$. En ese caso t se puede disparar en modo σ, obteniéndose el marcaje M' definido por $M'(p) = M(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$.*

Es inmediato ver que en la definición anterior, tras el disparo de una transición en un marcaje se obtiene otro marcaje según esa misma definición.

Proposición 5.1.4 *Sea N una n-CsPN y M_0 un marcaje de N. Entonces, para todo M alcanzable desde M_0 se cumple que:*

- *Para cada $i \in \{1,\ldots,n\}$ existe $m_i \in \mathbb{N}$ tal que $M(c_i) = \{(i,m_i)\}$,*
- *Si $(i,k) \in S(M)$ entonces $k \leq m_i$.*

*Demostración.*Basta ver que si M es un marcaje y $M[t(\sigma)\rangle M'$ entonces M' es un marcaje.

- Si $t \notin T_C$ entonces $M(c) = M'(c)$ para todo $c \in P_C$. Supongamos que t es la transición sucesor del lugar contador $c_i \in P_C$. Por definición de modo se tiene que $\sigma(\tau^-) = (i,m)$ y $\sigma(\tau^+) = (i,m+1)$, donde $M(c_i) = \{(i,m)\}$. Entonces $M'(c) = M(c) - \{\sigma(F(c_i,t)\} + \{\sigma(F(t,c_i))\} = \{(i,m)\} - \{(i,m)\} + \{(i,m+1)\} = \{(i,m+1)\}$.
- Si $(i,k) \in S(M')$ e $(i,k) \in S(M)$ entonces $k \leq m_i$. Si no, se tiene que $k = m_i + 1$ y concluimos.

□

En las redes con contadores consideraremos el mismo orden que en los sistemas MSPN con naturales, olvidándonos de la componente localidad de los marcajes. Por lo tanto, dados dos marcajes M_1 y M_2 de una red con contadores, escribiremos $M_1 \sqsubseteq M_2$ si para todo lugar p se tiene que $M_1(p) \subseteq M_2(p)$.

Veamos que las redes con contadores y las MSPNs son equivalentes.

Proposición 5.1.5 *Todo sistema MSPN con n componentes puede ser simulado monótonamente por una n-CsPN.*

Demostración. Dado un sistema MSPN $\mathcal{N} = \{N_1, \ldots, N_n\}$ con marcaje inicial $\mathcal{M}_0$, tenemos que definir la red con n-contadores $\mathcal{N}^* = (P^*, T^*, F^*)$ que la simula. Lo hacemos mediante un desdoblamiento típico, de manera similar a la construcción de la proposición 3.2.3. La idea es que la restricción de que las sincronizaciones sólo puedan ocurrir dentro de la misma localidad se puede traducir a restricciones de igualdades entre identificadores (que en la demostración de la proposición 3.2.3 traducíamos a su vez a la presencia o no de tokens en lugares seguros). Si guardamos en un lugar $@_i$ el valor que representa la posición de la red N_i entonces la restricción anterior se consigue obligando a que el valor de los correspondientes lugares $@_i$ sea el mismo.

Sea $\mathcal{K} = \{k_1, \ldots, k_m\}$ el conjunto de tokens localidad que aparecen en $\mathcal{M}_0$. Usaremos el identificador $(1, 0)$ para representar al token ordinario y los tokens $(1, 1), \ldots, (1, m)$ para representar los tokens $k_1, \ldots, k_m$, respectivamente. Entonces tomamos:

- $P^* = P \cup \{@_1, \ldots, @_n, \bullet\}$,
- $T^* = \{t \mid t \in T, \lambda(t) \in \mathcal{A}\} \cup \{(t, t') \mid t \text{ y } t' \text{ compatibles}\}$,
- F^* se define de la siguiente manera:
  - Para todo $t \in T$, $F^*(p, t) = F(p, t)$ y $F^*(t, p) = F(t, p)$,
  - Para todo $(t, t') \in T^*$, $F^*(p, (t, t')) = \begin{cases} F(p, t) \text{ si } p \in {}^\bullet t \\ F(p, t') \text{ si } p \in {}^\bullet t', \end{cases}$
  - Para todo $(t, t') \in T^*$, $F^*((t, t'), p) = \begin{cases} F(t, p) \text{ si } p \in t^\bullet \\ F(t', p) \text{ si } p \in t'^\bullet, \end{cases}$
  - Si $t \in T_i$ y $t' \in T_j$ entonces $F^*((t, t'), @_i) = F^*(@_i, (t, t')) = F^*((t, t'), @_j) = F^*(@_j, (t, t')) = @$ para alguna variable $@ \in \mathit{Var}$ nueva,
  - Si $t \in T_i$ y $\lambda(t) = \mathit{go}$ entonces $F^*(@_i, t) = @$ y $F^*(t, @_i) = F(p, t)$, donde p es la precondición de tipo localidad de t,
  - $F^*(t, \bullet) = F(\bullet, t) = \varepsilon$ para todo t.

Además, dado un marcaje $\mathcal{M} = (M, \mathit{loc})$ de $\mathcal{N}$, definimos el marcaje $\mathcal{M}^*$ de $\mathcal{N}^*$ de la forma siguiente:

- $\mathcal{M}^*(\bullet) = \{(1, 0)\}$.
- $\mathcal{M}^*(@_i) = \{(1, j)\}$ si $\mathit{loc}(N_i) = k_j$ para todo $i \in \{1, \ldots, n\}$.
- $\mathcal{M}^*(p)(i, j) = M(p)(h(i, j))$, donde $h(i, j) = \begin{cases} \bullet \text{ si } (i, j) = (1, 0), \\ k_j \text{ si } i = 1 \text{ y } 0 < j \leq m, \\ (1, m - j + 1) \text{ si } i = 1 \text{ y } j > m, \\ (i, j) \text{ en otro caso.} \end{cases}$

En primer lugar, $\mathcal{N}^*$ es una red con a lo sumo n contadores. Además, todas las transiciones autónomas t cumplen la condición de que todas las variables en $post(t)$ (excepto τ^+) lo sean de $pre(t)$. Esto es así porque ocurría en $\mathcal{N}$ para todas las variables excepto ε y hemos añadido explícitamente la variable ε a $pre(t)$ para toda transición t de $\mathcal{N}^*$, que sólo podrá instanciarse a $\bullet$.

Si llamamos $(\ )^*$ a la inversa de h entonces se tiene que

$$\mathcal{M}_1[u(\sigma)\rangle\mathcal{M}_2 \Leftrightarrow \mathcal{M}_1^*[u(\sigma^*)\rangle\mathcal{M}_2$$

donde $\sigma^*(x) = \sigma(x)^*$. Veámoslo por distinción de casos en u.

Si $u = t \in T$ con $\lambda(t) \in \mathcal{A} \setminus \{go\}$ entonces los lugares $@_i$ no son precondiciones ni postcondiciones de t y se tiene que $\mathcal{M}_1^*(\sigma(F(p,t))^*) = M_1(p)(\sigma(F(p,t)))$ y $\sigma^*(F(p,t)) = \sigma(F(p,t))^*$. Entonces $\sigma^*(F(p,t)) \in \mathcal{M}_1^*(p) \Leftrightarrow \sigma(F(p,t)) \in M_1(p)$ y se sigue que $t(\sigma)$ está activada en $\mathcal{N}$ si y sólo si $t(\sigma^*)$ lo está en $\mathcal{N}^*$. Además se cumple que $\{\sigma^*(F(p,t))\}(i,j) = \{\sigma(F(p,t))\}(h(i,j))$ (ambos términos valen 1 si $\sigma(F(p,t)) = h(i,j)$ o 0 en otro caso) y $\{\sigma^*(F(t,p))\}(i,j) = \{\sigma(F(t,p))\}(h(i,j))$ (análogamente). De ello se deriva por definición de $\mathcal{M}_i^*$ que $M_2(p)(h(i,j)) = M_1(p)(h(i,j)) - \{\sigma(F(p,t))\}(h(i,j)) + \{\sigma(F(t,p))\}(h(i,j)) \Leftrightarrow \mathcal{M}_2^*(p)(i,j) = \mathcal{M}_1^*(p)(i,j) - \{\sigma^*(F^*(p,t))\}(i,j) + \{\sigma^*(F^*(t,p))\}(i,j)$, lo que implica la tesis.

El caso en el que $u = (t,t')$ es completamente análogo al anterior. Supongamos entonces que $\lambda(t) = go$ y q es la precondición de tipo localidad. Igual que en el caso anterior se ve que $\sigma(F(p,t)) \in M_1(p) \Leftrightarrow \sigma^*(F(p,t)) \in \mathcal{M}^*1(p)$ para todo $p \in P$ y $M_2(p)(h(i,j)) = M_1(p)(h(i,j)) - \{\sigma(F(p,t))\}(h(i,j)) + \{\sigma(F(t,p))\}(h(i,j)) \Leftrightarrow \mathcal{M}_2^*(p)(i,j) = \mathcal{M}_1^*(p)(i,j) - \{\sigma^*(F^*(p,t))\}(i,j) + \{\sigma^*(F^*(t,p))\}(i,j)$. Además, por definición de $\mathcal{M}^*$ y construcción de la red se tiene que $loc_2(N_i) = \sigma(F(q,t)) \Leftrightarrow \mathcal{M}_2^*(@_i) = \{\sigma^*(F^*(q,t))\}$. Como loc_2 y loc_1 coinciden en el resto de N_j si y sólo si $\mathcal{M}_1^*$ y $\mathcal{M}_2^*$ lo hacen en el resto de $@_j$ se sigue que $\mathcal{M}_1[u(\sigma)\rangle\mathcal{M}_2 \Leftrightarrow \mathcal{M}_1^*[u(\sigma^*)\rangle\mathcal{M}_2$.

Para concluir veamos que podemos aplicar la proposición 2.6.7. Lo anterior nos dice que $\mathcal{N}^*$ simula a $\mathcal{N}$ paso a paso. Por construcción todo estado alcanzable de $\mathcal{N}^*$ contiene al token $(1,0)$ en el lugar $\bullet$ (los arcos adyacentes son de lectura, es decir, comprueban que hay un token pero no lo borran) y a exactamente un token $(1,j)$ con $0 < j \leq m$ en cada uno de los lugares (toda transición que elimina un token de un lugar $@_i$ pone otro de esos tokens en el mismo $@_i$). Claramente, todo marcaje que cumple esas condiciones es estable, de modo que todo marcaje alcanzable lo es. Por último, para ver la monotonía de $(\ )^*$ y de su inversa basta tener en cuenta que para todo par de marcajes $\mathcal{M}_1 = (M_1, loc_1)$ y $\mathcal{M}_2 = (M_2, loc_2)$, si $p \notin P$ entonces $\mathcal{M}_1^*(p) = \mathcal{M}_2^*(p)$ y si $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$ entonces $loc_1 = loc_2$. Entonces $\mathcal{M}_1^* \sqsubseteq \mathcal{M}_2^* \Leftrightarrow \mathcal{M}_1^*(p) \subseteq \mathcal{M}_2^*(p)$ para todo $p \in P \Leftrightarrow \mathcal{M}_1^*(p)(i,j) \leq \mathcal{M}_2^*(p)(i,j) \Leftrightarrow M_1(p)(h(i,j)) \leq M_2(p)(h(i,j)) \Leftrightarrow \mathcal{M}_1 \sqsubseteq \mathcal{M}_2$.

□

Veamos el recíproco de la proposición anterior.

Proposición 5.1.6 *Toda CsPN puede ser simulada por un sistema MSPN.*

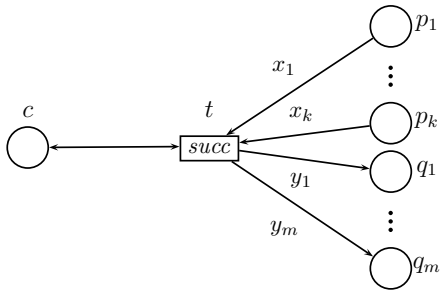

Figura 5.1: Forma general de transiciones sucesor en redes con contadores

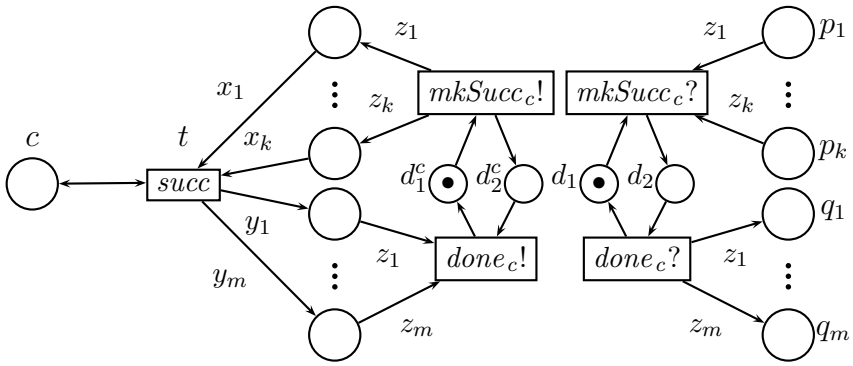

Figura 5.2: Simulación de CsPNs con MSPNs

Demostración. Dada una red N con n contadores tenemos que simularla con un sistema MSPN. Si $n = 1$ entonces es un sistema MSPN con una sola componente, sin lugares con tokens ordinarios ni localidades, ni transiciones de movimiento o de sincronización. Sin embargo, si la red tiene varios contadores, entonces su definición no se ajusta a la de los sistemas MSPN, ya que en éstos sólo permitimos una transición sucesor con su correspondiente contador por cada componente. Sin embargo, esta dificultad es fácilmente superable. Para ello, basta con separar la red en varias componentes, de manera que en cada una de ellas haya a lo sumo un lugar contador. Así que consideramos el siguiente sistema MSPN con $n + 1$ componentes. El grueso de N estará representado por una sola componente sin contadores, N^s, obtenida después de eliminar de N los contadores, junto con las transiciones sucesor, y sustituirlos por un par de transiciones para cada contador, etiquetadas por $mkSucc_c?$ y $done_c?$, controladas por dos nuevos lugares ordinarios d_1 y d_2, comunes a todos esos pares de transiciones. Cada contador c estará simulado por una componente N^c distinta, que sincroniza con la red principal en esas transiciones. La Figura 5.1 representa una transición sucesor cualquiera de la red original, mientras que en la Figura 5.2 a la izquierda se muestra la componente que representa a ese contador y a la derecha la parte de la red principal que interactúa

con el contador.

Formalmente, partimos de una red con n contadores $N = (P, T, F)$ y definimos el sistema MSPN $N^* = \{N^c \mid c \in P_C\} \cup \{N^s\}$ de la siguiente manera:

- Para cada $c \in P_C$, sea $N^c = (P^c, T^c, F^c, \lambda^c, C^c)$, donde
  - $P^c = \{c, d_1^c, d_2^c\} \cup \{p_c \mid p \in {}^\bullet t_c \cup t_c^\bullet\}$
  - $T^c = \{t_c, make_c, done_c\}$
  - $C^c(p) = \begin{cases} \bullet & \text{si } p \in \{d_1^c, d_2^c\} \\ Id & \text{en otro caso} \end{cases}$
  - $\lambda^c(t_c) = succ$, $\lambda^c(make_c) = mkSucc_c?$ y $\lambda^c(done_c) = done_c?$
  - F^c es la función parcial con menor dominio tal que
    - $F^c(c, t_c) = \tau^-$ y $F^c(t_c, c) = \tau^+$,
    - $F^c(p_c, t_c) = F(p, t_c)$ y $F^c(make_c, p_c) = z_p$ para todo $p \in {}^\bullet t_c \setminus \{c\}$,
    - $F^c(t_c, p_c) = F(t_c, p)$ y $F^c(p_c, done_c) = z'_p$ para todo $p \in t_c^\bullet \setminus \{c\}$,
    - $F^c(d_1^c, make_c) = F^c(make_c, d_2^c) = \varepsilon$,
    - $F^c(d_2^c, done_c) = F^c(done_c, d_1^c) = \varepsilon$.
- Sea $N^s = (P^s, T^s, F^s, \lambda^s, C^s)$ definida de la siguiente forma:
  - $P^s = P \setminus P_C \cup \{d^1, d^2 \mid c \in P_C\}$
  - $T^s = T \setminus T_C \cup \{done^c, make^c \mid c \in P_C\}$,
  - $C^s(p) = \begin{cases} \bullet & \text{si } p \in \{d^1, d^2\} \\ Id & \text{en otro caso,} \end{cases}$
  - $\lambda^s(t) = \lambda(t)$ para todo $t \in T \setminus T_C$, $\lambda^s(make^c) = mkSucc_c!$ y $\lambda^s(done^c) = done_c!$,
  - F^s es la función parcial con menor dominio tal que
    - $F^s(p, t) = F(p, t)$ y $F^s(t, p) = F(t, p)$ si $p \notin P_C$,
    - $F^s(p, make^c) = z_p$ si $c \neq p \in {}^\bullet t_c$,
    - $F^s(done^c, p) = z'_p$ si $c \neq p \in t_c^\bullet$,
    - $F^s(d^1, make^c) = F^s(make^c, d^2) = \varepsilon$,
    - $F^s(d^2, done^c) = F^s(done^c, d^1) = \varepsilon$.

Para acabar la construcción, dado un marcaje M de N, definimos M^* como el marcaje de N^* que coincide con M en los lugares comunes y tal que $M^*(p_c) = \emptyset$, $M^*(d_1^c) = M^*(d^1) = 1$ y $M^*(d_2^c) = M^*(d^2) = 0$ para todo $c \in P_C$.

En primer lugar, veamos que si $M_1[u(\sigma)\rangle M_2$ entonces $M_1^* \to^* M_2^*$. Si $u \notin T_C$ entonces u es una transición de N^* con las mismas precondiciones y postcondiciones que en N, de modo que $M_1^*[u(\sigma)\rangle M_2^*$ es un disparo en N^*. Veamos que en el caso en el que $u = t_c \in T_C$ existen modos σ_1 y σ_2 y marcajes M y M' de N^* tales que

$$M_1^*[(make_c, make^c)(\sigma_1)\rangle M[t_c(\sigma)\rangle M'[(done_c, done^c)(\sigma_2)\rangle M_2^*$$

Por la construcción anterior, ${}^\bullet make_c = \{d_1^c\}$ y ${}^\bullet make^c = \{d^1\} \cup \{p \mid p \in {}^\bullet t_c$ en $N\}$. Como $M_1[t_c(\sigma)\rangle M_2$ se cumple que $\sigma(F(p,t_c)) \in M_1(p)$, de modo que si definimos $\sigma_1(z_p) = \sigma(F(p,t_c))$ tenemos que $\sigma_1(F^s(p, make^c)) = \sigma(F(p,t_c)) \in M_1(p) = M_1^*(p)$ para $p \neq d_1^c$. Además, por la definición de M_1^*, los lugares d^1 y d_1^c están marcados, y el par de transiciones de sincronización $(make_c, make^c)$ está activado. Como $F^s(p, make^c) = F^c(make_c, p_c)$, $make^{c\bullet} = \{d^2\}$ y $make_c^\bullet = \{d_2^c\} \cup \{p_c \mid p \in {}^\bullet t_c$ en $N\}$, tras el disparo del par se obtiene el marcaje M tal que $M(p_c) = a_p = \sigma(F(p,t_c))$, $M(d_1^c) = M(d^1) = 0$ y $M(d_2^c) = M(d^2) = 1$.

Veamos que desde M se puede disparar t_c con modo σ. En primer lugar, ${}^\bullet t_c = \{c\} \cup \{p_c \mid p \in {}^\bullet t_c$ en $N\}$ y $t_c^\bullet = \{c\} \cup \{p_c \mid p \in t_c^\bullet$ en $N\}$. Además, como $M(p_c) = a_p = \sigma(F(p,t_c))$, $F^c(p_c,t_c) = F(p,t_c)$ y $F^c(t_c,p_c) = F(t_c,p)$ se cumple que se puede disparar t_c en modo σ, obteniendo M', que verifica $M'(p_c) = \sigma(F(t_c,p)) = M_2(p)$, $M'(c) = M_2(c)$ y en el resto de lugares coincide con M. Por ello, tomando $\sigma_2(z'_p) = \sigma(F(t_c,p_c))$, y teniendo en cuenta que ${}^\bullet done_c = \{d_2^c\} \cup \{p_c \mid p \in t_c^\bullet\}$ y ${}^\bullet done^c = \{d^2\}$ podemos disparar el par de sincronización $(done_c, done^c)$, obteniendo un marcaje M'' que, por ser $F^s(done^c, p) = F^c(p_c, done_c)$, es tal que $M''(p) = \sigma_2(F^s(done^c, p)) = \sigma(F^c(p_c, done_c)) = M'(p_c) = M_2(p)$, es decir, coincide con M_2 en los lugares adyacentes a t_c. Además, $M''(p_c) = \emptyset$ para todo $p_c \in P^C$, $M''(d^1) = M''(d_1^c) = 1$ y $M''(d^2) = M(d_2^c) = 0$, con lo que se concluye que $M'' = M_2^*$ y queda demostrado que $M_1^* \rightarrow^* M_2^*$.

Recíprocamente, si $M_1^*[t(\sigma)\rangle M'$ con $t \in T$ entonces claramente $M_1[t(\sigma)\rangle M_2$ y $M_2^* = M'$. Si por el contrario $t \notin T$ entonces por construcción ha de ser $t = (make_c, make^d)$ y no se puede desactivar el disparo de t_c primero y $(done_c, done^c)$ después, de modo que si $M_1^*[(make_c, make^c)(\sigma_1)\rangle M'$ entonces

$$M'[t_c(\sigma)\rangle M''[(done_c, done^c)(\sigma_2)\rangle M_2^*$$

y $M_1[t_c(\sigma)\rangle M_2$.

□

Proposición 5.1.7 *Los sistemas MSPN simulan monótonamente a las redes con contadores.*

Demostración. Veamos primero que $(\ )^*$ y su inversa son monótonas. Efectivamente, teniendo en cuenta que si $p \notin P$ entonces $M_1^*(p) = M_2^*(p)$ para cualquier par de marcajes M_1 y M_2, se tiene que $M_1^* \sqsubseteq M_2^* \Leftrightarrow M_1^*(q) \subseteq M_2^*(q)$ para todo $q \Leftrightarrow M_1(p) \subseteq M_2(p)$ para todo $p \in P \Leftrightarrow M_1 \sqsubseteq M_2$.

Además, existe d^1 seguro tal que $\overline{M}$ es estable si y sólo si $\overline{M}(d^1) = 1$, por lo que todo inestable $\overline{M}'$ alcanzable cumple que $\overline{M}'(d^1) = 0$ y no puede ser $\overline{M} \sqsubseteq \overline{M}'$.

□

Como corolario de la proposición anterior y la proposición 5.1.5 tenemos el siguiente resultado.

Corolario 5.1.8 *La alcanzabilidad y el recubrimiento son decidibles para los sistemas MSPN si y sólo si lo son para las redes con contadores.*

Hemos visto que las redes con n contadores son equivalentes a los sistemas MSPN con n componentes con transición sucesor. En particular, las redes sin contadores son equivalentes a los sistemas sin transiciones sucesor, que como vimos en el capítulo anterior, son equivalentes a las redes P/T y, por tanto, la alcanzabilidad y el recubrimiento son decidibles para las redes sin contadores.

5.2. ν-APNs

Para estudiar la decidibilidad de propiedades de los sistemas MSPN con creación de nombres abstractos, así como la traducción a teorías de reescritura que veremos en el capítulo 10, definimos las redes de Petri ν-abstractas (o ν-APNs, utilizando las siglas de *ν-abstract Petri nets* en inglés). Esencialmente, son una subclase de las redes de Petri coloreadas, excepto por el hecho de que son capaces de generar nombres frescos por definición, como ocurre en los cálculos basados en nombres, como el cálculo π [MPW92]. Técnicamente, estas redes son análogas a las redes OO-minimales de [Kum00], aunque allí no se crean nombres para la seguridad, sino objetos.

En esta sección supondremos que *Id* es un conjunto infinito arbitrario de identificadores.

Definición 5.2.1 *Una* ν-APN *es una tupla* $N = (P, T, F)$ *donde* P *es un conjunto de lugares,* T *es un conjunto de transiciones disjunto del anterior, y* $F : (P \times T) \cup (T \times P) \rightarrow Var$ *es una función parcial tal que para cada* $t \in T$, $post(t) \setminus \{\nu\} \subseteq pre(t)$ *y* $\nu \notin pre(t)$.

Definición 5.2.2 *Sea* t *una transición de la* ν*-APN* N. *Diremos que* $\sigma : Var(t) \rightarrow Id$ *es un* modo *de* t *y que* t *está activada en modo* σ *en el marcaje* M, *si se cumplen las siguientes condiciones:*

- *Si* $\nu \in Var(t)$ *entonces* $\sigma(\nu) \notin S(M)$.
- $\sigma(F(p,t)) \in M(p)$ *para todo* $p \in {}^{\bullet}t$.

En ese caso diremos que se puede disparar la transición, obteniéndose el marcaje M', *definido por* $M'(p) = M(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$.

Una ν-APN es un caso particular de sistema MSPN con identificadores abstractos, con una sola componente, y sin transiciones de movimiento o de sincronización. Con una construcción prácticamente idéntica a la de la proposición 5.1.5 del apartado anterior podemos probar el siguiente resultado.

Proposición 5.2.3 *Todo sistema MSPN con identificadores abstractos puede ser simulado monótonamente por una* ν*-APN.*

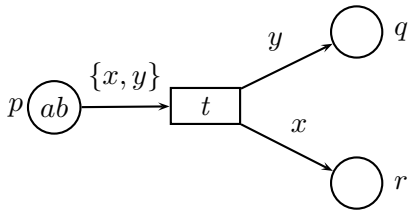

Figura 5.3: Red con conjuntos en arcos

Demostración. La definición de $\mathcal{N}^* = (P^*, T^*, F^*)$ es igual a la de la proposición 5.1.5, aunque ahora en P no hay lugares contador. En cuanto a los marcajes, antes usábamos el identificador $(1, 0)$ para representar al token $\bullet$ y $(1, i)$ para representar al token localidad $k_i \in \mathcal{K}$. En este caso basta con tomar un identificador arbitrario, que por comodidad podemos denotar $\bullet \in Id$, para representar al token negro y un token identificador por cada $k \in \mathcal{K}$, que podemos igualmente denotar $k \in Id$. Manteniendo esta elección en la definición del marcaje M^* de $\mathcal{N}^*$ a partir del marcaje M de $\mathcal{N}$ se cumple que $M_1[u(\sigma)\rangle M_2 \Leftrightarrow M_1^*[u(\sigma)\rangle M_2^*$, análogamente a la proposición 5.1.5, así como el resto de condiciones.

□

Tanto en el anterior resultado como en la proposición 5.1.5 estamos utilizando identificadores para simular al token $\bullet$. En efecto, la única diferencia de este token y los tokens identificador en los sistemas MSPN es que pueden ser creados por las transiciones. En estos dos resultados hemos supuesto que tenemos un lugar marcado con un token que representa a $\bullet$, usado para tomarlo cuando hace falta. Por tanto, cuando lo necesitemos podemos usar este token en modelos que en principio sólo tienen tokens identificador.

Si la ν-APN no tiene ningún arco etiquetado con la variable ν entonces diremos que es una APN. En este caso, una APN es un caso particular de un sistema MSPN sin identificadores que, como vimos en el capítulo anterior, se reduce a los sistemas ubicuos y, por tanto, a las redes P/T.

Veamos ahora un resultado técnico que nos permitirá simplificar algunas demostraciones. Se trata de probar que si extendemos las ν-APNs para permitir que los arcos estén etiquetados con conjuntos de variables entonces no se supera el poder expresiva de las ν-APNs. En este supuesto, dado un marcaje M, una transición t estará activada en un modo σ cuando $\sigma(F(p, t)) \subseteq M(p)$ para todo $p \in {}^\bullet t$, y su disparo producirá un marcaje $M'(p) = M(p) - \sigma(F(p, t)) + \sigma(F(t, p))$. Para cada t, si ${}^\bullet t = \{p_1, \ldots, p_k\}$ y $t^\bullet = \{q_1, \ldots, q_l\}$ vamos a denotar $\overline{pre}(t) = F(p_1, t) + \ldots + F(p_n, t)$ y $\overline{post}(t) = F(t, q_1) + \ldots + F(t, q_l)$, es decir, los multiconjuntos de variables en arcos precondición y postcondición respectivamente, que pueden repetirse de aparecer en distintos arcos.

Consideramos un orden total cualquiera (pero fijo) definido sobre el conjunto de todas las variables, de manera que escribiremos $(x_1, \ldots, x_n)$ para referirnos al

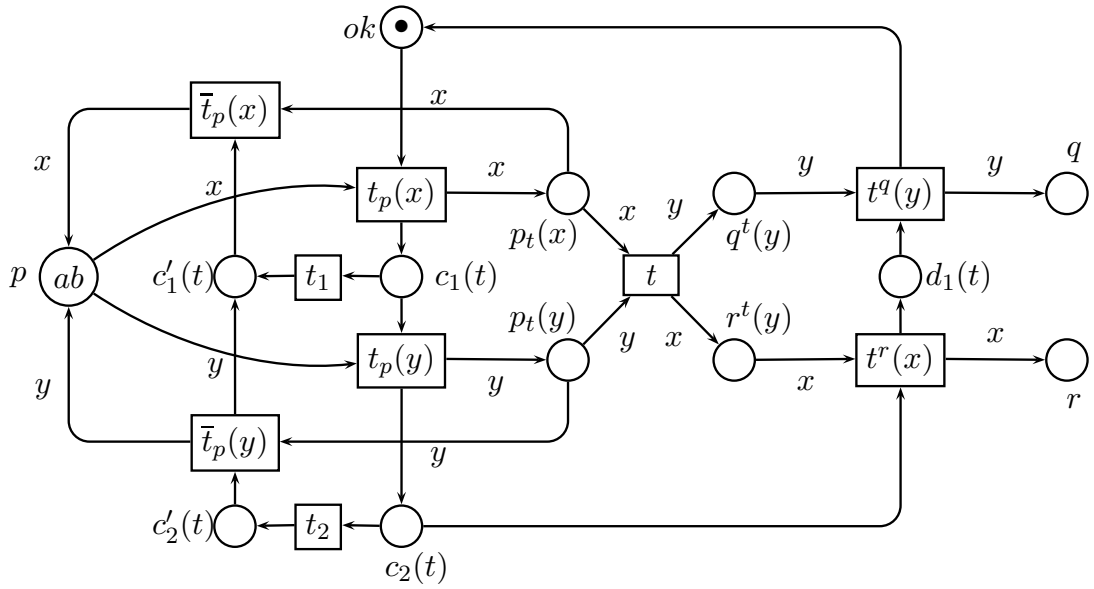

Figura 5.4: Simulación de la red en la figura 5.3 sin conjuntos en los arcos

conjunto $\{x_1, \ldots, x_n\}$ cuando $x_i < x_{i+1}$ para $i = 1, \ldots, n-1$.

Proposición 5.2.4 *Toda ν-APN con arcos etiquetados con conjuntos de variables se puede simular monótonamente mediante una ν-APN.*

Demostración. Simulamos el efecto del disparo de una transición t que puede tomar varios tokens del mismo lugar y añadir varios tokens al mismo lugar por medio de la secuencia de una serie de disparos que toman esos tokens de uno en uno y luego los añaden también de uno en uno. En la figura 5.4 se muestra la simulación de la red de la figura 5.3. En ella el disparo de t se simula mediante el disparo de la secuencia $t_p(x), t_p(y), t, t^r(x), t^q(y)$. La única complicación que tiene la simulación es que, al tomar los tokens de las precondiciones uno a uno, puede ser que después de haber tomado algunos no se puedan tomar más, con lo cual sería necesario deshacer los pasos ya dados. Para ello usamos las transiciones de la forma $\overline{t}_p(x)$.

Dada la ν-APN etiquetada con conjuntos $N = (P, T, F)$ tomamos $N^* = (P^*, T^*, F^*)$, donde

- $P^* = P \cup \{ok\} \cup \{p_t(x) \mid x \in F(p,t)\} \cup \{c_i(t), c_i'(t) \mid 1 \leq i \leq |\overline{pre}(t)|\} \cup$
  $\{p^t(x) \mid x \in F(t,p)\} \cup \{d_i(t) \mid 1 \leq i \leq |\overline{post}(t)| - 1\}$,
- $T^* = T \cup \{t_p(x), \overline{t}_p(x) \mid F(p,t) = x\} \cup$
  $\{t^p(x) \mid t \in T,\ F(t,p) = x\} \cup \{t_i \mid 1 \leq i \leq |\overline{pre}(t)|\}$,
- $F^*(p, t_p(x)) = F^*(t_p(x), p_t(x)) = x$ y $F^*(t^p(x), p) = F^*(p^t(x), t^p(x)) = x$,

- $F^*(p_t(x), t) = x$ y $F^*(t, p^t(x)) = x$,
- Para cada $t \in T$, si $\overline{pre}(t) = (x_1, \ldots, x_n)$ y y $\overline{post}(t) = (y_1, \ldots, y_m)$ entonces
  - $F^*(ok, t_p(x_1)) = \varepsilon$, $F^*(c_n(t), t^p(y_1)) = \varepsilon$, $F^*(t^p(y_m), ok) = \varepsilon$,
  - $F^*(c_i(t), t_p(x_{i+1})) = \varepsilon$ para $i = 1, \ldots, n-1$,
  - $F^*(t_p(x_i), c_i(t)) = \varepsilon$ para $i = 1, \ldots, n$,
  - $F^*(d_i(t), t^p(y_{i+1})) = \varepsilon$ para $i = 1, \ldots, m-1$,
  - $F^*(t^p(y_i), d_i(t)) = \varepsilon$ para $i = 1, \ldots, m-1$,
  - $F^*(c_i(t), t_i) = F^*(t_i, c'_i(t)) = \varepsilon$,
  - $F^*(c'_i(t), \overline{t}_p(x_i)) = \varepsilon$ para $i = 1, \ldots, n$,
  - $F^*(\overline{t}_p(x_i), c'_{i-1}(t)) = \varepsilon$ para $i = 2, \ldots, n$,
  - $F^*(\overline{t}_p(x_1), ok) = \varepsilon$,
  - $F^*(p_t(x_i), \overline{t}_p(x_i)) = F^*(\overline{t}_p(x_i), p) = x_i$ para $i = 1, \ldots, n$.

Además, para cada marcaje M de $\mathcal{N}$ definimos M^* que coincide con M en P, $M^*(ok) = \{\bullet\}$ y $M^*(q) = \emptyset$ en otro caso.

En primer lugar veamos que $M_1[t(\sigma)\rangle M_2$ implica que $M_1^* \to^* M_2^*$. Si $\overline{pre}(t) = (x_1, \ldots, x_n)$ y $\overline{post}(t) = (y_1, \ldots, y_m)$ tenemos que $\sigma(x_i) \in M(p)$ para todo i y p con $x_i \in F(p_i, t)$. Además, $M_1^*(ok) = 1$, por lo que se puede disparar la secuencia $t_{p_{i_1}}(x_1)(\sigma_1), \ldots, t_{p_{i_n}}(x_n)(\sigma_n)$ con $\sigma_i(x_i) = \sigma(x_i)$, obteniendo un marcaje $\overline{M}$ tal que $\overline{M}(p) = M_1(p) - \sigma(F(p,t))$, $\overline{M}(p_t(x_i)) = \{\sigma(x_i)\}$ y $\overline{M}(c_n(t)) = 1$. Entonces se puede disparar $t(\sigma)$ para obtener $\overline{M}_2$ que coincide en todos los lugares con $\overline{M}_1$ excepto por $\overline{M}_2(p_t(x_i)) = \emptyset$ para todo i y $\overline{M}_2(p^t(y_i)) = \sigma(y_i)$ para todo i. En esas condiciones se puede disparar la secuencia $t^{p_1}(y_1)(\sigma'_1), \ldots, t^{p_m}(y_m)(\sigma'_m)$ con $\sigma'_i(y_i) = \sigma(y_i)$. Tras ello se obtiene un marcaje $\overline{M}'$ tal que para todo $p \in P$, $\overline{M}'(p) = \overline{M}_2(p) + \sigma(post(t)) = \overline{M}(p) + \sigma(post(t)) = M_1(p) - \sigma(pre(t)) + \sigma(post(t)) = M_2(p)$. Además, $\sigma(ok) = 1$, $\sigma(c_i(t)) = \sigma(c'_i(t)) = 0$ para todo i y t, por lo que se puede concluir que $\overline{M}' = M_2^*$.

Veamos el recíproco. Supongamos que en $M_1^* \to^* \overline{M}$ no hay estados estables salvo M^*1 y que no hay ningún disparo de $t \in T$. Desde marcajes estables sólo pueden estar activadas las transiciones del tipo $t_p(x_1)$ con $\overline{pre}(t) = (x_1, \ldots, x_n)$. Entonces existe un único lugar marcado de entre los tipos $c_l(t)$ y $c'_l(t)$. Si estamos en el primer caso entonces se puede disparar t_l, y estamos en el segundo caso. En el segundo caso podríamos disparar la secuencia de transiciones $\overline{t}_{p_{i_{l-1}}}(x_{l-1}), \ldots, \overline{t}_{p_{i_1}}(x_1)$ que vuelve al marcaje M_1^* y trivialmente $M_1 \to^* M_1$.

Si hay exactamente un disparo de $t \in T$ para alguna transición $t \in T$ entonces se ha tenido que disparar la secuencia $t_{p_1}(x_1)(\sigma_1), \ldots, t_{p_m}(x_m)(\sigma_m)$. Como todos los alcanzados son inestables existe un único i tal que $\overline{M}(d_i(t)) = 1$. Entonces se pueden disparar $t^{q_{i+1}}(y_{i+1}), \ldots, t^{q_m}(y_m)$ para obtener un marcaje estable M_2^*. Si tomamos $\sigma(x_i) = \sigma_i(x_i)$ veamos que entonces $M_1[t(\sigma)\rangle M_2$. Como $\sigma_i(x_i) \in M_1^*(p)$ para todo i y $pre(t) = (x_1, \ldots, x_n)$ se tiene que $\sigma(F(p,t)) \subseteq M_1^*(p)$ y

la transición está activada. Además, $M_2(p) = M_2^*(p) = M_1^*(p) - \sigma(F(p,t)) + \sigma(F(t,p)) = M_1(p) - \sigma(F(p,t)) + \sigma(F(t,p))$ siguiendo el razonamiento inverso al de la implicación previa.

□

Para concluir este capítulo, entre las redes con contadores y las ν-APNs podemos probar la siguiente relación.

Proposición 5.2.5 *Las ν-APNs simulan débilmente a las redes con contadores.*

Demostración. Vamos a usar la técnica de detección estática de basura comentada en la sección 2.6. Representamos cada par de naturales (i,j) con un identificador abstracto distinto η_i^j. Para reconocer el identificador que representa a cada par, usaremos un lugar d_i que contendrá en toda simulación fidedigna, j veces el identificador η_i^j, para todo (i,j) menor o igual que el contenido del contador c_i. Además usaremos un lugar $current_{t_c}$ para cada contador c, que almacena k veces al identificador que representa a (i,k), si este es el contenido de c, lo que nos permitirá "incrementar" el valor del contador.

Vamos a ilustrar la simulación del incremento del contador mediante el ejemplo de la Figura 5.5, cuya simulación aparece en la Figura 5.6. El incremento del valor del contador se simula mediante la creación de un nuevo identificador en el lugar new_{t_c} primero, la sustitución de todos los tokens del lugar $current_c$ por el nuevo token creado, a la vez que se añade ese token a d_c tantas veces como tokens iguales al de c había en $current_c$. Por último se añade una vez más el nuevo token creado a d_c y a $current_c$. Es decir, el nuevo token aparece en d_c una vez más que el que había en c. En este proceso podría ocurrir que no todos los tokens de $current_c$ se sustituyesen por el nuevo token, lo que daría lugar a una simulación incorrecta. Sin embargo, en este caso, los tokens basura ya no podrían ser eliminados nunca, de modo que una simulación incorrecta no puede regresar al buen camino, seguirá siempre siendo incorrecta. Veámoslo con precisión.

Si $N = (P,T,F)$ es una red con n contadores, $P_C = \{c_1, \ldots, c_n\}$ es el conjunto de contadores, t_c es la transición sucesor para cada $c \in P_C$ y $T_C = \{t_c \mid c \in P_C\}$, entonces definimos la ν-APN $N^* = (P^*, T^*, F^*)$, donde

- $P^* = P \cup \{new_c, current_c, d_c \mid c \in P_C\} \cup \{control\}$,
- $T^* = T \cup \{t(move), t(done) \mid t \in T_C\}$,
- F^* es la función parcial con menor dominio tal que:
  - $F^*(p,t) = F(p,t)$
  - $F^*(t,p) = \begin{cases} F(t,p) \text{ si } F(t,p) \neq \tau^+ \\ \nu \text{ en otro caso} \end{cases}$ para todo $p \in P \setminus P_C$ y $t \in T$,
  - $F^*(control, t) = \varepsilon$ para todo $t \in T$,

- $F^*(t, control) = \varepsilon$ para todo $t \in T \setminus T_C$,
- $F^*(t, new_t) = \nu$ para todo $t \in T_C$,
- $F^*(current_c, t_c(move)) = F^*(c, t_c(move)) = F^*(t_c(move), c) = x$ para todo $c \in P_C$,
- $F^*(new_{t_c}, t_c(move)) = F^*(tt_c(move), new_{t_c}) = F^*(t_c(move), d_c) = y$ y $F^*(t_c(move), current_c) = y$ para todo $c \in P_C$,
- $F^*(c, t_c(done) = x$, $F^*(t_c(done), c) = F^*(t_c(done), current_c) = y$ y $F^*(t_c(done), d_c) = F^*(new_{t_c}, t_c(done)) = y$ para todo $c \in P_C$,
- $F^*(t_c(done), control) = \varepsilon$ para todo $c \in P_C$.

Además, para cada marcaje M de N definimos el marcaje M^* de N^* de la siguiente manera. Si para cada $i = 1, \ldots, n$, $M(c_i) = \{(i, m_i)\}$, sea $\{\eta_i^j \mid i = 1, \ldots, n\ j = 1, \ldots, m_i\}$ una familia cualquiera de identificadores distintos dos a dos. Entonces tomamos M^* definido de la siguiente manera:

- $M^*(p)(\eta_i^j) = M(p)(i, j)$ para todo $p \in P$,
- $M^*(new_t) = \emptyset$ para todo $t \in T_C$,
- $M^*(current_{t_{c_i}}) = \{\eta_i^{m_i}, \overset{m_i}{\ldots}, \eta_i^{m_i}\}$ para $i = 1, \ldots, n$,
- $M^*(d_{c_i})(\eta_i^j) = j$ para todo $i = 1, \ldots, n$ y $j = 1, \ldots, m_i$,
- $M^*(control) = 1$.

Veamos que N^* simula débilmente a N. En primer lugar veamos que si $M_1[t(\sigma)\rangle M_2$ entonces $M_1^* \to^* M_2^*$. Si $t \notin T_C$ entonces claramente el conjunto de precondiciones/postcondiciones de t en N coincide con el correspondiente en N^* y $M_1^*[t(\sigma')\rangle M_2^*$ con $\sigma'(x) = \eta_i^j$ si $\sigma(x) = (i, j)$. Supongamos ahora que $t \in T_C$ con $t = t_c$, $\sigma(\tau^-) = (l, m)$. Entonces el siguiente es un cómputo en N^*:

$$M_1^*[t(\sigma')\rangle \overline{M}_0[t(move)(\bar{\sigma})\rangle \overline{M}_1 \ldots \overline{M}_{m_l-1}[t(move)(\bar{\sigma})\rangle \overline{M}_{m_l}[t(done)(\sigma'')M_2^*$$

donde $\sigma'(x) = \begin{cases} \sigma(x) \text{ si } x \neq \nu \\ \eta \text{ nuevo si } x = \nu \end{cases}$, $\sigma''(x) = \eta$ y $\sigma''(y) = \eta'$ denotando $\eta' = \eta_i^{m_l}$ y

- $\overline{M}_i(p) = M_1^*(p) - \{\sigma(F(p, t))\} + \{\sigma(F(t, p))\}$ para todo $p \in P \setminus \{c\}$,
- $\overline{M}_i(control) = 0$,
- $\overline{M}_i(new_t) = \{\eta\}$,
- $\overline{M}_i(d_c) = M_1^*(d_c) + \{\eta, \overset{i}{\ldots}, \eta\}$,
- $\overline{M}_i(current_t) = \{\eta, \overset{i}{\ldots}, \eta, \eta', \overset{m_l-i}{\ldots}, \eta'\}$,

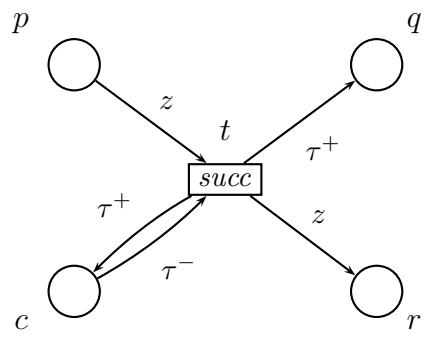

Figura 5.5: Simple red con contador

- $\overline{M}_i(c) = M_1^*(c)$.

En efecto, $t(\sigma')$ se puede disparar en M_1^* porque $t(\sigma)$ se puede disparar en M_1 y $M_1^*(control) = 1$. Su disparo quita el token de *control*, añade $\sigma'(\nu) = \eta$ a new_t y modifica los lugares en $P \setminus \{c\}$ como indican las variables, que coinciden con las de N, obteniendo así $\overline{M}_0$. Veamos ahora que $\overline{M}_i[t(move)(\bar{\sigma})\rangle\overline{M}_{i+1}$ para $i = 0, \ldots, m_l - 1$. Por construcción coinciden trivialmente en todos los lugares excepto c, d_c, $current_t$ y new_t. Como $\overline{M}_i(current_t) = \{\eta, \overset{i}{\ldots}, \eta, \eta', \overset{m_l-i}{\ldots}, \eta'\}$, para $i < m_l$ se tiene que $\eta' \in \overline{M}_i(current_t)$. Además, $\{\eta'\} = \{\eta_i^{m_l}\} = \overline{M}_i(c)$ y $\{\eta\} = \overline{M}_i(new_t)$ y por tanto la transición se puede disparar. Tras el disparo el contenido de c y new_t no ha variado, por lo que coincide con el contenido de esos lugares según $\overline{M}_{i+1}$, de manera que sólo queda verlo para d_c y $current_t$.

Pues bien, $\overline{M}_i(current_t) - \{\bar{\sigma}(F(current_t, t(move)))\} + \{\bar{\sigma}(F(t(move), current_t))\} = \{\eta, \overset{i}{\ldots}, \eta, \eta', \overset{m_l-i}{\ldots}, \eta'\} - \{\eta'\} + \{\eta\} = \{\eta, \overset{i+1}{\ldots}, \eta, \eta', \overset{m_l-i-1}{\ldots}, \eta'\} = \overline{M}_{i+1}(current_t)$. Para d_c se tiene que $\overline{M}_i(d_c) + \{\bar{\sigma}(F(t(move), d_c))\} = M_1^*(d_c) + \{\eta, \overset{i}{\ldots}, \eta\} + \{\eta\} = \overline{M}_{i+1}(d_c)$. Sólo falta ver que si $\overline{M}_{m_i}[t(done)\sigma''\rangle\overline{M}$ entonces $\overline{M} = M_2^*$. Denotemos $\eta = \eta_i^{m_l+1}$, $m_i' = m_i$ para $i \neq l$ y $m_l' = m_l + 1$, y tomemos la familia $\{\eta_i^j \mid i = 1, \ldots, n,\ j = 1, \ldots, m_i'\}$. Para $p \in P \setminus \{c\}$, $\overline{M}(p)(\eta_i^j) = \overline{M}_n(p)(\eta_i^j) = M_1^*(p)(\eta_i^j) - \{\sigma'(F(p,t))\}(\eta_i^j) + \{\sigma'(F(t,p))\}(\eta_i^j) = M_1(p)(i,j) - \{\sigma(F(p,t))\}(i,j) + \{\sigma(F(t,p))\}(i,j) = M_2(p)(i,j) = M_2^*(p)(\eta_i^j)$. Además, $\overline{M}(new_t) = \emptyset = M_2^*(new_t)$, $\overline{M}(current_t) = \overline{M}_{m_l}(current_t) + \{\sigma''(y)\} = \{\eta, \overset{m_l+1}{\ldots}, \eta\} = M_2^*(current_t)$. Además, $\overline{M}(c) = \{\eta_l^{m_l}\} = M_2^*(c)$ y $\overline{M}(d_c) = M_1^*(d_c) + \{\eta, \overset{m_l+1}{\ldots}, \eta\}$, por lo que $\overline{M}(d_c)(\eta_l^{m_l+1}) = m_l + 1 = M_2^*(\eta_l^{m_l+1})$.

Para el recíproco diremos que un marcaje $\overline{M}$ de N^* es legal si $S(\overline{M}(current_{t_c})) \subseteq \overline{M}(c) \cup \overline{M}(new_{t_c})$ para todo $c \in P_C$. Claramente todos los marcajes estables son legales y se cumple que si $\overline{M}$ es ilegal y $\overline{M}[t(\sigma)\rangle\overline{M}'$ entonces $\overline{M}'$ es ilegal. Efectivamente, supongamos que existe $\eta \in S(\overline{M}(current_{t_c}))$ tal que $\eta \notin \overline{M}(c) \cup \overline{M}(new_{t_c})$ para algún c. Como las únicas transiciones que afectan a esos lugares son t_c, $t_c(move)$ y $t_c(done)$ supongamos que t es una de esas tres. Si $t = t_c$ entonces claramente η sigue verificando las mismas condiciones, pues $\overline{M}'(new_t) = \{\eta'\}$ con

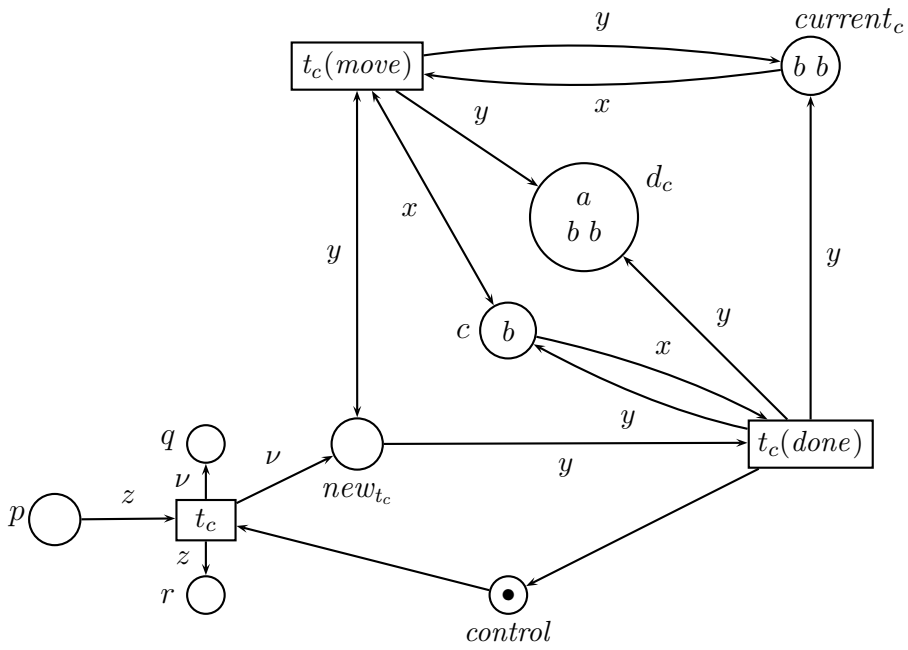

Figura 5.6: Simulación de red con contador con ν-APNs

η' nuevo, por lo que ha de ser $\eta \neq \eta'$. Si $t = t_c(move)$, como $F(c,t) = F(current_t, t)$ el token que se elimina de $current_t$ también está en c y, por lo tanto, no es η. Por último, si $t = t_c(done)$, como $\overline{M}'(c) = \overline{M}(new_t)$ y $\overline{M}'(new_t) = \emptyset$ se tiene que $\eta \notin \overline{M}'(c) \cup \overline{M}'(new_t)$.

Veamos ahora que si $M_1^* \to^* M_2^*$ entonces $M_1 \to^* M_2$ por inducción sobre la longitud del cómputo. Si en $M_1^* \to^* M_2^*$ no hay transiciones de T_C entonces es inmediato. Supongamos que la primera transición de T_C en el cómputo es $t \in T_C$ y que M_1^* y M_2^* son los únicos marcajes estables en ese cómputo, y veamos que entonces $M_1 \to^* M_2$, lo que implicará la tesis. Como las transiciones que no están en T_C producen marcajes estables a partir de marcajes estables t ha de ser la primera transición del cómputo. Sea k tal que $M_1^*(current_t) = \{\eta, \overset{k}{\ldots}, \eta\}$. Como los únicos marcajes con un token en el lugar *control* son los estables, las únicas transiciones que eliminan tokens de *control* son las de T_C y las únicas que le añaden tokens son las $t(done)$ con $t \in T_C$ ha de ser

$$M_1^*[t(\sigma')\rangle\overline{M}_0[t(move)(\bar{\sigma})\rangle\overline{M}_1 \ldots \overline{M}_{m-1}[t(move)(\bar{\sigma})\rangle\overline{M}_m[t(done)(\sigma'')\overline{M} \to^* M_2^*$$

como veíamos en la anterior implicación, con $0 \leq m \leq k$. Si $m < k$ entonces $S(M_2^*(c)) \cup S(M_2^*(new_t)) = \{\eta\} \not\supseteq \{\eta, \eta'\} = \overline{M}(current_t)$, lo que nos lleva a una contradicción, pues $\overline{M}$ sería ilegal y, por lo tanto, también lo sería M_2^*, lo que no puede ocurrir. Entonces $m = k$ y el cómputo es como el que veíamos en la otra implicación, es decir, existe M tal $\overline{M} = M^*$ y $M_1 \to^* M$. Entonces aplicamos inducción para obtener que $M \to^* M_2$, lo que nos permite concluir que

$M_1 \to^* M_2$, y la tesis se sigue.

$\square$

El recíproco del último resultado no es cierto, como veremos en el siguiente capítulo.

Capítulo 6

Expresividad de las redes de Petri móviles

Intuitivamente, la creación de nombres nuevos en sus distintas versiones introduce una complejidad extra en los sistemas. En este capítulo vamos a estudiar la decidibilidad de varios problemas en las dos versiones de creación de nombres introducidas: la alcanzabilidad, el recubrimiento y la acotación. Estos resultados, además de su importancia per se en la verificación de los sistemas, nos proporcionarán unos resultados de separación entre los distintos modelos que aparecen en esta tesis, estableciendo una jerarquía entre ellos.

6.1. Resultados de alcanzabilidad

En esta sección consideraremos problemas clásicos de alcanzabilidad. Probaremos que la alcanzabilidad es decidible para sistemas MSPN con naturales, pero indecidible para MSPNs con identificadores abstractos. Sin embargo, veremos que la alcanzabilidad de submarcajes también es indecidible para MSPNs con naturales, lo que las distinguirá de las redes P/T (para las que la alcanzabilidad de submarcajes es decidible), probando que son más expresivas que estas.

La alcanzabilidad es decidible para sistemas MSPN con naturales

En el capítulo anterior hemos visto que los sistemas MSPN con naturales son equivalentes a las redes con contadores, por lo que probaremos la decidibilidad de la alcanzabilidad para estas últimas. En particular, vamos a demostrarla reduciéndola a la alcanzabilidad en redes P/T, para las que la alcanzabilidad es decidible [Reu90]. Dada la red N con n contadores y el marcaje M que queremos alcanzar desde el marcaje inicial M_0, construimos una red P/T N^* con marcaje inicial M_0^* y un marcaje M^*, que será alcanzable en N^* desde M_0^* si y sólo si M es alcanzable desde M_0 en N.

Si $P_C = \{c_1, \ldots, c_n\}$ es el conjunto de contadores denotamos (i, n_i) al token

que se encuentra en c_i, es decir, $M(c_i) = \{(i, n_i)\}$, y $\overline{Id} = \{(i,j) \mid 1 \leq i \leq n,\ 0 \leq j \leq n_i\}$, el conjunto de identificadores que pueden aparecer en cualquier secuencia de marcajes que nos lleve a M, como se sigue de la proposición 5.1.4.

La clave de la demostración es desdoblar cada lugar $p \in P$ en el conjunto $\{p(i,j) \mid (i,j) \in \overline{Id}\}$, de manera que cada token en un lugar $p(i,j)$ simule una aparición del identificador (i,j) en p. Evidentemente, esta simulación sólo funciona durante un tiempo finito, más exactamente mientras que sólo aparezcan identificadores de $\overline{Id}$, lo que es suficiente para decidir la alcanzabilidad de M. En la demostración también desdoblaremos las transiciones $t \in T$. La figura 6.1 muestra el desdoblamiento de una red con un contador. En ese caso, $\overline{Id} = \{(1,0),(1,1),(1,2)\} \approx \{0,1,2\}$.

Formalmente, si $N = (P,T,F)$ definimos $N^* = (P^*, T^*, F^*)$ de la siguiente manera:

- $P^* = \{p(i,j) \mid p \in P,\ (i,j) \in \overline{Id}\}$,
- $T^* = \{t(\sigma) \mid t \in T,\ \sigma \text{ modo de } t,\ \sigma(Var(t)) \subseteq \overline{Id}\}$,
- $F^* = \{(p(i,j), t(\sigma)) \mid \sigma(F(p,t)) = (i,j)\} \cup$
  $\{(t(\sigma), p(i,j)) \mid \sigma(F(t,p)) = (i,j)\}$.

Para cualquier marcaje M de N con $S(M) \subseteq \overline{Id}$ definimos el marcaje M^* de N^* dado por $M^*(p(i,j)) = M(p)(i,j)$.

Con estas notaciones se cumple el siguiente resultado.

Proposición 6.1.1 *Si $S(M_i) \subseteq \overline{Id}$ para $i \in \{1,2\}$, entonces*

$$M_1[t(\sigma)\rangle M_2 \Leftrightarrow M_1^*[t(\sigma)\rangle M_2^*.$$

Demostración. En primer lugar, téngase en cuenta que ${}^\bullet t(\sigma) = \{p(i,j) \mid \sigma(F(p,t)) = (i,j)\}$ y $t(\sigma)^\bullet = \{p(i,j) \mid \sigma(F(t,p)) = (i,j)\}$. Por tanto, ${}^\bullet t(\sigma)(p(i,j)) = \{\sigma(F(p,t))\}(i,j)$ (ambos se reducen a 1 si $\sigma(F(p,t)) = (i,j)$ o a 0 en caso contrario) y $t(\sigma)^\bullet(p(i,j)) = \{\sigma(F(t,p))\}(i,j)$ (análogamente).

- $\Rightarrow$ Si $M_1[t(\sigma)\rangle M_2$ entonces $\sigma(F(p,t)) \in M_1(p)$ para todo $p \in {}^\bullet t$ y $M_2(p) = M_1(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$ para todo $p \in P$. Pues bien, si $p(i,j) \in {}^\bullet t(\sigma)$ (y por tanto $\sigma(F(p,t)) = (i,j)$), entonces $M_1^*(p(i,j)) = M_1(p)(i,j) > 0$ porque $(i,j) \in M_1(p)$. Así que la transición está activada, dando lugar tras su disparo a $\overline{M} = M_1^* - {}^\bullet t(\sigma) + t(\sigma)^\bullet$. Teniendo en cuenta las observaciones del principio de la demostración, $\overline{M}(p(i,j)) = M_1^*(p(i,j)) - {}^\bullet t(\sigma)(p(i,j)) + t(\sigma)^\bullet(p(i,j)) = M_1(p)(i,j) - \{\sigma(F(p,t))\}(i,j) + \{\sigma(F(t,p))\}(i,j) = M_2(p)(i,j) = M_2^*(p(i,j))$ y por tanto $\overline{M} = M_2^*$.

- $\Leftarrow$ Si $M_1^*[t(\sigma)\rangle M_2^*$ entonces ${}^\bullet t(\sigma) \subseteq M_1^*$ y $M_2^* = M_1^* - {}^\bullet t(\sigma) + t(\sigma)^\bullet$. Veamos que t está activada para M_1 en modo σ. Para ello tomamos $p \in {}^\bullet t$. Si $\sigma(F(p,t)) =$

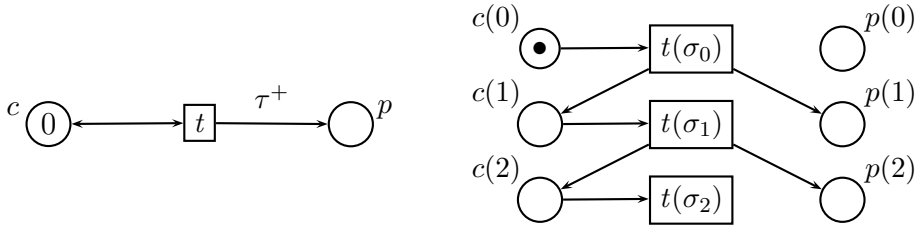

Figura 6.1: Desdoblamiento de una red con un contador respecto de $M(c) = \{2\}$ y $M(p) = \emptyset$

(i,j) entonces $p(i,j) \in {}^\bullet t(\sigma)$, de lo que se sigue que $p(i,j) \in M_1^*$, es decir, $M_1^*(p(i,j)) = M_1(p)(i,j) > 0$, y que $\sigma(F(p,t)) = (i,j) \in M_1(p)$, y la transición está activada. El marcaje obtenido coincide para todo p con $M_1(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$. Si $(i,j) \in Id$ entonces, teniendo de nueva en cuenta la observación del principio de la demostración, $M_1(p)(i,j) - \{\sigma(F(p,t))\}(i,j) + \{\sigma(F(t,p))\}(i,j) = M_1^*(p(i,j)) - {}^\bullet t(\sigma)(p(i,j)) + t(\sigma)^\bullet(p(i,j)) = M_2^*(p(i,j)) = M_2(p)(i,j)$, de lo que se sigue que $M_1[t(\sigma)\rangle M_2$, como queríamos demostrar.

□

Corolario 6.1.2 *Usando las notaciones anteriores, si $S(\overline{M}) \subseteq \overline{Id}$ entonces $\overline{M}$ es alcanzable en N si y sólo si $\overline{M}^*$ es alcanzable en $\overline{M}^*$.*

Demostración. Si $\overline{M}$ es alcanzable entonces por la proposición 5.1.4, M lo es a través de marcajes en los que sólo aparezcan identificadores en $\overline{Id}$. Entonces podemos aplicar la proposición anterior en cada paso para obtener un cómputo en N^* que lleva a $\overline{M}^*$ y $\overline{M}^*$ es alcanzable. Recíprocamente, para poder volver a aplicar la anterior proposición basta tener en cuenta que todo marcaje de N^* es imagen por $(\ )^*$ de un marcaje de N que tenga sólo identificadores de $\overline{Id}$. Efectivamente, si $\overline{M}'$ es un marcaje de N^* basta definir $\overline{M}(p)(i,j) = \overline{M}'(p(i,j))$, que trivialmente cumple que $\overline{M}^* = \overline{M}'$. Pues bien, si $\overline{M}^*$ es alcanzable entonces lo es a través de un cómputo de estados que son imagen por $(\ )^*$ de marcajes de N. Entonces podemos aplicar la proposición anterior a cada uno de esos disparos para obtener un cómputo en N que lleva a $\overline{M}$.

□

Como la alcanzabilidad es decidible para redes P/T obtenemos el siguiente resultado.

Corolario 6.1.3 *La alcanzabilidad es decidible para redes con contadores.*

Nótese que para el anterior resultado de decidibilidad es crucial poder contar con una cota superior del número de identificadores que ha creado cada contador.

La alcanzabilidad es indecidible para sistemas MSPN con identificadores abstractos

Ahora vamos a demostrar que la alcanzabilidad es indecidible para sistemas MSPN en los que se usen nombres abstractos como identificadores. Lo hacemos reduciendo la alcanzabilidad de un estado de la forma $\langle p, c_1 = 0, c_2 = 0\rangle$ de una máquina de dos contadores (TCM) a la alcanzabilidad de un marcaje de un sistema MSPN mediante una simulación débil, usando la técnica de detección estática de basura.

Sea $(\mathcal{S}, \mathcal{I})$ una TCM. Reducimos la alcanzabilidad en $(\mathcal{S}, \mathcal{I})$ a la alcanzabilidad en alguna MSPN con identificadores abstractos definiendo una MSPN con una sola red, sin lugares localidad y sin transiciones de movimiento o de sincronización, es decir, una ν-APN en la que, por simplicidad, permitimos tokens ordinarios.

En cada marcaje habrá un solo identificador "válido". Para especificar dicho identificador usaremos un lugar especial *current*, donde almacenamos el token identificador cuyo valor es el válido en cada momento. Simularemos el valor del contador c_i a través del número de tokens válidos en el lugar c_i. Usamos lugares ordinarios (que contengan tokens ordinarios) $\{s_0, \ldots, s_k\}$, uno para cada estado de la TCM, de manera que un token en s_i significa que la máquina se encuentra en el estado s_i. Además, usaremos otros lugares ordinarios auxiliares para manipular el control.

Las instrucciones de la forma $Inc(i, s, t)$ se simulan de manera inmediata, añadiendo a c_i una copia del token que se encuentre en *current*, y pasando el control (es decir, un token ordinario) de s a t (véase la figura 6.2). Sin embargo, las instrucciones de la forma $Dec(i, s, t, u)$ sólo se pueden simular débilmente, ya que no tenemos la posibilidad de chequear el cero. La idea clave es decidir de manera no determinista si el valor de c_i es o no nulo. Por tanto, podemos tener simulaciones de la TCM correctas, en las que todas las decisiones han sido acertadas, e incorrectas, en las que al menos una decisión fue errónea. Sin embargo, haremos la simulación de manera que en las simulaciones correctas de la máquina sólo aparezcan en c_1 y c_2 tokens válidos (los que coinciden con el valor almacenado en *current*).

La figura 6.3 muestra el caso en el que se elige decrementar el contador, lo que solamente puede hacerse cuando sea legal hacerlo, es decir, cuando haya un token válido en el contador. En ese caso, podemos eliminar uno de esos tokens del contador y mover un token ordinario del viejo al nuevo estado. El caso en el que elegimos proceder como si el valor de c_i fuese cero es más complicado, pues ésta podría ser una mala elección. La figura 6.4 muestra la simulación en este caso. Para identificar si hemos hecho una mala elección o no, generaremos un nuevo identificador válido y sustituiremos (transición *swap*) todos los tokens válidos que se encuentren en el otro contador por el nuevo identificador válido. Como no manipulamos el contador c_i, sólo mantendremos el invariante de que todos los tokens que aparecen en los contadores son válidos si la elección fue correcta (es decir, si el contador c_i no tenía tokens), ya que los tokens no válidos no se pueden

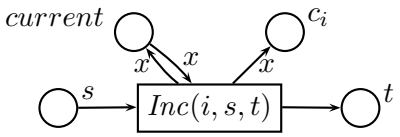

Figura 6.2: Simulación de $Inc(i, s, t)$

eliminar. Una vez que hemos cambiado los viejos tokens por los nuevos, podemos disparar la transición *reset*. De nuevo, sería posible disparar esta transición antes de sustituir todos los identificadores no válidos, pero esto también dejaría tokens no válidos y se identificaría por tanto como una simulación incorrecta.

A continuación formalizamos las ideas anteriores.

Definición 6.1.4 *Dada la TCM* $(\mathcal{S}, \mathcal{I})$ *definimos la* ν*-APN* $N = N(\mathcal{S}, \mathcal{I}) = (P, T, F)$ *de la siguiente manera:*

- $P = \mathcal{S} \cup \{current, c_1, c_2\} \cup \{a_D, fresh_D \mid D = Dec(i, s, t, u) \in \mathcal{I}\}$
- $T = \mathcal{I} \cup \{new_D, reset_D, swap_D, zero_D \mid D = Dec(i, s, t, u) \in \mathcal{I}\}$
- *F es la función parcial con menor dominio tal que:*
  - *Para cada* $I = Inc(i, s, t)$, $F(s, I) = F(I, t) = \varepsilon$ *y* $F(current, I) = F(I, current) = F(I, c_i) = x$,
  - *Para cada* $D = Dec(i, s, t, u)$,
    $F(s, D) = F(s, zero_D) = F(D, t) = F(zero_D, a_D) = F(a_D, new_D) = F(reset_D, u) = \varepsilon$,
    $F(new_D, fresh_D) = \nu$,
    $F(fresh_D, reset_D) = F(reset_D, current) = F(c_j, swap_D) = F(swap_D, current) = F(current, swap_D) = x$ *y*
    $F(fresh_d, swap_D) = F(swap_D, fresh_D) = F(current, reset_D) = y$.
- *Por último, dada una configuración de la TCM* $r = \langle s, c_1 = n_1, c_2 = n_2 \rangle$ *definimos el marcaje* M_r *de* N *dado por* $M_r(s) = 1$, $M_r(p) = \emptyset$ *para todo* $p \in P \setminus \{s, c_1, c_2\}$, $M_r(current) = \eta$ *y* $M_r(c_i) = \{\eta, \overset{n_i}{\ldots}, \eta\}$ *para* $i = 1, 2$.

Definición 6.1.5 *Un marcaje* M *de* N *es* legal *si* $M(s) = 1$ *para algún* $s \in \mathcal{S}$ *implica que existe* $\eta \in Id$ *tal que* $S(M) = \{\eta\}$. *Diremos que* M *es ilegal si no es legal.*

Lema 6.1.6 *Se cumplen las siguientes propiedades:*

1. *Para toda configuración* r, *el marcaje* M_r *es legal.*
2. *Si* M *es legal y existe* $s \in \mathcal{S}$ *tal que* $M(s) = 1$ *y* $M(t) = 0$ *para todo* $t \neq s$ *entonces* $M = M_r$ *con* $r = \langle s, c_1 = |M(c_1)|, c_2 = |M(c_2)| \rangle$.

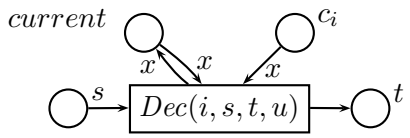

Figura 6.3: Simulación de $Dec(i,s,t,u)$ (eligiendo $c_i > 0$)

3. *Si $M \to^* M'$ y M' es legal entonces M es legal.*

4. *Si M_0 es legal y $M_0 \to M_1 \to \ldots M_n = M_r$ entonces M_i es legal para $i = 1, \ldots, n$.*

Demostración.

1 y 2 Trivial por las definiciones de marcajes legales y M_r.

3 Supongamos que M es ilegal y veamos que entonces también lo es M'. Por ser M ilegal existe $s \in \mathcal{S}$, $\eta, \eta' \in Id$ tales que $M(s) = 1$, $M(current) = \{\eta\}$ y $\eta' \in M(c_i)$ para algún $i \in \{1, 2\}$. Si vemos que para cualquier sucesor M' de M se cumple que $\eta' \in M'(c_i) \setminus M'(current)$ entonces M' no puede ser legal.

Las únicas transiciones adyacentes a *current* son del tipo $reset_D$ y $swap_D$. Como $F(current, swap_D) = F(swap_D, current)$ la transición $swap_D$ no modifica el contenido del lugar *current*. Sin embargo, $F(reset_D, current) \neq F(fresh_D, reset_D) = F(reset_D, current)$, de modo que el disparo de $reset_D$ sustituye el token de *current* por el que se encontraba en $fresh_D$, que a su vez es distinto de η' por definición de modos que tengan a ν en su dominio. El hecho de que $\eta' \notin M'(current)$ implica que $\eta' \in M'(c_i)$. Esto es así porque las únicas transiciones que eliminan tokens de c_i son del tipo $D = Dec(i,s,t,u)$ y $swap_D$. En ambos casos se tiene que $F(current, D) = F(c_i, D)$ y $F(current, swap_D) = F(c_i, D)$, de modo que el token eliminado de *current* coincide con el que se encuentra en *current*, que como acabamos de ver no puede ser η'.

4 Se sigue inmediatamente de los apartados anteriores.

□

Lema 6.1.7 *Si $D = Dec(i,s,t,u)$ y $M_r[zero_D\rangle \ldots M_{r'}$ entonces $M_r(c_i) = \emptyset$.*

Demostración. Supongamos que no hay ningún marcaje en la traza de la forma $M_{r''}$ además de M_r y $M_{r'}$. Existen η y η' distintos tales que $S(M_r) = \{\eta\}$ y $S(M_{r'}) = \{\eta'\}$, lo que implica que $S(M_r(c_i)) \subseteq \{\eta\}$ y $S(M_{r'}(c_i)) \subseteq \{\eta'\}$. Simplemente por inspección de la traza se puede ver que ninguna de las transiciones modifica el lugar c_i, es decir, $M_r(c_i) = M_{r'}(c_i)$ y por tanto, ambos han de ser vacíos.

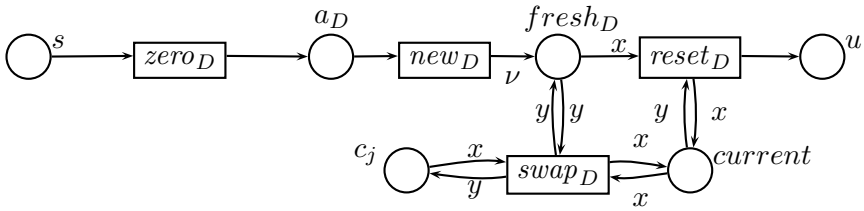

Figura 6.4: Simulación de $D = Dec(i, s, t, u)$ (eligiendo $c_i = 0$)

□

Proposición 6.1.8 *Dada una TCM* $(\mathcal{S}, \mathcal{I})$*, la configuración* $r = \langle s, c_1 = 0, c_2 = 0\rangle$ *es alcanzable desde* $r_0 = \langle s_0, c_1 = 0, c_2 = 0\rangle$ *si y sólo* M_r *es alcanzable en* $N(\mathcal{S}, \mathcal{I})$ *desde* M_{r_0}*.*

Demostración. La demostración se basa en el hecho de que toda ejecución del sistema MSPN construido que alcance un marcaje M en el que $M(c_i) = \emptyset$ simula una ejecución de la TCM, lo que significa que todas las elecciones han sido correctas, o de lo contrario habría tokens basura en algunos de los contadores, que no pueden ser eliminados por construcción. Además, todas las ejecuciones de la máquina se pueden simular de manera correcta, haciendo la elección correcta siempre que corresponda decidir si un lugar está o no vacío.

Para probar la corrección de la simulación veamos que si $r = \langle s, c_1 = n_1, c_2 = n_2\rangle \to r' = \langle t, c_1 = m_1, c_2 = m_2\rangle$ entonces $M_r \to^* M_{r'}$, por distinción de casos sobre la instrucción utilizada.

- $I = Inc(i, s, t)$: Para fijar ideas suponemos $i = 1$, pues el caso $i = 2$ es totalmente simétrico. Tenemos que $r = \langle s, c_1 = n_1, c_2 = n_2\rangle \to r' = \langle t, c_1 = n_1 + 1, c_2 = n_2\rangle$. Se cumple que ${}^\bullet Inc(1, s, t) = \{s, current\}$ y $Inc(1, s, t)^\bullet = \{current, c_1, t\}$. Si $M_r(current) = \{\eta\}$ entonces la transición $Inc(1, s, t) \in T$ está activada en modo σ, donde $\sigma(x) = \eta$, pues por definición de M_r, $M_r(s) = 1$. El marcaje alcanzado es M', que por construcción coincide con M excepto en los lugares s, t y c_1, para los que se verifica que $M'(s) = 0$, $M'(t) = 1$ y $M'(c_1) = M_r(c_1) + \{\sigma(F(I, c_1))\} = \{\eta, \overset{n_1+1}{\ldots}, \eta\} = M_{r'}(c_1)$.

- $D = Dec(1, s, t, u)$ con $n_1 > 0$: este caso es análogo al anterior, teniendo en cuenta que si $n_1 > 0$ entonces existe $\eta \in M_r(c_1) \cap M_r(current)$, y por tanto se puede disparar $Dec(1, s, t) \in T$ con modo $\sigma(x) = \eta$ tal que $M_r(current) = \{\eta\}$.

- $D = Dec(1, s, t, u)$ con $n_1 = 0$, es decir, $r = \langle s, c_1 = 0, c_2 = n_2\rangle \to r' = \langle u, c_1 = 0, c_2 = n_2\rangle$. Veamos que en este caso

$$M_r[zero_D\rangle M_1[new_D\rangle M_2[swap_D(\sigma)\rangle \overset{n_2}{\ldots}[swap_D(\sigma)\rangle M_3[reset_D(\sigma')\rangle M_{r'}$$

Claramente, desde M_r se pueden disparar consecutivamente $zero_D$ y new_D, obteniendo M_2 tal que $M_2(new_D) = \eta'$, $M_2(current) = \eta$, $M_2(c_1) = \emptyset$ y $M_2(c_2) = \{\eta, \overset{n_2}{\ldots}, \eta\}$. Si tomamos $\sigma(x) = \eta$ y $\sigma(y) = \eta'$ entonces podemos disparar $swap_D$ en modo σ, pues ${}^{\bullet}swap_D = \{fresh_D, c_2\} = swap_D^{\bullet}$, $F(c_2, swap_D) = x$ y $M_2(c_2) = \{\eta, \overset{n_2}{\ldots}, \eta\}$, obteniendo un marcaje que sólo difiere de M_2 en el contenido en c_2, que es $\{\eta, \overset{n_2-1}{\ldots}, \eta, \eta'\}$. Por lo tanto, podemos volver a disparar la misma transición en el mismo modo, y obtener, tras n_2 disparos, el marcaje M_3 que coincide con M_2 en todos los lugares salvo c_2, donde $M_3(c_2) = \{\eta', \overset{n_2}{\ldots}, \eta'\}$. En este momento podemos disparar $reset_D$, tomando $\sigma'(x) = \eta'$ y $\sigma'(y) = \eta$, obteniendo, por ser $F(fresh_D, reset_D) = F(reset_D, current) = x$ y $F(reset_D, u) = \varepsilon$, el marcaje $M_{r'}$.

En cualquier caso, hemos probado que si $r \to r'$ entonces $M_r \to^* M_{r'}$. Entonces, por inducción sobre la longitud de los cómputos, si $r = \langle s_0, c_1 = 0, c_2 = 0\rangle$ es alcanzable en la TCM entonces M_r es alcanzable en la simulación, y se sigue la implicación a la derecha.

Supongamos ahora que M_r es alcanzable desde M_{r_0} y veamos entonces que r es alcanzable desde r_0, lo que implicará la tesis. Como M_r es alcanzable existe una cadena $M_0[t_1(\sigma_1)\rangle M_1 \ldots M_{n-1}[t_n(\sigma_n)\rangle M_n = M_r$ y, por el lema 6.1.6, todos los M_i son legales.

Dada $r = \langle s, c_1 = n_1, c_2 = n_2\rangle$, si $M_r[u(\sigma)\rangle M$, como $M_r(t) = 0$ para todo $t \in \mathcal{S} \setminus \{s\}$, ha de ser necesariamente $u = I = Inc(i, s, t)$, $u = D = Dec(i, s, t, u)$ o $u = zero_D$. En los dos primeros casos, por el razonamiento inverso a los vistos en la implicación anterior, existe r' tal que $M = M_{r'}$ y $r \to r'$. En el tercer caso, por ser todos los marcajes de la cadena legales y existir r' tal que $M_{r'}$ sea el primer marcaje de esa forma posterior a M_r en la cadena, por construcción ha de ser necesariamente

$$M_r[zero_D\rangle M_1[new_D\rangle M_2[swap_D(\sigma_1)\rangle \overset{n}{\ldots} [swap_D(\sigma_n)\rangle M_3[reset_D(\sigma')\rangle M_{r'}$$

Por el lema 6.1.7 se cumple que $M_r(c_j) = \emptyset$ y por lo tanto desde r se puede ejecutar la instrucción D con $c_j = 0$, obteniendo r', como queríamos demostrar.

□

Corolario 6.1.9 *La alcanzabilidad es indecidible para sistemas MSPN con identificadores abstractos.*

Demostración. Es un hecho bien conocido que la alcanzabilidad en TCMs es equivalente a la alcanzabilidad de los estados especiales $\langle s, c_1 = 0, c_2 = 0\rangle$, y ambas son indecidibles [Min67, HU79]. Supongamos que la alcanzabilidad para sistemas MSPN con identificadores abstractos es decidible, y usemos esta suposición para decidir si un estado dado $\langle s, c_1 = 0, c_2 = 0\rangle$ es alcanzable, lo que demostraría la tesis. Tomamos M tal que $M(s) = 1$, $M(t) = 0$ para todo $t \neq s$

en $\mathcal{S}$, $M(current) = \eta$ para algún $\eta \in Id$, $M(q) = \emptyset$ con $q \in P_{Id} \backslash \{current\}$. Por la proposición 6.1.8 sabemos que M es alcanzable si y sólo si $\langle s, c_1 = 0, c_2 = 0 \rangle$ es alcanzable. Por tanto, basta con elegir uno de esos marcajes y decidir si es o no alcanzable para decidir la alcanzabilidad de $\langle s, c_1 = 0, c_2 = 0 \rangle$.

□

Así que la alcanzabilidad es decidible cuando consideramos identificadores naturales, pero indecidible si los consideramos abstractos. Este hecho nos dice que no es posible simular ν-APNs por medio de redes con contadores, ni siquiera débilmente, como ya adelantamos al final del capítulo anterior.

La alcanzabilidad de submarcajes es indecidible para sistemas MSPN con naturales

El resultado que titula esta sección se puede probar de manera muy similar a la indecidibilidad de la alcanzabilidad para sistemas MSPN con identificadores abstractos, pero preferimos presentarlo en una sección diferenciada debido a su importancia. En efecto, el hecho de que la alcanzabilidad de los sistemas MSPN con naturales sea decidible podría darnos la impresión errónea de que éstos son equivalentes a las redes P/T pero, como veremos a continuación, esto no es así, ya que la alcanzabilidad de submarcajes si es decidible para redes P/T. De hecho, vamos a ver que basta con un sólo contador para obtener este resultado.

Dada una configuración de una TCM, $\langle p, c_1 = 0, c_2 = 0 \rangle$, definimos un sistema MSPN con naturales que simula débilmente a la máquina. La simulación de Inc y Dec (en el caso $c_i > 0$) es exactamente como la que vimos en la sección anterior. En el caso en el que elijamos $c_i = 0$, sustituimos la etiqueta *new* por *succ*, con su lugar contador asociado, *counter*. Para controlar que el chequeo del cero se simula correctamente hacemos como en la sección anterior. Usamos la transición *succ* para generar los identificadores de la red. Cada vez que se dispara una transición $zero_D$ incrementamos el contador de la red y actualizamos los tokens que se encuentran en el otro contador (de la TCM) con el nuevo valor del lugar contador de la red. Sin embargo, por definición de los sistemas MSPN con naturales, sólo puede haber una transición sucesor, de manera que los lugares contador y las transiciones *succ* deben ser los mismos para todas esas instrucciones. Por ello, debemos ser cuidadosos y sólo permitir el disparo de la transición $reset_D$ que le corresponda a esa instrucción D. Para ello, añadimos un lugar de control extra como postcondición de $zero_D$ (ver figura 6.5).

Definición 6.1.10 *Dada la TCM* $(\mathcal{S}, \mathcal{I})$ *definimos la red con un contador* $N = N(\mathcal{S}, \mathcal{I}) = (P, T, F)$ *de la siguiente manera:*

- $P = \mathcal{S} \cup \{current, c_1, c_2, fresh, a, counter\} \cup \{a_D \mid D = Dec(i, s, t, u) \in \mathcal{I}\}$
- $T = \mathcal{I} \cup \{succ\} \cup \{swap_D, reset_D, zero_D \mid D = Dec(i, s, t, u) \in \mathcal{I}\}$

- *F es la función parcial con menor dominio tal que:*
  - *Para cada $I = Inc(i, s, t)$, $F(s, I) = F(I, t) = \varepsilon$ y $F(current, I) = F(I, current) = F(I, c_i) = x$,*
  - *Para cada $D = Dec(i, s, t, u)$:*
    $F(s, D) = F(s, zero_D) = F(D, t) = F(zero_D, a) = F(zero_D, a_D) = F(a, succ) = F(a_D, reset) = F(a_D, swap_D) = F(swap_D, a_D) = \varepsilon$,
    $F(reset_D, u) = \varepsilon$,
    $F(succ, fresh) = F(succ, counter) = \tau^+$ y $F(counter, succ) = \tau^-$,
    $F(fresh, reset_D) = F(reset_D, current) = F(c_j, swap_D) = x$,
    $F(swap_D, current) = F(current, swap_D) = x$,
    $F(fresh, swap_D) = F(swap_D, fresh) = F(current, reset_D) = y$.

Ahora podemos expresar la relación entre la TCM y su simulación débil por medio de la siguiente proposición.

Proposición 6.1.11 *Una configuración $\langle p, c_1 = 0, c_2 = 0\rangle$ de una TCM dada es alcanzable si y sólo si existe $N \in \mathbb{N}$ tal que el marcaje M_N dado por $M_N(counter) = M_N(current) = \{(1, N)\}$, $M_N(p) = 1$ y $M_N(q) = \emptyset$, para todo $q \notin \{counter, current, p\}$, es alcanzable en la simulación.*

Demostración. La demostración sigue un esquema prácticamente idéntico a la de la proposición 6.1.8. La única diferencia está en el hecho de que allí teníamos una transición new_D y un lugar $fresh_D$ para cada $D = Dec(i, s, t, u) \in \mathcal{I}$, mientras que ahora, por no poder tener más de una transición sucesor, contamos únicamente con la transición *succ* y un lugar $fresh$.

□

Como corolario inmediato tenemos el siguiente resultado.

Corolario 6.1.12 *La alcanzabilidad de submarcajes de sistemas MSPN con naturales es indecidible.*

Demostración. Dada una máquina de dos contadores y una configuración suya $\langle p, c_1 = 0, c_2 = 0\rangle$, construimos el sistema MSPN de la definición 6.1.10. Sea M el marcaje definido para todos los lugares, excepto para el lugar *counter* y para el lugar *current*, mediante $M(p) = 1$ y $M(q) = \emptyset$, para todo $q \notin \{counter, current, p\}$. Este submarcaje es alcanzable si y sólo si existe un marcaje M_N alcanzable (siguiendo las notaciones de la proposición anterior), lo que ocurre si y sólo si la configuración $\langle p, c_1 = 0, c_2 = 0\rangle$ es alcanzable en la TCM, aplicando la proposición anterior.

□

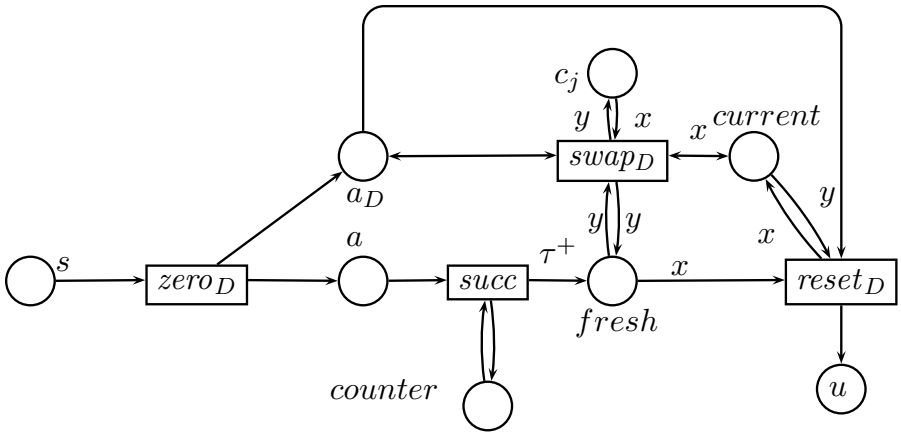

Figura 6.5: Simulación de $D = Dec(i, s, t, u)$ con identificadores naturales (eligiendo $c_i = 0$)

Es un hecho conocido [DR98] que la alcanzabilidad de submarcajes de redes P/T es equivalente a la alcanzabilidad propiamente dicha. Efectivamente, dado un marcaje parcial M se puede construir una nueva red con transiciones que eliminen tokens de los lugares que no se encuentran en el dominio de M. Por tanto, la alcanzabilidad del marcaje parcial M es equivalente a la alcanzabilidad del marcaje global M' que extiende a M tomando $M(p) = \emptyset$, para todo $p \notin Dom(M)$. No obstante, las restricciones impuestas sobre las transiciones *succ* evitan la posibilidad de eliminar tokens del lugar contador, de modo que este razonamiento no se puede aplicar a nuestras redes.

Las dos simulaciones de TCMs que hemos construido, la que usa identificadores naturales y la que usa identificadores abstractos, tienen un lugar *current* que contiene en cada momento el token válido. En el caso de los identificadores naturales, el valor del token en *current* coincide con el de *counter*. Sin embargo, la información que se puede extraer de ese valor es muy diferente en cada caso. Si usamos números naturales, conocemos parte de la historia del sistema: una cota superior del número de identificadores creados y el orden en el que éstos fueron creados. Sin embargo, si usamos identificadores abstractos no conocemos ninguno de estos datos. De hecho, no nos importa qué identificador concreto encontramos en *current* ya que, intuitivamente, y como formalizaremos en la siguiente sección, estamos trabajando módulo α-conversión y, por tanto, cualquier identificador podría haber sido creado en su lugar.

6.2. Resultados de recubrimiento

En la sección anterior consideramos problemas de alcanzabilidad. En particular, probamos que tanto la alcanzabilidad de sistemas MSPN con identificadores abstractos como la alcanzabilidad de submarcajes de sistemas MSPN con natu-

rales son indecidibles. Afortunadamente, las propiedades de seguridad de nuestros sistemas se definen habitualmente en términos de la propiedad de recubrimiento. En esta sección probaremos que el recubrimiento es decidible para sistemas MSPN con identificadores, tanto naturales como abstractos.

El recubrimiento es decidible para sistemas MSPN con identificadores abstractos

En esta sección probaremos la decidibilidad del problema del recubrimiento para MSPNs con identificadores abstractos. Como vimos en el capítulo 5, los sistemas MSPN son equivalentes a las ν-APNs, de modo que lo haremos para estas últimas. Para probar la decidibilidad del recubrimiento usaremos la técnica basada en buenos cuasi-órdenes [ACJT00, AN00, Mil85] descrita en el capítulo 2. Esta técnica se descompone en tres pasos. En primer lugar, es necesario tener un orden definido en el conjunto de estados, los marcajes en nuestro caso, que sea el que induce la propiedad de recubrimiento. Este orden tiene que ser monótono respecto a la relación de transición, es decir, respecto de los disparos de transiciones. El segundo paso es probar que el orden es de hecho un buen cuasi orden. Por último, tenemos que probar que la función predecesor, que dado un estado devuelve el conjunto de estados desde los que es posible llegar a él, es efectivamente computable, en el sentido que precisaremos más adelante. Cuando se cumplen todas estas propiedades, se dice que el sistema es *bien estructurado*, y para estos sistemas se prueba en [ACJT00] que el recubrimiento es decidible.

Empecemos viendo cuál es el orden en el que estamos interesados. El orden que induce la noción de recubrimiento que vimos en el capítulo 4 es el siguiente:

$$M_1 \sqsubseteq M_2 \Leftrightarrow M_1(p) \subseteq M_2(p) \text{ para todo } p \in P$$

Sin embargo, es trivial ver que este orden no es un buen cuasi orden, sin más que considerar el siguiente contraejemplo. Sea N una ν-APN con un sólo lugar p y sea $(a_i)_{i=1}^{\infty}$ una sucesión de identificadores distintos dos a dos. Entonces basta considerar $M_i(p) = \{a_i\}$ para todo $i = 1, 2, \ldots$ La secuencia $M_1, M_2, M_3, \ldots$ cumple trivialmente que para todo $i < j$ no es cierto que $M_i \sqsubseteq M_j$.

Sin embargo, este orden es demasiado restrictivo, ya que no tiene en cuenta el carácter abstracto de los nombres nuevos, es decir, el hecho de que cuando se crea un nuevo nombre en realidad cualquier otro nuevo nombre podría haber sido creado. Por tanto, la alcanzabilidad (o el recubrimiento) de un marcaje dado es equivalente a la alcanzabilidad (o el recubrimiento) de cualquier marcaje que resulte de renombrar todos los nombres creados en aquel. Sin embargo, por homogeneidad, vamos a suponer que podemos renombrar todos los nombres, incluso aquéllos que aparecían en el marcaje inicial (que, al fin y al cabo, son un número fijo de identificadores). Para capturar estas intuiciones, introducimos una noción de α-equivalencia entre marcajes, prestada del renombramiento de nombres ligados del π-cálculo [MPW92]: identificamos marcajes bajo renombramiento de nombres abstractos.

Definición 6.2.1 *Definimos $\equiv_\alpha$ como la menor relación de equivalencia entre marcajes que satisface $M \equiv_\alpha M[\eta'/\eta]$ con $\eta \in S(M)$ y $\eta' \notin S(M)$.*

El hecho de considerar que todo nombre puede ser renombrado viene impuesto por carecer de un operador de restricción o de declaración de ámbitos al estilo del operador ν del cálculo π. Siguiendo esta analogía, el poder renombrar todos los nombres se traduce a que todos los nombres están bajo un operador de restricción, y todos estos operadores están implícitos al nivel superior respecto del operador paralelo (que en nuestro caso es simplemente la unión de componentes). Intuitivamente, es como si en el momento de creación de cada nombre estuviésemos aplicando de izquierda a derecha la regla de extrusión del π-cálculo que veíamos en la sección 1.4 de la introducción.

Veamos una caracterización de la equivalencia definida en base al renombramiento concreto de todos los nombres del marcaje. Dados un marcaje M y $h : Id \to Id$ biyectiva definimos el marcaje $h(M)$, dado por $h(M)(p)(a) = M(p)(h^{-1}(a))$.

Lema 6.2.2 *Sean M y M' dos marcajes. Entonces $M \equiv_\alpha M'$ si y sólo si existe una biyección $h : S(M) \to S(M')$ tal que $M' = h(M)$.*

Demostración.

$\Rightarrow$ Probamos el resultado por inducción sobre la derivación de $M \equiv_\alpha M'$.

- Para $M \equiv_\alpha M$ basta tomar como h a la función identidad, que es una biyección.
- $M' \equiv_\alpha M \Rightarrow M \equiv_\alpha M'$. Por la hipótesis de inducción, existe $h : S(M') \to S(M)$ que verifica $h(M') = M$ y, por tanto, h^{-1} verifica la tesis: Por la hipótesis de inducción se cumple que $h(M)(p)(a) = M(p)(h^{-1}(a)) = M'(p)(a)$. Entonces, $h^{-1}(M')(p)(b) = M'(p)((h^{-1})^{-1}(b)) = M'(p)(h(b)) = M(p)(h^{-1}(h(b))) = M(p)(b)$, de manera que $M(p)(b) = h^{-1}(M')(p)(b)$ para cada p y b, con lo que $M = h^{-1}(M')$.
- Supongamos que hemos derivado $M \equiv_\alpha M'$ mediante transitividad a partir de $M \equiv_\alpha M''$ y $M'' \equiv_\alpha M'$. Entonces existen sendas biyecciones $h_1 : S(M) \to S(M'')$ y $h_2 : S(M'') \to S(M')$ tales que $h_1(M) = M''$ y $h_2(M'') = M'$. Tomemos $h = h_2 \circ h_1$, que es una biyección de $S(M)$ a $S(M')$ tal que $h(M) = M'$: Por hipótesis de inducción, $h_1(M)(p)(b) = M(p)(h_1^{-1}(b)) = M'(p)(b)$ para todo p y $b \in S(M)$ y $h_2(M)(p)(c) = M(p)(h_2^{-1}(c)) = M'(p)(c)$ para todo p y $c \in S(M')$. Por tanto, $h(M)(p)(c) = M(p)(h^{-1}(c)) = M(p)((h_2 \circ h_1)^{-1}(c)) = M(p)(h_1^{-1}(h_2^{-1}(c))) = M'(p)(h_2^{-1}(c)) = M''(p)(c)$, para todo p y $c \in S(M')$.
- $M \equiv_\alpha M[\eta'/\eta]$ con $\eta \in S(M)$ y $\eta' \notin S(M)$. Entonces $S(M[\eta'/\eta]) = (S(M) \setminus \{\eta\}) \cup \{\eta'\}$ y basta tomar $h(\eta) = \eta'$ y $h(a) = a$ para $a \neq \eta$.

$\Leftarrow$ Supongamos que $S(M) = \{\eta_1, \ldots, \eta_n\}$, $S(M') = \{\eta'_1, \ldots, \eta'_n\}$ y $h(\eta_i) = \eta'_i$. Procedemos por inducción sobre $m = |S(M) \cap S(M')|$.

- Si $m = 0$ entonces $S(M) \cap S(M') = \emptyset$. Esto implica que $\eta_{j+1} \in S(M[\eta'_i/\eta_i]_{i=1}^{j})$ y $\eta'_{j+1} \notin S(M[\eta'_i/\eta_i]_{i=1}^{j})$ para $j = 0, \ldots, n-1$, y por tanto la siguiente es una secuencia legal de α-equivalencias:

$$M \equiv_\alpha M[\eta'_1/\eta_1] \equiv_\alpha M[\eta'_1/\eta_1, \eta'_2/\eta_2] \equiv_\alpha \ldots \equiv_\alpha M[\eta'_i/\eta_i]_{i=1}^{n} = M'.$$

- Si $m > 0$ entonces existe $\eta \in S(M) \cap S(M')$. Sea $\eta' \notin S(M) \cup S(M')$. Entonces $M \equiv_\alpha M[\eta'/\eta]$ y $M' \equiv_\alpha M'[\eta'/\eta]$. Si denotamos h' a la biyección definida por $h'(\eta) = \eta'$ y $h'(a) = h(a)$ para $a \neq \eta$, entonces se cumple que $M'[\eta'/\eta] = h'(M[\eta'/\eta])$. Además, $|S(M[\eta'/\eta]) \cap S(M'[\eta'/\eta])| = m - 1$. Por hipótesis de inducción sabemos que $M[\eta'/\eta] \equiv_\alpha M'[\eta'/\eta]$ y por transitividad podemos concluir que $M \equiv_\alpha M'$.

$\square$

Escribiremos $M \equiv_h M'$ para indicar que $M \equiv_\alpha M'$ y $h(M) = M'$ (por lo tanto, $\equiv_h$ no es simétrica).

Proposición 6.2.3 *El comportamiento de las ν-APNs es invariante bajo α-conversión. Más especificamente, si $M_1 \equiv_h M_2$ y $M_1[t(\sigma)\rangle M'_1$ entonces existe M'_2 con $M'_1 \equiv_\alpha M'_2$ tal que $M_2[t(h(\sigma))\rangle M'_2$, donde $h(\sigma)(x) = h(\sigma(x))$.*

Demostración. Si $\lambda(t) \neq new$ entonces tomamos $M'_2 = h(M_2)$, donde $h(M)(p)(h(\eta)) = M(p)(\eta)$ para $p \in P_{Id}$. Se cumple que $M_2 \equiv_h M'_2$ y $M'_1[t(h(\sigma))\rangle M'_2$. Si $\lambda(t) = new$ entonces $S(M_2) = S(M_1) \cup \{\eta\}$ para algún η nuevo. Entonces se puede disparar t también en M_2 con modo $h(\sigma)$, obteniéndose M'_2 con $S(M'_2) = S(M'_1) \cup \{\eta'\}$. Definamos h' extendiendo h con $h(\eta) = \eta'$, que verifica que $M_2 \equiv_{h'} M'_2$.

$\square$

Por ejemplo, si representamos un marcaje M de la red de la figura 6.2 mediante una tupla $(M(p_1), M(p_2), M(p_3))$, entonces $M_1 = (\bullet, \eta, \emptyset)$ y $M_2 = (\bullet, \eta', \emptyset)$ son dos marcajes α-equivalentes de esa ν-APN. M_1 puede evolucionar a $(\emptyset, \emptyset, \eta)$ cuando dispara *aut* o a $(\emptyset, \eta, \eta'')$ para algún η'' nuevo, cuando dispara *new*. Análogamente, M_2 puede evolucionar a $(\emptyset, \emptyset, \eta')$ cuando dispara *aut* o a $(\emptyset, \eta', \eta''')$ para algún η''' cuando dispara *new*. En ambos casos, los marcajes alcanzados son de hecho α-equivalentes para cualesquiera η'' y η''' tales que $\eta'' \neq \eta$ y $\eta''' \neq \eta'$. En particular, podemos tomar $\eta'' = \eta'$ y $\eta''' = \eta$, lo que daría lugar a $(\emptyset, \eta, \eta')$ y $(\emptyset, \eta', \eta)$, que son equivalentes.

Definición 6.2.4 *Sean M_1 y M_2 dos marcajes de una ν-APN $\mathcal{N}$. We say $M \sqsubseteq M' \Leftrightarrow M(p) \subseteq M'(p)$ for all $p \in P$. Escribiremos $M_1 \sqsubseteq_\alpha M_2$ si existe un marcaje M'_1 tal que $M'_1 \equiv_\alpha M_1$ y $M'_1 \sqsubseteq M_2$.*

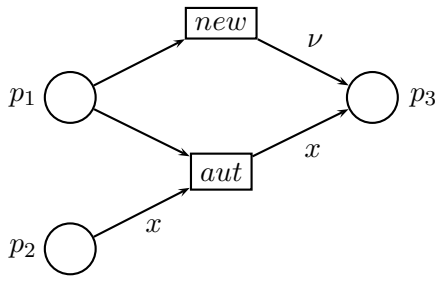

Figura 6.6: Una ν-APN

El orden $\sqsubseteq_\alpha$ induce una nueva noción de recubrimiento, que llamaremos α-recubrimiento. Veamos que $\sqsubseteq_\alpha$ cumple las condiciones que comentábamos más arriba.

Lema 6.2.5 *Sea A un conjunto y $\sqsubseteq$ el orden en $\mathcal{MS}(A)$ inducido por la identidad en A. Entonces, $\mathcal{A} \sqsubseteq \mathcal{A}' \Leftrightarrow \mathcal{A} \subseteq \mathcal{A}'$, para todo $\mathcal{A}, \mathcal{A}' \in \mathcal{MS}(A)$.*

Demostración.

$\Rightarrow$ Supongamos que $\mathcal{A} \sqsubseteq \mathcal{A}'$. Entonces $\mathcal{A}' \supseteq \sum_{a \in S(\mathcal{A})} \mathcal{A}'_a$ con $\mathcal{A}(a) = |\mathcal{A}'_a|$ y $a = b$ para todo $a \in S(\mathcal{A})$ y $b \in S(\mathcal{A}'_a)$. En particular, obtenemos que $S(\mathcal{A}'_a) = \{a\}$ para todo $a \in S(\mathcal{A})$. Entonces, $\mathcal{A}(a) = |\mathcal{A}'_a| = \mathcal{A}'_a(a) \leq \mathcal{A}'(a)$ para todo $a \in S(\mathcal{A})$.

$\Leftarrow$ Para cada $a \in S(\mathcal{A})$, sea $\mathcal{A}'_a$ el multiconjunto definido por $S(\mathcal{A}'_a) = \{a\}$ y $\mathcal{A}'_a(a) = \mathcal{A}(a)$. Entonces, si $b \in S(\mathcal{A})$, $\left(\sum_{a \in S(\mathcal{A})} \mathcal{A}'_a\right)(b) = \mathcal{A}'_b(b) = \mathcal{A}(b)$ y obtenemos que $\sum_{a \in S(\mathcal{A})} \mathcal{A}'_a \subseteq \mathcal{A}$. El resto de las condiciones se cumplen trivialmente, por definición de $\mathcal{A}'_a$.

□

Un marcaje es una función $M : P \to Id \to \mathbb{N}$ que nos dice, dado un lugar p y un identificador η, cuántas veces podemos encontrar al token η en p. Sin embargo, podemos representar los marcajes de la siguiente manera: $M : Id \to P \to \mathbb{N}$. Como los identificadores son nombres abstractos, el comportamiento de un sistema no depende de los nombres concretos elegidos, es decir, es invariante bajo renombramiento o α-conversión, como hemos probado en la proposición 6.2.3. Por tanto, podemos considerar que los marcajes (abstractos) son multiconjuntos en $\mathcal{MS}(P \to \mathbb{N})$, es decir, en $\mathcal{MS}(\mathcal{MS}(P))$.

De esta manera, representamos marcajes a través de multiconjuntos, con cardinalidad igual al número de identificadores distintos que aparecen en él. Consideremos por ejemplo una red con sólo dos lugares que contienen identificadores, p_1

y p_2, y un marcaje M tal que $M(p_1) = \{\eta, \eta, \eta', \eta''\}$ y $M(p_2) = \{\eta', \eta''\}$. Podemos representar ese marcaje por el multiconjunto de cardinalidad 3, ya que solamente aparecen en él 3 identificadores distintos, a saber $\{\{p_1, p_1\}, \{p_1, p_2\}, \{p_1, p_2\}\}$, donde el multiconjunto $\{p_1, p_1\}$ representa al identificador η, uno de los multiconjuntos $\{p_1, p_2\}$ representa a η' y el otro $\{p_1, p_2\}$ representa a η''.

Formalmente:

Definición 6.2.6 *Dado un marcaje M de una ν-APN $\mathcal{N}$ denotamos M_a al multiconjunto de lugares definido por $M_a(p) = M(p)(a)$, y $\overline{M}$ al multiconjunto de multiconjuntos de lugares definido por $\{M_a \mid a \in S(M)\}$. Escribiremos $\overline{Markings} = \{\overline{M} \mid M \in Markings\} \in \mathcal{MS}(\mathcal{MS}(P))$.*

Definición 6.2.7

- *Aplicando el lema 2.2.4 construimos $(P, =)$. Entonces, mediante el lema 2.2.6, obtenemos $\mathcal{MS}(P)$ con su orden.*
- *De nuevo, aplicando el lema 2.2.6 a partir de $\mathcal{MS}(P)$ con el orden obtenido en el punto anterior, obtenemos $(\mathcal{MS}(\mathcal{MS}(P)), \ll)$.*

Como consecuencia inmediata de cómo se ha definido $\ll$ y de los lemas 2.2.4 y 2.2.6 se tiene que

Lema 6.2.8 $(\overline{Markings}, \ll)$ *es un wqo.*

Lema 6.2.9 *Si M y M' son marcajes de una ν-APN tales que $\overline{M} \ll \overline{M'}$ entonces $M \sqsubseteq_\alpha M'$.*

Demostración. Si $S(\overline{M}) = \{M_1, \ldots, M_l\}$ entonces $\overline{M} \ll \overline{M}'$ implica que existen $\mathcal{A}'_1, \ldots, \mathcal{A}'_l$ tales que:

- $\sum_{i=1}^{l} \mathcal{A}'_i \subseteq \overline{M}'$
- $\overline{M}(M_i) = |\mathcal{A}'_i|$
- $M_i \sqsubseteq M''$ para todo i y $M'' \in S(\mathcal{A}'_i)$.

Denotemos $Id_i = \{a \in S(M) \mid M_a = M_i\}$ e $Id'_i = \{b \in S(M') \mid M'_b \in \mathcal{A}'_i\}$. Como $M(M_i) = |\mathcal{A}'_i|$ se cumple que $|Id_i| = |Id'_i|$, para cada i, y por tanto existe una biyección $h_i : Id_i \to Id'_i$. Como la familia $\{Id_i\}_{i=1}^n$ es una partición del conjunto $S(M)$ podemos definir $h : S(M) \to S(M')$ como $h(a) = h_i(a)$ cuando $a \in Id_i$, que es una biyección entre $S(M)$ y su imagen. Tomemos $M'' = h(M)$, con $h(M)$ definida como en el lema 6.2.2.

Por el lema 6.2.2 se cumple que $M \equiv_\alpha M''$. Veamos que $M'' \sqsubseteq M'$, lo que nos permitirá concluir que $M \sqsubseteq_\alpha M'$.

Sean $b \in S(M'')$ y $a \in S(M)$ tales que $h(a) = b$. En particular, la tercera de las condiciones anteriores nos dice que $M_a \sqsubseteq M_b$ lo que, por el lema 6.2.5, implica

que $M_a \subseteq M_b$. Esto significa que para todo $p \in P, M_a(p) \leq M'_b(p)$. Entonces, $M''(p)(b) = M(p)(a) = M_a(p) \leq M'_b(p) = M'(p)(b) \Rightarrow M''(p) \subseteq M'(p)$, para todo $p \in P$ y la tesis se sigue.

□

Finalmente, podemos concluir que el orden $\sqsubseteq_\alpha$ es, efectivamente, un wqo.

Proposición 6.2.10 *$\sqsubseteq_\alpha$ es un wqo.*

Demostración. Sea $M_0, M_1, M_2, \ldots$ una sucesión de marcajes. Consideremos la sucesión $\overline{M_0}, \overline{M_1}, \overline{M_2}, \ldots$. Como $\ll$ es un wqo, existen índices $i < j$ tales que $\overline{M_i} \ll \overline{M_j}$. Por el lema 6.2.9 se cumple que $M_i \sqsubseteq_\alpha M_j$, de lo que se sigue la tesis.

□

Es importante tener en cuenta que el propio orden que hemos definido toma en consideración la α-conversión, de manera que por ejemplo, en el caso de una red con sólo dos lugares identificador, los marcajes (α, β) y (β, α) están en el núcleo del orden. En particular, el contraejemplo que vimos antes para demostrar que $\sqsubseteq$ no es un buen cuasi orden ya no es válido, ya que todos aquellos marcajes están en el núcleo de $\sqsubseteq_\alpha$ y por tanto, según el siguiente resultado, son α-equivalentes.

Lema 6.2.11 *La relación de α-equivalencia $\equiv_\alpha$ es el núcleo de $\sqsubseteq_\alpha$.*

Demostración. Supongamos que $M \sqsubseteq_\alpha M'$ y $M' \sqsubseteq_\alpha M$. Entonces existen M_1 y M_2 tales que $M \equiv_\alpha M_1 \sqsubseteq M' \equiv_\alpha M_2 \sqsubseteq M$. Por el lema 6.2.2 sabemos que existen biyecciones $h_1 : S(M) \to S(M_1)$ y $h_2 : S(M') \to S(M_2)$ tales que $M_1 = h_1(M)$ y $M_2 = h_2(M')$. Entonces, tenemos que

$$M(p)(a) = M_1(p)(h_1(a)) \leq M'(p)(h_1(a)) = M_2(p)(h_2(h_1(a))) \leq M(p)((h_2(h_1(a)))$$

Definamos $h = h_2 \circ h_1$, que está bien definida porque $S(M_2) \subseteq S(M)$. Además, como $|S(M)| = |S(M_1)| \leq |S(M')| = |S(M_2)| \leq |S(M)|$ se tiene que $|S(M)| = |S(M_2)|$ y por tanto $S(M) = S(M_2)$. Este hecho nos demuestra que h también es biyectiva. Tenemos $M(p)(a) \leq M(p)(h(a)) \leq M(p)(h^2(a)) \leq \ldots$ para todo $a \in S(M)$. Entonces, como h es una biyección entre conjuntos finitos, existe un n tal que $h^n(a) = a$ y, por tanto, $M(p)(a) = M(p)(h(a)) = \ldots = M(p)(h^n(a))$. En particular, la anterior secuencia de igualdades y desigualdades resulta ser una sucesión de igualdades y, en particular, $M(p)(a) = M'(p)(h_1(a))$ para todo a, es decir, $M' = h_1(M)$. De nuevo por el lema 6.2.2 podemos concluir que $M \equiv_\alpha M'$.

□

Lema 6.2.12 *La relación de disparo de las ν-APNs es monótona respecto a $\sqsubseteq$.*

Demostración. Supongamos que $M_1[t(\sigma)\rangle M_2$ y $M_1 \sqsubseteq M_1'$. De lo primero se sigue, en primer lugar, que $\sigma(F(p,t)) \in M_1(p)$ para todo p por estar ese disparo activado, y que $M_2(p) = M_1(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$ por definición de disparo. De lo segundo se sigue que $M_1(p) \subseteq M_1'(p)$. Entonces, para todo p, $\sigma(F(p,t)) \in M_1(p) \subseteq M_1'(p)$ y, por tanto, la transición está activada en M_1'. Así que se puede disparar t para obtener $M_2'(p) = M_1'(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$. Como $M_1(p) \subseteq M_1'(p)$ se tiene que $M_2(p) = M_1(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\} \subseteq M_1'(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\} = M_2'(p)$ y se sigue la tesis.

□

Proposición 6.2.13 *La relación de disparo de las ν-APNs es monótona respecto a $\sqsubseteq_\alpha$.*

Demostración. Se sigue por el lema anterior y la proposición 6.2.3.

□

El recubrimiento de un marcaje M no es más que la alcanzabilidad del ideal (conjunto cerrado hacia arriba) generado por M, $\mathcal{C}(M) = \{M' \mid M \sqsubseteq_\alpha M'\}$. Definamos la función Pre que computa los predecesores de un conjunto de marcajes.

Definición 6.2.14 *Dada una transición t de una ν-APN, definimos Pre y Pre_t como las funciones de marcajes a conjuntos de marcajes definidas por $Pre(M) = \{M' \mid \exists t(\sigma)\ M[t(\sigma)\rangle M'\}$ y $Pre_t(M) = \{M' \mid \exists \sigma\ M[t(\sigma)\rangle M'\}$, y las extendemos puntualmente a conjuntos de marcajes.*

Para probar el lema 6.2.20 necesitamos otro que afirma que la relación de disparo no sólo es monótona, sino también *inyectiva* cuando trata con marcajes comparables.

Lema 6.2.15 *Si $M_i[t(\sigma)\rangle M_i'$ para $i \in \{1,2\}$ y $M_1' \sqsubseteq_\alpha M_2'$ entonces $M_1 \sqsubseteq_\alpha M_2$.*

Demostración. Como $M_1' \sqsubseteq_\alpha M_2'$, el lema 6.2.2 nos dice que existe una inyección h tal que $h(M_1') \sqsubseteq M_2'$. Veamos que $h(M_1) \sqsubseteq M_2$, lo que implicará que $M_1 \sqsubseteq_\alpha M_2$.

Dados a y p existen dos posibilidades: si $\sigma(F(p,t)) = a$ o $\sigma(F(t,p)) = a$ entonces necesariamente $h(a) = a$; en otro caso, $M_i(p)(a) = M_i'(p)(a)$ para $i = 1,2$. Si $h(a) = b$ con a en el primer caso, entonces $a = b$ y $h(M_1')(p)(a) = M_1'(p)(a) = M_1(p) - \sigma(F(p,t))(a) + \sigma(F(t,p))(a) \leq M_2'(p)(a) = M_2(p)(a) - \sigma(F(p,t))(a) + \sigma(F(t,p))(a)$, lo que implica que $M_1(p)(b) \leq M_2(p)(b)$. Si a cumplía el segundo de los supuestos entonces $h(M_1')(p)(b) = M_1'(p)(a) = M_1(p)(a) \leq M_2'(p)(b) = M_2(p)(b)$ y por tanto $M_1(p)(b) \leq M_2(p)(b)$. De ambos casos podemos deducir que $h(M_1)(p) \subseteq M_2(p)$.

□

Definición 6.2.16 *Dados dos marcajes M_1 y M_2 definimos $M_1 \sqcup M_2$ como el marcaje dado por $(M_1 \sqcup M_2)(p) = M_1(p) \sqcup M_2(p)$, para cada $p \in P$.*

Definición 6.2.17 *Sea t una transición de una ν-APN N, σ un modo de t y M un marcaje de N. Definimos*

$$min_{t(\sigma)}(M) = M \sqcup \sigma(F(t,-)),$$

donde $\sigma(F(t,-))$ es el marcaje de N definido por $\sigma(F(t,-))(p) = \sigma(F(t,p))$.

Lema 6.2.18 *Si $M \equiv_\alpha M'$ entonces $min_{t(\sigma)}(M) \equiv_\alpha min_{t(\sigma)}(M')$.*

Demostración. Llamemos $\bar{M}$ y $\bar{M'}$ a $min_{t,(\sigma)}(M)$ y $min_{t(\sigma)}(M')$ respectivamente. Como $M \equiv_\alpha M'$, por el lema 6.2.2 existe una biyección $h : S(M) \to S(M')$ tal que $M' = h(M)$. Además, se cumple que $S(\bar{M}) = S(M) \cup \sigma(Var(t))$ y $S(\bar{M'}) = S(M') \cup \sigma(Var(t))$. Extendemos h a $S(M) \cup \sigma(Var(t))$ por medio de la identidad. Entonces, $h : S(M) \cup \sigma(Var(t)) \to S(M') \cup \sigma(Var(t))$ es una biyección y de nuevo por el lema 6.2.2 se sigue la tesis.

□

Lema 6.2.19 *Sea M un marcaje de una ν-APN N, t una transición de N y σ un modo de t. Entonces $min_{t(\sigma)}(M)$ es el menor M' tal que $M \sqsubseteq_\alpha M'$ y $Pre_{t(\sigma)}(M') \neq \emptyset$.*

Demostración. Llamemos $\bar{M} = min_{t(\sigma)}(M)$. Trivialmente, $M \sqsubseteq_\alpha \bar{M}$. Veamos que $Pre_{t(\sigma)}(\bar{M}) \neq \emptyset$. Para ello sea M_0 el marcaje definido por $M_0(p) = (M \sqcup \sigma(F(t,-)))(p) - \{\sigma(F(t,p))\} + \{\sigma(F(p,t))\}$ y veamos que $M_0[t(\sigma)\rangle\bar{M}$. En primer lugar, $t(\sigma)$ está activada en M_0, ya que $\sigma(F(p,t)) \in M_0(p)$ para cada lugar p. Entonces la transición se puede disparar en modo σ y $M_0(p) - \sigma(F(p,t)) + \sigma(F(t,p)) = \bar{M}(p)$.

Finalmente, si $M_1[t(\sigma)\rangle M_2$ y $M \sqsubseteq_\alpha M_2$ veamos que $\bar{M} \sqsubseteq_\alpha M_2$, para lo que basta comprobar, según el lema 6.2.18, que si $M \sqsubseteq M_2$ entonces $\bar{M} \sqsubseteq M_2$. Como $M \sqsubseteq M_2$ se cumple que $M(p) \subseteq M_2(p)$, para todo p. Entonces $\bar{M}(p) = M(p) \sqcup \sigma(F(t,p)) \subseteq M_2(p) \sqcup \sigma(F(t,p)) \subseteq M_2(p)$, y la tesis se sigue.

□

Lema 6.2.20

$$min(Pre_{t(\sigma)}(\mathcal{C}(M))) = Pre_{t(\sigma)}(min_{t(\sigma)}(M)).$$

Demostración.

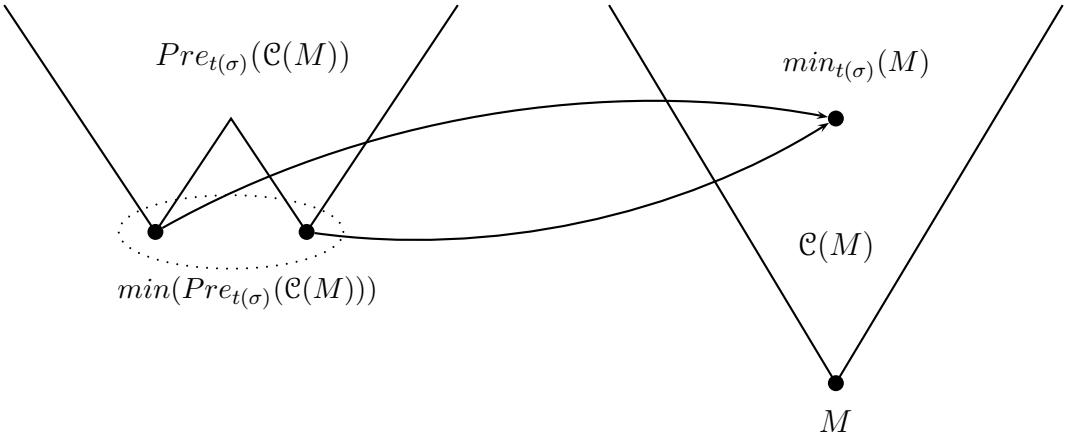

Figura 6.7: Cómputo de $Pre_{t(\sigma)}(\mathcal{C}(M))$

$\subseteq$ Sea $M' \in min(Pre_{t(\sigma)}(\mathcal{C}(M)))$. Entonces existe $\bar{M}$ tal que $M \sqsubseteq_\alpha \bar{M}$ y $M' \in Pre_{t(\sigma)}(\bar{M})$. Como $M \sqsubseteq_\alpha \bar{M}$, por el lema anterior, si escribimos $\bar{M}' = min_{t(\sigma)}(M)$, existe $M'' \in Pre_{t(\sigma)}(\bar{M}')$ y $\bar{M}' \sqsubseteq_\alpha \bar{M}$. Entonces, por el lema 6.2.15, $M'' \sqsubseteq_\alpha M'$. Además, como $M'' \in Pre_{t(\sigma)}(\mathcal{C}(M))$ y M' es minimal en $Pre_{t(\sigma)}(\mathcal{C}(M))$ se sigue por el lema 6.2.11 que $M' \equiv_\alpha M''$ y, por tanto, como $t(\sigma)$ está activada tanto en M' como en M'', $M' \in Pre_{t(\sigma)}(min_{t(\sigma)}(M))$.

$\supseteq$ Sea $M' \in Pre_{t(\sigma)}(min_{t(\sigma)}(M))$. Escribamos $\bar{M} = min_{t(\sigma)}(M)$. Trivialmente, por definición de $min_{t(\sigma)}$, se cumple que $M' \in Pre_{t(\sigma)}(\mathcal{C}(M))$. Veamos que además es minimal. Para ello sea $M'' \in Pre_{t(\sigma)}(\mathcal{C}(M))$ tal que $M'' \sqsubseteq_\alpha M'$ y veamos que $M' \sqsubseteq_\alpha M''$, lo que implicará por el lema 6.2.11 que $M' \equiv_\alpha M''$. Como $M'' \in Pre_{t(\sigma)}(\mathcal{C}(M))$ existe $\bar{M}'$ tal que $M \sqsubseteq_\alpha \bar{M}'$ y $M'' \in Pre_{t(\sigma)}(\bar{M}')$. Por monotonía, como $M'' \sqsubseteq M'$, se sigue que $\bar{M}' \sqsubseteq_\alpha \bar{M}$. Además, como $\bar{M}'$ es minimal, $\bar{M} \equiv_\alpha \bar{M}'$. En particular, por el lema 6.2.15, podemos concluir que $M' \equiv_\alpha M''$.

$\square$

Proposición 6.2.21 *Para cada marcaje M el conjunto $min(Pre_t(\mathcal{C}(M)))$ es computable.*

Demostración. Podemos computar $min(Pre_t(\mathcal{C}(M)))$ de la siguiente manera:

$$min(Pre_t(\mathcal{C}(M))) = min\Big(\bigcup_\sigma Pre_{t(\sigma)}(\mathcal{C}(M)\Big) = min\Big(\bigcup_\sigma min(Pre_{t(\sigma)}(\mathcal{C}(M)))\Big)$$

Por el lema anterior, el último término se reduce a $min\big(\bigcup_\sigma Pre_{t(\sigma)}(min_{t(\sigma)}(M))\big)$. Cada $Pre_{t(\sigma)}(min_{t(\sigma)}(M))\big)$ es computable, de modo que basta ver que sólo hay

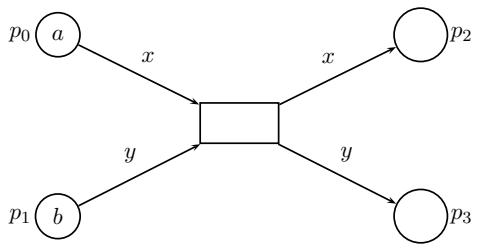

Figura 6.8: Un ejemplo sencillo

un número finito de modos. El único caso que no es inmediato es aquél en el que existe t con $x \in pre(t) \setminus post(t)$, es decir, cuando existe una transición que borra un identificador. En este caso basta considerar que el token borrado puede ser cualquiera de los que aparece en M, o cualquier otro, pero nos basta con escoger uno nuevo arbitrariamente, al estar trabajando módulo α-conversión.

□

Por definición de sistemas bien estructurados [ACJT00], tenemos el siguiente

Corolario 6.2.22 *Las ν-APNs son sistemas bien estructurados.*

Como se probó en [ACJT00], el recubrimiento (o alcanzabilidad de estados de control, como se le llama allí) es decidible para sistemas bien estructurados. La clave del resultado es que toda cadena ascendente infinita de ideales generados por un wqo se estabiliza. Es decir, si $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ es una cadena ascendente de ideales entonces existe un k tal que $I_i = I_k$ para todo $k \geq i$. En particular, podemos construir la siguiente cadena de ideales: si M es el marcaje que queremos cubrir, tomamos $I_0 = \mathcal{C}(M)$ e $I_{n+1} = I_n \cup Pre(I_n)$ para $n \geq 0$.

Como Pre es monótono se cumple que todos los I_i son ideales, y como $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots$ y $\sqsubseteq$ es un wqo, se sigue que existe un índice k tal que $I_l = I_k$ para todo $l \geq k$. Entonces, por construcción, I_0 es alcanzable (es decir, se puede cubrir M) si y sólo si el marcaje inicial $M_0 \in I_k$.

Corolario 6.2.23 *El α-recubrimiento es decidible para ν-APNs y para sistemas MSPN con identificadores abstractos.*

Puede parecer que hemos probado la decidibilidad de una versión débil del problema del recubrimiento, aquélla en la que permitimos un renombramiento arbitrario de los nombres. Por ejemplo, si consideramos la red de la figura 6.8 y preguntamos si se puede cubrir el marcaje M dado por $M(p_0) = M(p_1) = \emptyset$, $M(p_3) = \{b\}$ y $M(p_4) = \{a\}$, el resultado sería afirmativo ya que $M[a/b, b/a]$ es alcanzable en un paso y $M \equiv_\alpha M[a/b, b/a]$. Sin embargo, se puede usar esta versión aparentemente débil para decidir la versión fuerte.

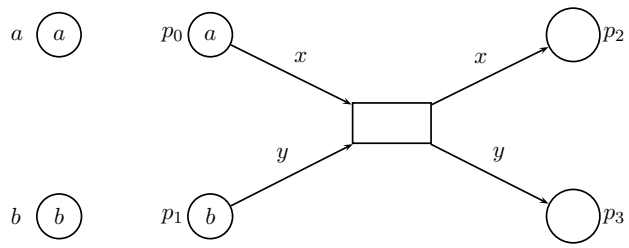

Figura 6.9: De nuevo el ejemplo sencillo

Proposición 6.2.24 *El problema del recubrimiento para las ν-APNs es decidible.*

Demostración. Sean M_0 y M dos marcajes de una ν-APN $N = (P, T, F)$. Queremos decidir si podemos cubrir M desde M_0. Sea $R = S(M_0) \cup S(M)$, que, intuitivamente, representa el conjunto de tokens que queremos monitorizar a lo largo del cómputo, es decir, aquellos tokens que no queremos renombrar. Definimos la ν-APN $N^* = (P \cup R, T, F)$. Para cualquier marcaje M definimos $M^*(p) = M(p)$ si $p \notin R$ y $M^*(r) = \{r\}$ para todo $r \in R$. Por construcción de N^*, en la que los lugares en R están aislados, sus tokens nunca se mueven o se eliminan, de manera que se puede cubrir M desde M_0 en N si y sólo si M^* se puede α-cubrir desde M_0^* en N^*.

□

Consideremos de nuevo el ejemplo de la figura 6.8. De acuerdo con la construcción del resultado anterior, que se puede ver en la figura 6.9, añadimos un lugar para a y otro para b. Cuando ejecutamos esta nueva red, falla el razonamiento que aplicábamos antes. Ahora, en un paso podemos alcanzar M' con $M'(p_0) = M'(p_1) = \emptyset$, $M'(p_3) = M'(a) = a$ y $M'(p_4) = M'(b) = b$. Sin embargo, no es cierto que $M' = M^*[a/b, b/a]$, gracias precisamente a los nuevos lugares añadidos.

Recapitulando, en esta sección hemos probado la decidibilidad del problema de recubrimiento, basándonos en una propiedad que en principio podría parecer más débil, como es el α-recubrimiento, aunque hemos visto que a la postre son equivalentes. En la sección anterior probamos la indecidibilidad de la alcanzabilidad. Podríamos preguntarnos si la propiedad análoga al α-recubrimiento, que podríamos llamar α-alcanzabilidad, es estrictamente más débil que la alcanzabilidad. Un marcaje M es α-alcanzable si existe M' alcanzable tal que $M \equiv_\alpha M'$. Sin embargo, es fácil ver que la decidibilidad de la α-alcanzabilidad implicaría la decidibilidad de la alcanzabilidad, usando exactamente el mismo truco que en la proposición 6.2.24.

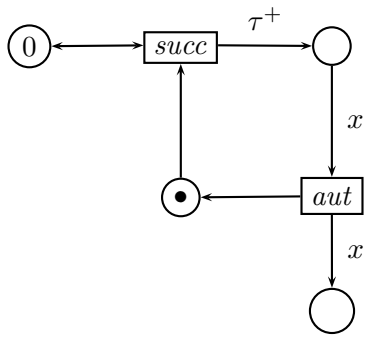

Figura 6.10: Red con un contador

El recubrimiento es decidible para sistemas MSPN con identificadores naturales

En esta sección probaremos que el recubrimiento también es decidible para sistemas MSPN con identificadores naturales, para lo que probaremos dicho resultado para las redes con contadores. En este caso, no podemos aplicar directamente la técnica de buenos cuasiórdenes usada en el apartado anterior, ya que no es inmediato encontrar un orden apropiado que no dependa del marcaje que se pretende cubrir. Sin embargo, podemos probar el resultado de decidibilidad reduciendo el problema al caso de las ν-APNs.

En una primera aproximación ingenua, podríamos intentar simular una red con contadores por la correspondiente ν-APN, simplemente sustituyendo las transiciones *succ* por la correspondiente *new* y eliminando los lugares contador. Sin embargo, las ν-APNs trabajan módulo α-conversion de los tokens identificador, de manera que un marcaje abstracto alcanzable de la red abstracta se correspondería en esta simulación ingenua con todas las permutaciones de los identificadores naturales del sistema original. En otras palabras, no tenemos forma de hacer corresponder cada identificador abstracto con el natural al que representa, y por tanto no tenemos manera de recuperar el orden en el que los naturales se habían generado, que era una información contenida en el marcaje. Por tanto, para que un marcaje se corresponda con una sola permutación necesitamos una manera de monitorizar y memorizar el orden en el que se han generado los nombres abstractos. Se podría pensar que esto no es posible, ya que necesitaríamos recordar ese orden para tiempos arbitrariamente grandes. Afortunadamente, dado el marcaje M que queremos cubrir, basta con recordar ese orden durante un tiempo finito: el correspondiente al mayor identificador que aparece en M.

Sean $N = (P, T, F)$ una n-CsPN y $\overline{M}$ un marcaje de N. Definamos una ν-APN N^* en el que el recubrimiento de $\overline{M}$ en N sea equivalente al de algún marcaje $\overline{M}^*$.

Definimos N^* partiendo de N, eliminando su lugar contador, y cambiando las etiquetas *succ* por *new*. Añadimos para cada i una colección de lugares $d_1^i, \ldots, d_{n_i}^i$, que se usarán para almacenar los n_i primeros nombres creados por el i-ésimo contador. Para ello, crearemos n_i copias de cada transición sucesor, de manera

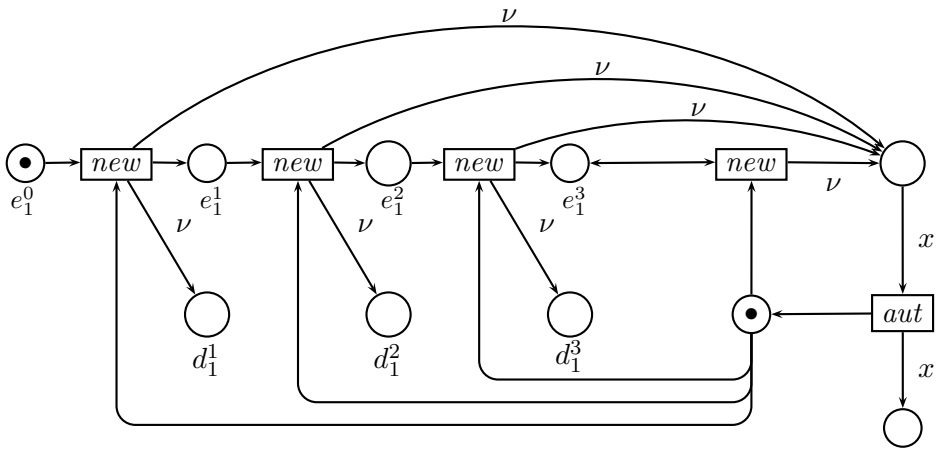

Figura 6.11: Simulación de la red de la figura 6.10 mediante una ν-APN

que contaremos con $n_i + 1$ transiciones *new* para cada i. El j-ésimo disparo de *succ* para $j \leq n_i$ lo simularemos mediante el disparo de la j-ésima copia de *new*, que almacenará el nombre creado en d_i^j, mientras que los sucesivos disparos de *succ* serán simulados por la misma transición *new* (véase la red con un contador en la figura 6.10 y su simulación por medio de una ν-APN en la figura 6.11).

Veámoslo formalmente. Supongamos que el conjunto de contadores de N es $P_c = \{c_1, \ldots, c_n\}$ y su correspondiente conjunto de transiciones sucesor es $T_c = \{t_1, \ldots, t_n\}$. Para cada i sea $n_i = max\{n \mid p \in P,\ (i,n) \in \overline{M}(p)\}$. Entonces definimos $N^* = (P^*, T^*, F^*)$, donde

- $P^* = P \setminus P_c \cup \{d_i^j \mid i = 1, \ldots, n,\ j = 1, \ldots, n_i\}$
  $\cup \{e_i^j \mid i = 1, \ldots, n,\ j = 0, \ldots, n_i\}$,
- $T^* = T \cup \{t_i^j \mid t \in T_c,\ i = 1, \ldots, n,\ j = 1, \ldots, n_i\}$,
- F^* es la función parcial con menor dominio tal que:
  - $F^*(p,t) = F(p,t)$ para todo $p \in P$ y $t \in T$,
  - $F^*(t,p) = \begin{cases} \nu \text{ si } F(t,p) = \tau^+ \\ F(t,p) \text{ en otro caso,} \end{cases}$
  - $F^*(p, t_i^j) = F(p, t_i)$ para todo t_i y $j = 1, \ldots, n_i$,
  - $F^*(t_i^j, p) = \begin{cases} \nu \text{ si } F(t_i,p) = \tau^+ \\ F(t_i,p) \text{ en otro caso,} \end{cases}$
  - $F^*(e_i^j, t_i^{j+1}) = \varepsilon$ para todo $i = 1, \ldots, n$ y $j = 0, \ldots, n_i$, identificando t_i^j con t_i cuando $j > n_i$,
  - $F^*(t_i^j, e_i) = \varepsilon$ para todo $i = 1, \ldots, n$ y $j = 1, \ldots, n_i$,
  - $F^*(t_i, e_i^{n_i}) = \varepsilon$ para $i = 1, \ldots, n$,

- $F^*(t_i^j, d_i^j) = \nu$ para $i = 1, \ldots, n$ y $j = 1, \ldots, n_i$.

Además, dado el marcaje M de N definimos el marcaje M^* de N^* de la siguiente manera. Si $M(c_i) = (i, k_i)$ entonces elegimos un conjunto arbitrario de identificadores $\{\eta_i^j \mid i = 1, \ldots, n,\ j = 1, \ldots, k_i\}$ y tomamos

- $M^*(p)(\eta_i^j) = M(p)(i, j)$ si $p \in P$,
- $M^*(d_i^j) = \begin{cases} \{\eta_i^j\} \text{ si } j \leq min\{n_i, k_i\} \\ \emptyset \text{ en otro caso} \end{cases}$
- $M^*(e_i^j) = \begin{cases} 1 \text{ si } j = min\{n_i, k_i\} \\ 0 \text{ en otro caso} \end{cases}$

Nótese que, estrictamente hablando, $(\ )^*$ no es una función, sino una relación, ya que su definición en principio depende de los identificadores elegidos. Sin embargo, todos los marcajes de la ν-APN obtenidos son iguales salvo renombramiento de sus identificadores y, por lo tanto, α-equivalentes.

Con las notaciones anteriores podemos probar el siguiente resultado.

Proposición 6.2.25 *Si M_1 y M_2 son marcajes de N entonces*

$$M_1 \to M_2 \Leftrightarrow M_1^* \to M_2^*$$

Demostración. Veamos primero que si $M_1[t(\sigma)\rangle M_2$ entonces $M_1^* \to M_2^*$. Si $t \notin T_c$ entonces claramente $M_1^*[t(\sigma)\rangle M_2^*$. Veamos qué ocurre si $t = t_l \in T_c$. Si $\{\eta_i^j \mid i = 1, \ldots, n,\ j = 1, \ldots, k_i\}$ son los identificadores usados para construir M_1^* definamos $\sigma'(x) = \begin{cases} \eta_i^j \text{ si } x \neq \nu \text{ y } \sigma(x) = (i, j) \\ \eta_l^{k_l+1} \text{ nuevo si } x = \nu \end{cases}$ Veamos que $t_l^{k_l+1}(\sigma')$ se puede disparar en M_1^* para algún j. Las precondiciones de los t_l^j son las de t_l y e_l^{j-1}. Dado $p \in {}^\bullet t$, si $\sigma(F(p, t)) = (i, j)$ se tiene que $\sigma'(F(p, t)) \in M_1^*(p) \Leftrightarrow \eta_i^j \in M_1^*(p) \Leftrightarrow (i, j) \in M_1(p) \Leftrightarrow \sigma(F(p, t)) \in M_1(p)$. Además, por definición de M_1^*, $M_1^*(e_l^{k_l}) = 1$, así que la transición $t_l^{k_l+1}$ se puede disparar, dando lugar a un marcaje $\overline{M}'$. Veamos que si tomamos los identificadores $\{\eta_i^j \mid i = 1, \ldots, n,\ j = 1, \ldots, k_i\} \cup \{\sigma(\nu)\}$ entonces $\overline{M}' = M_2^*$. Para los lugares de P se cumple que $\{\sigma'(F(p, t))\}(\eta_i^j) = \{\sigma(F(p, t))\}(i, j)$ para todo i y para todo $j = 1, \ldots, k_i'$, donde $k_l' = k_l + 1$ y $k_i' = k_i$ para $i \neq l$ (ambos términos se reducen a 1 cuando $\sigma(F(p, t)) = (i, j)$ y a 0 en otro caso). Análogamente se tiene que $\{\sigma'(F(t, p))\}(\eta_i^j) = \{\sigma(F(t, p))\}(i, j)$. Entonces $\overline{M}'(p)(\eta_i^j) = M_1^*(p)(\eta_i^j) - \{\sigma'(F(p, t))\}(\eta_i^j) + \{\sigma'(F(p, t))\}(\eta_i^j) = M_1(p)(i, j) - \{\sigma(F(p, t))\}(i, j) + \{\sigma(F(t, p))\}(i, j) = M_2(p)(i, j) = M_2^*(p)(\eta_i^j)$.

Veamos que $\overline{M}'$ también coincide con M_2^* en los otros lugares. Si $k_l \geq n_l$ entonces $min\{k_l, n_l\} = n_l$, es decir, $t_l^{k_l+1} = t_l$ y son iguales porque $\overline{M}'(e_l^{n_l}) = M_1^*(e_l^{n_l}) = M_2^*(e_l^{n_l})$. En el caso en el que $k_l < n_l$ también coinciden porque la transición $t_l^{k_l+1}$ mueve un token de $e_l^{k_l}$ a $e_l^{k_l+1}$ y añade a $d_l^{k_l+1}$ el identificador $\sigma(\nu) = \eta_l^{k_l+1}$.

Para el recíproco veamos que si $M_1^*[u(\sigma)\rangle M_2^*$ entonces $M_1[v(\sigma')\rangle M_2$, donde $u = v$ si $u \in T$ y $v = t_i$ si $u = t_i^j$. En cualquiera de los dos casos es inmediato pues el conjunto de precondiciones/postcondiciones de t_i^j es un subconjunto del conjunto de precondiciones/postcondiciones de t_i.

□

Corolario 6.2.26 *El recubrimiento es decidible para las redes con contador y los sistemas MSPN con identificadores naturales.*

Demostración. Sea $\overline{M}$ el marcaje cuyo recubrimiento queremos decidir en N y sea N^* la ν-APN de la proposición anterior construida a partir de $\overline{M}$. Claramente, todos los marcajes alcanzables en la construcción anterior son de la forma M^* para algún M. Entonces, basta ver que para todo M se tiene que $\overline{M} \sqsubseteq M \Leftrightarrow \overline{M}^* \sqsubseteq_\alpha M^*$. Como para todo $(i,j) \in S(\overline{M})$ se cumple que $j \leq n_i$ entonces basta tener en cuenta que $\overline{M}^*(p)(\eta_i^j) = \overline{M}(p)(i,j)$ y $M(p)(i,j) = M^*(p)(\eta_i^j)$ suponiendo que $M(d_i^j) = \overline{M}(d_i^j)$ (si no se cumple basta tomar otros representantes de los (i,j), que darían lugar a M' con $M'^* \equiv_\alpha M^*$).

Teniendo en cuenta lo anterior, concluimos que $M_0 \rightarrow^* M \sqsubseteq \overline{M} \Leftrightarrow M_0^* \rightarrow^* M^*{}_\alpha \sqsupseteq \overline{M}^*$ y la tesis se sigue.

□

Nótese que la anterior construcción no es una simulación por el simple hecho de que la función $(\ )^*$ no es inyectiva. En efecto, si consideramos una red con un sólo contador, $n = 1$, un marcaje suyo M_2 con $2 \in S(M)$ y M_3 el resultado de sustituir en M_2 todos los tokens 2 por el token 3 entonces se tiene que $M_2^* = M_3^*$. Sin embargo, si nos restringimos a los marcajes en los que no aparecen tokens (i,j) con $j > n_i$ el resultado anterior muestra una simulación paso a paso de redes con contadores con ν-APNs.

6.3. Resultados de acotación

En los dos apartados anteriores hemos determinado la medida en la que nuestros sistemas, modelados como MSPNs en sus distintas versiones, pueden ser analizados, a pesar de poder representar sistemas con un número infinito de estados. Sin embargo, cuando nos enfrentamos a una pregunta sobre un sistema concreto, una pregunta previa debería ser si el sistema en cuestión es de hecho finito, a pesar de estar especificado en un modelo con posibilidad de describir sistemas con infinitos estados. La pregunta previa a la que nos referimos no es más que la propiedad de acotación.

A diferencia de las redes de Petri ordinarias, con una sola dimensión de infinitud, nuestras ν-APNs presentan dos dimensiones de infinitud, una en el número de identificadores distintos y otra en la cantidad de veces que esos identificadores pueden aparecer. Consecuentemente, aparecen diversas nociones de acotación,

tanto en una dimensión, como en la otra, como en ambas conjuntamente (que no simultáneamente).

Definición 6.3.1 *Sea $N = (P, T, F)$ una ν-APN.*

- *Diremos que N es* acotada *si existe $n \in \mathbb{N}$ tal que para todo M alcanzable y para todo $p \in P, |M(p)| \leq n$.*
- *Diremos que N es* acotada en anchura *si existe $n \in \mathbb{N}$ tal que para todo M alcanzable y para todo $p \in P, |S(M(p))| \leq n$.*
- *Diremos que N es* acotada en profundidad *si existe $n \in \mathbb{N}$ tal que para todo M alcanzable, para todo $p \in P$ y para todo $a \in Id, M(p)(a) \leq n$.*
- *Diremos que N es* históricamente acotada *si existe $n \in \mathbb{N}$ tal que para toda traza $M_0[t_1(\sigma_1)\rangle \ldots [t_n(\sigma_n)\rangle M_n$, $|\bigcup_{i=0}^{n} S(M_i)| \leq n$.*

Proposición 6.3.2 *Dada una ν-APN N, se cumplen las siguiente propiedades:*

1. *N es acotada si y sólo si es acotada en anchura y profundidad.*
2. *N es acotada en anchura si y sólo si existe $n \in \mathbb{N}$ tal que para todo M alcanzable, $|S(M)| \leq n$.*
3. *N es acotada en profundidad si y sólo si existe $n \in \mathbb{N}$ tal que para todo M alcanzable y para todo $a \in Id$, $\sum_{p \in P} M(p)(a) \leq n$.*
4. *Si N es históricamente acotada, entonces es acotada en anchura.*

Demostración.

1. Para este punto basta tener en cuenta que $|M(p)| = |\sum_{a \in S(M(p))} M(p)(a)| \leq |\sum_{a \in S(M(p))} max\{M(p)(a) \mid a \in S(M(p))\}| = |S(M(p))| \cdot max\{M(p)(a) \mid a \in S(M(p))\}$. Supongamos en primer lugar que existe $n \in \mathbb{N}$ tal que $|M(p)| \leq n$. Entonces $|\sum_{a \in S(M(p))} M(p)(a)| \leq n$ y como por definición $S(M(p)) = \{a \in Id \mid M(p)(a) > 0\}$ tenemos que $|S(M(p))| \leq n$ y para todo $a \in S(M(p))$, $M(p)(a) \leq n$. Recíprocamente, supongamos que existen n y m tales que $|S(M(p))| \leq n$ y $M(p)(a) \leq m$. Del segundo hecho se sigue que $max\{M(p)(a) \mid a \in S(M(p))\} \leq m$. Entonces, por la observación anterior, $|M(p)| \leq n \cdot m$ y la tesis se sigue.
2. Basta tener en cuenta que $S(M) = \bigcup_{p \in P} S(M(p))$ y por tanto, $|S(M)| \leq \sum_{p \in P} |S(M(p))| \leq |P| \cdot max\{|S(M(p)| \mid p \in P\}$. Por tanto, si existe n tal

que para todo p, $|S(M(p))| \leq n$ entonces $max\{|S(M(p))| \mid p \in P\} \leq n$, lo que implica que $|S(M)| \leq |P| \cdot n$. Recíprocamente, si $|S(M)| \leq n$ entonces claramente para todo p, $|S(M(p))| \leq n$.

3. Basta tener en cuenta que $\sum_{p \in P} M(p)(a) \leq \sum_{p \in P} max\{M(p)(a) \mid a \in Id\} = |P| \cdot max\{M(p)(a) \mid a \in Id\}$. En primer lugar, si para todo p y a, $M(p)(a) \leq n$, entonces $max\{M(p)(a) \mid a \in Id\} \leq n$, y por la observación anterior $\sum_{p \in P} M(p)(a) \leq |P| \cdot n$. Recíprocamente, si $\sum_{p \in P} M(p)(a) \leq n$ entonces cada uno de los sumandos ha de ser menor o igual que n y la tesis se sigue.

4. Por el punto anterior es inmediato, ya que dada una traza

$$M_0[t_1(\sigma_1)\rangle \ldots [t_n(\sigma_n)\rangle M_n$$

para todo $i \in \{0, \ldots, n\}$ es trivial que $S(M_i) \subseteq \bigcup_{i=0}^{n} S(M_i)$.

□

En realidad, prácticamente ya hemos visto que la acotación es decidible para las ν-APNs. Esto es así porque hemos demostrado que las ν-APNs son sistemas bien estructurados para el orden natural. Pues bien, con ese orden son, de hecho, un sistema *estrictamente* bien estructurado.

Lema 6.3.3 *$M \sqsubset_\alpha M'$ si y solo si existe M'' tal que $M \sqsubset M'' \equiv_\alpha M'$.*

Demostración. Si $M \sqsubset_\alpha M'$ entonces $M \sqsubseteq_\alpha M'$ y $M' \not\sqsubset_\alpha M$, lo que, por el lema 6.2.11, implica que $M \not\equiv_\alpha M'$. Por definición de $\sqsubseteq_\alpha$ existe M'' tal que $M \sqsubseteq M'' \equiv_\alpha M'$. Por último, veamos que $M \neq M''$, lo que implicará la tesis. En efecto, si suponemos $M = M''$ entonces, como $M'' \equiv_\alpha M$, tenemos que $M \equiv_\alpha M'$, y llegamos a una contradicción.

Recíprocamente, $M \sqsubset M'' \equiv_\alpha M'$ implica que $M \sqsubseteq_\alpha M'$. Además, si $M \equiv_\alpha M'$ entonces tendríamos $M \equiv_\alpha M''$, lo que es una contradicción.

□

Lema 6.3.4 *La relación de disparo de las ν-APNs es estrictamente monótona respecto a $\sqsubseteq$.*

Demostración. Por el lema anterior, podemos repetir la demostración del lema 6.2.12 sustituyendo los $\sqsubseteq$ por $\sqsubset$ y sustituyendo los $\subseteq$ por $\subset$ para al menos un $p \in P$.

□

M_0 es la raíz del árbol, inicialmente no marcado,
abortar:=`falso`
while existen nodos no marcados en el árbol y no(abortar) **do**
  Sea M no marcado
  Marcar M
  Para todo $t(\sigma)$ activado y M' tal que $M[t(\sigma\rangle M'$:
  Para todo M'' en el camino de la raíz a M tal que $M'' \sqsubseteq_\alpha M'$
  if $M'' \not\equiv_\alpha M'$ **then**
    abortar:=`cierto`
  end if
  añadir el arco $(M, t(\sigma), M')$ al árbol
end while
devolver abortar

Figura 6.12: Algoritmo de decisión de la acotación

Lema 6.3.5 *Las ν-APNs con su orden natural son un sistema estrictamente bien estructurado.*

Demostración. Por el lema anterior y la proposición 6.2.3.

□

Lema 6.3.6 *Dada una ν-APN con un marcaje inicial, el conjunto de marcajes alcanzables es finito si y sólo si la red es acotada según la definición 6.3.1.*

Demostración. Veamos que el conjunto de marcajes alcanzables *Reach* es finito si y sólo si N es acotada. Si *Reach* es finito podemos definir $s = max\{|S(M)| \mid M \in Reach\}$ y $k = max\{M(p)(a) \mid M \in Reach,\ p \in P,\ a \in S(M)\}$. Entonces, para cada M alcanzable $|M(p)| = |\sum_{a \in S(M(p))} M(p)(a)| \leq k \cdot s$ y la red es acotada. El recíproco también es claro, ya que si N no es acotada entonces para cada n existe M_n tal que $|M_n(p)| > n$ para todo p.

□

Así que podemos usar el resultado de [FS01] acerca de los sistemas estrictamente bien estructurados para decidir nuestra noción de acotación.

Corolario 6.3.7 *La acotación es decidible para las ν-APNs.*

Para aplicar el anterior resultado basta construir la parte acotada del árbol de recubrimiento, como se muestra en la figura 6.12.

Es fácil ver que el algoritmo de la figura 6.12 termina. Supongamos que no lo hace. En ese caso existiría por el lema de König una cadena infinita de marcajes

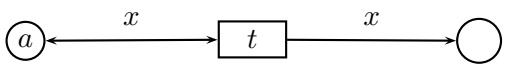

Figura 6.13: ν-APN acotada en anchura pero no en profundidad

$M_0 \to M_1 \to M_2 \to \ldots$ tales que para todo par de índices $i < j$, $M_i \not\sqsubseteq_\alpha M_j$, lo que no puede ocurrir por ser ese orden un buen cuasi orden.

Por tanto, con el resultado anterior y la proposición 6.3.2, sabemos que si una ν-APN está acotada entonces está acotada tanto en profundidad como en anchura. Sin embargo, si no está acotada aún podría estar o bien acotada en anchura (ver figura 6.13) o bien acotada en profundidad (ver figura 6.14), aunque no acotada simultáneamente en anchura y en profundidad o, de nuevo por la proposición 6.3.2, estaría acotada.

Veamos que la acotación en anchura también es decidible para las ν-APNs. Para ello vamos a reutilizar la demostración para redes P/T, basada en la construcción del árbol de recubrimiento o de Karp-Miller.

Empecemos viendo una condición necesaria y suficiente para que una red no sea acotada en anchura.

Proposición 6.3.8 *Sea N una ν-APN con marcaje inicial M_0. N no es acotada en anchura si y sólo si existen dos marcajes M_1 y M_2 tales que:*

- $M_0 \to^* M_1 \to^* M_2$,
- $M_1 \sqsubseteq_\alpha M_2$,
- $|S(M_1)| < |S(M_2)|$.

Demostración. Vamos a trabajar con la condición equivalente a la acotación en anchura de la proposición 6.3.2.

$\Leftarrow$ Por monotonía de las ν-APNs con respecto a $\sqsubseteq_\alpha$, si $t_1, \ldots, t_n$ es la secuencia de transiciones que llevan de M_1 a M_2, entonces dicha secuencia de transiciones está de nuevo activada en M_2 y, por tanto, existe M_3 tal que $M_2 \to^* M_3$, $M_2 \sqsubseteq_\alpha M_3$ y $|S(M_2)| < |S(M_3)|$. Repitiendo el proceso obtenemos una secuencia infinita $M_0 \to^* M_1 \to^* M_2 \to^* M_3 \to^* \ldots$ tal que $|S(M_i)| < |S(M_{i+1})|$ y, por tanto, N no está acotada en anchura.

$\Rightarrow$ Si N no está acotada en anchura entonces existe una traza $M_0 \to M_1 \to M_1 \to \ldots$ tal que para todo $k \in \mathbb{N}$ existe un índice i con $|S(M_i)| > k$. Entonces existe una subsecuencia de la anterior, que denotamos igual que ella, de manera que $M_0 \to^* M_1 \to^* M_2 \to^* \ldots$ y $|S(M_i)| < |S(M_{i+1})|$ para todo $i \in \mathbb{N}$. Pues bien, por ser $\sqsubseteq_\alpha$ un buen cuasi orden, existen dos índices $i < j$ tales que $M_i \sqsubseteq_\alpha M_j$, y por tanto M_i y M_j cumplen los tres requisitos buscados.

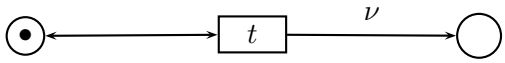

Figura 6.14: ν-APN acotada en profundidad pero no en anchura

□

Por tanto, para probar que la red no es acotada en anchura nos basta con encontrar unos testigos como los que establece el resultado anterior. Ahora veamos que es factible encontrar tales testigos, mediante la construcción de una versión modificada del árbol de Karp-Miller.

Llamaremos ω-marcajes a una extensión de los marcajes ordinarios, en los que permitimos que existan identificadores que se encuentren infinitas veces en uno o varios lugares. Sin embargo, sólo permitiremos que exista un número finito de identificadores.

Definición 6.3.9 *Llamaremos* ω-marcaje *de una ν-APN N con conjunto de lugares P a cualquier elemento de $\mathcal{MS}(\mathcal{MS}_\infty(P))$, donde $A \in \mathcal{MS}_\infty(P)$ es cualquier función $A : P \to \mathbb{N} \cup \{\infty\}$.*

Así que los ω-marcajes de nuestras ν-APNs representan clases de equivalencias de marcajes, módulo $\equiv_\alpha$, en los que algunos identificadores pueden aparecer un número infinito de veces en algún lugar. En ese caso, escribiremos $\infty(p)$.

Las definiciones de orden entre ω-marcajes, activación de una transición en un modo, disparo de una transición en un modo y ω-marcaje alcanzado por el disparo de una transición en un modo son análogas a las que teníamos, teniendo en cuenta la extensión a $\mathbb{N} \cup \{\infty\}$ de la aritmética habitual con $n < \infty$, $\infty + n = \infty$, $\infty + \infty = \infty$ y $\infty - n = \infty$ (no usaremos nunca $n - \infty$ o $\infty - \infty$), para todo $n \in \mathbb{N}$.

No está claro cómo sería el procedimiento de construcción del árbol de Karp-Miller de una ν-APN con el orden $\sqsubseteq_\alpha$, debido a las distintas dimensiones de infinitud que se dan en ellas, que no son ortogonales. Más concretamente, la representación homogénea de infinitos tokens distintos que representan los ω-marcajes de una red P/T, en principio no se puede trasladar a las ν-APNs. Por ejemplo, podría haber un número infinito de identificadores, unos acotados en profundidad y otros no, por lo que no se les podría tratar uniformemente, al menos no de forma inmediata. Sin embargo, al estar buscando la acotación en anchura de la red, podemos parar en el momento en el que encontramos un par de marcajes M_1, alcanzable desde M_0, y M_2, alcanzable desde M_1, tales que $|S(M_1)| < |S(M_2)|$ y $M_1 \sqsubseteq_\alpha M_2$. Sin embargo, cuando encontramos M_1 y M_2, este último alcanzable desde el primero pero tal que $|S(M_1)| = |S(M_2)|$ y $M_1 \sqsubset_\alpha M_2$ sabemos que ha de existir un identificador a y un lugar p tales que $M_1(p)(a) < M_2(p)(a)$ y, por tanto,

```
M_0 es la raíz del árbol, inicialmente no marcado
abortar:=falso
while existen nodos no marcados en el árbol y no(abortar) do
  Sea M no marcado
  Marcar M
  Para cualquier par t(σ) que esté activado y M' tal que M[t(σ⟩M':
  Para todo M'' en el camino de la raíz a M tal que M'' ⊑_α M'
  if M'' ≢_α M' then
    if |S(M'')| < |S(M')| then
      abortar:=cierto
    else
      Sean M'' = {A_1, ..., A_n} y M' = {B_1, ..., B_n} con A_i ⊆ B_i
      para todo i y p tales que A_i(p) < B_i(p) hacer B_i(p) = ∞
    end if
  end if
  añadir el arco (M, t(σ), M')
end while
devolver abortar
```

Figura 6.15: Algoritmo de decisión de la acotación en anchura

la secuencia de acciones que llevan de M_1 a M_2 se puede repetir por monotonía desde M_2, para obtener un M_3 que de nuevo cumple que $M_2(p)(a) < M_3(p)(a)$ y $|S(M_2)| = |S(M_3)|$. Entonces, en vez de añadir M_2 al árbol, añadiremos una modificación de este en la que el identificador a se encuentre un número infinito de veces en p. Sin embargo, este hecho no nos dice nada de la acotación en anchura de la red, por lo que es necesario proseguir con la construcción del árbol. Este algoritmo aparece detallado en la figura 6.15.

Llamaremos ω-secuencia a cualquier rama del árbol construida por el algoritmo anterior, y diremos que un ω-marcaje es alcanzable si aparece en alguna ω-secuencia.

Proposición 6.3.10 *El algoritmo de la figura 6.15 termina.*

Demostración. Si no termina entonces el árbol que construye es infinito, y por ser finitario y el lema de König, tiene una rama infinita. Sea $M_0 \to M_1 \to M_2 \to M_3 \ldots$ esa rama. En primer lugar, como para cualquier marcaje M el conjunto de marcajes estrictamente menores que él según $\sqsubseteq_\alpha$ es finito, podemos extraer de esta traza una subtraza de manera que para todo par $i < j$ se cumple que $M_j \not\sqsubseteq_\alpha M_i$. Por claridad en las notaciones denotaremos a la nueva secuencia igual que a la original. Como por construcción del árbol, en ninguna rama aparecen marcajes α-equivalentes, se tiene que para todo par $i < j$ o bien $M_i \sqsubset_\alpha M_j$ o bien M_i y M_j son incomparables. Como $\sqsubseteq_\alpha$ es un buen cuasi orden, ninguna cadena infinita

puede contener una cantidad infinita de marcajes incomparables dos a dos, por lo que tiene que existir una subcadena de marcajes estrictamente creciente, es decir, $M_{i_0}, M_{i_1}, M_{i_2}, \ldots$ tal que $M_{i_j} \sqsubset_\alpha M_{i_{j+1}}$. Por construcción del árbol, se cumple que existe un m tal que para cualquier j, $|S(M_{i_j})| = m$ y en $M_{i_{j+1}}$ hay, al menos, un identificador nuevo que aparece infinitas veces en algún lugar, es decir, existe a y p tal que $M_{i_j}(p)(a) \neq \infty = M_{i_{j+1}}(p)(a)$. Sin embargo, sólo puede haber m identificadores, y sólo hay $|P| < \infty$ lugares, de modo que a lo sumo se podría aumentar el número de "infinitos" en el marcaje $m \cdot |P|$ veces, lo que nos lleva a una contradicción.

□

Veamos que, además de terminar siempre, el algoritmo es correcto. Para ello vamos a dar las relaciones habituales entre ω-marcajes y marcajes ordinarios.

Proposición 6.3.11 *Sea N una ν-APN y $\overline{M}$ un ω-marcaje alcanzable. Para cada $n \in \mathbb{N}$ existe un marcaje alcanzable M tal que $S(M) = S(\overline{M})$ y para todo lugar $p \in P$ e identificador $a \in S(M)$:*

- *Si $\overline{M}(p)(a) \neq \infty$ entonces $M(p)(a) = \overline{M}(p)(a)$;*
- *Si $\overline{M}(p)(a) = \infty$ entonces $M(p)(a) \geq n$.*

Demostración. Basta adaptar rutinariamente la demostración del resultado análogo en [DR98, página 144].

□

Proposición 6.3.12 *Sea M un marcaje alcanzable de una ν-APN N. Entonces existe un ω-marcaje alcanzable $\overline{M}$ tal que $S(M) \subseteq S(\overline{M})$ y $M(p)(a) = \overline{M}(p)(a)$ para todo lugar p e identificador a tales que $\overline{M}(p)(a) \neq \infty$. Además, si no existe M' en la traza que lleva a M tal que $M' \sqsubseteq M$, entonces $\overline{M} = M$.*

Demostración. Procedemos por inducción sobre la longitud de la traza que lleva a M. Si esa traza es de longitud 0, es decir, si M es el marcaje inicial, entonces trivialmente podemos tomar $\overline{M} = M$.

Supongamos que M' es un marcaje alcanzable tal que $M' \stackrel{t(\sigma)}{\rightarrow} M$. Por hipótesis de inducción existe $\overline{M}'$ alcanzable tal que $S(M') \subseteq S(\overline{M}')$ y $M'(p)(a) = \overline{M}'(p)(a)$ para todos a y p tales que $\overline{M}'(p)(a) < \infty$. Sea $\overline{M}''$ tal que $\overline{M}' \stackrel{t(\sigma)}{\rightarrow} \overline{M}''$ y $\overline{M}$ el marcaje obtenido de $\overline{M}''$ aplicando el algoritmo, es decir, $\overline{M}(p)(a) = \overline{M}''(p)(a)$ si no existe otro marcaje anterior a $\overline{M}'$ tal que $\overline{M}' \sqsubseteq M''$ y $\overline{M}'(p)(a) < \overline{M}''(p)(a)$ y $\overline{M}(p)(a) = \infty$, en otro caso. Veamos que $\overline{M}$ cumple la tesis.

Por definición de alcanzabilidad de ω-marcajes, $\overline{M}$ es alcanzable. Además, si $\overline{M}(p)(a) < \infty$ entonces necesariamente $\overline{M}'(p)(a) < \infty$ y $\overline{M}''(p)(a) = \overline{M}(p)(a)$. Entonces, $\overline{M}(p)(a) = \overline{M}'(p)(a) - \{\sigma(F(p,t))\}(a) + \{\sigma(F(t,p))\}(a) = M(p)(a)$, ya que por hipótesis de inducción $\overline{M}'(p)(a) = M'(p)(a)$.

Veamos finalmente que $S(M) \subseteq S(\overline{M})$. Para ello basta tener en cuenta los hechos siguientes: En primer lugar, la hipótesis de inducción nos dice que $S(M') \subseteq S(\overline{M}')$. Además, el disparo de $t(\sigma)$ crea los mismos identificadores tanto en M como en $\overline{M}$. Sin embargo, por el párrafo anterior, sabemos que $t(\sigma)$ destruye al dispararse más identificadores en M que en $\overline{M}$. De todo lo anterior se deduce que $S(M) \subseteq S(\overline{M})$. La observación final es cierta trivialmente por construcción del algoritmo.

□

Proposición 6.3.13 *N es acotada en anchura si y sólo si el algoritmo de la figura 6.15 devuelve* `falso`.

Demostración. Si N es no acotada entonces según la proposición 6.3.8 existen M_1 y M_2 tales que $M_0 \rightarrow^* M_1 \rightarrow^* M_2$ con $M_1 \sqsubseteq M_2$ y $S(M_1) \subset S(M_2)$, y tal que M_1 es incomparable a todos los anteriores marcajes de la traza (tomando representantes adecuados de la clase módulo $\equiv_\alpha$). Según la observación del lema anterior, M_1 es alcanzable como ω-marcaje. Además, de nuevo por el lema anterior, existe $\overline{M}_2$ tal que $M_1 \rightarrow^* \overline{M}_2$ con $S(M_2) \subseteq S(\overline{M}_2)$ y de manera que M_2 y $\overline{M}_2$ coinciden cuando este último no vale ∞.

En primer lugar, $\overline{M}_2$ es un ω-marcaje alcanzable. Además, $S(M_1) \subset S(M_2) \subseteq S(\overline{M}_2)$ y por tanto, $S(M_1) \subset S(\overline{M}_2)$. Por último, veamos que $M_1 \sqsubseteq \overline{M}_2$. Si $\overline{M}_2(p)(a) = \infty$ entonces claramente $M_1(p)(a) < \overline{M}_2(p)(a)$. Si, por el contrario, $\overline{M}_2(p)(a) < \infty$ entonces $M_1(p)(a) \leq M_2(p)(a) = \overline{M}_2(p)(a)$. De los dos hechos anteriores se deduce que $M_1 \sqsubseteq \overline{M}_2$.

Pues bien, cuando el algoritmo de la figura 6.15 encuentra un ω-marcaje como $\overline{M}_2$, este devuelve `cierto`, y la tesis se sigue.

Supongamos ahora el algoritmo devuelve `cierto`. Si ello ocurre es porque el algoritmo ha encontrado un marcaje M_1 y un ω-marcaje $\overline{M}_2$, tales que $M_0 \rightarrow^* M_1 \rightarrow^* \overline{M}_2$ con $M_1 \sqsubseteq \overline{M}_2$ y $S(M_1) \subset S(\overline{M}_2)$ (tomando representantes adecuados). Entonces, por la proposición 6.3.11 existe M_2 tal que $S(M_2) = S(\overline{M}_2)$ y M_2 coincide con $\overline{M}_2$ donde este último no vale ∞, y supera cualquier valor prefijado en el resto. Consideremos un valor suficientemente grande para que $M_1 \sqsubseteq M_2$. Además, $S(M_1) \subset S(\overline{M}_2) = S(M_2)$, así que se cumplen las condiciones de la proposición 6.3.8 y la red es no acotada.

□

Corolario 6.3.14 *El problema de la acotación en anchura de las ν-APNs es decidible.*

Hemos visto que, en general, el problema de alcanzabilidad es indecidible para las ν-APNs. Evidentemente, en el caso en el que la red sea acotada el espacio de estados que genera es finito, y por tanto la alcanzabilidad para la clase de ν-APNs acotadas es decidible. Pues bien, de hecho esto también es cierto para

$$s_1 = \{\{p_1\}\}$$
$$\downarrow t$$
$$s_2 = \{\{p_1, \infty(p_2)\}\}$$
$$t$$

Figura 6.16: Árbol de Karp-Miller de la ν-APN de la figura 6.13

la clase de ν-APNs acotadas en anchura, aunque puedan generar un espacio de infinitos estados. Ello se debe a que es posible simular toda ν-APN por medio de una APN, es decir, una red que no crea identificadores (en la que, por simplicidad, permitiremos tokens ordinarios). Esta simulación se basa en añadirle a la red un control y un repositorio de identificadores disponibles para ser utilizados.

Sea $N = (P, T, F, M_0)$ una ν-APN acotada en anchura. En primer lugar, vamos a suponer que N no tiene transiciones que pueden crear un identificador y destruir otro, es decir, suponemos que no existen transiciones con $x \in pre(t) \setminus post(t)$ y $\nu \in post(t)$. Si la red no cumple esa condición, es inmediato construir otra equivalente que sí la cumple, desdoblando todas las transiciones que no cumplan esa condición en dos que se disparen sucesivamente, tales que la primera borra el identificador y la segunda crea el nuevo. Si N está acotada en anchura por el número n, consideramos un conjunto I de n identificadores tal que $S(M_0) \subseteq I$. Pues bien, por ser N acotada en anchura, el árbol de Karp-Miller generalizado $\mathcal{T} = (S, A)$ de N es finito. Vamos a definir la APN $N^* = (P^*, T^*, F^*, M_0^*)$ que simula N, con algunos arcos etiquetados con conjuntos de variables, lo que podemos hacer según la proposición 5.2.4. Usaremos la variable $\nu' \in Var$ en N^*, que será una variable más. Definimos la relación *kill* como la menor relación que verifica $kill(M_1 \overset{t(\sigma)}{\rightarrow} M_2, X)$ si y sólo si $M_1 \overset{t(\sigma)}{\rightarrow} M_2 \in A$ y σ instancia todas las variables de X (y sólo a ellas) a algún identificador que aparece en $S(M_1)$ pero no en $S(M_2)$[†]. Entonces tomamos:

- $P^* = P \cup S \cup \{R\}$,
- $T^* = \{t(s_1, s_2) \mid s_1 \overset{t(\sigma)}{\rightarrow} s_2 \in A\}$,
- $F^*(p, t(s_1, s_2)) = F(p, t)$ y $F^*(t(s_1, s_2), p) = \begin{cases} F(t,p) \text{ si } F(t,p) \neq \nu \\ \nu' \text{ si } F(t,p) = \nu \end{cases}$ para todo $p \in P$ y $t(s_1, s_2) \in T^*$,
- $F^*(s_1, t(s_1, s_2)) = F^*(t(s_1, s_2), s_2) = \varepsilon$,
- Si $\nu \in post(t)$ entonces $F^*(R, t(s_1, s_2)) = \nu'$,

[†] Con las notaciones que introduciremos en el capítulo 10, $X_i \in part(\sigma)$, $pre_{X_i}(t) \in M_1$ y $post_{X_i}(t) = \emptyset$.

- $F^*(t(s_1,s_2),R) = X$ si $kill(s_1 \overset{t(\sigma)}{\to} s_2, X_i)$ para $X_1,\ldots,X_m$ y $X = \{x_1,\ldots,x_m \mid x_i \in X_i\}$ es un conjunto cualquiera formado por un representante de cada conjunto de variables X_i,
- Además, dado el marcaje alcanzable M de N definimos el marcaje M^* de N^* de la siguiente manera: la proposición 6.3.12 nos garantiza la existencia de un ω-marcaje s en S que coincide con M donde no toma el valor ω, que por tanto es único. Entonces definimos $M^*(p) = M(p)$ si $p \in P$, $M^*(s) = 1$, $M^*(s') = 0$ para todo $s \neq s'$, y $M^*(R) = I \setminus S(s)$.

Proposición 6.3.15

$$M_1[t(\sigma)\rangle M_2 \Leftrightarrow M_1^*[t(s_1,s_2)(\sigma)\rangle M_2^*$$

Demostración. En primer lugar, veamos que si $M_1[t(\sigma)\rangle M_2$ entonces $M_1^*[t(s_1,s_2)(\sigma)\rangle M_2^*$. Sea s_1 el ω-marcaje que nos da la proposición 6.3.12. Por construcción de $\mathcal{T}$, como $M_1[t(\sigma)\rangle M_2$ también $s_1[t(\sigma)\rangle s_2$, y por tanto s_2 es el ω-marcaje que nos da la proposición 6.3.12 para M_2 con $s_1 \overset{t(\sigma)}{\to} s_2 \in A$ y por tanto $t(s_1,s_2) \in T^*$. Por definición de N^* se tiene que

$$^\bullet t(s_1,s_2) = \begin{cases} {}^\bullet t \cup \{s_1\} \text{ si } \nu \notin Var(t) \\ {}^\bullet t \cup \{s_1, R\} \text{ si } \nu \in Var(t) \end{cases} \quad \text{y}$$

$$t(s_1,s_2)^\bullet = \begin{cases} t^\bullet \cup \{s_2, R\} \text{ si } kill(s_1 \overset{t(\sigma)}{\to} s_2, X_i) \\ t^\bullet \cup \{s_2\} \text{ en otro caso,} \end{cases}$$

Pues bien, si $p \in P$ y $F(t,p) \neq \nu$ entonces $M_1^*(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\} = M_1(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\} = M_2(p) = M_2^*(p)$. Claramente el token de s_1 en M_1^* pasa a s_2 en M_2^*. Si $\nu \notin Var(t)$ y no existe X con $kill(s_1 t(\sigma) \to s_2, X)$ entonces $S(M_1) = S(M_2)$ y $M_2^*(R) = M_1^*(R)$. Si $F(t,p) = \nu$ para algún p entonces $F^*(R, t(s_1,s_2)) = F^*(t(s_1,s_2),p) = \nu'$. Como hemos supuesto que en tal caso no existe x con $x \in pre(t) \setminus post(t)$, se tiene que $S(M_1) \subset S(M_2) \subset I$. Entonces existe $a \in Id$ tal que $\sigma(\nu) = a \in M_1^*(R)$ y $a \notin M(p)$ para $p \neq R$, y por tanto $M_1^*(p) - \{\sigma(F(p,t))\} + \{a\} = M_1(p) - \{\sigma(F(p,t))\} + \{a\} = M_2(p) = M_2^*(p)$. Además, como $S(M_2) = S(M_1) \cup \{a\}$ se cumple para R que $M_1^*(R) - \{a\} = I \setminus (S(M_1) \cup \{a\}) = M_2^*(R)$. Por último, supongamos que $kill(s_1 t(\sigma) \to s_2, X_i)$ para $i = 1,\ldots,k$ y $F^*(t(s_1,s_2),R) = X = \{x_1,\ldots,x_k\}$. Por definición de *kill* se cumple que $\sigma(F(t(s_1,s_2)),R) = \sigma(X) = S(s_1) \setminus S(s_2)$, es decir, $S(s_2) = S(s_1) \cup \sigma(X)$. Entonces $M_2^*(R) = I \setminus S(s_2) = (I \setminus S(s_1)) - \sigma(X) = M_1^*(R) - \sigma(X)$, como queríamos demostrar.

Para el recíproco claramente si $M_1^*[t(s_1,s_2)(\sigma)\rangle M_2^*$ entonces $\sigma(F(p,t)) \in M_1^*(p) = M_1(p)$ si $p \in P$ y $t(\sigma)$ se puede disparar en M_1 para alcanzar M_2 pues $M_2^*|_P = M_2$.

□

La APN de la figura 6.17 es la que resulta de aplicar la anterior construcción a la ν-APN de la figura 6.13*. El árbol de Karp-Miller para esa red es el de la

*En realidad, esa red ya es una APN.

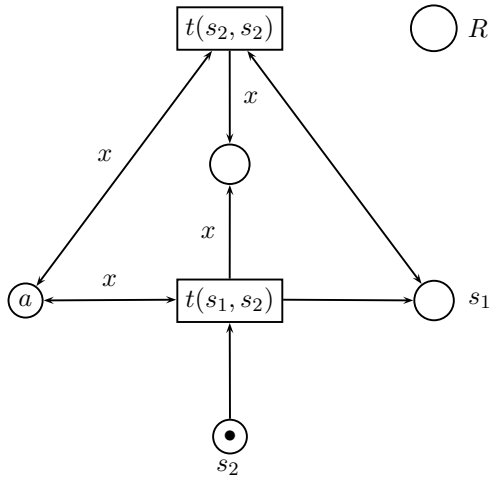

Figura 6.17: APN que simula la ν-APN de la figura 6.13

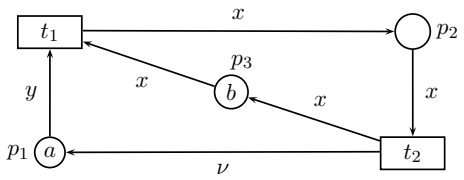

Figura 6.18: Otra ν-APN acotada en anchura

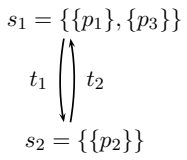

Figura 6.19: Árbol de Karp-Miller de la ν-APN de la figura 6.18

figura 6.16. Otro ejemplo se muestra en la figura 6.18, cuyo árbol de Karp-Miller aparece en la figura 6.19, y la APN que la simula está en la figura 6.20.

Corolario 6.3.16 *El problema de alcanzabilidad para redes acotadas en anchura es decidible.*

Demostración. Dado un marcaje M de una ν-APN N, construimos la APN N^* a partir de su árbol de Karp-Miller $\mathcal{T} = (S, A)$, que es finito. Supongamos que $|S(\overline{M})| \leq n$ para todo $\overline{M} \in S$. Si $|S(M)| > n$ entonces trivialmente no es alcanzable. Supongamos por tanto que $|S(M)| \leq n$. Pues bien, al ser todos los marcajes alcanzables de N^* estables, por la proposición anterior M es alcanzable en N si y sólo si existen $\overline{M} \in S$ y una biyección $h : I \to S(M)$ tales que:

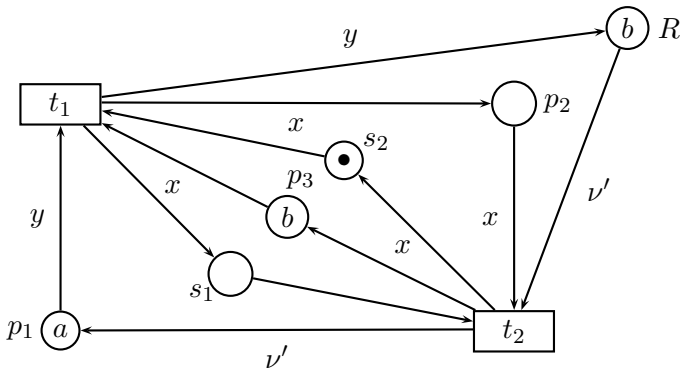

Figura 6.20: APN que simula la ν-APN de la figura 6.18

- $M \sqsubseteq_\alpha \overline{M}$
- M^* es alcanzable en N^*, donde
  - $M^*(p)(a) = M(p)(h(a))$ para todo $p \in P$,
  - $M^*(\overline{M}) = 1$ y $M^*(s) = 0$ para todo $s \neq \overline{M}$.

Como sólo existen un número finito de ω-marcajes en S y un número finito de biyecciones entre I y $S(M)$, la alcanzabilidad en N se reduce a la alcanzabilidad en N^*, siendo ésta decidible, como se probó en el capítulo 4.

□

Por último, es fácil ver que la acotación histórica también es decidible, es decir, el problema de decidir si toda traza contiene un número acotado de identificadores. Para probarlo, basta tener en cuenta que dada una red N podemos construir N^* de manera que N es históricamente acotada si y sólo si N^* es acotada en anchura, sin más que añadir a N^* un lugar c en el que se depositen todos los identificadores que se crean, es decir, con $F^*(t, c) = \nu$ para todo t tal que $\nu \in post(t)$. Si ponemos en c todos los identificadores que aparecen en el marcaje inicial, es decir, $M_0^*(c) = S(M_0)$, y de c nunca se eliminan identificadores, entonces c contiene en cada marcaje alcanzable M todos los identificadores que se han generado en la traza que lleva a M. Por ser la acotación en anchura decidible, como acabamos de probar, tenemos el siguiente resultado.

Proposición 6.3.17 *La acotación histórica es decidible.*

Los resultados vistos en este capítulo nos permiten establecer una jerarquía entre los distintos modelos, que se presentan en la figura 6.21.

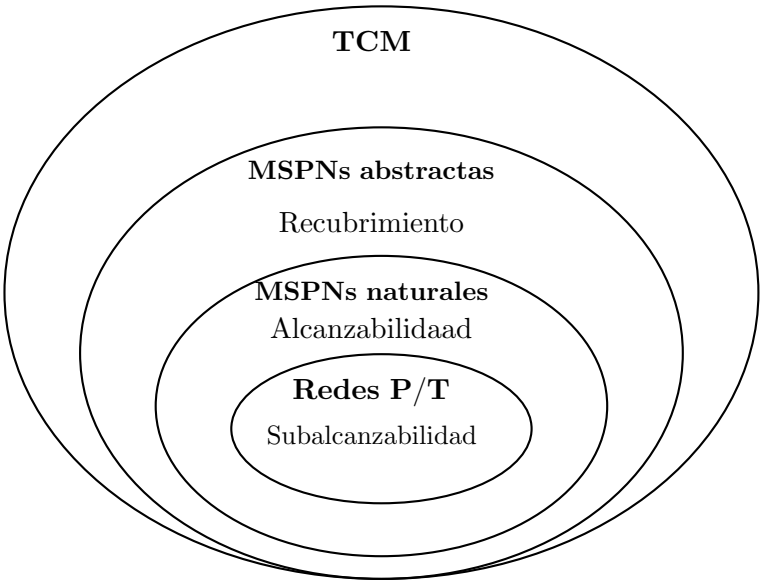

Figura 6.21: Resumen de resultados de decidibilidad

6.4. Desigualdades

Hasta ahora hemos considerado que el único control que tenemos sobre los distintos tipos de identificadores involucrados en el disparo de una transición viene dado por el uso de las mismas variables en distintos arcos, lo que obliga a que los identificadores a los que se instancian sean los mismos. En consecuencia, podemos especificar la restricción de que dos identificadores que tomamos de distintas precondiciones sean de hecho el mismo. Sin embargo, no contamos con ningún mecanismo que nos permita hacer lo mismo con la desigualdad de identificadores.

Vamos a incorporar un mecanismo que nos permita chequear la desigualdad de identificadores, de manera que obtendremos de manera más exacta lo que Andrew Gordon llama *nombres puros* [Gor00], que sólo pueden ser comparados entre sí mediante la igualdad o la desigualdad.

Dado el conjunto de variables $V \subseteq \mathit{Var}$, denotamos mediante $\mathit{Expr}(V)$ al menor conjunto que satisface

$$\mathit{Expr}(V) = \{x = y, x \neq y \mid x, y \in V\} \cup \{e_1 \wedge e_2 \mid e_1, e_2 \in \mathit{Expr}(V)\}$$

y $\mathit{Expr} = \bigcup_{V \subseteq \mathit{Var}} \mathit{Expr}(V)$. Escribiremos $\mathit{Var}(x = y) = \{x, y\}$, $\mathit{Var}(x \neq y) = \{x, y\}$ y $\mathit{Var}(e_1 \wedge e_2) = \mathit{Var}(e_1) \cup \mathit{Var}(e_2)$.

Una $\nu_{\neq}$-APN es una tupla $N = (P, T, F, \lambda)$ tal que (P, T, F) es una ν-APN y $\lambda : T \to \mathit{Expr}$ es una función tal que $\mathit{Var}(\lambda(t)) \subseteq \mathit{Var}(t)$ para todo $t \in T$.

Dada una transición $t \in T$, $e \in \mathit{Expr}(\mathit{Var}(t))$ y una función $\sigma : \mathit{Var}(t) \to \mathit{Id}$ escribiremos $\sigma \vdash e$ si ello se puede derivar aplicando las siguientes reglas:

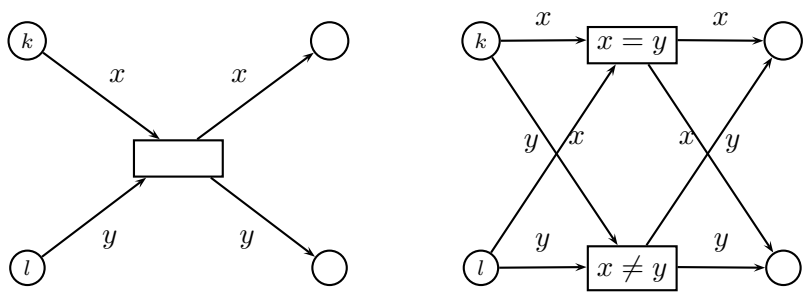

Figura 6.22: Simulación de una ν-APN (izquierda) por una $\nu_{\neq}$-APN (derecha)

- $\sigma \vdash x = y \Leftrightarrow \sigma(x) = \sigma(y)$
- $\sigma \vdash x \neq y \Leftrightarrow \sigma(x) \neq \sigma(y)$
- $\sigma \vdash e_1 \wedge e_2 \Leftrightarrow \sigma \vdash e_1$ y $\sigma \vdash e_2$

Diremos que $\sigma : Var(t) \to Id$ es un modo de t si $\sigma \vdash \lambda(t)$.

Toda ν-APN puede ser simulada por una $\nu_{\neq}$-APN, sin más que tener en cuenta que una transición de una ν-APN en la que intervienen varias variables se puede simular con una elección (no determinista desde el punto de vista estático, pero que será determinista dinámicamente) de transiciones. Por ejemplo, en el caso en el que la transición tuviese dos variables x e y adyacentes, simularíamos la transición mediante la elección de una transición etiquetada por $x = y$ y otra etiquetada por $x \neq y$ (ver figura 6.22). Con ello todos los resultados de indecidibilidad de las ν-APNs se conservan cuando admitimos también el uso de desigualdades entre identificadores.

Proposición 6.4.1 *El problema de la alcanzabilidad para las $\nu_{\neq}$-APNs es indecidible.*

Pero, ¿qué ocurre con nuestro principal resultado, es decir, con la decidibilidad del recubrimiento para las ν-APNs? Pues bien, se puede ver que la demostración que vimos de la decidibilidad del recubrimiento se puede trasladar al nuevo modelo. En efecto, el orden a considerar es exactamente el mismo que allí, y el carácter de buen cuasi orden del mismo no depende de la semántica. La monotonía del orden con respecto a la relación de disparo también se preserva en presencia de la desigualdad. Por último, el conjunto $min(Pre_t(\mathcal{C}(M)))$ sigue siendo computable, sin más que restringir el conjunto de modos en los que se puede disparar t para alcanzar un marcaje mayor que M.

Proposición 6.4.2 *El problema del recubrimiento para las $\nu_{\neq}$-APNs es decidible.*

Así que las $\nu_{\neq}$-APNs son más expresivas que las ν-APNs, pero los problemas considerados son igualmente (in)decidibles en ambos modelos, por lo que queda como problema abierto el distinguir entre sí a ambos.

6.5. Arcos de transferencia y reset

Ahora veamos que el modelo de las redes de Petri con arcos de transferencia es, en cierto sentido, un caso particular de nuestras ν-APNs. En particular, vamos a ver que es posible realizar una simulación débil de toda red con arcos de transferencia por medio de una ν-APN, es decir, una simulación que en principio permite más cómputos que los simulados, pero de manera que los cómputos que no respeten la simulación no cumplan determinado invariante fácilmente chequeable.

Definición 6.5.1 *Una red de Petri con* arcos de transferencia *es una tupla* $N = (P, T, F, F_t)$*, donde*

- P *es un conjunto de lugares,*
- T *es un conjunto de transiciones,*
- $F \subseteq (P \times T) \cup (T \times P)$*,*
- $F_t \subseteq P \times T \times P$*.*

Las tripletas de F_t son los arcos de transferencia, que denotaremos en las figuras por flechas dobles. Dada una transición $t \in T$, escribiremos ${}^\bullet t = \{p \in P \mid (p, t) \in F\}$ y ${}^t t = \{p \in P \mid (p, t, p') \in F_t\}$, y análogamente para $t^\bullet$ y t^t. También escribiremos P_t para referirnos al conjunto de lugares que son precondición de algún arco de transferencia, es decir, $\{p \in P \mid t \in T,\ p \in {}^t t\}$ y T_t para el conjunto de transiciones que pertenecen a algún arco de transferencia, es decir, $\{t \in T \mid (p, t, p') \in F_t$ para p y $p'\}$.

Definición 6.5.2 *Un* marcaje *de una red de Petri con arcos de transferencia* N *es un multiconjunto de lugares de* N*. Diremos que una transición* t *de* N *está activada si* $M(p) > 0$ *para todo* $p \in {}^\bullet t \cup {}^t t$*. Entonces* t *se puede disparar, produciendo el marcaje* M'*, definido por:*

- $M'(p) = M(p) - F(p, t) + F(t, p)$ *para todo* $p \notin {}^t t \cup t^t$*,*
- $M'(p) = 0$ *para todo* $p \in {}^t t$*,*
- $M'(p) = M(p) + M(p')$ *para todo* $p \in t^t$*, donde* $(p', t, p) \in F_t$*.*

Veamos ahora cómo podemos simular las redes con arcos de transferencia por medio de las ν-APNs. La idea es usar, para cada lugar precondición de transferencia de una transición, un identificador que identifique los tokens válidos en ese lugar. De este modo, para cada lugar p tendremos otro lugar $c(p)$, que contendrá el token válido para p en cada momento. Así, si el token válido es a (es decir, en $c(p)$ se encuentra el identificador a) entonces un cierto número de tokens negros en p vendrán identificados en N^* por el mismo número de tokens a en p.

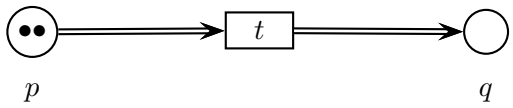

Figura 6.23: Red con arcos de transferencia

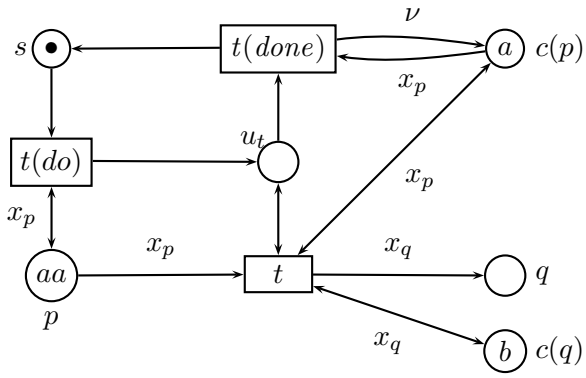

Figura 6.24: Simulación débil de la red con arcos de transferencia

Simulamos cada transición t por medio del disparo de la transición $t(do)$ primero, el disparo de $t(a_1), \ldots, t(a_n)$ después y finalmente el disparo de $t(done)$, donde $a_1, \ldots, a_n$ son los arcos de transferencia que contienen a t, para $n \geq 0$. Nótese que si una transición no tiene arcos de transferencia entonces no sería necesario desdoblarla en dos, aunque lo haremos por homogeneidad. Además, todas las transiciones $t(do)$ tienen como precondición a un lugar s, que es postcondición de $t(done)$, de modo que hasta que no se ha acabado de simular el disparo de t ninguna otra transición podrá usar los tokens que las transiciones $t(a)$ van colocando en las postcondiciones de t.

Así que dada una red con arcos de transferencia $N = (P, T, F, F_t)$ vamos a construir $N^* = (P^*, T^*, F^*)$ que la simula débilmente. Consideraremos una variable distinta x_p para cada $p \in P$. Definimos N^* a continuación:

- $P^* = P \cup \{c(p) \mid p \in P\} \cup \{s\} \cup \{u_t \mid t \in T\}$,
- $T^* = \{t(do), t(a), t(done) \mid t \in T,\ a = (p, t, q) \in F_t\}$,
- F^* se define de la siguiente manera:
  - $F^*(s, t(do)) = F^*(t(do), u_t) = F^*(u_t, t(done)) = F^*(t(done), s) = F^*(u_t, t(a)) = F^*(t(a), u_t) = \varepsilon$ para todo $t \in T$ y para todo $a = (p, t, q) \in F_t$,
  - $F^*(p, t(do)) = x_p$ para todo $p \in {}^\bullet t$,
  - $F^*(p, t(do)) = F^*(t(do), p) = x_p$ para todo $p \in {}^t t$,

- $F^*(c(p), t(do)) = F^*(t(do), c(p)) = x_p$ para todo $p \in {}^\bullet t \cup {}^t t$,
- Si $a = (p, t, q) \in F_t$ entonces $F^*(p, t(a)) = F^*(c(p), t(a)) = F^*(t(a), c(p)) = x_p$, $F^*(t(a), q) = F^*(c(q), t(a)) = F^*(t(a), c(q)) = x_q$, $F^*(c(p), t(done)) = x_p$ y $F^*(t(done), c(p)) = \nu$,
- $F^*(c(p), t(done)) = F^*(t(done), c(p)) = x_p$ y $F^*(t(done), p) = x_p$ para todo $p \in t^\bullet$.

Además, dado un marcaje M de N, definimos el siguiente marcaje M^* de N^*: Sea $\{\eta_p \mid p \in P\}$ una familia de identificadores cualesquiera de Id. Entonces tomamos

- $M^*(p)(\eta_p) = M(p)$ y $M^*(p)(b) = 0$ para todo $p \in P$ y $b \neq \eta_p$,
- $M^*(c(p)) = \{\eta_p\}$ para todo $p \in P$,
- $M^*(s) = 1$ y $M^*(u_t) = 0$ para todo $t \in T$.

Proposición 6.5.3 *Sea N una red con arcos de transferencia y N^* la ν-APN definida anteriormente. Entonces $M_1 \to^* M_2$ si y sólo si $M_1^* \to^* M_2^*$.*

Demostración. Veamos primero que si $M_1[t\rangle M_2$ entonces $M_1 \to^* M_2$. Supongamos que existen n arcos de la forma $a_i = (p_i, t, q_i)$ en F_t, con $n \geq 0$. Veamos entonces que la siguiente es una traza en N^*:

$$M_1^*[t(do)\rangle \overline{M}_0[t(a_1)(\sigma_1)\rangle^{l_1} \overline{M}_1 \ldots \overline{M}_{n-1}[t(a_n)(\sigma_n)\rangle^{l_n} \overline{M}_n[t(done)(\sigma')\rangle M_2^* \quad (6.1)$$

donde para cada $i = 1, \ldots, n$, $l_i = M_1(p_i)$ y $\overline{M}_i$ se define de la siguiente forma:

- $\overline{M}_i(s) = 0$ y $\overline{M}_i(u) = 1$,
- $\overline{M}_i(p_j) = \emptyset$ para $j = 1, \ldots i$,
- $\overline{M}_i(p_j) = M_1^*(p_j)$ para $j = i+1, \ldots, n$,
- $\overline{M}_i(q_j) = \{\eta_j, \overset{M_1(p_j)+M_1(q_j)}{\ldots}, \eta_j\}$ para $j = 1, \ldots, i$,
- $\overline{M}_i(q_j) = M_1^*(q_j)$ para $j = i+1, \ldots, n$,
- $\overline{M}_i(p) = \{\eta, \overset{M_1(p)-1}{\ldots\ldots}, \eta\}$ para $p \in {}^\bullet t$.

Veámoslo por inducción sobre i. Para $i = 0$ tenemos que ver que tras disparar $t(do)$ se alcanza un marcaje $\overline{M}_0$ que coincide con M_1^* excepto en s, u_t y $p \in {}^\bullet t$. Por construcción de N^* tenemos que ${}^\bullet t(do) = \{s\} \cup \{p, c(p) \mid p \in {}^\bullet t \cup {}^t t\}$ y $t(do)^\bullet = \{u_t\} \cup \{p, c(p) \mid p \in {}^t t\} \cup \{c(p) \mid p \in {}^\bullet t\}$. Por hipótesis t se puede disparar en M_1, de modo que $M_1(p) > 0$ para todo $p \in {}^\bullet t \cup {}^t t$ y, por lo tanto, $M_1^*(p) \neq \emptyset$. Además, $M_1^*(s) = 1$ y los lugares $c(p)$ nunca están vacíos en marcajes de la forma M^*, así que la transición $t(do)$ se puede disparar en modo σ, donde $\sigma(x_p) = \eta_p$ si

$c(p) = \{\eta_p\}$, alcanzando un marcaje $\overline{M}_0$. Veamos que $\overline{M}_0$ cumple las condiciones anteriores. Como ${}^\bullet t(do) \setminus t(do)^\bullet = \{s\} \cup \{p \mid p \in {}^\bullet t\}$ y $t(do)^\bullet \setminus {}^\bullet t(do) = \{u_t\}$ tenemos que tras el disparo de $t(do)$, $\overline{M}_0(u_t) = 0$, $\overline{M}_0(s) = 0$ y $\overline{M}_0(p) = M_1^*(p) - \{\sigma(x_p)\} = M_1^*(p) - \{\eta_p\}$ para $p \in {}^\bullet t$, como queríamos demostrar.

Suponiendo que $M_1^*[t(do)\rangle \overline{M}_0[t(a_1)(\sigma_1)\rangle^{l_1} \overline{M}_1 \ldots \overline{M}_{i-2}[t(a_{i-1})(\sigma_{i-1})\rangle^{l_{i-1}} \overline{M}_{i-1}$ veamos que $\overline{M}_{i-1}[t(a_i)(\sigma_i)\rangle^{l_i} \overline{M}_i$. Por construcción de N^* se cumple que ${}^\bullet t(a_i) \setminus t(a_i)^\bullet = \{p_i\}$ y $t(a_i)^\bullet \setminus {}^\bullet t(a) = \{q_i\}$. La hipótesis de inducción nos dice que $\overline{M}_{i-1}(p_i) = l_i$, por lo que podemos disparar sucesivamente l_i veces la transición $t(a_i)$ en modo σ_i, con $\sigma_i(x_p) = \{\eta_p\}$ si $M_1^*(c(p)) = \{\eta_p\}$. El marcaje obtenido coincide con $\overline{M}_{i-1}$ excepto en los lugares p_i y q_i, donde contiene $M_1^*(p_i) - \{\eta_{p_i}, \overset{l_i}{\ldots}, \eta_{p_i}\} = \emptyset = \overline{M}_i(p_i)$ y $M^*(q_i) + \{\eta_{q_i}, \overset{l_i}{\ldots}, \eta_{q_i}\} = \{\eta_{q_i}, \overset{M_1(q_i)+M_1(p_i)}{\ldots}, \eta_{q_i}\} = \overline{M}_i(q_i)$, y, por lo tanto, coincide con $\overline{M}_i$.

Sólo queda por ver que $\overline{M}_n[t(done)(\sigma')M_2^*$ para tener que 6.1 es una traza en N^*. Como ${}^\bullet t(done) \setminus t(done)^\bullet = \{u_t\}$ y $t(done)^\bullet \setminus {}^\bullet t(done) = \{s\} \cup \{p \mid p \in t^\bullet\}$, $t(done)$ se puede disparar, obteniendo un marcaje M. Veamos que $M = M_2^*$: En primer lugar, $M(u_t) = 0$ y $M(s) = 1$. Además, para todo $p \in t^\bullet$, $M(p) = \overline{M}_n(p) + \{\eta_p\}$ si $M_1^*(c(p)) = \{\eta_p\}$ y $M(p) = \overline{M}_n(p)$ para el resto de lugares $p \in P$. Además, $M(p) = \eta$ para algún η nuevo y para todo $p \in {}^t t$. De lo anterior se tiene que:

- Si $p \in {}^t t$ entonces $p = p_i$ para algún i y $M(p) = \overline{M}_n(p) = \emptyset = M_2(p)$, pues $M_2(p) = 0$,
- Si $p \in t^t$ entonces $p = q_i$ para algún i y $M(p) = \overline{M}_n(p) = \{\eta_p, \overset{M_1(q_i)+M_1(p_i)}{\ldots}, \eta_p\} = \{\eta_p, \overset{M_2(p)}{\ldots}, \eta_p\} = M_2^*(p)$,
- Si $p \in t^\bullet \setminus {}^\bullet t$, $M(p) = \overline{M}_n(p) + \{\eta_p\} = M_1^*(p) + \{\eta_p\} = M_2^*(p)$,
- Si $p \in {}^\bullet t \setminus t^\bullet$, $M(p) = \overline{M}_n(p) = \{\eta, \overset{M_1(p)-1}{\ldots}, \eta\} = M_2^*(p)$

De todo lo visto se concluye que $M = M_2^*$ y hemos probado que $M_1[t\rangle M_2$ implica $M_1^* \to^* M_2^*$, de lo que se deduce que $M_1 \to^* M_2$ implica $M_1^* \to^* M_2^*$.

Veamos ahora el recíproco. Supongamos que en el cómputo $M_1^* \to^* M_2^*$, M_2^* es el primer marcaje estable (de lo contrario trabajamos con el primero que lo sea). Como $M_1^*(s) = 1$ y $M_1^*(u_t) = 0$ para todo $t \in T$, necesariamente la primera transición disparada ha de ser del tipo $t(do)$, pues todas las demás tienen a algún u_t como precondición. Así que existe t tal que $M_1^*[t(do)(\sigma)\rangle \to^* M_2^*$. Veamos que entonces $M_1[t\rangle M_2$, lo que implicará la tesis. En primer lugar, como ${}^\bullet t(do) = \{s\} \cup \{p, c(p) \mid p \in {}^\bullet t \cup {}^t t\}$ y $t(do)$ se puede disparar en modo σ desde M_1^* se tiene que $\sigma(F(c(p), t)) = \sigma(F(p, t)) = \sigma(x_p) \in M_1^*(p)$ para todo $p \in {}^\bullet t \cup {}^t t$. Entonces $M_1(p) > 0$ para todo $p \in {}^\bullet t \cup {}^t t$ y la transición t está activada y $M_1[t\rangle M'$. Veamos que $M' = M_2$.

Por definición de disparo en las redes con arcos de transferencia tenemos que:

- $M'(p) = 0$ y $M'(q) = M_1(p) + M_1(q)$ para todo $(p, t, q) \in F_t$,

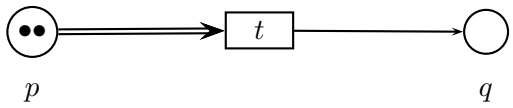

Figura 6.25: Red con arcos reset

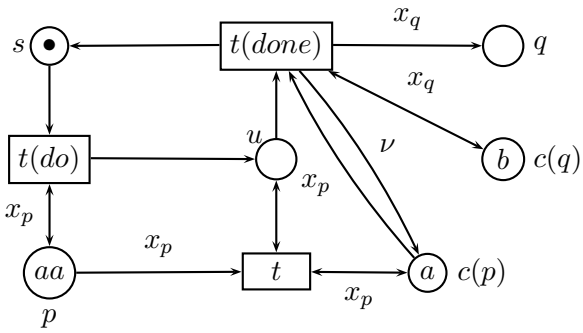

Figura 6.26: Simulación débil de la red reset

- $M'(p) = M_1(p) - 1$ para $p \in {}^\bullet t \setminus t^\bullet$,
- $M'(p) = M_1(p) + 1$ para $p \in t^\bullet \setminus {}^\bullet t$,
- $M'(p) = M_1(p)$ para el resto de lugares $p \in P$.

Por la forma de ${}^\bullet t(do)$ y $t(do)^\bullet$, como $M_1(p) = M'(p) \neq \emptyset$ para todo $p \in {}^t t$ y $M'(u_t) = 1$, las únicas transiciones activadas son $t(a)$ para todo $a = (p, t, q) \in F_t$ y $t(done)$. Por un razonamiento análogo al utilizado en la demostración de la proposición 6.1.8, se demuestra que sólo se dispara $t(done)$ después de hacerlo $M_1(p)$ veces $t(a)$ para cada $a = (p, t, q)$ o de lo contrario, se generaría un marcaje ilegal, que nunca podría volver a legalizarse, en contradicción con el hecho de que M_2^* es legal. En este caso un marcaje $\overline{M}$ de N^* es legal cuando $S\overline{M}(c(p)) \subseteq S(\overline{M}(p))$ para todo $p \in P$.

Tras disparar $M_1(p)$ veces cada una de las $t(a)$ obtenemos un marcaje M'' tal que $M''(p) = \emptyset$ $M''(q) = \{\eta_q, \overset{M_1(p)+M_1(q)}{\cdots\cdots}, \eta_q\}$. Además, $M''(p) = M_1^*(p) - \{\eta_p\}$ para todo $p \in {}^\bullet t$. En este momento ya no se puede disparar ninguna de las $t(a)$, pues todas las precondiciones $p \in {}^t t$ están vacías, de modo que $M''[t(done)(\sigma')\rangle M_2^*$. El marcaje M_2^* coincide con M'' excepto en los lugares s, u_t, $c(p)$ y, más importante, en los $p \in t^\bullet$, donde verifica que $M_2^*(p) = M''(p) + \{\eta_p\}$. De lo anterior se deduce que $M'^* = M_2^*$ y, por lo tanto, $M' = M_2$ y concluimos la demostración.

□

Una construcción similar a la vista nos permite simular débilmente las redes con arcos reset por medio de las ν-APNs. Un arco reset (o de puesta a cero) es un par (p, t) con $p \in P$ y $t \in T$. El disparo de la transición t tiene el efecto de vacíar el contenido del lugar p, aunque sin colocar esos tokens en ningún otro sitio, como ocurría con los arcos de transferencia.

Definición 6.5.4 *Una red de Petri con* arcos reset *es una tupla* $N = (P, T, F, F_r)$, *donde*

- P *es un conjunto de lugares,*
- T *es un conjunto de transiciones,*
- $F \subseteq (P \times T) \cup (T \times P)$,
- $F_r \subseteq P \times T$.

Los pares F_r son los arcos reset. Dada una transición $t \in T$ escribiremos ${}^\bullet t = \{p \in P \mid (p, t) \in F\}$ y ${}^r t = \{p \in P \mid (p, t) \in F_r\}$, y análogamente para $t^\bullet$.

Definición 6.5.5 *Un* marcaje *de una red de Petri con arcos reset N es un multiconjunto de lugares de N. Diremos que una transición t de N está activada si $M(p) > 0$ para todo $p \in {}^\bullet t \cup {}^r t$. Entonces t se puede disparar, produciendo el marcaje M', definido por:*

- $M'(p) = M(p) - F(p, t) + F(t, p)$ *para todo* $p \notin {}^r t$,
- $M'(p) = 0$ *para todo* $p \in {}^r t$.

Proposición 6.5.6 *Sea N una red con arcos reset. Entonces existe una ν-APN N^* y un marcaje M^* de N^* para cada marcaje M de N tal que $M_1 \rightarrow^* M_2$ si y sólo si $M_1^* \rightarrow^* M_2^*$.*

Demostración. La demostración es prácticamente idéntica a la de la proposición 6.1.8. La única diferencia con la construcción vista en el caso de las redes con arcos de transferencia está en el hecho de que el disparo de $t(a)$ con $a = (p, t)$ sólo tiene el efecto de eliminar un token de p igual al que se encuentre en $c(p)$.

□

Una red con arcos de transferencia puede simular una red con arcos reset, sin más que llevar esos tokens de p a un lugar "contenedor", aunque esta simulación no preserva la propiedad de acotación, precisamente debido a la presencia de este lugar contenedor, que alberga todos los tokens que han sido alguna vez eliminados por un arco reset en la red original. Nuestra simulación débil sí que preserva la propiedad de acotación en las simulaciones correctas. En la figura 6.25 se muestra una red con arcos reset, cuya simulación débil se puede ver en la figura 6.26.

Capítulo 7

Redes ubicuas replicadas

Hasta ahora hemos considerado nuestros sistemas como formados por componentes que pueden actuar de una manera autónoma. Algunas de estas componentes se pueden interpretar como agentes (posiblemente móviles), tanto con un comportamiento reactivo como proactivo. Sin embargo, por el momento el modelo no cuenta con un mecanismo tan ampliamente usado en la práctica como es la clonación o generación de nuevos agentes. En este capítulo añadimos al formalismo una primitiva al estilo de la replicación en el cálculo π o el cálculo de ambientes, que tiene el efecto de crear una nueva red componente, con la misma estructura que la red que invoca esa primitiva, e inicialmente marcada de una manera fija. Se trata, por tanto, de una primitiva de replicación, que con la ayuda de la sincronización se puede usar para implementar un operador de creación de nuevos agentes.

En primer lugar añadiremos la primitiva al modelo básico, y después la incluiremos en el formalismo con nombres. Esta primitiva introduce en el modelo una nueva fuente de infinitud. Ahora, no sólo podemos tener una cantidad arbitraria de tokens en los lugares, o una cantidad arbitraria de nombres distintos, en el caso de que consideremos nombres, sino que podemos tener una cantidad no acotada de componentes. Estudiaremos las consecuencias de introducir la replicación en cuanto a la capacidad expresiva del modelo y a la decidibilidad de sus propiedades.

Cada componente replicada puede evolucionar de una manera independiente, aunque interaccionando eventualmente con otros procesos. Por ello, consideraremos la creación dinámica de hebras, que pueden ser de infinitos estados y que pueden sincronizar entre sí.

En este capítulo probaremos que el mecanismo de replicación y el de creación de nombres del capítulo 4 son equivalentes, es decir, que se pueden simular mutuamente. Para la demostración de esta equivalencia es esencial considerar un mecanismo de recolección de basura, que elimina hebras inactivas, ya que contamos con el análogo a ese mecanismo para los identificadores.

Sin embargo, veremos que aunque sean equivalentes, de algún modo no se

solapan, en el sentido de que un modelo que maneje simultáneamente a ambos sobrepasa la capacidad de cualquiera de ellos, obteniendo de hecho la Turing completitud. La explicación intuitiva de este hecho es que, una vez que consideramos nombres, estos pueden ser usados para distinguir entre lo que eran componentes indistinguibles, de manera que con nombres, las hebras tienen una personalidad única.

7.1. Redes ubicuas replicadas

En esta sección vamos a introducir formal e intuitivamente las *redes ubicuas replicadas.*

Definición 7.1.1 *Una* red ubicua replicada*, RUN por las siglas en inglés de* Replicated Ubiquitous Nets*, es una red de Petri etiquetada* $N = (P, T, F, \lambda)$*, donde:*

- P *y* T *son conjuntos disjuntos y finitos de lugares y transiciones, respectivamente,*
- $F \subseteq (P \times T) \cup (T \times P)$ *es el conjunto de arcos de la red,*
- $\lambda : T \to \mathcal{A} \cup Sync \cup \mathcal{MS}(P)$.

La única diferencia con las redes ubicuas básicas está en el conjunto de etiquetas de las transiciones. Ahora algunas transiciones pueden estar etiquetadas con multiconjuntos de lugares, es decir, por un marcaje. Este multiconjunto se corresponderá con el marcaje inicial de la red que se crea cuando se dispara dicha transición.

Los marcajes de las RUN se definen de la siguiente manera.

Definición 7.1.2 *Un* marcaje *de una RUN* $N = (P, T, F, \lambda)$ *es un par* (M, k)*, donde* $M \in \mathcal{MS}(P)$ *y* $k \in \mathcal{L}$*. Denotaremos por* $Markings(N)$ *al conjunto de marcajes de* N*.*

El marcaje de un sistema RUN no puede ser tan sólo un conjunto de marcajes, uno por cada red del sistema, ya que podría haber varias copias activas de las componentes.

Definición 7.1.3 *Un* sistema RUN *es un conjunto* $\mathcal{N} = \{N_1, \ldots, N_n\}$ *de RUNs disjuntas dos a dos. Un marcaje* $\mathcal{M}$ *de tales sistemas es una función* $\mathcal{M} : \mathcal{N} \to \bigcup_{i=1}^{n} \mathcal{MS}(Markings(N_i))$ *tal que para todo* $i \in \{1, \ldots, n\}$, $\mathcal{M}(N_i) \in \mathcal{MS}(Markings(N_i))$.

Por tanto, $\mathcal{M}(N)$ representa al marcaje del subsistema de $\mathcal{N}$ compuesto sólamente de copias de N. Las definiciones de transiciones autónomas y pares de sincronizacion son análogas a los anteriores capítulos. Por ejemplo, la siguiente sería la definición de disparo de una transición autónoma.

Definición 7.1.4 *Dado un sistema RUN* $\mathcal{N} = \{N_1, \ldots, N_n\}$ *con* $N_i = (P_i, T_i, F_i, \lambda_i)$ *y un marcaje suyo* $\mathcal{M}$*, decimos que* $t \in T_i$ *con* $\lambda(t) \in \mathcal{A}$ *está* (M, k)*-activada si* $(M, k) \in \mathcal{M}(N_i)$ *y* ${}^\bullet t \subseteq M$*. Entonces decimos que* t *se puede disparar, y el marcaje alcanzado después del* (M, k)*-disparo de* t *es* $\mathcal{M}'$*, definido por:*

- $\mathcal{M}'(N_j) = \mathcal{M}(N_j)$ *para todo* j *con* $i \neq j$*,*
- $\mathcal{M}'(N_i) = \mathcal{M}(N_i) - \{(M, k)\} + \{(M', \ell)\}$*, donde:*
  - $M' = M - {}^\bullet t + t^\bullet$*,*
  - $\ell = k$ *o* $\lambda(t) = go\ \ell$*.*

En la definición anterior, se sustituye en $\mathcal{M}(N)$ el marcaje de la componente que dispara la transición por el que resulta del disparo, donde N es el tipo de la componente. De acuerdo con el último punto de la definición, si la transición estaba etiquetada con *go* ℓ entonces la localidad de la componente cambia adecuadamente. En caso contrario, ha de ser cierto que $k = \ell$, lo que expresa que la red en cuestión no se ha movido.

También las definiciones de transiciones de sincronización compatibles y disparo de un par de transiciones compatibles son análogas a las del capítulo 3. Vamos a detallar el disparo de una transición de replicación.

Definición 7.1.5 *Dado un sistema RUN* $\mathcal{N} = \{N_1, \ldots, N_n\}$ *con* $N_i = (P_i, T_i, F_i, \lambda_i)$ *y un marcaje suyo* $\mathcal{M}$*, decimos que* $t \in T_i$ *con* $\lambda(t) = M' \in \mathcal{MS}(P_i)$ *está* (M, k)*-activada si* $(M, k) \in \mathcal{M}(N_i)$ *y* ${}^\bullet t \subseteq M$*. Decimos entonces que* t *se puede disparar, y el marcaje alcanzado después del* (M, k)*-disparo de* t *es* $\mathcal{M}'$*, definido por:*

- $\mathcal{M}'(N_j) = \mathcal{M}(N_j)$ *para todo* j *con* $i \neq j$*.*
- $\mathcal{M}'(N_i) = \mathcal{M}(N_i) - \{(M, k)\} + \{(M', k), (M'', k)\}$*, donde* $M'' = M - {}^\bullet t + t^\bullet$*.*

Por tanto, la red que dispara la transición de replicación cambia su marcaje como si se tratara de una transición autónoma, al tiempo que se crea una nueva red del mismo tipo, marcada inicialmente por $\lambda(t)$, situada inicialmente donde se encuentra la red que la ha creado.

En cualquiera de los casos, escribiremos $\mathcal{N}(\mathcal{M})[u\rangle\mathcal{N}(\mathcal{M}')$ si se alcanza $\mathcal{M}'$ desde $\mathcal{M}$ tras el disparo de u, o simplemente $\mathcal{M}[u\rangle\mathcal{M}'$, si no hay confusión, donde u puede ser $t(M, k)$ o $(t(M, k), t'(M', k'))$.

Además, como un marcaje de un sistema RUN es un multiconjunto de marcajes de redes RUN, consideraremos el orden natural de multiconjuntos inducido por el orden entre marcajes de redes RUN definido por $(M, k) \sqsubseteq (M', k) \Leftrightarrow M \subseteq M'$.

Ejemplo 3 Vamos a modificar un ejemplo visto en el capítulo 3, para que aproveche la nueva primitiva de replicación. Modelamos un sistema compuesto por cuatro componentes, N, TH, C y A. Las componentes C y A se encuentran inicialmente

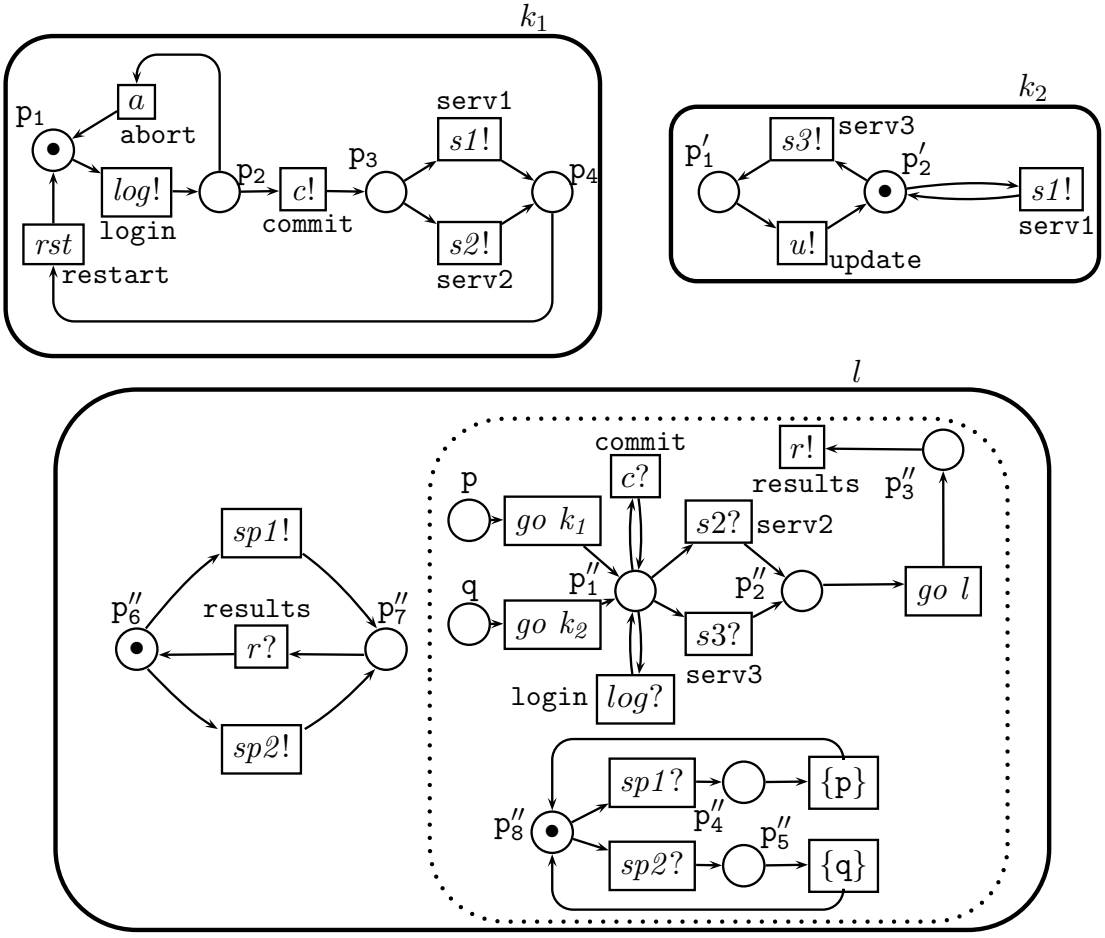

Figura 7.1: Sistema ubicuo modelado con RUNs

en la misma localidad l, mientras que NS y TH se encuentran en localidades distintas, k_1 y k_2, respectivamente (véase la figura 7.1). El procesador NS puede ser considerado como un sistema de notas electrónicas [FC03], que requiere autenticación para ver su contenido (acción identificada por el servicio `serv2`), mientras que se puede ver a TH como un termómetro electrónico [Wan04], en el que la acción de consultar la temperatura se denota por `serv3`. Además, ambos pueden devolver la hora local, denotado por el servicio `serv1`. C es un cliente y A (dentro de la línea discontinua) es un agente de C.

A es un proceso compuesto de dos componentes conexas, ambas dentro de la línea discontinua. Inicialmente, está esperando que el cliente le mande clonarse. Si el cliente sincroniza con él en $sp1$ entonces crea una nueva copia de sí mismo con un token en `p`, y si sincroniza en $sp2$ entonces ese token se encuentra en `q`. En cualquiera de los casos, el agente se mueve a la localidad indicada e intenta obtener los servicios `serv2` o `serv3` de manera no determinista, intentando ver los

contenidos del sistema de notas o la temperatura, respectivamente. En el primer caso, el sistema de notas obligará al agente a seguir un protocolo de sincronización si es que este quiere obtener el servicio (en caso contrario, puede abortar). Nótese que el agente intenta obtener el servicio $s2$ incluso sin seguir el protocolo, es decir, tiene activa la transición $s2?$ aunque no haya disparado antes $log?$ ni $c?$. Después de obtener el servicio, el agente vuelve a l y le ofrece los resultados a C.

La siguiente traza se corresponde con la situación en la que el proceso quiere recibir el servicio $s3$.

$$\mathcal{M}_0[(sp2?, sp2!)\rangle\mathcal{M}_1[rep_2\rangle\mathcal{M}_2[go\ k_2\rangle\mathcal{M}_3[(s3?, s3!)\rangle\mathcal{M}_4[go\ l\rangle\mathcal{M}_5[(r?, r!)\rangle\mathcal{M}_6$$

donde

$$\begin{aligned}
\mathcal{M}_0 &= \{(\{p_1\}, k_1), (\{p'_2\}, k_2), (\{p''_8\}, l), (\{p''_6\}, l)\}\\
\mathcal{M}_1 &= \{(\{p_1\}, k_1), (\{p'_2\}, k_2), (\{p''_5\}, l), (\{p''_7\}, l)\}\\
\mathcal{M}_2 &= \{(\{p_1\}, k_1), (\{p'_2\}, k_2), (\{p''_8\}, l), (\{p''_7\}, l), (\{q\}, l)\}\\
\mathcal{M}_3 &= \{(\{p_1\}, k_1), (\{p'_2\}, k_2), (\{p''_8\}, l), (\{p''_7\}, l), (\{p''_1\}, k_2)\}\\
\mathcal{M}_4 &= \{(\{p_1\}, k_1), (\{p'_1\}, k_2), (\{p''_8\}, l), (\{p''_7\}, l), (\{p''_2\}, k_2)\}\\
\mathcal{M}_5 &= \{(\{p_1\}, k_1), (\{p'_1\}, k_2), (\{p''_8\}, l), (\{p''_7\}, l), (\{p''_3\}, l)\}\\
\mathcal{M}_6 &= \{(\{p_1\}, k_1), (\{p'_1\}, k_2), (\{p''_8\}, l), (\{p''_6\}, l), (\emptyset, l)\}
\end{aligned}$$

7.2. Recolección de basura en sistemas RUN

Acabamos de introducir un tipo especial de transiciones que crean componentes con la misma estructura que la red que la dispara, pero marcadas con un marcaje posiblemente distinto, aquel especificado estáticamente en la definición de la red. Cada vez que se dispara una de esas transiciones el sistema gana una componente extra, de manera que los sistemas podrían crecer arbitrariamente, siempre que esas transiciones se puedan disparar un número arbitrario de veces.

Por el contrario, según las definiciones anteriores, no es posible que un sistema vea reducido su número de componentes. En otras palabras, una vez que se crea una nueva red componente, ya nunca puede ser eliminada, incluso si está definitivamente bloqueada. En esta sección proponemos una solución a este problema, considerando como basura a todas las componentes sin tokens, es decir, aquellas con un marcaje en el que todos sus lugares estén vacíos. Asumimos que las redes RUN no tienen transiciones sin precondiciones, ya que tales transiciones estarían siempre activadas, de modo que ninguna red con una transición así debería ser considerada basura. Si alguna sí tuviese una transición así basta con añadir un lugar que sea precondición y postcondición suya para obtener una red equivalente que sí cumple esa condición. En este supuesto, el conjunto de redes sin tokens forman un subconjunto de las redes bloqueadas, por lo que la recogida de basura

será correcta en el sentido de que no evitará el disparo de ninguna transición. Nótese que esta es una aproximación conservadora, ya que no eliminamos todas las hebras inactivas. Esto ilustra el hecho de que lo que llamamos recogida de basura en el formalismo con nombres en realidad no es tal cosa, ya que allí tampoco eliminamos los nombres que, aun apareciendo en el marcaje, no van a participar en futuros disparos de transiciones.

Para formalizar lo anterior, vamos a identificar los marcajes módulo una relación de equivalencia que ignora los marcajes vacíos (que representan componentes muertas).

Definición 7.2.1 *Dados dos marcajes $\mathcal{M}$ y $\mathcal{M}'$ de un sistema RUN $\mathcal{N}$ escribiremos $\mathcal{M} \equiv \mathcal{M}'$ si para todo $N \in \mathcal{N}$ se cumple que $\mathcal{M}(N) \equiv_N \mathcal{M}'(N)$, donde $\equiv_N$ es la menor relación de equivalencia sobre multiconjuntos de marcajes de N tal que $A \equiv_N A + \{(\emptyset, k)\}$ para todo $k \in \mathcal{L}$.*

Consideremos de nuevo el ejemplo de la sección 3. Cada vez que el cliente C sincroniza con A se crea un nuevo agente. Una vez que el nuevo agente le devuelve los resultados a C, no alberga ningún token. Efectivamente el marcaje $\mathcal{M}_6$ alcanzado tiene una componente $(\emptyset, l)$ que representa al cliente y que, de acuerdo con la definición anterior, podría ser eliminado de los marcajes.

7.3. Sistemas RUN centralizados

En esta sección vamos a ver una vez más, al igual que lo hacíamos en el capítulo 5, que la movilidad no es una característica esencial del modelo, y puede ser simulada por un formalismo que no la trate explícitamente. Por el contrario, las componentes sí juegan un papel esencial en los sistemas RUN, de manera que al contrario de lo que hicimos en el capítulo 5, no las eliminaremos.

En primer lugar, definiremos un tipo especial de sistemas RUN, aquéllos en los que sus componentes son estacionarias, es decir, no tienen transiciones de movimiento y todas se encuentran en la misma localidad.

Definición 7.3.1 *Una* red replicada centralizada *es una red replicada $N = (P, T, F, \lambda)$ tal que $\lambda(t) \neq go\ k$, para todo $k \in \mathcal{L}$ y $t \in T$.*

Definición 7.3.2 *Un* sistema RUN centralizado $\mathcal{N}$ *es un sistema RUN en el que todas sus componentes son redes centralizadas. Una función $\mathcal{M} : \mathcal{N} \to \bigcup_{N \in \mathcal{N}} \mathcal{MS}(Markings(N))$ es un marcaje de $\mathcal{N}$ (como sistema centralizado) si existe $k \in \mathcal{L}$ tal que para todo $N \in \mathcal{N}, \mathcal{M}(N) \in \mathcal{MS}(Markings(N))$ y para todo $(M, \ell) \in \mathcal{M}(N)$ se cumple que $\ell = k$.*

Por definición, toda red en un RUN centralizado es estacionaria, es decir, no tiene transiciones de movimiento. Además, cuando hablemos de sistemas centralizados, supondremos que todas las redes se encuentran en la misma localidad, de modo que podemos omitir la componente que indica la localidad en los marcajes.

A continuación presentamos la simulación por medio de la cual podemos codificar cualquier sistema RUN por medio de uno centralizado. Sea $\mathcal{N}$ un sistema RUN con marcaje inicial $\mathcal{M}_0$. Suponemos que existen etiquetas $go(l,k) \in \mathcal{A}$ para cualesquiera l y k en $\mathcal{L}$, distintas a cualquier otra de las etiquetas de transiciones autónomas que aparecen en $\mathcal{N}$. Sea $L = \{\ell_1, \ldots, \ell_m\} \subseteq \mathcal{L}$ el conjunto de localidades que aparecen tanto en $\mathcal{M}_0$ como localidades de sus componentes, como en $\mathcal{N}$, como destino de transiciones de movimiento. Como L es un conjunto finito, podemos añadir a cada red N el conjunto de lugares $\{N@\ell \mid \ell \in L\}$, de manera que un token en $N@\ell$ nos indicaría que la correspondiente componente de tipo N se encuentra en ℓ. Para simular los movimientos, sustituimos todas las transiciones etiquetadas por $go\ k$ por un conjunto de transiciones, etiquetadas por $go(l,k)$, para todo $l \in L$, y para cada una de ellas añadimos arcos que eliminan un token de $N@l$ y ponen uno en $N@k$.

Finalmente, para garantizar que las sincronizaciones sólo ocurren entre redes que se encuentran en la misma localidad, sustituimos todas las transiciones de sincronización t por un conjunto de transiciones también de sincronización de la forma t_l, para cada $l \in L$, y para cada t_l añadimos arcos de $N@l$ a t_l, y viceversa, de modo que t_l se dispare sólo cuando hay un token en el lugar $N@l$.

Formalmente, para cada $N = (P, T, F, \lambda) \in \mathcal{N}$ definimos la red centralizada $N^* = (P^*, T^*, F^*, \lambda^*)$ de la siguiente manera:

- $P^* = P \cup \{N@\ell \mid \ell \in L\}$,
- $T^* = \{t \mid t \in T, \lambda(t) \in \mathcal{A}, \lambda(t) \neq go\ k\} \cup$
  $\{t_{go}(\ell,k) \mid t \in T,\ \ell \in L,\ \lambda(t) = go\ k\} \cup$
  $\{t_\ell \mid t \in T, \lambda(t) \in Sync \cup \mathcal{MS}(P),\ \ell \in L\}$,
- $\lambda^*(t) = \lambda(t)$ para todo t tal que $\lambda(t) \in \mathcal{A}$ y $\lambda(t) \neq go\ k$ para todo k,
- $\lambda^*(t_\ell) = \begin{cases} \lambda(t) + \{N@\ell\} \text{ si } \lambda(t) \in \mathcal{MS}(P) \\ \lambda(t) \text{ si } \lambda(t) \in Sync \end{cases}$,
- $\lambda^*(t_{go}(\ell,k)) = go(\ell,k) \in \mathcal{A}$,
- F^* es el menor conjunto de pares que satisface:
  - Si $(t,p) \in F$ (y análogamente si $(p,t) \in F$) entonces:
    - $\lambda(t) \in \mathcal{A} \cup \mathcal{MS}(P) \Rightarrow (t,p) \in F^*$,
    - $\lambda(t) = go\ k \Rightarrow (t_{go}(\ell,k), p) \in F^*$ para todo $\ell \in L$,
    - $\lambda(t) \in Sync \Rightarrow (t_\ell, p) \in F^*$ para todo $\ell \in L$.
  - $(N@\ell, t_\ell), (t_\ell, N@\ell) \in F^*$ para todo $t_\ell \in T^*$,
  - $\lambda(t) = go\ k \Rightarrow (N@\ell, t_{go}(\ell,k)) \in F^*$ y $(t_{go}(\ell,k), N@k)) \in F^*$ para todo $\ell \in K$.

Dado un marcaje (M,k) de N definimos el marcaje $(M,k)^*$ de N^* mediante $(M,k)^*(p) = M(p)$ para todo $p \in P$, $(M,k)^*(N@\ell) = \begin{cases} 0 \text{ si } \ell \neq k \\ 1 \text{ si } \ell = k \end{cases}$

y extendemos esta definición puntualmente para marcajes de sistemas RUN. Entonces, si consideramos el sistema RUN centralizado RUN $\mathcal{N}^* = \{N^* \mid N \in \mathcal{N}\}$, se tiene la siguiente

Proposición 7.3.3

$$\mathcal{N}(\mathcal{M}_1)\ [u\rangle\ \mathcal{N}(\mathcal{M}_2) \Longleftrightarrow \mathcal{N}^*(\mathcal{M}_1^*)\ [v\rangle\ \mathcal{N}^*(\mathcal{M}_2^*).$$

Es más:

- $u = t(M,k)$ *con* $\lambda(t) \in \mathcal{A} \setminus \{go\ l \mid l \in \mathcal{L}\} \Leftrightarrow v = t(M,k)^*$,
- $u = (t^1(M_1,k), t^2(M_2,k)) \Leftrightarrow v = (t^1_k(M_1,k)^*, t^2_k(M_2,k^*))$,
- $u = t(M,k)$ *con* $\lambda(t) \in \mathcal{MS}(P) \Leftrightarrow v = t_k(M,k)^*$,
- $u = t(M,k)$ *con* $\lambda(t) = go\ l \Leftrightarrow v = t_{go}(k,l)(M,k)^*$.

Demostración. La demostración sigue ideas similares a las de la proposición 3.2.3, donde también se incorporaba a los marcajes las posiciones de cada red, aunque allí también se eliminaban las sincronizaciones.

- Este caso es inmediato, pues para estas transiciones el conjunto de precondiciones y postcondiciones en $\mathcal{N}$ coincide con el respectivo conjunto de precondiciones y postcondiciones en $\mathcal{N}^*$.
- En primer lugar, ${}^\bullet t^i_k = {}^\bullet t^i \cup \{N_i@k\}$ si t^i es una transición de la red N_i para $i = 1,2$. Como por definición $(M_i,k)^*(N_i@k) = 1$ se tiene que $t^1(M_1,k)$ y $t^2(M_2,k)$ pueden sincronizar si y sólo si lo pueden hacer $t^1_k(M_1,k)^*$ y $t^2_k(M_2,k)$. Además, $(t^i_k)^\bullet = (t^i)^\bullet \cup \{N_i@k\}$ y $(M,k)^*$ y $(M,k)^*$ coinciden en todos los lugares comunes, de modo que se sigue la tesis.
- En este caso también tenemos que ${}^\bullet t_k = {}^\bullet t \cup \{N@k\}$ y $t^\bullet_k = t^\bullet \cup \{N@k\}$, de manera que lo relativo al movimiento de tokens en N se ve igual que en el apartado anterior. En $\mathcal{N}$ se crea la componente $(\lambda(t),k)$, mientras que en $\mathcal{N}^*$ se crea $\lambda^*(t)$. Sin embargo, para concluir basta tener en cuenta que de hecho las redes creadas son las mismas (o las correspondientes) pues $\lambda^*(t_k) = \lambda(t) + \{N@k\} = (\lambda(t),k)^*$.
- Análogo a los anteriores, teniendo en cuenta que ${}^\bullet t_{go}(k,l) = {}^\bullet t \cup \{N@k\}$ y $t_{go}(k,l)^\bullet = t^\bullet \cup \{N@l\}$.

□

Corolario 7.3.4 *Los sistemas RUN centralizados simulan monótonamente a los sistemas RUN.*

Demostración. Basta ver que estamos en las hipótesis de la proposición 2.6.7. El resultado anterior nos dice que las simulan paso a paso. Además, todos los marcajes alcanzables son estables (por la misma razón que en 3.2). Por último, tanto $(\ )^*$ como su inversa son monótonas, para lo que basta ver que $(M,k)^* \sqsubseteq (M',k') \Leftrightarrow M \subseteq M'$ y $k = k' \Leftrightarrow (M,k) \sqsubseteq (M',k')$.

□

Por tanto, los sistemas centralizados pueden simular paso a paso cualquier sistema RUN, de manera que a partir de ahora usaremos los primeros para demostrar los sucesivos resultados técnicos sobre estos últimos. El anterior resultado no es válido cuando consideramos recogida de basura, que para los sistemas RUN centralizados viene dada por $A \equiv_N A + \emptyset$. En efecto, nótese que una red N sin tokens localizada en una localidad l (que es basura) es simulada por una red con un sólo token en su lugar $N@l$, y por tanto su simulación no sería considerada basura.

7.4. Verificación de sistemas RUN

Una vez extendido el modelo básico de redes ubicuas con primitivas de replicación, vamos a estudiar la potencia expresiva de los modelos resultantes, tanto para la versión sin recogida de basura como para la versión que la incluye.

Decidabilidad de la alcanzabilidad y del recubrimiento para sistemas RUN

En esta sección probaremos la decidibilidad de la alcanzabilidad y la del recubrimiento para sistemas RUN, reduciéndolas a la alcanzabilidad y al recubrimiento de sistemas MSPN con naturales, para los que sabemos que ambas propiedades son decidibles. La simulación consiste básicamente en usar una misma red para simular todas las copias de un mismo tipo, usando identificadores para distinguir entre tokens pertenecientes a redes distintas.

Antes de proceder con dicha simulación, necesitamos un par de resultados técnicos que nos serán de utilidad en las construcciones. En primer lugar, por simplicidad en las notaciones de la construcción, suponemos que toda red aparece una sola vez en el marcaje inicial, y que el marcaje inicial asociado a todas las transiciones de replicación t es seguro, es decir, tiene como mucho un token por lugar, lo que quiere decir que p aparece como mucho una vez en $\lambda(t)$. Esto tampoco supone ninguna restricción, ya que podríamos introducir en las redes replicantes un *encabezamiento*, de modo que un marcaje $\overline{M}$ dado podría ser generado partiendo de un marcaje 1-seguro. Este encabezamiento vendría dado por:

- lugares $p(1), \ldots, p(n_p)$ para cada lugar p, suponiendo que $\overline{M}(p) = n_p$,
- una transición $t_{p(i)}$ para cada uno de esos lugares,
- para cada p, arcos $(p(i), t_{p(i)})$ y $(t_{p(i)}, p)$ para $i = 1, \ldots, n_p$ y $(t_{p(i)}, p(i+1))$ para $i = 1, \ldots, n_p - 1$.

El marcaje 1-seguro es $M_0(q) = \begin{cases} 1 \text{ si } q = p(1) \text{ para algún } p \in P, \\ 0 \text{ en otro caso} \end{cases}$

Proposición 7.4.1 *Con las notaciones anteriores, si para cada marcaje M denotamos como M^* al marcaje que extiende M a los lugares auxiliares, dándoles el valor 0, entonces*

- $M_0 \rightarrow^* \overline{M}^*$,
- $\overline{M} \rightarrow^* M$ *si y sólo si* $\overline{M}^* \rightarrow^* M^*$,
- *Si M^* es alcanzable en la simulación entonces M es alcanzable.*

Demostración.

- Basta tener en cuenta que ${}^\bullet t_{p(i)} = \{p(i)\}$ y que

$$t^\bullet_{p(i)} = \begin{cases} \{p, p(i+1)\} \text{ si } i = 1, \ldots, n_p - 1 \\ \{p\} \text{ si } i = n_p, \end{cases}$$

  de modo que para cualquier ordenación $p^1, \ldots, p^m$ del conjunto de lugares podemos disparar la secuencia de transiciones

$$t_{p^1(1)}, \ldots, t_{p^1(n_{p^1})}, \ldots\ldots, t_{p^m(1)}, \ldots, t_{p^m(n_{p^m})}$$

  Cada $t^i_p(j)$ añade un token a p^i, de modo que tras el disparo de esa secuencia se obtiene un marcaje sin tokens en los lugares auxiliares y con n_{p^j} tokens en p^j. Como $n_{p^j} = \overline{M}(p^j)$ podemos concluir que el marcaje alcanzado es $\overline{M}^*$.

- Inmediato pues, por construcción, los conjuntos de lugares precondición y postcondición coinciden con los respectivos en la simulación para toda transición $t \in T$.

- Este punto se basa en el hecho de que siempre podemos reordenar los cómputos de la simulación para que las primeras transiciones que se ejecuten sean las $t(p(i))$.

  En efecto, supongamos que en un cómputo una transición t se dispara inmediatamente antes de $t_{p(n)}$. Antes de t en algún momento se ha disparado $t_{p(n-1)}$, tras lo cual $t_{p(n)}$ está siempre activada (hasta que se dispara).

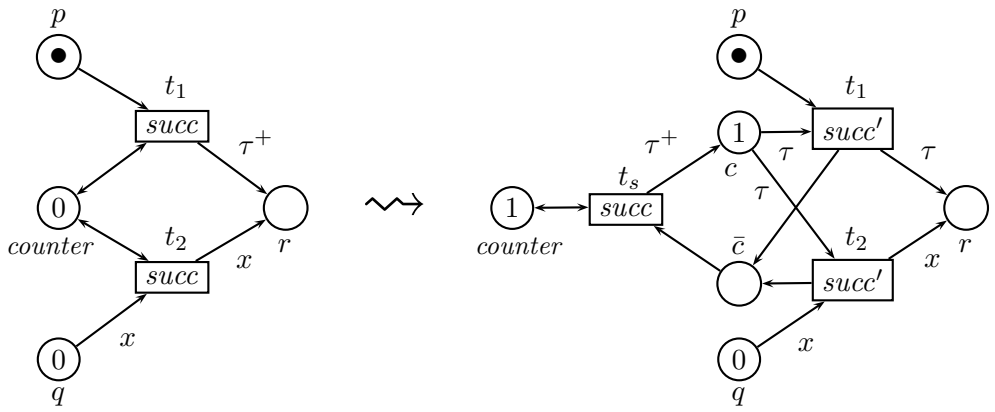

Figura 7.2: De los sistemas MSPN extendidos a los sistemas MSPN

Además, el disparo de $t_{p(n)}$ sólo elimina tokens de los lugares auxiliares y añade tokens en los lugares originales, de modo que por monotonía t también se puede disparar después de $t_{p(n)}$. Entonces podemos intercambiar el orden de disparo de las transiciones t y $t_{p(n)}$ alcanzado de las dos maneras el mismo marcaje.

Una vez que se han vaciado todos los lugares auxiliares no pueden volver a ser ocupados (ninguna transición tiene a uno de ellos como postcondición y a la vez ninguno de ellos como precondición). Además, todas las transiciones $t_{p(i)}$ (y sólo ellas) tienen a los lugares auxiliares como precondiciones. Entonces podemos suponer que las transiciones auxiliares se disparan todas antes de las originales, obteniendo justamente $\overline{M}^*$, como vimos en el primer apartado. Por lo tanto, basta ver que si $\overline{M}^* \rightarrow^* M^*$ entonces $\overline{M} \rightarrow^* M$, que se cumple por el apartado anterior.

□

A continuación, vamos a considerar una ligera versión de los sistemas MSPN, aquellos en los que permitimos que varias transiciones actúen sobre el mismo contador. Más concretamente, cada contador c podrá contar con $t_c^1, \ldots, t_c^n$ con $n > 0$ tales que cada una de ellas cumple las mismas condiciones que cumplía la única t_c que se permitía en la definición 5.1.1 y análogamente en la definición 4.4.1. Si llamamos sistemas *MSPN extendidos* a esta extensión, entonces tenemos el siguiente resultado.

Lema 7.4.2 *Todo sistema MSPN extendido puede ser simulado monótonamente por un sistema MSPN.*

Demostración. Intuitivamente, para cada componente sustituimos el contador (lugar *counter*) por un falso contador (lugar c) y todas las transiciones sucesor

por transiciones autónomas que actúan sobre el falso contador y que disparan una transición sucesor nueva, la única en la componente, que es la única que actúa sobre *counter* (véase la figura 7.2).

Dado $\mathcal{N}$ definimos $\mathcal{N}^* = \{N^* \mid N \in \mathcal{N}\}$ que lo simula, donde para cada $N = (P, T, F, \lambda)$, si $T_s = \{t \in T \mid \lambda(t) = succ\}$ entonces $N^* = (P^*, T^*, F^*, \lambda^*)$ con

- $P^* = P \cup \{c, \bar{c}\}$,
- $T^* = T \cup \{t_s\}$,
- F^* es la función parcial con menor dominio tal que:
  - $F^*(p,t) = F(p,t)$ y $F^*(t,p) = F(t,p)$ para todo $t \notin T_s \cup \{t_s\}$ y $p \notin \{counter, c, \bar{c}\}$,
  - $F^*(counter, t_s) = \tau^-$, $F^*(t_s, counter) = F^*(t_s, c) = \tau^+$ y $F^*(\bar{c}, t_s) = \varepsilon$,
  - Para cada $t \in T_S$,
    - $F^*(p,t) = \begin{cases} \tau & \text{si } p = c \\ F(p,t) & \text{si } p \neq c, \end{cases}$
    - $F^*(t,p) = \begin{cases} \tau & \text{si } F(t,p) = \tau^+ \\ F(p,t) & \text{en otro caso,} \end{cases}$
    - $F^*(t, \bar{c}) = \varepsilon$,
  - $\lambda^*(t) = \begin{cases} \lambda(t) & \text{si } t \notin T_s \cup \{t_s\} \\ succ & \text{si } t = t_s \\ succ' \in \mathcal{A} & \text{si } t \in T_s. \end{cases}$

  Además, para cada M marcaje de $\mathcal{N}$ definimos

$$M^*(p) = \begin{cases} M(p) & \text{si } p \notin \{c, \bar{c}, counter\} \\ \{(i, n+1)\} & \text{si } p \in \{c, counter\} \text{ y } M(counter) = (i,n) \\ 0 & \text{si } p = \bar{c}. \end{cases}$$

Entonces se cumple lo siguiente:

- Si $M_1[u\rangle M_2$ entonces $M_1^* \to^* M_2^*$.
- Si $M_1^* \to^* M'$ y $M'(\bar{c}) = 0$ entonces existe M_2 tal que $M_1 \to^* M_2$ y $M' = M_2^*$.

Veámoslo:

- Para u autónoma o par de sincronización es inmediato porque sus conjuntos de precondiciones y postcondiciones coinciden (así como las variables que etiquetan los respectivos arcos), y no contienen a los lugares c, $\bar{c}$ y *counter* y, por lo tanto, $M^*(p) = M(p)$ para todos esos lugares. Supongamos por lo tanto que $u = t(\sigma)$ con $\lambda(t) = succ$. Veamos entonces que $M_1^*[t(\sigma')\rangle M_2'[t_s(\sigma'')\rangle M_2^*$ con $\sigma'(x) = \begin{cases} \sigma(x) & \text{si } x \neq \tau \\ \sigma(\tau^+) & \text{si } x = \tau \end{cases}$ Como

$M_1[t(\sigma)\rangle M_2$ se cumple que $\sigma(F(p,t)) \in M_1(p)$ para todo $p \in {}^\bullet t$. En particular, si $M_1(counter) = \{(i,n)\}$ entonces $\sigma(\tau^+) = (i, n+1)$. Pues bien, para $p \in {}^\bullet t$ en $\mathcal{N}^*$ y $p \neq c$ se tiene que $\sigma'(F(p,t)) = \sigma(F(p,t)) \in M_1(p) = M_1^*(p)$. Además, $\sigma'(F(c,t)) = \sigma'(\tau) = \sigma(\tau^+) = (i, n+1) \in \{(i, n+1)\} = M_1^*(p)$. Por lo tanto, $t(\sigma)$ se puede disparar, alcanzando un marcaje M_2' que coincide con M_2^* en todos los lugares salvo en $\{c, \bar{c}, counter\}$, donde cumple que $M_2'(c) = \emptyset$, $M_2'(\bar{c}) = 1$ y $M_2'(counter) = \{(i, n+1)\}$. Entonces podemos dispara $t_s(\sigma'')$ con $\sigma''(\tau^-) = (i, n+1)$ y $\sigma''(\tau^+) = (i, n+2)$, alcanzando M_2^*, ya que como $M_2(counter) = (i, n+1)$ se tiene que $M_2^*(counter) = M_2^*(c) = \{(i, n+2)\}$.

- Veámoslo por inducción sobre la longitud del cómputo $M_1^* \rightarrow^* M'$. Si $M_1^* = M'$ entonces la tesis se sigue trivialmente con $M_2 = M_1$. Supongamos entonces que $M_1^* \rightarrow^* M'' \xrightarrow{u} M'$. Si $M''(\bar{c}) = 0$ entonces podemos aplicar directamente la hipótesis de inducción para establecer la existencia de M_2' tal que $M_1 \rightarrow^* M_2'$ y $M_2'^* = M''$. Como además u es una transición que no está en $T_s \cup \{t_s\}$ se cumple que existe M_2 con $M_2'[u\rangle M_2$ y $M_2^* = M'$.

  Queda el caso en el que $M''(\bar{c}) \neq 0$. Entonces han de existir $t \in T_s$, $t_1, \ldots, t_n \notin T_s \cup \{t_s\}$, M_1' y M_2' tales que

  $$M_1^* \rightarrow^* M_1' \xrightarrow{t} M_2' \xrightarrow{t_1} \ldots \xrightarrow{t_n} M'' \xrightarrow{t_s} M'$$

  Como todas las t_i tienen conjuntos de precondiciones y postcondiciones disjuntos a los de t_s se puede disparar t_s inmediatamente después de t, obteniendo el cómputo

  $$M_1^* \rightarrow^* M_1' \xrightarrow{t} M_2' \xrightarrow{t_s} M_2'' \xrightarrow{t_1} \ldots \xrightarrow{t_n} M'$$

  que podemos dividir en dos partes, $M_1^* \rightarrow^* M_1' \overset{t}{M}{}_2' \xrightarrow{t_s} M_2''$ y $M_2'' \xrightarrow{t_1} \ldots \xrightarrow{t_n} M'$. Como tras t_s se cumple que en $\bar{c}$ no hay tokens podemos aplicar la hipótesis de inducción al primero de los cómputos para concluir que existe M_3 tal que $M_1 \rightarrow^* M_3$ y $M_3^* = M_2''$. Ahora sí podemos aplicar la hipótesis de inducción al segundo cómputo y obtener un M_2 tal que $M_3 \rightarrow^* M_2$ y $M_2^* = M'$. Pues bien, encadenando ambos cómputos de $\mathcal{N}$ se tiene que $M_1 \rightarrow^* M_2$ y $M_2^* = M'$, como queríamos demostrar.

Los puntos anteriores nos dicen que N^* simula a N. Además, $(\ )^*$ y su inversa son trivialmente monótonas porque son la identidad para $p \in P$ y todos los marcajes estables coinciden en el resto de lugares. Por último, existe el lugar c, que es seguro, que cumple que todo marcaje de $\overline{M}$ de N^* es estable si y sólo si $\overline{M}(c) \neq \emptyset$, por lo que no pueden existir marcajes inestables alcanzables mayores que uno estable.

□

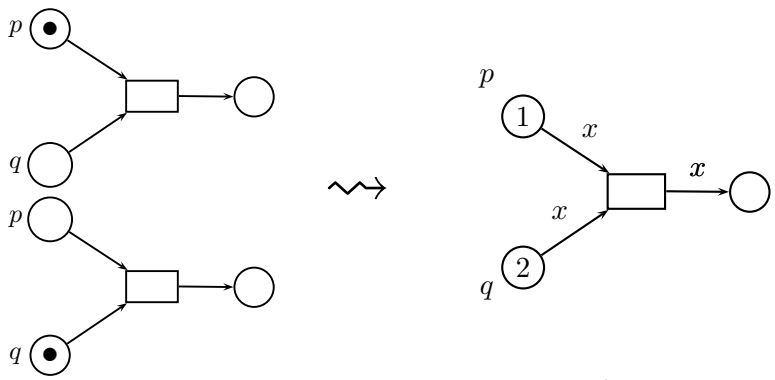

Figura 7.3: Transiciones no activadas

Ya estamos en condiciones de establecer la simulación de redes RUN por medio de sistemas MSPN con identificadores naturales. Sea $\mathcal{N} = \{N_1, \ldots, N_m\}$ un sistema RUN, que puede ser centralizado en virtud del corolario 7.3.4. Como hemos dicho antes, usaremos cada $N \in \mathcal{N}$ para simular el comportamiento de todas las copias de tipo N que se creen a lo largo de la historia del sistema. Para distinguir entre las distintas copias del mismo tipo usaremos tokens de la forma (i, n), de modo que un token (i, n) representa un token de la n-ésima copia de N_i. Para evitar confusión entre distintas copias de la misma red, etiquetamos todos los arcos de la red con una única variable x, pero usando una variable distinta en cada red, de modo que todos los tokens que se mueven de las precondiciones a las postcondiciones de cualquier transición *pertenecen* a la misma copia de la red. Por ejemplo, la red a la derecha de la figura 7.3 se corresponde con el sistema de la izquierda de la misma figura, que tiene dos copias de la misma red, una con un token en p y otra con un token en q.

Esta simple construcción no funcionaría si hubiese transiciones sin precondiciones, ya que no podemos usar el token que se encuentra en la precondición para identificar a la red que está disparando la transición. Sin embargo, no sería difícil modificarla para que también funcionase en ese caso. Bastaría con añadir un lugar extra para guardar todos los tokens identificador usados hasta el momento, y hacer a este lugar tanto precondición como postcondición de todas las transiciones que no tienen originalmente precondiciones.

Simulamos la creación de una nueva red por medio de la generación de un nuevo token identificador (i, n), lo que se consigue por medio de transiciones sucesor (véase la figura 7.4). Las transiciones de replicación las simulan transiciones sucesor (podemos contar con varias por la proposición anterior) que colocan el nuevo identificador creado en todos los lugares $\lambda(t)$, creando el marcaje inicial de la red creada (podemos suponer que $\lambda(t)$ es seguro por la proposición 7.4.1).

Para formalizar estas ideas veamos una definición auxiliar.

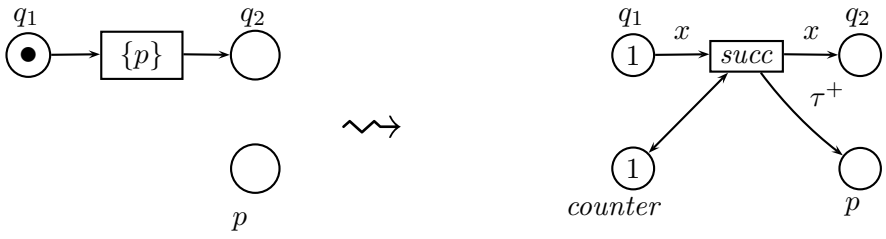

Figura 7.4: Simulación de transiciones de replicación

Definición 7.4.3 *Sea $\mathcal{M}$ un marcaje de un sistema RUN $\mathcal{N} = \{N_i \mid i = 1, \ldots, n\}$. Si denotamos por n_i al número de marcajes de N_i en $\mathcal{M}$ entonces diremos que una biyección $\theta : \{(i,j) \mid i = 1, \ldots, n,\ j = 1, \ldots, n_i\} \to \mathcal{M}$ es una* ordenación *de $\mathcal{M}$ si $\theta(i,j)$ es un marcaje de N_i para cada (i,j).*

Definición 7.4.4 *Sea $\mathcal{N}$ un sistema RUN centralizado. Si para cada $N_i = N \in \mathcal{N}$ elegimos una variable distinta $x_N \in Var$, definimos $\mathcal{N} = \{N^* \mid N \in \mathcal{N}\}$, donde para cada $N = (P, T, F, \lambda) \in \mathcal{N}$ tomamos $N^* = (P^*, T^*, F^*, \lambda^*)$, con*

- $P^* = P \cup \{counter_i\}$,
- $T^* = T$,
- *F^* es la función parcial con menor dominio tal que*
  - *$F^*(p,t) = x_N$ para todo $(p,t) \in F$,*
  - *$F^*(t,p) = x_N$ para todo $(t,p) \in F$,*
  - *Para todo t tal que $\lambda(t) \in \mathcal{MS}(P)$, $F^*(counter_i, t) = \tau^-$, $F^*(t, counter_i) = \tau^+$ y $F^*(t,p) = \tau^+$ para todo $p \in \lambda(t)$,*
- $\lambda^*(t) = \begin{cases} \lambda(t) & si\ \lambda(t) \in \mathcal{A} \cup Sync \\ succ & si\ \lambda(t) \in \mathcal{MS}(P). \end{cases}$

Además, para cada ordenación θ de un marcaje $\mathcal{M}$ de $\mathcal{N}$ definimos el marcaje $\mathcal{M}^\theta$ de $\mathcal{N}^*$ de la siguiente manera:

- *Para $p \in P$, $\mathcal{M}^\theta(p)(i,j) = \begin{cases} \theta(i,j)(p) & para\ i = 1, \ldots, n\ y\ j = 1, \ldots, n_i \\ 0 & en\ otro\ caso, \end{cases}$*
- $\mathcal{M}^\theta(counter_i) = (i, n_i + 1)$.

Proposición 7.4.5 *Se cumplen las siguientes propiedades:*

- *Si $\mathcal{M}_1 \to \mathcal{M}_2$ entonces para cada θ_1 existe θ_2 tal que $\mathcal{M}_1^{\theta_1} \to \mathcal{M}_2^{\theta_2}$.*
- *Si $\mathcal{M}_0^{\theta_0} \to M$ entonces existen $\mathcal{M}$ y θ tales que $\mathcal{M}_0 \to \mathcal{M}$ y $\mathcal{M}^\theta = M$.*

Demostración.

- Veamos que si $\mathcal{M}_1[u\rangle\mathcal{M}_2$ entonces existe θ_2 tal que $\mathcal{M}_1^{\theta_1} \to \mathcal{M}_2^{\theta_2}$. Lo hacemos distinguiendo los tres posibles casos para u. Si $u = t(M)$ con $\lambda(t) \in \mathcal{A}$ entonces $\mathcal{M}_2 = \mathcal{M}_1 - \{M\} + \{M'\}$ con $M' = M - {}^\bullet t + t^\bullet$. Veamos entonces que $\mathcal{M}_1^{\theta_1}[t(\sigma)\rangle\mathcal{M}_2^{\theta_2}$ donde $\sigma(x_N) = \theta_1^{-1}(M)$ y θ_2 es la ordenación que coincide con θ_1 excepto en $\theta_1^{-1}(M)$, donde vale M'. Efectivamente, si $p \in {}^\bullet t$ entonces $\mathcal{M}_1^{\theta_1}(p)(\sigma(F^*(p,t))) = \theta_1(\sigma(x_N))(p) = M(p) > 0$, esto último porque $p \in {}^\bullet t \subseteq M$. Entonces $t(\sigma)$ se puede disparar, alcanzando un marcaje $\overline{M}$. Veamos que $\overline{M} = \mathcal{M}_2^{\theta_2}$. En primer lugar, $\overline{M}(p)(\theta_1^{-1}(M)) = \mathcal{M}_1^{\theta_1}(p)(\theta_1^{-1}(M)) - \{\sigma(F^*(p,t))\}(\theta_1^{-1}(M)) + \{\sigma(F^*(t,p))\}(\theta_1^{-1}(M)) = M(p) - F(p,t) + F(t,p) = M'(p) = \theta_2(\theta_1^{-1}(M))(p) = \mathcal{M}_2^{\theta_2}(p)(\theta_1^{-1}(M))$. Además, para $a \neq \theta_1^{-1}(M)$, $\overline{M}(p)(a) = \mathcal{M}_1^{\theta_1}(p)(a) = \theta_1(a)(p) = \theta_2(a)(p) = \mathcal{M}_2^{\theta_2}(p)(a)$ y podemos concluir que $\mathcal{M}_1^{\theta_1}[t(\sigma)\rangle\mathcal{M}_2^{\theta_2}$.

  En el caso $u = (t_1(M_1), t_2(M_2))$ se ve análogamente que $\mathcal{M}_1^{\theta_1}[(t_1,t_2)(\sigma)\rangle\mathcal{M}_2^{\theta_2}$ con $\sigma(x_{N_i}) = \theta_1^{-1}(M_i)$ para $i = 1,2$, y tomando θ_2 como la ordenación que coincide con θ_1 excepto en $\theta_1^{-1}(M_i)$, donde vale $M_i' = M_i - {}^\bullet t_i + t_i^\bullet$ para $i = 1,2$.

  El último caso es $u = t(M)$ con $\lambda(t) \in \mathcal{MS}(P)$. Supongamos que t es una transición de la componente N_i. Ahora $\mathcal{M}_2 = \mathcal{M}_1 - \{M\} + \{M', \lambda(t)\}$ con $M' = M - {}^\bullet t + t^\bullet$. Veamos que $\mathcal{M}_1^{\theta_1}[t(\sigma)\rangle\mathcal{M}_2^{\theta_2}$ con $\sigma(x_N) = \theta_1^{-1}(M)$, $\sigma(\tau^-) = (i, n_i)$ y $\sigma(\tau^+) = (i, n_i+1)$, y θ_2 es la ordenación que coincide con θ_1 excepto en $\theta_1^{-1}(M)$, donde vale M', y la extiende con $\theta_2(i, n_i+1) = \lambda(t)$. Para t se cumple que ${}^\bullet t = \{p \mid (p,t) \in F\} \cup \{c\}$. Para p tal que $(p,t) \in F$ se ve igual que antes que $\sigma(F^*(p,t)) \in \mathcal{M}_1^{\theta_1}(p)$. Además, por definición de $\mathcal{M}_1^{\theta_1}$ tenemos que $\sigma(F^*(counter_i, t)) = (i, n_i+1) \in \mathcal{M}_1^{\theta_1}(counter_i) = \{(i, n_i+1)\}$ y la transición $t(\sigma)$ está activada. Para todo $a \neq (i, n_i+1)$ se ve igual que en el primer apartado que el marcaje $\overline{M}$ alcanzado coincide con $\mathcal{M}_2^{\theta_2}$. Por construcción, para $p \neq counter_i$ se tiene que $\overline{M}(p)(i, n_i+1) = \lambda(t)(p) = \theta_2(i, n_i+1)(p) = \mathcal{M}_2^{\theta_2}(p)(i, n_i+1)$. Por último, claramente $\overline{M}(counter_i)(i, n_i+1) = 0 = \mathcal{M}_2^*\theta_2(counter_i)$ y $\overline{M}(counter_i)(i, n_i+2) = 1 = \mathcal{M}_2^*\theta_2(counter_i)$, por lo que podemos concluir que $\mathcal{M}_1^{\theta_1}[t(\sigma)\rangle\mathcal{M}_2^{\theta_2}$.

- Veamos ahora que si $\mathcal{M}^\theta[u\rangle M'$ entonces existen θ' y $\mathcal{M}'$ tales que $\mathcal{M}[v\rangle\mathcal{M}'$ y $M' = \mathcal{M}'^{\theta'}$. Como es habitual, distinguimos los posibles casos para u. Supongamos primero que $u = t(\sigma)$ con $\lambda(t) \in \mathcal{A}$, es decir, que $M'(p) = M(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$. Denotemos $a = \sigma(x_N)$ si $Vars(t) = \{x_N\}$. Veamos entonces que se puede disparar $t(\theta(a))$. En primer lugar, $\theta(a)(p) = \mathcal{M}^\theta(p)(a) > 0$ para todo $p \in {}^\bullet t$ porque $\sigma(F(p,t)) = \sigma(x_N) = a \in \mathcal{M}^\theta(p)$ para todos esos lugares p. Entonces la transición se puede disparar, obteniendo $\mathcal{M}' = \mathcal{M} - \{\theta(a)\} + \{\overline{M}\}$ con $\overline{M} = \theta(a) - {}^\bullet t + t^\bullet$. Si definimos $\theta'(b) = \begin{cases} \overline{M} \text{ si } b = a \\ \theta(b) \text{ si } b \neq a \end{cases}$ sólo falta ver que $\mathcal{M}'^{\theta'} = M'$. Pues bien, $\mathcal{M}'^{\theta'}(p)(a) = \theta'(a)(p) = \overline{M}(p) = \theta(a)(p) - F(p,t) + F(t,p) = \mathcal{M}^\theta(p)(a) - F(p,t) + F(t,p) = M'(p)(a)$. Por último, para cualquier $b \neq a$ se tiene que

$\mathcal{M}'^{\theta'}(p)(b) = \theta'(b)(p) = \theta(b)(p) = \mathcal{M}^{\theta}(p)(b) = M'(p)(b) = M(p)(b)$ y la tesis se sigue.

El caso $u = (t_1(M_1), t_2(M_2))$ es completamente análogo al anterior. Veamos el caso $u = t(\sigma)$ con $\lambda(t) = succ$, $\sigma(x_N) = a$, $\sigma(\tau^-) = (i, n_i + 1)$ y $\sigma(\tau^+) = (i, n_i + 2)$. Ahora se puede disparar $t(\theta(a))$ para obtener $\mathcal{M}' = \mathcal{M} - \theta(a) + \{\overline{M}, \lambda(t)\}$ con $\overline{M} = \theta(a) - {}^{\bullet}t + t^{\bullet}$. Definamos

$$\theta'(b) = \begin{cases} \overline{M} & \text{si } b = a \\ \lambda(t) & \text{si } b = (i, n_i + 1) \\ \theta(b) & \text{en otro caso.} \end{cases}$$

Para $b = a$ y $b \notin \{a, (i, n_i+1)\}$ se ve exactamente igual que en el apartado anterior que $\mathcal{M}'^{\theta'}$ y M' coinciden. Además, para $p \neq counter_i$, $\mathcal{M}'^{*}\theta'(p)(i, n_i + 1) = \theta'(i, n_i+1)(p) = \lambda(t)(p) = M'(p)(i, n_i+1)$ porque por construcción los únicos lugares postcondición de t (distintos de $counter_i$) etiquetados con τ^+ son los de $\lambda(t)$. Por último, para $counter_i$ se cumple que tanto el marcaje $\mathcal{M}'^{\theta'}$ como M' contienen $(i, n_i + 2)$, el primero por definición de $\mathcal{M}'^{\theta'}$ y el segundo porque $\mathcal{M}^{\theta}(counter_i) = \{(i, n_i + 1)\}$ y $\lambda^*(t) = succ$, y por lo tanto $M' = \mathcal{M}'^{\theta'}$ y la tesis se sigue.

□

Como detalle importante, en nuestra simulación el valor del contador nos dice cuántas copias de cada red han sido creadas desde el marcaje inicial. Esto no es ningún problema, ya que los estados de los sistemas RUN también explicitan el número de copias que se han creado. Sin embargo, nuestra simulación también recuerda el orden en el que las distintas copias han sido creadas, cosa que no se recuerda necesariamente en el sistema original. Adelantando acontecimientos, siguiendo la terminología del próximo capítulo, la proposición anterior prueba que los sistemas MSPN son un etiquetado de los sistemas RUN sin basura, donde las etiquetas θ recuerdan el orden en el que se generan las componentes.

Sin embargo, dado un marcaje cuya alcanzabilidad (o recubrimiento) queremos decidir, existe una cantidad finita de órdenes en los que se han generado las distintas copias. Por tanto, como la alcanzabilidad y el recubrimiento son decidibles para sistemas MSPN tenemos la siguiente

Corolario 7.4.6 *La alcanzabilidad y el recubrimiento son decidibles para sistemas RUN.*

Demostración. Sea $\mathcal{M}$ el marcaje cuya alcanzabilidad (recubrimiento) queremos decidir. Por la proposición anterior, $\mathcal{M}$ se puede alcanzar (cubrir) si y sólo si existe una ordenación θ tal que $\mathcal{M}^{\theta}$ se pueda alcanzar (cubrir). Para concluir basta tener en cuenta que la alcanzabilidad (recubrimiento) es decidible para los sistemas MSPN con identificadores naturales y que para un marcaje sólo existen finitas ordenaciones distintas.

□

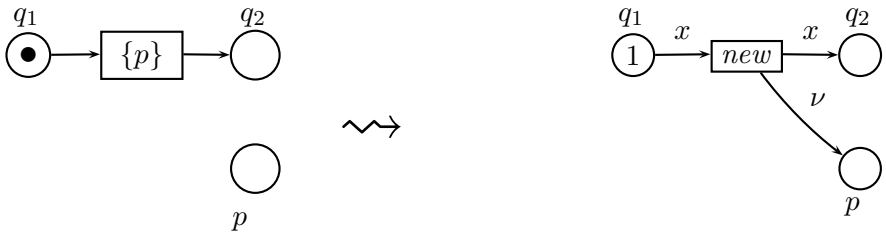

Figura 7.5: Simulación de transiciones de replicación con recogida de basura

Poder expresivo de los sistemas RUN con recogida de basura

La simulación vista en el apartado anterior deja de funcionar cuando consideramos la versión de los sistemas RUN que incorpora recogida de basura. En ella usábamos identificadores naturales para distinguir entre la aparición de distintas copias del mismo tipo. En particular, toda red tenía su lugar contador, que cuenta cuántas redes de ese tipo han sido creadas (y el orden en el que se han creado). En ausencia de recolección de basura ésta era una propiedad deseable, ya que esa información es parte de los marcajes, porque las redes se creaban, pero nunca se destruían.

Sin embargo, cuando introducimos el mecanismo de recogida de basura, los marcajes ya no contienen necesariamente información acerca de cuántas componentes han sido creadas, ya que toda red no marcada va a ser eliminada. Necesitamos por ello una manera más abstracta de simular los sistemas RUN con recogida de basura, que olvide, por decirlo de algún modo, el número de redes que han sido creadas hasta la fecha.

Una manera de conseguirlo es usar identificadores abstractos, en lugar de identificadores naturales. Todo lo que necesitamos para simular correctamente los sistemas RUN con recogida de basura es distinguir entre las distintas copias de la misma red, para lo que basta usar tal tipo de identificadores. La construcción es exactamente igual a la del apartado anterior, excepto por el hecho de que en lugar de transiciones sucesor usamos transiciones *new* y, en particular, ya no contamos con los lugares contador (véase la figura 7.5). La simulación funciona de la misma manera, excepto por el hecho de que cuando un identificador desaparece del sistema puede ser reutilizado, obteniendo así el mecanismo de recolección de basura deseado.

Además, nuestra simulación con identificadores abstractos tampoco recuerda el orden en el que se generaron las copias de las redes, de modo que no es necesario buscar entre los distintos órdenes como hacíamos en el apartado anterior.

Proposición 7.4.7 *Todo sistema RUN centralizado con recogida de basura se puede simular monótonamente con un sistema MSPN con identificadores abstractos.*

Demostración. Dado un sistema RUN $\mathcal{N}$ definamos $\mathcal{N}^* = \{N^* \mid N \in \mathcal{N}\}$. Para cada $N \in \mathcal{N}$ elegimos una variable $x_N \in \mathit{Var}$. Si $N = (P, T, F, \lambda)$ entonces $N^* = (P, T, F^*, \lambda^*)$, donde

- F^* es la función parcial con menor dominio tal que
  - $F^*(p,t) = x_N$ para todo $(p,t) \in F$,
  - $F^*(t,p) = x_N$ para todo $(t,p) \in F$,
  - $F^*(t,p) = \nu$ para todo t y p tales que $p \in \lambda(t) \in \mathcal{MS}(P)$.
- $\lambda^*(t) = \begin{cases} \lambda(t) & \text{si } \lambda(t) \notin \mathcal{MS}(P) \\ rep \in \mathcal{A} & \text{si } \lambda(t) \in \mathcal{MS}(P) \end{cases}$

Además, para cada marcaje $\mathcal{M}$ de $\mathcal{N}$ definimos el marcaje $\mathcal{M}^*$ de $\mathcal{N}^*$ de la siguiente manera. Para cada $M \in \mathcal{M}$ elegimos un identificador distinto η. Diremos que η representa a M y escribiremos M_η para referirnos a la componente representada por η. Entonces $\mathcal{M}^*(p)(\eta) = M_\eta(p)$ para todo η que represente a alguna componente.

Veamos primero que si $\mathcal{M}[u\rangle\mathcal{M}'$ entonces $\mathcal{M}^*[v\rangle\mathcal{M}'^*$. Si $u = t(M_\eta)$ con $\lambda(t) \in \mathcal{A}$ se cumple que $\mathcal{M}^*[t(\sigma)\rangle\mathcal{M}'^*$ con $\sigma(x_N) = \eta$, si t es una transición de N. Efectivamente, $\mathcal{M}^*(p)(\eta) = M_\eta(p) > 0$ si $p \in {}^\bullet t$ y $\mathcal{M}^*[t(\sigma)\rangle\overline{M}$. Veamos que $\overline{M} = \mathcal{M}_2^*$ con el mismo conjunto de representantes que $\mathcal{M}_1^*$. Por definición de disparo de transiciones en sistemas RUN se tiene que $\mathcal{M}_2 = \mathcal{M}_1 - \{M_\eta\} + \{M'_\eta\}$, donde $M'_\eta = M_\eta - {}^\bullet t + t^\bullet$. Para $\eta' \neq \eta$ se cumple que $\overline{M}(p)(\eta') = \mathcal{M}_1^*(p)(\eta') = M_{\eta'}(p) = \mathcal{M}_2^*(p)(\eta')$, pues $M_{\eta'}$ pertenece tanto a $\mathcal{M}_1$ como a $\mathcal{M}_2$. Sin embargo, para η tenemos que $\overline{M}(p)(\eta) = \mathcal{M}_1^*(p)(\eta) - F(p,t) + F(t,p) = M_\eta(p) - F(p,t) + F(t,p) = M'_\eta(p) = \mathcal{M}^*(p)(\eta)$.

El caso $u = (t_1(M_{\eta_1}), t_2(M_{\eta_2}))$ es completamente análogo al anterior, sin más que considerar simultáneamente ambas transiciones. Veamos el último caso, cuando $u = t(M_\eta)$ con $\lambda(t) \in \mathcal{MS}(P)$. En este caso $\mathcal{M}_2 = \mathcal{M}_1 - \{M_\eta\} + \{M'_\eta, \lambda(t)\}$ con $M'_\eta = M_\eta - {}^\bullet t + t^\bullet$. Por el mismo motivo que en el caso previo, $t(M_\eta)$ se puede disparar en $\mathcal{M}_1^*$ en modo σ, con $\sigma(x_N) = \eta$ y $\sigma(\nu) = \eta_f$, produciendo un marcaje $\overline{M}$. Veamos que $\overline{M} = \mathcal{M}_2^*$ tomando como representante de la componente $\lambda(t)$ al identificador η_f. El flujo de tokens de las precondiciones a la postcondiciones de t se simula de manera idéntica al caso de la transición autónoma, de modo que $\overline{M}(p)(\eta') = \mathcal{M}_2^*(p)$ para todo $\eta' \neq \eta_f$. Además, $\overline{M}(p)(\eta_f) = \lambda(t)(p)$ porque $F(t,p) = \nu$ si y sólo si $p \in \mathcal{MS}(p)$. Además, por definición de $\mathcal{M}_2^*$, y tomando η_f como representante de $\lambda(t)$, se tiene que $\mathcal{M}^*(p)(\eta_f) = \lambda(t)(p)$, con lo que hemos probado que $\overline{M} = \mathcal{M}_2^*$.

Veamos ahora el recíproco. Más exactamente, se cumple que si $\mathcal{M}^*[u\rangle\overline{M}$ entonces existe $\mathcal{M}'$ tal que $\mathcal{M}[v\rangle\mathcal{M}'$ y $\mathcal{M}'^* = \overline{M}$. Supongamos primero que $u = t(\sigma)$ con $\lambda^*(t) \neq rep$. Por construcción de $\mathcal{N}^*$ se cumple que $Dom(\sigma) = x$. Sea M la componente de $\mathcal{M}$ representada por $\sigma(x) = \eta$ y veamos que entonces $\mathcal{M}[t(M)\rangle\mathcal{M}'$ y $\mathcal{M}'^* = \overline{M}$. Que η represente a M quiere decir que $\mathcal{M}^*(p)(\eta) = M(p)$. Como $\sigma(F(p,t)) = \sigma(x) = \eta \in \mathcal{M}^*(p)$ para todo $p \in {}^\bullet t$ por estar $t(\sigma)$ activada en $\mathcal{M}^*$ tenemos que $M(p) > 0$ para todo $p \in {}^\bullet t$ y $t(M)$ se puede disparar, obteniendo $\mathcal{M}' = \mathcal{M} - \{M\} + \{M'\}$, donde $M' = M - {}^\bullet t + t^\bullet$. Teniendo en cuenta que $\{\sigma(F(p,t))\}(\eta) = F(p,t)$ (ambos valen 1 si $(p,t) \in F$ y 0 en caso contrario) y que

$\{\sigma(F(p,t))\}(\eta) = F(p,t)$ (análogamente con (t,p)), observamos que $\mathcal{M}'^*(p)(\eta) = M'(p) = M(p) - F(p,t) + F(t,p) = M(p) - \{\sigma(F(p,t))\}(\eta) + \{\sigma(F(t,p))\}(\eta) = \mathcal{M}^*(p)(\eta) - \{\sigma(F(p,t))\}(\eta) + \{\sigma(F(t,p))\}(\eta) = \overline{M}(p)(\eta)$, lo que nos permite concluir que $\mathcal{M}'^* = \overline{M}$, pues para $\eta' \neq \eta$ claramente $\mathcal{M}'^*(p)(\eta') = \mathcal{M}^*(p)(\eta') = \overline{M}(p)(\eta')$. El resto de los casos son análogos a este. En el caso de $\lambda^*(t) = rep$ basta tomar como representante de $\lambda(t)$ al identificador $\sigma(\nu)$.

Así que $\mathcal{N}^*$ simula paso a paso a $\mathcal{N}$ y todo marcaje alcanzable es estable. Por último, para poder aplicar la proposición 2.6.7, veamos que tanto $(\ )^*$ como su inversa son monótonas: Si $\mathcal{M} \sqsubseteq \mathcal{M}'$ podemos elegir $\eta_1, \ldots, \eta_m$ tales que $\mathcal{M} = \{M_{\eta_1}, \ldots, M_{\eta_m}\}$ y $\mathcal{M}' = \{M'_{\eta_1}, \ldots, M'_{\eta_m}, M'_{m+1}, \ldots, M'_r\}$ con $M_{\eta_i} \subseteq M'_{\eta_i}$ para todo $i = 1, \ldots, m$. Entonces $\mathcal{M}^*(p)(\eta_i) = M_{\eta_i}(p) \leq M'_{\eta_i}(p) = \mathcal{M}'(p)(\eta_i)$ y $\mathcal{M}^*(p)(\eta) = 0 \leq \mathcal{M}'(p)(\eta)$ para $\eta \notin \{\eta_1, \ldots, \eta_m\}$, por lo que $\mathcal{M}^* \sqsubseteq_\alpha \mathcal{M}'^*$. Recíprocamente, si $\mathcal{M}^* \sqsubseteq_\alpha \mathcal{M}'^*$ entonces existe una biyección $h : S(\mathcal{M}^*) \to S(\mathcal{M}'^*)$ tal que $\mathcal{M}^*(p)(\eta) \leq \mathcal{M}'^*(p)(h(\eta))$ (véase el lema 6.2.2). Entonces si η representa a M_η y $h(\eta)$ representa a $M'_{h(\eta)}$ se tiene que $M_\eta \subseteq M'_{h(\eta)}$ y la tesis se sigue.

□

La simulación anterior sólo nos daría como corolario la decidibilidad del recubrimiento para sistemas RUN centralizados. Sin embargo, esta también se tiene en general.

Proposición 7.4.8 *El recubrimiento es decidible para sistemas RUN con recogida de basura.*

Demostración. Basta tener en cuenta que un marcaje $\mathcal{M}$ de un sistema g-RUN se puede cubrir si y sólo si se puede cubrir en el correspondiente sistema RUN que no considera basura, y en la anterior sección veíamos que el recubrimiento es decidible para los sistemas RUN.

□

Sin embargo, la alcanzabilidad es indecidible para los sistemas MSPN con identificadores abstractos, así que la anterior simulación no nos dice nada acerca de la decidiblidad de la alcanzabilidad para sistemas RUN con recogida de basura. Sin embargo, veremos que una simulación a la inversa también es posible, es decir, se pueden usar los sistemas RUN con recogida de basura para simular sistemas MSPN con identificadores abstractos, de modo que ambos mecanismos (la creación de nombres y la replicación) son completamente equivalentes.

Para probar la proposición 7.4.16 vamos a definir una pequeña variante de los sistemas RUN, en la que permitiremos que las sincronizaciones no ocurran sólo entre dos componentes, sino entre un número predeterminado de componentes, posiblemente mayor que dos. En la siguiente definición, suponemos que existe una función $arity : S \to \mathbb{N}$ definida sobre el conjunto de nombres de servicios y $Sync = \{s(i) \mid s \in \mathcal{S},\ 1 \leq i \leq arity(s)\}$. Identificamos las transiciones autónomas con los servicios de aridad uno.

Definición 7.4.9 *Una* red RUN multiparte *(m-RUN) es una tupla* $N = (P, T, F, \lambda)$ *tal que* P *y* T *son conjuntos disjuntos,* $F : (P \times T) \cup (T \times P) \to Var$ *y* $\lambda : T \to Sync \cup \mathcal{MS}(P)$*. Un marcaje de* N *es un multiconjunto de lugares de* P *y denotamos* $Markings(N)$ *al conjunto de marcajes de* N*.*

Las definiciones de sistema, marcaje de un sistema son lo que uno espera.

Definición 7.4.10 *Un* sistema m-RUN $\mathcal{N}$ *es un conjunto de redes m-RUN y un marcaje de* $\mathcal{N}$ *es un multiconjunto de marcajes de sus redes. Llamamos* $Markings(\mathcal{N})$ *al conjunto de marcajes de* $\mathcal{N}$*.*

La definición de activación y disparo de transiciones de replicación es idéntica que para los sistemas RUN. A continuación definimos la noción de compatibilidad de transiciones de sincronización y activación y disparo de una tupla de transiciones compatibles.

Definición 7.4.11 *Sea un sistema m-RUN* $\mathcal{N}$ *y* $s \in S$ *con* $arity(s) = n$*. Diremos que las transiciones* $\overline{t} = (t_1, \ldots, t_n)$ *son* compatibles *para* s *si* $\lambda(t_i) = s(i)$ *para todo* $i \in \{1, \ldots, n\}$*. Llamaremos* modo *de* $\overline{t}$ *a cualquier función* $\sigma : \overline{t} \to Markings(\mathcal{N})$ *tal que si* t *es una transción de* $N \in \mathcal{N}$ *entonces* $\sigma(t) \in Markings(N)$*.*

Escribiremos ${}^\bullet\overline{t} = \sum_{i=1}^{n} {}^\bullet t_i$ y $\overline{t}^\bullet = \sum_{i=1}^{n} t_i^\bullet$.

Definición 7.4.12 *Dados un sistema m-RUN* $\mathcal{N}$*,* $\mathcal{M}$ *un marcaje de* $\mathcal{N}$ *y* $\overline{t}$ *una tupla de transiciones compatibles diremos que* $\overline{t}$ *está activada en* $\mathcal{M}$ *con modo* σ *si:*

- $\sigma(\overline{t}) \subseteq \mathcal{M}$,
- $\sum_{\sigma(t)=M} {}^\bullet t \subseteq M$ *para todo* $M \in \mathcal{M}$*.*

En tal caso, $\overline{t}$ *se puede disparar en modo* σ*, alcanzando el marcaje*

$$\mathcal{M}' = \mathcal{M} - \sigma(\overline{t}) + \{\overline{M} \mid M \in \sigma(\overline{t})\},$$

donde para cada $M \in \sigma(\overline{t})$ *tomamos* $\overline{M} = M - \sum_{\sigma(t)=M} {}^\bullet t + \sum_{\sigma(t)=M} t^\bullet$*.*

Intuitivamente, una tupla de n transiciones compatibles etiquetada por un nombre de servicio s de aridad n se puede disparar si cada una de ellas está activada según la regla ordinaria de disparo. Escribiremos $M \xrightarrow{t_1(M_1),\ldots,t_n(M_n)} M'$ si M' es el marcaje alcanzado tras el disparo de $\overline{t}$ en el marcaje M con modo $\sigma(t_i) = M_i$. También podemos considerar el mecanismo de recogida de basura para los m-RUN, obteniendo así los sistemas gm-RUN.

Veamos que el hecho de permitir sincronizaciones multiparte no introduce capacidad expresiva extra, ya que son sincronizaciones acotadas entre un número prefijado de componentes, y no se trata de una primitiva de difusión como las discutidas en [EFM99].

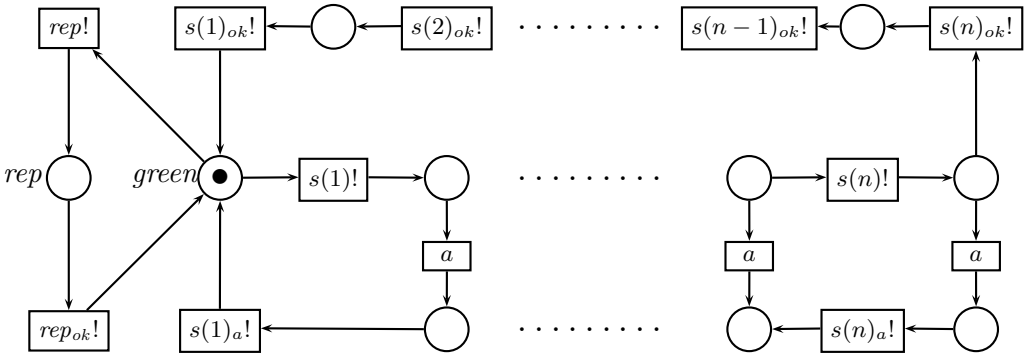

Figura 7.6: Monitor

Lema 7.4.13 *Todo sistema m-RUN se puede simular monótonamente con un sistema RUN. Todo sistema gm-RUN se puede simular monótonamente con un sistema g-RUN.*

Demostración. La demostración no depende el hecho de que estemos considerando basura o no. Para un k fijo, podemos simular una sincronización entre k partes por medio de una secuencia de sincronizaciones ordinarias. Como es usual para este tipo de protocolos, necesitamos un monitor (véase la figura 7.6) que nos proporcione exclusión mutua cada vez que uno de los k participantes comience el protocolo, de modo que cuando se esté a la mitad del protocolo no se pueda interferir con las componentes que han intervenido hasta entonces, hasta que no se acabe de ejecutar el protocolo. Dicho protocolo consiste en la sincronización sucesiva del monitor con los k participantes, marcando un lugar red_t en cada uno de ellos. Si se puede sincronizar con todos ellos entonces la fase de finalización del protocolo consiste únicamente en eliminar el token de cada lugar red_t. Si, por el contrario, el protocolo no se puede finalizar (porque algunos de los participantes no están preparados) entonces se puede abortar el protocolo, deshaciendo las sincronizaciones ya hechas.

El monitor contará con un lugar *green*, marcado únicamente cuando no se esté a medias de la ejecución del protocolo (si y sólo si ningún red_t lo está).

Formalmente, dado un sistema m-RUN $\mathcal{N}$ definamos un sistema $\mathcal{N} = \{N^* \mid N \in \mathcal{N}\} \cup \{Mon\}$ que lo simula. La componente *Mon* jugará el papel del monitor citado previamente. Si $N = (P, T, F, \lambda)$ entonces $N^* = (P^*, T^*, F^*, \lambda^*)$, donde

- $P^* = P \cup \{red_t \mid t \in T\} \cup \{rep_t \mid \lambda(t) \in \mathcal{MS}(P)\}$,
- $T^* = T \cup \{t(a), t(ok) \mid \lambda(t) \in Sync\} \cup \{t(rep), t(ok) \mid \lambda(t) \in \mathcal{MS}(P)\}$,
- $F^* = F \cup \{(t(a), p) \mid \lambda(t) \in Sync,\ p \in {}^\bullet t\} \cup$
  $\{(red_t, t(ok)), (red_t, t(a)) \mid \lambda(t) \in Sync\} \cup$

$$\{(t, red_t) \mid t \in T\} \cup \{(red_t, t(rep)), (rep_t, t(ok)) \mid \lambda(t) \in \mathcal{MS}(P)\},$$

- Para cada $t \in T$ con $\lambda(t) = s(i)$, $\lambda^*(t) = s(i)?$, $\lambda^*(t(ok)) = s(i)_{ok}?$ y $\lambda^*(t(a)) = s(i)_a?$.

- Para cada $t \in T$ con $\lambda(t) \in \mathcal{MS}(P)$, $\lambda^*(t) = rep?$, $\lambda^*(t(ok)) = rep_{ok}?$ y $\lambda^*(t(rep)) = \lambda(t)$.

Nótese que cada N^* es una extensión sintáctica de N, así que se puede interpretar que le estamos añadiendo a cada N un controlador, y todos esos controladores están a su vez controlados globalmente por el monitor, que pasamos a definir formalmente. Si $\overline{S}$ es el conjunto de nombres de servicio en $\mathcal{N}$, es decir, $\overline{S} = \{s \in S \mid \lambda(t) = s(i)$ para algún t de $\mathcal{N}\}$ definimos $Mon = (P_M, T_M, F_M, \lambda_M)$, donde

- $P_M = \{green\} \cup \{p_{s(i)}, a_{s(i)}, ok_{s(i)} \mid s \in S,\ i = 1, \ldots, arity(s)\} \cup \{rep\}$,

- $T_M = \{s(i), s(i)_{ok}, s(i)_a, s(i)_q \mid s \in S,\ i = 1, \ldots, arity(s)\} \cup$
  $\{rep(do), rep(ok)\}$,

- $F_M = \bigcup_{s \in \overline{S}} F_s$ donde si $arity(s) = n$ entonces
  $F_s = \{(green, s(1)), (s(1)_{ok}, green)\} \cup \{(s(1)_a, green), (p_{s(n)}, s(n)_{ok})\} \cup$
  $\{(s(i), p_{s(i)}), (ok_{s(i)}, s(i)_{ok}) \mid i = 1, \ldots, n\} \cup$
  $\{(p_{s(i)}, s(i)_q), (s(i)_q, a_{s(i)}), (a_{s(i)}, s(i)_a) \mid i = 1, \ldots, n\} \cup$
  $\{(p_{s(i)}, s(i+1)), (s(i+1)_{ok}, ok_{s(i)}), (s(i+1)_a, a_{s(i)}) \mid i = 1, \ldots, n-1\} \cup$
  $\{(green, rep(do)), (rep(do), rep), (rep, rep(ok)), (rep(ok), green)\}$,

- $\lambda_M(s(i)) = s(i)!$, $\lambda_M(s(i)_{ok}) = s(i)_{ok}!$, $\lambda_M(s(i)_a) = s(i)_a!$, $\lambda_M(s(i)_q) = a \in \mathcal{A}$, $\lambda_M(rep(do)) = rep!$ y $\lambda_M(rep(ok)) = rep_{ok}!$.

Además, para cada marcaje $\mathcal{M}$ de $\mathcal{N}$ definimos el marcaje $\mathcal{M}^* = \{M^* \mid M \in \mathcal{M}\} \cup \{M_0\}$, donde $M_0 = \{green\}$ y $M^* = M \in \mathcal{MS}(P^*)$. Así que $\mathcal{M}^*$ coincide con $\mathcal{M}$ en todas las componentes, dejando vacíos los nuevos lugares red_t, añadiendo el marcaje del monitor, con un solo token en *green*. Por eso, una componente no tiene tokens si y sólo si la componente que la simula tampoco tiene tokens, por lo que la construcción preserva la basura. Nótese que la construcción del monitor no depende de $\mathcal{N}$, de modo que el mismo monitor funcionaría con cualquier sistema m-RUN que sólo utilice servicios en $\overline{S}$. Además, *Mon* es una máquina de estados, ya que todas sus transiciones tienen una sola precondición y una sola postcondición, y M_0 contiene un único token. Todas las transiciones de todas las N^* son de sincronización, y han de sincronizar con él (excepto las de replicación, aunque para dispararlas también tienen que sincronizar previamente con el monitor).

Veamos que $\mathcal{N}^*$ simula a $\mathcal{N}$:

- Si $\mathcal{M}[u\rangle\mathcal{M}'$ entonces $\mathcal{M}^* \to^* \mathcal{M}'^*$: Supongamos en primer caso que $u = (t_1(M_1), \ldots, t_n(M_n))$ con $\lambda(t_i) = s(i)$. Entonces ha de ser $M_i \in \mathcal{M}$ con ${}^\bullet t_i \subseteq M_i$ y $\mathcal{M}' = M - \{M_1, \ldots, M_n\} + \{M'_1, \ldots, M'_n\}$, donde $M'_i = M_i - {}^\bullet t_i + t_i^\bullet$ para $i = 1, \ldots, n$.

  Denotamos $\overline{M}_i$ al marcaje de *Mon* con un token en $p_{s(i)}$ para $i = 0, \ldots, n$ (identificando $p_{s(0)}$ con *green*) y definimos $\overline{\mathcal{M}}_i = \mathcal{M}^* - \{M_1^*, \ldots, M_i^*, \overline{M}_0\} + \{M_1'', \ldots, M_i'', \overline{M}_i\}$ con $M_j'' = M_j^* - {}^\bullet t_i + t_i^\bullet + \{red_{t_i}\}$. Entonces $\overline{\mathcal{M}}_0 = \mathcal{M}^*$ y $\overline{\mathcal{M}}_i \xrightarrow{u_i} \overline{\mathcal{M}}_{i+1}$ con $u_i = (t_i(M_i^*), s(i)(\overline{M}_i))$ para $i = 0, \ldots, n-1$. Lo primero es evidente. Veamos lo segundo.

  Como $M_i \in \mathcal{M}$ entonces $M_i^* \in \mathcal{M}^*$, por lo que $M_i^* \in \overline{\mathcal{M}}_i$. Además, como ${}^\bullet t_i \subseteq M_i$ por construcción se tiene que ${}^\bullet t_i \subseteq M_i^*$. Por otro lado, $\overline{M}_i \in \overline{\mathcal{M}}_i$ y por definición $\overline{M}_i(p_{s(i)}) = 1$. Por último, $\lambda^*(t_i) = s(i)?$ y $\lambda_M(s(i)) = s(i)!$, por lo que las transiciones son compatibles y el par está activado. Claramente $\overline{M}_i - {}^\bullet s(i) + s(i)^\bullet = \overline{M}_{i+1}$ por lo que $\overline{\mathcal{M}}_i - \{M_i^*, \overline{M}_i\} + \{M_i'', \overline{M}_{i+1}\} = \overline{\mathcal{M}}_{i+1}$.

  Para $i = n$ se tiene que $\overline{\mathcal{M}}_n = \mathcal{M}^* - \{M_1^*, \ldots, M_n^*, \overline{M}_0\} + \{M_1'', \ldots, M_n'', \overline{M}_n\}$, que no es $\mathcal{M}'^*$ porque los M_i'' tienen marcado red_{t_i} y el monitor tiene marcado $s(n)$ en lugar de *green*. Pero en estas condiciones, como la única precondición de $t_i(ok)$ es red_{t_i} (y no tiene postcondiciones), $\lambda^*(t_i(ok)) = s(i)_{ok}?$ y $\lambda_M(s(i)_{ok}) = s(i)_{ok}?$ se pueden disparar los pares $(t_i(ok)(M_i''), s(i)_{ok}(\overline{M}_i'))$ para $i = n, \ldots, 1$, donde $\overline{M}_i'$ es el marcaje de *Mon* con un token en $ok_{s(i)}$ (identificando $ok_{s(0)}$ con *green*). Como cada $t_i(ok)$ tiene el único efecto de eliminar el token de red_{t_i} para $i = 1$ se llega a $\mathcal{M}'^*$, como queríamos demostrar. El caso de que $u = t(M)$ con $\lambda(t) \in \mathcal{MS}(P)$ es totalmente análogo, excepto por el hecho de que la secuencia de disparos que la simula en $\mathcal{N}^*$, $(t(rep), rep(do))(M, \overline{M}_0), t(M''), (t(ok)(M'), rep(ok)(\overline{M}_1))$, tiene una transición de replicación.

- Si $\mathcal{M}^* \to^* \overline{\mathcal{M}}$ entonces existe $\mathcal{M}'$ tal que $\overline{\mathcal{M}} \to^* \mathcal{M}'^*$ y $\mathcal{M} \to^* \mathcal{M}'$: Todas las transiciones del cómputo (excepto las de replicación) lo son de sincronización con el monitor. Como *Mon* es una máquina de estados, claramente las únicas secuencias que puede disparar son repeticiones de secuencias del tipo

  $$s(1), \ldots, s(n), s(n)_{ok}, \ldots, s(1)_{ok}$$

  o del tipo

  $$s(1), \ldots, s(i), s(i)_q, s(i)_a, \ldots, s(1)_a,$$

  acabando en alguna subsecuencia de las anteriores. Procedemos por inducción sobre el número de esas repeticiones.

  Para el paso inductivo basta ver que si la secuencia disparada por el monitor es de alguno de esos tipos entonces se corresponde con un cómputo de la red original. Si la secuencia es del primer tipo entonces la traza ejecutada es la que obteníamos en el punto anterior y, siguiendo los razonamientos

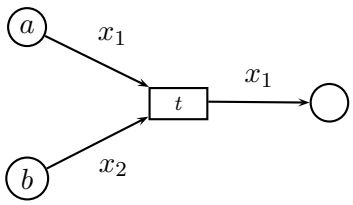

Figura 7.7: APN sin creación de nombres

inversos, se sigue que $\mathcal{M}[u\rangle\mathcal{M}'$ con $\mathcal{M}'^* = \overline{\mathcal{M}}$. Si la secuencia es del segundo tipo entonces, como las precondiciones de t coinciden con las postcondiciones de $t(a)$ y viceversa se tiene que tras la ejecución de la correspondiente traza se llega a $\mathcal{M}^*$, y el cómputo vacío en la red original es el que se corresponde con esa secuencia.

Para el caso base, veamos que cualquier subsecuencia de las anteriores se puede completar de manera que tendremos, de la misma manera que en el caso anterior, un cómputo del original. Si la secuencia no contiene $s(n)_{ok}$ entonces es del tipo $s(1), \ldots, s(i)$ o $s(1), \ldots, s(i), s(i)_q, s(i)_a, \ldots, s(j)_a$. Claramente, la del primer tipo se puede completar a la del segundo, y ésta al caso en el que $j = 1$, y se vuelve a llegar al marcaje $\mathcal{M}^*$, lo que simula al cómputo vacío en $\mathcal{N}$. Si la subsecuencia es del tipo $s(1), \ldots, s(n), s(n)_{ok}, \ldots, s(i)_{ok}$ entonces se puede completar al caso $i = 1$ e, igual que veíamos en el punto anterior, simula la ejecución de una tupla $(t_1, \ldots, t_n)$ con $\lambda(t_i) = s(i)$ para $i = 1, \ldots, n$.

Además, la simulación es claramente monótona porque $\mathcal{M}^*$ coincide con $\mathcal{M}$ en los lugares comunes y para cualquier $\mathcal{M}'$, $\mathcal{M}'^*$ coincide con $\mathcal{M}^*$ en los lugares extra. Además, como hemos visto que el lugar *green* del monitor está marcado si y sólo si el marcaje es estable, no pueden existir marcajes inestables alcanzables mayores que marcajes estables.

□

Por el lema anterior, dada una ν-APN, nos basta con encontrar un sistema mg-RUN N^* que lo simule. La idea clave es usar una componente distinta para cada nombre, todas ellas del mismo tipo. El tipo de las componentes tiene los mismos lugares que N (además de algunos lugares auxiliares), de modo que un token negro en el lugar p de la componente que representa al identificador a simula la aparición del token a en el lugar p de la red original (véanse las figuras 7.7 y 7.8).

Esta es una forma inmediata de representar los marcajes de N en N^*. Sin embargo, la simulación de los disparos de las transiciones es bastante más enrevesada. Dada una transición t de N, si todo arco adyacente a ella está etiquetado con la misma variable, entonces el disparo de esa transición sólo involucra a tokens con el mismo nombre. Si ese nombre es a, entonces la componente RUN que representa

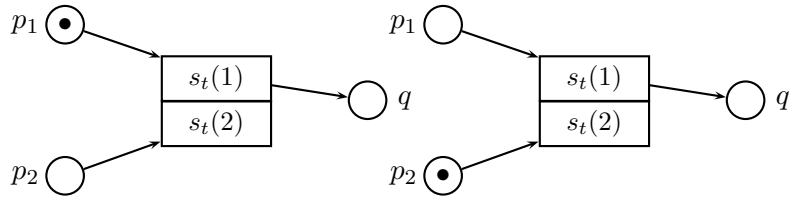

Figura 7.8: Simulación de la APN de la figura 7.7 por medio de sistemas RUN con recogida de basura

a a puede disparar una transición autónoma que simule ese disparo, moviendo los tokens de las precondiciones de t a sus postcondiciones.

Sin embargo, si hay más de una variable adyacente a t entonces eso no es posible. Considérese la red de la figura 7.7, con dos variables junto a su única transición, x e y. Aun puede darse el caso de que t se dispare con x e y instanciadas al mismo valor, de modo que podríamos pensar que necesitamos una transición autónoma que simule la transición original, al igual que en el caso anterior. Sin embargo, como estamos permitiendo la autosincronización, en lugar de tener una transición autónoma para este caso vamos a considerar dos transiciones distintas, una para x y otra para y, que sincronicen entre sí. La ventaja de permitir esta autosincronización es que la construcción también funciona cuando t se dispara tomando dos tokens distintos, por ejemplo a y b, para x y y, respectivamente. En ese caso, dos redes componentes, la que represente a a y la que represente a b, pueden interaccionar por medio de transiciones de sincronización para simular el disparo. La que representa a a debe eliminar un token de p_1 y ponerlo en q, mientras que la que representa a b debe solamente eliminar un token de su lugar q (véase la figura 7.8).

La creación de nombres la simulamos con creación de componentes, como no podía ser de otra manera. Más concretamente, simulamos toda transición con arcos postcondición etiquetados con ν mediante una transición de replicación. El marcaje inicial de la nueva componente es aquél con un token en todas las postcondiciones que están unidas a la transición mediante un arco etiquetado con ν (véase la figura 7.9). Nótese que en la simulación es preciso eliminar los arcos etiquetados con ν, ya que las transiciones de replicación marcan automáticamente esos lugares. Esta construcción funciona si suponemos que las transiciones que crean nombres no involucran a más de dos identificadores distintos. Si no es así basta desdoblar estas transiciones en dos que se disparen secuencialmente, una que no involucre a ν y otra que sólo tenga a ν y otra variable en arcos adyacentes, de modo similar a como hacíamos en la proposición 7.4.1.

En general, para una transición cualquiera t, tenemos que considerar todas las

variables del conjunto $Var(t)$, añadiendo tantas transiciones como variables en él. Cuando $Var(t)$ tiene más de dos elementos, en la simulación tienen que sincronizar más de dos componentes, lo que podemos permitir por la proposición 7.4.13.

Consideramos un orden total cualquiera como hacíamos en la proposición 5.2.4. Además, para cada transición consideraremos una etiqueta de servicio $s_t \in S$ con aridad $|Var(t)|$.

Definición 7.4.14 *Dada la ν-APN $N = (P, T, F)$ definimos el sistema mg-RUN $\mathcal{N}^* = \{N^*\}$ con $N^* = (P, T^*, F^*, \lambda^*)$, donde*

- $T^* = \{t_x \mid t \in T,\ \nu \neq x \in Var(t)\}$,
- $F^* = \{(p, t_x) \mid F(p,t) = x\} \cup \{(t_x, p) \mid F(t,p) = x\}$,
- $\lambda^*(t_x) = \begin{cases} s_t(i) & \text{si } \nu \notin Var(t) = (x_1, \ldots, x_n) \text{ y } x = x_i \\ \{p \mid F(t,p) = \nu\} & \text{en otro caso} \end{cases}$

Además, dado el marcaje M de N definimos el marcaje de $\mathcal{N}^*$, $M^* = \{M_a \mid a \in S(M)\}$, donde $M_a(p) = M(p)(a)$.

Lema 7.4.15 *$M[t(\sigma)\rangle M'$ si y sólo si $M^* \xrightarrow{u} M'^*$*

Demostración. Para la implicación a la derecha supongamos primero que $\nu \notin Var(t) = (x_1, \ldots, x_n)$ y denotemos $a_i = \sigma(x_i)$. En primer lugar, para cada i existe p tal que $F(p,t) = x_i$ y, por lo tanto, $a_i = \sigma(F(p,t)) \in M(p)$. Entonces $\{\sigma(x_i) \mid i = 1, \ldots, n\} \subseteq S(M)$ y, por lo tanto, $M_{a_i} \in M^*$ para todo i. Además, si $p \in {}^\bullet t_{x_i}$ entonces $F(p,t) = x_i$ y, por tanto, $\sigma(F(p,t)) = a_i \in M(p)$. De ello se deduce que $M_{a_i}(p) = M(p)(a_i) > 0$ y la sincronización $t_{x_1}(M_{a_1}), \ldots, t_{x_n}(M_{a_n})$ está activada (son compatibles por definición de λ^*). Tras su disparo se alcanza un marcaje $\mathcal{M}' = M^* - \{M_{a_1}, \ldots, M_{a_n}\} + \{M'_1, \ldots, M'_n\}$ con $M'_i = M_{a_i} - {}^\bullet t_{x_i} + t^\bullet_{x_i}$. Para concluir que $\mathcal{M}' = M'^*$ basta comprobar que $M'_{a_i} = M'_i$, pues el resto de componentes no se ven afectadas por el disparo. En efecto, como

$$ {}^\bullet t_{x_i}(p) = \{\sigma(F(p,t))\}(a_i) = \begin{cases} 1 \text{ si } \sigma(F(p,t)) = a_i \\ 0 \text{ en otro caso} \end{cases} \quad y $$

$$ t^\bullet_{x_i}(p) = \{\sigma(F(t,p))\}(a_i) = \begin{cases} 1 \text{ si } \sigma(F(t,p)) = a_i \\ 0 \text{ en otro caso,} \end{cases} $$

se cumple que $M'_{a_i}(p) = M'(p)(a_i) = M(p)(a_i) - \{\sigma(F(p,t))\}(a_i) + \{\sigma(F(t,p))\}(a_i) = M_{a_i}(p) - {}^\bullet t_{x_i}(p) + t^\bullet_{x_i}(p) = M'_i(p)$.

Si $\nu \in Var(t)$ entonces $Var(t) = \{x, \nu\}$ y $\lambda^*(t_x) \in \mathcal{MS}(P)$. Denotemos $\sigma(x) = a$ y $\sigma(\nu) = \eta$. Igual que en el caso anterior se ve que $M_a \in M^*$ y $M_a(p) > 0$ para todo $p \in {}^\bullet t_x$, y por lo tanto la transición t_x está activada para M_a. Su disparo produce un marcaje $\mathcal{M}' = M^* - \{M_a\} + \{\overline{M}, \lambda^*(t_x)\}$. Para ver que $\mathcal{M}' = M'^*$ basta ver que $M'_a = \overline{M}$ y $M'_\eta = \lambda^*(t_x)$. Lo primero se ve igual que en el caso anterior para las a_i y x_i. Además, $M'_\eta(p) = M'(p)(\eta) = M(p)(\eta) - \{\sigma(F(p,t))\}(\eta) +$

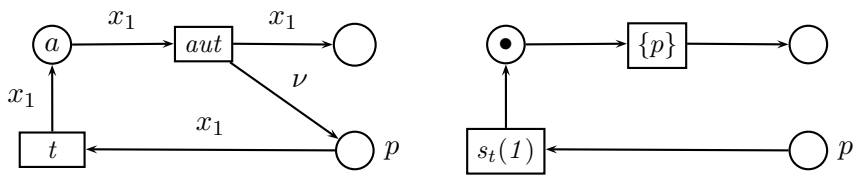

Figura 7.9: ν-APN y la RUN que la simula

$\{\sigma(F(t,p))\}(\eta) = \{\sigma(F(t,p))\}(\eta) = \lambda^*(t_x)(p)$, la tercera igualdad por ser η fresco y la última porque ambos términos se reducen a 1 en el caso de que $F(t,p) = \nu$ o 0 en otro caso.

La otra implicación sigue un esquema parecido. Procedemos por distinción de casos sobre u. Por construcción de N^* los únicos dos casos son $u = t_{x_1}(M_{a_1}), \ldots, t_{x_n}(M_{a_n})$ con $\lambda^*(t_{x_i}) = s_t(i)$ y $u = t_x(M)$ con $\lambda^*(t_x) \in \mathcal{MS}(P)$. Veamos que en el primero de los casos se puede disparar $t(\sigma)$ con $\sigma(x_i) = a_i$. En efecto, si p es precondición de t y $F(p,t) = x_i$ entonces $\sigma(F(p,t)) = a_i \in M(p)$ porque $M(p)(a_i) = M_{a_i}(p) > 0$. Además, tras su disparo se obtiene M' y la tesis se sigue. El otro caso es totalmente análogo.

□

Proposición 7.4.16 *Los sistemas RUN centralizados con recogida de basura simulan monótonamente a las ν-APNs.*

Demostración. Veamos que estamos en las hipótesis de la proposición 2.6.7. El lema anterior nos dice que los sistemas RUN simulan paso a paso a las ν-APNs. Además, todo marcaje $\mathcal{M}$ de $\mathcal{N}^*$ es imagen por $(\ )^*$ de algún marcaje de N, de manera que todos los estados alcanzables son estables. En efecto, si $\mathcal{M} = \{M_1, \ldots, M_n\}$ basta tomar n identificadores distintos dos a dos, $a_1, \ldots, a_n$ y definir $M(p)(a_i) = M_i(p)$, que cumple que $M^* = \mathcal{M}$.

Por último, veamos que tanto $(\ )^*$ como su inversa son monótonas. En primer lugar, $M \equiv_\alpha M' \Leftrightarrow M^* = M'^*$. Efectivamente, $M \equiv_\alpha M' \Leftrightarrow M' = h(M) \Leftrightarrow M_a(p) = M(p)(a) = M'(p)(h(a)) = M'_{h(a)}(p) \Leftrightarrow M^* = M'^*$ para $h : S(M) \to S(M')$ biyección. Además, si $M_1 \sqsubseteq M_2$ entonces para todo $p \in P$, $M_1(p) \subseteq M_2(p)$. Entonces $(M_1)_a(p) = M_1(p)(a) \leq M_2(p)(a) = (M_2)_a(p)$ y, por tanto, $M_1^* \sqsubseteq M_2^*$. Juntando ambas cosas se concluye que si $M_1 \sqsubseteq_\alpha M_2$ entonces $M_1^* \sqsubseteq M_2^*$.

Para concluir tenemos que ver que si $M_1^* \sqsubseteq M_2^*$ entonces $M_1 \sqsubseteq_\alpha M_2$. Por definición del orden entre multiconjuntos existe una función $h : M_1^* \to M_2^*$ inyectiva tal que $M \subseteq h(M)$ para todo $M \in M_1^*$. Veamos entonces que si definimos $h'(a) = b$ si $(M_2)_b = h((M_1)_a)$ se cumple que $h(M_1) \sqsubseteq M_2$, lo que, por el lema 6.2.2, nos permitirá concluir que $M_1 \sqsubseteq_\alpha M_2$. Pues bien, $h'(M_1)(p)(h'(a)) = M_1(p)(a) = (M_1)_a(p) \leq h((M_1)_a)(p) = (M_2)_{h'(a)}(p) = M_2(p)(h'(a))$ y la tesis se sigue.

□

Corolario 7.4.17 *El problema de alcanzabilidad para sistemas RUN con recogida de basura es indecidible, mientras que el correspondiente problema de recubrimiento es decidible .*

Podríamos pensar que una construcción análoga a la anterior probaría la simulación monótona de los sistemas MSPN con identificadores naturales por parte de sistemas RUN sin basura. Sin embargo, esa construcción no sería una simulación por el razonamiento inverso al que seguíamos allí, cuando decíamos que los sistemas MSPN son un etiquetado de los sistemas RUN. Allí la simulación recordaba información que no estaba presente en los marcajes originales (el orden de creación de las componentes). Ahora la construcción estaría ignorando esa información. En otras palabras, la función $(\ )^*$ de transformación de marcajes no sería ni siquiera una inyección.

Para terminar, indicaremos que todos los resultados de acotación de identificadores que vimos en el capítulo 6 se pueden transferir al modelo con replicación, gracias a la proposición 7.4.7, que preserva las nociones de acotación. En consecuencia, la versión análoga a la acotación en anchura, que podemos llamar *acotación de componentes*, es decidible para los sistemas RUN con recogida de basura, y la alcanzabilidad es decidible para sistemas RUN con recogida de basura que tengan un número de componentes acotado.

7.5. Creación de nombres y replicación

Ahora consideramos un modelo en el que combinamos las dos características de las secciones anteriores, es decir, sistemas compuestos de redes que pueden tanto crear identificadores nuevos como replicarse. Formalmente, un sistema ν-RN es un sistema MSPN centralizado con identificadores abstractos donde algunas transiciones pueden ir etiquetadas con marcajes de componentes, como en la sección anterior, sólo que ahora esos marcajes están compuestos por tokens identificador.

Sorprendentemente, aunque las potencias expresivas que se obtienen al añadir o bien nombres o bien replicación son idénticas, como probamos en la sección anterior, resulta que son de alguna manera ortogonales, de manera que cuando se combinan se alcanza la Turing completitud, como probaremos a continuación. Informalmente, podemos decir que ambos modelos generan sistemas bidimensionales de infinitos estados. Las ν-APNs pueden tener una cantidad no acotada de tokens distintos, y cada uno de ellos puede aparecer una cantidad no acotada de veces. Análogamente, en el caso de los sistemas RUN, podemos tener una cantidad no acotada de componentes, cada una de ellas con una cantidad no acotada de tokens. Sin embargo, estos dos espacios bidimensionales no son el mismo, aunque sean de alguna manera equivalentes entre sí, según las proposiciones 7.4.7 y 7.4.16. Pero cuando fundimos las tres fuentes de infinitud, sólo la dimensión común se solapa, o siguiendo la metáfora geométrica, tenemos planos perpendiculares que ya no generan un espacio bidimensional, sino el espacio tridimensional completo.

Veremos que este espacio es demasiado rico, obteniendo así un formalismo Turing completo.

Proposición 7.5.1 *Cualquier máquina de Turing se puede simular con un sistema ν-RN.*

Demostración. Dada una máquina de Turing, tenemos que simularla usando un sistema ν-RN. El principal problema a la hora de simular máquinas de Turing suele ser la representación del conjunto potencialmente infinito de celdas de la cinta. Representamos la cinta como una lista doblemente enlazada. Cada nodo de la lista (o celda de la cinta) se representa mediante una copia distinta de la misma ν-RN, ilustrada en la figura 7.10. Esta red tiene una memoria con dos tipos de datos. En primer lugar, tiene que saber si su valor es un 0 o un 1. Para ello, tiene dos lugares llamados 0 y 1, marcados en exclusión mutua. En segundo lugar, y lo que es más importante, la celda debe contener información acerca de qué componentes representan las celdas anterior y siguiente. Para ello, las componentes que representan celdas tienen tres lugares, *me*, *prev* y *next*, que contienen tres identificadores distintos: el nombre de la propia celda en *me*, el de la celda anterior en *prev* y el de la siguiente celda en *next*.

En cualquier momento la cinta tiene una última celda, que conoce su propio nombre y el de la celda anterior (se ha inicializado disparando su transición *init*?), pero no conoce el nombre de la celda de su derecha, que de hecho aún no existe. Para expandir la cinta, esta celda crea una nueva, disparando su transición de replicación etiquetada con $\{i, 0\}$. El marcaje de la nueva celda cuenta solamente con dos tokens ordinarios, uno en el lugar i y otro en 0. Por tanto, puede generar su propio nombre, después de lo cual está dispuesta a sincronizar con su red padre (la celda del final de la cinta). La sincronización de las transiciones *init*! y *init*? produce el intercambio de los nombres respectivos de la celda, colocándolos en los correspondientes lugares *next* y *prev*, de modo que la red de la cinta ahora conoce el nombre de la celda a su derecha, y la nueva celda pasa a ser la última de la cinta.

Finalmente, las celdas tienen transiciones de sincronización que devuelven el nombre de las celdas anterior y siguiente (transiciones *left*! y *right*!, respectivamente), otras dos que devuelven el contenido actual de la celda (transiciones $r(0)$! y $r(1)$!) y finalmente otras cuatro que cambian el valor actual según indique el programa (las etiquetadas con $w(0)$? y $w(1)$?). Todas estas transiciones tienen al lugar *me* tanto como precondición como postcondición, con arcos etiquetados con la variable a.

La parte de la construcción que hemos descrito hasta ahora es la misma para cualquier máquina de Turing. El resto de la simulación sólo necesita una red adicional para el control, con un lugar *now* que contiene el identificador de la celda a donde la cabeza de la máquina apunta en cada momento. Todas las sincronizaciones entre las celdas y el programa que se mencionaban en el párrafo anterior se hacen forzando la igualdad entre el valor en el lugar *now* del programa y el valor

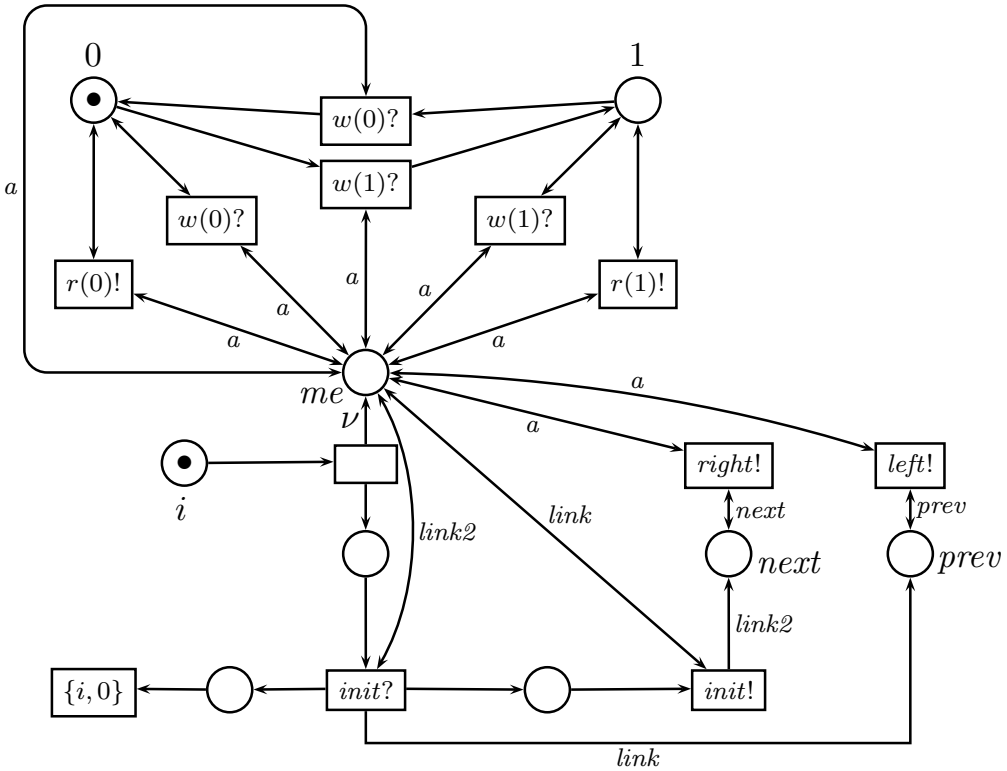

Figura 7.10: Celda de la cinta de una máquina de Turing

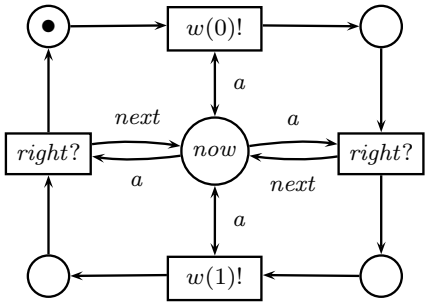

Figura 7.11: Máquina de Turing que computa 0101 . . .

del lugar *me* de la celda, por medio de la variable a, de modo que la cabeza sólo puede leer y modificar la celda a la que apunta. Por supuesto, también necesitamos un lugar por cada estado de la máquina, marcados en exclusión mutua, cada uno de ellos con las correspondientes acciones. Como ejemplo, la figura 7.11 muestra la simulación de la cabeza de la máquina de Turing que computa la secuencia infinita 0101 . . .

□

Es preciso tener en cuenta que en la demostración del resultado anterior no

$\mathcal{M}^1$	$\mathcal{M}^2$	$\mathcal{M}^3$	$\mathcal{M}^4$
1	1	1	1
1 2	2 2	2 2	2 2
	2 3 3	3 3 3	3 3 3
		3 4 4 4	4 4 4 4
			5 4 4 4 4

Figura 7.12: El orden natural de los sistemas ν-RN no es un wqo

hemos usado la recogida de basura, de modo que la Turing completitud se alcanza incluso en ausencia de ese mecanismo. De hecho, como comentábamos en el capítulo 4, estamos considerando implícitamente una especie de recogida de basura para identificadores, en el sentido de que estos pueden desaparecer de los marcajes después del disparo de alguna transición (como ocurre con el identificador b en la figura 7.7 después del disparo de aut), tornándose así reutilizables. Conjeturamos que si evitamos esta recogida automática de basura, incorporando al estado el conjunto completo de nombres creados a lo largo del cómputo, o usando la versión de creación de nombres con naturales, entonces obtendríamos una situación análoga a la que se tiene en [Del], de modo que la alcanzabilidad también sería decidible. Si no evitamos la recogida de basura para identificadores, obtenemos el siguiente resultado.

Corolario 7.5.2 *Los problemas de alcanzabilidad y de recubrimiento son indecidibles para los sistemas ν-RN.*

Demostración. Claramente, la alcanzabilidad permanece indecidible, ya que ya era indecidible en el modelo menos general de sistemas ν-APN. Además, el recubrimiento también es indecidible para los sistemas ν-RN, o de lo contrario podríamos usar la simulación anterior para decidir el problema de parada de las máquinas de Turing, sin más que decidir si el marcaje con un token en el lugar que representa el estado final se puede cubrir.

□

Sería interesante encontrar restricciones adecuadas al uso de la variable ν o al uso de las transiciones de replicación, de modo que un modelo restringido, pero que incluyese tanto creación de nombres como de componentes, no fuese Turing completo, obteniendo así resultados análogos a los de la sección 6.3 para el modelo con replicación. Claramente, si el número de veces que se pueden disparar transiciones de creación de nombres está acotado (o alternativamente, transiciones de replicación), entonces el modelo así restringido dejaría de ser Turing completo. Es más, la simulación de máquinas de Turing que hemos presentado no funcionaría

si en cada marcaje alcanzable sólo pudiesen aparecer una cantidad acotada de nombres distintos, incluso si se pueden crear un número arbitrario de ellos, lo que sugiere que el recubrimiento seguiría siendo decidible en ese caso.

Merece la pena resaltar que, como consecuencia del anterior resultado de indecidibilidad, aunque los órdenes naturales tanto para las ν-APNs como para los sistemas RUN son wqo, el orden natural de los sistemas ν-RN, que extiende a ambos, no puede ser un wqo.

Definición 7.5.3 *Dados dos marcajes $\mathcal{M}_1$ y $\mathcal{M}_2$ de un sistema ν-RN $\mathcal{N}$ definimos el orden $\sqsubseteq$ mediante $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$ si existen dos inyecciones $h_1 : \mathcal{M}_1 \to \mathcal{M}_2$ y $h_2 : S(\mathcal{M}_1) \to S(\mathcal{M}_2)$ tales que para todo $M \in \mathcal{M}_1$, $h_1(M)(p) \subseteq h_2(M(p))$.*

La inyección h_1 tiene el papel de identificar componentes de $\mathcal{M}_1$ con componentes de $\mathcal{M}_2$, mientras que h_2 tiene el papel de identificar nombres de $\mathcal{M}_1$ con nombres de $\mathcal{M}_2$. En efecto, el orden definido no es un wqo. Para verlo, basta con considerar el caso con sistemas con un sólo tipo de componentes, y con un solo lugar, de modo que un marcaje de un sistema así se puede especificar mediante un multiconjunto de multiconjuntos de identificadores. En este caso, la definición anterior se simplifica a $\mathcal{M}_1 \sqsubseteq \mathcal{M}_2$ si existen dos inyecciones $h_1 : \mathcal{M}_1 \to \mathcal{M}_2$ y $h_2 : S(\mathcal{M}_1) \to S(\mathcal{M}_2)$ tales que para todo $M \in \mathcal{M}_1$, $h_1(M) \subseteq h_2(M)$. Consideremos $Id = \mathbb{N}$ y la sucesión de marcajes mostrada en la figura 7.12, $\mathcal{M}^i = \{M^i_1, \ldots, M^i_i, M^i_{i+1}\}$, para $i = 1, 2, \ldots$, donde $M^i_k = \{k, \overset{i}{\ldots}, k\}$ para $k = 1, \ldots, i$ y $M^i_{i+1} = \{i, i+1, \overset{i}{\ldots}, i+1\}$, para la que no existen dos índices $i < j$ tales que $\mathcal{M}^i \sqsubseteq \mathcal{M}^j$. Esto es así porque, para $i < j$, por cardinalidad la única opción es tomar h_1 como $h_1(M^i_k) = M^j_k$ para $k = 1, \ldots, i$, y h_2 como la identidad, que tampoco funcionan pues $i \in M^i_{i+1}$ pero $i \notin M^j_k$, para todo $k \geq i+1$.

Capítulo 8

Sistemas etiquetados

En este capítulo vamos a presentar una técnica general para la especificación de propiedades que dependen de la historia del sistema. En el siguiente capítulo usaremos esta técnica para el estudio de sistemas abiertos, es decir, de sistemas que pueden encontrarse y ejecutarse en un entorno arbitrario.

En la literatura se pueden encontrar muchos marcos para la definición de propiedades de seguridad [HR02b, HR02a, FG01, FGM04, BB02]. Una primera forma de hacerlo consiste en buscar los posibles estados erróneos del sistema analizado. El carácter erróneo de algunos de esos estados no depende de la manera en la que hayan sido alcanzados, en cuyo caso los mismos pueden ser identificados estáticamente. Un ejemplo claro de propiedad de este tipo es la *confidencialidad*, para la que los estados erróneos no son más que aquéllos en los que cierta información confidencial está expuesta al entorno externo.*

Sin embargo, algunas propiedades no pueden ser comprobadas de esta forma, simplemente mediante inspección del estado actual. Esto ocurre, en particular, con todas las propiedades que dependen de la historia del cómputo. Un ejemplo de propiedad tal es el control de acceso a recursos, donde para saber si el acceso a un recurso por parte de un proceso es erróneo debemos conocer los permisos adquiridos por dicho proceso a lo largo de su historia [HR02b].

Una forma de comprobar de manera estática la ausencia de errores del primer tipo es con la ayuda de un sistema de tipos apropiado, de manera que los sistemas en un estado erróneo no sean tipables. De este modo, si el proceso original es tipable y el sistema de tipos cumple la propiedad de *reducción del sujeto*, es decir, que el tipo de un proceso se mantiene durante su ejecución, entonces podemos garantizar que no se va a producir ningún error a lo largo de la ejecución del proceso.

Sin embargo, la aproximación anterior no es válida si pretendemos prevenir errores del segundo tipo, ya que los errores no son propiedad de un único estado, sino de todo el cómputo. En tal caso es necesario realizar un análisis de trazas.

*Ciertamente, mediante esta propiedad tan simple no podríamos controlar flujos implícitos de información.

La manera más natural y habitual de llevar a cabo este análisis es *recordando* de alguna manera la información acumulada a lo largo del cómputo que sea de interés para el establecimiento del tipo de error considerado. Ello da lugar a nuevos estados ampliados en los que se incorpora la citada información. En ocasiones, otros autores trabajando en este tema, [HR02b] o [PS93], han definido la correspondiente semántica extendida (con etiquetas) de una manera ad hoc, sin relacionarla de manera explícita con la semántica original, lo que exige posteriores demostraciones, también ad hoc, de cualquier resultado que relacione la semántica original y la extendida.

Nosotros proponemos una metodología sistemática para los errores dependientes de la historia, que comienza por la definición del concepto de *lenguaje etiquetado*. Caracterizamos los lenguajes etiquetados como cocientes del árbol de alcanzabilidad definido por los cómputos del lenguaje original. De este resultado podemos deducir que las propiedades especificadas por cada lenguaje etiquetado son de hecho propiedades del original. De esta forma podemos trabajar en un marco común, en lugar de trabajar con distintas semánticas ad hoc, una para cada propiedad. En particular, podríamos apoyarnos de nuevo en un análisis basado en tipos para garantizar que a lo largo del cómputo nunca se alcanza un estado etiquetado erróneo.

Además, probamos que el conjunto de sistemas cocientes tiene estructura de retículo completo, y nos aprovechamos de esta estructura algebraica para definir la mezcla de dos versiones etiquetadas de un mismo lenguaje, capturando así la conjunción de las dos propiedades.

8.1. Sistemas de transiciones etiquetados

A continuación definimos formalmente lo que entendemos por lenguaje etiquetado. Empezaremos definiendo la sintaxis etiquetada generada por una signatura heterogénea (*many-sorted* en inglés) [EM85, Hen90] respecto de un símbolo de la signatura y del conjunto de etiquetas considerado.

Definición 8.1.1 *Dada una signatura heterogénea* $(\mathcal{S}, \Sigma)$*, sea* E *el álgebra de* $\mathcal{S}$*-términos de* Σ*,* f *un símbolo de* Σ *y* $\mathcal{L}$ *un conjunto de etiquetas* $(\Gamma, \Delta, \ldots \in \mathcal{L})$*. Llamaremos* sintaxis $(f, \mathcal{L})$-etiquetada *de* E*, y la denotaremos por* $\bar{E}_f^{\mathcal{L}}$ *(o a veces simplemente* $\bar{E}$*), al álgebra de* $\mathcal{S}$*-términos de* $\Sigma_f^{\mathcal{L}} = (\Sigma - \{f\}) \cup \{f_\Gamma \mid \Gamma \in \mathcal{L}\}$*, donde* $aridad(f_\Gamma) = aridad(f)$*, para cada* $\Gamma \in \mathcal{L}$*.*

En términos de árboles de la sintaxis abstracta, los términos etiquetados no son más que árboles en los que algunos nodos, bien hojas cuando el símbolo distinguido f es un operador constante, bien nodos internos si no lo es, están anotados con etiquetas de $\mathcal{L}$. Una versión más general de sintaxis etiquetada, $\bar{E}_{f_1, f_2, \ldots}^{\mathcal{L}_1, \mathcal{L}_2, \ldots}$, consideraría una colección de símbolos $f_1, f_2, \ldots$ de Σ y un conjunto de etiquetas para cada uno de ellos, posiblemente diferentes.

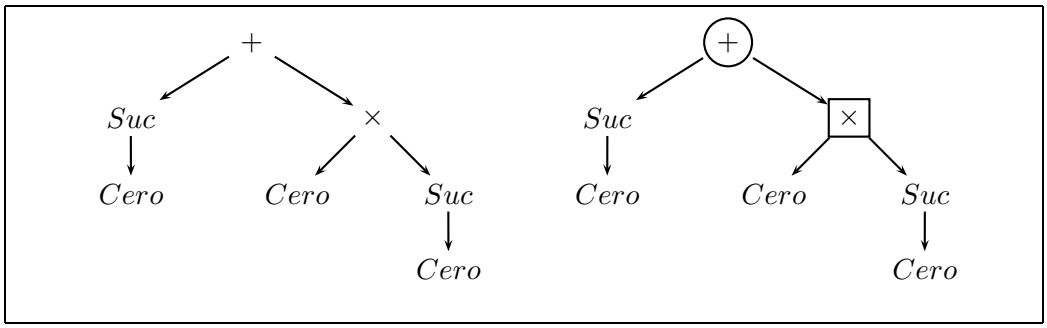

Figura 8.1: Árboles de sintaxis abstracta de $1 + (0 \times 1)$ y $1 \oplus (0 \boxtimes 1)$

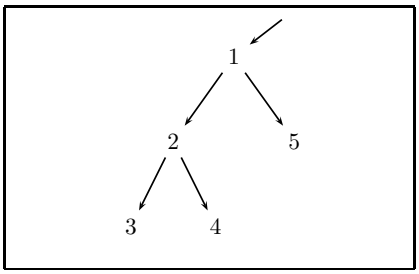

Figura 8.2: Ejemplo de sistema de transiciones 5

Ejemplo 4 Vamos a considerar la signatura $(\mathcal{S}, \Sigma)$, donde $\mathcal{S} = \{Nat\}$ y $\Sigma = \{Cero, Suc, +, \times\}$, con $aridad(Cero) = Nat$, $aridad(Suc) = Nat \to Nat$ y $aridad(+) = aridad(\times) = Nat \to Nat \to Nat$. Consideremos también el conjunto de etiquetas $\mathcal{L} = \{\square, \circ\}$. El álgebra de $\mathcal{S}$-términos de Σ define un conjunto de expresiones aritméticas. Un ejemplo de sintaxis etiquetada sería aquélla generada por $\Sigma_{+,\times}^{\mathcal{L},\mathcal{L}} = \{Cero, Suc, \oplus, \otimes, \boxplus, \boxtimes\}$, que define expresiones aritméticas en las que los operadores han sido sustituidos por versiones suyas, ora recuadradas ora circunvaladas. La figura 8.1 muestra los árboles de sintaxis abstracta de las expresiones $1 + (0 \times 1)$ y $1 \oplus (0 \boxtimes 1)$.

A continuación suponemos que partimos de un sistema de transiciones cuyo conjunto de estados es el álgebra de términos de una signatura. Definiremos cuándo un sistema de transiciones cuyo conjunto de estados es la sintaxis etiquetada de la original se considera un *etiquetado* del sistema original. Denotaremos por $O(e)$ al término que se obtiene del término etiquetado e cuando se borran de él todas las etiquetas, definido por inducción estructural como:

$$O(h(\overline{e_1}, \ldots, \overline{e_n}) = h(O(\overline{e_1}), \ldots, O(\overline{e_n})) \text{ si } f \neq h,$$
$$O(f_\Gamma(\overline{e_1}, \ldots, \overline{e_n}) = f(O(\overline{e_1}), \ldots, O(\overline{e_n})).$$

Definición 8.1.2 *Sea E el álgebra de $\mathcal{S}$-términos de Σ y $\bar{E}$ una sintaxis $(f, \mathcal{L})$-etiquetada de E. Sean también $S = (E, V, \to, e_0)$ y $T = (\bar{E}, V, \mapsto, \bar{e}_0)$ dos sistemas de transiciones con $O(\bar{e}_0) = e_0$. Diremos que T es un* etiquetado *de S si se cumplen las siguientes condiciones:*

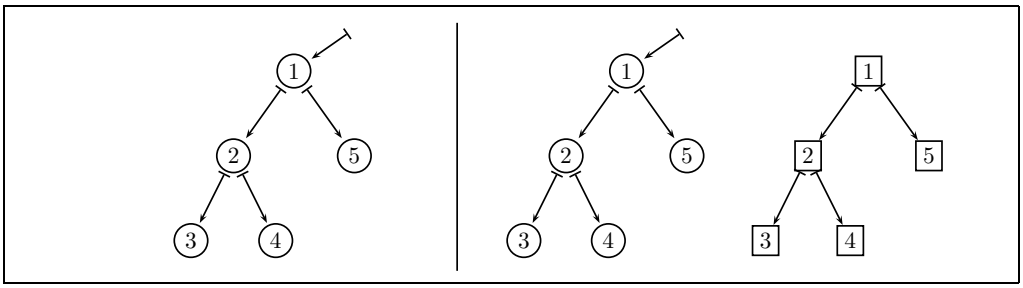

Figura 8.3: Sistemas etiquetados del Ejemplo 5

1. *Si* $\bar{e} \overset{a}{\mapsto} \bar{e}'$ *entonces* $O(\bar{e}) \overset{a}{\to} O(\bar{e}')$.

2. *Si* $e \overset{a}{\to} e'$ *entonces para cada* $\bar{e}$ *alcanzable desde* $\bar{e}_0$ *tal que* $O(\bar{e}) = e$ *existe un único* $\bar{e}'$ *tal que* $O(\bar{e}') = e'$ *y* $\bar{e} \overset{a}{\mapsto} \bar{e}'$.

La primera condición es una propiedad de corrección de los sistemas etiquetados: los sistemas etiquetados no introducen transiciones nuevas, es decir, todas sus transiciones son versiones entre términos etiquetados de transiciones que existen entre términos sin etiquetar. La segunda es una condición de completitud: existe una única versión etiquetada de cada transición original. Además, la unicidad nos dice intuitivamente que no podemos recordar hechos distintos de un mismo pasado.

Nótese que, como decíamos en el capítulo anterior, la proposición 7.4.5 nos dice justamente que los sistemas MSPN son un etiquetado, según la definición anterior, de los sistemas RUN sin basura.

Ejemplo 5 Consideremos la signatura $(\mathcal{S}, \Sigma)$ donde $\mathcal{S} = \{Nat\}$, $\Sigma = \{1, 2, 3, 4, 5\}$ con $aridad(i) = Nat$ y el conjunto de etiquetas $\mathcal{L} = \{\circ, \square\}$. Los términos obtenidos cuando etiquetamos todos los símbolos con etiquetas de $\mathcal{L}$, no son más que los elementos de $\{1, \ldots, 5\}$ rodeados por $\circ$ o por $\square$. Consideremos también el sistema de transiciones definido por el grafo de la figura 8.2. En este ejemplo sencillo, en el que el grafo que define el sistema de transiciones es de hecho un árbol, un sistema etiquetado consta de una o varias copias del sistema original, a lo sumo una copia por cada manera distinta de etiquetar el símbolo inicial, como en los ejemplos de la figura 8.3. Sin embargo, las distintas copias podrían solaparse para términos no alcanzables, como sucede por ejemplo en los sistemas que se muestran en la figura 8.4.

Es preciso tener en cuenta que en la definición de sistemas etiquetados permitimos la presencia de algunos *nodos basura* (bajo ciertas condiciones), que no serán alcanzables desde el estado inicial. Permitimos la existencia de basura para obtener una manera más flexible de especificar sistemas etiquetados, sin necesidad de conocer el conjunto de términos alcanzables. Por supuesto, en la práctica

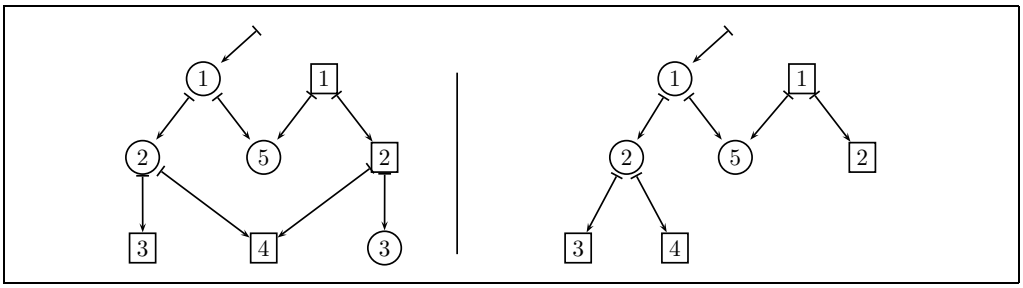

Figura 8.4: Más sistemas etiquetados del Ejemplo 5

intentaremos definir los sistemas etiquetados con la menor cantidad posible de nodos basura, para obtener un sistema tan simple como sea posible. Los nodos basura son totalmente inútiles, pero al mismo tiempo son absolutamente inocuos, salvo en lo que respecta al crecimiento del tamaño del sistema.

Véase que en este ejemplo todos los sistemas etiquetados son de hecho isomorfos entre sí y, a su vez, isomorfos al sistema original, según la definición 2.2.2: si eliminamos todos los términos no alcanzables son iguales, salvo los nombres de los estados. Esto es así debido a que el grafo de transiciones es un árbol y, por tanto, cada nodo determina un único camino desde el nodo inicial (el grafo de transiciones y el árbol de alcanzabilidad son isomorfos).

Ejemplo 6 Veamos un nuevo ejemplo, que presenta sistemas etiquetados que no resultan ser isomorfos entre sí. Consideremos el sistema de transiciones definido por el grafo a la izquierda de la figura 8.5, que no es un árbol. En particular, podemos interpretarlo como un sistema en el que un agente puede encontrarse en tres servidores distintos y moverse entre ellos siguiendo las flechas. Los sistemas etiquetados correspondientes, dibujados a la derecha, pueden discriminar entre las diferentes maneras en las que el agente puede alcanzar el servidor 3. Por ejemplo, si consideramos que el servidor 2 es maligno y no queremos que nuestros agentes pasen por 2, estamos interesados en la propiedad capturada por el sistema etiquetado situado en el centro de la figura. En este sistema, la etiqueta $\circ$ identifica cómputos en los que el agente ha visitado el servidor 2, en contraposición a la etiqueta $\square$. Si sólo estuviésemos interesados en saber qué servidor se ha visitado inmediatamente después de 1, deberíamos usar el sistema de transiciones a la derecha de la figura.

Con estos sencillos sistemas se muestra que es precisamente la existencia de nodos alcanzables etiquetados con distintas etiquetas lo que hace que éstas sean útiles: basta examinar la etiqueta del estado alcanzable para obtener un conocimiento (parcial) acerca de cómo ha sido alcanzado ese estado.

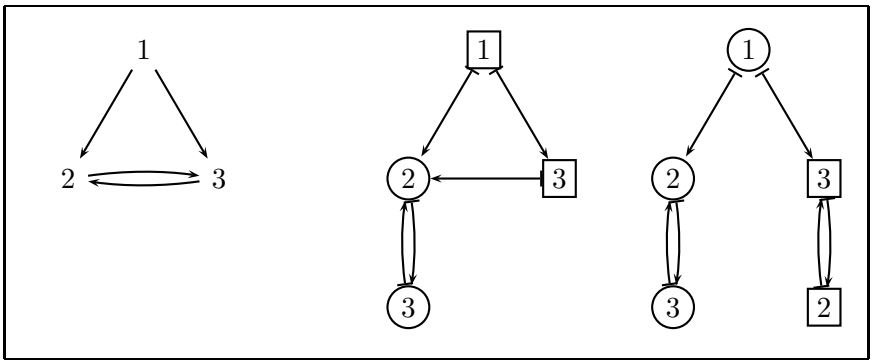

Figura 8.5: Sistema de transiciones (izquierda) y sistemas etiquetados (derecha) del ejemplo 6

8.2. Caracterización alternativa de los sistemas etiquetados

En esta sección vamos a caracterizar a los sistemas etiquetados que hemos definido en la sección anterior como cocientes de su árbol de alcanzabilidad. Para ser más exactos, los sistemas etiquetados son isomorfos (en el sentido de la definición 2.2.2) al árbol de cómputos, una vez que hemos identificado algunos de sus nodos. Empezamos definiendo los árboles de cómputo.

Definición 8.2.1 *Dado un sistema de transiciones etiquetadas $S = (E, V, \rightarrow, e_0)$ y el conjunto de trazas de S, $\mathcal{C}$, definimos el* árbol de cómputos *de S como el sistema de transiciones $S^t = (\mathcal{C}, V, \mapsto, e_0)$, que se obtiene al considerar $\mapsto \subset \mathcal{C} \times V \times \mathcal{C}$, definida por la regla*

$$\frac{e \xrightarrow{a} e' \quad last(L) = e}{L \overset{a}{\mapsto} (L \xrightarrow{a} e')}$$

donde "last" es la función que devuelve el último estado alcanzado en una traza.

El árbol de cómputos no es más que el sistema de transiciones definido por el árbol de alcanzabilidad del sistema original. Simplemente acumula en sus estados la traza de estados que han sido visitados a lo largo del cómputo y las transiciones entre ellos. Por tanto, los nodos contienen toda la información acerca de la historia del proceso. A veces lo llamaremos *sistema totalmente etiquetado.*

Sin embargo, en general estamos interesados sólamente en un aspecto concreto de dicha información. Usaremos relaciones de equivalencia entre cómputos para identificar la información buscada.

Definición 8.2.2 *Diremos que una relación de equivalencia $\sim \ \subset \mathcal{C} \times \mathcal{C}$ es* monótona *si satisface las siguientes condiciones:*

1. *Si* $L_1 \sim L_2$ *entonces* $last(L_1) = last(L_2)$

2. *Si* $L_1, L_2 \in \mathcal{C}$ *y* $L_1 \sim L_2$ *entonces para cada* $a \in V$ *y* $e \in E$ *tales que* $last(L_1) \xrightarrow{a} e$ *(y* $last(L_2) \xrightarrow{a} e$*) se cumple que* $(L_1 \xrightarrow{a} e) \sim (L_2 \xrightarrow{a} e)$.

Por ejemplo, las relaciones $\sim_\perp$ definida por $L_1 \sim_\perp L_2 \Leftrightarrow last(L_1) = last(L_2)$, y $\sim_\top$, definida por $L_1 \sim_\top L_2 \Leftrightarrow L_1 = L_2$, son ejemplos triviales de equivalencias monótonas.

Usaremos las equivalencias monótonas para caracterizar la información acerca del pasado en la que estamos interesados. Por ejemplo, $\sim_\perp$ es la relación que olvida todo el pasado, mientras que $\sim_\top$ es la que se acuerda de absolutamente todo. A continuación definimos el sistema de transiciones cociente generado por una cualesquiera de estas equivalencias.

Definición 8.2.3 *Sea* $S = (E, V, \rightarrow, e_0)$ *un sistema de transiciones etiquetadas y* $\sim \ \subset \mathcal{C} \times \mathcal{C}$ *una relación de equivalencia monótona. Definimos* $S^\sim = (\mathcal{C}/_\sim, V, \mapsto_\sim, [e_0])$*, y lo llamamos* sistema de transiciones cociente de S respecto de $\sim$*, como el sistema de transiciones definido por la regla*

$$\frac{L \overset{a}{\mapsto} L'}{[L] \overset{a}{\mapsto}_\sim [L']}$$

donde $L, L' \in \mathcal{C}$.

La segunda condición en la definición 8.2.2 garantiza que los sistemas de transiciones cociente estén bien definidos, es decir, que sus transiciones no dependen de los representantes de las clases de equivalencia elegidos. La primera condición afirma que la relación recuerda al menos el estado final alcanzado por el cómputo.

Para las equivalencias $\sim_\perp$ $\sim_\top$ definidas antes, resulta que $S^{\sim_\perp}$ y $S^{\sim_\top}$ son isomorfos a S y S^t, respectivamente.

Un sistema de transiciones cociente es una extensión conservadora del sistema de transiciones del que proviene, en el sentido expresado por la siguiente proposición.

Proposición 8.2.4 *Si* $[L_1], [L_2] \in \mathcal{C}/_\sim$ *son dos clases de equivalencia tales que* $[L_1] \overset{a}{\mapsto}_\sim [L_2]$ *entonces* $last(L_1) \xrightarrow{a} last(L_2)$.

Demostración. Supongamos que $[L_1] \overset{a}{\mapsto}_\sim [L_2]$. Por definición de $\mapsto_\sim$ existen L_1' y L_2' tales que $L_1 \sim L_1'$, $L_2 \sim L_2'$ y $L_1' \overset{a}{\mapsto} L_2'$. También por definición de $\mapsto$, se cumple que $last(L_1') \overset{a}{\mapsto} last(L_2')$ y, como $\sim$ es monótona y $L_i \sim L_i'$, se sigue que $last(L_i) = last(L_i')$, con lo que se concluye la tesis.

□

Veamos ahora más de cerca la estructura del conjunto de sistemas de transiciones cociente de un sistema de transiciones.

Definición 8.2.5 *Dado un sistema de transiciones etiquetadas S, denotamos por $\boldsymbol{S}$ al conjunto de sus sistemas cociente, es decir, $\boldsymbol{S} = \{S^{\sim} \mid \sim \text{ monótona}\}$. Definimos $\sqcup$ y $\sqcap$ como las operaciones sobre elementos de $\boldsymbol{S}$ tales que $S^{\sim_1} \sqcup S^{\sim_2} \stackrel{\triangle}{=} S^{\sim_1 \cap \sim_2}$ y $S^{\sim_1} \sqcap S^{\sim_2} \stackrel{\triangle}{=} S^{\sim_1 \cup^* \sim_2}$, donde $R_1 \cup^* R_2$ es el cierre transitivo de $R_1 \cup R_2$.*

Esas operaciones inducen en $\mathbf{S}$ un orden $\preceq$, a saber $S^{\sim_1} \preceq S^{\sim_2} \Leftrightarrow \sim_1 \supseteq \sim_2$. Vemos entonces que cuanta más información capture la relación, más arriba se encuentra el correspondiente sistema cociente en la ordenación de $\mathbf{S}$. De hecho, este orden le da a este conjunto estructura de retículo, como se demuestra en la siguiente proposición.

Proposición 8.2.6 *$(\boldsymbol{S}, \sqcap, \sqcup)$ es un retículo completo con S y S^t como elementos ínfimo y supremo, respectivamente.*

Demostración. $(\mathbf{S}, \sqcap, \sqcup)$ es un retículo, ya que tanto $\sqcap$ como $\sqcup$ son operadores idempotentes, conmutativos, asociativos y satisfacen la ley de absorción. Para probar que es completo tenemos que ver que todo subconjunto de $\mathbf{S}$ tiene supremo e ínfimo. Sea $\{S^{\sim_i} \mid i \in I\}$ un subconjunto de $\mathbf{S}$ y tomemos $\sim_{inf} = \bigcup^*_{i \in I} \sim_i$ y $\sim_{sup} = \bigcap_{i \in I} \sim_i$. Tanto $\sim_{inf}$ como $\sim_{sup}$ son relaciones de equivalencia monótonas, y $S^{\sim_{inf}}$ y $S^{\sim_{sup}}$ son el ínfimo y el supremo de $\{S^{\sim_i} \mid i \in I\}$, respectivamente. Además, si $\sim$ es una relación de equivalencia monótona entonces $\sim_\top \subseteq \sim \subseteq \sim_\perp$ y, por tanto $S^{\sim_\perp} \preceq S^{\sim} \preceq S^{\sim_\top}$. Para concluir, falta ver que S y S^t son isomorfos a $S^{\sim_\perp}$ y $S^{\sim_\top}$, respectivamente. El segundo isomorfismo se establece trivialmente. Veamos el primero. Consideremos la función h dada por $h(e) = [e]_{\sim_\perp}$ para $e \in E$, es decir, la función de paso al cociente restringida a trazas de longitud 0. Veamos que h es una biyección que satisface las condiciones requeridas. En primer lugar, h es una biyección ya que $\overline{last}([L]_{\sim_\perp}) = last(L)$ es su función inversa. En particular está bien definida, por ser $\sim_\perp$ monótona. Además, como $[e \xrightarrow{a} e']_{\sim_\perp} = [e']_{\sim_\perp}$, se sigue que $e \xrightarrow{a} e' \Leftrightarrow h(e) \stackrel{a}{\mapsto} h(e')$, para cada $e \in Reach(S)$, como requiere la definición 2.2.2.

□

A continuación, enunciamos y demostramos el resultado principal de esta sección.

Proposición 8.2.7 *Sea $S = (E, V, \rightarrow, e_0)$ un sistema de transiciones etiquetadas y $T = (\bar{E}, V, \mapsto, \bar{e}_0)$ un etiquetado de S. Entonces T es isomorfo a un sistema de transiciones cociente de S.*

Demostración. Sea T un etiquetado de S. Tenemos que encontrar una relación monótona $\sim$, tal que T sea isomorfo a $S^{\sim}$. Para ello definimos una función π sobre trazas de S que verifique $O(\pi(L)) = last(L)$. Entonces basta con tomar $\sim$ como el núcleo de π, es decir, $L_1 \sim L_2 \stackrel{\triangle}{\Leftrightarrow} \pi(L_1) = \pi(L_2)$. Para cada traza L

$$\begin{tikzcd}
e_0 \arrow[r, "a"] \arrow[d, dotted, "\pi"'] & e_1 \arrow[r, "b"] \arrow[d, dotted, "\pi"'] & e_2 \arrow[r, "c"] \arrow[d, dotted, "\pi"'] & e_3 \arrow[d, dotted, "\pi"'] \\
\bar{e}_0 \arrow[r, mapsto, "a"'] & \bar{e}_1 \arrow[r, mapsto, "b"'] & \bar{e}_2 \arrow[r, mapsto, "c"'] & \bar{e}_3
\end{tikzcd}$$

Figura 8.6: Diagrama conmutativo inducido por un sistema etiquetado

definimos $\pi(L)$ por inducción sobre la longitud de L. El caso base es inmediato, $\pi(e_0) = \bar{e}_0$, que satisface $O(\pi(e_0)) = e_0 = last(e_0)$. Supongamos cierta la hipótesis de inducción, es decir, que hemos definido $\pi(L) = \bar{e}$ y queremos definir $\pi(L \xrightarrow{a} e')$. Si $e = last(L)$, como $e \xrightarrow{a} e'$ y estamos suponiendo que $O(\bar{e}) = e$, sabemos que existe un único $\bar{e}'$ que satisface la segunda condición en la definición de sistemas etiquetados. Entonces basta tomar $\pi(L \xrightarrow{a} e') = \bar{e}'$, que también satisface que $O(\bar{e}') = last(L \xrightarrow{a} e')$. La relación así definida es monótona y T es isomorfo a $S^{\sim}$. Para verlo basta considerar $h([L]) = \pi(L)$, que está bien definida por la manera en la que $\sim$ se ha definido, es una biyección entre los estados alcanzables y satisface las condiciones que requieren los isomorfismos de sistemas de transiciones.

□

La definición de sistemas etiquetados induce un diagrama conmutativo para cada traza, tal y como se puede ver en la figura 8.6. En el diagrama la función π está definida sobre los cómputos que alcanzan cada e_i. Veamos que el recíproco del resultado anterior también es cierto.

Proposición 8.2.8 *Sea $S = (E, V, \rightarrow, e_0)$ un sistema de transiciones etiquetadas y $\sim$ una relación de equivalencia monótona. Entonces $S^{\sim}$ es isomorfo a un etiquetado de S.*

Demostración. Sea $S = (E, V, \rightarrow, e_0)$ un sistema de transiciones etiquetadas y $\sim$ una equivalencia monótona que relaciona trazas de S. Tenemos que definir un etiquetado de S, $\bar{S}$, que sea isomorfo a $S^{\sim}$. En primer lugar, etiquetaremos todos los términos del lenguaje, de manera que los términos etiquetados se pueden ver como pares (e, Γ), donde $e \in E$. Tomamos como etiquetas los elementos del conjunto $\mathcal{C}/_{\sim}$ de clases de equivalencia definidas por la relación $\sim$. Tomamos $\mapsto$ definida por

$$\frac{e \xrightarrow{a} e' \quad last(L) = e}{(e, [L]) \overset{a}{\mapsto} (e', [L \xrightarrow{a} e'])}$$

La relación de transición $\mapsto$ está bien definida por ser $\sim$ monótona y se verifica que $\bar{S} = (E \times \mathcal{C}/_{\sim}, V, \mapsto, (e_0, [e_0]))$ es un etiquetado de S. Para acabar la

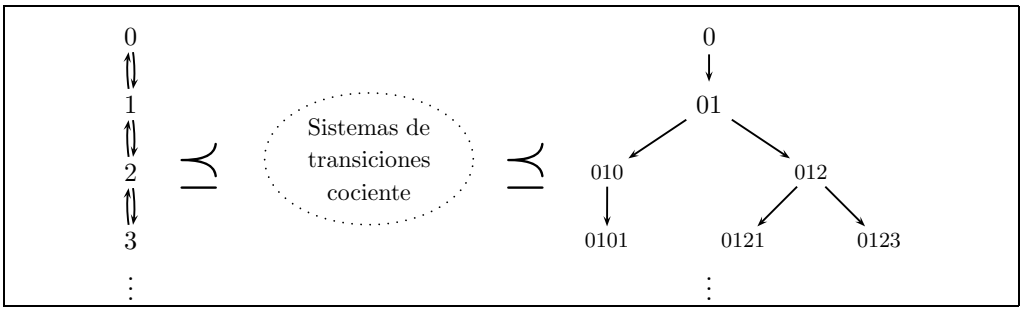

Figura 8.7: Grafo de transiciones (izquierda) y árbol de alcanzabilidad (derecha) del ejemplo 7

demostración, falta comprobar que la función $h([L]) = (last(L), [L])$ define un isomorfismo entre $\bar{S}$ y $S^{\sim}$, lo que es inmediato.

□

En el ejemplo 5 el retículo generado por el sistema de transiciones es trivial, ya que los elementos supremo e ínfimo son isomorfos, al igual que todos los posibles etiquetados. Veamos un ejemplo algo más complejo.

Ejemplo 7 Consideremos el lenguaje $D\pi$[HR02b], que definiremos formalmente en la Sección 8.4. Por el momento no será necesario conocer los detalles formales para entender este ejemplo. Consideremos el proceso $\ell[\![*(go\ \kappa.go\ \ell)]\!]$, que crea sencillos agentes en una localidad llamada ℓ, cuya única tarea consiste en migrar a otra localidad κ para después regresar a ℓ, donde mueren. Usaremos el conjunto de etiquetas $\mathcal{L} = \{\Box, \circ\}$ para etiquetar el sistema de transiciones generado por ese término. Para cada número natural n vamos a denotar por n al término $\kappa[\![go\ \ell]\!] \mid \overset{n}{\ldots} \mid \kappa[\![go\ \ell]\!] \mid \ell[\![*(go\ \kappa.go\ \ell)]\!]$, es decir, el término que representa un estado del sistema en el que hay exactamente n agentes en κ. La figura 8.7 presenta a la izquierda el grafo de transiciones y a la derecha el correspondiente árbol de alcanzabilidad.

Los dos grafos que aparecen en la figura 8.8 definen sendos sistemas de transiciones cociente. El de la izquierda determina la paridad del número de agentes que han muerto, mientras que el de la derecha determina solamente si alguna vez ha muerto algún agente. Por tanto, en el segundo caso el estado $\textcircled{n}$ se corresponde con la traza $01\ldots n$, mientras que el estado $\boxed{n}$ identifica todas las trazas que acaban en el estado n, excepto la traza $01\ldots n$. En términos de la relación de equivalencia monótona, $\sim$ identifica todos los cómputos de la forma $01\ldots n$ (la rama de la derecha del árbol de alcanzabilidad) sólo consigo mismos, mientras que relaciona entre sí todas las trazas que acaban en un mismo estado y no sean de la forma $01\ldots n$, por ejemplo, 0101 y 0121.

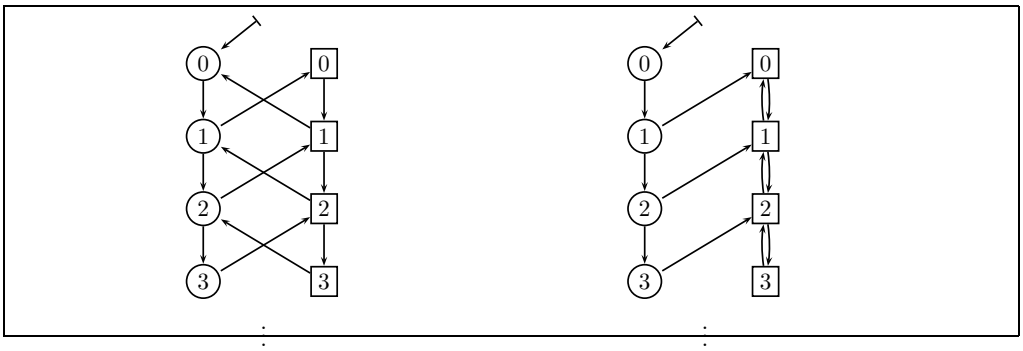

Figura 8.8: Dos sistemas cociente del ejemplo 7

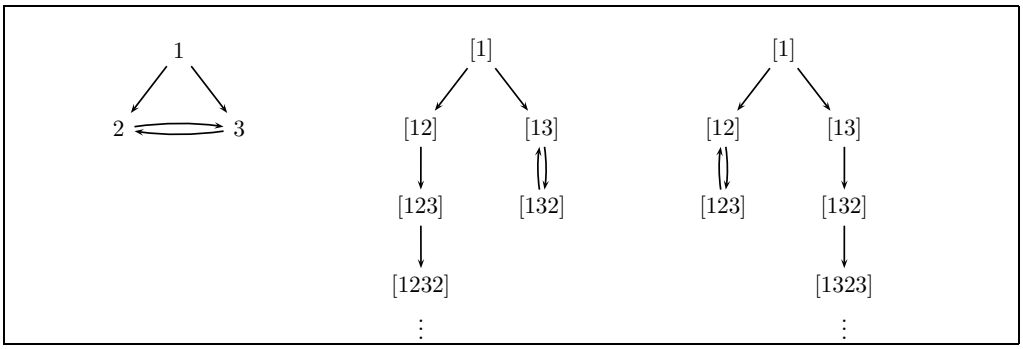

Figura 8.9: Sistemas cociente isomorfos del ejemplo 8

La proposición 8.2.7 afirma que para cada etiquetado de un sistema de transiciones existe un sistema cociente isomorfo a él. Sin embargo, dicho sistema cociente no es necesariamente único, debido esencialmente a que nuestra noción de isomorfía abstrae totalmente la identidad de los estados.

Ejemplo 8 En la figura 8.9, el grafo de la izquierda define un sencillo sistema de transiciones, y los dos árboles son dos sistemas cociente distintos del anterior. Los dos sistemas cociente son isomorfos pero intuitivamente capturan propiedades distintas del sistema original: el de la izquierda identifica cómputos que empiezan por 3, mientras que el de la derecha identifica trazas que empiezan por 2.

Para obtener la unicidad (véase el teorema 8.2.13) necesitaremos una versión refinada de isomorfismo entre sistemas de transiciones, de manera que también se preservan algunas de las propiedades de los estados. Definiremos formalmente estas propiedades mediante funciones que llevan a los estados de sistemas de transiciones distintos a un dominio común.

Definición 8.2.9 *Dados dos sistemas de transiciones* $S = (E, V, \rightarrow, e_0)$ *y* $T = (F, V, \mapsto, f_0)$ *y una función* $f : E \cup F \rightarrow States$ *que lleva a los estados de* E *y* F

*a un conjunto común States, decimos que una función h es un f-*isomorfismo *si se cumplen las dos condiciones siguientes:*

1. *h es un isomorfismo entre sistemas de transiciones,*
2. *$f(h(e)) = f(e)$, para todo $e \in Reach(S)$.*

Escribiremos $S \approx_f T$ cuando exista un f-isomorfismo entre S y T.

Queremos que nuestros isomorfismos preserven el estado alcanzado, tanto en los sistemas etiquetados como en los cocientes. Para ello basta identificar nuestros sistemas de transiciones módulo st-isomorfismos, donde $st(\overline{e}) = O(e)$ y $st([L]_\sim) = last(L)$. Para ser más precisos, aunque abusando un poco de la notación, cuando estemos comparando sistemas cociente usaremos $st : \mathcal{C}/_{\sim_1} \cup \mathcal{C}/_{\sim_2} \to E$, y cuando comparemos un sistema etiquetado con un sistema cociente usaremos $st : \bar{E}_f^{\mathcal{L}} \cup \mathcal{C}/_\sim \to E$. Necesitaremos los siguientes lemas.

Lema 8.2.10 *Si $[L_0]_\sim \overset{a}{\mapsto}_\sim [L_1]_\sim$, $[L_0]_\sim \overset{a}{\mapsto}_\sim [L_2]_\sim$ y $last(L_1) = last(L_2)$, entonces $[L_1]_\sim = [L_2]_\sim$.*

Demostración. Si $[L_0]_\sim \overset{a}{\mapsto}_\sim [L_1]_\sim$ y $[L_0]_\sim \overset{a}{\mapsto}_\sim [L_2]_\sim$ entonces existen L_0', L_0'', L_1' y L_2' tales que $L_0' \sim L_0 \sim L_0''$, $L_i \sim L_i'$, para $i = 1, 2$, $L_0' \overset{a}{\mapsto} L_1'$ y $L_0'' \overset{a}{\mapsto} L_2'$. Como $last(L_1) = last(L_2)$ y $\sim$ es monótona, se sigue que existe un e tal que $L_1' = L_0' \overset{a}{\mapsto} e$ y $L_2' = L_0'' \overset{a}{\mapsto} e$. De nuevo, como $L_0' \sim L_0''$ y $\sim$ es monótona, se cumple que $L_1' \sim L_2'$. Por transitividad de $\sim$, obtenemos que $L_1 \sim L_2$, y por tanto $[L_1]_\sim = [L_2]_\sim$.

□

Lema 8.2.11 *Si los sistemas cociente $S^{\sim_1}$ y $S^{\sim_2}$ son st-isomorfos entonces existe un único st-isomorfismo entre $S^{\sim_1}$ y $S^{\sim_2}$, a saber la función h definida por $h([L]_{\sim_1}) = [L]_{\sim_2}$.*

Demostración. Sea h un st-isomorfismo entre $S^{\sim_1}$ y $S^{\sim_2}$. Veamos que $h([L]_{\sim_1}) = [L]_{\sim_2}$, por inducción sobre la longitud de L. Si $L = e_0$ entonces $h([e_0]_{\sim_1}) = [e_0]_{\sim_2}$, por ser h un isomorfismo y ser $[e_0]_{\sim_1}$ y $[e_0]_{\sim_2}$ los estados iniciales de $S^{\sim_1}$ y $S^{\sim_2}$, respectivamente. Sea ahora $L = L' \overset{a}{\to} e$. La hipótesis de inducción nos dice que $h([L']_{\sim_1}) = [L']_{\sim_2}$. Como se tiene la transición $L' \overset{a}{\mapsto} L$, se cumple que $[L']_{\sim_i} \overset{a}{\mapsto}_{\sim_i} [L]_{\sim_i}$. Como h es un isomorfismo, para $i = 1$ obtenemos que $h([L']_{\sim_1}) = [L']_{\sim_2} \overset{a}{\mapsto}_{\sim_2} h([L]_{\sim_1})$. Sea $L'' \in \mathcal{C}$ tal que $[L'']_{\sim_2} = h([L]_{\sim_1})$. Como h preserva estados, $last(L'') = st(h([L]_{\sim_1})) = st([L]_{\sim_1}) = last(L)$. Ahora basta con aplicar el lema anterior para concluir que $h([L]_{\sim_1}) = [L'']_{\sim_2} = [L]_{\sim_2}$.

□

Lema 8.2.12 *Si $S^{\sim_1} \approx_{st} S^{\sim_2}$ entonces $\sim_1 = \sim_2$.*

Demostración. Como $S^{\sim_1} \approx_{st} S^{\sim_2}$, existe un st-isomorfismo h como el del lema anterior. Entonces $L \sim_1 L' \Leftrightarrow [L]_{\sim_1} = [L']_{\sim_1} \Leftrightarrow h([L]_{\sim_1}) = h([L']_{\sim_1}) \Leftrightarrow [L]_{\sim_2} = [L']_{\sim_2} \Leftrightarrow L \sim_2 L'$, donde la segunda equivalencia se cumple por ser h biyectiva.

□

Obsérvese que el ejemplo 8 no era un contraejemplo para el lema anterior, ya que aunque $S^{\sim_1}$ y $S^{\sim_2}$ eran isomorfos, no eran st-isomorfos. Enunciamos y probamos ahora el teorema que perseguimos.

Teorema 8.2.13 *Si T es un etiquetado de S entonces existe un único sistema cociente $S^{\sim}$ tal que $T \approx_{st} S^{\sim}$.*

Demostración.

- Existencia: la equivalencia $\sim$ y la función h, tal y como fueron definidas en la demostración de la proposición 8.2.7, cumplía $h([L]_{\sim}) = \pi(L)$, con lo que h define un isomorfismo entre T y $S^{\sim}$. Además, h es un st-isomorfismo, ya que $st(h([L]_{\sim})) = st(\pi(L)) = last(L) = st([L]_{\sim})$.

- Unicidad: Supongamos que existen $\sim_1$ y $\sim_2$ tales que $S^{\sim_1} \approx_{st} T$ y $T \approx_{st} S^{\sim_2}$. Por transitividad de $\approx_{st}$ se sigue que $S^{\sim_1} \approx_{st} S^{\sim_2}$ y aplicando el anterior lema se obtiene que $S^{\sim_1} = S^{\sim_2}$.

□

8.3. Derivación de lenguajes etiquetados

En esta sección nos enfrentamos al problema de decidir si una semántica operacional, definida como una relación de reducción sobre una sintaxis etiquetada, es de hecho un etiquetado (y por tanto un cociente) de la semántica original.

Las semánticas etiquetadas están parametrizadas por el conjunto de posibles formas de inicializar las etiquetas, formalizadas por funciones $tag : E \to \bar{E}$, ya que la función elegida sólo depende de la aplicación específica en la que estemos interesados. En particular, las funciones de etiquetado han de cumplir que $O \circ tag = Id_E$, pero no al revés. Típicamente, estas funciones capturan en las etiquetas la información estática necesaria para formalizar la propiedad en la que estamos interesados.

En general, una semántica sobre una sintaxis etiquetada es un etiquetado de otra semántica operacional, con respecto a una función de etiquetado tag, si para cada P el sistema de transiciones que genera $tag(P)$ es un etiquetado del sistema de transiciones generado por P.

Definición 8.3.1 *Diremos que una relación de reducción* $\bar{\mathcal{R}} = (\bar{E}, V, \mapsto)$ *es un* etiquetado *de* $\mathcal{R} = (E, V, \rightarrow)$ *con respecto a la función de etiquetado inicial tag, si para cada* $e \in E$ *el sistema de transiciones etiquetadas* $(\bar{E}, V, \mapsto, tag(e))$ *es un etiquetado de* $(E, V, \rightarrow, e)$.

En lo que sigue consideraremos sólo un caso particular de la definición anterior, que será suficiente para nosotros: aquél en el que la relación de reducciones está definida estructuralmente. Supongamos que $\rightarrow$ queda definida por medio de un conjunto de axiomas $A_i : P_i \stackrel{a_i}{\rightarrow} Q_i$ para $i = 1, 2, \ldots$ y un conjunto de reglas estructurales R_i de la forma

$$\frac{P \stackrel{a}{\rightarrow} Q}{C_i(P) \stackrel{a}{\rightarrow} C_i(Q)}$$

con contextos $C_i = f^i(t_1, .., \Box, .., t_{n_i})$, $i = 1, 2, \ldots$, donde las funciones f^i que aparecen en los contextos no son símbolos distinguidos de la sintaxis. Si se cumplen estas condiciones, estas reglas estructurales pueden verse directamente como reglas para la sintaxis etiquetada. Así, basta con sustituir cada axioma A_i por el conjunto de versiones suyas etiquetadas $\overline{A}_i$ para cada manera de etiquetar los P_i, para obtener un etiquetado de $\mathcal{R}$.

Proposición 8.3.2 *Consideremos* $\mathcal{R}$ *definida por medio de los axiomas* A_i *y las reglas* R_i*, de la forma anterior. Consideremos también un conjunto de axiomas* $\bar{A}_i : \bar{P}_i \stackrel{a_i}{\mapsto} \bar{Q}_i$*, con* $i = 1, 2, \ldots$*, tales que* $O(\bar{P}_i) = P_i$ *y* $O(\bar{Q}_i) = Q_i$*. Supongamos que cada* $\bar{A}_i$ *está definido para todas las maneras posibles de etiquetar*[**] P_i *y sea* $\bar{\mathcal{R}}$ *la relación de reducción definida por el conjunto de axiomas* $\bar{A}_1, \bar{A}_2, \ldots$ *y el conjunto de reglas composicionales* $R_1, R_2, \ldots$ *Entonces* $\bar{\mathcal{R}}$ *es un etiquetado de* $\mathcal{R}$.

Demostración. Dado P_0, probamos la primera condición de los sistemas etiquetados por inducción sobre las reglas que definen $\mapsto$ y la segunda condición por inducción sobre las reglas que definen $\rightarrow$:

1. Si $\bar{P} \stackrel{a}{\mapsto} \bar{P}'$ entonces $O(\bar{P}) \stackrel{a}{\rightarrow} O(\bar{P}')$

   a) $\bar{A}_i : \bar{P}_i \stackrel{a_i}{\mapsto} \bar{Q}_i$. Por hipótesis $A_i : O(\bar{P}_i) \stackrel{a_i}{\mapsto} O(\bar{Q}_i)$.

   b) Para una mejor legibilidad de la demostración, supongamos que f^i es un constructor unario, de modo que tenemos que $R_i : \bar{P} = f^i(\bar{Q}) \stackrel{a}{\mapsto} f^i(\bar{Q}') = \bar{P}'$ con $\bar{Q} \stackrel{a}{\mapsto} \bar{Q}'$. Por hipótesis de inducción tenemos que $O(\bar{Q}) \stackrel{a}{\rightarrow} O(\bar{Q}')$. Como el símbolo f^i no es un símbolo distinguido de la sintaxis, $O(\bar{P}) = O(f^i(\bar{Q})) = f^i(O(\bar{Q}))$, por lo que podemos aplicar la regla R_i para obtener que $O(\bar{P}) = f^i(O(\bar{Q})) \stackrel{a}{\rightarrow} f^i(O(\bar{Q}')) = O(f^i(\bar{Q}')) = O(\bar{P}')$.

[**]Para ser exactos, tendremos un axioma distinto $\bar{A}_{i,\bar{P}_i} : \bar{P}_i \stackrel{a_i}{\rightarrow} \bar{Q}_i$ para cada $\bar{P}_i$ tal que $O(\bar{P}_i) = P_i$, pero habitualmente todos ellos tendrán una estructura homogénea, por lo que usamos esta notación más concisa.

2. Si $P \xrightarrow{a} P'$ y $\bar{P} \in Reach(tag(P_0))$ tales que $O(\bar{P}) = P$ entonces existe un único $\bar{P}'$ con $O(\bar{P}') = P'$ y $\bar{P} \overset{a}{\mapsto} \bar{P}'$:

   a) $A_i : P_i \xrightarrow{a_i} Q_i$. Por hipótesis, para cada etiquetado $\bar{P}_i$ de P_i tenemos una única versión etiquetada del axioma, $\bar{A}_i$.

   b) De nuevo, supongamos que $P = f^i(Q) \xrightarrow{a} P' = f^i(Q')$ con $Q \xrightarrow{a} Q'$. Dado $\bar{P}$ como lo tomamos antes, al no ser f^i un símbolo distinguido, se tiene que $\bar{P} = f^i(\bar{Q})$, con $O(\bar{Q}) = Q$. Con lo que podemos aplicar la hipótesis de inducción: existe un único $\bar{Q}'$ tal que $O(\bar{Q}') = Q'$ y $\bar{Q} \overset{a}{\mapsto} \bar{Q}'$. Pues bien, $\bar{P}' = f^i(\bar{Q}')$, es el único término tal que $O(\bar{P}') = P'$ y $\bar{P} \overset{a}{\mapsto} \bar{P}'$.

□

Esta es la forma más simple de derivar una semántica etiquetada a partir de una semántica dada. Cuando la semántica está definida por medio de una congruencia estructural, de manera que ni en los axiomas ni en las reglas de su definición aparecen símbolos distinguidos de la sintaxis etiquetada, también se puede proceder de la misma manera. En este caso, tanto los axiomas como las reglas de la congruencia pueden ser utilizados para definir una congruencia estructural entre términos etiquetados. Para ello basta con añadir al conjunto anterior de reglas dadas la regla de congruencia para obtener de nuevo una semántica etiquetada de la original. Precisamente este es el caso que detallaremos en la siguiente sección a través del prometido ejemplo.

8.4. Lenguaje $D\pi$ etiquetado

En esta sección vamos a ilustrar la metodología propuesta, aplicándosela al lenguaje $D\pi$ [HR02b], una versión distribuida del cálculo π. De hecho, por simplicidad, usaremos la versión del cálculo presentada en [HMR03]. En [HR02b] se define un análisis de tipos para la detección de sistemas en los que se pueden producir violaciones de la política de permisos de acceso a los recursos, que son simplemente canales. Para probar la corrección del sistema de tipos, los autores definen una semántica extendida sobre una sintaxis etiquetada, en la que se anota a los agentes con los permisos que éstos acumulan a lo largo de la ejecución del sistema. Prueban después un resultado de seguridad (los sistemas tipables no producen violaciones de permisos) y un teorema de reducción del sujeto (los sistemas preservan su tipo a lo largo de la ejecución), de manera que se concluye que los sistemas con agentes etiquetados para los que se puede derivar un tipo nunca producen violaciones en la política de seguridad, con respecto a la semántica extendida. Sin embargo, no se formaliza ninguna conexión entre la semántica original y la extendida, de modo que, en principio, la propiedad capturada por el sistema de tipos es una propiedad sólo de la semántica extendida, que no está relacionada de modo claro con la semántica original.

Nombres		**Valores**	
$e ::= k$	localidad	$u, v ::=$ bv	valores básicos
$\mid a$	canal	$\mid e$	nombre
		$\mid x$	variable
Patrones		$\mid u@v$	nombres localizados
$X, Y ::= x$	variable	$\mid \tilde{u}$	tupla
$\mid X@z$	patrón localizado		
$\mid \tilde{X}$	tupla		
Procesos		**Sistemas**	
$P, Q ::= stop$	terminación	$M, N ::= 0$	vacío
$\mid P \mid Q$	composición	$\mid M \mid N$	composición
$\mid *P$	replicación	$\mid (\nu_k e : E)N$	restricción
$\mid a?(X : T).P$	entrada	$\mid k[\![P]\!]$	agente
$\mid a!\langle v \rangle P$	salida		
$\mid if\ u = v\ then\ P$ $else\ Q$	condicional		
$\mid go\ u.P$	movimiento		
$\mid (\nu e : E)P$	restricción		

Figura 8.10: Sintaxis del $D\pi$

(S-EXTR)	$M \mid (\nu_k e : E)N$	$\equiv$	$(\nu_k e : E)(M \mid N)$	si $e \notin fn(M)$
(S-GARB$_1$)	$(\nu_k e : E)stop$	$\equiv$	$stop$	
(S-GARB$_2$)	$k[\![stop]\!]$	$\equiv$	$stop$	

Figura 8.11: Equivalencia estructural del lenguaje $D\pi$

En esta sección usamos la metodología desarrollada anteriormente para proporcionar la deseada relación entre ambas semánticas, probando que el $D\pi$ etiquetado es un etiquetado según nuestra definición, y que, por tanto, se trata de un cociente, de manera que las propiedades capturadas por el sistema de tipos lo son también del sistema original. Además, nuestra aproximación es general, en el sentido de que podríamos hacer lo mismo para otras propiedades dependientes de la historia.

En la figura 8.10 se define la sintaxis del lenguaje $D\pi$. Además de las primitivas habituales del cálculo π (terminación, composición, replicación, entrada, salida y operador condicional [MPW92]), se introduce un nuevo tipo de término, $k[\![P]\!]$, cuyo significado es que el proceso P se encuentra en la localidad k, y una primitiva *go* para la movilidad. Los nombres que se crean, además de canales, pueden ser localidades. Estos nombres son locales, de modo que si se quieren hacer referencias globales habría que usar nombres localizados, como $u@v$.

En las figuras 8.11 y 8.12 se encuentra la definición de la semántica opera-

(R-GO)	$\ell[\![go\ k.P]\!] \rightarrow k[\![P]\!]$	
(R-COMM)	$k[\![a!\langle v\rangle P]\!] \mid k[\![a?(X:T).Q]\!] \rightarrow k[\![P]\!] \mid k[\![Q\{X := v\}]\!]$	
(R-EQ$_1$)	$k[\![if\ u = u\ then\ P\ else\ Q]\!] \rightarrow k[\![P]\!]$	
(R-EQ$_2$)	$k[\![if\ u = v\ then\ P\ else\ Q]\!] \rightarrow k[\![Q]\!]$	si $u \neq v$
(S-COPY)	$k[\![*P]\!] \rightarrow k[\![P]\!] \mid k[\![*P]\!]$	
(S-SPLIT)	$k[\![P \mid Q]\!] \rightarrow k[\![P]\!] \mid k[\![Q]\!]$	
(S-NEW)	$k[\![(\nu e : E)P]\!] \rightarrow (\nu_k e : E)k[\![P]\!]$	si $e \neq k$

$$\text{(R-STR)}\quad \frac{N \rightarrow N'}{(\nu e)N \rightarrow (\nu e)N'} \qquad \frac{N \rightarrow N'}{M \mid N \rightarrow M \mid N'} \qquad \frac{N \equiv M \quad M \rightarrow M' \quad M' \equiv N'}{N \rightarrow N'}$$

Figura 8.12: Semántica del lenguaje $D\pi$

cional del lenguaje. La congruencia estructural $\equiv$ es la menor congruencia que cumple los axiomas de la figura 8.11. La mayoría de las reglas en ambas figuras son análogas a las correspondientes del cálculo π. Las únicas reglas específicas de $D\pi$ son (R-GO) y (R-COMM). La primera de ellas nos indica que el proceso que ejecuta una instrucción *go* se mueve de una localidad a otra, mientras que la segunda establece que la comunicación es local, es decir, restringiendo la comunicación de modo que sólo pueden comunicarse procesos que se encuentren en la misma localidad.

El $D\pi$ etiquetado es una versión del lenguaje $D\pi$ en la que el operador $[\![\ ]\!]$ se sustituye por $[\![\ ]\!]_\Gamma$, donde Γ recorre los entornos de tipo, es decir, funciones parciales de identificadores de localidad a $\mathcal{K}$, el conjunto de *tipos localidad*, cuya definición formal es irrelevante para nuestros propósitos [HR02b]. Intuitivamente, los tipos localidad asignan a los agentes permisos para acceder a los canales de cada localidad. Por ejemplo, si un agente está etiquetado por Γ y $\Gamma(k) = K$ entonces los permisos de dicho agente en la localidad k son los especificados por K.

La semántica de $D\pi$ etiquetado la definen los axiomas y reglas de la figura 8.13, donde $\Gamma, \{_k e : E\}$ denota la extensión de Γ en k con el nuevo nombre e, al que se le asigna tipo E. La regla más importante de la semántica etiquetada es (R-COMM), responsable de que los agentes puedan adquirir nuevos permisos a través de la comunicación con otros agentes.

Por tanto, el conjunto de etiquetas $\mathcal{L}$ es el conjunto de los entornos de tipo, y el símbolo distinguido es $[\![\ ]\!]$. Basta con examinar las reglas que definen el lenguaje etiquetado para ver que satisfacen las condiciones impuestas en la sección anterior. Efectivamente, tenemos una versión etiquetada de todos los axiomas, para cualquier forma de etiquetar el término de la parte izquierda del axioma. El resto de las reglas del lenguaje original pueden ser consideradas como reglas del lenguaje etiquetado, ya que el símbolo $[\![\ ]\!]$ no aparece en ellas. Por tanto, como corolario de la proposición 8.3.2, tenemos el siguiente resultado.

$$
\begin{array}{lll}
\text{(R-GO)} & \ell[\![go\ k.P]\!]_\Gamma \rightarrow k[\![P]\!]_\Gamma & \\
\text{(R-COMM)} & k[\![a!\langle v\rangle P]\!]_\Gamma \mid k[\![a?(X:T).Q]\!]_\Delta \rightarrow k[\![P]\!]_\Gamma \mid k[\![Q\{X:=v\}]\!]_{\Delta\sqcap\{_k v:T\}} & \\
\text{(R-EQ}_1\text{)} & k[\![if\ u=u\ then\ P\ else\ Q]\!]_\Gamma \rightarrow k[\![P]\!]_\Gamma & \\
\text{(R-EQ}_2\text{)} & k[\![if\ u=v\ then\ P\ else\ Q]\!]_\Gamma \rightarrow k[\![Q]\!]_\Gamma & \text{si} \quad u \neq v \\
\text{(S-COPY)} & k[\![*P]\!]_\Gamma \rightarrow k[\![P]\!]_\Gamma \mid k[\![*P]\!]_\Gamma & \\
\text{(S-SPLIT)} & k[\![P \mid Q]\!]_\Gamma \rightarrow k[\![P]\!]_\Gamma \mid k[\![Q]\!]_\Gamma & \\
\text{(S-NEW)} & k[\![(\nu e:E)P]\!]_\Gamma \rightarrow (\nu_k e:E)k[\![P]\!]_{\Gamma,\{_k e:E\}} & \text{si} \quad e \neq k
\end{array}
$$

$$
\text{(R-STR)} \quad \frac{N \rightarrow N'}{(\nu e)N \rightarrow (\nu e)N'} \qquad \frac{N \rightarrow N'}{M \mid N \rightarrow M \mid N'} \qquad \frac{N \equiv M \quad M \rightarrow M' \quad M' \equiv N'}{N \rightarrow N'}
$$

Figura 8.13: Semántica del $D\pi$ etiquetado

Teorema 8.4.1 *El $D\pi$ etiquetado es un* etiquetado *de $D\pi$.*

Según este resultado, y según nuestra caracterización de los sistemas etiquetados, el sistema de transiciones generado por cada término de $D\pi$ etiquetado es isomorfo a un cociente del sistema de transiciones generado por el mismo término, sin etiquetar. Por tanto, toda propiedad definida para el término $D\pi$ etiquetado es también una propiedad del término sin etiquetar. En particular, todo conjunto de términos etiquetados (por ejemplo, los términos erróneos que violan la política de permisos) define un conjunto de cómputos (los cómputos erróneos).

Este resultado es cierto independientemente de la función de etiquetado inicial que se tome. Sin embargo, la elección de esta función es crucial para formalizar de manera adecuada los errores de tiempo de ejecución que tenemos en mente, en este caso los errores de acceso a recursos. En [HR02b] se define *tag* solamente para procesos del cálculo original que pueden ser tipados, de manera que si M no es tipable entonces $tag(M) = \emptyset$. Consecuentemente, no es posible describir los errores de acceso a recursos producidos por sistemas sin tipo.

Para hacer independiente la definición de error de acceso a recursos del sistema de tipos (que no debe ser más que un mecanismo estático para prevenir la aparición de los errores definidos), proponemos una alternativa a la función de etiquetado inicial de [HR02b], definida para cualquier sistema, y no sólo para aquéllos que tienen tipo. Para ello, añadimos a las etiquetas los nombres creados por los agentes, que son exactamente los que éstos pueden usar inicialmente. Usaremos funciones $tag_\Gamma(M)$, donde Γ representa los permisos iniciales de M. Consideramos el sistema definido por $(\nu a:A)(k[\![b!\langle a\rangle]\!] \mid k[\![b?(x:A).x!\langle c\rangle]\!])$. Aunque el canal a se crea en el ámbito de los dos agentes, sólo el agente de la izquierda conoce a. De hecho, siempre se puede extender el ámbito de los nombres por medio de la extrusión. En particular, el sistema anterior y $(\nu a:A)k[\![b!\langle a\rangle]\!] \mid k[\![b?(x:A).x!\langle c\rangle]\!]$ son equivalentes. Para lidiar con este hecho, restringimos el dominio de $tag_\Gamma(M)$ al conjunto de nombres libres de M, es decir, siempre se cumplirá el invariante

$dom(\Gamma) = fn(M)$. Denotaremos mediante Γ_M a la restricción $\Gamma\mid_{fn(M)}$ y mediante Φ a la función parcial vacía (con dominio vacía) y rango $\mathcal{L}$. A continuación definimos la función de etiquetado inicial.

Definición 8.4.2 *Definimos la función* $tag : \mathcal{M} \to \bar{\mathcal{M}}$ *por medio de:*

- $tag(M) = tag_\Phi(M)$ *si* M *es un sistema cerrado (sin variables)*
- $tag_\Gamma(0) = 0$
- $tag_\Gamma(M \mid N) = tag_{\Gamma_M}(M) \mid tag_{\Gamma_N}(N)$
- $tag_\Gamma((\nu_\ell e : E)M) = (\nu_\ell e : E)tag_{\Gamma,\{_\ell e:E\}}(M)$ *si* $e \in fn(M)$
- $tag_\Gamma((\nu_\ell e : E)M) = (\nu_\ell e : E)tag_\Gamma(M)$ *si* $e \notin fn(M)$
- $tag_\Gamma(\ell[\![P]\!]) = \ell[\![P]\!]_\Gamma$.

En [HR02b] se definen los errores de ejecución para los sistemas etiquetados, escribiéndose $M \to err$ si M puede producir un error de ejecución. Para eliminar estos errores se define un sistema de tipos $\vdash$ para el lenguaje original y otro $\Vdash$ para los sistemas etiquetados. Pues bien, con nuestra función de etiquetado inicial se cumple el siguiente resultado.

Lema 8.4.3 *Si* $\Gamma \vdash M$ *entonces* $\Gamma \Vdash tag_\Gamma(M)$.

Demostración. La demostración es inmediata por inducción sobre las reglas usadas para derivar $\Gamma \vdash M$ (véase [HR02b]):

1. $\Gamma \vdash 0$. Como $tag_\Gamma(0) = 0$ y $\Gamma \Vdash 0$, se sigue la tesis.
2. $\Gamma \vdash k[\![P]\!]$. Entonces se cumple que $\Gamma \vdash_k P$. Como $\Gamma <: \Gamma$ se sigue que $\Gamma \Vdash k[\![P]\!]$.
3. $\Gamma \vdash M_1 \mid M_2$ y $\Gamma \vdash M_i$. Por reforzamiento se cumple que $\Gamma_{M_i} \vdash M_i$, por hipótesis de inducción, $\Gamma_{M_i} \Vdash tag_{\Gamma_{M_i}}(M_i)$ y ahora por debilitamiento, $\Gamma \Vdash tag_{\Gamma_{M_i}}(M_i)$. Como $tag_\Gamma(M_1 \mid M_2) = tag_{\Gamma_{M_1}}(M_1) \mid tag_{\Gamma_{M_2}}(M_2)$ se sigue que $\Gamma \Vdash tag_\Gamma(M_1 \mid M_2)$.
4. $\Gamma \vdash (\nu_k e : E)M$ con $\Gamma, \{_k e : E\} \vdash M$. Por hipótesis de inducción se cumple que $\Gamma, \{_k e : E\} \Vdash tag_{\Gamma,\{_k e:E\}}(M)$.
   a) Si $e \in fn(M)$ entonces $tag((\nu_k e : E)M) = (\nu_k e : E)tag_{\Gamma,\{_k e:E\}}(M)$ y podemos concluir que $\Gamma \Vdash tag_\Gamma((\nu_k e : E)M)$.
   b) Si $e \notin fn(M)$, por reforzamiento también se cumple que $\Gamma \Vdash tag_{\Gamma,\{_k e:E\}}(M)$, y como $tag((\nu_k e : E)M) = (\nu_k e : E)tag_{\Gamma,\{_k e:E\}}(M)$, se sigue la tesis.

$\square$

En [HR02b] se demuestran un teorema de reducción del sujeto y otro de seguridad del lenguaje etiquetado respecto del sistema de tipos. A partir de ellos se concluyen los siguientes resultados de seguridad.

Teorema 8.4.4 *Si* $\Gamma \vdash M$ *y* $tag_\Gamma(M) \mapsto^* M'$ *entonces* $M' \not\rightarrow err$.

Corolario 8.4.5 *Si* $\Phi \vdash M$ *y* $tag(M) \mapsto^* M'$ *entonces* $M' \not\rightarrow err$.

Aunque este resultado también aparecía en [HR02b], allí no se hacía ninguna interpretación suya en términos de la semántica original del lenguaje. Ahora podemos interpretar $M \not\rightarrow err$ como la imposibilidad de usar un recurso cuando no se ha obtenido permiso para hacerlo a lo largo del cómputo, ya que ésta es justamente la información capturada por las etiquetas.

Ejemplo 9 Consideremos el proceso

$$M = (\nu\ell : loc\{a : res\langle T\rangle\})(\nu_k b : res\langle T\rangle)(\nu_\ell c : T)\ell[\![b!\langle c\rangle]\!]$$

Podemos etiquetar M, obteniendo

$$tag(M) = (\nu\ell : loc\{a : res\langle T\rangle\})(\nu_k b : res\langle T\rangle)(\nu_\ell c : T)\ell[\![b!\langle c\rangle]\!]_\Gamma$$

con

$$\Gamma = \{\ell : loc\{a : res\langle T\rangle, c : T\}, k : loc\{b : res\langle T\rangle\}\}$$

Sin embargo, M no puede ser tipado y, de hecho, $tag(M) \rightarrow err$, ya que intenta usar el canal b en la localidad l, lo que no está permitido según Γ. Evidentemente, ya que el sistema de tipos no es completo, podría darse el caso de que M no tuviera tipo, aunque $tag(M)$ no produjera ningún error.

8.5. Aplicaciones de la estructura algebraica del retículo de sistemas cociente

En esta sección describiremos brevemente cómo se puede aprovechar la estructura algebraica del retículo de sistemas cociente, en particular para combinar distintos etiquetados del mismo sistema. Empezamos definiendo la mezcla de dos etiquetados distintos en cierta sintaxis.

Definición 8.5.1 *Sean* $(\mathcal{S}, \Sigma)$ *una signatura heterogénea,* E *el álgebra de* $\mathcal{S}$*-términos de* Σ*,* f_1 *y* f_2 *dos símbolos de* Σ *y* $\mathcal{L}_1$ *y* $\mathcal{L}_2$ *dos conjuntos (no vacíos) de etiquetas. Definimos la* mezcla *de* $\bar{E}^{\mathcal{L}_1}_{f_1}$ *y* $\bar{E}^{\mathcal{L}_2}_{f_2}$*, y la denotamos por* $\bar{E}^{\mathcal{L}_1}_{f_1} \oplus \bar{E}^{\mathcal{L}_2}_{f_2}$*, como* $\bar{E}^{\mathcal{L}_1,\mathcal{L}_2}_{f_1,f_2}$ *si* $f_1 \neq f_2$ *o como* $\bar{E}^{\mathcal{L}_1\times\mathcal{L}_2}_{f_1}$ *si* $f_1 = f_2$*. Esta definición se generaliza fácilmente para considerar varios símbolos distinguidos y varios conjuntos de etiquetas.*

$$\begin{tikzcd}
 & \bar{E}_{f_1}^{\mathcal{L}_1} & & \\
E & & \bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2} \arrow[ul, "O_1"'] \arrow[ll, "O"'] \arrow[dl, "O_2"] & \bar{E}_{f_1}^{\mathcal{L}_1} \times \bar{E}_{f_2}^{\mathcal{L}_2} \arrow[l, "\oplus"'] \\
 & \bar{E}_{f_2}^{\mathcal{L}_2} \arrow[ul, "O"] &
\end{tikzcd}$$

Figura 8.14: Operadores O_1, O_2 y $\oplus$

Si O_1 y O_2 son las funciones que borran las etiquetas de $\mathcal{L}_2$ y $\mathcal{L}_1$, respectivamente, dados $\overline{e}^1 \in \bar{E}_{f_1}^{\mathcal{L}_1}$ y $\overline{e}^2 \in \bar{E}_{f_2}^{\mathcal{L}_2}$ tales que $O(\overline{e}^1) = O(\overline{e}^2)$ existe un único $\overline{e} \in \bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}$, que denotamos $\overline{e}^1 \oplus \overline{e}^2$, tal que $O_1(\overline{e}) = \overline{e}^1$ y $O_2(\overline{e}) = \overline{e}^2$. Por tanto, podemos definir el operador parcial $\oplus$ del conjunto $\bar{E}_{f_1}^{\mathcal{L}_1} \times \bar{E}_{f_2}^{\mathcal{L}_2}$ a $\bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}$, tal y como se muestra en la figura 8.5. Este operador toma un término etiquetado de dos maneras distintas y combina esas etiquetas.

A continuación usamos las notaciones definidas para combinar distintos etiquetados.

Definición 8.5.2 *Sea $S = (E, V, \rightarrow, e_0)$ un sistema de transiciones etiquetadas y $T_1 = (\bar{E}_{f_1}^{\mathcal{L}_1}, V, \mapsto_1, \overline{e_0}^1)$ y $T_2 = (\bar{E}_{f_2}^{\mathcal{L}_2}, V, \mapsto_2, \overline{e_0}^2)$ dos etiquetados de S. Definimos la* mezcla *de T_1 y T_2 como $T_1 \oplus T_2 = (\bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}, V, \mapsto, \overline{e_0}^1 \oplus \overline{e_0}^2)$, donde $\mapsto$ se define por medio de:*

$$\frac{O_i(\overline{e_1}) \overset{a}{\mapsto}_i O_i(\overline{e_2}) \quad i = 1, 2}{\overline{e_1} \overset{a}{\mapsto} \overline{e_2}}$$

Proposición 8.5.3 *Si T_1 y T_2 son etiquetados del sistema de transiciones S, entonces $T_1 \oplus T_2$ es un etiquetado de S.*

Demostración. Tenemos que comprobar que se cumplen las dos condiciones para que un sistema de transiciones sobre la sintaxis etiquetada sea de hecho un sistema etiquetado:

- Si $\overline{e_1} \overset{a}{\mapsto} \overline{e_2}$ entonces $O(\overline{e_1}) \overset{a}{\rightarrow} O(\overline{e_2})$: por definición de $\mapsto$, $O_1(\overline{e_1}) \overset{a}{\mapsto}_1 O_1(\overline{e_2})$. T_1 es un etiquetado de S y, por tanto, $O(O_1(\overline{e_1})) \overset{a}{\rightarrow} O(O_1(\overline{e_2}))$, es decir (véase el diagrama), $O(\overline{e_1}) \overset{a}{\rightarrow} O(\overline{e_2})$.

- Sean $e_1 \overset{a}{\rightarrow} e_2$ y $\overline{e_1} \in Reach(T_1 \oplus T_2)$ tales que $O(\overline{e_1}) = e_1$. Se cumple $\overline{e_1} = \overline{e_1}^1 \oplus \overline{e_1}^2$ con $\overline{e_1}^i \in Reach(T_i)$ (i=1,2). Como T_1 y T_2 son etiquetados de S y $O(\overline{e_1}^i) = e_1$ existen un único $\overline{e_2}^1$ y un único $\overline{e_2}^2$ tales que $\overline{e_1}^i \overset{a}{\mapsto}_i \overline{e_2}^i$.

Entonces basta tomar $\overline{e_2} = \overline{e_2}^1 \oplus \overline{e_2}^2$, que es el único término de $\bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}$ tal que $O(\overline{e_2}) = e_2$ y $\overline{e_1} \overset{a}{\mapsto} \overline{e_2}$.

□

Es más, $T_1 \oplus T_2$ captura exactamente aquéllas propiedades capturadas tanto por T_1 como por T_2, tal y como formaliza el siguiente teorema.

Teorema 8.5.4 *Sean $S = (E, V, \rightarrow, e_0)$ un sistema de transiciones etiquetadas y $T_1 = (\bar{E}_{f_1}^{\mathcal{L}_1}, V, \mapsto_1, \overline{e_0}^1)$ y $T_2 = (\bar{E}_{f_2}^{\mathcal{L}_2}, V, \mapsto_2, \overline{e_0}^2)$ dos etiquetados de S. Supongamos que $S^{\sim_1}$ y $S^{\sim_2}$ son los sistemas cociente st-isomorfos a T_1 y a T_2, respectivamente. Entonces $T_1 \oplus T_2$ es st-isomorfo a $S^{\sim_1} \sqcup S^{\sim_2}$.*

Demostración. Supongamos que h_i es un st-isomorfismo entre $S^{\sim_i}$ y T_i, para $i = 1, 2$, y tomemos $h : \mathcal{C}/_\sim \rightarrow \bar{E}_{f_1}^{\mathcal{L}_1} \oplus \bar{E}_{f_2}^{\mathcal{L}_2}$, definida como $h([L]_\sim) = h_1([L]_{\sim_1}) \oplus h_2([L]_{\sim_2})$, donde $\sim = \sim_1 \cap \sim_2$.

- h está bien definida: si $L \sim L'$ entonces $L \sim_i L'$ para $i = 1, 2$ por definición de $\sim$. Como h_i está bien definida, $h_i([L]_{\sim_i}) = h_i([L']_{\sim_i})$ para $i = 1, 2$, y por tanto, $h([L]_\sim) = h([L']_\sim)$. Además, $\oplus$ está definido para $h_1([L]_{\sim_1})$ y $h_2([L]_{\sim_2})$ porque $O(h_i([L]_{\sim_i})) = st(h_i([L]_{\sim_i})) = st([L]_{\sim_i}) = last(L)$ para $i = 1, 2$, y por tanto, $O(h_1([L]_{\sim_1})) = O(h_2([L]_{\sim_2}))$.

- h es inyectiva: $h([L]_\sim) = h([L']_\sim) \Rightarrow h_1([L]_{\sim_1}) \oplus h_2([L]_{\sim_2}) = h_1([L']_{\sim_1}) \oplus h_2([L']_{\sim_2})$. Esto implica (por definición de $\oplus$) que $h_i([L]_{\sim_i}) = h_i([L']_{\sim_i})$ para $i = 1, 2$. Como h_1 y h_2 son inyectivas, $[L]_{\sim_i} = [L']_{\sim_i}$ para $i = 1, 2$ y por tanto, $[L]_\sim = [L']_\sim$.

- h es sobreyectiva: sea $\overline{e} \in Reach(T_1 \oplus T_2)$ y tomemos $O_i(\overline{e}) = \overline{e}^i \in Reach(T_i)$. Como h_i es sobreyectiva existe $[L_i]_{\sim_i}$ tal que $h_i([L_i]_{\sim_i}) = \overline{e}^i$. De nuevo, como $\overline{e} \in Reach(T_1 \oplus T_2)$ existe $L \in \mathcal{C}$ tal que $L_1 \sim_1 L \sim_2 L_2$. Entonces, $h([L]_\sim) = h_1([L]_{\sim_1}) \oplus h_2([L]_{\sim_2}) = h_1([L_1]_{\sim_1}) \oplus h_2([L_2]_{\sim_2}) = \overline{e}^1 \oplus \overline{e}^2 = \overline{e}$.

- h es un isomorfismo de sistemas de transiciones:

  1. $h([e_0]_\sim) = h_1([e_0]_{\sim_1}) \oplus h_2([e_0]_{\sim_2}) = \overline{e_0}^1 \oplus \overline{e_0}^2$, ya que h_1 y h_2 son isomorfismos.
  2. $[L]_\sim \overset{a}{\mapsto}_\sim [L']_\sim \Leftrightarrow [L]_{\sim_i} \overset{a}{\mapsto}_{\sim_i} [L']_{\sim_i} \Leftrightarrow h_i([L]_{\sim_i}) \overset{a}{\mapsto}_i h_i([L']_{\sim_i}) \Leftrightarrow h([L]_\sim) \overset{a}{\mapsto} h([L']_\sim)$.

- h preserva los estados: $st(h([L]_\sim)) = st(h_1([L]_{\sim_1}) \oplus h_2([L]_{\sim_2})) = O(h_1([L]_{\sim_1})) = st(h_1([L]_{\sim_1})) = st([L]_{\sim_1}) = last(L) = st([L]_\sim)$.

□

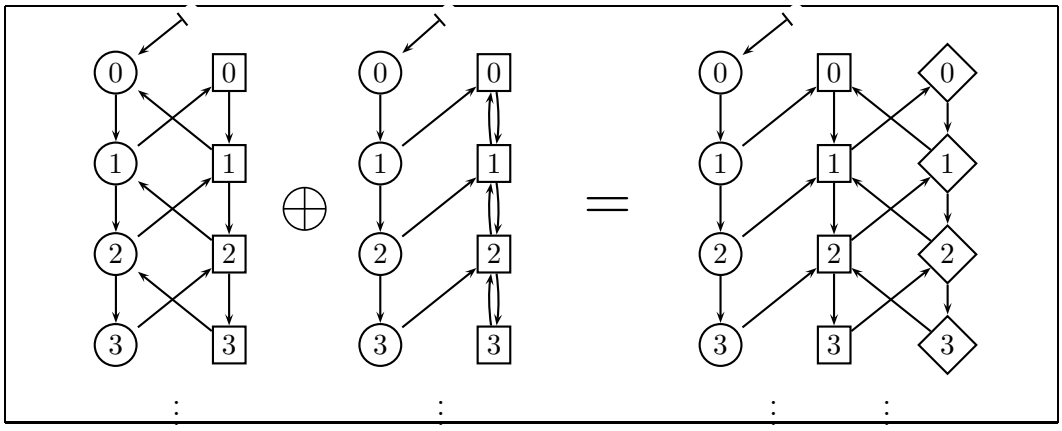

Figura 8.15: Mezcla de sistemas etiquetados

Ejemplo 10 Recordemos el ejemplo 7 que vimos en la sección 8.2. Como todos los símbolos de la sintaxis están etiquetados, todas las etiquetas en el sistema mezcla son pares (Γ, Δ) con $\Gamma, \Delta \in \{\Box, \circ\}$. Para simplificar la notación y mantener los símbolos usados en las figuras, tomamos $\circ = (\circ, \circ)$, $\Box = (\Box, \Box)$ y $\Diamond = (\circ, \Box)$ (no necesitaremos la etiqueta $(\Box, \circ)$ ya que los términos etiquetados con ella no son alcanzables). La mezcla de las dos propiedades, como muestra la figura 8.15, nos dice si algún agente ha muerto y, en caso afirmativo, si tal cosa ha ocurrido un número par de veces. El teorema 8.5.4 nos dice que el nuevo sistema de transiciones captura la conjunción de las dos propiedades, es decir, los términos etiquetados con $\circ$ identifican aquéllos cómputos en los que ningún agente ha muerto; $\Box$ identifica aquéllos cómputos en los que un número impar de agentes han muerto; y $\Diamond$ identifica los cómputos en los que han muerto un número par, pero no nulo, de agentes. Sin embargo, al considerar la intersección de los dos sistemas de transiciones (es decir, la disyunción de ambas propiedades) nos vamos al ínfimo del retículo.

Hemos elaborado los sistemas etiquetados en el marco muy general de los sistemas de transiciones, de manera que puedan ser aplicados a una gran variedad de modelos y propiedades. En este capítulo hemos ejemplificado su utilidad en el caso del cálculo π distribuido, pero en los capítulos siguientes veremos que también resultan útiles en el caso en que nosotros estamos aquí especialmente interesados, esto es, sobre las redes de Petri.

Solamente hemos aplicado la estructura algebraica del conjunto de sistemas cociente para un caso muy particular, aquél en el que queremos estudiar varias propiedades simultáneamente. Sería interesante llevar a cabo un estudio más en profundidad de esa estructura, intentando explotarla para el estudio de la combinación de varias propiedades de una forma más general, de modo que se incluya, por ejemplo, la disyunción de propiedades, lo que a primera vista parece ser bastante más complicado.

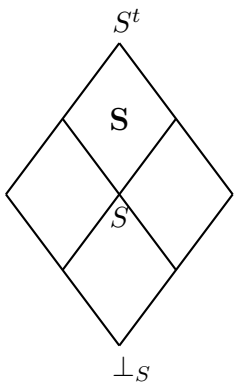

Figura 8.16: Retículo extendido de sistemas cociente

Además, hemos considerado equivalencias que recuerdan, al menos, el estado alcanzado, de manera que siempre se obtienen sistemas etiquetados. Sin embargo, podríamos relajar esta restricción para obtener un superconjunto de la clase de nuestros sistemas cociente (véase figura 8.16). En ese caso, el elemento minimal representaría al sistema que se olvida de absolutamente todo, incluso del estado alcanzado. Además, tendríamos sistemas de transiciones en el rombo inferior de la figura 8.16, que recuerdan sólo algunas propiedades del estado alcanzado, y sistemas de transiciones en los dos rombos de los lados, que recuerdan algunas cosas del pasado, pero sólo parte de la información del presente. Esto podría ser interesante cuando, por ejemplo, la sintaxis de un álgebra de procesos está definida como el álgebra de términos de una signatura, módulo una teoría ecuacional. En ese caso, no nos basta con identificar cómputos que alcanzan términos idénticos, sino que hemos de identificar también aquéllos que alcancen términos que sean equivalentes en esa teoría. Por supuesto, serían necesarias condiciones adicionales adecuadas que establecieran la corrección entre esa teoría y el sistema de transiciones.

Capítulo 9

Verificación de sistemas abiertos

En capítulos anteriores hemos estudiado el poder expresivo de las redes de Petri móviles, tanto en su versión con identificadores abstractos, como aquella que usa identificadores naturales, o la que incorpora replicación. En particular hemos probado la decidibilidad del recubrimiento en todos los casos (excepto para la versión que combina creación de nombres y replicación), lo que puede usarse para especificar propiedades de seguridad de nuestros sistemas. Sin embargo, estos resultados se obtienen considerando a los sistemas como sistemas cerrados, de manera que, en particular, si el sistema está esperando a sincronizar con el ambiente, entonces está bloqueado. Cuando se diseña un sistema como abierto, para que sea usado en un entorno arbitrario (como es claramente el caso de los sistemas ubicuos), es deseable contar con una semántica alternativa que tome en consideración la existencia de dicho entorno.

Existen modelos relacionados de naturaleza modular, como nuestros sistemas MSPN. Hay varios trabajos [BCEH05, Kin97, SVW01] para modelar el comportamiento de sistemas abiertos en el contexto de las redes de Petri, pero en ellos los autores centran su interés en la modularidad de los sistemas, buscando componentes abiertos reutilizables. En general, la semántica composicional garantiza que podemos reemplazar cualquier componente de un sistema por cualquier otra que tenga la misma semántica, sin alterar la semántica del sistema completo. Además, en estos modelos la noción de composición de componentes es completamente estática, en el sentido de que la manera en que las distintas componentes cooperan está fija, lo que implica una arquitectura estática, así como el hecho de que las interfaces que usan las componentes para componerse no pueden ser reutilizadas de manera dinámica por otros usuarios, como sería deseable en los sistemas ubicuos.

Nosotros seguimos una aproximación orientada a la seguridad, la de probar que cierta propiedad se cumple para un sistema, cualquiera que sea su entorno. De esta manera obtenemos propiedades de seguridad incluso en presencia de com-

ponentes malignos. Por lo tanto, aunque estamos considerando sistemas basados en componentes, y estamos definiendo una semántica abierta para considerar al entorno desconocido, las semánticas composicionales existentes para sistemas de redes basados en componentes no son adecuadas para nosotros. Esto es así porque al probar alguna propiedad de seguridad de un sistema tenemos que considerar de manera global todas las interacciones entre las componentes. Por ejemplo, mientras que no consideremos a un usuario como parte del sistema no podrá ser distinguido de un usuario malicioso, y por tanto no podemos hacer ninguna suposición acerca de su comportamiento esperado.

En este capítulo definimos la semántica mencionada, que llamamos *semántica abierta*, que define el comportamiento de un sistema como la unión de los comportamientos que este tendría en presencia de cualquier entorno posible. Esta definición es ciertamente inmanejable de manera directa, debido a su carácter infinitario, por lo que hemos desarrollado una semántica simbólica, que representa al entorno de manera abstracta, obteniendo así una representación finitaria. Hemos demostrado que la semántica simbólica es correcta y completa respecto a la semántica abierta.

9.1. Ejemplo: control de acceso a recursos

En esta sección presentamos una variación del ejemplo 2 visto en el capítulo 4, que describe la venta de billetes y el control de acceso posterior a un teatro. El sistema lo componen tres participantes: la ventanilla de billetes, el teatro y un agente que actualiza la base de datos del teatro con los billetes vendidos. La ventanilla y el agente aparecen en la figura 9.1, que representa a la localidad k, mientras que el teatro aparece en la figura 9.2, que representa la localidad llamada l. La ventanilla simplemente genera billetes nuevos y los vende, de modo que estén listos para que el agente actualize con ellos en el teatro. El agente (formado por dos componentes conexas, dentro de la línea de puntos) se mueve alternativamente entre las localidades donde se encuentran la ventanilla y el teatro, k y l, respectivamente. Cuando está en la misma localidad que la ventanilla, está dispuesto a recibir nuevos billetes válidos, y cuando se encuentra en la misma localidad que el teatro está dispuesto a comunicarle que dichos billetes son válidos. La ventanilla y el agente se comunican usando la clave privada compartida K_1, mientras que el agente y el teatro usan para comunicarse entre ellos la clave K_2. Esto se formaliza suponiendo que la variable c que etiqueta los arcos unidos a los lugares que las contienen es una variable de autenticación, es decir, $c \in Var_{Auth}$, de manera que cada vez que aparezca en una de las partes de un par de transiciones de sincronización compatibles, también ha de aparecer en la otra.

El teatro guarda los billetes vendidos en el lugar DB, que son justamente los que los clientes pueden exhibir para entrar al teatro. Pueden acceder al teatro por dos puertas: una puerta normal, `door1`, y otra entrada para personas que vayan acompañadas por un niño, `door2`. Cualquier persona que entre por esta

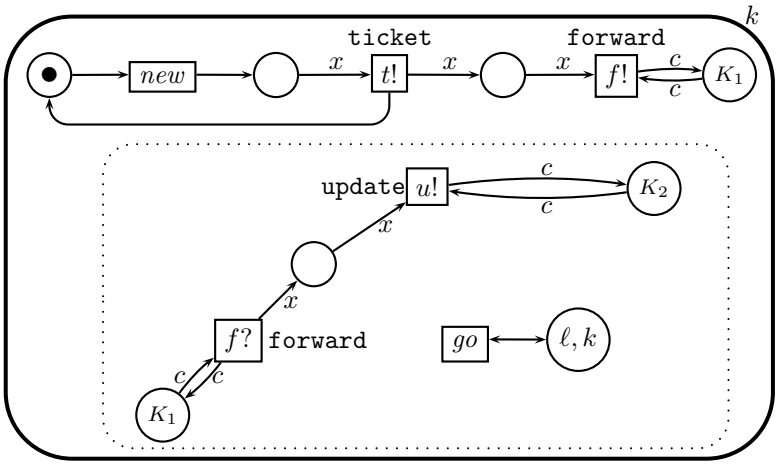

Figura 9.1: Ventanilla de billetes y agente

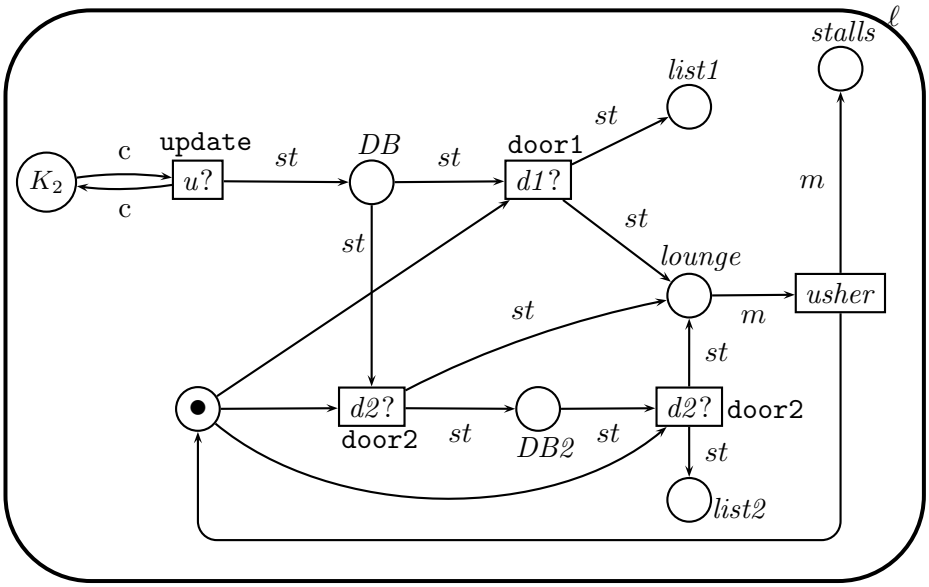

Figura 9.2: Teatro

segunda puerta puede usar un único asiento para él y el niño. Todo el mundo accede después de entrar al mismo espacio, donde les espera un ujier (o donde han de esperar la llegada de un ujier) que les acompañará hasta su asiento.

Queremos evitar la situación no deseada en la cual dos clientes que entraron por la puerta normal quieren sentarse en el mismo asiento, o más de dos clientes, contando a un niño si es que entraron por la otra puerta. Esta propiedad se puede expresar en términos de recubrimiento. Supongamos que un cliente tiene un billete η y accede por la puerta normal. En tal caso, el nombre η se anotará en la lista normal, *list1*, de manera que la propiedad anterior se viola para ese cliente siempre que sea alcanzable algún marcaje en el que η aparezca tanto en la lista como al menos dos veces en el patio de butacas, es decir, en el lugar *stalls*. Análogamente, cualquier cliente que entre por la otra puerta esperará que no se pueda superar

ningún marcaje en el que su billete aparece en *list2* y más de dos veces en *stalls*.

Otra propiedad que el personal del teatro puede estar interesado en comprobar es que nunca dos clientes que vayan a asientos distintos estén al mismo tiempo en la entrada, ya que en este ejemplo sencillo sólo se dispone de un ujier. En términos de recubrimiento, esto ocurriría si se puede cubrir el marcaje en el que tanto η como η' (identificadores distintos) aparecen al mismo tiempo en *lounge*. Téngase en cuenta que, con estos requerimientos, un marcaje con el mismo ticket dos veces en el vestíbulo sería legal, aunque esta situación no se puede llegar a dar en nuestro ejemplo.

Por supuesto, el sistema aislado es trivialmente seguro, ya que en ausencia de clientes no se puede dar ninguna de las situaciones descritas. De hecho, sin clientes el sistema está bloqueado después de que la ventanilla genera el primer billete, que no se llega a vender, excepto por el hecho de que el agente podría saltar alternativamente entre las localidades l y k. Por tanto, para estudiar la seguridad del sistema, deberíamos considerar a los clientes como parte del entorno, de manera que puedan ser desde padres honestos hasta reventas sin escrúpulos [Sch05]. Todo lo que debemos asumir de los clientes es que no conocen K_1 ni K_2, las claves secretas que las distintas componentes del sistema usan para comunicarse. Sin embargo, pueden (y deben) conocer las localidades donde se encuentran la ventanilla y el teatro.

9.2. Sistemas abiertos

Estamos interesados en verificar propiedades de nuestros sistemas cuando estos se encuentran en un entorno desconocido, no fiable y posiblemente malicioso. Hasta ahora, sólo hemos tratado con sistemas cerrados. Por tanto, necesitamos un modo de mezclar un sistema con otro, que puede ser considerado su entorno. En nuestra aproximación, mezclar es simplemente poner en paralelo. Dada la naturaleza modular de nuestros sistemas, la composición paralela de dos sistemas MSPN se define de manera inmediata.

Definición 9.2.1 *Sean $\mathcal{N}_1$ y $\mathcal{N}_2$ dos sistemas MSPN. Si $\mathcal{N}_1$ y $\mathcal{N}_2$ tienen conjuntos de lugares y transiciones disjuntos entonces su unión $\mathcal{N}_1 \cup \mathcal{N}_2$ también es un sistema MSPN, que llamaremos* composición paralela *de $\mathcal{N}_1$ y $\mathcal{N}_2$ y denotaremos por $\mathcal{N}_1 \mid \mathcal{N}_2$.*

Como consecuencia de la definición anterior, los marcajes de $\mathcal{N}_1$, $\mathcal{N}_2$ y $\mathcal{N}_1 \mid \mathcal{N}_2$ están relacionados de la siguiente manera: Si (M_i, loc_i) es un marcaje de $\mathcal{N}_i$ para $i \in \{1, 2\}$, entonces $(M_1 + M_2, loc_1 + loc_2)$ es un marcaje de $\mathcal{N}_1 \mid \mathcal{N}_2$. Recíprocamente, si (M, loc) es un marcaje de $\mathcal{N}_1 \mid \mathcal{N}_2$, entonces $(M|_{P_{\mathcal{N}_i}}, loc|_{\mathcal{N}_i})$ es un marcaje de $\mathcal{N}_i$, para $i \in \{1, 2\}$.

A continuación introducimos una notación auxiliar para denotar al conjunto de tokens localidad e identificadores que aparecen en una red marcada.

Definición 9.2.2 *Sea $\mathcal{N}$ un sistema MSPN y (M, loc) un marcaje de $\mathcal{N}$. Definimos $locs(\mathcal{N}(M, loc))$ como el conjunto de localidades que aparecen en (M, loc), es decir, $locs(\mathcal{N}(M, loc)) = \{k \in \mathcal{L} \mid \exists p \in P_{\mathcal{L}}, k \in M(p)\} \cup loc(\mathcal{N})$. Análogamente, definimos $Ids(\mathcal{N}(M, loc)) = \{a \in Id \mid \exists p \in P_{Id}, a \in M(p)\}$.*

La unión de estos dos conjuntos es lo que denotamos como $S(\mathcal{N}(M, loc))$, $S(\mathcal{N})$ o $S(M, loc)$, cuando no exista confusión. En la siguiente definición introducimos nuestra noción de entorno.

Definición 9.2.3 *Sea $\mathcal{N}$ un sistema MSPN marcado con (M_0, loc_0) y $\Phi \subseteq \mathcal{L} \cup Id$. Llamamos* $(\Phi, \mathcal{N})$-entorno *a cualquier sistema MSPN $\mathcal{P}$, inicialmente marcado con (M, loc), que sea disjunto con $\mathcal{N}$ y verifique $S(\mathcal{N}(M_0, loc_0)) \cap S(\mathcal{P}(M, loc)) \subseteq \Phi$. Denotamos por $Env(\Phi, \mathcal{N})$ al conjunto de $(\Phi, \mathcal{N})$-entornos.*

En breve, usaremos estos entornos para definir la semántica alternativa de sistemas MSPN que nos permitirá tratar con sistemas abiertos. Intuitivamente, las trazas abiertas de un sistema relativas a Φ son aquéllas trazas de la semántica original producidas por el sistema cuando funciona en paralelo con algún entorno cuyo conocimiento inicial acerca del sistema está contenido en Φ. Veamos primero algunas definiciones auxiliares.

Definición 9.2.4 *Sea tr una traza de un sistema MSPN $\mathcal{N} \mid \mathcal{P}$. Definimos la* proyección de tr sobre $\mathcal{N}$, *que denotamos por $tr|_{\mathcal{N}}$, por inducción sobre la longitud de tr, de la siguiente manera:*

- $(M, loc)|_{\mathcal{N}} = (M|_{\mathcal{N}}, loc|_{\mathcal{N}})$.
- $tr[u(\sigma)\rangle(M, loc)|_{\mathcal{N}} = tr|_{\mathcal{N}}$ *si* $u \in T_{\mathcal{P}} \cup T_{\mathcal{P}} \times T_{\mathcal{P}}$.
- $tr[u(\sigma)\rangle(M, loc)|_{\mathcal{N}} = tr|_{\mathcal{N}}[u(\sigma)|_{\mathcal{N}}\rangle(M|_{\mathcal{N}}, loc|_{\mathcal{N}})$ *si* $u \notin T_{\mathcal{P}} \cup T_{\mathcal{P}} \times T_{\mathcal{P}}$.
- $u(\sigma)|_{\mathcal{N}} = u(\sigma)$ *si* $u \in T_{\mathcal{N}} \cup T_{\mathcal{N}} \times T_{\mathcal{N}}$.
- $(t, t')(\sigma)|_{\mathcal{N}} = t(\sigma|_{\mathcal{N}})$ *si* $t \in T_{\mathcal{N}}$ *y* $t' \in T_{\mathcal{P}}$.
- $(t, t')(\sigma)|_{\mathcal{N}} = t'(\sigma|_{\mathcal{N}})$ *si* $t' \in T_{\mathcal{N}}$ *y* $t \in T_{\mathcal{P}}$.

Definición 9.2.5 *Sea $\mathcal{N}$ un sistema MSPN marcado y $\Phi \subseteq \mathcal{L} \cup Id$. Diremos que una secuencia*

$$tr = (M_0, loc_0)[u_1(\sigma_1)\rangle^{\Phi}(M_1, loc_1) \ldots (M_{n-1}, loc_{n-1})[u_n(\sigma_n)\rangle^{\Phi}(M_n, loc_n)$$

es una traza Φ-abierta *de $\mathcal{N}$ si existe un entorno $\mathcal{P} \in Env(\Phi, \mathcal{N})$ y $tr' \in [\![\mathcal{N} \mid \mathcal{P}]\!]$ tales que $tr = tr'|_{\mathcal{N}}$. Denotaremos mediante $[\![\mathcal{N}]\!]^{\Phi}$ al conjunto de trazas Φ-abiertas de $\mathcal{N}$.*

Comentemos algunos de los conceptos que aparecen en las definiciones anteriores. El conjunto $[\![N]\!]^{\Phi}$ contiene la proyección sobre $\mathcal{N}$ de todas las trazas que genera $\mathcal{N}$ en presencia de cualquier entorno, siempre y cuando el conocimiento inicial que este entorno tiene sobre el sistema sea un subconjunto de Φ. Este hecho está formalizado por la condición $S(\mathcal{P}) \cap S(\mathcal{N}) \subseteq \Phi$ impuesta sobre los $(\Phi, \mathcal{N})$-entornos, de modo que cualquier token identificador o localidad que aparezca en $\mathcal{P}$ que también aparezca en $\mathcal{N}$ ha de ser uno de los que se encuentran en Φ.

Tal y como la hemos definido, la composición $\mathcal{N} \mid \mathcal{P}$ es un sistema MSPN, con su semántica de trazas. Sin embargo, no estamos interesados en el comportamiento de los entornos, sino sólo en el comportamiento de $\mathcal{N}$. Por ello, tomamos las proyecciones de dichas trazas sobre $\mathcal{N}$, es decir, eliminamos las transiciones autónomas disparadas por el entorno y cualquier información de $\mathcal{P}$ que aparezca en los marcajes y en las transiciones que sincronizan con el entorno. Por esto último, es posible encontrar pasos como $(M, loc)[t(\sigma)\rangle(M', loc')$ con $\lambda(t) \in Sync$ formando parte de una traza abierta, lo que quiere decir que $\mathcal{N}$ ha sincronizado con su entorno.

Definición 9.2.6 *Sea $\mathcal{N}$ un sistema MSPN marcado. Diremos que $\mathcal{P} \in Env(\Phi, \mathcal{N})$ es* no olvidadizo*, relativo a una traza Φ-abierta tr, si:*

- $S(\mathcal{P}) = \Phi$,
- $pre(t) \subseteq post(t)$ *para toda transición* $t \in T_{\mathcal{P}}$, *y*
- $pre(t) \subseteq post(t')$ *para cualquier paso en tr etiquetado por (t, t') o (t', t), con $t \in T_{\mathcal{N}}$ y $t' \in T_{\mathcal{P}}$.*

Un entorno es no olvidadizo cuando su marcaje inicial contiene toda la información del sistema que puede tener, nunca pierde dicha información y aprovecha todas las comunicaciones con el sistema para recibir toda la información que este le proporciona.

Hemos definido los entornos no olvidadizos porque cumplen la siguiente propiedad: contienen en cada paso del cómputo todos los tokens coloreados del sistema que pueden tener, a saber los que podían conocer en el estado inicial y aquéllos que el sistema les proporciona según la traza considerada. Esta propiedad será de gran importancia en secciones sucesivas.

Proposición 9.2.7 *Sea $\mathcal{N}$ un sistema MSPN marcado, $\Phi \subseteq \mathcal{L} \cup Id$,*

$$tr = \mathcal{M}_0[u_1(\sigma_1)\rangle^{\Phi}\mathcal{M}_1 \ldots \mathcal{M}_{m-1}[u_m(\sigma_m)\rangle^{\Phi}\mathcal{M}_m$$

una traza Φ-abierta generada por un entorno $\mathcal{P} \in Env(\Phi, \mathcal{N})$ relativo a tr y $tr' \in [\![\mathcal{N} \mid \mathcal{P}]\!]$ tal que $\mathcal{M}_m = (\mathcal{M} + \mathcal{M}_{\mathcal{P}})|_{\mathcal{N}}$ donde $\mathcal{M}_{\mathcal{P}}$ es un marcaje de $\mathcal{P}$. Entonces, si $\mathcal{P}$ es no olvidadizo se cumple que $\Phi \cup \bigcup_{\substack{i=1 \\ \lambda(u_i) \in Sync}}^{m} \sigma_i(Var(u_i)) \subseteq S(\mathcal{M}_{\mathcal{P}})$.

Demostración. Por la segunda condición de entorno no olvidadizo se tiene que el soporte del marcaje del entorno sólo puede crecer a lo largo del cómputo. Por la primera condición el marcaje inicial del entorno contiene a todo Φ y por la observación anterior también lo contiene el marcaje $\mathcal{M}_{\mathcal{P}}$. Por último, la última condición de entornos no olvidadizos nos dice que si se dispara una transición $(t, t')(\sigma)$ donde t es del sistema y t' del entorno (o viceversa), pasando el entorno de $\mathcal{M}^1_{\mathcal{P}}$ a $\mathcal{M}^2_{\mathcal{P}}$ entonces $S(\mathcal{M}_{\mathcal{P}}) \supseteq S(\mathcal{M}^2_{\mathcal{P}}) = S(\mathcal{M}^1_{\mathcal{P}}) \cup \sigma(post(t')) \supseteq S(\mathcal{M}^1_{\mathcal{P}}) \cup \sigma(pre(t))$. Además, por definición de compatibilidad de transiciones se cumple que si $x \in post(t) \setminus pre(t)$ entonces $x \in pre(t')$, y por tanto $\sigma(Var(t)) \subseteq S(\mathcal{M}^2_{\mathcal{P}}) \subseteq S(\mathcal{M}_{\mathcal{P}})$.

□

A continuación veremos que en la definición 9.2.5 podemos imponer la condición de que el entorno $\mathcal{P}$ sea no olvidadizo, sin por ello cambiar el conjunto de trazas Φ-abiertas definidas, de modo que siempre que lo necesitemos podremos suponer que los entornos son no olvidadizos.

Proposición 9.2.8 *Si tr es una traza Φ-abierta entonces existen $\mathcal{P} \in Env(\Phi, \mathcal{N})$ no olvidadizo y $tr' \in [\![\mathcal{P} \mid \mathcal{N}]\!]$ tal que $tr = tr'|_{\mathcal{N}}$.*

Demostración. Por definición de traza Φ-abierta existen $\mathcal{P}' \in Env(\Phi, \mathcal{N})$ y $tr' \in [\![\mathcal{P}' \mid \mathcal{N}]\!]$ tales que $tr = tr'|_{\mathcal{N}}$. Modificamos $\mathcal{P}'$, obteniendo $\mathcal{P}$ no olvidadizo añadiendo un lugar c marcado inicialmente con Φ, con lo que automáticamente se cumple el primer punto de la condición de no olvidadizos. Además, para todo $t \in T_{\mathcal{P}}$ añadimos arcos $F(t, c) = x$ para todo $x \in pre(t) \setminus post(t)$, con lo que tenemos también la segunda condición, y para todo paso (t, t') o (t', t) en tr' con t transición de $\mathcal{N}$ y t' de $\mathcal{P}$ añadimos $F(t', c) = x$ para todo $x \in pre(t) \setminus post(t')$, con lo que concluimos que $\mathcal{P}$ es no olvidadizo. Además, los marcajes de $\mathcal{P}$ contienen a los de $\mathcal{P}'$ por lo que, por monotonía, en $\mathcal{P}$ se pueden disparar las mismas transiciones que en $\mathcal{P}'$ y la tesis se sigue.

□

Aunque el sistema del ejemplo en la sección 9.1 está prácticamente bloqueado según la semántica original, esto no es así si consideramos su semántica abierta. Como dijimos antes, podemos asumir que todo lo que el entorno puede conocer del sistema en su estado inicial es k y l, las localidades donde se encuentran las componentes. Por tanto, estamos interesados en estudiar su comportamiento como sistema $\{k, l\}$-abierto, es decir, sus trazas $\{k, l\}$-abiertas. Por ejemplo, podríamos considerar al cliente de la figura 9.3, que es un $(\{k, l\}, \mathcal{N})$-entorno. Es preciso tener en cuenta que aunque este cliente se encuentra inicialmente en una localidad k', distinta de las localidades k y l, esto no contradice la condición sobre los $(\{k, l\}, \mathcal{N})$-entornos, simplemente porque $\mathcal{N}$ no conoce esta localidad.

Una vez que hemos introducido esta nueva semántica, podemos usarla como base para el estudio del comportamiento de nuestros sistemas en el nuevo marco. Por ejemplo, podemos comenzar definiendo las nociones análogas de alcanzabilidad y recubrimiento.

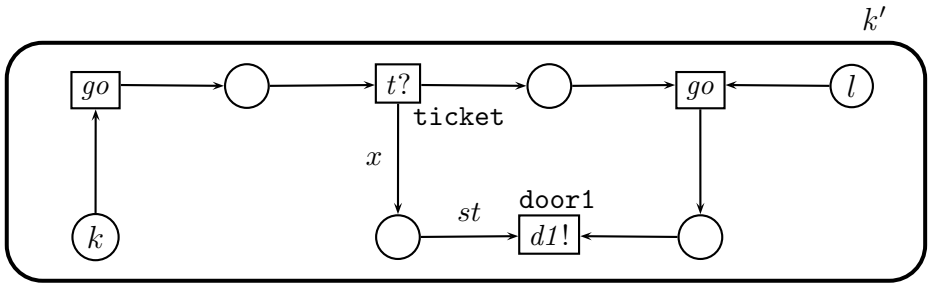

Figura 9.3: Posible $(\{k, l\}, \mathcal{N})$-entorno para el ejemplo de la sección 9.1

Definición 9.2.9 *El problema de la* Φ-alcanzabilidad *consiste en decidir, dados dos marcajes de un sistema MSPN* $\mathcal{N}$*,* (M_0, loc_0) *y* (M, loc)*, si* (M, loc) *es alcanzable desde* (M_0, loc_0) *por medio de una traza en* $[\![\mathcal{N}(M_0, loc_0)]\!]^{\Phi}$*. Análogamente, el problema del* Φ-recubrimiento *consiste en decidir, dado* (M_0, loc_0) *y* (M, loc)*, si existe un marcaje alcanzable desde* (M_0, loc_0) *por medio de una traza* $[\![\mathcal{N}(M_0, loc_0)]\!]^{\Phi}$ *que cubre a* (M, loc)*.*

Aplicando estas nociones extendidas podemos estudiar propiedades interesantes de los sistemas abiertos. En particular, las propiedades de seguridad discutidas en el ejemplo de la sección 9.1 se pueden especificar por medio de problemas de $\{k, l\}$-recubrimiento.

9.3. Una clasificación informal de algunas propiedades de seguridad

La semántica abierta definida en la sección anterior divide al mundo computacional en dos partes: por un lado está el sistema, constituido por una colección conocida de componentes MSPN; por otro lado está el entorno del sistema, del cual lo único que conocemos es el conjunto de nombres del sistema que puede usar.

Esta separación entre sistema y resto del mundo plantea la pregunta de qué componentes deberían ser consideradas como parte del sistema en cuestión, a la hora de especificar y verificar propiedades concretas de seguridad. La respuesta establece una clasificación informal de las propiedades de seguridad, en base a qué componentes se consideran parte del sistema y cuáles parte del entorno. Consideremos el caso en el que un proveedor de servicios ofrece un servicio a clientes. El tipo de propiedades de seguridad que podemos estudiar depende de qué clientes consideramos como parte del sistema.

- *Ningún cliente:* suponemos que el sistema está compuesto únicamente por el proveedor del servicio, de manera que los potenciales usuarios son desconocidos y, por tanto, atacantes potenciales. En tal caso, sólo podemos hacer referencia a propiedades relativas al proveedor del servicio, como pueden ser la disponibilidad o la confidencialidad de sus claves secretas, pero, obviamente, ésta sólo se puede garantizar si estamos seguros de que esas claves

nunca se transmiten a ningún usuario, ya que no hay manera de distinguir entre usuarios honestos y deshonestos.

- *Un cliente honesto:* en este caso el sistema lo constituyen el proveedor de servicios y un cliente honesto. Cualquier propiedad acerca del proveedor de servicios aún se cumple, ya que el cliente considerado como parte del sistema era antes considerado como parte del entorno. Sin embargo, ahora podemos estudiar además propiedades concretas de ese cliente en particular. Por ejemplo, podríamos probar que siempre recibe el servicio de manera satisfactoria, una vez autenticado adecuadamente por el proveedor.

- *Todos los clientes honestos:* finalmente, podemos considerar como parte del sistema a todos los posibles clientes honestos (o a un conjunto finito de ellos). De nuevo, las propiedades que probábamos antes se preservan. Además, ahora podemos hablar de propiedades globales, que afectan a todos los clientes, como por ejemplo el orden en el que reciben los servicios.

Podríamos pensar que también sería interesante considerar como parte del entorno a clientes deshonestos, pero esto de hecho no es necesario, ya que cualquier cliente de ese tipo puede aparecer como parte del entorno arbitrario.

La clasificación anterior establece un ten con ten entre la *calidad* de las propiedades capturadas y su generalidad. Es decir, cuantas más componentes consideremos como parte del sistema

- *menos generales* son las propiedades, en el sentido de que se referirán a todas las posibles componentes del sistema (respecto de cualquier entorno maligno);

- *más precisas* son esas propiedades, ya que pueden hacer referencia al comportamiento específico de esas componentes, cosa que no se podría hacer si se considerasen parte del entorno.

Merece la pena tener en cuenta que nuestra aproximación es similar a la seguida en [FG01, FGM04] para las álgebras de procesos *Security Process Algebra* (SPA) y *CryptoSPA*, una versión criptográfica de SPA. En estos trabajos, al igual que en nuestro caso, el sistema observado (en el sentido de la bisimulación) es atacado explícitamente por todos los posibles atacantes, al contrario que en [AG97], donde el observador (en el sentido de testeo may) juega también el papel de atacante.

Sin embargo, nuestra clasificación de propiedades de seguridad se establece en términos del conjunto de componentes que tenemos que considerar para estudiar cada propiedad, en contraposición a [FG01, FGM04], donde ésta se establece en términos de lo restrictivo de los requerimientos que aparecen en las definiciones correspondientes.

Veamos en qué se traducen los comentarios anteriores en el caso del ejemplo de la sección 9.1 de este capítulo. Consideremos primero que el sistema sólo

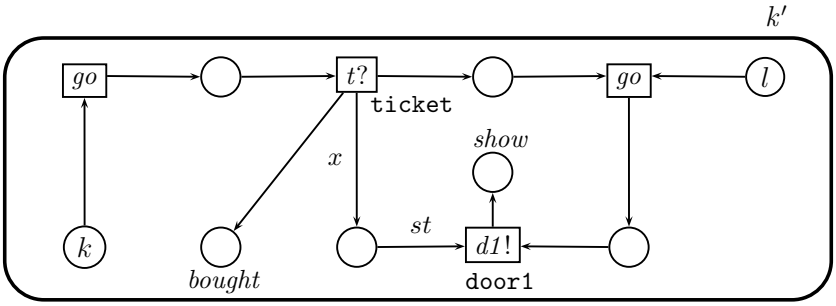

Figura 9.4: Posible cliente

está compuesto por el proveedor de servicios, o sea, por las componentes descritas anteriormente. En este caso, no podemos suponer nada acerca de los clientes (honestos o deshonestos) y, por tanto, las propiedades que pueden ser capturadas hablan únicamente del proveedor de servicios, aunque en presencia de cualquier usuario. Por ejemplo, se podría comprobar que cualquiera de los tickets que los usuarios enseñan para acceder al teatro es de hecho uno de los tickets vendidos en ventanilla (integridad). Esta propiedad se puede especificar en términos de recubrimiento.

Sin embargo, es posible que clientes que hayan comprado un ticket copien su entrada y se la vendan fraudulentamente a otros clientes. En este caso el teatro aún puede evitar la desagradable situación en la que dos clientes quieren sentarse en el mismo asiento, si entraron por la puerta normal, o más de dos si es que entraron por la otra puerta. Esto se consigue permitiendo la entrada solamente al primer cliente que exhiba una de las copias del ticket [Sch05]. El hecho de que el sistema implemente de verdad esta restricción (control de acceso) se puede enunciar como una instancia del problema de α-recubrimiento.

Añadamos ahora al sistema un cliente honesto, como el que aparece en la figura 9.4. Tras ello podríamos estudiar propiedades que afectan a este cliente en particular, ya que podemos usar la información que nos da su comportamiento concreto. Por ejemplo, podríamos estudiar si siempre le es posible ver el espectáculo después de pasar por ventanilla (disponibilidad), propiedad que se puede formalizar como una propiedad *home space* [Joh88]. De nuevo, queremos garantizar que esa propiedad se cumple en presencia de cualquier otro cliente, incluso de reventas sin escrúpulos.

Finalmente, podemos considerar como parte del sistema varios clientes honestos. Por ejemplo, podríamos tener en el sistema grupos de clientes, de manera que exactamente un miembro de cada grupo estuviese a cargo de comprar los tickets, para después ofrecérselos al resto de los miembros del grupo. En este supuesto podríamos especificar por medio de una propiedad de recubrimiento (más exactamente, como combinación de problemas de recubrimiento) que a lo sumo un miembro de cada grupo puede entrar en el teatro o, en términos de *home space*, que al menos un cliente de cada grupo puede entrar en el teatro.

9.4. Semántica simbólica de sistemas MSPN

Para conseguir que la semántica abierta de sistemas MSPN que hemos definido en el apartado anterior sea manejable, necesitamos eliminar de la definición de trazas abiertas el cuantificador existencial que cuantifica a los entornos. Para ello, vamos a desarrollar a continuación una semántica simbólica de los sistemas MSPN, que tiene en cuenta de manera abstracta cualquier entorno posible. La idea principal para desarrollar dicha semántica es la de permitir que se disparen las transiciones de sincronización, incluso cuando no está disponible la parte que ha de sincronizar con el sistema, y siempre que dicha sincronización sea legal según Φ (lo que intuitivamente querrá decir que podría ser parte del entorno). Por tanto, Φ jugará el papel del conocimiento acerca del sistema que ha adquirido el entorno a lo largo de su historia. Como este conocimiento puede cambiar debido a la interacción entre ambos, tenemos que considerarlo como parte de los marcajes. Por tanto, añadimos a los marcajes de un sistema MSPN una tercera componente, de modo que si $\mathcal{M} = (M, loc)$ escribiremos $\mathcal{M}^{\Phi}$ para referirnos a (M, loc, Φ). Llamaremos *marcajes simbólicos* a esas ternas.

Mediante la expresión "conocimiento acerca del sistema" nos referimos a la parte del conjunto de tokens que posee el sistema y que el entorno conoce. Este conocimiento puede venir de dos fuentes: el sistema puede comunicarle al entorno tokens que el entorno aún no conocía; dualmente, el entorno puede darle al sistema tokens que este no conocía previamente. Estos últimos tokens pasan a formar parte del sistema y, obviamente, son conocidos por el entorno.

Para diferenciar entre ambas fuentes de conocimiento dividimos cada Φ en Φ^{out} y Φ^{in}, de manera que $\Phi = (\Phi^{out}, \Phi^{in})$. En lo sucesivo escribiremos Φ^{io} para denotar a la unión (disjunta) de Φ^{out} y Φ^{in}.

La definición de la semántica simbólica se basa en la noción de disparo simbólico de una transición de un sistema MSPN. Hasta ahora no necesitábamos modos para las transiciones de sincronización, ya que éstas nunca se disparaban de manera aislada, sino en pares. Ahora sí necesitaremos modos de transiciones de sincronización, que se definen como cabe esperar.

Definición 9.4.1 *Sea $\mathcal{S}$ un sistema MSPN y $t \in T$ tal que $\lambda(t) \in Sync$. Un* modo simbólico *de t es una función $\sigma : Var(t) \to Tokens$ tal que $\sigma(x) \in \mathcal{T} \Leftrightarrow x \in Var_{\mathcal{T}}$ para $\mathcal{T} \in \{\bullet, \mathcal{L}, Id\}$.*

En este punto es preciso tener en cuenta que las transiciones de sincronización no estaban sintácticamente restringidas como lo estaban las transiciones autónomas, en cuanto al conjunto de variables en sus arcos adyacentes. Por tanto, podía darse el caso de que una variable x estuviese en un arco saliente de la transición, pero no estuviese en ningún arco entrante, es decir, $x \in post(t) \setminus pre(t)$. En este caso, los modos simbólicos de una transición pueden asignarle a esa variable cualquier valor del color correspondiente. A continuación definimos los disparos simbólicos de transiciones.

Definición 9.4.2 *Sea* $\mathcal{S} = (\mathcal{N}, \mathcal{M}_0)$ *un sistema MSPN,* $\Phi = (\Phi^{in}, \Phi^{out}) \subseteq C\,Tokens \times C\,Tokens$ *con* $\Phi^{in} \cap \Phi^{out} = \emptyset$*. Consideremos también la componente* $N \in \mathcal{N}$ *dada por* $N = (P_N, T_N, F_N, \lambda_N, C_N)$*. Definimos* $[u(\sigma)\rangle^{\#}$ *como la menor relación entre marcajes simbólicos tal que:*

1. *Si* $\mathcal{M}_1[t(\sigma)\rangle\mathcal{M}_2$ *con* $\lambda(t) \in \mathcal{A} \setminus \{new\}$*, entonces* $\mathcal{M}_1^{\Phi}[t(\sigma)\rangle^{\#}\mathcal{M}_2^{\Phi}$*.*
2. *Si* $\mathcal{M}_1[t(\sigma)\rangle\mathcal{M}_2$ *con* $\lambda(t) = new$ *y* $\sigma(\nu) \notin \Phi^{io}$*, entonces* $\mathcal{M}_1^{\Phi}[t(\sigma)\rangle^{\#}\mathcal{M}_2^{\Phi}$*.*
3. *Si* $\mathcal{M}_1[(t,t')(\sigma)\rangle\mathcal{M}_2$ *con* $\lambda(t) \in Sync$*, entonces* $\mathcal{M}_1^{\Phi}[(t,t')(\sigma)\rangle^{\#}\mathcal{M}_2^{\Phi}$*.*
4. *Sea* $t \in T_N$*,* $\lambda(t) \in Sync$*,* σ *un modo simbólico para* t *y* $\mathcal{M}_1$ *un marcaje de* $\mathcal{S}$ *tal que* $\sigma(F(p,t)) \in M_1(p)$*, para todo* $p \in P_N$*. Si* $loc_1(N) \in \Phi^{io}$*,* $\sigma(x) \in \Phi^{io}$ *para todo* $x \in pre(t) \cap Var_{Aut}$ *y* $\sigma(x) \notin S(\mathcal{M}_1) \setminus \Phi^{io}$*, para todo* $x \in post(t) \setminus pre(t)$*, entonces tomando*
   - $M_2(p) = M_1(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$ *para cada* $p \in P$*,*
   - $\Phi_2 = \big(\Phi^{in} \cup (\sigma(post(t)) \setminus \Phi^{io}), \Phi^{out} \cup (\sigma(pre(t)) \setminus \Phi^{io})\big)$

   y $\mathcal{M}_2 = (M_2, loc_1)$*, tenemos que* $\mathcal{M}_1^{\Phi}[t(\sigma)\rangle^{\#}\mathcal{M}_2^{\Phi_2}$ *es un disparo simbólico de la transición* t *con modo* σ*.*

Diremos que un marcaje simbólico $\mathcal{M}^{\Phi}$ *es simbólicamente* Φ_0*-alcanzable si existe una sucesión de disparos simbólicos, que llamaremos* traza simbólica*, que acaba en* $\mathcal{M}^{\Phi}$*:*

$$\mathcal{M}_0^{\Phi_0}[u_1(\sigma_1)\rangle^{\#}\mathcal{M}_1^{\Phi_1} \cdots \mathcal{M}_{n-1}^{\Phi_{n-1}}[u_n(\sigma_n)\rangle^{\#}\mathcal{M}^{\Phi}$$

Vamos a comentar la definición anterior. Siempre que un sistema MSPN realiza un paso (interno) se realiza también un paso simbólico, sin cambiar el conocimiento del entorno, como se afirma en los puntos 1, 2 y 3. Además, en el punto 2, se dice que siempre que se crea un nuevo nombre tiene que tratarse de uno que no se encuentre ni en el marcaje ni en Φ^{io}. Esta condición no impone ninguna restricción en el conjunto de posibles disparos, pero sí en el conjunto de nombres producidos: cualquier nombre que no se encuentre en Φ^{io} puede ser elegido como nuevo nombre.

En el último punto reproducimos a la transición de sincronización necesaria para realizar el disparo, incluso cuando sólo una de las transiciones de sincronización está activada. El entorno, que intuitivamente es el encargado de disparar la transición compatible, debe cumplir una serie de condiciones para poder sincronizar. En primer lugar, ha de conocer el nombre de la localidad donde se produce la sincronización, es decir, $loc_1(N) \in \Phi^{io}$. En segundo lugar, si existe algún arco de autenticación, etiquetado por una variable x, entonces la transición compatible del entorno ha de conocer su valor, es decir, $\sigma(x) \in \Phi^{io}$. Por último, el entorno no le puede ofrecer al sistema un identificador o localidad que no puede conocer, de modo que ningún arco que vaya a una postcondición puede estar etiquetado por una variable x instanciada a un valor conocido por el sistema pero no por el

entorno, es decir, $\sigma(x) \notin S(\mathcal{M}) \setminus \Phi^{io}$. El marcaje obtenido por el disparo simbólico es el resultado de disparar t sin su transición compatible. Además, el conocimiento del entorno se ve incrementado por los valores de los tokens que se toman de las precondiciones, es decir, aquéllos que el sistema ofrece al entorno, y por los valores en los arcos que van a postcondiciones, aquéllos que el entorno ofrece al sistema y no eran parte previamente de ese conocimiento (de manera que sigue estando formado por dos componentes disjuntas).

A continuación vamos a estudiar la relación existente entre ambas semánticas. En la sección 9.2 usábamos conjuntos $\Phi \subseteq CTokens$ para representar el conocimiento del entorno. Sin embargo, en la definición de la semántica simbólica de la sección anterior hemos dividido ese conocimiento en dos conjuntos, Φ^{in} y Φ^{out}, para distinguir las dos maneras en las que el entorno puede aprender nombres del sistema. Si queremos movernos del marco simbólico al marco abierto, basta olvidarse de esta distinción, considerando la unión de ambos conjuntos. Recíprocamente, cuando tenemos $\Phi_0 \subseteq CTokens$ y queremos movernos al marco simbólico, tenemos que elegir una manera de dividir ese conocimiento. Denotaremos mediante $split(\Phi)$ al conjunto de formas en que se puede dividir Φ.

Ciertamente, podríamos haber hecho una presentación uniforme de ambas semánticas. Para ello bastaría con introducir también la distinción entre Φ^{in} y Φ^{out} en la definición de semántica abierta, o eliminar esa distinción en la definición de semántica simbólica. Sin embargo, no hemos hecho lo primero porque establecer esa distinción en aquel momento hubiese sido un tanto ad-hoc, ya que en el estado inicial no existe la intuición de "cómo ha aprendido el entorno un nombre". Tampoco hemos hecho lo segundo pues deseábamos mantener esa distinción una vez que tiene sentido hacerlo, ya que está claro que los nombres en cada conjunto juegan un papel muy distinto. Por ejemplo, podemos especificar propiedades de confidencialidad usando los nombres en Φ^{out}, lo que no sería posible si no contásemos con la división en el origen de los nombres.

A partir de ahora denotaremos mediante $\mathcal{S}|_{\mathcal{T}}$ al subsistema de $\mathcal{S}$ compuesto por sus subredes $\mathcal{S}$ cuyos lugares son todos de tipo $\mathcal{T}$.

Proposición 9.4.3 *Dado un sistema MSPN, si $\mathcal{M}^{\Phi}$ es simbólicamente Φ_0-alcanzable entonces $\mathcal{M}$ es Φ_0^{io}-alcanzable.*

Demostración. Basta probar que si

$$(M_0, loc_0, \Phi_0)[u_1(\sigma_1)\rangle^{\#} \ldots [u_n(\sigma_n)\rangle^{\#}(M_n, loc_n, \Phi_n)$$

es una traza simbólica de un sistema MSPN $\mathcal{S} = (\mathcal{N}, (M_0, loc_0))$, entonces $(M_0, loc_0)[u_1(\sigma_1)\rangle \ldots [u_n(\sigma_n)\rangle(M_n, loc_n)$ es una traza Φ_0^{io}-abierta de $\mathcal{S}$. Para ello, tenemos que ver que existen $\mathcal{P} \in Env(\Phi_0^{io}, \mathcal{S})$ y $tr \in [\![\mathcal{S} \mid \mathcal{P}]\!]$, tales que $(M_0, loc_0)[u_1(\sigma_1)\rangle \ldots [u_n(\sigma_n)\rangle(M_n, loc_n)$ es la proyección de tr sobre $\mathcal{N}$. Por simplicidad en las notaciones, vamos a suponer que para cada i perteneciente al conjunto $\{1, \ldots, n\}$ se tienen $u_i = t_i \in T, \lambda(t_i) = s_i?$, es decir, que la traza está compuesta solamente de sincronizaciones de $\mathcal{S}$

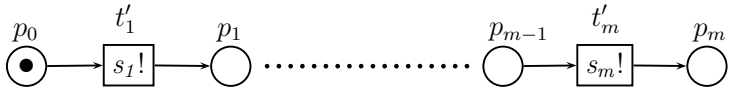

Figura 9.5: Restricción de $\mathcal{P}$ a lugares de $P_\bullet$

con su entorno, y que todas las sincronizaciones se producen en la misma localidad. Si esto no fuese así, bastaría distinguir entre sincronizaciones con el entorno y transiciones autónomas del sistema, que se corresponderían directamente con transiciones de la traza abierta. Además, habría que añadirle al entorno que vamos a construir transiciones de movimiento que lo coloquen en la localidad donde va a tener lugar la sincronización.

A lo largo del resto de la demostración vamos a omitir de los marcajes la componente localidad ya que, debido a las suposiciones del párrafo anterior, las localidades no juegan ningún papel. Más concretamente, vamos a probar que existe un sistema MSPN compuesto por una sola red, que llamaremos simplemente $\mathcal{P}$, marcada inicialmente con M'_0, tal que:

- $S(M'_0) \cap S(M_0) \subseteq \Phi_0^{io}$,
- $\mathcal{P}|_\bullet$ es la red de la figura 9.5 con $m = n$,
- $(M_0 + M'_0)[(t_1, t'_1)(\sigma_1)\rangle \cdots [(t_n, t'_n)(\sigma_n)\rangle(M_n + M'_n)$, donde M'_i es un marcaje de $\mathcal{P}$ y t'_i es una transición de $\mathcal{P}$, para todo $i \in \{1, \ldots, n\}$,
- $\Phi_n^{io} \subseteq S(M'_n)$.

Si probamos lo anterior, basta proyectar la traza sobre $\mathcal{N}$ para obtener el resultado deseado. Lo probamos por inducción sobre n, la longitud de la traza.

Si $n = 0$ tenemos que probar que existe un $\mathcal{P}$ inicialmente marcado con M'_0 tal que $S(M'_0) \cap S(M_0) \subseteq \Phi_0^{io} \subseteq S(M'_0)$ y $\mathcal{P}|_\bullet$ consiste en un solo lugar aislado p_0. Tomamos $\mathcal{P} = (P, \emptyset, \emptyset, \emptyset, C)$ con $P = \{p_0\} \cup \{p(a) \mid a \in \Phi_0^{io}\}$, $C(p_0) = \bullet$, $C(p(a)) = C(a)$, $M'_0(p_0) = 1$ y $M'_0(p(a)) = \{a\}$. Claramente, $\mathcal{P}$ verifica las condiciones anteriores.

Supongamos ahora que $n > 0$ y $(M_0, \Phi_0)[u_1(\sigma_1)\rangle^\# \ldots [u_n(\sigma_n)\rangle^\#(M_n, \Phi_n)$. Por la hipótesis de inducción existe un $\mathcal{P}'$ con $S(\mathcal{P}') \cap S(M_0) \subseteq \Phi_0^{io}$, $\Phi_{n-1}^{io} \subseteq S(M'_{n-1})$, $(M_0 + M'_0)[u_1(\sigma_1)\rangle^\# \ldots [u_{n-1}(\sigma_{n-1})\rangle^\#(M_{n-1} + M'_{n-1})$ y tal que $\mathcal{P}'|_\bullet$ es como la red de la figura 9.5, con $m = n - 1$. Entonces definimos el sistema $\mathcal{P}$ que verifica la tesis de la siguiente manera:

1. Le añadimos a $\mathcal{P}'$ la transición t'_n con $\lambda(t'_n) = \overline{\lambda(t_n)}$ (es decir, $\lambda(t'_n) = s?$ si $\lambda(t_n) = s!$, y viceversa), un nuevo lugar p_n y dos arcos (p_{n-1}, t'_n) y (t'_n, p_n) (etiquetados por ε), de manera que $\mathcal{P}|_\bullet$ sea la red de la figura 9.5 con $m = n$.

2. Para cada $x \in pre(t_n)$, si $a = \sigma_n(x)$ añadimos $p(a) \in t'^\bullet_n$ con $F(t'_n, p(a)) = x$, es decir, añadimos lugares y arcos que permitan a $\mathcal{P}$ recibir los tokens que el sistema ofrece.

3. Para cada $x \in post(t_n) \setminus pre(t_n)$, es decir, para cada arco de entrada para el sistema, tenemos que añadir lugares y arcos en $\mathcal{P}$ que permitan comunicarle $a = \sigma_n(x)$ a $\mathcal{S}$. En particular:

   - Si $a \notin \Phi^{io}_{n-1}$ entonces no pertenece al sistema. Entonces añadimos $p(a) \in t'^{\bullet}_n \cap {}^{\bullet}t'_n$, etiquetando esos arcos con x, y con $M'_0(p(a)) = \{a\}$.
   - Si $a \in \Phi^{io}_{n-1}$ entonces es un token del sistema conocido por el entorno. En este caso no basta con añadir un lugar que contenga a a en M'_0, ya que puede que no pueda conocer ese nombre en el marcaje inicial M'_0. Sin embargo, la hipótesis de inducción nos dice que $\Phi^{io}_{n-1} \subseteq S(M'_{n-1})$, así que existe un lugar q tal que $a \in M'_{n-1}(q)$. En consecuencia, nos basta con añadir arcos (q, t'_n) y (t'_n, q), ambos etiquetados por x.

4. Si $x \in pre(t_n) \cap Var_{Aut}$, la definición de $[\rangle^{\#}$ nos asegura que $\sigma_n(x) \in \Phi^{io}_{n-1}$, de manera que por hipótesis de inducción, $\sigma_n(x) \in M'_{n-1}(q)$ para algún q. Entonces podemos proceder como en el punto anterior, añadiendo dos arcos (q, t'_n) y (t'_n, q) etiquetados con x.

Falta probar lo siguiente:

1. $(M_{n-1} + M'_{n-1})[(t_n, t'_n)(\sigma_n)\rangle(M_n + M'_n)$
2. $\Phi^{io}_n \subseteq S(M'_n)$

1. Como $(M_{n-1}, \Phi_{n-1})[t_n(\sigma_n)\rangle^{\#}(M_n, \Phi_n)$ se tiene, por definición de la semántica simbólica, que $\sigma_n(F(p, t_n)) \in M_{n-1}(p)$, para cada $p \in {}^{\bullet}t_n$. Los lugares en ${}^{\bullet}t'_n$ son los siguientes:

   a) p_{n-1}, que está marcado después del disparo de (t_{n-1}, t'_{n-1}),

   b) $p(a)$ con $a = \sigma_n(x) \notin \Phi^{io}_{n-1}$, $x \in post(t_n) \setminus pre(t_n)$. En este caso hemos añadido explícitamente a M'_0 el token a en el lugar $p(a)$. Como los otros disparos no toman tokens de $p(a)$, a sigue en $p(a)$ en el marcaje M'_{n-1},

   c) q con $F(q, t'_n) = x \in post(t_n) \setminus pre(t_n)$ y $a = \sigma_n(x) \in \Phi^{io}_{n-1}$. Por construcción de $\mathcal{P}$, $a \in M'_{n-1}(q)$. De modo similar se prueba el caso en el que está etiquetado con una variable de autenticación.

   Entonces el par de transiciones de sincronización compatibles (t_n, t'_n) está activado en modo σ_n y su disparo produce el marcaje $M_n + M'_n$, para algún M'_n.

2. Por definición de disparo simbólico,

$$\Phi^{io}_n = \Phi^{io}_{n-1} \cup \ \sigma_n(post(t_n)) \cup \ \sigma_n(pre(t_n)).$$

   Si $a \in \Phi^{io}_{n-1}$, por hipótesis de inducción se tiene que $a \in S(M'_{n-1})$, y como, por construcción, nunca se eliminan tokens, $a \in S(M'_n)$. Si $a \in \sigma_n(post(t_n))$,

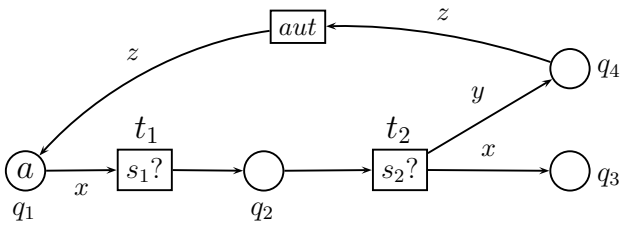

Figura 9.6: Red del ejemplo 11

como t_n y t'_n son compatibles y la sincronización está activada en modo σ_n, ha de ocurrir que $a \in S(M'_{n-1})$ y, al igual que antes, $a \in S(M'_n)$. Finalmente, si $a \in \sigma_n(pre(t_n))$ entonces $a = \sigma(x)$ para $x \in pre(t_n)$. De nuevo, por construcción de $\mathcal{P}$, existe un lugar $p(a) \in t'^{\bullet}_n$ con $F(t'_n, p(a)) = x$, de manera que $a \in S(M'_n)$. En cualquier caso hemos concluido que $a \in S(M'_n)$, así que $\Phi^{io}_n \subseteq S(M'_n)$.

□

El resultado anterior afirma que toda traza simbólica se obtiene a base de añadir las componentes Φ a los marcajes de una traza abierta. Ilustramos la construcción de la demostración anterior por medio de un simple ejemplo.

Ejemplo 11 Consideremos el sistema $\mathcal{S}$ ilustrado en la figura 9.6, compuesto por una sola red y sin transiciones de movimiento, de modo que podemos ignorar la componente localidad de los marcajes, tal y como hacíamos en la demostración. En este ejemplo representaremos los marcajes mediante tuplas de la forma $M = (M^1, M^2, M^3, M^4)$, donde $M^i = M(q_i) \in \mathcal{MS}(Tokens)$. Consideremos también la traza simbólica $M_0^{\Phi_0}[t_1(\sigma_1)\rangle^{\#} M_1^{\Phi_1}[t_2(\sigma_2)\rangle^{\#} M_2^{\Phi_2}$

$$\begin{array}{llll} \textit{donde} & M_0 = (\{a\}, \emptyset, \emptyset, \emptyset) & \Phi_0 = (\emptyset, \emptyset) & \sigma_1 = [x \to a] \\ & M_1 = (\emptyset, \{a\}, \emptyset, \emptyset) & \Phi_1 = (\emptyset, \{a\}) & \sigma_2 = [x \to a, y \to b] \\ & M_2 = (\emptyset, \emptyset, \{a\}, \{b\}) & \Phi_2 = (\{b\}, \{a\}) & \end{array}$$

Se puede comprobar que, en efecto, se trata de una traza simbólica legal, obtenida aplicando dos veces seguidas la regla 4 de la definición de disparos simbólicos. En el primer disparo de la traza, el entorno recibe del sistema el token a a través de la sincronización con este utilizando la transición s_1, de modo que aprende el nombre a. Entonces, en el segundo disparo, el entorno devuelve al sistema el nombre a que acaba de aprender de este, además de un nuevo token b, que el sistema no conocía previamente.

Dada esa traza simbólica, tenemos que construir un entorno legal que genere la correspondiente traza $\emptyset$-abierta.* Como la traza tiene dos pasos que usan la regla

*Para ser más precisos, el entorno ha de conocer, al menos, la localidad donde se encuentra el sistema.

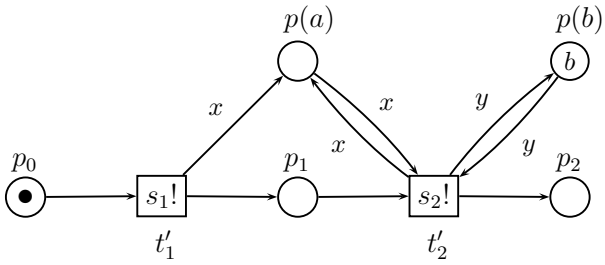

Figura 9.7: $(\emptyset, \mathcal{S})$-entorno $\mathcal{P}$ para la traza del ejemplo 11

4, el entorno ha de tener dos transiciones de sincronización compatibles con las de $\mathcal{S}$, que han de ser disparadas también consecutivamente (por eso añadimos los lugares p_0, p_1 y p_2). Entonces añadimos al entorno dos lugares, $p(a)$ y $p(b)$, que este usará para albergar los tokens a y b, respectivamente. Además, añadimos arcos que permiten al entorno recibir a después de la primera sincronización, y arcos que permiten comunicar al sistema los tokens a y b en la segunda sincronización. Es preciso tener en cuenta que al marcaje inicial sí le está permitido contener el token b en $p(b)$, pero no así con a, ya que es un token del sistema que no se encuentra en $\Phi_0 = \emptyset$. El resultado de esta construcción aparece en la figura 9.7.

El resultado anterior afirma que la semántica simbólica es correcta con respecto a la semántica abierta. A continuación, enunciamos y demostramos el correspondiente resultado de completitud.

Proposición 9.4.4 *Dado un sistema MSPN, si $\mathcal{M}$ es Φ_0-alcanzable y $\Phi'_0 \in split(\Phi_0)$, entonces existe un único Φ tal que $\mathcal{M}^\Phi$ es simbólicamente Φ'_0-alcanzable.*

Demostración. Veamos que si $(M_0, loc_0)[u_1(\sigma_1)\rangle(M_1, loc_1)\ldots[u_n(\sigma_n)\rangle(M_n, loc_n)$ es una traza Φ-abierta de un sistema MSPN $\mathcal{S}$, y $\Phi_0 \in split(\Phi)$, entonces existen $\Phi_1, \ldots, \Phi_n$ únicos tales que

$$(M_0, loc_0, \Phi_0)[u_1(\sigma_1)\rangle^{\#}(M_1, loc_1, \Phi_1)\ldots[u_n(\sigma_n)\rangle^{\#}(M_n, loc_n, \Phi_n)$$

es una traza simbólica de $\mathcal{S}$, lo que implicará la tesis. Como $(M_0, loc_0)[u_1(\sigma_1)\rangle(M_1, loc_1)\ldots$ $[u_n(\sigma_n)\rangle(M_n, loc_n)$ es una traza Φ-abierta, existe un entorno no olvidadizo $\mathcal{P} \in Env(\Phi, \mathcal{M}_0)$ tal que

$$(ML_0 + ML_0^{\mathcal{P}})[\bar{u}_1(\bar{\sigma}_1)\rangle\ldots(ML_{n-1} + ML_{n-1}^{\mathcal{P}})[\bar{u}_n(\bar{\sigma}_n)\rangle(ML_n + ML_n^{\mathcal{P}}),$$

donde $ML_i = (M_i, loc_i)$ y $ML_i^{\mathcal{P}} = (M_i^{\mathcal{P}}, loc_i^{\mathcal{P}})$ son marcajes de $\mathcal{P}$.

Por simplicidad de las notaciones, vamos a suponer que $\bar{u}_i \notin T_{\mathcal{P}} \cup T_{\mathcal{P}} \times T_{\mathcal{P}}$, es decir, que toda $\bar{u}_i$ es o bien una transición autónoma del sistema, o bien un par de transiciones de sincronización del sistema, o la sincronización de una transición del sistema y otra del entorno.

Vamos a probar por inducción sobre n que existen únicos pares de conjuntos de tokens $\Phi_1, \ldots, \Phi_n$ tales que

$$(M_0, loc_0, \Phi_0)[u_1(\sigma_1)\rangle^{\#}(M_1, loc_1, \Phi_1) \ldots [u_n(\sigma_n)\rangle^{\#}(M_n, loc_n, \Phi_n)$$

es una traza simbólica de $\mathcal{S}$ con $\bar{u}_i(\bar{\sigma}_i)|_{\mathcal{S}} = u_i(\sigma_i)$ y tal que $S(ML_n) \cap S(ML_n^{\mathcal{P}}) \subseteq \Phi_n^{io} \subseteq S(ML_n^{\mathcal{P}})$.

Para $n = 0$ tenemos que ver que $S(ML_0) \cap S(ML_0^{\mathcal{P}}) \subseteq \Phi_0^{io} \subseteq S(ML_0^{\mathcal{P}})$, lo que es trivial debido a que $\mathcal{P}$ es un $(\Phi, \mathcal{S})$-entorno no olvidadizo y $\Phi = \Phi_0^{io}$.

Supongamos que los resultados se cumplen para $n-1$ y veamos que también se cumplen para n, suponiendo que $(ML_{n-1} + ML_{n-1}^{\mathcal{P}})[\bar{u}_n(\bar{\sigma}_n)\rangle(ML_n + ML_n^{\mathcal{P}})$. Procedemos por distinción de casos sobre $\bar{u}_n$:

- $\bar{u}_n \in T_{\mathcal{S}} \times T_{\mathcal{S}}$ o $\bar{u}_n \in T_{\mathcal{S}}$, con $\lambda(\bar{u}_n) \in \mathcal{A} \setminus \{new\}$. En este caso el disparo $ML_{n-1}[u_n(\sigma_n)\rangle ML_n$ sigue siendo posible y, por las reglas 1 o 3 de la definición de semántica simbólica, $(M_{n-1}, loc_{n-1}, \Phi_{n-1})[u_n(\sigma_n)\rangle^{\#}(M_n, loc_n, \Phi_n)$ con $\Phi_n = \Phi_{n-1}$, que es el único disparo posible etiquetado con $u_n(\sigma_n)$. Además, como $S(ML_n) \subseteq S(ML_{n-1})$ (las transiciones disparadas de manera autónoma por el sistema distintas de *new* no pueden generar nombres nuevos), $S(ML_n^{\mathcal{P}}) = S(ML_{n-1}^{\mathcal{P}})$ y $\Phi_n = \Phi_{n-1}$, tenemos que $S(ML_n) \cap S(ML_n^{\mathcal{P}}) \subseteq S(ML_{n-1}) \cap S(ML_{n-1}^{\mathcal{P}}) \subseteq \Phi_{n-1}^{io} = \Phi_n^{io} \subseteq S(ML_n^{\mathcal{P}})$.

- $\bar{u}_n \in T_{\mathcal{S}}$ con $\lambda(\bar{u}_n) = new$. Al igual que en el caso anterior, también tenemos $ML_{n-1}[u_n(\sigma_n)\rangle ML_n$. Además, si η es el nombre creado, veamos que $\eta \notin \Phi_{n-1}^{io}$. En efecto, por definición de las transiciones *new*, $\eta \notin S(ML_{n-1}) \cup S(ML_{n-1}^{\mathcal{P}})$. En particular, $\eta \notin S(ML_{n-1}^{\mathcal{P}})$ y como la hipótesis de inducción nos dice que $\Phi_{n-1}^{io} \subseteq S(M_{n-1}^{\mathcal{P}})$, entonces $\eta \notin \Phi_{n-1}^{io}$. Ahora podemos obtener $(M_{n-1}, loc_{n-1}, \Phi_{n-1})[u_n(\sigma_n)\rangle^{\#}(M_n, loc_n, \Phi_n)$ aplicando la regla 2 de la semántica simbólica con $\Phi_n = \Phi_{n-1}$, y ése es el único disparo posible. La otra condición se cumple porque $S(ML_n) \subseteq S(ML_{n-1}) \cup \{\eta\}$, $S(ML_n^{\mathcal{P}}) = S(ML_{n-1}^{\mathcal{P}})$, $\Phi_n = \Phi_{n-1}$ y $\eta \notin S(ML_{n-1}^{\mathcal{P}})$.

- $\bar{u}_n = (t, t') \in T_{N_i} \times T_{\mathcal{P}_i}$. Como $(ML_{n-1} + ML_{n-1}^{\mathcal{P}})[(t, t')(\sigma_n)\rangle(ML_n + ML_n)$ es un disparo legal, podemos inferir que

  - $loc_{n-1}(N_i) = loc_{n-1}^{\mathcal{P}}(\mathcal{P}_i)$. Denotamos $k = loc_{n-1}(N_i)$,
  - $\sigma_n(x) \in S(ML_{n-1}) \cap S(ML_{n-1}^{\mathcal{P}})$ para cada $x \in pre(t) \cap Var_{Aut}$,
  - $\sigma_n(x) \in S(ML_{n-1}^{\mathcal{P}})$ para todo $x \in post(t) \setminus pre(t)$,
  - $\sigma_n(F(p, t)) \in M_{n-1}(p)$ para todo $p \in {}^{\bullet}t$.

  Del primer hecho se sigue que $k \in S(ML_{n-1}) \cap S(ML_{n-1}^{\mathcal{P}}) \subseteq \Phi_{n-1}^{io}$, con lo que $k \in \Phi_{n-1}^{io}$. Análogamente, el segundo punto nos dice que $\sigma_n(x) \in \Phi_{n-1}^{io}$, para todo $x \in pre(t) \cap Var_{Aut}$. Además, se cumple que $\sigma_n(x) \notin S(ML_{n-1}) \setminus \Phi_{n-1}^{io}$, para todo $x \in post(t) \setminus pre(t)$, porque de lo contrario, usando el tercero de los hechos, podríamos obtener $\sigma_n(x) \in S(ML_{n-1}) \cap S(ML_{n-1}^{\mathcal{P}}) \subseteq \Phi_{n-1}^{io}$.

De todas estas conclusiones, junto con el cuarto hecho, podemos concluir que $(M_{n-1}, loc_{n-1}, \Phi_{n-1})[t(\sigma_n)\rangle^{\#}(M_n, loc_n, \Phi_n)$ es un disparo simbólico con $\Phi_n = (\Phi^{in}_{n-1} \cup (\sigma_n(post(t)) \setminus \Phi^{io}_{n-1}), \Phi^{out}_{n-1} \cup (\sigma_n(pre(t)) \setminus \Phi^{io}_{n-1}))$, y claramente es el único posible.

Veamos que $S(ML_n) \cap S(ML^{\mathcal{P}}_n) \subseteq \Phi^{io}_n \subseteq S(ML^{\mathcal{P}}_n)$. Como $loc_n + loc^{\mathcal{P}}_n = loc_{n-1} + loc^{\mathcal{P}}_{n-1}$ podemos concentrarnos en M_n y $M^{\mathcal{P}}_n$. Sea $a \in S(M_n) \cap S(M^{\mathcal{P}}_n)$ y veamos que está en Φ^{io}_n. Si $a \in \Phi^{io}_{n-1}$ entonces, como $\Phi^{io}_{n-1} \subseteq \Phi^{io}_n$ hemos concluido. En caso contrario, en principio se podrían dar cuatro casos:

- $a \in S(M_{n-1}) \setminus S(M^{\mathcal{P}}_{n-1})$. En ese caso, se trata de un token producido por el sistema, es decir, $a \in \sigma_n(pre(t))$, luego $a \in \Phi^{out}_n$.
- $a \in S(M^{\mathcal{P}}_{n-1}) \setminus S(M_{n-1})$. En ese caso, se trata de un token producido por el entorno, es decir, $a \in \sigma_n(post(t))$, con lo que $a \in \Phi^{in}_n$.
- No puede ocurrir que $a \in S(M_{n-1}) \cap S(M^{\mathcal{P}}_{n-1})$ porque $S(M_{n-1}) \cap S(M^{\mathcal{P}}_{n-1}) \subseteq \Phi^{io}_{n-1}$ y estamos suponiendo que $a \notin \Phi^{io}_{n-1}$.
- Tampoco puede ocurrir que $a \notin S(M_{n-1}) \cup S(M^{\mathcal{P}}_{n-1})$ ya que está en $S(M_n) \cap S(M^{\mathcal{P}}_n)$ y las transiciones de sincronización no pueden crear nombres nuevos.

Finalmente, como $\mathcal{P}$ es no olvidadizo, sabemos que $S(M^{\mathcal{P}}_{n-1}) \subseteq S(M^{\mathcal{P}}_n)$ (nunca elimina tokens) y $\sigma_n(pre(t)) \subseteq S(M^{\mathcal{P}}_n)$ (está dispuesto a aceptar todos los tokens que el sistema le ofrece). Entonces, si $a \in \Phi^{io}_n$ tenemos, o bien que $a \in \Phi^{io}_{n-1}$, $a \in \sigma_n(pre(t))$, o bien que $a \in \sigma_n(post(t))$, y de las observaciones anteriores y aplicando la hipótesis de inducción $\Phi^{io}_{n-1} \subseteq S(M^{\mathcal{P}}_{n-1})$, podemos concluir que $a \in S(M^{\mathcal{P}}_n)$, con lo que se sigue la tesis buscada.

□

En consecuencia, se puede *representar* cualquier traza Φ-abierta mediante una traza simbólica. Esto es así sea cual sea la forma en la que decidamos dividir Φ, ya que la activación de las transiciones simbólicas sólo depende de Φ^{io}.

Los dos resultados anteriores prueban de hecho que la semántica simbólica es un *etiquetado* de la semántica Φ-abierta, usando como función de etiquetado inicial justamente la que añade ese Φ al marcaje. Como veíamos en la proposición 9.2.7 las etiquetas se corresponden justamente con la información acerca del sistema acumulada por el entorno a lo largo de la ejecución. Por tanto, según los resultados del capítulo anterior, la semántica simbólica se puede ver como una versión de la semántica abierta, en la que se guarda esa información, con lo que toda propiedad que se verifique en ella es también una propiedad de la semántica original.

9.5. Verificación de sistemas abiertos

En esta sección vamos a estudiar los problemas de alcanzabilidad y recubrimiento para la semántica abierta, que hemos llamado Φ-alcanzabilidad y Φ-recubrimiento.

Se puede ver en primer lugar que la semántica abierta es una extensión de la semántica original.

Proposición 9.5.1 *Todos sistema MSPN puede ser simulado monótonamente por un sistema MSPN abierto.*

Demostración. Dado el sistema MSPN $\mathcal{N}$ basta considerar el correspondiente sistema $\emptyset$-abierto. Entonces estamos suponiendo que cualquier entorno $\mathcal{P}$ cumple que $S(\mathcal{P}) \cap S(\mathcal{N}) = \emptyset$, es decir, no conoce nada acerca del sistema. En particular, para todo k tal que $loc(N) = k$ para alguna componente $N \in \mathcal{N}$, $k \notin S(\mathcal{P})$, es decir, no conoce ninguna de las localidades en las que se pueden encontrar sus componentes. Por tanto, no tiene manera de sincronizar con ellas, y por consiguiente tampoco tiene manera de aprender a hacerlo. Entonces toda traza $\emptyset$-abierta es una traza de $\mathcal{N}$ y viceversa, con lo que el comportamiento del sistema como sistema abierto es equivalente a su comportamiento como sistema cerrado, y la tesis se sigue.

□

Como consecuencia inmediata, por ser la alcanzabilidad indecidible para los sistemas MSPN con identificadores abstractos, tenemos el siguiente corolario.

Corolario 9.5.2 *El problema de Φ-alcanzabilidad es indecidible para sistemas MSPN.*

Afortunadamente, el Φ-recubrimiento se mantiene decidible. Dedicaremos el resto de la sección a probarlo. A partir de ahora trataremos con sistemas centralizados, con todas las componentes en la misma localidad y sin transiciones de movimiento, ya que con una construcción similar a la de la proposición 7.3.4, pero usando un único lugar $@_i$ para guardar la posición de la red N_i, como se hacía en la proposición 5.2.3, podemos simular los sistemas MSPN abiertos con sistemas MSPN centralizados.

Proposición 9.5.3 *Dado una MSPN N, existe un Φ-entorno $\mathcal{P}$ tal que $[\![N \mid \mathcal{P}]\!]$ implementa la semántica simbólica. Más concretamente:*

1. *Si (M, Φ) es simbólicamente alcanzable desde (M_0, Φ_0) entonces $(M, \Phi)^*$ es alcanzable desde $(M_0, \Phi_0)^*$.*

2. *Si $(M, \Phi)^*$ es alcanzable desde $(M_0, \Phi_0)^*$ entonces existe $\bar{\Phi}$ tal que $\bar{\Phi}^{io} \subseteq \Phi^{io}$ y $(M, \bar{\Phi})$ es simbólicamente alcanzable desde (M_0, Φ_0).*

Demostración. Sea $N = (P, T, F, \lambda)$ y definamos $\mathcal{P} = (\overline{P}, \overline{T}, \overline{F}, \overline{\lambda})$, que tendrá arcos etiquetados con conjuntos de variables, lo que se puede permitir por la proposición 5.2.4. El conjunto de lugares $\overline{P}$ contendrá un único lugar, que contendrá los tokens que el sistema le dé, junto con los que el entorno genere.

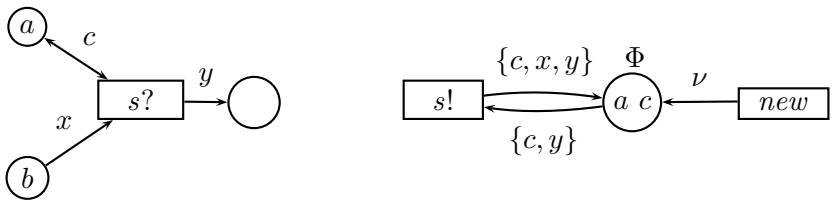

Figura 9.8: Sistema MSPN y $\{a\}$-entorno que implementa su semántica simbólica

Tendremos una transición por cada transición de sincronización del sistema, que sea compatible con ella, y con arcos adyacentes etiquetados de manera que reciba del sistema todo lo que puede recibir y le proporcione lo que tenga que proporcionarle. Además, el conjunto $\overline{T}$ contiene una transición que cree identificadores. Formalmente tenemos lo siguiente.

- $\overline{P} = \{\Phi\}$,
- $\overline{T} = \{\overline{t} \mid t \in T,\ \lambda(t) \in \mathit{Sync}\} \cup \{t_{new}\}$,
- $\overline{F}(\Phi, \overline{t}) = (\mathit{post}(t) \setminus \mathit{pre}(t)) \cup (\mathit{pre}(t) \cap \mathit{Var}_{Auth})$,
- $\overline{F}(\overline{t}, \Phi) = \mathit{pre}(t) \cup F(\overline{t}, \Phi)$,
- $\overline{F}(t_{new}, \Phi) = \nu$,
- $\overline{\lambda}(\overline{t}) = \overline{\lambda(t)}$,
- $\overline{\lambda}(t_{new}) = \mathit{new}$

Además, dado un marcaje simbólico (M, Φ) de $\mathcal{N}$ definimos el marcaje $(M, \Phi)^*$ de $\mathcal{N} \mid \mathcal{P}$ dado por $(M, \Phi)^*(p) = \begin{cases} M(p) \text{ si } p \neq \Phi \\ \Phi^{io} \text{ si } p = \Phi \end{cases}$

Veamos primero que si (M, Φ) es simbólicamente alcanzable desde (M_0, Φ_0) entonces $(M, \Phi)^*$ es alcanzable desde $(M_0, \Phi_0)^*$. Veamos que si $(M_1, \Phi_1)[u(\sigma)\rangle^{\#}(M_2, \Phi_2)$ entonces $(M_1, \Phi_1)^* \rightarrow^* (M_2, \Phi_2)^*$, con lo que tendremos la implicación a la derecha. En cualquiera de los casos 1, 2 y 3 de la definición 9.4.2 se tiene que $M_1[u(\sigma)\rangle M_2$ con u transición (o par de transiciones) de $\mathcal{N}$ y, por tanto, también se tiene que $(M_1, \Phi_1)^*[u(\sigma)\rangle(M_2, \Phi_2)^*$. Supongamos entonces que estamos en el caso 4 de la definición 9.4.2, es decir, $u = t$ con $\lambda(t) \in \mathit{Sync}$ y

- $\sigma(F(p, t)) \in M_1(p)$ para toda precondición p de t en $\mathcal{N}$,
- $\sigma(x) \in \Phi_1^{io}$ para todo $x \in \mathit{pre}(t) \cap \mathit{Var}_{Auth}$,
- $\sigma(x) \notin S(M_1) \setminus \Phi_1^{io}$ para $x \in \mathit{post}(t) \setminus \mathit{pre}(t)$.

Entonces se puede disparar $t(\sigma)$ simbólicamente, obteniendo un marcaje (M_2, Φ_2). Veamos que en $\mathcal{N} \mid \mathcal{P}$ se puede disparar la sincronización $(t, \overline{t})(\sigma)$, obteniendo así $(M_2, \Phi_2)^*$. Como para todo p de $\mathcal{N}$ el marcaje $(M_1, \Phi_1)^*$ coincide con M_1, para ver que la sincronización está activada basta ver que $\sigma(F(\Phi, \overline{t})) \subseteq (M_1, \Phi_1)^*(\Phi)$. En efecto, si $x \in F(\Phi, \overline{t})$ tenemos que ver que $\sigma(x) \in \Phi_1^{io}$. Por construcción, por ser $x \in pre(\overline{t})$ ha de ser o bien $x \in (post(t) \setminus pre(t)) \setminus Var_{Auth}$ o bien $x \in pre(t) \cap Var_{Auth}$. En el primero de los casos, por el último punto de la lista anterior tenemos que $\sigma(x) \notin S(M_1) \setminus \Phi_1^{io}$, es decir, $\sigma(x) \notin S(M_1)$ o $\sigma(x) \in S(M_1) \cap \Phi_1^{io}$. Sin embargo, si ocurre lo primero aún puede ocurrir que $\sigma(x) \in S(M_1)$. Si no ocurre entonces $\sigma(x)$ no aparece en ningún lugar y, por lo tanto, podemos disparar la transición t_{new} para producir exactamente $\sigma(x)$ en el lugar Φ. Supongamos ahora que estamos en el segundo caso, es decir, $x \in pre(t) \cap Var_{Auth}$. En este caso el segundo punto del anterior listado nos dice directamente que $\sigma(x) \in (M_1, \Phi_1)^*(\Phi) = \Phi_1^{io}$ y la transición está activada.

El disparo alcanzado $\overline{M}$ coincide con $(M_2, \Phi_2)^*$ en los lugares de $\mathcal{N}$ porque para cualquier p de $\mathcal{N}$ se tiene que $(M_2, \Phi_2)^*(p) = M_2(p) = M_1(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$. Veamos que también coinciden en el lugar Φ. Por construcción de $\mathcal{P}$, todo token que hubiese en Φ sigue en él porque $\overline{F}(\Phi, \overline{t}) \subseteq \overline{F}(\overline{t}, \Phi)$. Entonces $\overline{M}(\Phi) = \Phi_1^{io} \cup \sigma(pre(t)) \cup \{\sigma(x) \mid x \in (post(t) \setminus pre(t)) \setminus Var_{Auth}\}$, donde $\{\sigma(x) \mid x \in (post(t) \setminus pre(t)) \setminus Var_{Auth}\} \subseteq \overline{M}(\Phi)$ porque para todos los que no estaban en Φ hemos disparado t_{new} para crearlos. Por definición de semántica simbólica tenemos que $\Phi_2^{io} = \Phi_1^{io} \cup \sigma(pre(t)) \cup \sigma(post(t))$. Claramente, $\overline{M}(\Phi) \subseteq \Phi_2^{io}$. Veamos que también se tiene la otra inclusión. El único caso no inmediato es el de $\sigma(x)$ con $x \in post(t)$ tal que $x \notin (post(t) \setminus pre(t)) \setminus Var_{Auth}$. Si x no está en ese conjunto estando en $post(t)$ puede ser porque también pertenezca a $pre(t)$ o porque, no perteneciendo a $pre(t)$ sea una variable de Var_{Auth}. Si $x \in pre(t)$ entonces claramente $\sigma(x) \in \Phi_2^{io}$. El segundo caso no puede ocurrir porque habíamos descartado transiciones con variables $x \in (post(t) \setminus pre(t)) \cap Var_{Auth}$.

Para el recíproco, veamos que si $(M, \Phi)^*$ es alcanzable desde (M_0, Φ_0) entonces existe $\bar{\Phi}$ tal que $\bar{\Phi}^{io} \subseteq \Phi^{io}$ y $\Phi^{io} = \bar{\Phi}^{io} \cup \{\eta_1, \ldots, \eta_k\}$ con $\eta_i \notin S(M)$ para todo $i = 1, \ldots, k$ y $(M, \bar{\Phi})$ es simbólicamente alcanzable desde (M_0, Φ_0). Procedemos por inducción sobre la longitud del cómputo. Si dicha longitud es nula entonces el resultado se sigue trivialmente. Como para todo marcaje de $\mathcal{N} \mid \mathcal{P}$ es imagen por $(\ )^*$ de algún marcaje simbólico de $\mathcal{N}$ para el paso inductivo tenemos que ver que si $(M', \Phi')^*$ es simbólicamente alcanzable y $(M', \Phi')^*[u(\sigma)\rangle^{\#}(M, \Phi)^*$ también se sigue la tesis. La hipótesis de inducción nos dice que existe $\bar{\Phi}'^{io} \subseteq \Phi'^{io}$ tal que $(M', \bar{\Phi})$ es simbólicamente alcanzable y $\Phi^{io} = \bar{\Phi}^{io} \cup \{\eta_1, \ldots, \eta_k\}$ con $\eta_i \notin S(M)$ para todo $i = 1, \ldots, k$. Distingamos entre los posibles casos para u.

- Si $u = t_{new}(\sigma)$ entonces $M = M'$ y $\Phi^{io} = \Phi'^{io} \cup \{\sigma(\nu)\}$, por lo que $\bar{\Phi}^{io} = \bar{\Phi}'^{io} \cup \{\eta_1, \ldots, \eta_k, \sigma(\nu)\}$ con $\sigma(\nu) \notin S(M)$ y la tesis se sigue.

- Si u es una transición (o par de transiciones) de $\mathcal{N}$ se cumple que $M'[u\rangle M$ y por lo tanto $(M', \Phi')[u\rangle^{\#}(M, \Phi')$, por lo que la tesis se sigue tomando $\bar{\Phi} = \bar{\Phi}'$.

- Supongamos ahora que $u = (t, \overline{t})(\sigma)$ con t transición de $\mathcal{P}$ y $\overline{t}$ de $\mathcal{N}$ y veamos que se cumple que $(M', \bar{\Phi}')[\overline{t}(\sigma)\rangle^{\#}(M, \Phi)$.

  - En primer lugar, si σ es un modo de $(t, \overline{t})$ entonces es modo simbólico de $\overline{t}$.
  - Veamos ahora que $\sigma(x) \in \bar{\Phi}'^{io}$ para todo $x \in pre(\overline{t}) \cap Var_{Auth}$. Por definición de transiciones compatibles ha de ocurrir que $x \in pre(t)$ y, por lo tanto, $\sigma(x) \in (M', \Phi')^*(\Phi) = \Phi'^{io}$. Además, como $x \in pre(t)$ se tiene que $\sigma(x) \in S(M')$ y por lo tanto $\sigma(x) \in \bar{\Phi}'^{io}$.
  - Si $x \in post(\overline{t}) \setminus pre(\overline{t})$ entonces, por definición de compatibilidad de transiciones se tiene que $x \in pre(t)$, de modo que $\sigma(x) \in (M', \Phi')^*(\Phi) = \Phi'^{io}$. Pues bien, si $\sigma(x) \notin S(M')$ claramente $\sigma(x) \notin S(M') \setminus \bar{\Phi}'^{io}$ y si, por el contrario, $\sigma(x) \in S(M')$ entonces $\sigma(x)$ no es de los identificadores frescos de Φ'^{io}, lo que, como en el punto anterior, quiere decir que $\sigma(x) \in \bar{\Phi}'^{io}$ y también se deduce que $\sigma(x) \notin S(M') \setminus \bar{\Phi}'^{io}$.
  - Por último, claramente $\sigma(F(p, \overline{t})) \in M'(p)$ para toda precondición p de $\overline{t}$, pues también lo es en $\mathcal{N} \mid \mathcal{P}$.

  Entonces $\overline{t}(\sigma)$ se puede disparar simbólicamente, alcanzando un marcaje simbólico $(M, \bar{\Phi})$. Sólo queda ver que $\Phi^{io} = \Phi'^{io} \cup \{\eta_i\}$. En efecto, $\Phi^{io} = \Phi'^{io} \cup \sigma(Var(\overline{t})) = \bar{\Phi}'^{io} \cup \sigma(Var(\overline{t})) \cup \{\eta_i\} = \bar{\Phi} \cup \{\eta_i\}$ y la tesis se sigue.

□

Se puede ver un ejemplo de la construcción anterior en la figura 9.8, suponiendo que la variable c están en Var_{Auth}. Esta construcción no es una simulación monótona por la sencilla razón de que la función de representación de marcajes simbólicos no es inyectiva, ya que no se distingue entre los identificadores que deberían estar en Φ^{in} y en Φ^{out}. Es más, incluso si nos olvidásemos de esta distinción en la semántica simbólica, seguiría sin ser una simulación, porque el contenido del lugar Φ es en realidad un superconjunto del contenido de Φ en la semántica simbólica. Aun así, la construcción nos permite establecer el siguiente resultado.

Corolario 9.5.4 *El problema del Φ-recubrimiento es decidible.*

Demostración. Por las proposiciones 9.4.3, 9.4.4, 9.5.3 y 5.2.4 se tiene que M es Φ-recubrible desde $M_0 \Leftrightarrow (M, \emptyset)$ es recubrible desde $(M_0, \Phi) \Leftrightarrow (M, \emptyset)^*$ es recubrible desde $(M, \emptyset)^*$ en la anterior simulación. Como el recubrimiento es decidible para sistemas MSPN con identificadores abstractos la tesis se sigue.

□

Capítulo 10

Implementación de las redes de Petri móviles

Los capítulos anteriores han establecido las bases teóricas necesarias para la verificación de sistemas modelados por medio de sistemas de redes de Petri móviles. En este capítulo abordamos el problema de su verificación automática. Para ello hemos elegido el marco de la lógica de reescritura [MOM02]. Esta lógica admite de manera natural la representación de sistemas distribuidos y concurrentes, y ha sido implementada eficientemente utilizando el sistema Maude [CDE+02]. Dicho lenguaje ha sido utilizado ampliamente para implementar numerosas álgebras de procesos y otros formalismos para la concurrencia, con la posibilidad de verificar de forma automática algunas de sus propiedades [VMO02, TSMO02, RVSV06, SMÖ01a].

Para llevar a cabo la traducción de los sistemas MSPN a la lógica de reescritura usaremos como paso intermedio las ν-APNs, que, como hemos visto, tienen la potencia de los sistemas MSPN y son fácilmente representables como teorías de reescritura.

Como la teoría de reescritura que resulta de traducir primero un sistema MSPN a una ν-APN, y después ésta a la lógica de reescritura, dista bastante de la original, usamos las propiedades reflexivas de Maude para realizar automáticamente esa traducción, así como comandos predefinidos de Maude para llevar a cabo algunos análisis.

10.1. ν-APNs en lógica de reescritura

Queremos representar las ν-APNs en Maude de manera similar a como se hace habitualmente para las redes de Petri ordinarias. Empezaremos suponiendo que nuestras redes no etiquetan ningún arco con la variable ν, es decir, que son APNs. Para representar una APN en Maude definiremos una *forma normal* de conjuntos de marcajes equivalentes. Ahora que tenemos tokens distinguibles no basta con considerar el conjunto de lugares ocupados por tokens, sino que tenemos que hacer

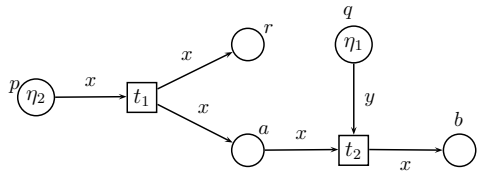

Figura 10.1: Ejemplo de APN

eso mismo para cada uno de los distintos tokens, con lo que se obtiene no un multiconjunto de lugares, sino un multiconjunto de multiconjuntos de lugares con un multiconjunto por cada identificador del marcaje. Ello nos lleva a representar no marcajes, sino clases de equivalencia de marcajes módulo $\equiv_\alpha$.

Sin embargo, la estructura habitual de los multiconjuntos (de multiconjuntos) no basta para nuestros propósitos. Esto es debido a que no queremos considerar multiconjuntos que contengan al multiconjunto vacío, ya que, intuitivamente, el multiconjunto vacío se correspondería con un identificador que no aparece y por tanto puede ser eliminado. En lugar de restringir nuestro dominio a los multiconjuntos que no contienen al multiconjunto vacío, por homogeneidad en otras definiciones futuras y por cercanía a como se implementará en Maude preferimos proceder de la siguiente manera:

Definición 10.1.1 *Sea $\backsim$ la menor relación de equivalencia en $\mathcal{MS}(\mathcal{MS}(P))$ tal que $\mathcal{A} + \{\emptyset\} \backsim \mathcal{A}$ para todo $\mathcal{A} \in \mathcal{MS}(\mathcal{MS}(P))$.*

En lo sucesivo, trabajaremos sobre el cociente $\mathcal{MS}(\mathcal{MS}(P))/_\backsim$. Sin embargo, para no complicar demasiado las notaciones, escribiremos simplemente $\mathcal{A}$, en lugar de $[\mathcal{A}]_\backsim$, para denotar los elementos de $\mathcal{MS}(\mathcal{MS}(P))/_\backsim$. Además, en ocasiones usaremos la mera yuxtaposición para representar la unión en $\mathcal{MS}(P)$, y el operador $+$ para denotar la unión en $\mathcal{MS}(\mathcal{MS}(P))$, junto con los correspondientes operadores extendidos $\prod$ y $\sum$. Asimismo, si $\mathcal{A}$ es un multiconjunto, cuando no haya confusión escribiremos simplemente $\mathcal{A}$ para referirnos al multiconjunto (de multiconjuntos) unitario $\{\mathcal{A}\}$.

Antes de pasar a definir formalmente la representación de los marcajes como multiconjuntos, veamos un ejemplo. El marcaje que se muestra en la figura 10.1 contiene dos identificadores distintos, η_1 y η_2, de modo que necesitaremos dos multiconjuntos para representarlo, uno por cada nombre. El identificador η_1 aparece en q, así que lo representamos mediante el multiconjunto $\{q\}$, y η_2 sólo aparece en p, de manera que lo representaremos mediante $\{p\}$. Entonces al marcaje en cuestión le haremos corresponder el multiconjunto $\{\{p\}, \{q\}\}$, que denotaremos mediante $p+q$. Después de los disparos de t_1 y t_2 el marcaje alcanzado es aquél con solamente η_2 en los lugares r y b. Nos referiremos al correspondiente multiconjunto simplemente mediante rb.

Si M y M' son dos marcajes de una APN N, escribiremos $M + M'$ para denotar al marcaje definido mediante $(M + M')(p) = M(p) + M'(p)$. Teniendo

cuidado, podemos trasladar esta definición a clases de marcajes.

Definición 10.1.2 *Sean* $[M_1]_{\equiv_\alpha}, [M_2]_{\equiv_\alpha} \in Markings/_{\equiv_\alpha}$. *Definimos* $[M_1]_{\equiv_\alpha} + [M_2]_{\equiv_\alpha}$ *como* $[M_1' + M_2']_{\equiv_\alpha}$, *donde* $M_1 \equiv_\alpha M_1'$, $M_2 \equiv_\alpha M_2'$ *y* $S(M_1') \cap S(M_2') = \emptyset$.

Lema 10.1.3 *La definición anterior no depende de los representantes de las clases utilizados en ella.*

Demostración. Por transitividad de $\equiv_\alpha$ tenemos que ver que si $M_1 \equiv_\alpha M_1'$ y $M_2 \equiv_\alpha M_2'$, con $S(M_1) \cap S(M_2) = S(M_1') \cap S(M_2') = \emptyset$, entonces $M_1 + M_2 \equiv_\alpha M_1' + M_2'$. Por el lema 6.2.2 existen biyecciones $h_i : S(M_i) \to S(M_i')$ tales que $M_i' = h_i(M_i)$, para $i = 1, 2$. Como $S(M_1 + M_2) = S(M_1) \cup S(M_2)$ y $S(M_1' + M_2') = S(M_1') \cup S(M_2')$, podemos definir $h : S(M_1 + M_2) \to S(M_2' + M_2')$ mediante $h(a) = \begin{cases} h_1(a) \text{ si } a \in S(M_1) \\ h_2(a) \text{ si } a \in S(M_2) \end{cases}$. Como $S(M_1) \cap S(M_2) = \emptyset$ la función h está bien definida, y es una biyección porque h_1 y h_2 lo eran y $S(M_1') \cap S(M_2') = \emptyset$. Además, se puede ver que $h(M_1 + M_2) = M_1' + M_2'$. Por ejemplo, si $b \in S(M_1')$ entonces $h(M_1 + M_2)(p)(b) = (M_1 + M_2)(p)(h^{-1}(b)) = M_1(p)(h^{-1}(b)) + M_2(p)(h^{-1}(b)) = M_1(p)(h^{-1}(b)) = h_1(M_1)(p)(b) = M_1'(p)(b) = (M_1' + M_2')(p)(b)$. De nuevo por el lema 6.2.2, podemos concluir que $M_1 + M_2 \equiv_\alpha M_1' + M_2'$, y la tesis se sigue.

□

La siguiente definición es análoga a la definición 6.2.6 del capítulo 6, para clases de marcajes.

Definición 10.1.4 *Sea* $N = (P, T, F)$ *una APN. Definimos* $NF : Markings(N)/_{\equiv_\alpha} \to \mathcal{MS}(\mathcal{MS}(P))$ *por medio de* $NF([M]_{\equiv_\alpha}) = \mathcal{A}_M = \{M_a \mid a \in S(M)\}$, *donde* $M_a \in \mathcal{MS}(P)$ *se define como* $M_a(p) = M(p)(a)$.

En principio, la definición anterior también está formalizada usando representantes concretos, aunque cualesquiera, de las clases de equivalencia. Pero es fácil ver que todos los representantes se comportan de la misma manera para esa definición y que, por tanto, NF caracteriza representantes canónicos de las clases de equivalencia de $\equiv_\alpha$.

Lema 10.1.5 $M \equiv_\alpha M' \Leftrightarrow \mathcal{A}_M = \mathcal{A}_{M'}$

Demostración.

$\Rightarrow$ Lo probamos por inducción sobre las reglas usadas para establecer $M \equiv_\alpha M'$. Los casos de reflexividad, simetría y transitividad son triviales. Supongamos pues que $M' = M[b/a]$ con $a \in S(M)$ y $b \notin S(M)$. Si $c \in Id$ es distinto de a y de b entonces $M_c(p) = M(p)(c) = M'(p)(c) = M_c'(p)$ y, por tanto, $M_c = M_c'$. Es inmediato que $M_b = M_a' = \emptyset$.

Finalmente, $M_b'(p) = M'(p)(b) = M[b/a](p)(b) = M(p)(a) = M_a(p)$, lo que implica que $M_b' = M_a$. Este hecho nos permite concluir que $\mathcal{A}_{M'} = (\sum_{c \neq a,b} M_c') + M_a' + M_b' = (\sum_{c \neq a,b} M_c) + M_b + M_a' = \mathcal{A}_M$.

$\Leftarrow$ Supongamos que $\mathcal{A}_M = \mathcal{A}_{M'}$ y veamos que existe una biyección $h : S(M) \to S(M')$ tal que $h(M) = M'$, lo que implicará, por el lema 6.2.2 que $M \equiv_\alpha M'$. Sabemos que $\sum_{a \in Id} M_a = \sum_{a \in Id} M'_a$. Sin embargo, si $a \notin S(M)$ entonces $M_a = \emptyset$, así que podemos restringir esas uniones, obteniendo $\sum_{a \in S(M)} M_a = \sum_{a \in S(M')} M'_a$, donde todos los M_a y los M'_a son no vacíos. Entonces, existe una biyección $h : S(M) \to S(M')$ tal que $M_a = M'_{h(a)}$. Por tanto, $M(p)(a) = M_a(p) = M'_{h(a)}(p) = M'(p)(h(a))$, lo que quiere decir que $M' = h(M)$, como queríamos demostrar.

□

Corolario 10.1.6 *NF está bien definida y es una inyección.*

Demostración. La implicación $\Rightarrow$ del lema anterior nos demuestra que la definición de $NF([M]_\alpha)$ no depende del representante de $[M]_{\equiv_\alpha}$ que se haya elegido. La otra implicación prueba la inyectividad de NF.

□

Por supuesto, la unión en $\mathcal{MS}(\mathcal{MS}(P))$ es conmutativa y asociativa, de manera que $(\mathcal{MS}(\mathcal{MS}(P)), +, \emptyset)$ es un monoide conmutativo. Estas propiedades se mantienen en el cociente.

Proposición 10.1.7 $(\mathcal{MS}(\mathcal{MS}(P))/_\backsim, +, [\{\}]_\backsim)$ *es un monoide conmutativo, donde $+$ está (bien) definido por $[\mathcal{A}]_\backsim + [\mathcal{B}]_\backsim = [\mathcal{A} + \mathcal{B}]_\backsim$.*

Queremos usar el dominio anterior para representar los marcajes, así que vamos a ver que el análogo al resultado anterior también se cumple para ellos. En la próxima definición denotaremos mediante 0 al marcaje definido por $0(p) = \emptyset$ para todo p.

Proposición 10.1.8 *Dada una APN N, $(Markings(N)/_{\equiv_\alpha}, +, [0]_{\equiv_\alpha})$ es un monoide conmutativo.*

Demostración. La unión de marcajes es conmutativa y asociativa, y tiene a $0(p) = \emptyset$ para todo p, como unidad. Es inmediato ver que estas propiedades se mantienen cuando pasamos al conjunto cociente. Es decir, si suponemos que M_1, M_2 y M_3 tienen conjuntos de identificadores disjuntos, entonces $[M_1]_{\equiv_\alpha} + ([M_2]_{\equiv_\alpha} + [M_3]_{\equiv_\alpha}) = ([M_1]_{\equiv_\alpha} + [M_2]_{\equiv_\alpha}) + [M_3]_{\equiv_\alpha}$ y $[M_1]_{\equiv_\alpha} + [M_2]_{\equiv_\alpha} = [M_2]_{\equiv_\alpha} + [M_1]_{\equiv_\alpha}$. Además, $[M]_{\equiv_\alpha} + [0]_{\equiv_\alpha} = [M]_{\equiv_\alpha}$.

□

Veamos que nuestra traducción preserva las citadas propiedades.

Proposición 10.1.9 *NF es un isomorfismo de monoides conmutativos.*

Demostración. Por el corolario 10.1.6 basta ver que NF es sobreyectiva y que tanto NF como NF^{-1} son homomorfismos. Veamos primero que NF es sobreyectiva. Para ello tomemos $\mathcal{A} = \{A_1, \ldots, A_n\}$, con $A_i \in \mathcal{MS}(P)$. Sea $I = \{\eta_i \mid i = 1, \ldots, n\}$ un conjunto cualquiera de n elementos de Id distintos dos a dos. Definimos el marcaje M como $M(p)(a) = 0$ para todo $a \notin I$, y $M(p)(\eta_i) = A_i(p)$ para $i = 1, \ldots, n$. Pues bien, el marcaje M satisface $NF([M]_\alpha) = \mathcal{A}$.

Ahora veamos que $NF([M]_{\equiv_\alpha} + [M']_{\equiv_\alpha}) = NF([M]_{\equiv_\alpha}) + NF([M']_{\equiv_\alpha})$. Para ello, elijamos M y M' de modo que se cumpla $S(M) \cap S(M') = \emptyset$ y veamos que $\mathcal{A}_{M+M'} = \mathcal{A}_M + \mathcal{A}_{M'}$. En efecto, $\mathcal{A}_{M+M'} = \sum\limits_{a \in Id} (M + M')_a = \sum\limits_{a \in S(M)} (M + M')_a + \sum\limits_{a \in S(M')} (M + M')_a = \sum\limits_{a \in S(M)} M_a + \sum\limits_{a \in S(M')} M'_a = \sum\limits_{a \in Id} M_a + \sum\limits_{a \in Id} M'_a = \mathcal{A}_M + \mathcal{A}_{M'}$.

Por último, veamos que $NF^{-1}(A_1 + A_2) = NF^{-1}(A_1) + NF^{-1}(A_2)$. Dada la sobreyectividad de NF, basta probar que si $NF([M]_{\equiv_\alpha}) = A_1 + A_2$ entonces existen M_1 y M_2 tales que $NF([M_1]_{\equiv_\alpha}) = A_1$, $NF([M_2]_{\equiv_\alpha}) = A_2$, y $[M]_{\equiv_\alpha} = [M_1]_{\equiv_\alpha} + [M_2]_{\equiv_\alpha}$. Como $NF([M]_{\equiv_\alpha}) = \sum\limits_{a \in Id} M_a = A_1 + A_2$ existen $I_1, I_2 \subseteq Id$ tales que $I_1 \cap I_2 = \emptyset$ y $A_i = \sum\limits_{a \in I_i} M_a$. Definamos $M_i(p)(a) = M(p)(a)$ si $a \in I_i$ y $M_i(p)(a) = 0$ en caso contrario. Trivialmente, $M = M_1 + M_2$ y como $S(M_i) = I_i$ entonces $[M]_{\equiv_\alpha} = [M_1 + M_2]_{\equiv_\alpha} = [M_1]_{\equiv_\alpha} + [M_2]_{\equiv_\alpha}$. Finalmente, $NF([M_i]_{\equiv_\alpha}) = \sum\limits_{a \in Id} (M_i)_a = \sum\limits_{a \in I_i} (M_i)_a = \sum\limits_{a \in I_i} M_a = A_i$. Para concluir basta ver que $NF([0]_{\equiv_\alpha}) = \sum\limits_{a \in Id} 0_a = \emptyset$, ya que $0_a = \emptyset$, para todo $a \in Id$, y estamos trabajando módulo $\backsim$.

□

Por tanto, podemos representar toda una clase de marcajes, $[M]_{\equiv_\alpha}$ como el multiconjunto (de multiconjuntos) $NF([M]_{\equiv_\alpha})$. Ahora veamos cómo traducimos el comportamiento operacional de una APN a ese dominio. Las transiciones tienen la forma $M_1[t(\sigma_1)\rangle M'_1$. Sin embargo, en los sistemas MSPN (y en las ν-APNs), se pueden crear tokens con nombres nuevos. En particular, el nombre específico de estos tokens es irrelevante, ya que cuando se crea un nuevo nombre se puede utilizar como tal cualquier nombre nuevo. Los marcajes abstractos de las APNs capturan precisamente esta propiedad a través de la α-equivalencia. De hecho, vimos en el capítulo 6 que si $M_1 \equiv_\alpha M_2$ entonces existe un σ_2 tal que $M_2[t(\sigma_2)\rangle M'_2$, con $M'_1 \equiv_\alpha M'_2$. Además, cuando se trabaja módulo α-conversión, un modo sólo identifica las relaciones entre los distintos tokens involucrados en el disparo de la transición y no los valores individuales de esos tokens. Todo ello queda formalizado por medio de las siguientes definiciones.

Definición 10.1.10 *Sea σ un modo de una transición t. Definimos la relación $\sim_\sigma$ entre variables mediante $x \sim_\sigma y \Leftrightarrow \sigma(x) = \sigma(y)$.*

La relación $\sim_\sigma$ identifica las variables que están instanciadas por el modo σ al mismo valor. Ésa es exactamente la información acerca de los modos en la que estamos interesados, de manera que identificamos a los modos módulo esa información, como se formaliza en la siguiente definición.

Definición 10.1.11 *Sea t una transición. Definimos la relación $\approx_t$ entre modos de t mediante $\sigma_1 \approx_t \sigma_2 \Leftrightarrow \ \sim_{\sigma_1} = \sim_{\sigma_2}$.*

Proposición 10.1.12 *Supongamos que t está activada en M_1 con modo σ_1 y $M_1 \equiv_\alpha M_2$. Entonces existe un modo σ_2 con $\sigma_1 \approx_t \sigma_2$ tal que t está activada en M_2 con modo σ_2. Además, los marcajes obtenidos por los disparos de $t(\sigma_1)$ y $t(\sigma_2)$ sobre M_1 y M_2, respectivamente, son también α-equivalentes.*

Demostración. Basta tener en cuenta que en la proposición 6.2.3, por ser h una biyección, $\sigma(x) = \sigma(y) \Leftrightarrow h(\sigma)(x) = h(\sigma)(y)$ y, por lo tanto, $\sigma \approx_t h(\sigma)$, por lo que basta tomar $\sigma_2 = h(\sigma_1)$ para obtener la tesis.

□

Por tanto, cuando trabajamos módulo α-conversión, los disparos tienen de hecho la forma $[M_1]_{\equiv_\alpha}[t([\sigma]_{\approx_t})\rangle[M_2]_{\equiv_\alpha}$. Nuestro objetivo es simular esos disparos por medio de reescrituras del tipo $NF([M_1]_{\equiv_\alpha}) \to NF([M_2]_{\equiv_\alpha})$. Veamos qué reglas generan esas reescrituras. De hecho, veremos que cada par $t([\sigma]_{\approx_t})$ da lugar a una regla diferente. Antes, una definición auxiliar.

Definición 10.1.13 *Sea t una transición de una APN $N = (P, T, F)$ y σ un modo de t. Definimos $\mathcal{P}(t(\sigma)) = Var(t)/_{\sim_\sigma}$.*

Trataremos a las clases de $\mathcal{P}(t(\sigma))$ como conjuntos formados por sus representantes, de manera que, en particular, escribiremos expresiones como $Var \supseteq X \in \mathcal{P}(t(\sigma))$.

Lema 10.1.14 *Si $\sigma_1 \approx_t \sigma_2$ entonces $\mathcal{P}(t(\sigma_1)) = \mathcal{P}(t(\sigma_2))$.*

Demostración. $\sigma_1 \approx_t \sigma_2 \Rightarrow \ \sim_{\sigma_1} = \sim_{\sigma_2} \Rightarrow Var(t)/_{\sim_{\sigma_1}} = Var(t)/_{\sim_{\sigma 2}} \Rightarrow \mathcal{P}(t(\sigma_1)) = \mathcal{P}(t(\sigma_2))$

□

Ya hemos asentado toda la maquinaria que necesitaremos para la traducción de la semántica operacional de las APNs a la lógica de reescritura. En lo sucesivo, denotaremos mediante V al conjunto de metavariables de la lógica, para distinguirlas de las variables de una APN. Además, usaremos inyecciones $FV : \mathcal{P}(t(\sigma)) \to V$.

Definición 10.1.15 *Sea σ un modo de una transición t. Definimos $rl(t(\sigma))$ como la regla $l_{t(\sigma)} \to r_{t(\sigma)}$, donde*

$$l_{t(\sigma)} = \sum_{X \in \mathcal{P}(t(\sigma))} (\prod_{x \in X} Pre_x(t)) FV(X),$$

$$r_{t(\sigma)} = \sum_{X \in \mathcal{P}(t(\sigma))} (\prod_{x \in X} Post_x(t)) FV(X).$$

Lema 10.1.16 *Si $\sigma_1 \approx_t \sigma_2$ entonces $rl(t(\sigma_1))$ y $rl(t(\sigma_2))$ son iguales, salvo renombramiento consistente de sus variables libres.*

Demostración. Trivial por el lema 10.1.14.

□

Si rl es una regla de una teoría de reescritura, escribiremos $t_1 \xrightarrow{rl} t_2$ si se puede probar $t_1 \to t_2$ usando la regla de *reemplazamiento* una vez, utilizando en concreto la regla rl. En la demostración del teorema 10.1.19, resultado principal de este capítulo, usaremos los dos lemas siguientes. El primero es un hecho trivial en la lógica de reescritura, que usaremos a menudo.

Lema 10.1.17 *Si se puede probar $t \to t'$ sin aplicar la regla de reemplazamiento, entonces $t = t'$.*

Lema 10.1.18 *Si $M[t(\sigma)\rangle M'$ y $M_a = (\prod_{x \in X} Pre_x(t))Q$, para $X \in \mathcal{P}(t(\sigma))$ y $Q \in \mathcal{MS}(P)$, entonces $M'_a = (\prod_{x \in X} Post_x(t))Q$.*

Demostración. Por definición de disparo, $M'(p) = M(p) - \{\sigma(F(p,t))\} + \{\sigma(F(t,p))\}$. Entonces, $M'(p)(a) = M(p)(a) + \delta^p_X$, donde

$$\delta^p_X = \begin{cases} 1 \text{ si } p \in \prod_{x \in X} Post_x(t) \text{ y } p \notin \prod_{x \in X} Pre_x(t) \\ 0 \text{ si } p \in \prod_{x \in X} Post_x(t) \Leftrightarrow p \in \prod_{x \in X} Pre_x(t) \\ -1 \text{ si } p \notin \prod_{x \in X} Post_x(t) \text{ y } p \in \prod_{x \in X} Pre_x(t) \end{cases}$$

Además, $(\prod_{x \in X} Pre_x(t))(p) = (F(p,t) \in X)$ y $(\prod_{x \in X} Post_x(t))(p) = (F(t,p) \in X)$. Para concluir, basta tener en cuenta que $\delta^p_X = (F(t,p) \in X) - (F(p,t) \in X)$, pues en ese caso $M'_a(p) = M'(p)(a) = M(p)(a) + \delta^p_X = (\prod_{x \in X} Pre_x(t))(p) + Q(p) + \delta^p_X = (F(p,t) \in X) + Q(p) + \delta^p_X = Q(p) + (F(t,p) \in X) = (\prod_{x \in X} Post_x(t))(p) + Q(p)$, como queríamos demostrar.

□

Teorema 10.1.19 $[M_1]_{\equiv_\alpha}[t([\sigma]_{\approx_t})\rangle[M_2]_{\equiv_\alpha} \Leftrightarrow NF([M_1]_{\equiv_\alpha}) \stackrel{rl(t(\sigma))}{\longrightarrow} NF([M_2]_{\equiv_\alpha})$.

Demostración.

$\Rightarrow$ Veamos que si $[M_1]_{\equiv_\alpha}[t([\sigma]_{\approx_t})\rangle[M_2]_{\equiv_\alpha}$ entonces $NF([M_1]_{\equiv_\alpha}) \stackrel{rl(t(\sigma))}{\longrightarrow} NF([M_2]_{\equiv_\alpha})$. Elijamos representantes que verifiquen $M_1[t(\sigma)\rangle M_2$. Probamos la tesis por inducción sobre el tamaño de $S(M_1)$. Si existen M, M_1' y M_2' tales que $M_i = M + M_i'$ y $S(M) \cap S(M_i') = \emptyset$, para $i = 1, 2$ (lo que implica que $[M]_{\equiv_\alpha} + [M_i']_{\equiv_\alpha} = [M + M_i']_{\equiv_\alpha}$), y $M_1'[t(\sigma)\rangle M_2'$ entonces, por hipótesis de inducción $NF([M_1']_{\equiv_\alpha}) \stackrel{rl(t(\sigma))}{\longrightarrow} NF([M_2']_{\equiv_\alpha})$. Aplicando entonces la regla de congruencia se puede probar que $NF([M]_{\equiv_\alpha}) + NF([M_1']_{\equiv_\alpha}) \stackrel{rl(t(\sigma))}{\longrightarrow} NF([M]_{\equiv_\alpha}) + NF([M_2']_{\equiv_\alpha})$ lo que al ser NF homomorfismo se puede reducir, aplicando la observación anterior, a $NF([M_1]_{\equiv_\alpha}) \stackrel{rl(t(\sigma))}{\longrightarrow} NF([M_2]_{\equiv_\alpha})$.

El caso base es aquel en el que $S(M_1) = \sigma(Var(t))$, que es simplemente $\sigma(pre(t))$, y las APNs siempre satisfacen que $post(t) \subseteq pre(t)$. Para cada $\eta \in S(M_1)$ y todo p tal que $\sigma(F(p,t)) = \eta$ se cumple que $p \in (M_1)_\eta$. Entonces, si $X \in \mathcal{P}(t(\sigma))$ es tal que $\sigma(X) = \eta_X$, tenemos que $\prod_{x \in X} Pre_x(t) \subseteq (M_1)_{\eta_X}$. Por tanto existe $Q(X) \in \mathcal{MS}(P)$ tal que $(M_1)_{\eta_X} = \big(\prod_{x \in X} Pre_x(t)\big)Q(X)$. Por el lema 10.1.18 sabemos que $(M_2)_{\eta_X} = \big(\prod_{x \in X} Post_x(t)\big)Q(X)$. Entonces se puede probar la tesis aplicando la regla de reemplazamiento al nivel superior de $\mathcal{A}_{M_1}$ usando la regla $rl(t(\sigma))$.

$\Leftarrow$ Lo probamos por inducción sobre la derivación de $NF([M_1]_{\equiv_\alpha}) \stackrel{rl(t(\sigma))}{\longrightarrow} NF([M_2]_{\equiv_\alpha})$. La regla reflexiva no ha podido ser aplicada al nivel superior. El caso de la transitividad es trivial por el lema anterior y la hipótesis de inducción. Supongamos que se ha aplicado la regla de congruencia, es decir, $NF([M_1]_{\equiv_\alpha}) = A + C$, $NF([M_2]_{\equiv_\alpha}) = B + C$ y $A \stackrel{rl(t(\sigma))}{\longrightarrow} B$. Como NF es un isomorfismo existen $[M_1']_{\equiv_\alpha}$, $[M_2']_{\equiv_\alpha}$ y $[M'']_{\equiv_\alpha}$ tales que $NF([M_1']_{\equiv_\alpha}) = A$, $NF([M_2']_{\equiv_\alpha}) = B$, $NF([M'']_{\equiv_\alpha}) = C$, $[M_1]_{\equiv_\alpha} = [M_1']_{\equiv_\alpha} + [M']_{\equiv_\alpha}$ y $[M_2]_{\equiv_\alpha} = [M_2']_{\equiv_\alpha} + [M']_{\equiv_\alpha}$. La hipótesis de inducción nos dice que $[M_1']_{\equiv_\alpha}[t([\sigma]_{\approx_t})\rangle[M_2']_{\equiv_\alpha}$. Por monotonía, se verifica que $[M_1]_{\equiv_\alpha} = [M_1']_{\equiv_\alpha} + [M']_{\equiv_\alpha}[t([\sigma]_{\approx_t})\rangle[M_2']_{\equiv_\alpha} + [M']_{\equiv_\alpha} = [M_2]_{\equiv_\alpha}$.

Finalmente, supongamos que se ha usado la regla de reemplazamiento al nivel superior con la regla $l_{t(\sigma)} \to r_{t(\sigma)}$. Entonces $NF([M_1]_{\equiv_\alpha}) = \sum_{X \in \mathcal{P}(t(\sigma))} \big(\prod_{x \in X} Pre_x(t)\big)Q(X)$, $NF([M_2]_{\equiv_\alpha}) = \sum_{X \in \mathcal{P}(t(\sigma))} \big(\prod_{x \in X} Post_x(t)\big)R(X)$ y $R(X) \to Q(X)$ para todo $X \in \mathcal{P}(t(\sigma))$ sin reemplazamiento, de modo que por el lema 10.1.17, $R(X) = Q(X)$ para todo $X \in \mathcal{P}(t(\sigma))$. Sea $\{\eta_X \mid X \in \mathcal{P}(t(\sigma))\}$ el conjunto de identificadores distintos dos a dos tales que $(M_1)_{\eta_X} = \big(Pre_x(t)\big)Q(X)$ y definamos $\bar{\sigma}$ tal que $\bar{\sigma}(X) = \{\eta_X\}$. En primer lugar, se cumple que

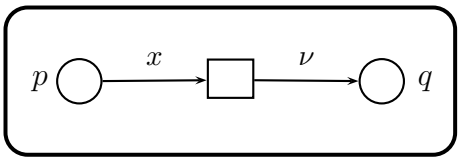

Figura 10.2: Una ν-APN

$\bar{\sigma} \in [\sigma]_{\approx_t}$. Veamos que $t(\bar{\sigma})$ está activada en M_1, es decir, dado p veamos que $\bar{\sigma}(F(p,t)) \in M_1(p)$. Por definición de $\mathcal{P}(t(\bar{\sigma}))$ y por el lema 10.1.14 existe $X \in \mathcal{P}(t(\sigma))$ tal que $F(p,t) \in X$. Entonces $p \in Pre_{F(p,t)}(t) \subseteq \big(\prod_{x\in X} Pre_x(t)\big)Q(X) = (M_1)_{\eta_X}$. Esto implica que $p \in (M_1)_{\eta_X}$ y, por tanto, $\bar{\sigma}(F(p,t)) = \eta_X \in M_1(p)$. Entonces, la transición está activada con modo $\bar{\sigma}$, de manera que se puede disparar para producir un marcaje M_2'. Queda por ver que $M_2 \equiv_\alpha M_2'$, para lo que es suficiente ver que $\mathcal{A}_{M_2'} = \mathcal{A}_{M_2}$. En primer lugar, $\mathcal{A}_{M_2'} = \sum_{a\in\{\eta_X \mid X\in\mathcal{P}(t(\sigma))\}} (M_2')_{\eta_X} = \sum_{X\in\mathcal{P}(t(\sigma))} (M_2')_{\eta_X}$. Por el lema anterior, como $(M_1)_{\eta_X} = \big(\prod_{x\in X} Pre_x(t)\big)Q(X)$ se cumple que $(M_2)_{\eta_X} = \big(\prod_{x\in X} Post_x(t)\big)Q(X)$, de lo que se sigue la tesis.

□

Veamos ahora cómo podemos representar las ν-APNs en Maude, es decir, cómo podemos extender lo anterior a APNs en las que también se usa la variable especial ν. En primer lugar, vamos a restringir la definición de *modo* de una transición.

Vamos a restringir los modos de manera que $\sigma(\nu) \neq \sigma(x)$ para todo $x \in Var(t)$. La única diferencia con los modos que teníamos hasta ahora es que hemos prohibido que los modos instancien la variable ν al mismo valor que ninguna otra variable, ya que ninguna transición podrá estar activada en ningún modo que no cumpla esa restricción. Entonces todo modo de una transición con variables ν produce una clase de equivalencia formada únicamente por la variable ν en $Var(t)/_{\sim_\sigma}$.

Cuando representemos a los marcajes en $\mathcal{MS}(\mathcal{MS}(P))$ se puede implementar el mecanismo de creación de nombres descrito de una manera local, sin necesidad de explorar todo el marcaje para garantizar que el nombre es en efecto nuevo, y sin necesidad de una variable global, ya que el nuevo nombre se corresponderá simplemente con un nuevo multiconjunto, que en el momento de la creación del nombre contendrá solamente el lugar o los lugares donde este ha sido creado.

Todos los resultados anteriores acerca de la representación de una APN por medio de una teoría de reescritura son válidos, siempre que extendamos las inyecciones FV a $FV : \mathcal{P}(t(\sigma)) \to Var \cup \{\emptyset\}$, con $FV(\{\nu\}) = \emptyset$. Consideremos por

ejemplo la red de la figura 10.2, que toma un token de p y produce un identificador nuevo en q. Como los modos no pueden instanciar ν y x al mismo valor, la única clase de equivalencia en $Modes(t)/_{\approx_t}$ es $\mathcal{P} = \{\{x\}, \{\nu\}\}$, es decir, aquélla en la que x y ν no están relacionadas. Por tanto, según la definición anterior, la única regla producida por esa red es

$$Pre_x(t)FV(\{x\}) + Pre_\nu(t)FV(\{\nu\}) \longrightarrow Post_x(t)FV(\{x\}) + Post_\nu(t)FV(\{\nu\})$$

Si tomamos $FV(\{x\}) = M$, como necesariamente ha de tenerse $FV(\{\nu\}) = \emptyset$, la regla resultante es simplemente $p\ M \to M + q$, de modo que se garantiza que el nombre que aparece en q es en efecto nuevo.

Proposición 10.1.20 $[M_1]_{\equiv_\alpha}[t([\sigma]_{\approx_t})\rangle[M_2]_{\equiv_\alpha} \Leftrightarrow NF([M_1]_{\equiv_\alpha}) \stackrel{rl(t(\sigma))}{\longrightarrow} NF([M_2]_{\equiv_\alpha})$.

Demostración. Supongamos que $M_1[t(\sigma)\rangle M_2$. Si borramos los arcos adyacentes a t etiquetados con ν entonces ese disparo se puede ver como el disparo de una APN. Sea σ^* la restricción de σ a todo su dominio excepto ν, y M_2^* la restricción de M_2 a $\{p \mid \sigma(F(t,p)) \neq \nu\}$. Por la proposición 10.1.19 se tiene que $M_1[t(\sigma^*)\rangle M_2^* \Leftrightarrow NF(M_1) \stackrel{rl(t(\sigma^*))}{\longrightarrow} NF(M_2)$. Para obtener la tesis, basta tener en cuenta que $pre_\nu(t) = \emptyset$, $FV(\{\nu\}) = \emptyset$ y $NF(M_2) = NF(M_2^*) + post_\nu(t)$.

□

10.2. ν-APNs en Maude

Hemos visto una traducción canónica de la semántica operacional de las ν-APNs a teorías de reescritura. Estas teorías pueden ser expresadas de manera inmediata en Maude. En principio, las ν-APNs sólo tienen un color para identificadores, y por tanto no existe el token negro como token distinguido, pero el mismo puede ser fácilmente simulado utilizando un determinado token identificador. Sin embargo, por conveniencia y eficiencia en la implementación en Maude de las ν-APNs vamos a considerar explícitamente tokens negros.

Respecto a los tokens negros podemos hacer exactamente igual que como se hacía en [SMÖ01a], es decir, identificar un marcaje (restringiéndonos a estos tokens) con el multiconjunto de los lugares que contienen dichos tokens. Al efecto, necesitaremos un tipo `PlaceB` para los lugares y un tipo `MSPlacesB` para los multiconjuntos de lugares.

```
sort PlaceB MSPlacesB .
subsort PlaceB < MSPlacesB .
```

Respecto a los lugares que contienen identificadores, procedemos como hemos descrito en la sección anterior, de manera que necesitaremos no sólo tipos para los lugares y los multiconjuntos de lugares, sino también para los multiconjuntos de multiconjuntos de lugares.

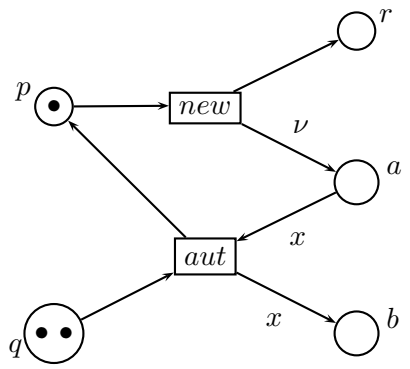

Figura 10.3: Ejemplo de ν-APN

```
sort PlaceC MSPlacesC MS2PlacesC .
subsort PlaceC < MSPlacesC < MS2PlacesC .
```

A continuación mostramos los constructores para los multiconjuntos de lugares ordinarios, multiconjuntos de lugares coloreados y multiconjuntos de multiconjuntos de lugares coloreados, respectivamente.

```
op _,,_ : MSPlacesB  MSPlacesB  -> MSPlacesB  [comm assoc id: emptySetB] .
op  __  : MSPlacesC  MSPlacesC  -> MSPlacesC  [comm assoc id: emptySetC] .
op _+_  : MS2PlacesC MS2PlacesC -> MS2PlacesC [comm assoc id: emptySetC] .
```

Finalmente, un marcaje no es más que un par que tiene como primera componente el marcaje restringido a lugares ordinarios y como segunda componente el marcaje restringido a lugares coloreados.

```
op _++_ : MSPlacesB MS2PlacesC -> Marking .
```

Entonces, una ν-APN es simplemente un módulo de sistema de Maude que reescribe marcajes, como se vio antes. Por ejemplo, la red de la figura 10.3 da lugar a las siguientes reglas:

```
rl [new] : p ,, P ++ C       => r ,, P ++ C + a .
rl [aut] : q ,, P ++ a A + C => p ,, P ++ b A + C .
```

La imagen por NF del marcaje inicial mostrado en el ejemplo se representa en Maude mediante el término `p ,, q ,, q ++ emptySetC`.

```
rew p ,, q ,, q ++ emptySetC .
result: Marking
        r ,, r ,, r ++ a + b + b
```

que se corresponde con el marcaje con tres tokens negros en el lugar `r` y tres tokens coloreados distintos, uno en el lugar `a` y los otros dos en el lugar `b`.

10.3. De las MSPNs a las ν-APNs

Las ν-APNs son una clase particular de sistemas MSPN con una sola componente, sin lugares localidad ni transiciones de movimiento o de sincronización. Además, como vimos en el capítulo 5, las ν-APNs pueden simular paso a paso, y de manera monótona, cualquier sistema MSPN. La demostración consistía básicamente en añadirle a cada red un lugar extra que, en cada momento, contiene un sólo token identificador, que caracteriza la localidad en la que se encuentra la red, obligando a que los tokens en los respectivos lugares de cada componente coincidan en las sincronizaciones, para que estas sólo tengan lugar cuando las redes están co-localizadas. Además, cada par de transiciones compatibles se simula por medio de una sola transición.

Ciertamente, la construcción anterior puede resultar bastante enrevesada, incluso para sistemas de tamaño medio. Por tanto, la representación de un sistema MSPN mediante el módulo de sistema de Maude que se corresponde con la traducción a una ν-APN puede ser también bastante complicada y distante del sistema original. Para automatizar el proceso de traducción, definimos una signatura para sistemas MSPN, de manera que puedan ser representados como términos Maude de tipo `NetSystem`. Así podemos aprovecharnos de las propiedades reflexivas que ofrece Maude para realizar automáticamente la traducción por medio de una función

```
op moduleOf : NetSystem -> SModule .
```

que devuelve la metarepresentación del módulo de sistema de Maude que representa la traducción a una ν-APN del sistema MSPN dado. Empecemos describiendo dicha signatura para sistemas MSPN. En primer lugar, añadimos dos tipos para lugares localidad e identificadores, que son ambos lugares coloreados, es decir, subtipos del tipo `PlaceC`.

```
sorts PlaceL PlaceId .
subsorts PlaceL PlaceId < PlaceC .
```

Los constructores para cada tipo de lugar son:

```
op `(_,_`) : PId Colour -> Place .
var pid : PId .
mb (pid,black)      : PlaceB .
mb (pid,locality)   : PlaceL .
mb (pid,identifier) : PlaceId .
```

donde `Pid` es un tipo para nombre de lugares y `Colour` es un tipo que contiene sólo las constantes `black`, `locality` y `identifier`.

También añadimos tipos para las transiciones, que pueden ser autónomas, de sincronización, de movimiento o de creación de nombres.

```
sorts Transition TransitionAut TransitionSync TransitionGo TransitionNew .
subsorts TransitionAut TransitionSync TransitionGo TransitionNew < Transition .
```

Las transiciones se etiquetan mediante una función λ con etiquetas en $\mathcal{A} \cup Sync$ con $go, new \in \mathcal{A}$. Usaremos el tipo `Go` para go, `New` para new, `Aut` para el resto de las etiquetas de $\mathcal{A}$ y `Sync` para etiquetas de sincronización, que pueden ser de dos formas, como se ve en la definición de los siguientes constructores:

```
op _? : Services -> Sync .
op _! : Services -> Sync .
```

donde `Services` es un tipo para nombres de servicios. De esta manera, la función de conjugación de etiquetas de sincronización se define trivialmente:

```
op ^_ : Sync -> Sync .
eq ^(s ?) = s ! .
eq ^(s !) = s ? .
```

Los siguientes constructores son los de las transiciones:

```
op [_,_] : TId Aut -> TransitionAut .
op [_,_] : TId Go -> TransitionGo .
op [_,_] : TId New -> TransitionNew .
op [_,_] : TId Sync -> TransitionSync .
```

donde, una vez más, `TId` es un tipo para los nombres de las transiciones. Suponiendo definidas funciones `pre` y `post` que calculan los respectivos conjuntos de variables, el predicado de compatibilidad de transiciones de sincronización es inmediato:

```
op comp : Transition Transition NetSystem -> Bool .
eq comp ([tid1,s1],[tid2,s2],NS) = (s1 == (^ s2)) and
        post([tid1,s1],[tid2,s2],NS) << pre([tid1,s1],[tid2,s2],NS) .
eq comp ( t1 , t2 , NS ) = false [owise] .
```

Podemos representar una MSPN como un conjunto de arcos entre lugares y transiciones y viceversa. Para los arcos introducimos el tipo `Flow`.

```
op _-_->_ : PlaceB VarEps Transition -> Flow .
op _-_->_ : PlaceL VarL Transition   -> Flow .
op _-_->_ : PlaceId VarId Transition -> Flow .

op _-_->_ : Transition VarEps PlaceB  -> Flow .
op _-_->_ : Transition VarL PlaceL    -> Flow .
op _-_->_ : Transition VarId PlaceId  -> Flow .
```

En las declaraciones anteriores hemos usado los tipos `VarEps`, `VarL` y `VarId` para variables en $Var_{\bullet}$, $Var_{\mathcal{L}}$ y Var_{Id}, respectivamente. En particular, `VarEps` sólo contiene una constante `eps`, que representa a la variable ε usada en arcos adyacentes a lugares que contienen tokens normales.

Ahora sí, una MSPN es un conjunto de arcos y de lugares (estos últimos para tener en cuenta la existencia de lugares aislados, mientras que no permitimos la existencia de transiciones aisladas) y un sistema MSPN es un conjunto de componentes MSPN.

```
subsorts Place Flow < Net < NetSystem .
op _,_ : Net Net -> Net [assoc comm id: emptyNet] .
op _-_ : NetSystem NetSystem -> NetSystem [assoc comm id: emptyNet] .
```

Por ejemplo, podemos definir las constantes:

```
ops N1 N2 : -> Net .
op S : -> NetSystem .

eq N1 = ((pid1,black) - eps -> [t1id , s ?]) ,
        ((rid1,black) - eps -> [t2id , s !]) ,
        ( [t1id,s ?]  - eps -> (qid1,black)) ,
        ( [t2id,s !]  - eps -> (sid1,black)) .

eq N2 = ((pid2,locality) - xL -> [t3id,s ?]) ,
        ((rid2,black)   - eps -> [t4id,s !])   ,
        (  [t3id,s ?]   - eps -> (qid2,black)) ,
        ([t4id,s !]  - xL -> (sid2,locality)) .

eq S = N1 - N2 .
```

Ahora que podemos representar los sistemas MSPN como términos de Maude, definimos una operación que, dado un término de tipo `NetSystem` nos devuelva el módulo de sistema asociado. Como dijimos antes, hay una regla distinta por cada transición y clase de equivalencia en $Modes(t)/_{\approx_t}$. Para calcular esas clases de equivalencia hemos definido la función

```
op Parts : VarSet -> VarPartSet .
```

que, dado un conjunto de variables, $Var(t)$, devuelve los elementos de $Modes(t)/_{\approx_t}$. Cada elemento se representa como un término de tipo `VarPart`:

```
subsorts VarEps VarL VarId < Var < VarSet < VarPart < VarPartSet .
op _;_ : VarPart VarPart -> VarPart [assoc comm] .
op _;;_: VarPartSet VarPartSet -> VarPartSet [assoc comm] .
```

Por ejemplo, el resultado de reducir el término `Parts(x & y)` es el término `xy ;; x ; y`. El primer elemento, `xy`, representa la clase de equivalencia en la que `x` e `y` están relacionadas, mientras que `x ; y` representa la clase en la que no lo están.

Entonces, con ayuda de esa función definimos:

```
op moduleOf : NetSystem -> SModule .

eq moduleOf(NS) =
(mod 'MSPN is
    including 'NETS .
    including 'META-LEVEL .
    sorts none .
    none none none none
    rules(NS)
endm) .
```

Así, la traducción es un módulo llamado `MSPN`, que importa el módulo `NETS` y el metanivel, sin declaraciones de tipos, ni de relaciones de subtipados, ni pertenencias, y cuyas reglas están especificadas por `rules(NS)`. Esa operación se aplica homomórficamente a los conjuntos de transiciones y modos, de manera que, al final, va a computar `rules(NS,t,part)`. No mostramos los detalles aquí (véase el apéndice), pero sí decir que su definición sigue en efecto los pasos mostrados en la sección anterior*.

Por ejemplo, si computamos `moduleOf(S)` el resultado es

```
rl (pid1,black),,(rid1,black),,P:MSPlacesB ++ I:MS2PlacesC =>
   (qid1,black),,(sid1,black),,P:MSPlacesB ++ I:MS2PlacesC .

rl (rid1,black),,P ++ (pid2,locality) V:MSPlacesC + I:MS2PlacesC =>
   (sid1,black),,P ++ (qid2,locality) V:MSPlacesC + I:MS2PlacesC .

rl (rid2,black),,P ++ (pid2,locality) V:MSPlacesC + I:MS2PlacesC =>
   (qid2,black),,P ++ (sid2,locality) V:MSPlacesC + I:MS2PlacesC .
```

Ahora podemos usar el módulo resultante para analizar el comportamiento de S. Por ejemplo, podemos ejecutar S desde un marcaje inicial:

```
red metaRewrite
  (moduleOf(Net6),upTerm((rid1,black) ++ (pid2,locality)),unbounded) .
```

o preguntar si un marcaje es alcanzable:

```
red metaSearch(moduleOf(Net6),upTerm((rid1,black) ++ (pid2,locality),
             upTerm((rid2,black) ++ (sid1,locality)),'*,unbounded,0) .
```

En el capítulo 6 probamos que la alcanzabilidad es indecidible para sistemas MSPNs con identificadores abstractos, de manera que el método anterior (y cualquier otro) será siempre incompleto, es decir, puede no terminar si la búsqueda es no acotada.

*De hecho, maneja dos particiones, ya que trabaja directamente con los sistemas MSPN, de manera que se puede distinguir entre variables para identificadores y para localidades.

10.4. Algoritmo de decisión para el problema del recubrimiento

Aunque vimos que la alcanzabilidad es indecidible para sistemas MSPNs, también vimos que el recubrimiento sí que es decidible. El procedimiento de decisión usado allí era el siguiente análisis hacia atrás:

$\mathcal{M} \leftarrow \mathcal{C}(M_1)$
while $Pre(\mathcal{M}) \not\subseteq \mathcal{M}$ **do**
  $\mathcal{M} \leftarrow \mathcal{M} \cup Pre(\mathcal{M})$
end while
devolver $M_2 \in \mathcal{M}$

El objetivo de ese procedimiento es el de computar $I_0 = \mathcal{C}(M_1)$, $I_{i+1} = I_i \cup Pre(I_i)$. Por ser nuestras redes un ejemplo de sistema bien estructurado, ese procedimiento se puede transformar en un algoritmo efectivo. En particular, podemos usar $J_i = min(I_i)$ como representación finita de I_i, que se puede computar como

$$J_{i+1} = min(J_i \cup \bigcup_{M \in J_i} min(pre(\mathcal{C}(M)))).$$

En nuestro caso, como vimos en el capítulo 6, podemos computar $min(Pre_{t,\sigma}(\mathcal{C}(M)))$ como $Pre_{t,\sigma}(M \sqcup \sigma(F(t,-)))$. En nuestra representación de sistemas, cada regla se corresponde con un par $t(\sigma)$. Por tanto, la función

```
op needed : Marking -> Marking .
```

será responsable de computar $\sigma(F(t,-))$, donde el marcaje que aparece como argumento será la parte derecha de la regla correspondiente. De hecho, como nuestro análisis es hacia atrás, trabajaremos con el sistema en el que hemos invertido todas las reglas aplicando la función

```
op reverse : SModule -> SModule .
eq reverse( mod A is B sorts C . D E F G Rules endm ) =
   mod reverse(A) is B sorts C . D E F G reverseRules(Rules) endm .

op reverseRules : RuleSet -> RuleSet .
eq reverseRules ( rl T1 => T2 [ AS ] . ) = ( rl T2 => T1 [ AS ] . ) .
```

Definimos la función `Pre` que toma un marcaje `M` y (la metarepresentación de) la regla `Rl` y devuelve el conjunto de precedesores de `M` correspondientes a la aplicación de la regla `Rl`, y la extendemos a conjuntos de marcajes y de reglas, elemento a elemento:

```
op Pre : Marking Rule SModule -> MarkingSet .
eq Pre ( M ,  rl T1 => T2 [label( I )] . , netModule ) =
   PreV ( M U needed(T1) , netModule ) .
```

La definición de `PreV` aprovecha las posibilidades reflexivas que ofrece Maude para el cómputo de los predecesores de `M U needed(T1)`, que sabemos que existen, como

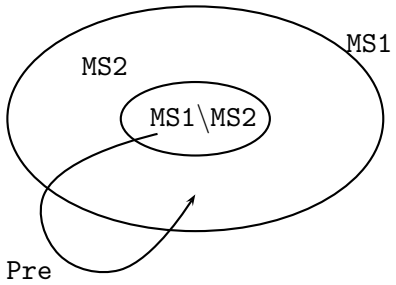

Figura 10.4: Invariante de `cover`

probamos en el capítulo 6. Más concretamente, usa la operación `metaRewrite`, definida en el módulo `META-LEVEL`, para reescribir la metarepresentación de ese marcaje en el módulo metarepresentado por `netModule`.

Para manejar el operador *min* más externo basta añadir la siguiente ecuación definida sobre conjuntos de marcajes:

```
ceq M1 * M2 = M1 if M1 [= M2 .
```

donde `*` es el constructor para conjuntos de marcajes y `[=` es el orden natural entre marcajes.

Finalmente, el algoritmo de recubrimiento es el siguiente:

```
op coveri : Marking MarkingSet MarkingSet SModule -> Bool .
ceq coveri(M1,MS1,MS2,netModule) = M1 =< MS or-else
                      if MS =< MS1 then false
                      else coveri( M1 , MS * MS1 , MS , netModule) fi
                      if MS := Pre(MS2,netModule) .

op cover : Marking Marking SModule -> Bool .
eq cover ( M1 , M2 , netModule ) = M2 [= M1 or-else
                           coveri ( M1 , M2 , M2 , reverse(netModule) ) .
```

con invariante `MS2`$\subseteq$`MS1`,Pre(`MS1`$\backslash$`MS2`) $\subseteq$`MS1` (véase la figura 10.4).

10.5. Arquitectura de la implementación

En la sección anterior hemos definido una signatura para sistemas MSPN, de manera que podemos representar un sistema MSPN como un término Maude, más manejable que su representación como módulo de sistema. Sin embargo, las redes de Petri deben gran parte de su utilidad práctica a su representación gráfica, que las hace amenas y fácilmente comprensibles. La representación de un sistema MSPN como término Maude, aunque más manejable, sigue siendo una representación simbólica, no gráfica. Para evitar al usuario el tormento de tener que introducir un sistema manualmente, le permitiremos usar alguna de las muchas herramientas de edición de redes de Petri existentes. En concreto, nosotros hemos elegido CPN Tools [RWL$^+$03], por ser probablemente la herramienta más usada en la comunidad de redes de Petri para la edición y análisis de las mismas. El usuario de

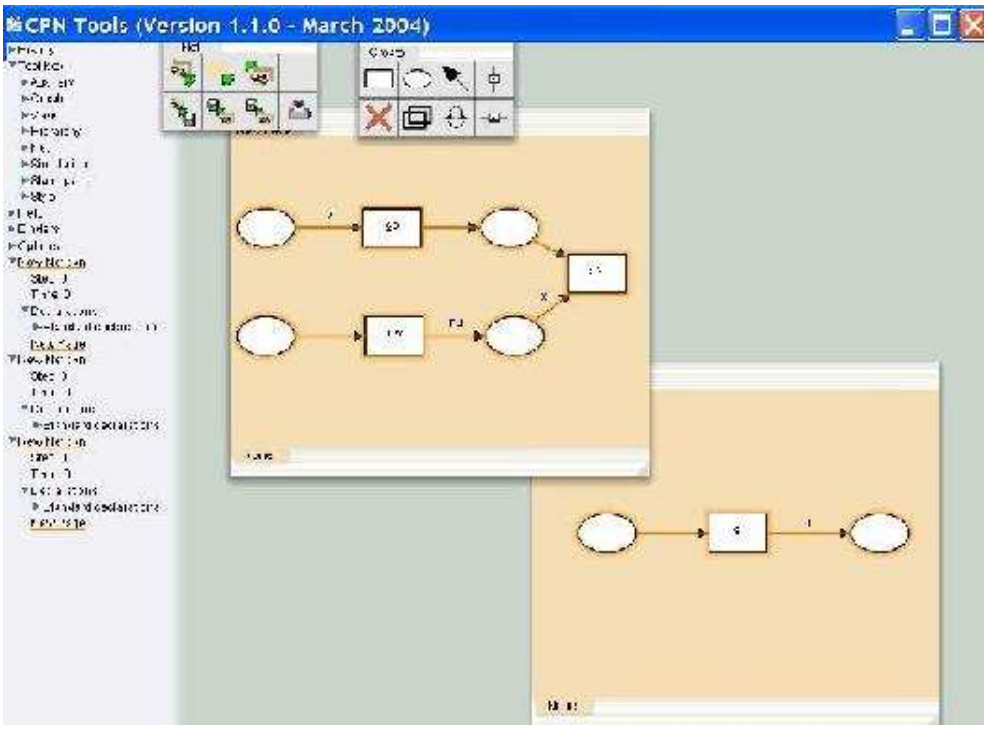

Figura 10.5: CPN Tools

CPN Tools puede dibujar de manera cómoda redes de Petri coloreadas. En la figura 10.5 se muestra el interfaz para la edición de redes de Petri de CPN Tools. Esas redes se almacenan en un formato especial llamado CPNML, que no es más que una definición de tipo de documentos XML. Hemos usado el traductor de XML a Maude escrito por Steven Eker como un script Python. El fichero traducido contiene un módulo `TRANSLATION` que define una constante `translation` que se reduce a un término de tipo `Element` sobre una signatura para XML en Maude. Entonces, nos basta con definir la correspondencia entre esas traducciones y nuestras redes:

```
op translate : Element -> NetSystem .

op translatedNet : -> NetSystem .
eq translatedNet = translate(translation) .
```

Por tanto, el término `moduleOf(translatedNet)` es la metarepresentación del módulo de sistema que se corresponde con el sistema MSPN dibujado en CPN Tools. Ese término lo podemos usar, por ejemplo, como entrada del algoritmo de decisión para el recubrimiento visto en la sección anterior.

Veamos algunos ejemplos de uso de la herramienta para el sistema MSPN de la figura 10.6, que es de hecho el sistema que aparece dibujado en CPN Tools en la figura 10.5. Para ilustrar mejor los resultados, mostramos también su traducción a la ν-APN de la

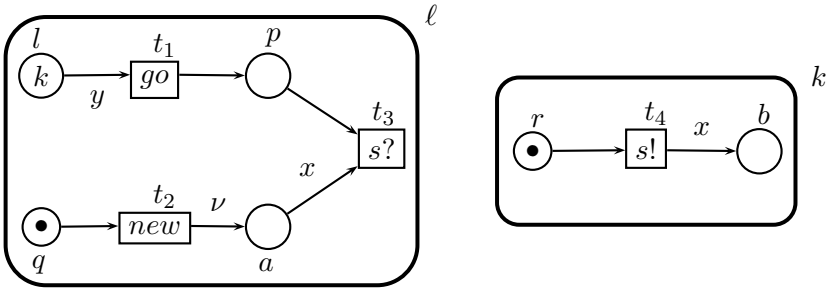

Figura 10.6: Sistema MSPN

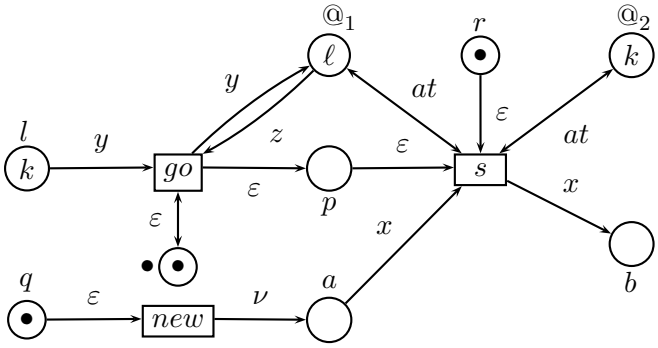

Figura 10.7: ν-APN que simula al sistema MSPN de la figura 10.6

figura 10.7 según la proposición 5.2.3. Para mostrar el conjunto de reglas que definen un sistema hemos definido el operador

```
op showRules : SModule -> CRegla .
```

que también puede usarse como herramienta de depuración. Veamos qué reglas obtenemos en el ejemplo:**

```
Maude> red showRules(moduleOf(translatedNet)) .

reduce in XML2NETS : showRules(moduleOf(translatedNet)) .
rewrites: 16581 in 101ms cpu (214ms real) (162582 rewrites/second)
result CRegla:

[P:MSPlacesB ++ VF4:MSPlacesC + II:MS2PlacesC + VF2:MSPlacesC @1 l =>
 P:MSPlacesB,,p ++ VF4:MSPlacesC + II:MS2PlacesC + VF2:MSPlacesC @1]

[P:MSPlacesB   ++
 VF4:MSPlacesC + II:MS2PlacesC + VF2:MSPlacesC @1 + VF3:MSPlacesC l =>
P:MSPlacesB,,p ++
 VF2:MSPlacesC + VF4:MSPlacesC + II:MS2PlacesC + VF3:MSPlacesC @1]
```

**Las reglas obtenidas han sido maquilladas manualmente para su mejor comprensión cambiando el nombre de los lugares, que la herramienta genera automáticamente, para que coincidan con los de la figura.

```
[P:MSPlacesB,,q ++ VF2:MSPlacesC + II:MS2PlacesC =>
 P:MSPlacesB    ++ VF2:MSPlacesC + II:MS2PlacesC + a]

[P:MSPlacesB,,p,,r ++
 II:MS2PlacesC + VF3:MSPlacesC a + VF2:MSPlacesC @1 @2 =>
 P:MSPlacesB ++
 II:MS2PlacesC + VF3:MSPlacesC b + VF2:MSPlacesC @1 @2]
```

La primera regla representa el disparo de la transición de movimiento cuando el origen y el destino coinciden. La segunda regla representa también al movimiento de la red, pero ahora cuando de verdad hay tal movimiento, es decir, cuando el origen y el destino no coinciden (en la figura, cuando $k \neq \ell$). La tercera regla representa la creación de identificadores en el lugar a por parte de la transición etiquetada por *new*. El nuevo identificador queda caracterizado por el nuevo multiconjunto que contiene únicamente a ese lugar. Por último, la quinta regla se corresponde con el disparo del par de sincronización etiquetado por $(s?, s!)$. El hecho de que ambas redes tengan que estar en la misma localidad queda reflejado en la condición de que $@_1$ y $@_2$ tengan que contener al mismo token localidad, es decir, `@1` y `@2` han de pertenecer al mismo multiconjunto.

El marcaje inicial que aparece en la figura 10.6 es el siguiente:

```
q ,, r ++ l @2 + @1
```

En lugar de tener que usar directamente las operaciones de `META-LEVEL` hemos definido una operación `fire` que reescribe su primer argumento según el módulo que recibe como segundo argumento. Podemos usar esta función para reescribir el término que representa al marcaje inicial de nuestro ejemplo:

```
Maude> red fire( q ,, r ++ l @2 + @1 , moduleOf(translatedNet)) .

reduce in XML2NETS : fire(q,,r ++ @1 + @2 l,moduleOf(translatedNet)) .

rewrites: 16545 in 141ms cpu (354ms real) (116532 rewrites/second)
result Marking: emptySetB ++ @1 @2 + b
```

Así que el resultado es el marcaje en el que ambas redes están en la misma localidad y en el que hay un único token, que es de tipo identificador y se encuentra en el lugar b.

El término `emptySetB ++ b` representa el marcaje con un identificador en el lugar b y con el resto de sus lugares vacíos. Nos preguntamos si se puede cubrir este marcaje desde el marcaje inicial.

```
Maude> red cover( ( q ,, r ++ l @2 + @1 ) , (emptySetB ++ b) ,
moduleOf(translatedNet) ) .

reduce in XML2NETS : cover(q,,r ++ @1 + @2 l, emptySetB ++ b,
moduleOf(translatedNet)) .

rewrites: 14019341 in 117526ms cpu (125882ms real) (119287 rewrites/second)
result Bool: true
```

El resultado es afirmativo, como cabía esperar, pues el marcaje `emptySetB ++ @1 @2 + b` que obtuvimos al reescribir el marcaje inicial es mayor que `emptySetB ++ b`. ¿Qué ocurre si preguntamos si el marcaje `emptySetB ++ a a` con dos identificadores iguales en el lugar a se puede cubrir desde el marcaje inicial? Pues bien, en ese caso la respuesta es negativa.

```
Maude> red cover( ( q ,, r ++ l @2 + @1 ) , (emptySetB ++ a a ) ,
moduleOf(translatedNet) ) .

reduce in XML2NETS : cover(q,,r ++ @1 + @2 l, emptySetB ++ a a,
moduleOf(translatedNet)) .

rewrites: 19714 in 173ms cpu (332ms real) (113315 rewrites/second)
result Bool: false
```

Capítulo 11

Asociación segura transitoria

Como ya se ha mencionado anteriormente, la computación ubicua afecta a las hipótesis de trabajo tradicionales, en particular a las relacionadas con la seguridad de los sistemas. La autenticación es probablemente el principal problema de seguridad para los sistemas ubicuos, por tratarse de una precondición para otras propiedades como pueden ser la confidencialidad o la integridad. En el nuevo contexto, debido a la naturaleza no fiable de las redes ad-hoc, no podemos apoyarnos en una política *siempre-online*, en la que contrastemos en tiempo real la identidad de los participantes en base a un servidor de nombres (globales) centralizado, un repositorio de claves públicas o una tercera parte de confianza.

Para solventar este problema, se han propuesto varias versiones débiles de la propiedad de autenticación, que sustituyen a la autenticación basada en identidades globales. Una de esas propuestas es la de *asociación segura transitoria*, implementada por la *política del pato que resucita* [Sta02]. Esta política se basa en el hecho de que, en ocasiones, un principal puede no conocer nada acerca de un aparato (por ejemplo, cuando un usuario acaba de comprarse una PDA), y sin embargo quiere ser el único que puede usarlo (por ejemplo, por ser el primero que tiene la oportunidad de pulsar su botón de encendido). En este caso, aun cuando el principal y el aparato no tienen un conocimiento mutuo previo, pueden asociarse, pasando a compartir una relación asimétrica en la que uno es el esclavo y el otro el amo. Otras propuestas para intentar resolver el mismo problema, que no consideraremos aquí, se basan en la noción de confianza [CNS04].

En este capítulo desarrollaremos un modelo formal para la coordinación en sistemas ubicuos, con primitivas que implementen la política del pato que resucita. El modelo, que denominaremos TSA, es una ligera variante de los modelos que hemos visto en capítulos anteriores y un ejemplo de la utilidad de los distintos resultados teóricos obtenidos en ellos. En particular, en primer lugar probaremos que las especificaciones TSA se pueden ver como azúcar sintáctico de redes P/T. Además, veremos que un sistema TSA se puede simular por medio de un sistema MSPN cerrado y sin creación de nombres, que como vimos anteriormente, son equivalentes a las redes P/T. Este resultado nos da un mecanismo de más bajo nivel (el manejo de nombres puros) para implementar un sistema TSA. Por último, si queremos implementar los sistemas TSA en un marco abierto, veremos que la implementación anterior no es correcta, por lo que necesitaremos una variación suya, en la que sí usaremos la creación de nombres.

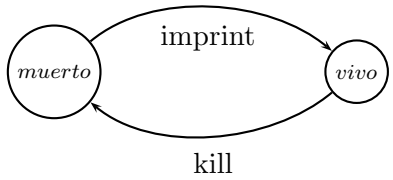

Figura 11.1: La política del pato que resucita

11.1. La política del pato que resucita

En esta sección describimos brevemente la asociación segura transitoria y el modelo de seguridad del pato que resucita. La asociación segura transitoria se basa en crear asociaciones entre componentes, que establecen relaciones amo-esclavo (típicamente, entre un controlador y un periférico). Esta asociación ha de ser segura, de manera que ningún otro principal asuma el papel de amo o de esclavo. Estas asociaciones producen una topología en forma de árboles de relaciones amo-esclavo entre componentes. Además, las asociaciones han de poder ser transitorias, en el sentido de que la topología mencionada tiene que poder cambiar a lo largo de la ejecución del sistema, siempre que así lo deseen los correspondientes amos.

La política del pato que resucita implementa el principio de la asociación segura transitoria, siguiendo la siguiente metáfora: *Un pato al nacer reconoce como su madre al primer objeto que se mueva y emita un sonido que se le presente.* A partir de ese momento, el pato recién nacido obedece a su madre hasta su muerte. De modo similar, un periférico puede reconocer como su madre a la primera entidad que le manda su clave secreta, llamada *clave de impresión*, que usará para reconocer a su amo desde ese momento. Sin embargo, para que esa asociación sea transitoria, suponemos que el pato puede morir y resucitar, reconociendo a una entidad posiblemente diferente como su nueva madre, a la que será leal en lo sucesivo.

La política se puede resumir en la siguiente lista de principios fundamentales:

1. **Principio de los dos estados.** Las entidades pueden tener dos estados: *imprimibles* (muertas) o *impresas* (vivas). Las entidades imprimibles pueden ser impresas por cualquiera, y las entidades impresas solamente obedecen a mamá pato.

2. **Principio de impresión.** El paso de imprimible a impreso se llama *impresión*, y ocurre cuando una entidad, mamá pato, envía una clave al patito (véase la figura 11.1).

3. **Principio de muerte.** El paso de impreso a imprimible se llama *muerte* (véase la figura 11.1). La causa de muerte por defecto es que la mamá pato ordene al patito que cometa seppuku, más popularmente conocido como harakiri,* aunque se contemplan otras causas de muerte, como muerte por envejecimiento o por compleción de su tarea.

4. **Principio de asesinato.** No debe ser rentable económicamente para un atacante matar a un pato (causar su muerte).

*Suicidio ritual tradicional samurai.

11.2. Sistemas TSA

Nuestro objetivo es definir un modelo para el estudio de la coordinación de sistemas ubicuos, que llamaremos TSA, que implemente la política del pato que resucita. Será una variación de las redes ubicuas que vimos en el capítulo 3, de modo que un sistema TSA será esencialmente un conjunto de redes localizadas que pueden moverse entre localidades, imprimir a otras redes, dar órdenes a las redes que han impreso previamente, matar esas redes, ser impresas y recibir órdenes de su amo cuando hayan sido impresas. Para definir nuestro modelo tendremos que especificar algunos aspectos de la política que no son del todo precisos. Supondremos que las distintas componentes de un sistema tienen un tipo, tomado de un conjunto $\mathcal{O}$. Usaremos el tipo especial $\top \in \mathcal{O}$, que es el tipo de las componentes que no necesitan haber sido impresas para llevar a cabo su actividad (por ejemplo, los usuarios). Las componentes pueden estar vivas o muertas. Si una componente es de tipo $\top$ o está impresa entonces está viva; de lo contrario está muerta. Las componentes vivas pueden imprimir componentes muertas, creando una asociación única y asimétrica entre ellas. Por asimétrica entendemos que a partir de ese momento la componente impresa pasa a ser esclava de la otra, obedeciendo todas las órdenes que le da su amo. Sin embargo, cuando ejecuta su código, la componente impresa puede a su vez imprimir otras componentes, creando una jerarquía de relaciones amo-esclavo. Por otra parte, cuando una componente mata a otra, no sólo esta última pasa a estar muerta, sino también todos sus esclavos, y así recursivamente.

Suponemos que las componentes sólo pueden ser impresas dentro de un cierto rango, que depende del método de impresión que se esté usando. Abstraemos los métodos particulares, y asumimos simplemente que la impresión se puede llevar a cabo siempre que las dos componentes estén en la misma localidad. En particular, suponemos que la comunicación de claves que se lleva a cabo cuando se imprime a una componente es segura. Además, una componente sólo puede dar órdenes a las componentes esclavo que estén en la misma localidad que ella.

Asimismo, una componente sólo puede matar a un esclavo cuando los dos están en la misma localidad. Por tanto, estamos implementando la versión por defecto del principio de muerte, en la que la causa de muerte de los patos es la orden de su madre. Finalmente, suponemos que los participantes sólo pueden interactuar entre sí por medio de las primitivas del modelo, de manera que el principio de asesinato también se cumple, ya que los atacantes no pueden causar la muerte de una componente de una manera no especificada por la política.

Usaremos de nuevo el conjunto $\mathcal{L}$ de nombres de localidades. Para $m > 0$ denotamos mediante $\mathit{Labels}(m)$ al conjunto $\mathcal{A} \cup S? \cup \bigcup_{i=1}^{m} (\{i\} \times \mathit{Labels}_{Auth})$, donde $\mathcal{A}$ contiene, entre todas, las etiquetas *go* k para cada $k \in \mathcal{L}$ y $\mathit{Labels}_{Auth} = S! \cup \{\mathit{imprint}(o) \mid o \in \mathcal{O} \setminus \{\top\}\} \cup \{\mathit{kill}\}$.

Definición 11.2.1 *Una* componente TSA *es una tupla* $N = (P_N, T_N, F_N, \lambda_N, o_N)$ *tal que* P_N *es un conjunto finito de lugares,* T_N *es un conjunto finito de transiciones,* $F_N \subseteq (P_N \times T_N) \cup (T_N \times P_N)$*,* $\lambda_N : T_N \to \mathit{Labels}(m)$ *para algún* $m > 0$ *y* $o_N \in \mathcal{O}$*. Diremos que* N *tiene tipo* o_N*, que* m *es la capacidad de* N*, y denotaremos mediante* $\mathit{cap}(N)$ *al conjunto* $\{1, \ldots, m\}$*.*

Si $\mathit{cap}(N) = \{1\}$ y $\lambda(t) = (1, \ell)$, para simplificar en ocasiones escribiremos simplemente $\lambda(t) = \ell$.

Por tanto, una componente TSA es una red de Petri tipada y etiquetada. El tipo o de una componente es simplemente una categoría sintáctica, que se puede entender intuitivamente como el tipo de aparato del que se trata (por ejemplo, una PDA, una nota electrónica,...), como el protocolo de impresión utilizado o ambos simultáneamente. La capacidad de una componente representa el máximo número de esclavos que puede tener simultáneamente.

Las etiquetas de las transiciones de la red establecen una partición en el conjunto de transiciones. Las transiciones t con $\lambda(t) \in \mathcal{A}$ son transiciones autónomas, es decir, transiciones que pueden ser ejecutadas de forma independiente, ya sea por un amo o por un esclavo. Si $\lambda(t) \in S?$ entonces la transición se usa para recibir órdenes del correspondiente amo, lo que exige que la componente esté ya impresa. La opción restante es que $\lambda(t) \in \{i\} \times \mathit{Labels}_{Auth}$. En este caso, la transición t se usa para *comunicarse* con su i-esclava. Si una red dispara t con $\lambda(t) = (i, \mathit{imprint}(o))$ entonces alguna componente de tipo o pasa a ser su i-esclava. Si $\lambda(t) = (i, s!)$ entonces el disparo de t tiene el efecto de ordenar al correspondiente esclavo que ejecute la acción $s?$. Finalmente, si $\lambda(t) = (i, \mathit{kill})$ la red está ordenando a su i-esclavo hacerse el harakiri. A continuación introducimos formalmente cada uno de estos conceptos.

Definición 11.2.2 *Un* sistema TSA $\mathcal{N}$ *es un conjunto de componentes TSA disjuntas dos a dos.*

Intuitivamente, las redes N con $o_N = \top$ son componentes que no necesitan haber sido impresas para ejecutarse.

Definición 11.2.3 *Dado un sistema TSA $\mathcal{N}$, una* función de dependencia *de $\mathcal{N}$ es una función parcial $R : \{(N, j) \mid N \in \mathcal{N}, j \in \mathit{cap}(N)\} \to \mathcal{N}$. Una función de dependencia R induce un grafo de dependencia $G(R)$ sobre el conjunto de nodos $\mathcal{N}$, y con un arco de N a N' si existe un j tal que $R(N, j) = N'$, en cuyo caso diremos que N' es un j-esclavo (o simplemente un esclavo) de N en R.*

En lo sucesivo llamaremos *árbol dirigido* a todo grafo dirigido para el que existe un nodo, al que llamaremos *raíz*, tal que para todo nodo del grafo existe un único camino de la raíz hasta él. Además, llamaremos bosque dirigido a todo conjunto de árboles dirigidos disjuntos dos a dos. Por último, si en un árbol dirigido hay un camino de un nodo s a otro s' diremos que s' es un descendiente de s. Denotaremos ι al árbol vacío.

Definición 11.2.4 *Un* marcaje *de un sistema TSA $\mathcal{N}$ es una tupla $\mathcal{M} = (M, \mathit{loc}, R)$, donde $M \in \mathcal{MS}(P_{\mathcal{N}})$, $\mathit{loc} : \mathcal{N} \to \mathcal{L}$ y R es una función de dependencia de $\mathcal{N}$ tal que:*

- *$G(R)$ es un bosque dirigido,*
- *Para cada $N \in \mathcal{N}$, si $o_N = \top$ entonces N no es esclavo en R.*

Por tanto, un marcaje de un sistema TSA está compuesto por un marcaje ordinario, como los vistos en el capítulo 3, junto con una función de dependencia. Diremos que una componente N está viva en R si $o_N = \top$ o es un esclavo en R. En caso contrario, diremos que está muerta.

A continuación definimos el comportamiento de los sistemas TSA. En general, sólo las componentes vivas pueden disparar transiciones. En primer lugar, definamos las transiciones autónomas, cuya única diferencia con las transiciones autónomas de nuestras redes ubicuas, es la expresada en la frase anterior.

Definición 11.2.5 *Sea $\mathcal{N}$ un sistema TSA, $\mathcal{M} = (M, loc, R)$ un marcaje de $\mathcal{N}$, N una componente viva en R, y $t \in T_N$ con $\lambda(t) \in \mathcal{A}$. Diremos que t está activada en $\mathcal{M}$ si ${}^\bullet t \subseteq M$, en cuyo caso diremos que t se puede disparar, produciendo el marcaje (M', loc', R) dado por*

- $M' = M - {}^\bullet t + t^\bullet$.
- *Si $\lambda(t) = go\ k$ entonces $loc'(N) = k$ y $loc'(N) = loc(N)$, para todo $N' \neq N$. En caso contrario, $loc' = loc$.*

Ahora definamos el disparo de transiciones de impresión. La red que la dispara especifica el tipo de la red que quiere imprimir.

Definición 11.2.6 *Sea $\mathcal{N}$ un sistema TSA, $\mathcal{M} = (M, loc, R)$ un marcaje de $\mathcal{N}$, N y N' dos componentes co-localizadas, viva y muerta en R, respectivamente, y $t \in T_N$ con $\lambda(t) = (i, imprint(o_{N'}))$. Decimos que t está activada en $\mathcal{M}$ si*

- ${}^\bullet t \subseteq M$,
- $(N, i) \notin Dom(R)$.

En ese caso t se puede disparar, produciendo el marcaje (M', loc, R'), donde $M' = M - {}^\bullet t + t^\bullet$ y R' extiende R con $R'(N, i) = N'$.

Por tanto, la red impresa ha de estar muerta para poder serlo, y del tipo requerido por la componente que la imprime. Además, la red que imprime no ha de tener ya una red i-esclava.

Ahora definamos las transiciones que representan órdenes de los amos a los esclavos, que se corresponden con las sincronizaciones ordinarias de los capítulos anteriores.

Definición 11.2.7 *Sea $\mathcal{N}$ un sistema TSA, $\mathcal{M} = (M, loc, R)$ un marcaje de $\mathcal{N}$, y N y N' dos componentes vivas y co-localizadas tales que N' sea una i-esclava de N en R. Sean también $t \in T_N$ con $\lambda(t) = (i, s!)$ y $t' \in T_{N'}$ con $\lambda(t') = s?$. Diremos que el par de transiciones (t, t') está activado en $\mathcal{M}$ si ${}^\bullet t \cup {}^\bullet t' \subseteq M$, en cuyo caso se puede disparar, obteniéndose el marcaje (M', loc, R), donde $M' = M - {}^\bullet t - {}^\bullet t' + t^\bullet + t'^\bullet$.*

Por tanto, si N' es un i-esclavo de N entonces N le puede dar órdenes a N', lo cual queda formalizado por medio del disparo síncrono de un par de transiciones. Finalmente, veamos cómo un amo puede matar a uno de sus esclavos.

Definición 11.2.8 *Sea $\mathcal{N}$ un sistema TSA, $\mathcal{M} = (M, loc, R)$ un marcaje de $\mathcal{N}$, N y N' dos componentes vivas y co-localizadas tales que N' es una i-esclava de N, y $t \in T_N$ tal que $\lambda(t) = (i, kill)$. Diremos que t está activada en $\mathcal{M}$ si ${}^\bullet t \subseteq M$, en cuyo caso se puede disparar, produciendo (M', loc, R'), donde $M' = M - {}^\bullet t + t^\bullet$ y R' es el resultado de*

- *borrar (N, i) del dominio de R, y*
- *borrar $(\overline{N}, j)$ del dominio de R, para cada $\overline{N}$ descendiente de N' en $G(R)$ y cada j posible.*

Como indicamos antes, al matar a N' también se mata indirectamente a todas las componentes que dependen de ella. Esto es así incluso en el caso de que esas componentes no estén co-localizadas con N y N'. Alternativamente, podríamos forzar a que N' buscase

y matase explícitamente a sus esclavos, y así recursivamente, aunque esto no modificaría el potencial comportamiento de las componentes que mueren.

Como siempre, usaremos $u, u', \ldots$ para referirnos a transiciones o pares de transiciones. Por tanto, en cualquiera de los casos vistos escribiremos $\mathcal{M}[u\rangle\mathcal{M}'$ si $\mathcal{M}'$ es el resultado de disparar u en $\mathcal{M}$.

Claramente, el marcaje alcanzado después del disparo de una transición o de un par de transiciones es efectivamente un marcaje, es decir, el grafo de dependencia sigue siendo siempre un bosque dirigido. Este hecho es obvio, sin más que tener en cuenta que sólo las componentes vivas pueden imprimir, y sólo a componentes muertas.

Por último, vamos a definir el orden entre marcajes de un sistema TSA con el que trabajaremos. Diremos que $(M_1, loc_1, R_1) \sqsubseteq (M_2, loc_2, R_2)$ si $M_1 \subseteq M_2$, $loc_1 = loc_2$ y $R_1 = R_2$.

11.3. Un ejemplo: Termómetros electrónicos

En esta sección presentamos un ejemplo sencillo, que nos ayudará a ilustrar las definiciones anteriores. Consideraremos un escenario con varios pacientes en dos cuartos de un hospital, k_1 y k_2. Cada uno de los pacientes tiene colocado un termómetro electrónico [Wan04] que lee la temperatura cada cierto tiempo. Los termómetros están dispuestos a enviar (por ejemplo, usando RFID) un mensaje indicando simplemente si todo va bien o no con el paciente controlado por ellos. Supondremos que los termómetros están inactivos (es decir, muertos según nuestra terminología), hasta que un controlador tome el control sobre ellos. Intuitivamente, podemos pensar que en cada cuarto tenemos uno de estos controladores, que lee los mensajes enviados por los termómetros y envía un mensaje de alarma a la enfermera cuando algo marcha mal. Para simplificar, supondremos que en realidad todos estos controladores forman parte de una única componente que se mueve entre los distintos cuartos del hospital y que sólo puede controlar simultáneamente a dos termómetros, de modo que tiene capacidad dos.

Finalmente, hay una enfermera con turno de noche, que tiene que controlar la temperatura de los pacientes. Para ello, puede imprimir un controlador, de manera que sólo ella reciba los mensajes de alarma. Por tanto, el ciclo de vida de una enfermera con turno de noche consiste en ir de su casa a la enfermería, imprimir un controlador de termómetros, irse a descansar a la enfermería, ver qué le ocurre a los pacientes en caso de que reciba un mensaje de alarma y matar al controlador antes de marcharse de nuevo a casa.

Modelamos este escenario por medio de un sistema TSA con varias componentes. La primera, mostrada en la figura 11.2, representa a la enfermera E, que suponemos que tiene tipo $\top$. Inicialmente, se encuentra en la localidad *home*. Entonces, puede disparar *go inf*, para moverse a la enfermería, donde puede imprimir un controlador e irse a descansar. Como la enfermera tiene capacidad uno, en las figuras usamos etiquetas simples, como *kill* en lugar de $(1, \mathit{kill})$.

En la figura 11.3 se muestra el controlador de termómetros C, inicialmente en la localidad *inf*. El controlador no tiene tipo $\top$, de manera que tiene que esperar a ser impreso antes de efectuar cualquier acción. Después de ser impreso, puede ir a la localidad k_1 o a k_2 donde están los pacientes T_1 y T_2 respectivamente, que aparecen en la figura 11.4, ambos de tipo t. Ahora el controlador se encuentra ante una opción entre *good*!, *bad*! o *imprint*(*patient*). Sin embargo, como los termómetros de los pacientes están aún muertos (en nuestro sentido), por el momento el controlador sólo puede imprimir uno de ellos. Después podrá elegir de nuevo entre ambas localidades. Si elige la misma que la vez anterior puede leer el termómetro que imprimió y, en caso de encontrarse con el valor

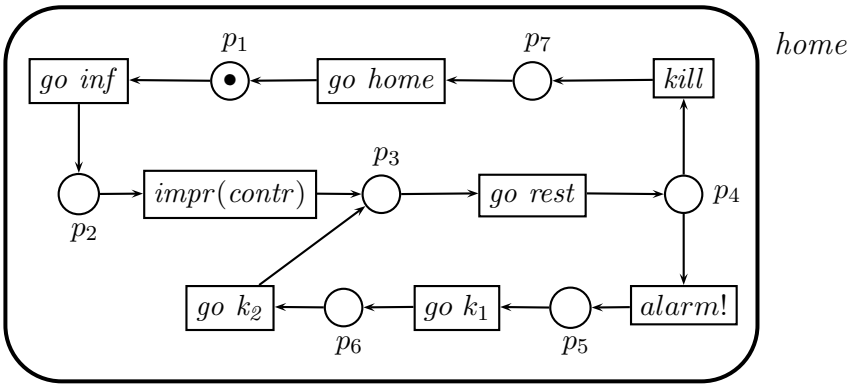

Figura 11.2: Enfermera

bad, es decir, si pudiera sincronizar con el termómetro en la transición *bad*!, se dirigiría al cuarto donde descansa la enfermera, enviándole el mensaje de alarma.

En todo momento, la enfermera puede disparar su transición *kill*, matando así al controlador (y, por tanto, matando también a los termómetros impresos por el controlador). Obsérvese que los controladores no tienen ninguna transición *kill*, de manera que la única manera de matar a sus termómetros es que enfermera lo mate de hecho a él.

Veamos con detalle la sucesión de acontecimientos en la que:

1. la enfermera va a la enfermería,
2. la enfermera imprime el controlador,
3. la enfermera va al cuarto de descanso,
4. el controlador va a k_1,
5. el controlador imprime el termómetro del cliente,
6. el controlador va a k_2,
7. el controlador imprime el termómetro del otro cliente,
8. el controlador vuelve a k_1,
9. el controlador comprueba que el paciente en k_1 está bien,
10. el controlador va a k_2,
11. el controlador detecta que el paciente en k_2 tiene fiebre,
12. el controlador va al cuarto de descanso,
13. el controlador le da a la enfermera la voz de alarma.

Vamos a representar las funciones de localización del sistema mediante pares (l_1, l_2), donde l_1 es la localización de la enfermera y l_2 la del controlador, ya que los pacientes están inmóviles en las localidades k_1 y k_2. Pues bien, cada una de las acciones anteriores se corresponde con un disparo en la siguiente traza:

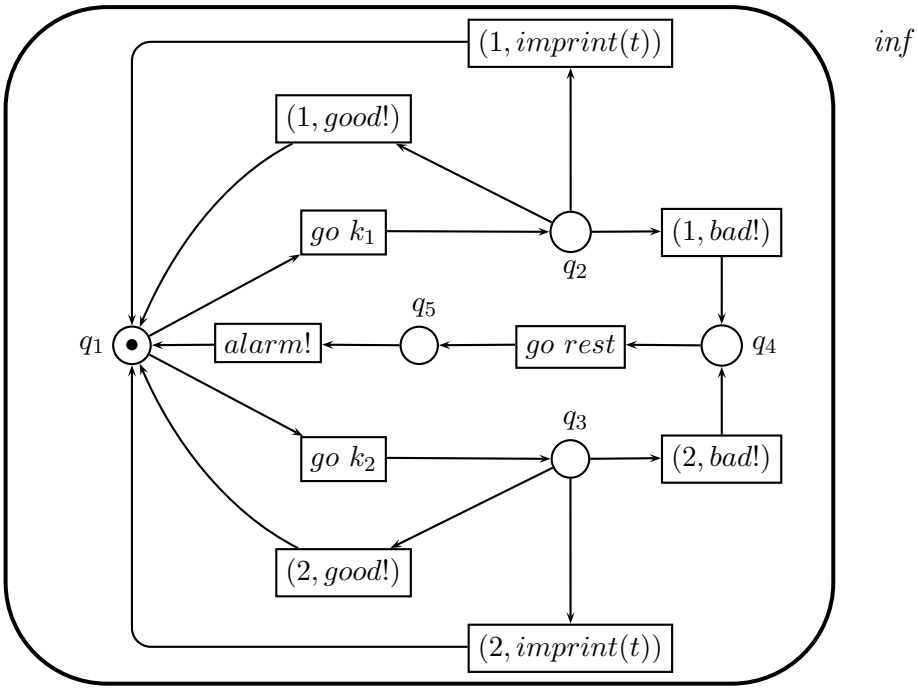

Figura 11.3: Controlador de termómetros

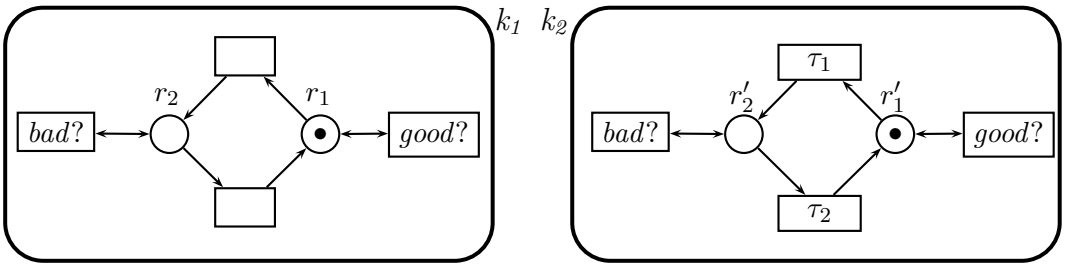

Figura 11.4: Pacientes

$(M_0, (home, inf), R_0)$	$[go\ inf\rangle$	$(M_1, (inf, inf), R_0)$	$[impr(contr)\rangle$
$(M_2, (inf, inf), R_1)$	$[go\ rest\rangle$	$(M_3, (rest, inf), R_1)$	$[go\ k_1\rangle$
$(M_4, (rest, k_1), R_1)$	$[(1, impr(t))\rangle$	$(M_3, (rest, k_1), R_2)$	$[go\ k_2\rangle$
$(M_5, (rest, k_2), R_2)$	$[(2, impr(t))\rangle$	$(M_3, (rest, k_2), R_3)$	$[go\ k_1\rangle$
$(M_4, (rest, k_1), R_3)$	$[((1, good!), good?)\rangle$	$(M_3, (rest, k_1), R_3)$	$[go\ k_2\rangle$
$(M_5, (rest, k_2), R_3)$	$[\tau_2\rangle$	$(M_6, (rest, k_2), R_3)$	$[(1, bad!), bad?)\rangle$
$(M_7, (rest, k_2), R_3)$	$[go\ rest\rangle$	$(M_8, (rest, rest), R_3)$	$[(alarm!, alarm?)\rangle$
$(M_9, (rest, rest), R_3)$			

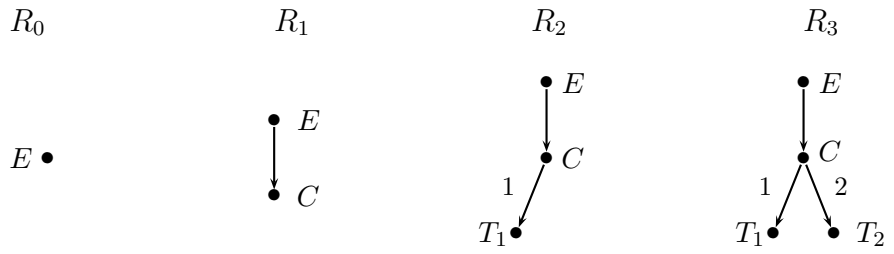

Figura 11.5: Árboles de dependencia

donde $M_0 = \{p_1, q_1, r_1, r'_1\}$ $M_1 = \{p_2, q_1, r_1, r'_1\}$ $M_2 = \{p_3, q_1, r_1, r'_1\}$
$M_3 = \{p_4, q_1, r_1, r'_1\}$ $M_4 = \{p_4, q_2, r_1, r'_1\}$ $M_5 = \{p_4, q_3, r_1, r'_1\}$
$M_6 = \{p_4, q_3, r_1, r'_2\}$ $M_7 = \{p_4, q_4, r_1, r'_2\}$ $M_8 = \{p_4, q_5, r_1, r'_2\}$
$M_9 = \{p_5, q_1, r_1, r'_2\}$,

las funciones de dependencia están representadas en la figura 11.5 y estamos identificando las transiciones con sus etiquetas, ya que no hay posibilidad de malentendidos por tener todas las transiciones una etiqueta distinta.

11.4. Verificación de sistemas TSA

En la sección anterior hemos presentado un modelo que nos permite especificar sistemas que usen las primitivas de la política del pato que resucita, que son correctas por definición. En esta sección vamos a usar los resultados de decidibilidad probados en el capítulo 6 para la verificación de sistemas TSA.

Antes de nada, veamos que, una vez más, la componente localidad de los sistemas TSA no juega un papel relevante en el modelo. Diremos que un sistema TSA es *centralizado* si ninguna de sus componentes tiene transiciones de movimiento y todas ellas están inicialmente localizadas en la misma localidad. Bajo estas hipótesis, podemos ignorar la componente localidad de los marcajes, escribiendo simplemente (M, R), en lugar de (M, loc, R). En el siguiente resultado escribiremos $\mathcal{A}_c$ para referirnos al conjunto $\mathcal{A} \setminus \{go\ k \mid k \in \mathcal{L}\}$.

Lema 11.4.1 *Todo sistema TSA puede ser simulado monótonamente por un sistema TSA centralizado.*

Demostración. Sea $\mathcal{N}$ un sistema TSA. Dada $N \in \mathcal{N}$ definimos la componente TSA centralizada N^*, de modo que el sistema centralizado $\mathcal{N}^* = \{N^* \mid N \in \mathcal{N}\}$ simule $\mathcal{N}$. Sea $\mathcal{K} = \{k \mid \exists t \in T_{\mathcal{N}}, \lambda(t) = go\ k\} \cup loc_0(\mathcal{N}) \subseteq \mathcal{L}$, que es el conjunto de localidades en las que pueden encontrarse las componentes en cualquier marcaje alcanzable. Como ese conjunto es finito, podemos añadir un lugar $N@k$ para cada componente N y localidad $k \in \mathcal{K}$, todos ellos marcados en exclusión mutua, quedando representada así la localización de la componente como parte de su marcaje ordinario. Entonces, para cada transición no autónoma consideraremos una transición distinta por cada localidad, de manera que el disparo de una transición no autónoma t en k está representado por el disparo de la transición $t(k)$. Por tanto, el lugar $N@k$ deberá ser tanto precondición como postcondición de la transición $t(k)$ si t es de sincronización (si imprime, mata, o da o recibe órdenes) y,

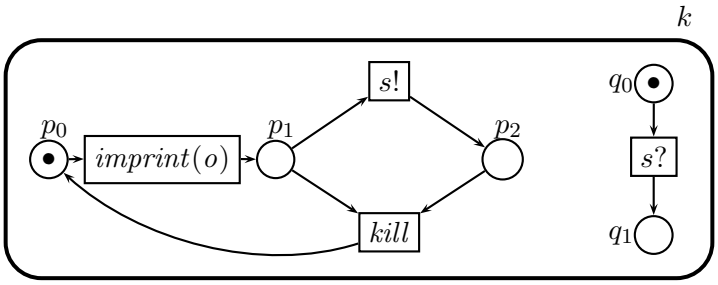

Figura 11.6: Ejemplo simple de sistema TSA

si t es de movimiento, deberá mover el token de los nuevos lugares como corresponda, del lugar que representa a la localidad donde se encuentra la red al lugar que representa la localidad destino. Para cada $k \in \mathcal{L}$ consideraremos una etiqueta autónoma $mov(k)$ en $\mathcal{A}$. Si $N = (P, T, F, \lambda, o)$ tomamos $N^* = (P^*, T^*, F^*, \lambda^*, o)$, donde

- $P^* = P \cup \{N@k \mid k \in \mathcal{K}\}$,
- $T^* = \{t \in T \mid \lambda(t) \in \mathcal{A}_c\} \cup \{t(k) \mid \lambda(t) \notin \mathcal{A}_c,\ k \in \mathcal{K}\}$,
- $F^* = \{(p,t) \in F \mid \lambda(t) \in \mathcal{A}_c\} \cup \{(t,p) \in F \mid \lambda(t) \in \mathcal{A}_c\} \cup$
  $\{(p, t(k)) \mid (p,t) \in F, \lambda(t) \notin \mathcal{A}_c\} \cup \{(t(k), p) \mid (t,p) \in F, \lambda(t) \notin \mathcal{A}_c\} \cup$
  $\{(N@k, t(k)), (t(k), N@k) \mid k \in \mathcal{K},\ \lambda(t) \neq go\ k'\} \cup$
  $\{(t(k), N@k') \mid \lambda(t) = go\ k'\}$,
- $\lambda^*(t) = \lambda(t)$ y $\lambda^*(t(k)) = \begin{cases} mov(k) \text{ si } \lambda(t) = go\ k \\ \lambda(t) \text{ en otro caso} \end{cases}$.

Siendo $\mathcal{M} = (M, loc, R)$ un marcaje alcanzable definimos $\mathcal{M}^* = (M^*, R)$, donde $M^*(p) = M(p)$ para cada $p \in P$, $M^*(N@loc(N)) = 1$ y $M^*(N@k) = 0$ si $k \neq loc(N)$, para todo N. Entonces se cumple que

$$\mathcal{M}_1[u\rangle\mathcal{M}_2 \Leftrightarrow \mathcal{M}_1^*[u^*\rangle\mathcal{M}_2^*$$

donde, si $\mathcal{M}_1 = (M_1, loc_1, R_1)$ tendremos

$$u^* = \begin{cases} u \text{ si } \lambda(u) \in \mathcal{A}_c \\ u(k) \text{ si } \lambda(u) = go\ k' \\ (t(k), t'(k)) \text{ si } u = (t,t') \in T_N \times T_{N'},\ loc_1(N) = loc_1(N') = k. \end{cases}$$

La demostración es análoga a la de la proposición 7.3.3. Allí algunas transiciones creaban copias de componentes, cosa que no ocurre aquí. Sin embargo, aquí algunas transiciones modifican la función de dependencia, lo que permanece intacto en la simulación.

Además, todo marcaje alcanzable en $\mathcal{N}^*$ es estable y como $M_i^*(p) = M_i(p)$ para todo $p \in P$ y $M_i(N@k) = \{loc_i(N)\}(k)$ se tiene que $(M_1, loc_1, R_1) \sqsubseteq (M_2, loc_2, R_2) \Leftrightarrow loc_1 = loc_2,\ R_1 = R_2,\ loc_1 = loc_2 \Leftrightarrow (M_1^*, R_1) \sqsubseteq (M_2^*, R_2)^* \Leftrightarrow (M_1, loc_1, R_1)^* \sqsubseteq (M_2, loc_2, R_2)^*$. Entonces podemos aplicar la proposición 2.6.7 y la tesis se sigue.

□

En primer lugar, probamos que todo sistema TSA puede ser simulado por una red P/T.

Proposición 11.4.2 *Todo sistema TSA puede ser simulado monótonamente por una red P/T.*

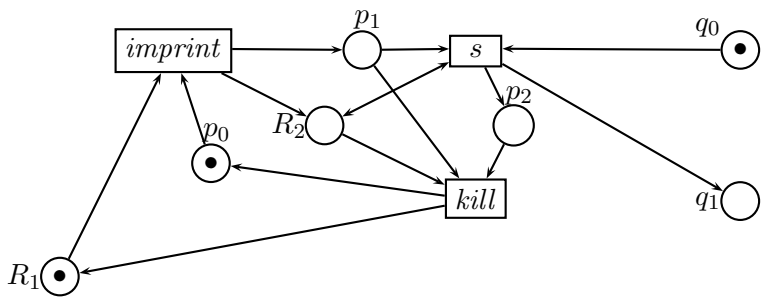

Figura 11.7: Traducción del sistema TSA de la figura 11.6

Demostración. La demostración se basa en el hecho de que, como los sistemas TSA son cerrados, el número de componentes que tendremos en todo momento será el mismo que en el marcaje inicial y, por tanto, el número de funciones de dependencia es finito. Así que podemos especificar la función de dependencia de un marcaje, simplemente teniendo un lugar distinto para cada una de ellas, marcando el lugar correspondiente, introduciendo esos lugares como precondición y postcondición de las transiciones de impresión y *kill*. Según el lema anterior podemos asumir que partimos de un sistema TSA centralizado $\mathcal{N}$ con marcaje inicial $\mathcal{M}_0 = (M_0, R_0)$, y tenemos que construir una red P/T $N^* = (P^*, T^*, F^*)$ con marcaje inicial $\mathcal{M}_0^*$ que la simula. Primero vamos a introducir algunas notaciones. Escribiremos $OK(t, t', R)$ siempre que $t \in T_N$, $t' \in T_{N'}$, tanto N como N' estén vivas y N' sea una esclava de N, según R. Análogamente, escribiremos $OK(t, N, R)$ siempre que $\lambda(t) = \mathit{imprint}(o_N)$ y N esté muerta según R, o siempre que $\lambda(t) = \mathit{kill}$ y N esté viva según R. Finalmente, dada una función de dependencia R, escribiremos $R(t, N)$ para denotar la función de dependencia que resulta de disparar t con N como objetivo. En particular, si $\lambda(t) = (i, \mathit{imprint}(o_N))$ entonces $R(t, N)$ es el resultado de añadir a R, $R(N', i) = N$, donde $t \in T_{N'}$. Si $\lambda(t) = (i, \mathit{kill})$ entonces $R(t, N)$ es el resultado de eliminar (i, N) del dominio de R, así como todo descendiente de N.

Para simular el sistema TSA por medio de una red P/T, para cada componente N añadimos lugares $\mathit{dead}(N)$ y $\mathit{alive}(N)$, marcados en exclusión mutua, que indicarán si la componente está viva o muerta. Como hay un número finito de componentes, cada una de ellas con una capacidad finita, hay sólo un número finito de funciones de dependencia. Entonces podemos considerar un lugar para cada una de ellas, marcados también en exclusión mutua, que utilizaremos para conseguir que las sincronizaciones sólo tengan lugar en las configuraciones correctas. Por ejemplo, para cada par (t, t') de transiciones de sincronización, etiquetadas respectivamente por $s?$ e $(i, s!)$, y para cada función de dependencia R consideramos la transición (t, t', R), que representa el disparo sincronizado del par (t, t') en el estado de dependencias R. Por supuesto, esta sincronización sólo se puede llevar a cabo cuando según R la red que dispara t' sea una i-esclava de la que dispara t'. Por tanto, esta transición deberá tener R como precondición y postcondición, así como las precondiciones y postcondiciones tanto de t como de t'. Una construcción análoga se aplica a las transiciones de impresión y a las transiciones *kill*.

Formalmente, definimos los siguientes conjuntos:

- $P^* = P_{\mathcal{N}} \cup \{\mathit{alive}(N), \mathit{dead}(N) \mid N \in \mathcal{N}\} \cup$
  $\{R \mid R \text{ función de dependencia de } \mathcal{N}\}$,
- $T^* = \{t \in T_{\mathcal{N}} \mid \lambda(t) \in \mathcal{A}\} \cup$

$\{(t,t',R) \mid \lambda(t) = s?, \lambda(t') = (i,s!), OK(t,t',R)\} \cup$
$\{(t,N,R) \mid OK(t,N,R),\ \lambda(t) = \mathit{kill}\ o\ \lambda(t) = \mathit{imprint}(o_N)\}$,

- $F^* = F_1 \cup F_2 \cup F_3$, donde
  - $F_1 = \{(p,t) \mid \lambda(t) \in \mathcal{A},\ p \in {}^\bullet t\} \cup \{(t,p) \mid \lambda(t) \in \mathcal{A},\ p \in t^\bullet\} \cup$
    $\{(\mathit{alive}(N),t),(t,\mathit{alive}(N)) \mid t \in T_N\}$,
  - $F_2 = \{(p,(t,t',R)) \mid p \in {}^\bullet t \cup {}^\bullet t'\} \cup \{((t,t',R),p) \mid p \in t^\bullet \cup t'^\bullet\} \cup$
    $\{(R,(t,t',R)),((t,t',R),R) \mid (t,t',R) \in T^*\}$,
  - $F_3 = \{(p,(t,N,R)) \mid p \in {}^\bullet t\} \cup \{((t,N,R),p) \mid p \in t^\bullet\} \cup$
    $\{(R,(t,N,R)) \mid (t,N,R) \in T^*\} \cup \{((t,N,R),R(t,N))\} \cup$
    $\{(\mathit{alive}(N),(t,N,R)) \mid \lambda(t) = \mathit{kill}\} \cup$
    $\{((t,N,R),\mathit{dead}(N)) \mid \lambda(t) = \mathit{kill}\} \cup$
    $\{(\mathit{dead}(N),(t,N,R)) \mid \lambda(t) = \mathit{imprint}(o_N)\} \cup$
    $\{((t,N,R),\mathit{alive}(N)) \mid \lambda(t) = \mathit{imprint}(o_N)\}$.

Finalmente, dado un marcaje $\mathcal{M} = (M,R)$ de $\mathcal{N}$, definimos el marcaje $\mathcal{M}^*$ de $\mathcal{N}^*$ como sigue:

- $\mathcal{M}^*(p) = M(p)$ para todo $p \in P_{\mathcal{N}}$,
- $\mathcal{M}^*(R) = 1$ y $\mathcal{M}^*(R') = 0$ para todo $R' \neq R$,
- $\mathcal{M}^*(\mathit{alive}(N)) = \begin{cases} 1 \text{ si } N \text{ viva en } R \\ 0 \text{ en otro caso,} \end{cases}$
- $\mathcal{M}^*(\mathit{dead}(N)) = \begin{cases} 1 \text{ si } N \text{ muerta en } R \\ 0 \text{ en otro caso.} \end{cases}$

Usando estas notaciones, se cumple que

$$(M_1,R_1)[u\rangle(M_2,R_2) \Leftrightarrow (M_1,R_1)^*[u^*\rangle(M_2,R_2)^*,$$

donde $u^* = \begin{cases} t & \text{si } u \in T_{\mathcal{N}} \text{ y } \lambda(t) \in \mathcal{A} \\ (t,t',R_1) & \text{si } u = (t,t') \\ (t,N,R_1) & \text{si } u = t \text{ con } \lambda(t) = \mathit{kill}, \text{ o } \lambda(t) = \mathit{imprint}(o_N). \end{cases}$

El caso en el que u sea una transición autónoma de una red o una sincronización de dos transiciones compatibles es análogo a los casos respectivos de la demostración de la proposición 3.2.3, teniendo en cuenta que los lugares $\mathit{alive}(N)$ están marcados en la simulación si y sólo si la correspondiente red está viva por definición de $(\ )^*$. Veamos qué ocurre si $u = t$, $\lambda(t) = \mathit{imprint}(o_{N'})$ y $u^* = (t,N,R_1)$. En este caso basta tener en cuenta que ${}^\bullet(t,N,R_1) = {}^\bullet t \cup \{\mathit{alive}(N),\mathit{dead}(N'),R_1\}$ y $(t,N,R_1)^\bullet = t^\bullet \cup \{\mathit{alive}(N),\mathit{alive}(N'),R_1(t,N)\}$, de modo que t se puede disparar si y sólo si todos sus precondiciones tienen tokens, N está viva y N' está muerta según R_1, lo que ocurre si y sólo si (t,N,R_1) tiene tokens en todas sus precondiciones, y los marcajes alcanzados son los correspondientes. Si $\lambda(t) = \mathit{kill}$ entonces es análogo, con ${}^\bullet(t,N,R_1) = {}^\bullet t \cup \{\mathit{alive}(N),\mathit{alive}(N'),R_1\}$ y y $(t,N,R_1)^\bullet = t^\bullet \cup \{\mathit{alive}(N),\mathit{dead}(N'),R_1(t,N)\}$.

Todo marcaje alcanzable tiene un sólo token en los lugares de R y para cada N un token en $\mathit{alive}(N)$ si y sólo si no lo tiene en $\mathit{dead}(N)$ y N está viva en según el único R marcado, por lo que todo marcaje alcanzable es estable. Además, $(M_1,R_1) \sqsubseteq (M_2,R_2) \Leftrightarrow M_1 \subseteq M_2$ y $R_1 = R_2 \Leftrightarrow (M_1,R_1)^* \sqsubseteq (M_2,R_2)^*$. Esta última implicación a la derecha es trivial por definición de $(\ )^*$ y la recíproca se tiene porque si $(M_1,R_1)^*(R) \leq$

$(M_2, R_2)^*(R)$ para todo R, como $(M_i, R_i)^*(R_i) = 1$ y $(M_i, R_i)^*(R_j) = 0$ para $i \neq j$ entonces $(M_1, R_1)^*(R) = (M_2, R_2)^*(R) = 1$ para todo R y por lo tanto $R_1 = R_2$. Una vez más la proposición 2.6.7 nos permite concluir la tesis.

□

La figura 11.7 muestra la red P/T que simula al sistema TSA de la figura 11.6, suponiendo que la red de la derecha tiene tipo o. En esa figura usamos el lugar R_1 para la función de dependencia en la que no existen asociaciones entre componentes, y R_2 para aquélla en la que la componente de la izquierda de la figura 11.6 es amo de la de la derecha. Entonces, la transición etiquetada por $imprint(o)$ sólo se puede disparar si hay un token en R_1, y su disparo tiene el efecto de mover el token de R_1 a R_2, además de mover el token de p_0 a p_1. La transición *kill* tiene un comportamiento análogo. Finalmente, las transiciones $s!$ y $s?$ se mezclan en una sola transición s, que tiene a R_2 tanto como precondición como postcondición.

Mediante la aplicación de este resultado, podemos usar toda la artillería existente para el análisis de redes P/T en el análisis de nuestros sistemas TSA. Por ejemplo, los problemas de recubrimiento y *home state* son decidibles para las redes P/T. Por tanto, en principio podemos chequear algorítmicamente cualquier propiedad especificada en tales términos. Así, para el ejemplo de este capítulo, podríamos comprobar que cuando una enfermera imprime un termómetro entonces, cuando es necesario, este último eventualmente da la señal de alarma, y sólo al enfermero apropiado, ya que ésta es una propiedad *home state*. También podríamos preguntarnos si siempre que la enfermera recibe una señal de alarma visita a los pacientes. Por último, si suponemos que hay varias enfermeras como la de la figura 11.2 y varios controladores como el de la figura 11.3, podríamos especificar en términos de recubrimiento la propiedad que dice que cada enfermera posee a lo sumo un controlador.

Hasta ahora estamos tratando los sistemas TSA como especificaciones de sistemas que usan la política del pato que resucita, ya que nuestras primitivas suponen que los mecanismos para imprimir y para matar son siempre correctos. Sin embargo, es deseable poder llevar a cabo también un análisis de más bajo nivel, con suposiciones más débiles. Para ello, vamos a probar que todo sistema TSA puede ser implementado por un sistema MSPN como los del capítulo 4. La simulación sólo maneja nombres puros, que se pueden ver intuitivamente como las claves utilizadas en la impresión, así que no cuenta con primitivas para la asociación segura transitoria que se asuman correctas. Además, esta simulación preservará la estructura basada en componentes de los sistemas MSPN, por lo que se puede ver como una implementación. Es más, para implementar los sistemas TSA nos bastaría con la subclase de los sistemas MSPN que no crean nombres que, como vimos en el capítulo 6, son equivalentes a las redes P/T, de modo que todos los comentarios que hicimos antes acerca de la comprobación de propiedades se pueden seguir aplicando.

Dado un sistema TSA $\mathcal{N}$, que en virtud del lema 11.4.1 podemos suponer centralizado, para cada $N = (P, T, F, \lambda, o) \in \mathcal{N}$ queremos definir una MSPN N^* de manera que el sistema MSPN $\mathcal{N}^* = \{N^* \mid N \in \mathcal{N}\}$ implemente a $\mathcal{N}$. La principal diferencia con el resultado anterior está en el hecho de que las componentes no pueden tener una visión global de las dependencias en el sistema, sino sólo de aquellas relaciones por las que se vean directamente afectadas, es decir, sólo pueden saber qué componentes son sus amos y cuáles sus esclavos. Por ello, añadimos a cada componente un lugar $master_N$, que contendrá en cada momento la clave que se usa para comunicarse con su amo. Análogamente, si una red tiene capacidad k entonces añadimos lugares $slave_N(1), \ldots, slave_N(k)$, que contendrán las claves usadas para comunicarse con los correspondientes esclavos. Las componentes vivas

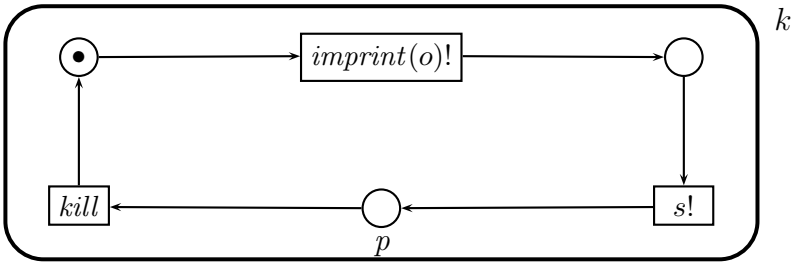

Figura 11.8: Sistema TSA

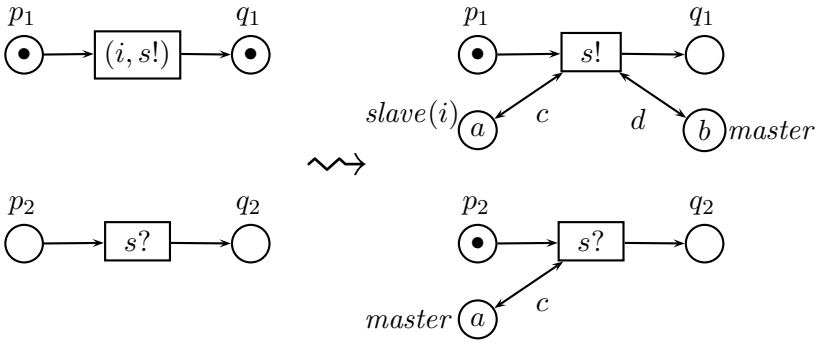

Figura 11.9: Implementación de sincronizaciones

serán aquéllas que tengan algún token en su lugar $master_N$. Como solamente éstas pueden ejecutar acciones, añadimos ese lugar como precondición de todas sus transiciones (por tanto, las componentes de tipo $\top$, que siempre estarán vivas, han de tener un token en ese lugar en cada momento, aunque este no se use para comunicarse con un amo, que de hecho no tienen). También añadimos un lugar $dead_N$, marcado con un token negro en exclusión mutua con $master_N$. Finalmente, añadimos para cada N e $i \in \{1, \ldots, k\}$ un lugar $\overline{slave}_N(i)$ donde N almacena la clave que usa para comunicarse con su i-esclavo cuando no tiene tal i-esclavo.

Veamos ahora cómo se implementan las transiciones de cada tipo. Las transiciones autónomas se dejan como están, excepto por el hecho ya mencionado de que es necesario añadir el lugar $master_N$ como precondición y, en este caso, también como postcondición, de modo que sólo puedan dispararse cuando la red está viva y la siga estándolo.

La sincronización de dos transiciones (véase la figura 11.9), etiquetadas por $(i, s!)$ y $s?$, respectivamente, se implementa mediante la sincronización de dos transiciones, ahora etiquetadas con $s!$ y $s?$. Sin embargo, queremos que estas transiciones sólo sincronicen en el caso de que las correspondientes redes compartan la relación de dependencia adecuada. Para ello añadimos el lugar $slave(i)_N$ como precondición (y postcondición) de $s!$ y el lugar $master_N$ como precondición (y postcondición) de $s?$, etiquetando todos esos arcos con la misma variable. De esta manera, sólo las componentes que contengan el mismo token en esos lugares, y que comparten pues la misma clave, podrán comunicarse entre ellos.

Por tanto, para que la implementación de las sincronizaciones sea correcta, debemos garantizar que las componentes sólo comparten claves en el caso de que así lo indiquen las primitivas de impresión y muerte. En la implementación, las transiciones de impresión y las etiquetadas con *kill* no son más que transiciones de sincronización, de modo que supondremos que existen etiquetas de sincronización *kill* e $imprint(o)$, para todo $o \in \mathcal{O}$. La

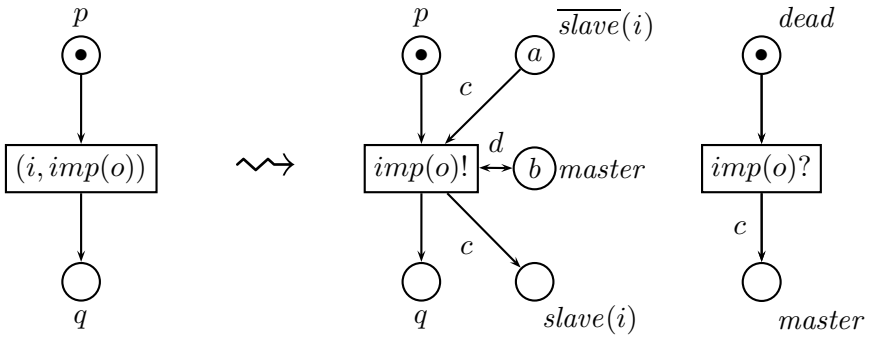

Figura 11.10: Implementación de impresiones

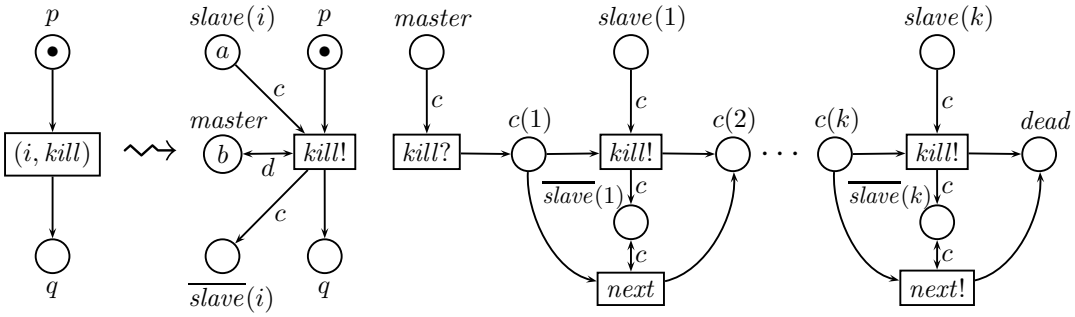

Figura 11.11: Implementación de transiciones *kill*

impresión de una componente de tipo o se implementa mediante el disparo sincronizado de dos transiciones, etiquetadas con $imprint(o)!$ e $imprint(o)?$ (véase la figura 11.10). Téngase en cuenta que la segunda transición sólo puede ser disparada en el caso de que la componente esté muerta, de modo que ésa es la única transición que tiene al lugar *dead*, y no a *master*, como precondición. Cuando estas dos transiciones sincronizan se toma un token, digamos η, del lugar $\overline{slave}(i)$ de la componente que imprime y se coloca otro igual tanto en su correspondiente lugar $slave(i)$ como en el lugar *master* de la componente impresa.

El comportamiento de la implementación de las transiciones *kill* es análogo, por medio de la sincronización de dos transiciones etiquetadas por *kill*! y *kill*?. Básicamente, la red que mata mueve el token del correspondiente lugar $slave(i)$ a $\overline{slave}(i)$, y la red muerta elimina el token de su lugar *master* e introduce un token negro en el lugar *dead*. Sin embargo, la muerte de una componente también ha de causar la muerte de todos sus esclavos, de modo que antes de colocar ese token en *dead* debemos matar a sus esclavos, de idéntica forma. Para ello añadimos transiciones $next(i)$ y $kill(i)$ para cada $i \in cap(N)$, de modo que si la red tiene un i-esclavo (tiene marcado el lugar $slave(i)$) entonces dispara la transición $kill(i)$ que mata a su i-esclava y procede con las siguientes, y si no tenía i-esclavo simplemente prosigue (disparando la transición $next(i)$).

Veamos estas ideas formalmente.

Definición 11.4.3 *Dado un sistema TSA centralizado $\mathcal{N}$ definimos el sistema MSPN $\mathcal{N}^* = \{N^* \mid N \in \mathcal{N}\}$ donde si $N = (P, T, F, \lambda, o)$ entonces $N^* = (P^*, T^*, F^*, \lambda^*)$ con:*

- $P^* = P \cup \{master_N\} \cup \{c_N(i), \overline{slave}_N(i) slave_N(i) \mid i \in cap(N)\}$,

- $T^* = T \cup \{kill_N(i), next_N(i) \mid i \in cap(N)\} \cup \{die_N, impr_N\}$,
- F^* *es la función parcial con menor dominio tal que:*
  - $F^*(p,t) = \varepsilon$ *si* $(p,t) \in F$ *y* $F^*(t,p) = \varepsilon$ *si* $(t,p) \in F$,
  - $F^*(master_N, t) = F(t, master_N) = d$ *para todo* $t \in T$ *con* $\lambda(t) \neq s?$ *para todo* s,
  - $F^*(slave_N(i), t) = F^*(t, slave_N(i) = c$ *si* $\lambda(t) = (i, s!)$,
  - $F^*(master_N, t) = F(t, master_N) = c$ *si* $\lambda(t) = s?$,
  - $F^*(dead_N, impr) = \varepsilon$ *y* $F^*(impr, master_N = c)$,
  - $F^*(\overline{slave}_N(i), t) = F^*(t, slave_N(i)) = c$ *si* $\lambda(t) = (i, imprint(o'))$,
  - $F^*(slave_N(i), t) = F^*(t, \overline{slave}_N(i)) = c$ *y* $F^*(slave_N(i), t) = F^*(t, master_N) = d$ *si* $\lambda(t) = (i, kill)$,
  - *Si* $cap(N) = \{1, \ldots, k\}$ *entonces:*
    - $F^*(master_N, die_N) = F^*(die(R), c_N(1)) = \varepsilon$,
    - $F^*(c_N(i), kill_N(i)) = F^*(c_N(i), next_N(i)) = \varepsilon$ *para todo* $i \in cap(N)$,
    - $F^*(kill_N(i), c_N(i+1)) = F^*(next_N(i), c_N(i+1)) = \varepsilon$ *para* $i = 1, \ldots, k-1$,
    - $F^*(kill_N(k), dead_N) = F^*(next_N(k), dead_N) = \varepsilon$,
    - $F^*(slave_N(i), kill_N(i)) = F^*(kill_N(i), \overline{slave}_N(i)) = c$ *para todo* $i \in cap(N)$,
    - $F^*(next_N(i), \overline{slave}_N(i)) = F^*(\overline{slave}_N(i), next_N(i)) = c$ *para todo* $i \in cap(N)$.
- $\lambda^*(t) = \begin{cases} \lambda(t) \text{ si } \lambda(t) \in \mathcal{A} \cup S? \\ s! \text{ si } \lambda(t) = (i, s!) \\ kill! \text{ si } \lambda(t) = (i, kill) \text{ o } t = kill(i) \\ kill? \text{ si } t = die_N \\ imprint(o')! \text{ si } \lambda(t) = (i, imprint(o') \\ imprint(o)? \text{ si } t = impr_N \text{ y } N \text{ tiene tipo } o \\ next \in \mathcal{A} \text{ si } t = next_N(i) \end{cases}$

Además, dado (M,R) *definimos* $(M,R)^*$ *de la siguiente manera. Sea* $\overline{Id} = \{\eta_i^j\} \cup \{\eta_N \mid N$ *tiene tipo* $\top\}$ *para* $i = 1, \ldots, n$ *y* $j = 1, \ldots, cap(N_i)$ *y* $G(R^*)$ *el resultado de etiquetar los arcos de* $G(R)$ *con identificadores de* $\overline{Id}$*, de manera que si* η_i^j *etiqueta un arco que va de* N *a* N' *entonces* $N = N_i$*. Entonces tomamos:*

- $(M,R)^*(p) = M(p)$ *si* $p \in P$,
- $(M,R)^*(c_N(i)) = 0$ *para todo* N *e* $i \in cap(N)$,
- $(M,R)^*(master_N) = \begin{cases} \{\eta_i^j\} \text{ si } N \text{ es } j\text{-esclavo de } N_i \text{ en } R \\ \{\eta_N\} \text{ si } N \text{ tiene tipo } \top, \end{cases}$
- $(M,R)^*(slave_{N_i}(j)) = \begin{cases} \{\eta_i^j\} \text{ si } N_i \text{ tiene un } j\text{-esclavo} \\ \emptyset \text{ en otro caso} \end{cases}$
- $(M,R)^*(slave_{N_i}(j)) = \{\eta_i^j\} \setminus (M,R)^*(slave_{N_i}(j))$.

Queremos ver que $\mathcal{N}^*$ simula a $\mathcal{N}$. Primero veamos que tenemos las siguientes simulaciones paso a paso.

Lema 11.4.4 *Se cumplen las siguientes propiedades:*

- *Si* $\lambda(t) \in \mathcal{A}$ *entonces* $(M,R)[t\rangle(M',R)$ *si y sólo si* $(M,R)^*[t(\sigma)\rangle(M',R)^*$.
- *Si* $\lambda(t_1) = s?$ *y* $\lambda(t_2) = (i, s!)$ *entonces* $(M,R)[(t_1,t_2)\rangle(M',R)$ *si y sólo si* $(M,R)^*[(t_1,t_2)(\sigma)\rangle(M',R)^*$.
- *Si* $\lambda(t) = (i, imprint(o_{N'}))$ *entonces se tiene que* $(M,R)[t\rangle(M',R')$ *si y sólo si* $(M,R)^*[(t, impr_{N'})(\sigma)\rangle(M',R')^*$.

Demostración.

- Este caso es inmediato teniendo en cuenta que ${}^\bullet t = \{p \mid (p,t) \in F\} \cup \{master_N\}$, $t^\bullet = \{p \mid (t,p) \in F\} \cup \{master_N\}$ y $F^*(master_N, t) = F^*(t, master_N)$. Entonces t está activada en $\mathcal{N}$ si y sólo si N está viva y todas sus precondiciones tiene tokens, lo que ocurre según la observación anterior si y sólo si todas las precondiciones de t en $\mathcal{N}^*$ tienen tokens, y los marcajes obtenidos están relacionados por $(\ )^*$.
- En este caso ${}^\bullet t_1 = \{p \mid (p,t_1) \in F\} \cup \{slave_N(i), master_N\}$, $t_1^\bullet = \{p \mid (t_1,p) \in F\} \cup \{slave_N(i), master_N\}$, ${}^\bullet t_2 = \{p \mid (p,t_2) \in F\} \cup \{master_{N'}\}$, $t_2^\bullet = \{p \mid (t_2,p) \in F\} \cup \{master_{N'}\}$ y $F^*(t_1, slave_N(i)) = F^*(slave_N(i), t_1) = F^*(master_{N'}, t_2) = F^*(t_2, master_{N'})$. Entonces N y N' están vivas y N' es i-esclava de N si y sólo si los lugares $master_N$, $master_{N'}$ y $slave_N(i)$ tienen un token y el token en $master_{N'}$ coincide con el token en $slave_N(i)$. Entonces (t_1,t_2) se puede disparar en $\mathcal{N}$ si y sólo si se puede disparar en $\mathcal{N}^*$ con modo σ que toma los únicos valores posibles según la observación anterior, alcanzando marcajes también relacionados por $(\ )^*$.
- Si la transición es de impresión, entonces ${}^\bullet t = \{p \mid (p,t) \in F\} \cup \{master_N, \overline{slave}_N(i)\}$, $t^\bullet = \{p \mid (t,p) \in F\} \cup \{master_N, slave_N(i)\}$, ${}^\bullet impr_{N'} = \{dead_{N'}\}$ y $impr_{N'}^\bullet = \{master_{N'}\}$. Por tanto $(M,R)[t\rangle(M',R')$ si y sólo si N está viva y N' muerta segun R, $M' = M - {}^\bullet t + t^\bullet$ y R' extiende R con $R(N,i) = N'$, lo que, según las observaciones anteriores, ocurre si y sólo si $(M,R)^*$ tiene todas sus precondiciones con tokens, el token que había en $\overline{slave}_N(i)$ acaba en los lugares $slave_N(i)$ y $master_{N'}$ y el resto queda como indiquen los arcos, que coinciden con la red original. Esto último ocurre si y sólo si $(M,R)^*[(t, impr_{N'})(\sigma)\rangle(M',R')^*$, donde σ es el único modo posible y la tesis se sigue.

□

Por culpa de las simulaciones de las transiciones *kill* perdemos la propiedad de que la simulación sea un isomorfismo. Veamos ahora que si $(M,R)[t\rangle(M',R')$ con $\lambda(t) = (l, kill)$ entonces $(M,R)^* \to^* (M',R')$. Antes probemos un lema técnico. Dado un marcaje M vamos a denotar M_N a la restricción de ese marcaje a los lugares de N, de manera que podemos ver un marcaje M como el conjunto $\{M_N \mid N \in \mathcal{N}\}$, y análogamente para $\mathcal{N}^*$. Además, si $cap(N) = \{1, \ldots, k\}$ definimos para $i = 1, \ldots, k+1$, identificando $c_N(k+1)$ con $dead_N$,

$$M_N^i(p) = \begin{cases} 1 \text{ si } p = c_N(i) \\ 0 \text{ si } p = c_N(j) \text{ con } i \neq j \\ \emptyset \text{ si } p = master_N \text{ o } p = slave_N(j) \text{ con } i < j \\ \{\eta_N^i\} \text{ si } p = \overline{slave}_N(j) \text{ con } i < j \\ M_N(p) \text{ en otro caso} \end{cases}$$

Entonces el siguiente lema afirma intuitivamente que si a una componente en la simulación le ordenan cometer seppuku entonces lo hace, matando a todos sus esclavos.

Lema 11.4.5 *Sea R una función de dependencia tal que $G(R)$ es un árbol con raíz N. Dado un marcaje M de todas las componente de $\mathcal{N}$ que están vivas en R sea $\overline{M}$ el marcaje de $\mathcal{N}^*$ tal que $\overline{M}_N = M_N^1$ y $\overline{M}_{N'} = (M_{N'}, R)^*$. Entonces se tiene que $\overline{M} \to^* (M, \iota)^*$.*

Demostración. Procedemos por inducción sobre la profundidad de R. Si la profundidad es cero entonces N no tiene esclavos según R, es decir, $\overline{slave}_N(i) = \{\eta_i\}$ y $slave_N(i) = \emptyset$ para todo $i = 1, \ldots, k$. Entonces se tiene que $\overline{M} = M_N^1$ y se puede disparar la secuencia de transiciones

$$M_N^1[next_N(1)(\sigma_1)\rangle M_N^2 \ldots M_N^k[next_N(k)(\sigma_k)\rangle M_N^{k+1} = (M, \iota)^*.$$

Claramente $M_N^i[next_N(i)(\sigma_i)\rangle M_N^{i+1}$ para $i = 1, \ldots, k$ con $\sigma_i(c) = \eta_i$ pues en M_N^i el lugar $\overline{slave}_N(i)$ está marcado por $\{\eta_i\}$. Además, M_N^{k+1} marca el lugar $dead_N$, todos los $\overline{slave}_N(i)$ y ninguno de los $slave_N(i)$, de modo que representa al árbol vacío y $M_N^{k+1} = (M, \iota)^*$.

Supongamos ahora que la profundidad de R no es nula, es decir, que N tiene esclavos. Sean $i_1, \ldots, i_h \subseteq cap(N)$ tales que N tiene un i_j-esclavo, que denotaremos mediante N_{i_j}. Entonces R tiene subárboles $R_1, \ldots, R_h$ con raíces $N_{i_1}, \ldots, N_{i_h}$ respectivamente, todos de profundidad estrictamente menor que la de R. Sea M_j las restricción de M a las componentes de R_j. El marcaje M_N^1 tiene un token η_i en $slave_N(i)$ si y sólo si $i = i_j$ para algún j, lo que ocurre si y sólo si no lo tiene en $\overline{slave}_N(i)$. Entonces se puede disparar la secuencia $u_1, \ldots, u_k$ donde $u_i = (kill_i, die_{N_{i_j}})$ si $i = i_j$ para algún j y $u_i = next_N(i)$ en otro caso. Entonces, siguiendo esa secuencia, desde $\overline{M}$ se llega a $\{M_N^{k+1}, \overline{M}_1, \ldots, \overline{M}_h\}$. Cada par $\overline{M}_j$ está en las hipótesis del lema con un árbol R_j de profundidad menor a R, ya que $(\overline{M}_j)_{N_{i_j}} = M_{N_{i_j}}^1$ y $(\overline{M}_j)_{N'} = ((M_1)_{N'}, R_i)^*$. Entonces se tiene que $\overline{M}_j \to^* (M_j, \iota)^*$ para todo $j = 1, \ldots, h$. Además, todas las transiciones de todos los cómputos son independientes entre sí (lo son de distintas componentes), y todas ellas independientes de las transiciones u_l para $l > j$. Entonces podemos disparar sucesivamente todas esas secuencias de cómputos tras la que ya teníamos, obteniendo así $\{M_N^{k+1}, (M_1, \iota)^*, \ldots, (M_h, \iota)^*\} = (M, \iota)^*$, como queríamos demostrar.

□

Lema 11.4.6 *Si $(M, R)[t\rangle(M', R')$ con $\lambda(t) = (l, kill)$ entonces $(M, R)^* \to^* (M', R')$*

Demostración. Supongamos que t es una transición de la componente N' con capacidad k y que N es la l-esclava de N'. Como N está viva se tiene que hay un nodo de $G(R)$ etiquetado por N y, por tanto hay un token η en $slave_{N'}(l)$ y en $master_N$ y otro η' en $master_{N'}$ según $(M, R)^*$.

Llamemos $\overline{R}$ al subárbol de R que tiene a N como raíz (de modo que R' es el resultado de eliminar $\overline{R}$ de R) y $M|_{\overline{R}}$ a $(M, R)^*$ restringido a las componentes vivas en $\overline{R}$, de manera que $(M, R)^* = \{(M_{N'}, R)^*, M|_{\overline{R}}\} \cup \mathcal{M}$, donde $\mathcal{M}$ es el marcaje del resto de las componentes. Entonces se puede producir el disparo $(M, R)^*[(t, die_N)(\sigma)\rangle\{(M'_{N'}, R')^*, \overline{M}|_{\overline{R}}\} \cup \mathcal{M}$ con $\sigma(c) = \eta$, $\sigma(d) = \eta'$ y $M'_{N'} = M_{N'} - {}^\bullet t + t^\bullet$, donde $(\overline{M}|_{\overline{R}})_{N''} = (M_{N''}, R)^*$ para todo N'' viva en $\overline{R}$. El marcaje $\overline{M}|_{\overline{R}}$ está en las hipótesis del lema anterior porque N es la raíz de $\overline{R}$ y $(\overline{M}|_{\overline{R}})_N = M_N^1$, por lo que obtenemos que $\overline{M}_{\overline{R}} \to^* (M|_{\overline{R}}, \iota)^*$. Entonces $(M, R)^*[(t, die_N)(\sigma)\rangle\{(M'_{N'}, R')^*, \overline{M}|_{\overline{R}}\} \cup \mathcal{M} \to^* \{(M'_{N'}, R')^*, (M|_{\overline{R}}, \iota)^*\} \cup \mathcal{M} = (M', R')^*$, como queríamos demostrar.

□

Proposición 11.4.7 *Los sistemas MSPN con identificadores abstractos simulan monótonamente a los sistemas TSA.*

Demostración. Veamos que $\mathcal{N}^*$, como lo hemos definido en la definición 11.4.3, simula a $\mathcal{N}$. Por los lemas 11.4.4 y 11.4.6 se tiene que si $(M, R) \to (M', R')$ entonces $(M, R)^* \to^* (M', R')^*$. Supongamos ahora que $(M, R)^* \to^* \overline{M}$ y que todos los estados intermedios son inestables. Veamos por inducción sobre el número de disparos de pares de la forma (t, die_N) con $\lambda(t) = (i, kill)$ en ese cómputo que $\overline{M} \to^* (M', R')^*$ donde en ese cómputo no hay disparos de esa forma y $(M, R) \to^* (M', R')$. Si no hay disparos de esa forma entonces todos son de los recogidos en el lema 11.4.4 y por ese mismo lema se tiene que en ese caso $\overline{M} = (M', R')^*$ y $(M, R) \to^* (M', R')$. Supongamos ahora que (t, die_N) es el último disparo de esa forma que aparece en el cómputo, es decir, $(M, R)^* \to^* \overline{M}' \to^* \overline{M}$. En el subcómputo $(M, R)^* \to^* \overline{M}'$ podemos aplicar la hipótesis de inducción para obtener que existe (M'', R'') tal que $\overline{M}' \to^* (M'', R'')^*$ (sin disparos $(t', die_{N'})$) y $(M, R) \to^* (M'', R'')$. Se cumple que todos los disparos detrás de $(t, die_{N'})$ son independientes de los del cómputo $\overline{M}' \to^* (M'', R'')^*$, por lo que se pueden disparar después de (M'', R''). Así que basta ver que si $(M, R)[(t, die_N)\rangle\overline{M} \to^* \overline{M}'$ donde en $\overline{M} \to^* \overline{M}'$ no hay más disparos de esa forma entonces existe (M', R') tal que $\overline{M}' \to^* (M', R')^*$ y $(M, R) \to^* (M', R')$. Todas las transiciones en $\overline{M} \to^* \overline{M}'$ son independientes de (t, die_N), por lo que existe $\overline{M}''$ tal que $(M, R) \to^* \overline{M}''[(t, die_N)\rangle\overline{M}'$. Como las transiciones en $(M, R) \to^* \overline{M}''$ no son de la forma (t, die_N) se tiene por el lema 11.4.4 que $\overline{M}'' = (M'', R'')^*$ y $(M, R) \to^* (M'', R'')$. Hemos vuelto a reducir el problema al caso $(M, R)[(t, die_N)\rangle\overline{M}$, que se puede completar siguiendo el cómputo que veíamos en la demostración de los lemas 11.4.5 y 11.4.6 y la tesis se sigue.

Para concluir, $(M, R) \sqsubseteq (M', R') \Leftrightarrow (M, R)^* \sqsubseteq (M', R')^*$ y todos los marcajes estables de $\mathcal{N}^*$ cumple para cada N que todos los lugares $c_N(i)$ están vacios y que o bien tienen marcado el lugar $master_N$ o bien tienen marcado el lugar $dead_N$. Por otro lado, todos los alcanzables inestables tienen a $master_N$ y $dead_N$ simultáneamente vacíos y $c_N(i)$ marcado para algún N, por lo que no hay marcajes alcanzables inestables mayores que uno estable y la tesis se sigue.

□

La figura 11.12 muestra la implementación del sistema TSA de la figura 11.8, si es una componente de tipo $\top$.

La construcción anterior tiene la ventaja respecto a la vista en la proposición 11.4.2 que preserva la estructura basada en componentes de los sistemas TSA (aunque para formalizarlo habría que trabajar con una noción más refinada de simulación). Además, esta construcción es monótona respecto a un orden más flexible de los sistemas TSA, que relaciona no sólo marcajes con la misma función de dependencia, sino que permitimos que la función de dependencia del marcaje mayor extienda al otro, es decir, $(M, R) \sqsubseteq' (M', R')$ si $M \subseteq M'$ y R' extiende a R.

La anterior demostración se aprovecha del hecho de que estamos viendo los sistemas TSA como sistemas cerrados, de modo que podemos analizar estática y extensamente todos sus participantes. En particular, podemos comprobar (y forzar) que las componentes impresas, es decir, que reciben una clave de impresión de su amo, eliminan esa misma clave al morir. Por tanto, las componentes pueden reutilizar esas claves en impresiones futuras, sabiendo que nunca van a ser usadas de manera ilícita por antiguos esclavos.

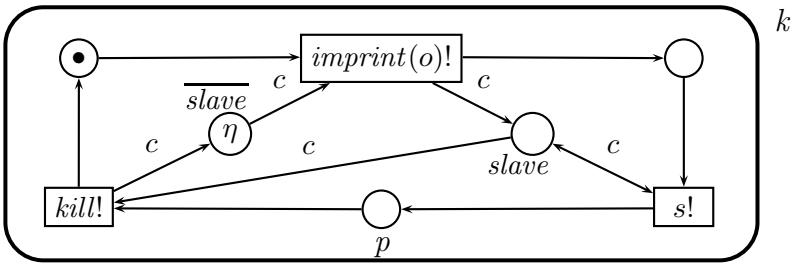

Figura 11.12: Implementación del sistema TSA de la figura 11.8

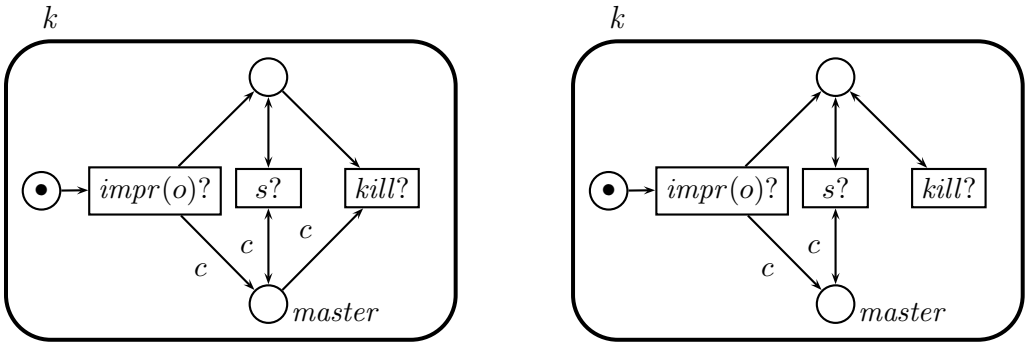

Figura 11.13: Entornos para el sistema TSA de la figura 11.8

Sin embargo, como ya hemos comentado repetidamente en este trabajo, en el campo de la computación ubicua no es muy realista suponer que tratamos con sistemas cerrados. En este contexto queremos que nuestros servicios se ofrezcan de manera global y uniforme, de modo que algunos de los participantes que imprimen o son impresos pueden no ser parte del sistema, sino parte del entorno del mismo. En tal caso, ya no podríamos comprobar, y menos aún forzar, que se eliminan las claves de impresión después de una muerte, de manera que si las reutilizáramos, los participantes del entorno podrían usar claves antiguas para interactuar con una componente del sistema de manera ilegal.

Consideremos de nuevo el ejemplo de la componente TSA de la figura 11.8 o, más exactamente, su implementación como sistema MSPN (sin creación de nombres) en la figura 11.12. Esta componente intenta imprimir un dispositivo de tipo o, usar un servicio s ofrecido por este dispositivo y finalmente matarlo, pasando así de nuevo a su estado inicial. Si la componente está en un entorno como el que se muestra a la izquierda de la figura 11.13, entonces su comportamiento sería el que se espera de ella. En efecto, después de morir, esa componente tira la clave de impresión (el identificador del lugar *master*), de modo que ésta se puede reutilizar de manera segura.

Supongamos ahora que el sistema se encuentra en un entorno como el de la derecha de la figura 11.13, que es prácticamente igual que el de la izquierda, excepto por el hecho de que no se elimina el identificador (es decir, no hay un arco del lugar *master* a la transición *kill*?) y se puede pedir el servicio s incluso después de sincronizar en la transición *kill* (que, en este caso, no tiene el efecto de matar a la componente). En tal caso la figura 11.12 sería una implementación incorrecta del sistema TSA de la figura 11.8 si consideramos nuestro sistema como un sistema abierto.

Afortunadamente, podemos hacer uso de la creación de nombres para obtener una

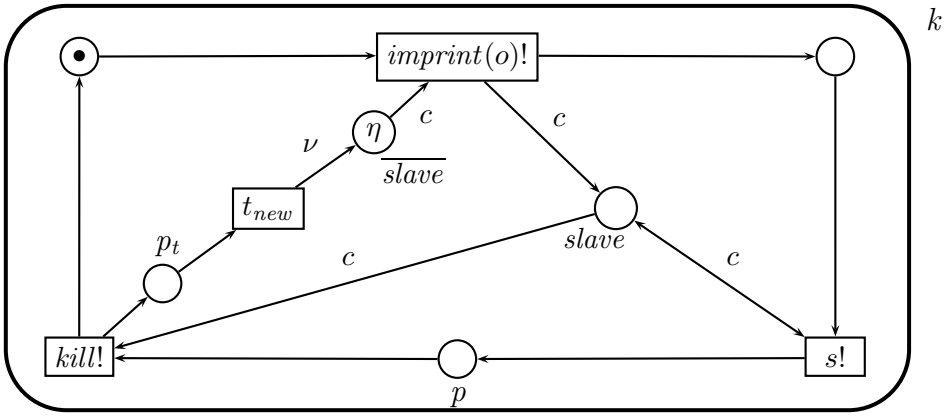

Figura 11.14: Implementación del sistema TSA de la figura 11.8

implementación correcta de los sistemas TSA, aun cuando consideráramos los sistemas MSPN bajo la semántica abierta que asumía la existencia de un entorno arbitrario definida en el capítulo 9.

Proposición 11.4.8 *Todo sistema TSA puede ser simulado por un sistema MSPN abierto.*

Demostración. La construcción del correspondiente sistema MSPN es prácticamente idéntica a la de la proposición anterior, pero con una diferencia crucial: en lugar de disponer de un conjunto predefinido de identificadores que se usan como claves para imprimir otras componentes, cada componente crea un identificador nuevo tras cada transición *kill* para usarlo en la siguiente impresión, lo que queda formalizado por el hecho de que donde teníamos $F^*(t, \overline{slave}_N(i)) = c$ ahora tenemos $F^*(t, t_{new}) = F^*(t_{new}, p_t) = \varepsilon$ y $F^*(t_{new}, \overline{slave}_N(i)) = \nu$.

□

La figura 11.14 muestra la implementación abierta del sistema TSA de la figura 11.8, análoga a la de la figura 11.12, excepto por el uso de la variable especial ν que etiqueta el arco que va al lugar $\overline{slave}$. En particular, ahora podemos ver que el entorno de la derecha de la figura 11.13 ya no puede ofrecer el servicio s después de su muerte, aunque no borre la clave antigua, ya que en una nueva impresión la componente TSA de la figura 11.14 usaría una clave de impresión distinta para comunicarse con su esclavo.

Como probamos en los capítulos 6 y 9, la clase general de sistemas MSPN (en la que permitimos creación de nombres) ya no es equivalente a la de las redes P/T, pero el recubrimiento sigue siendo decidible, incluso en el caso abierto, de modo que cualquier propiedad de la implementación que se pueda especificar en términos de recubrimiento puede ser verificada.

Capítulo 12

Conclusiones

En esta tesis hemos presentado varias extensiones muy naturales del modelo clásico de redes de Petri para la especificación de sistemas móviles, concurrentes y distribuidos, con especial atención a las características propias de la computación ubicua.

Hemos realizado una presentación progresiva de los modelos, que se ha ido viendo interrumpida por el estudio de la decidibilidad de sus propiedades, que nos han permitido distinguir las distintas capacidades expresivas de cada uno de los modelos.

En nuestra andadura, nos hemos topado con asuntos fundamentales en el campo de las redes de Petri que necesitaban aún ser estudiados, y que van más allá de la definición de un nuevo modelo para la especificación y verificación de sistemas ubicuos. La sutil diferencia entre las distintas versiones para la creación de nombres es un claro ejemplo de esto, obteniendo con ambas modelos más expresivos que las redes de Petri, sin llegar a la potencia de las máquinas de Turing, pero siendo ambos modelos incomparables entre sí. Otro ejemplo son las relaciones existentes entre la creación de nombres y la creación de componentes en el marco de las redes de Petri. De hecho, la restricción del modelo general de redes de Petri coloreadas a modelos que, aun siendo coloreados no sean Turing completos es motivo de intenso trabajo en la actualidad. En [LNO+07] los autores definen un marco general de redes de Petri en el que los tokens se toman de un conjunto ordenado linealmente, no necesariamente total, y en el que se permiten operaciones totales en los lugares (ejemplos de las cuales son la transferencia de tokens o la puesta a cero) y se prueba la decidibilidad del recubrimiento en el marco general. En el caso particular en el que el orden es el discreto, es decir, cuando no hay elementos distintos comparables, afirman obtener un modelo equivalente a nuestras APNs. Sin embargo, los autores no abordan la creación de nuevos nombres.

En la figura 12.1 se sintetizan los resultados más importantes obtenidos en la tesis. Una flecha sin etiquetar indica que el destino es una extensión semántica (no necesariamente estricta) del origen. Ese es el caso de, por ejemplo, la flecha de las APNs a las ν-APNs. Una flecha etiquetada indica una simulación monótona del origen por medio del destino, y la etiqueta referencia el resultado donde se probó dicha simulación. Señalamos las componentes fuertemente conexas del grafo resultante con las líneas de puntos. Por lo tanto, dentro de cada región hay un camino entre dos cualesquiera de los modelos, y por tanto son todos equivalentes entre sí. Hemos obtenido en total cinco regiones (sin contar las regiones unitarias que forman los sistemas g-RUN, las $\nu_{\neq}$-APNs, las ν-APNs acotadas en anchura, las MSPN abiertas y las MSPN con la semántica simbólica). Dentro de cada región todas las propiedades preservadas por la simulación monótona son equivalentes.

Centrándonos en los problemas que más hemos considerado, es decir, en la alcanzabilidad, la alcanzabilidad de submarcajes y el recubrimiento, hemos obtenido que todos estos problemas son decidibles en la región que contiene a las redes P/T; la alcanzabilidad y el recubrimiento son decidibles en la región que contiene a las redes con contadores, pero no la alcanzabilidad de submarcajes; la alcanzabilidad y el recubrimiento son decidibles en la que contiene a los sistemas RUN; el recubrimiento es decidible pero no la alcanzabilidad en la región que contiene a las ν-APNs; y el recubrimiento es indecidible para la que contiene a las máquinas de Turing. Las flechas a rayas indican que el destino es un etiquetado del origen y las flechas más tenues indican algún otro tipo de relación. Además, los modelos que se encuentran "cerca" de las regiones comparten con sus modelos los resultados de decidibilidad e indecidibilidad. Así, para las $\nu_{\neq}$-APNs y los sistemas g-RUN el recubrimiento es decidible y la alcanzabilidad es indecidible, y para las ν-APNs acotadas en anchura tanto la alcanzabilidad como el recubrimiento son decidibles.

Aunque no era el objetivo primero de esta tesis, en el capítulo 10 hemos abordado el problema de la implementación de un prototipo de herramienta para la verificación de sistemas especificados como sistemas de redes de Petri móviles. En este capítulo presentamos un algoritmo que decide el problema del recubrimiento para los sistemas MSPN. Sin embargo, ese algoritmo dista mucho de ser eficiente, motivo por el cual no se han presentado ejemplos de uso de la herramienta en ejemplos no triviales. Aunque algunas de las causas se pueden mitigar en una versión más sofisticada de la implementación, el problema radica en el hecho de que los análisis hacia atrás producen una explosión en el conjunto de marcajes predecesores, incluso con la representación basada en ideales que se usa. Pretendemos llevar a cabo un estudio más a fondo de la complejidad del algoritmo, estudiando la posibilidad de usar un análisis hacia adelante como en [ACABJ04] (o una combinación de ambos [ADMN04]) para mejorar su eficiencia.

En general, los contenidos de la tesis suponen un trabajo bastante cerrado. Quedarían algunos flecos pendientes de estudio, como se puede concluir observando el diagrama de resultados, como distinguir entre las redes ν-APNs y las $\nu_{\neq}$-APNs, los sistemas RUN de las redes con contadores y establecer la equivalencia en alguna medida entre los sistemas MSPN abiertos y los cerrados.

Hay un gran número de caminos que es necesario explorar. Estamos definiendo nuevas características que incrementen la utilidad de nuestro modelo, como un mecanismo basado en tipos para obtener estáticamente el conjunto de servicios que se ofrecen o se requieren en cada localidad del sistema. Concretamente, sería deseable poder especificar por medio de tipos el comportamiento observable de los procesos e intentar capturar de manera estática ese comportamiento, incluso cuando estemos en presencia de un entorno hostil como en el capítulo 9.

Además, es factible incrementar la expresividad del modelo mediante nuevas extensiones. Por ejemplo, estamos extendiéndolo para poder tratar propiedades de confianza, etiquetando cada proceso con su política de confianza y su estado de confianza. La evolución de dicho estado, una vez fijadas las políticas de confianza, sería una propiedad que depende solamente de la semántica original del modelo, como veíamos en el capítulo 8, de modo que no sería necesaria ninguna semántica extendida de manera ad hoc para tratarla. Esto nos conducirá a propiedades de seguridad más complejas, que pretendemos estudiar con sistemas de tipos. De hecho, estamos sorprendidos de no haber encontrado trabajos de redes de Petri tipadas, donde los tipos no nos den solamente cierta información sintáctica, como ocurre en nuestras definiciones o en cualquier modelo de redes coloreadas con varios géneros, sino información semántica como la que se obtiene en [CGG02] para estudiar el comportamiento de los procesos y, en particular, propiedades de seguridad.

También sería deseable establecer una jerarquía completa de los modelos que han ido apareciendo en esta tesis, así como de otros modelos relacionados, como [Del02, Del, Laz05, LNR05]. Esta jerarquía podría estar basada en la cantidad de información acerca del pasado que las configuraciones de un sistema recuerdan, como se veía en el capítulo 8. Por ejemplo, la diferencia entre la replicación con y sin recogida de basura es que en el primer caso obligamos a que las configuraciones recuerden el número de componentes replicadas, cosa que no hacemos en el segundo caso.

Otra mejora será la inclusión de una topología de red no plana para poder estudiar de una manera más fiel los problemas que surgen en las redes ad hoc, e incluso características continuas o probabilísticas.

También pretendemos estudiar nuestro modelo con una primitiva alternativa para la interacción de componentes, la difusión o *broadcasting* en lugar de la sincronización, ya que esta es una primitiva muy usada en el campo de la computación ubicua.

Por último, hemos encontrado interesantes conexiones entre los conceptos y resultados discutidos y obtenidos en el capítulo 7 y aquellos estudiados en [DLMS99]. En particular, el estudio de las restricciones necesarias para preservar la decidibilidad del recubrimiento para las redes ν-RN podrían estar relacionados con estudios recientes en el campo de los protocolos de seguridad [RS03, RS05].

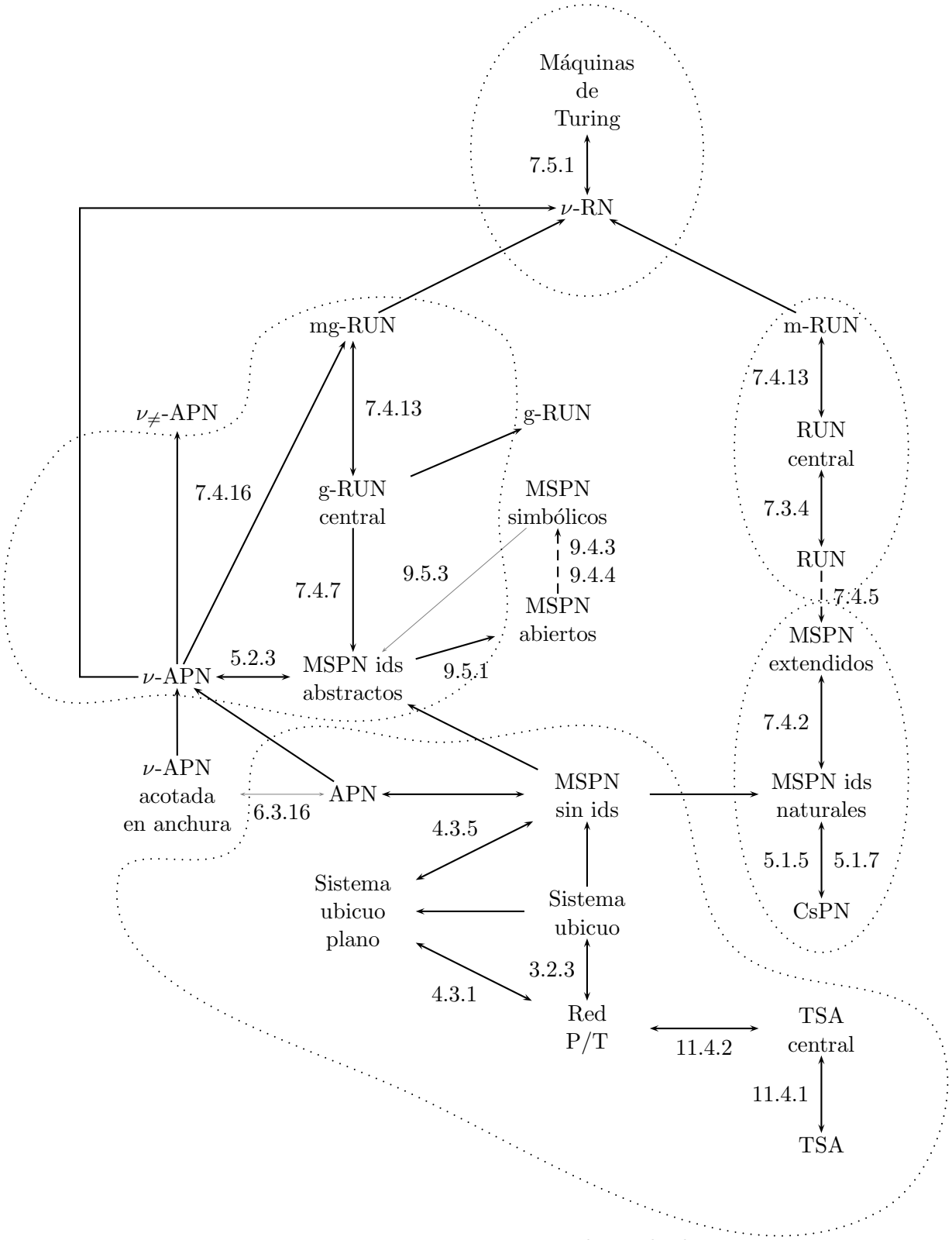

Figura 12.1: Diagrama de resultados

Apéndice A

Código del prototipo

```
***
*** XML as a Maude data type.
***
*** Version 0.1    3/11/04
***
*** We omit document prologs/DTDs.
***

fmod ATTRIBUTE-LIST is
   pr STRING .
   sort Attribute AttributeList .
   subsort Attribute < AttributeList .

*** an attribute list is a potentially nil list of String = String pairs
   op nil : -> AttributeList [ctor] .
   op _=_ : String String -> Attribute [prec 39 ctor] .
   op __ : AttributeList AttributeList -> AttributeList [ctor assoc id: nil] .
endfm


fmod XML is
   pr ATTRIBUTE-LIST .
   sorts Prolog Element Document Content .
   subsort Element String < Content .

*** a document consists of a prolog and a root element
   op doc : Prolog Element -> Document .

*** prolog may be empty
   op nil : -> Prolog .

*** an element has a name, attribute list and content
   op [_,_,_] : String AttributeList Content -> Element
     [format (d d d s d n++i n--i d)].

*** content is a potentially empty list of elements and character data
   op nil : -> Content .
```

```
   op __ : Content Content -> Content [assoc id: nil format (d ni d)] .

endfm


***********************************************************************


fmod APN is

protecting NAT .
protecting QID .

sorts PlaceB PlaceC
      MSPlacesB MSPlacesC MS2PlacesC SetMS2Places MSPlaces
      Marking MarkingSet .


subsorts PlaceB < MSPlacesB .
subsorts PlaceC < MSPlacesC < MS2PlacesC < SetMS2Places .
subsort  Marking < MarkingSet .
subsort  MSPlacesB MSPlacesC < MSPlaces .


var  n : Nat .
vars P P1 P2 : MSPlacesB .
vars C C1 C2 : MSPlacesC .
vars MM MM1 MM2 : Marking .
vars MS1 MS2 MS : MarkingSet .
vars CC CC3 CC4 CC1 CC2 : MS2PlacesC .


op emptySetB : -> MSPlacesB .
op emptySetC : -> MSPlacesC .


op _,,_ : MSPlacesB MSPlacesB -> MSPlacesB [comm assoc id: emptySetB prec 10] .
op __ : MSPlacesC MSPlacesC -> MSPlacesC [comm assoc id: emptySetC prec 10] .
op _+_ : MS2PlacesC MS2PlacesC -> MS2PlacesC [comm assoc id: emptySetC prec 20] .
op _++_ : MSPlacesB MS2PlacesC -> Marking [prec  30] .


***SET ORDER AND JOIN

op _[=B_ : MSPlacesB MSPlacesB -> Bool .
op _UB_ : MSPlacesB MSPlacesB -> MSPlacesB [comm assoc] .
op _[=B_ : MSPlacesC MSPlacesC -> Bool .
op _UB_ : MSPlacesC MSPlacesC -> MSPlacesC [comm assoc] .

eq ( emptySetB [=B P ) = true .
ceq ( P ,, P1 [=B P ,, P2 ) = ( P1 [=B P2 ) if P =/= emptySetB .
eq P1 [=B P2 = false [owise] .

eq emptySetB UB P = P .
```

```
ceq ( ( P ,, P1 ) UB ( P ,, P2 ) ) = P ,, ( P1 UB P2 ) if P =/= emptySetB .
ceq ( P ,, P1 ) UB P2 = P ,, ( P1 UB P2 ) if P =/= emptySetB [owise] .

eq ( emptySetC [=B C ) = true .
ceq ( C C1 [=B C C2 ) = ( C1 [=B C2 ) if C =/= emptySetC .
eq  C1 [=B C2 = false [owise] .

eq emptySetC UB C = C .
ceq ( ( C C1 ) UB ( C C2 ) ) = C ( C1 UB C2 ) if C =/= emptySetC .
ceq ( C C1 ) UB C2 = C ( C1 UB C2 ) if C =/= emptySetC [owise] .



***MULTISET ORDER AND JOINT

op some : MSPlacesC MS2PlacesC MS2PlacesC MS2PlacesC -> Bool .

eq some ( C , C1 , CC , CC2 ) = ( C1 [=B C ) and (CC [=C CC2) .
ceq some ( C , (C1 + CC1) , CC , CC2 ) =
         ( C1 [=B C ) and ((CC1 + CC) [=C CC2))
         or some ( C , CC1 , (C1 + CC) , CC2 )
         if C1 =/= emptySetC /\ CC1 =/= emptySetC .


op _[=C_ : MS2PlacesC MS2PlacesC -> Bool .
op _U_ : MS2PlacesC MS2PlacesC -> MS2PlacesC [comm].


eq emptySetC [=C CC = true .
ceq CC [=C emptySetC = false if CC =/= emptySetC .


ceq C1 [=C C2 + CC = (C1 [=B C2) or (C1 [=C CC)
                   if C1 =/= emptySetC /\ C2 =/= emptySetC .
ceq ((C1 + CC1) [=C (C2 + CC2)) = ((C1 + CC1) [=C CC2)
                                  or some(C2, (C1 + CC1) , emptySetC , CC2)
                                  if C1 =/= emptySetC /\ C2 =/= emptySetC /\
                                  CC1 =/= emptySetC /\ CC2 =/= emptySetC .
ceq ((C1 + CC1) [=C C2) = false
               if C1 =/= emptySetC /\ CC1 =/= emptySetC /\ C2 =/= emptySetC .


op someU : MSPlacesC MS2PlacesC MS2PlacesC MS2PlacesC -> MS2PlacesC .

eq someU ( C , C1 , CC , CC2 ) = ( C1 UB C ) +2 (CC U CC2) .
eq someU ( C , (C1 + CC1) , CC , CC2 ) = (( C1 UB C ) +2 ((CC1 + CC) U CC2)) *B
          someU ( C , CC1 , (C1 + CC) , CC2 ) .



eq emptySetC U CC = CC .

ceq C1 U (C2 + CC) = ((C1 UB C2) + CC) *B (C2 +2 (C1 U CC))
 if C1 =/= emptySetC /\ C2 =/= emptySetC /\ CC =/= emptySetC .
```

```
ceq ((C1 + CC1) U (C2 + CC2)) =
           ((C2 +2 ((C1 + CC1) U CC2)) *B someU(C2, (C1 + CC1) , emptySetC , CC2))
           if C1 =/= emptySetC /\ C2 =/= emptySetC /\ CC1 =/= emptySetC
           or CC2 =/= emptySetC .

ceq C1 U C2 = (C1 UB C2) *B (C1 + C2)
    if C1 =/= emptySetC /\ C2 =/= emptySetC .



***MARKING ORDER AND JOIN


op _[=_ : Marking Marking -> Bool .
op _U_ : Marking Marking -> Marking [comm] .

eq ((P1 ++ CC1) [= (P2 ++ CC2)) = ((P1 [=B P2) and (CC1 [=C CC2)) .
eq (P1 ++ CC1) U (P2 ++ CC2) = (P1 UB P2) ++ (CC1 U CC2) .



***NORMAL FORM IN MARKING SETS


vars SC SC1 SC2 : SetMS2Places .

op emptyMarkingSet : -> MarkingSet .
op emptySetMS : -> SetMS2Places .

op _*_ : MarkingSet MarkingSet -> MarkingSet [comm assoc id: emptyMarkingSet] .
op _*B_ : SetMS2Places SetMS2Places -> SetMS2Places [comm assoc id: emptySetMS] .


ceq P ++ (CC1 *B SC2) = ( P ++ CC1) * ( P ++ SC2) if SC2 =/=
emptySetMS . ceq CC1 *B CC2 = CC1 if CC1 [=C CC2 .

op _+2_ : SetMS2Places SetMS2Places -> SetMS2Places [comm] .

ceq emptySetMS +2 SC = emptySetMS if SC =/= emptySetMS .
eq CC1 +2 CC2 = CC1 + CC2 .
ceq CC1 +2 (CC2 *B SC) = (CC1 + CC2) *B (CC1 +2 SC) if SC =/= emptySetMS .
ceq (CC1 *B SC1) +2 (CC2 *B SC2) = (CC1 + CC2 ) *B (CC1 +2 SC2) *B
     (CC2 +2 SC1) *B (CC2 +2 SC1) if SC1 =/= emptySetMS or SC2 =/= emptySetMS .

endfm


*******************************************************************************


fmod REVERSE-RULES is

protecting META-LEVEL .
```

```
op reverse : SModule -> SModule .

var A : Qid .
var B : ImportList .
var C : SortSet .
var D : SubsortDeclSet .
var E : OpDeclSet .
var F : MembAxSet .
var G : EquationSet .
var Rules : RuleSet .
var R : Rule .
vars T1 T2 : Term .
var AS : AttrSet .

eq reverse( mod A is B sorts C . D E F G Rules endm ) =
   mod reverse(A) is B sorts C . D E F G reverseRules(Rules) endm .

op reverse : Qid -> Qid .
eq reverse(A) = qid(string(A) + "-REVERSE") .

op reverseRules : RuleSet -> RuleSet .
eq reverseRules( R Rules ) = reverseRule( R ) reverseRules( Rules ) .
eq reverseRules( none ) = none .

op reverseRule : Rule -> Rule .
eq reverseRule ( rl T1 => T2 [ AS ] . ) = ( rl T2 => T1 [ AS ] . ) .

endfm



**********************************************************************


fmod NETS is

protecting APN .


sorts At PId Place PlaceL PlaceId PlaceSet .
sort Colour .
sorts TId TransitionAut TransitionGo TransitionNew TransitionSync
      Transition TransitionSet .
sorts Services Lambda Go New Aut Sync .
sorts VarEps VarL VarId Var VarSet VarPart VarPartSet .
sorts Flow Net NetSystem .
sort ResultPart .

subsorts Aut Sync New Go < Lambda .
subsort String < PId TId Services VarEps VarL VarId .
subsorts PlaceB  PlaceId PlaceL < Place < PlaceSet .
subsorts TransitionAut TransitionGo TransitionNew TransitionSync <
         Transition < TransitionSet .
subsorts VarEps VarL VarId < Var < VarSet < VarPart < VarPartSet .
```

```
subsorts Place Flow < Net < NetSystem .
*** Place < Net in order to admit isolated places
subsort At < PlaceL < PlaceC .
subsort PlaceId < PlaceC .

op @{_} : Nat -> At [prec 5] .
op @{_} : Qid -> At [prec 5] .

ops black locality identifier : -> Colour .
op new : -> New .
op go : -> Go .
op tau : -> Aut .

op _? : Services -> Sync .
op _! : Services -> Sync .


***Variables

op eps : -> VarEps .
op _&_ : VarSet VarSet -> VarSet [assoc comm prec 10] .
op emptyV : -> VarSet .
eq emptyV & x = x .
eq V & V = V .

op _;_ : VarPart VarPart -> VarPart [assoc comm prec 20] .
op _;;_ : VarPartSet VarPartSet -> VarPartSet [assoc comm prec 30] .

eq Prs ;; Prs = Prs . ***En Maude no se permite assoc + idem

vars V V1 V2 : VarSet .
vars Pr1 Pr Pr2 : VarPart .
vars Prs Prs1 Prs2 : VarPartSet .
vars x y z x1 x2 y1 y2 z1 z2 : Var .


***ORDER IN SETS OF VARIABLES

op _<<_ : VarSet VarSet -> Bool .

eq emptyV << V = true .
eq V << V = true .
eq V << V & V1 = true .
eq V & V1 << V & V2 = V1 << V2 .
eq V1 << V2 = false [owise] .


vars s : Services .

op ^_ : Sync -> Sync .
eq ^(s ?) = s ! .
eq ^(s !) = s ? .
```

```
op [_,_] : TId Aut -> TransitionAut .
op [_,_] : TId Go -> TransitionGo .
op [_,_] : TId New -> TransitionNew .
op [_,_] : TId Sync -> TransitionSync .


op `(_,_`) : PId Colour -> Place .

var pid : PId .

mb (pid,black) : PlaceB .
mb (pid,locality) : PlaceL .
mb (pid,identifier) : PlaceId .



op _-_->_ : PlaceB VarEps Transition -> Flow .
op _-_->_ : PlaceL VarL Transition -> Flow .
op _-_->_ : PlaceId VarId Transition -> Flow .



op _-_->_ : Transition VarEps PlaceB -> Flow .
op _-_->_ : Transition VarL PlaceL -> Flow .
op _-_->_ : Transition VarId PlaceId -> Flow .


op TSet : TransitionSet TransitionSet -> TransitionSet [assoc comm id: emptyT] .
op PSet : PlaceSet PlaceSet -> PlaceSet [assoc comm id: emptyP] .
op emptyT : -> TransitionSet .
op emptyP : -> PlaceSet .

var TS : TransitionSet .
var PS : PlaceSet .

eq TSet(t,t) = t . eq PSet(p,p) = p .

op emptyNet : -> Net .
op _,_ : Net Net -> Net [assoc comm id: emptyNet prec 10] .

op _-_ : NetSystem NetSystem -> NetSystem [assoc comm id: emptyNet prec 30] .


***AUXILIARY OPERATIONS



***Union of sets T, P and F

vars t t1 t2 : Transition .
vars ts ts1 ts2 : TransitionSync .
var p : Place .
vars N N1 N2 : Net .
vars NS NS1 NS2 : NetSystem .
```

```
op T : NetSystem -> TransitionSet .

eq T(emptyNet) = emptyT .
eq T(p) = emptyT .
eq T(((p - x -> t) , N) - NS ) = TSet(t,T(N - NS)) .
eq T(((t - x -> p), N) - NS) = TSet(t,T(N - NS)) .



op member : Transition NetSystem -> Net .

ceq member(t,N) = N if member(t,T(N)) .
ceq member(t,NS1 - NS2) = if member(t,T(NS1)) then
                          member(t,NS1) else member(t,NS2) fi
         if NS1 =/= emptyNet and NS2 =/= emptyNet .

op member : Transition TransitionSet -> Bool .

eq member(t,TSet(t,TS))= true .
ceq member(t,TSet(t1,TS))= member(t,TS) if t =/= t1 .
eq member(t,emptyT) = false .


vars tid tid1 tid2 : TId .
var lm : Lambda .
vars s1 s2 : Sync .


***Output and input variables


op O : Transition NetSystem -> VarSet .

eq O(t,p) = emptyV .
eq O(t,(p - x -> t)) = x .
eq O(t,(t2 - x ->p)) = emptyV .
ceq O(t,(N1 , N2)) = O(t,N1) & O(t,N2) if N1 =/= emptyNet /\ N2 =/= emptyNet .
ceq O(t,(NS1 - NS2)) = O(t,NS1) & O(t,NS2)
                                     if NS1 =/= emptyNet /\ NS2 =/= emptyNet .

op I : Transition NetSystem -> VarSet .

eq I(t,p) = emptyV .
eq I(t,(p - x -> t2)) = emptyV .
eq I(t,(t - x -> p)) = x .
ceq I(t,(N1 , N2)) = I(t,N1) & I(t,N2) if N1 =/= emptyNet /\ N2 =/= emptyNet .
ceq I(t,(NS1 - NS2)) = I(t,NS1) & I(t,NS2)
                                     if NS1 =/= emptyNet /\ NS2 =/= emptyNet .

op Vars : Transition NetSystem -> VarSet .
```

```
eq Vars(t,NS) = O(t,NS) & I(t,NS) .


***Coloured output and input variables


op OC : Transition Colour NetSystem -> VarSet .

eq OC(t,c,p) = emptyV .

eq OC(t,c,((pid,c) - x -> t)) = x .
ceq OC(t,c,((pid,c2) - x -> t2)) = emptyV if t =/= t2 or c =/= c2 .
eq OC(t,c,(t2 - x -> p)) = emptyV .
ceq OC(t,c,(N1 , N2)) = OC(t,c,N1) & OC(t,c,N2)
                        if N1 =/= emptyNet /\ N2 =/= emptyNet  .
ceq OC(t,c,(NS1 - NS2)) = OC(t,c,NS1) & OC(t,c,NS2)
                        if NS1 =/= emptyNet /\ NS2 =/= emptyNet .

var I : Qid .

eq OC(t,locality,(@{I} - atVar -> t)) = atVar .
eq OC(t,identifier,(@{I} - atVar -> t)) = emptyV .
ceq OC(t,c,(@{I} - atVar -> t2)) = emptyV if t =/= t2 .

op IC : Transition Colour NetSystem -> VarSet .

eq IC(t,c,p) = emptyV .
eq IC(t,c,(p - x -> t2)) = emptyV .
eq IC(t,c,(t - x -> (pid,c))) = x .
ceq IC(t,c,(t2 - x -> (pid,c2))) = emptyV if t =/= t2 or c =/= c2 .
ceq IC(t,c,(N1 , N2)) = IC(t,c,N1) & IC(t,c,N2)
                                if N1 =/= emptyNet /\ N2 =/= emptyNet .
ceq IC(t,c,(NS1 - NS2)) = IC(t,c,NS1) & IC(t,c,NS2)
                                if NS1 =/= emptyNet /\ NS2 =/= emptyNet .
eq IC(t,locality,(t - atVar -> @{I})) = atVar .
eq IC(t,identifier,(t - atVar -> @{I})) = emptyV .
ceq IC(t,c,(t2 - atVar -> @{I})) = emptyV if t =/= t2 .


op OCC : Transition NetSystem -> VarSet .
eq OCC(t,NS) = OC(t,locality,NS) & OC(t,identifier,NS) .

op OCC : TransitionSync TransitionSync NetSystem -> VarSet .
eq OCC(ts1,ts2,NS) = OCC(ts1,NS) & OCC(ts2,NS) .

op ICC : Transition NetSystem -> VarSet .
eq ICC(t,NS) = IC(t,locality,NS) & IC(t,identifier,NS) .

op ICC : TransitionSync TransitionSync NetSystem -> VarSet .
eq ICC(ts1,ts2,NS) = ICC(ts1,NS) & ICC(ts2,NS) .

op VarsC : Transition Colour NetSystem -> VarSet .
eq VarsC(t,c,NS) = OC(t,c,NS) & IC(t,c,NS) .
```

```
op VarsC : TransitionSync TransitionSync Colour NetSystem -> VarSet .
eq VarsC(t1,t2,c,NS) = VarsC(t1,c,NS) & VarsC(t2,c,NS) .


*** Preconditions and postconditions

vars c c1 c2 : Colour .

op Pre : Transition Colour NetSystem -> MSPlacesC .

eq Pre(t,c,p) = emptySetC .
eq Pre(t,c,((pid,c) - x -> t)) = (pid,c) .
ceq Pre(t,c,((pid,c2) - x -> t2)) = emptySetC if t =/= t2 or c =/= c2 .
eq Pre(t,locality,(@{I} - atVar -> t)) = @{I} .
eq Pre(t,identifier,(@{I} - atVar -> t)) = emptySetC .
ceq Pre(t,c,(@{I} - atVar -> t2)) = emptySetC if t =/= t2 .
eq Pre(t,c,(t2 - x -> p)) = emptySetC .
ceq Pre(t,c,(N1 , N2)) = Pre(t,c,N1) Pre(t,c,N2)
                                    if N1 =/= emptyNet /\ N2 =/= emptyNet .
ceq Pre(t,c,(NS1 - NS2)) = Pre(t,c,NS1) Pre(t,c,NS2)
                                    if NS1 =/= emptyNet /\ NS2 =/= emptyNet .


op Post : Transition Colour NetSystem -> MSPlacesC .

eq Post(t,c,p) = emptySetC .
eq Post(t,c,(t - x -> (pid,c))) = (pid,c) .
ceq Post(t,c,(t2 - x -> (pid,c2))) = emptySetC if t =/= t2 or c =/= c2 .
eq Post(t,c,(p - x -> t2)) = emptySetC .
ceq Post(t,c,(N1 , N2)) = Post(t,c,N1) Post(t,c,N2)
                                   if N1 =/= emptyNet /\ N2 =/= emptyNet .
ceq Post(t,c,(NS1 - NS2)) = Post(t,c,NS1) Post(t,c,NS2)
                                   if NS1 =/= emptyNet /\ NS2 =/= emptyNet .
eq Post(t,locality,(t - atVar -> @{I})) = @{I} .
eq Post(t,identifier,(t - atVar -> @{I})) = emptySetC .
ceq Post(t,c,(t2 - atVar -> @{I})) = emptySetC if t =/= t2 .


op Pre : Transition VarSet NetSystem -> MSPlacesC .

eq Pre(t,x,p) = emptySetC .
eq Pre(t,x,(p - x -> t)) = p .
ceq Pre(t,x,(p - x2 -> t2)) = emptySetC if t =/= t2 or x =/= x2 .
eq Pre(t,x,(t2 - x2 -> p)) = emptySetC .
eq Pre(t,x,(N1 , N2)) = Pre(t,x,N1) Pre(t,x,N2) .
eq Pre(t,x,(NS1 - NS2)) = Pre(t,x,NS1) Pre(t,x,NS2) .
eq Pre(t,emptyV,NS) = emptySetC .
eq Pre(t,x & V,NS) = Pre(t,x,NS) Pre(t,V,NS) .


op Post : Transition VarSet NetSystem -> MSPlacesC .

eq Post(t,x,p) = emptySetC .
eq Post(t,x,(t - x -> p)) = p .
```

```
ceq Post(t,x2,(t2 - x -> p)) = emptySetC if t =/= t2 or x =/= x2 .
eq Post(t,x,(p - x2 -> t2)) = emptySetC .
eq Post(t,x,(N1 , N2)) = Post(t,x,N1) Post(t,x,N2) .
eq Post(t,x,(NS1 - NS2)) = Post(t,x,NS1) Post(t,x,NS2) .
eq Post(t,emptyV,NS) = emptySetC .
eq Post(t,x & V,NS) = Post(t,x,NS) Post(t,V,NS) .


op PreB : Transition VarSet NetSystem -> MSPlacesB .

eq PreB(t,x,p) = emptySetB . eq PreB(t,x,(p - x -> t)) = p .
ceq PreB(t,x,(p - x2 -> t2)) = emptySetB if t =/= t2 or x =/= x2 .
eq PreB(t,x,(t2 - x2 -> p)) = emptySetB .
eq PreB(t,x,(N1 , N2)) = PreB(t,x,N1) ,, PreB(t,x,N2) .
eq PreB(t,x,(NS1 - NS2)) = PreB(t,x,NS1) ,, PreB(t,x,NS2) .
eq PreB(t,emptyV,NS) = emptySetB .
eq PreB(t,x & V,NS) = PreB(t,x,NS) ,, PreB(t,V,NS) .


op PostB : Transition VarSet NetSystem -> MSPlacesB .

eq PostB(t,x,p) = emptySetB .
eq PostB(t,x,(t - x -> p)) = p .
ceq PostB(t,x2,(t2 - x -> p)) = emptySetB if t =/= t2 or x =/= x2 .
eq PostB(t,x,(p - x2 -> t2)) = emptySetB .
eq PostB(t,x,(N1 , N2)) = PostB(t,x,N1) ,, PostB(t,x,N2) .
eq PostB(t,x,(NS1 - NS2)) = PostB(t,x,NS1) ,, PostB(t,x,NS2) .
eq PostB(t,emptyV,NS) = emptySetB .
eq PostB(t,x & V,NS) = PostB(t,x,NS) ,, PostB(t,V,NS) .


*** Synchronizing transitions compatibility

op comp : Transition Transition NetSystem -> Bool .

eq comp ([tid1,s1],[tid2,s2],NS) = (s1 == (^ s2)) and
(ICC([tid1,s1],[tid2,s2],NS)<<OCC([tid1,s1],[tid2,s2],NS)) .
eq comp ( t1 , t2 , NS ) = false [owise] .


***Variable partition generation

op {_,_} : VarPartSet Nat -> ResultPart .

op addLeft : Var VarPartSet -> VarPartSet .

eq addLeft(x, (V1 ; V2)) = ( (x & V1) ; V2 ) .
eq addLeft(x, (Prs1 ;; Prs2)) = addLeft(x,Prs1) ;; addLeft(x,Prs2) .

op addRight : Var VarPartSet -> VarPartSet .

eq addRight(x , (V1 ; V2)) = ( V1 ; (x & V2) ) .
eq addRight(x , (Prs1 ;; Prs2)) = addRight(x,Prs1) ;; addRight(x,Prs2) .
```

```
op atVar : -> VarL .

op split : VarSet -> VarPartSet .
***conjunto de maneras de partir en 2

eq split(x & y) = ( x ; y ) .
ceq split(x & V) = (x ; V) ;; addLeft(x,Prs) ;; addRight(x,Prs)
                    if Prs := split(V) .


op merge : VarPartSet VarPartSet -> VarPartSet .

eq merge(Pr1 , Pr2) = Pr1 ; Pr2 .
eq merge(Pr1, (Pr2 ;; Prs)) = (Pr1 ; Pr2) ;; merge(Pr1,Prs) .
eq merge((Pr ;; Prs1) , Prs2) = merge(Pr,Prs2) ;; merge(Prs1,Prs2) .


op Parts : VarSet -> VarPartSet .
ceq Parts(V) = V ;; PartsI(Prs) if Prs := split(V) [owise] .
eq Parts(x) = x .
eq Parts(emptyV) = emptyV .

op PartsI : VarPartSet -> VarPartSet .

ceq PartsI(V1 ; V2) = Prs
     if Prs1 := Parts(V1) /\ Prs2 := Parts(V2) /\
         Prs := merge(Prs1,Prs2) .
eq PartsI(Prs1 ;; Prs2) = PartsI(Prs1) ;; PartsI(Prs2) .

endfm


*********************************************************************


mod NETS2APN  is

including NETS .
protecting APN .
protecting META-LEVEL .

sort TTermNat .

subsort PlaceL < Place .

var pl : Place .
vars c c1 c2 : Colour .
vars pid p1id p2id : PId .
vars lm lm1 lm2 : Lambda .
var t : Transition .
vars Pr Pr1 Pr2 : VarPart .
vars Prs Prs1 Prs2 : VarPartSet .
vars NS NS1 NS2 : NetSystem .
vars Net1 Net2 : Net .
```

```
var n m n1 n2 n3 : Nat .
var MSB MS1B MS2B : MSPlacesB .
var MS MS1 MS2 MS3 MS4 : MSPlacesC .
var T T1 T2 T'1 T'2 T3 T4 T1B T'1L T'2L T1L T1I T2B T2L T2I : Term .
var V V1 V2 : VarSet .


op addAtPlaces : NetSystem -> NetSystem .

ceq addAtPlaces(NS1 - NS2) = addAtPlaces(NS1) - addAtPlaces(NS2)
                if NS1 =/= emptyNet /\ NS2 =/= emptyNet .
eq addAtPlaces(Net1) = Net1 , addAtPlaces(ident(Net1),T(Net1)) .

op addAtPlaces : Qid TransitionSet -> Net .

eq addAtPlaces(I,emptyT) = emptyNet .
eq addAtPlaces(I,TSet(ts,TrS)) =
  (ts - atVar -> @{I}) , (@{I} - atVar -> ts) , addAtPlaces(I,TrS) .
eq addAtPlaces(I,TSet(tGo,TrS)) = addAtPlaces(I,TrS) .
eq addAtPlaces(I,TSet(tAut,TrS)) = addAtPlaces(I,TrS) .
eq addAtPlaces(I,TSet(tNew,TrS)) = addAtPlaces(I,TrS) .

***Fresh variables

op Vf : Nat -> Qid . eq Vf(n) = qid("VF" + char(n) + ":MSPlacesC") .

var R : SModule .
vars MM TM TM1 TM2 TM3 : Term .
var I : Qid .
var SO : Sort .
var SU : Substitution .
var RL : Rule .
var RLS : RuleSet .
var s : Services .
var tid : TId .

var tAut : TransitionAut .
var tNew : TransitionNew .
var tGo : TransitionGo .
vars ts t1 t2 : TransitionSync .
vars x y : Var .
vars p q : Place .
vars qid1 qid2 : Qid .

op {_,_,_} : Term Term Nat -> TTermNat .

op TrRule : Nat Transition VarPart NetSystem -> TTermNat .

ceq TrRule(n,t,V,NS) =
      {'__[Vf(n),upTerm(MS1)],'__[Vf(n),upTerm(MS2)], n + 1}
      if MS1 := Pre(t,V,NS) /\ MS2 := Post(t,V,NS) .

ceq TrRule(n,t,(Pr1 ; Pr2),NS) = {'_+_[T1,T'1],'_+_[T2,T'2],n2}
            if {T1,T2,n1}   := TrRule(n,t,Pr1,NS) /\
```

```
              {T'1,T'2,n2} := TrRule(n1,t,Pr2,NS) .

op TrRuleNew : Nat TransitionNew NetSystem -> TTermNat .
ceq TrRuleNew(n,tNew,NS) = {upTerm(emptySetC),upTerm(MS2), n + 1}
    if MS2 := Post(tNew,identifier,NS) .


*** The place @{I} characterizes the location of net labelled by I

op TrRuleGoEq : Nat TransitionGo NetSystem -> TTermNat .
ceq TrRuleGoEq(n,tGo,NS) =
  {'__[upTerm(@{qid1}),upTerm(MS1),Vf(n)],
   '__[upTerm(@{qid1}),upTerm(MS2),Vf(n)],
   n + 1}
       if qid1 := numbering(tGo,NS) /\
          MS1 := Pre(tGo,locality,NS) /\
          MS2 := Post(tGo,locality,NS) .


op TrRuleGoNEq : Nat TransitionGo NetSystem -> TTermNat .
ceq TrRuleGoNEq(n,tGo,NS) =
    {'_+_['__[upTerm(@{qid1}),Vf(n)],
     '__[upTerm(MS1),Vf(n + 1)]],
     '_+_[Vf(n),'__[upTerm(@{qid1}),
     upTerm(MS2),Vf(n + 1)]],
     n + 2 }
     if qid1 := numbering(tGo,NS) /\
        MS1 := Pre(tGo,locality,NS) /\
        MS2 := Post(tGo,locality,NS) .


op TrRuleSync : Nat Transition Transition Colour
                VarPart NetSystem -> TTermNat .

ceq TrRuleSync(n,t1,t2,c,V,NS) =
     {'__[Vf(n),upTerm(MS1 MS3)],
      '__[Vf(n),upTerm(MS2 MS4)], n + 1}
      if MS1 := Pre(t1,V,NS) /\
         MS2 := Post(t1,V,NS) /\
         MS3 := Pre(t2,V,NS) /\
         MS4 := Post(t2,V,NS) .

ceq TrRuleSync(n,t1,t2,c,(Pr1 ; Pr2),NS) =
     {'_+_[T1,T'1],'_+_[T2,T'2],n2}
     if {T1,T2,n1} := TrRuleSync(n,t1,t2,c,Pr1,NS) /\
     {T'1,T'2,n2} := TrRuleSync(n1,t1,t2,c,Pr2,NS) .

op TransitionRule : Transition VarPartSet
                    VarPartSet NetSystem -> RuleSet .

op atTransition : Transition NetSystem -> At .
eq atTransition(t,NS) = @{ident(member(t,NS))} .

op labelling : Transition -> Qid .
```

```
eq labelling([tid,lm]) = getName(upTerm(tid)) .

op minid : Qid Qid -> Qid .
eq minid(qid1,qid2) =
  if string(getName(upTerm(qid1))) <  string(getName(upTerm(qid2)))
                      then qid1 else qid2 fi .


*** We labelled each net by the minimal name of its places
*** (since they are disjoint)

op ident : Net -> Qid . ceq ident(Net1 , Net2) =
minid(ident(Net1),ident(Net2)) if Net1 =/= emptyNet /\ Net2 =/=
emptyNet . eq ident(@{I}) = I . eq ident((pid,c)) =
getName(upTerm(pid)) . eq ident((pl - x -> t)) = ident(pl) . eq
ident((t - x -> pl)) = ident(pl) .

op numbering : Transition NetSystem -> Qid . ceq numbering(t,NS) =
ident(Net1) if Net1 := member(t,NS) .



ceq TransitionRule(tAut,Pr1,Pr2,NS) = (rl T1 => T2
[label(labelling(tAut))] .) if T1B := upTerm(PreB(tAut,eps,NS)) /\
T2B := upTerm(PostB(tAut,eps,NS)) /\ {T1L,T2L,n} :=
TrRule(50,tAut,Pr1,NS) /\ {T1I,T2I,n1} := TrRule(n,tAut,Pr2,NS) /\
   T1 := '_++_['_`,`,_[T1B,'P:MSPlacesB],'_+_[T1L,T1I,'II:MS2PlacesC]] /\
   T2 := '_++_['_`,`,_[T2B,'P:MSPlacesB],'_+_[T2L,T2I,'II:MS2PlacesC]] .


*** We assume that "new" transitions do not have id preconditions.

ceq TransitionRule(tNew,Pr1,Pr2,NS) = (rl T1 => T2
[label(labelling(tNew))] .) if T1B := upTerm(PreB(tNew,eps,NS)) /\
T2B := upTerm(PostB(tNew,eps,NS)) /\ {T1L,T2L,n} :=
TrRule(50,tNew,Pr1,NS) /\ {T1I,T2I,n1} := TrRuleNew(n,tNew,NS) /\
   T1 := '_++_['_`,`,_[T1B,'P:MSPlacesB],'_+_[T1L,T1I,'II:MS2PlacesC]] /\
   T2 := '_++_['_`,`,_[T2B,'P:MSPlacesB],'_+_[T2L,T2I,'II:MS2PlacesC]] .


ceq TransitionRule(tGo,Pr1,Pr2,NS) =
   (rl T1 => T2[label(qid(string(labelling(tGo)) + "Eq"))] .)
   (rl T3 => T4 [label(qid(string(labelling(tGo)) + "NEq"))] .)
    if T1B := upTerm(PreB(tGo,eps,NS)) /\
       T2B := upTerm(PostB(tGo,eps,NS))  /\
     {T'1L,T'2L,n} := TrRuleGoNEq(50,tGo,NS) /\
     {T1L,T2L,m} := TrRuleGoEq(50,tGo,NS) /\
     {T1I,T2I,n1} := TrRule(n,tGo,Pr2,NS) /\
     T1 := '_++_['_`,`,_[T1B,'P:MSPlacesB],'_+_[T1L,T1I,'II:MS2PlacesC]] /\
     T2 := '_++_['_`,`,_[T2B,'P:MSPlacesB],'_+_[T2L,T2I,'II:MS2PlacesC]] /\
     T3 := '_++_['_`,`,_[T1B,'P:MSPlacesB],'_+_[T'1L,T1I,'II:MS2PlacesC]] /\
     T4 := '_++_['_`,`,_[T2B,'P:MSPlacesB],'_+_[T'2L,T2I,'II:MS2PlacesC]] .

eq TransitionRule(t , (Pr1 ;; Prs1) ,Prs2,NS) =
```

```
        TransitionRule(t,Pr1,Prs2,NS) TransitionRule(t,Prs1,Prs2,NS) .
eq TransitionRule(t , Prs1 ,(Pr2 ;; Prs2),NS) =
        TransitionRule(t,Prs1,Pr2,NS) TransitionRule(t,Prs1,Prs2,NS) .

op TransitionRule : Transition NetSystem -> RuleSet .

eq TransitionRule(t,NS) =
  TransitionRule(t,Parts(VarsC(t,locality,NS)),Parts(VarsC(t,identifier,NS)),NS) .

op TransitionRuleSync : TransitionSync TransitionSync
                        VarPart VarPart NetSystem -> RuleSet .

ceq TransitionRuleSync (t1,t2,Pr1,Pr2,NS) =
   (rl T1 => T2 [label(labelling(t1))] .)
   if T1B := upTerm(PreB(t1,eps,NS) ,, PreB(t2,eps,NS)) /\
      T2B := upTerm(PostB(t1,eps,NS) ,, PostB(t2,eps,NS)) /\
   {T1L,T2L,n} := TrRuleSync(50,t1,t2,locality,Pr1,NS) /\
   {T1I,T2I,n1} := TrRuleSync(n,t1,t2,identifier,Pr2,NS) /\
   T1 := '_++_['_`,`,_[T1B,'P:MSPlacesB],'_+_[T1L,T1I,'II:MS2PlacesC]] /\
   T2 := '_++_['_`,`,_[T2B,'P:MSPlacesB],'_+_[T2L,T2I,'II:MS2PlacesC]] .

eq TransitionRuleSync(t1,t2 , Prs , (Pr2 ;; Prs2),NS) =
        TransitionRuleSync(t1,t2,Prs,Pr2,NS)
        TransitionRuleSync(t1,t2,Prs,Prs2,NS) .

eq TransitionRuleSync(t1,t2 , (Pr1 ;; Prs1) , Prs,NS) =
      TransitionRuleSync(t1,t2,Pr1,Prs,NS)
      TransitionRuleSync(t1,t2,Prs1,Prs,NS) .

op TransitionRuleSync : TransitionSync TransitionSync NetSystem -> RuleSet .

eq TransitionRuleSync(t1,t2,NS) =
TransitionRuleSync(t1,t2,Parts(VarsC(t1,t2,locality,NS)),
                  Parts(VarsC(t1,t2,identifier,NS)),NS) .

op allComp : TransitionSync NetSystem NetSystem -> RuleSet .

ceq allComp(ts, ( NS1 - NS2 ) , NS) = allComp(ts,NS1,NS) allComp(ts,NS2,NS)
      if NS1 =/= emptyNet and NS2 =/= emptyNet .
ceq allComp(ts, (Net1 , Net2) , NS) = allComp(ts,Net1,NS) allComp(ts,Net2,NS)
      if Net1 =/= emptyNet and Net2 =/= emptyNet .
eq allComp(ts, ( tNew - x -> p ) , NS) = none .
eq allComp(ts, ( tAut - x -> p ) , NS) = none .
eq allComp(ts, ( tGo - x -> p ) , NS) = none .
eq allComp(ts, ( t1 - x -> p ),NS) =
       if comp(ts,t1,NS) then TransitionRuleSync(ts,t1,NS) else none fi .
eq allComp(ts, ( p - x -> tNew ) , NS) = none .
eq allComp(ts, ( p - x -> tAut ) , NS) = none .
eq allComp(ts, ( p - x -> tGo ) , NS) = none .
eq allComp(ts, ( p - x -> t1 ),NS) =
  if comp(ts,t1,NS) then TransitionRuleSync(ts,t1,NS) else none fi .

vars TrS TrS1 TrS2 : TransitionSet .
```

```
op rules : NetSystem -> RuleSet .
eq rules(NS) = rules(T(NS) , NS) .

op rules : TransitionSet NetSystem -> RuleSet .

eq rules(TrS,emptyNet) = none .
eq rules(tAut , NS) = TransitionRule(tAut,NS) .
eq rules(tNew , NS) = TransitionRule(tNew,NS) .
eq rules(tGo , NS) = TransitionRule(tGo,NS) .
eq rules([tid , s ?] , NS) = allComp([tid , s ?] , NS , NS) .
eq rules([tid , s !] , NS) = none .
ceq rules(TSet(TrS1,TrS2),NS) = rules(TrS1,NS) rules(TrS2,NS)
    if TrS1 =/= emptyT and TrS2 =/= emptyT .

op idDef : NetSystem -> OpDeclSet .

eq idDef((p1id,c)) = (op getName(upTerm(p1id)) : nil -> 'PId[none] .) .
eq idDef(((pid,c) - x -> [tid,lm])) = idDef((pid,c)) .
eq idDef(([tid,lm] - x -> (pid,c))) = idDef((pid,c)) .
ceq idDef ((Net1 , Net2) ) = idDef(Net1) idDef(Net2)
     if Net1 =/= emptyNet and Net2 =/= emptyNet .
ceq idDef (NS1 - NS2) = idDef(NS1) idDef(NS2)
     if NS1 =/= emptyNet and NS2 =/= emptyNet .


op moduleOf : NetSystem -> SModule .

eq moduleOf (NS) =
(mod 'MSPN is
including 'NETS .
including 'META-LEVEL .
sorts none . *** sorts
none *** subsorts
idDef(NS)
none *** membership axioms
none *** equations
rules(addAtPlaces(NS)) *** rules
endm) .

*** Version of moduleOf in which we use strings as identifiers
*** of places and transitions

op moduleOfStr : NetSystem -> SModule .

eq moduleOfStr (NS) =

(mod 'MSPN is

including 'NETS .
including 'META-LEVEL .
sorts none .
none *** subsorts
none ***not needed, now they are strings
none *** membership axioms
```

```
none *** equations
rules(addAtPlaces(NS)) *** rules
endm) .

endm


*********************************************************************


mod COVERABILITY is

protecting NETS2APN .
protecting EXT-BOOL .
protecting REVERSE-RULES .
protecting META-LEVEL .

sort TermSet .
subsort Term < TermSet .
op empty : -> TermSet .
op set : TermSet TermSet -> TermSet [assoc comm idem id: empty] .

op _=<_ : MarkingSet MarkingSet -> Bool .
***que representan ideales

eq emptyMarkingSet =< MS = true .
ceq MS =< emptyMarkingSet = false if MS =/= emptyMarkingSet .

eq MMM1 =< MMM2 = MMM2 [= MMM1 .
ceq MMM1 =< MMM2 * MS = (MMM2 [= MMM1) or (MMM1 =< MS)
       if MS =/= emptyMarkingSet .
ceq ((MMM1 * MS1) =< MS2) = MMM1 =< MS2 and MS1 =< MS2
       if MS1 =/= emptyMarkingSet /\ MS2 =/= emptyMarkingSet .


var netModule : SModule .
vars MM1 MM2 : TermSet .
var R : SModule .
vars MM TM TM1 TM2 TM3 : Term .
var I : Qid .
var n : Nat .
var TS TS2 : TermSet .
var SO : Sort .
var SU : Substitution .
var RL : Rule .
var RLS : RuleSet .


op oneRule : TermSet Qid Nat TermSet SModule -> TermSet .

ceq oneRule ( TM , I , n , TS , netModule ) =
    oneRule ( TM , I , n + 1 , set ( TS , TM2 ) , netModule )
    if { TM2 , SO , SU } := metaApply( netModule , TM , I , none, n ) .
ceq oneRule ( TM , I , n , TS , netModule ) = TS
```

```
    if metaApply( netModule , TM , I , none , n ) == failure .
eq oneRule ( set ( TM , TS ) , I , n , TS2 , netModule ) =
           set ( oneRule ( TM , I , n , TS2 , netModule ) ,
           oneRule ( TS , I , n , TS2 , netModule ) ) .


******COVERABILITY

vars PP1 PP2 : MSPlacesB .
vars PPI PPI1 PPI2 : MSPlacesC .
vars MMM MMM1 MMM2 MMM3 MMM4 : Marking .
vars TL TL1 TL2 : TermList .
vars V V1 V2 : Variable .
vars C C1 C2 : Constant .

op needed : TermList -> TermList .

eq needed ( (MM , TL) ) = needed ( MM ) , needed ( TL ) .
eq needed ('_+_[TL , V]) = needed('_+_[TL]) .
eq needed ('_+_[V , TL]) = needed('_+_[TL]) .
eq needed ('_+_[MM,TL]) = '_+_[needed((MM,TL))] [owise] .
eq needed ('_+_[MM]) = needed(MM) .
eq needed(V) = if getType(V) == 'MSPlacesB then
    'emptySetB.MSPlacesB else 'emptySetC.MSPlacesC fi .

eq needed('`(_`,_`)[C1,C2]) = '`(_`,_`)[C1,C2] .
eq needed (C) = C .
eq needed ('_`{_`}[C,TM]) = '_`{_`}[C,TM] .
eq needed ('@`{_`}[TM]) = '@`{_`}[TM] .
eq needed ('__[V,TL]) = needed('__[TL]) .
eq needed ('__[TL,V]) = needed('__[TL]) .
eq needed ('__[MM,TL]) = '__[needed((MM,TL))] [owise] .
eq needed ('__[MM]) = needed(MM) .
eq needed ('_`,`,_[V,TL]) = needed('_`,`,_[TL]) .
eq needed ('_`,`,_[TL,V]) = needed('_`,`,_[TL]) .
eq needed ('_`,_`,[MM,TL]) = '_`,`,_[needed((MM,TL))] [owise] .
eq needed ('_`,`,_[MM]) = needed(MM) .
eq needed ('_++_[ TM1 , TM2 ] ) = '_++_[ needed(TM1) , needed(TM2) ] .


op upMarkingSet : MarkingSet -> TermSet .

eq upMarkingSet ( MMM1 ) = upTerm( MMM1 ) .
ceq upMarkingSet ( MMM1 * MS ) = set(upTerm(MMM1),upMarkingSet(MS))
   if MS =/= emptyMarkingSet .

op downMarkingSet : TermSet -> MarkingSet .

eq downMarkingSet ( empty ) = emptyMarkingSet .
eq downMarkingSet ( MM ) = downTerm( MM , emptySetB ++ emptySetC) .
ceq downMarkingSet ( set ( MM , TS ) ) =
  downTerm ( MM , emptySetB ++ emptySetC) * downMarkingSet ( TS )
  if TS =/= empty .
```

```
op Pre : MarkingSet SModule RuleSet -> MarkingSet .

ceq Pre ( MMM * MS , netModule , RLS ) =
    Pre ( MMM , netModule , RLS ) * Pre ( MS , netModule,RLS )
    if MS =/= emptyMarkingSet  .
ceq Pre ( MMM , netModule , RL RLS ) =
    Pre ( MMM , netModule , RL ) *
    Pre ( MMM , netModule , RLS )  if RLS =/= none .
eq Pre ( MMM ,  netModule , rl TM1 => TM2 [label( I )] . ) =
PreV (MMM U downTerm(needed(TM1),emptySetB ++ emptySetC),I,netModule) .

vars MS MS1 MS2 MS4 MS3 : MarkingSet .

op PreV : MarkingSet Qid SModule -> MarkingSet .

eq PreV ( MS , I , netModule ) =
downMarkingSet ( oneRule (upMarkingSet( MS ) , I , 0 , empty , netModule )) .


op addMarkings : MarkingSet MarkingSet -> MarkingSet .

eq addMarkings(MMM1,MS) = if MMM1 =< MS then MS else MMM1 * MS fi .
ceq addMarkings(MS1 * MS2, MS) = addMarkings(MS1,MS) * addMarkings(MS2,MS)
     if MS1 =/= emptyMarkingSet /\ MS2 =/= emptyMarkingSet .


***(

  Invariante de coveri

  MS2 =< MS1
  Pre(MS1 - MS2) =< MS1
  not(MMM1 =< MS1)

  Termination:  MMM1 =< Pre(MS2) => true
                Pre(MS2) =< MS1 =< false

)


op coveri : Marking MarkingSet MarkingSet SModule RuleSet -> Bool .
ceq coveri(MMM1,MS1,MS2,netModule,RLS) = MMM1 =< MS or-else
   if MS =< MS1 then false
   else coveri(MMM1,addMarkings(MS,MS1),MS,netModule,RLS) fi
   if MS := Pre(MS2,netModule,RLS) .

op cover : Marking Marking SModule -> Bool .

ceq cover ( MMM1 , MMM2 , netModule ) = MMM2 [= MMM1 or-else
  coveri ( MMM1 , MMM2 , MMM2 , R , getRls(R) ) if R := reverse(netModule) .

endm
```

```
******************************************************************************

in example.maude

mod XML2NETS is

including TRANSLATION .
including NETS2APN .
including COVERABILITY .

vars E E1 E2 : Element .
vars S S1 S2 : String .
vars A A1 A2 : Attribute .
vars AL AL1 AL2 : AttributeList .
vars C C1 C2 : Content .
var PS : PlaceSet .
var TS : TransitionSet .
var s : Services .
var p : Place .
var t : Transition .
var x : Var .
var c : Colour .
var lm : Lambda .
var n : Nat .


op translate : Element -> NetSystem .

op translatedNet : -> NetSystem .
eq translatedNet = translate(translation) .

eq translate(["workspaceElements",nil,(E1 ["cpnet",nil,(E2 C)])])=
   translateCore(C,placesOf(C),transitionsOf(C)) .

op placesOf : Content -> PlaceSet .
op placesOf : String Content -> PlaceSet .
op transitionsOf : Content -> TransitionSet .
op transitionsOf : String Content -> TransitionSet .
op translateCore : Content PlaceSet TransitionSet -> NetSystem .
op translateCore2 : Content PlaceSet TransitionSet -> Net .

ceq placesOf(C1 C2) = PSet(placesOf(C1),placesOf(C2))
  if C1 =/= nil and C2 =/= nil .
eq placesOf(["page",AL,C]) = placesOf(C) .
eq placesOf(["place",("id" = S) , C]) = placesOf(S,C) .
eq placesOf(C) = emptyP [owise] .
eq placesOf(S,C1 ["text",nil,"black"] C2) = (S,black) .
eq placesOf(S,C1 ["text",nil,"locality"] C2) = (S,locality) .
eq placesOf(S,C1 ["text",nil,"identifier"] C2) = (S,identifier) .
eq placesOf(S,C) = (S,black) [owise] .
```

```
op pred : Nat -> Nat .
eq pred(s(N:Nat)) = N:Nat .


ceq transitionsOf(C1 C2) = TSet(transitionsOf(C1),transitionsOf(C2))
   if C1 =/= nil and C2 =/= nil .
eq transitionsOf(["page",AL,C]) = transitionsOf(C) .
eq transitionsOf(["trans",("id" = S) , C]) = transitionsOf(S,C) .
eq transitionsOf(C) = emptyT [owise] .
eq transitionsOf(S,C1 ["text",nil,"tau"] C2) = [S,tau] .
eq transitionsOf(S,C1 ["text",nil,"go"] C2) = [S,go] .
eq transitionsOf(S,C1 ["text",nil,"new"] C2) = [S,new] .
ceq transitionsOf(S1,(C1 ["text",nil,S2] C2)) =
                                [S1, substr(S2,0,pred(length(S2))) ?]
    if substr(S2,pred(length(S2)),1) == "?" .
ceq transitionsOf(S1,(C1 ["text",nil,S2] C2)) =
                                [S1, substr(S2,0,pred(length(S2))) !]
     if substr(S2,pred(length(S2)),1) == "!" .
eq transitionsOf(S,C) = [S,tau] [owise] .

ceq translateCore(C1 C2,PS,TS) = translateCore(C1,PS,TS) -
        translateCore(C2,PS,TS) if C1 =/= nil and C2 =/= nil .
eq translateCore(["page",AL,C],PS,TS) = translateCore2(C,PS,TS) .

ceq translateCore2(C1 C2,PS,TS) = (translateCore2(C1,PS,TS) ,
    translateCore2(C2,PS,TS)) if C1 =/= nil and C2 =/= nil .
eq translateCore2(["arc",(A "orientation" = "PtoT"),C],PS,TS) =
   translatePtoT(C,PS,TS) .
eq translateCore2(["arc",(A "orientation" = "TtoP"),C],PS,TS) =
   translateTtoP(C,PS,TS) .
eq translateCore2(C,PS,TS) = emptyNet [owise] .

ops translatePtoT translateTtoP : Content PlaceSet TransitionSet -> Flow .

eq translatePtoT ( C , PS , TS ) =
    selectP(C,PS) - variableIn(C) ->  selectT(C,TS) .
eq translateTtoP(C,PS,TS) =
    selectT(C,TS) - variableIn(C) -> selectP(C,PS) .

op selectT : Content TransitionSet -> Transition .
op selectP :Content PlaceSet -> Place .
op variableIn : Content -> Var .

eq selectP(C1 ["placeend","idref" = S,nil] C2,PSet((S,c),PS)) = (S,c) .
eq selectT(C1 ["transend","idref" = S,nil] C2,TSet([S,lm],TS)) = [S,lm] .

ceq variableIn(C ["annot",AL,(C1 ["text",nil,S])] C2) = S if S =/= "eps" .
eq variableIn(C) = eps [owise] .

sort Regla CRegla .
subsort Regla < CRegla .
op noRegla : -> Regla .
op [_=>_] : Marking Marking -> Regla [format (d d d n+ d d)] .
op __ : CRegla CRegla -> CRegla [assoc comm id: noRegla format(d n n)] .
```

```
op showRules : SModule -> CRegla .
var sm : SModule .
eq showRules( sm ) = showRules(getRls(sm)) .
op showRules : RuleSet -> CRegla .
vars T1 T2 : Term .
vars RS1 RS2 : RuleSet .
var I J : Qid .
ceq showRules( RS1 RS2 ) = showRules(RS1) showRules(RS2)
   if RS1 =/= none /\ RS2 =/= none .
eq showRules(none) = noRegla .
eq showRules(rl T1 => T2 [label(I)] .) =
  [downTerm(T1,emptySetB ++ emptySetC) => downTerm(T2,emptySetB ++ emptySetC)] .

op fire : Marking SModule -> Marking .
var M : Marking .

ceq fire(M,sm) = downTerm(T1,emptySetB ++ emptySetC)
  if {T1,I} := metaRewrite(sm,upTerm(M),unbounded) .

endm

eof

Example in Fig. 10.6

Correspondence between names of places in Fig. 10.6 and names of
places in the following statements


("ID2053",black) -> q
("ID4151",black) -> r
("ID2012",locality) -> l
("ID2031",black) -> p
("ID2078",identifier) -> a
("ID4215",identifier) -> b
@{'"ID2012"} -> location of net in the left of Fig. 10.6
@{'"ID4215"} -> location of net in the right of Fig. 10.6


Maude> red showRules(moduleOfStr(translatedNet)) .

reduce in XML2NETS : showRules(moduleOfStr(translatedNet)) .
rewrites: 16581 in 101ms cpu (214ms real) (162582 rewrites/second)
result CRegla:

[P:MSPlacesB ++ VF4:MSPlacesC + II:MS2PlacesC + VF2:MSPlacesC
@{'"ID2012"} ("ID2012",locality) => P:MSPlacesB,,("ID2031",black)
++ VF4:MSPlacesC + II:MS2PlacesC + VF2:MSPlacesC @{'"ID2012"}]

[P:MSPlacesB ++ VF4:MSPlacesC + II:MS2PlacesC + VF2:MSPlacesC
@{'"ID2012"} + VF3:MSPlacesC ("ID2012",locality) =>
P:MSPlacesB,,("ID2031",black) ++ VF2:MSPlacesC + VF4:MSPlacesC +
II:MS2PlacesC + VF3:MSPlacesC @{'"ID2012"}]
```

```
[P:MSPlacesB,,("ID2053",black) ++ VF2:MSPlacesC + II:MS2PlacesC =>
P:MSPlacesB ++ VF2:MSPlacesC + II:MS2PlacesC +
("ID2078",identifier)]

[P:MSPlacesB,,("ID2031",black),,("ID4151",black) ++ II:MS2PlacesC
+ VF3:MSPlacesC ("ID2078",identifier) + VF2:MSPlacesC @{'"ID2012"}
@{'"ID4151"} => P:MSPlacesB ++ II:MS2PlacesC + VF3:MSPlacesC
("ID4215",identifier) + VF2:MSPlacesC @{'"ID2012"} @{'"ID4151"}]


The first rule corresponds to the movement transition in the case
that the origin and the destination of the movement are the same
location.1 The second rule correponds to the movement transition
in the case that the origin and the destination are different. The
third rule corresponds to the new transition. The fourth rule
corresponds to the synchronization when both nets are co-located.



Initial marking of example in Fig. 10.6:

("ID2053",black) ,, ("ID4151",black) ++ ("ID2012",locality)
@{'"ID4151"} + @{'"ID2012"}


Execution from the initial marking:

Maude> red fire( ("ID2053",black) ,, ("ID4151",black) ++
("ID2012",locality) @{'"ID4151"} + @{'"ID2012"} , moduleOfStr(translatedNet) ) .

reduce in XML2NETS : fire(("ID2053",black),,("ID4151",black) ++
@{'"ID2012"} + @{'"ID4151"} ("ID2012",locality),
moduleOfStr(translatedNet)) .

rewrites: 16545 in 141ms cpu (354ms real) (116532 rewrites/second)
result Marking: emptySetB ++ @{'"ID2012"} @{'"ID4151"} + ("ID4215",identifier)

The result is the marking in which both nets are co-located and
there is only one token, an identifier token in b

Marking in which there is an identifier in b and the rest of the
places are empty:

emptySetB ++ ("ID4215",identifier)

Can that marking be covered from the initial marking?

Maude> red cover( ( ("ID2053",black) ,, ("ID4151",black) ++
("ID2012",locality) @{'"ID4151"} + @{'"ID2012"} ) , (emptySetB ++
("ID4215",identifier)) , moduleOfStr(translatedNet) ) .

reduce in XML2NETS : cover(("ID2053",black),,("ID4151",black) ++
@{'"ID2012"} + @{'"ID4151"} ("ID2012",locality), emptySetB ++
("ID4215",identifier), moduleOfStr(translatedNet)) .
```

```
rewrites: 14019341 in 117526ms cpu (125882ms real) (119287
rewrites/second) result Bool: true

Marking in which there are to identical identifiers in a:

emptySetB ++ ("ID2078",identifier) ("ID2078",identifier)

Can that marking be covered from the initial marking?

Maude> red cover( ( ("ID2053",black) ,, ("ID4151",black) ++
("ID2012",locality) @{'"ID4151"} + @{'"ID2012"} ) , (emptySetB ++
("ID2078",identifier) ("ID2078",identifier) ) ,
moduleOfStr(translatedNet) ) .

reduce in XML2NETS : cover(("ID2053",black),,("ID4151",black) ++
@{'"ID2012"} + @{'"ID4151"} ("ID2012",locality), emptySetB ++
("ID2078",identifier) ("ID2078",identifier), moduleOfStr(translatedNet)) .

rewrites: 19714 in 173ms cpu (332ms real) (113315 rewrites/second)
result Bool: false
```

Bibliografía

[AB96] A. Asperti y N. Busi. Mobile Petri nets. Informe técnico UBLCS-96-10, University of Bologna, 1996.

[Aba98] M. Abadi. Two facets of authentication. En *Proceedings of the 11th IEEE Computer Security Foundations Workshop*, páginas 27–32. IEEE Computer Society, 1998. ISBN 0-8186-8488-7.

[ACABJ04] P. A. Abdulla, A. Collomb-Annichini, A. Bouajjani y B. Jonsson. Using forward reachability analysis for verification of lossy channel systems. *Formal Methods in System Design*, 25(1):39–65, 2004.

[ACJT00] P. A. Abdulla, K. Cerans, B. Jonsson y Y.-K. Tsay. Algorithmic analysis of programs with well quasi-ordered domains. *Information and Computation*, 160(1-2):109–127, 2000. ISSN 0890-5401.

[ADMN04] P. A. Abdulla, J. Deneux, P. Mahata y A.Ñylén. Forward reachability analysis of timed Petri nets. En Y. Lakhnech y S. Yovine, editores, *FORMATS/FTRTFT*, volumen 3253 de *Lecture Notes in Computer Science*, páginas 343–362. Springer-Verlag, 2004. ISBN 3-540-23167-6.

[AG97] M. Abadi y A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. En *Fourth ACM Conference on Computer and Communications Security*, páginas 36–47. ACM Press, 1997.

[AN00] P. A. Abdulla y A.Ñylén. Better is better than well: On efficient verification of infinite-state systems. En *LICS '00: Proceedings of the 15th Annual IEEE Symposium on Logic in Computer Science*, páginas 132–140. IEEE Computer Society, 2000.

[And01] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001. ISBN 0471389226.

[BB02] M. Boreale y M. G. Buscemi. A framework for the analysis of security protocols. En *CONCUR '02: Proceedings of the 13th International Conference on Concurrency Theory*, volumen 2421 de *Lecture Notes in Computers Science*, páginas 483–498. Springer-Verlag, London, UK, 2002. ISBN 3-540-44043-7.

[BBPP04] M. A. Bednarczyk, L. Bernardinello, W. Pawlowski y L. Pomello. Modelling mobility with Petri hypernets. En J. L. Fiadeiro, P. D. Mosses y

F. Orejas, editores, *WADT*, volumen 3423 de *Lecture Notes in Computer Science*, páginas 28–44. Springer-Verlag, 2004. ISBN 3-540-25327-0.

[BCEH05] P. Baldan, A. Corradini, H. Ehrig y R. Heckel. Compositional semantics for open Petri nets based on deterministic processes. *Mathematical. Structures in Comp. Sci.*, 15(1):1–35, 2005. ISSN 0960-1295.

[BCR01] T. Ball, S. Chaki y S. K. Rajamani. Parameterized verification of multithreaded software libraries. En T. Margaria y W. Yi, editores, *Tools and algorithms for the construction and analysis of systems*, volumen 2031 de *Lecture Notes in Computer Science*, páginas 158–173. Springer-Verlag, 2001. ISBN 3-540-41865-2.

[BDK01] E. Best, R. Devillers y M. Koutny. *Petri net algebra*. Springer-Verlag New York, Inc., New York, NY, USA, 2001. ISBN 3-540-67398-9.

[BET03] A. Bouajjani, J. Esparza y T. Touili. A generic approach to the static analysis of concurrent programs with procedures. En *Principles of Programming Languages*, páginas 62–73. 2003.

[BG95] N. Busi y R. Gorrieri. A Petri net semantics for pi-calculus. En *6th International Conference on Concurrency Theory, CONCUR'95*, volumen 962 de *Lecture Notes in Computer Science*, páginas 145–159. Springer-Verlag, 1995.

[BJM00] A. Bouhoula, J.-P. Jouannaud y J. Meseguer. Specification and proof in membership equational logic. *Theor. Comput. Sci.*, 236(1-2):35–132, 2000.

[Bou92] G. Boudol. Asynchrony and the pi-calculus. Informe técnico INRIA RR-1702, Sophia Antipolis, 1992.

[BZ05] N. Busi y G. Zavattaro. Deciding reachability in mobile ambients. En *14th European Symposium on Programming Languages and Systems*, volumen 3444 de *Lecture Notes in Computer Science*, páginas 248–262. Springer-Verlag, 2005.

[Car99] L. Cardelli. Abstractions for mobile computation. En *Secure Internet Programming, Security Issues for Mobile and Distributed Objects*, volumen 1603 de *Lecture Notes in Computer Science*, páginas 51–94. Springer-Verlag, 1999.

[CDE+02] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer y J. F. Quesada. Maude: specification and programming in rewriting logic. *Theor. Comput. Sci.*, 285(2):187–243, 2002.

[CDF+03] K. Cheverst, A. J. Dix, D. Fitton, A. Friday y M. Rouncefield. Exploring the utility of remote messaging and situated office door displays. En L. Chittaro, editor, *Mobile HCI*, volumen 2795 de *Lecture Notes in Computer Science*, páginas 336–341. Springer-Verlag, 2003. ISBN 3-540-40821-5.

[CG98] L. Cardelli y A. D. Gordon. Mobile ambients. En *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98*, volumen 1378 de *Lecture Notes in Computer Science*, páginas 140–155. Springer-Verlag, 1998.

[CGG02] L. Cardelli, G. Ghelli y A. D. Gordon. Types for the ambient calculus. *Inf. Comput.*, 177(2):160–194, 2002. ISSN 0890-5401.

[CH94] S. Christensen y N. D. Hansen. Coloured Petri nets extended with channels for synchronous communication. En R. Valette, editor, *Application and Theory of Petri Nets*, volumen 815 de *Lecture Notes in Computer Science*, páginas 159–178. Springer-Verlag, 1994. ISBN 3-540-58152-9.

[CNS04] M. Carbone, M.Ñielsen y V. Sassone. A calculus for trust management. En K. Lodaya y M. Mahajan, editores, *FSTTCS*, volumen 3328 de *Lecture Notes in Computer Science*, páginas 161–173. Springer-Verlag, 2004. ISBN 3-540-24058-6.

[Del] G. Delzanno. Constraint-based automatic verification of abstract models of multithreaded programs. *Journal of Theory and Practice of Logic Programming (to appear)*.

[Del02] G. Delzanno. An overview of MSR(C): A CLP-based framework for the symbolic verification of parameterized concurrent systems. En *11th International Workshop of Functional and (Constraint) Logic Programming, WFLP'02*, volumen 76 de *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[DKK04] R. R. Devillers, H. Klaudel y M. Koutny. Petri net semantics of the finite pi-calculus. En *24th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems*, volumen 3235 de *Lecture Notes in Computer Science*, páginas 309–325. 2004.

[DLMS99] N. Durgin, P. Lincoln, J. Mitchell y A. Scedrov. Undecidability of bounded security protocols. En N. Heintze y E. Clarke, editores, *Proceedings of the Workshop on Formal Methods and Security Protocols — FMSP, Trento, Italy*. julio 1999.

[DR98] J. Desel y W. Reisig. Place/transition Petri nets. *Lecture Notes in Computer Science: Lectures on Petri Nets I: Basic Models*, 1491:122–173, 1998.

[EFM99] J. Esparza, A. Finkel y R. Mayr. On the verification of broadcast protocols. En *LICS '99: Proceedings of the 14th Annual IEEE Symposium on Logic in Computer Science*, página 352. IEEE Computer Society, Washington, DC, USA, 1999. ISBN 0-7695-0158-3.

[EM85] H. Ehrig y B. Mahr. *Fundamentals of Algebraic Specification I*. Springer-Verlag New York, Inc., 1985. ISBN 0387137181.

[FC03] D. Fitton y K. Cheverst. Experiences managing and maintaining a collection of interactive office door displays. En E. H. L. Aarts, R. Collier, E. van Loenen y B. E. R. de Ruyter, editores, *EUSAI*, volumen 2875 de *Lecture*

Notes in Computer Science, páginas 394–409. Springer-Verlag, 2003. ISBN 3-540-20418-0.

[FEA03] D. de Frutos-Escrig y O. M. Alonso. Ambient Petri nets. En *Foundations of Global Computing, FGC'03*, volumen 85 de *Electronic Notes in Theoretical Computer Science*. Elsevier, 2003.

[FEMA03] D. de Frutos-Escrig y O. Marroquín-Alonso. Replicated ambient Petri nets. En P. M. A. Sloot, D. Abramson, A. V. Bogdanov, J. Dongarra, A. Y. Zomaya y Y. E. Gorbachev, editores, *International Conference on Computational Science*, volumen 2658 de *Lecture Notes in Computer Science*, páginas 774–783. Springer-Verlag, 2003. ISBN 3-540-40195-4.

[FERA99] D. de Frutos-Escrig, V. V. Ruiz y O. M. Alonso. Decidability of properties of timed-arc Petri nets. En *20th International Conference on Application and Theory of Petri Nets, ICATPN'99*, volumen 1639 de *Lecture Notes in Computer Science*, páginas 187–206. Springer-Verlag, 1999.

[FERVMA05] D. de Frutos-Escrig, F. Rosa-Velardo y O. Marroquín-Alonso. Ubiquitous systems and petri nets. En O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar y C. J. K. Tan, editores, *ICCSA 2005*, volumen 3481 de *Lecture Notes in Computer Science*, páginas 1156–1166. Springer, 2005.

[FG96] C. Fournet y G. Gonthier. The reflexive cham and the join-calculus. En *POPL '96: Proceedings of the 23rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, páginas 372–385. ACM Press, New York, NY, USA, 1996. ISBN 0-89791-769-3.

[FG01] R. Foccardi y R. Gorrieri. Classification of security properties (Part I: Information flow). En *Foundations of Security Analysis and Design, FOSAD'00*, volumen 2171 de *Lecture Notes in Computer Science*, páginas 331–396. Springer-Verlag, 2001.

[FGL+96] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget y D. Rémy. A calculus of mobile agents. En U. Montanari y V. Sassone, editores, *CONCUR*, volumen 1119 de *Lecture Notes in Computer Science*, páginas 406–421. Springer-Verlag, 1996. ISBN 3-540-61604-7.

[FGM04] R. Foccardi, R. Gorrieri y F. Martinelli. Classification of security properties (Part II: Network security). En *Foundations of Security Analysis and Design II, FOSAD'02*, volumen 2946 de *Lecture Notes in Computer Science*, páginas 139–185. Springer-Verlag, 2004.

[FS01] A. Finkel y P. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1–2):63–92, 2001.

[GHKR04] S. Gilmore, J. Hillston, L. Kloul y M. Ribaudo. Software performance modelling using PEPA nets. En J. J. Dujmovic, V. A. F. Almeida y D. Lea, editores, *WOSP*, páginas 13–23. ACM, 2004.

[Gor00] A. D. Gordon. Notes on nominal calculi for security and mobility. En R. Focardi y R. Gorrieri, editores, *FOSAD*, volumen 2171 de *Lecture Notes in Computer Science*, páginas 262–330. Springer-Verlag, 2000. ISBN 3-540-42896-8.

[Hen90] M. Hennessy. *The Semantics of Programming Languages: An Elementary Introduction Using Structural Operational Semantics*. John Wiley and Sons, 1990.

[HMR03] M. Hennessy, M. Merro y J. Rathke. Towards a behavioural theory of access and mobility control in distributed systems. En *Foundations of Software Science and Computational Structures, 6th International Conference, FOSSACS'03*, volumen 2620 de *Lecture Notes in Computer Science*, páginas 282–298. Springer-Verlag, 2003.

[Hoa85] C. A. R. Hoare. *Communicating sequential processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985. ISBN 0-13-153271-5.

[HR02a] M. Hennessy y J. Riely. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 24(5):566–591, 2002. ISSN 0164-0925.

[HR02b] M. Hennessy y J. Riely. Resource access control in systems of mobile agents. *Inf. Comput.*, 173(1):82–120, 2002.

[HR04] J. Hillston y M. Ribaudo. Modelling mobility with PEPA nets. En C. Aykanat, T. Dayar y I. Korpeoglu, editores, *ISCIS*, volumen 3280 de *Lecture Notes in Computer Science*, páginas 513–522. Springer-Verlag, 2004. ISBN 3-540-23526-4.

[HU79] J. E. Hopcroft y J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts, 1979.

[Jen96] K. Jensen. *Coloured Petri nets (2nd ed.): basic concepts, analysis methods and practical use: volume 1*. Springer-Verlag, London, UK, 1996. ISBN 3-540-60943-1.

[Joh88] C. Johnen. Decidability of home space property. *Petri Net Newsletter*, (29):20–21, abril 1988.

[KEB94] M. Koutny, J. Esparza y E. Best. Operational semantics for the Petri box calculus. En B. Jonsson y J. Parrow, editores, *CONCUR*, volumen 836 de *Lecture Notes in Computer Science*, páginas 210–225. Springer-Verlag, 1994. ISBN 3-540-58329-7.

[Kin97] E. Kindler. A compositional partial order semantics for Petri net components. En *ICATPN '97: Proceedings of the 18th International Conference on Application and Theory of Petri Nets*, páginas 235–252. Springer-Verlag, London, UK, 1997. ISBN 3-540-63139-9.

[Kum00] O. Kummer. Undecidability in object-oriented Petri nets. *Petri Net Newsletter*, 59:18–23, 2000.

[Lak05] C. Lakos. A Petri net view of mobility. En F. Wang, editor, *25th IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems*, volumen 3731 de *Lecture Notes in Computer Science*, páginas 174–188. Springer-Verlag, 2005. ISBN 3-540-29189-X.

[Laz05] R. Lazic. Decidability of reachability for polymorphic systems with arrays: A complete classification. En *6th International Workshop on Verification of Infinite-State Systems, INFINITY'04*, volumen 138 de *Electronic Notes in Theoretical Computer Science*, páginas 3–19. Elsevier, 2005.

[LNO+07] R. Lazic, T. C. Newcomb, J. Ouaknine, A. W. Roscoe y J. Worrell. Nets with tokens which carry data. En *28th International Conference on Application and Theory of Petri Nets and other models of concurrency*, volumen 4546 de *Lecture Notes in Computer Science*, páginas 301–320. Springer-Verlag, 2007.

[LNR05] R. Lazic, T. C. Newcomb y B. Roscoe. Polymorphic systems with arrays, 2-counter machines and multiset rewriting. En *6th International Workshop on Verification of Infinite-State Systems, INFINITY'04*, volumen 138, páginas 61–86. Elsevier, 2005.

[Lom01a] I. A. Lomazova. Nested Petri nets: Multi-level and recursive systems. *Fundam. Inform.*, 47(3-4):283–293, 2001.

[Lom01b] I. A. Lomazova. Recursive nested Petri nets: Analysis of semantic properties and expressibility. *Programming and Computer Software*, 27(4):183–193, 2001.

[Lom02] I. A. Lomazova. Modeling dynamic objects in distributed systems with nested Petri nets. *Fundam. Inf.*, 51(1):121–133, 2002. ISSN 0169-2968.

[LS99] I. A. Lomazova y P. Schnoebelen. Some decidability results for nested Petri nets. En D. Bjørner, M. Broy y A. V. Zamulin, editores, *Ershov Memorial Conference*, volumen 1755 de *Lecture Notes in Computer Science*, páginas 208–220. Springer-Verlag, 1999. ISBN 3-540-67102-1.

[Mil82] R. Milner. *A Calculus of Communicating Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1982. ISBN 0387102353.

[Mil85] E. C. Milner. Basic wqo- and bqo-theory. *Graphs and order, NATO Adv. Sci. Inst. Ser. C Math. Phys. Sci.*, 147:487–502, 1985.

[Mil04] R. Milner. Theories for the global ubiquitous computer. En *7th International Conference on Foundations of Software Science and Computation Structures, FOSSACS'04*, volumen 2987 de *Lecture Notes in Computers Sciente*, páginas 5–11. Springer-Verlag, 2004.

[Min67] M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967. ISBN 0-13-165563-9.

[MM90] J. Meseguer y U. Montanari. Petri nets are monoids. *Information and Computation*, 88:105–155, 1990.

[MOM02] N. Martí-Oliet y J. Meseguer. Rewriting logic: roadmap and bibliography. *Theoretical Computer Science*, 285(2):121–154, 2002.

[MPW92] R. Milner, J. Parrow y D. Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992. ISSN 0890-5401.

[MVO96] A. J. Menezes, S. A. Vanstone y P. C. V. Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996. ISBN 0849385237.

[Nee93] R. M. Needham. Names. páginas 315–327, 1993.

[NHN03] F.Ñielson, R. R. Hansen y H. R. Nielson. Abstract interpretation of mobile ambients. *Sci. Comput. Program.*, 47(2-3):145–175, 2003. ISSN 0167-6423.

[Pet62] C. A. Petri. *Kommunikation mit Automaten*. PhD thesis, Institut fur Instrumentelle Mathematik, Bonn, 1962.

[PS93] B. C. Pierce y D. Sangiorgi. Typing and subtyping for mobile processes. En *Proceedings 8th IEEE Logics in Computer Science, LICS'93*, páginas 376–385. IEEE Computer Society Press, 1993.

[Ram00] G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.

[Reu90] C. Reutenauer. *The mathematics of Petri nets*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1990. ISBN 0-13-561887-8.

[Ric00] G. G. Richard. Service advertisement and discovery: Enabling universal device cooperation. *IEEE Internet Computing*, 4(5):18–26, 2000. ISSN 1089-7801.

[Rin01] M. C. Rinard. Analysis of multithreaded programs. En P. Cousot, editor, *SAS*, volumen 2126 de *Lecture Notes in Computer Science*, páginas 1–19. Springer-Verlag, 2001. ISBN 3-540-42314-1.

[RS03] R. Ramanujam y S. P. Suresh. Tagging makes secrecy decidable with unbounded nonces as well. En P. K. Pandya y J. Radhakrishnan, editores, *FSTTCS*, volumen 2914 de *Lecture Notes in Computer Science*, páginas 363–374. Springer-Verlag, 2003. ISBN 3-540-20680-9.

[RS05] R. Ramanujam y S. P. Suresh. Decidability of context-explicit security protocols. *Journal of Computer Security*, 13(1):135–165, 2005.

[RV04] F. Rosa-Velardo. *Typing Techniques for Security in Mobile Agent Systems*. Trabajo de tercer ciclo, Depto. de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 2004.

[RV07a] F. Rosa-Velardo. Coding mobile synchronizing petri nets into rewriting logic. En *7th International Workshop on Rule-Based Programming, RULE'06*, volumen 174 de *Electronic Notes in Theoretical Computer Science*, páginas 83–98. Elsevier, 2007.

[RV07b] F. Rosa-Velardo. Petri nets for the verification of ubiquitous systems with transient secure association. En J. Indulska, J. Ma, L. T. Yang, T. Ungerer y J. Cao, editores, *4th International Conference on Ubiquitous Intelligence and Computing, UIC'07*, volumen 4611 de *Lecture Notes in Computer Science*, páginas 1148–1158. Springer-Verlag, 2007.

[RVFE06] F. Rosa-Velardo y D. de Frutos-Escrig. Symbolic semantics for the verification of security properties of mobile Petri nets. En S. Graf y W. Zhang, editores, *4th International Symposium on Automated Technology for Verification and Analysis, ATVA'06*, volumen 4218 de *Lecture Notes in Computer Science*, páginas 461–476. 2006. ISBN 3-540-47237-1.

[RVFE07] F. Rosa-Velardo y D. de Frutos-Escrig. Name creation vs. replication in Petri net systems. En J. Kleijn y A. Yakovlev, editores, *28th International Conference on Application and Theory of Petri Nets and other models of concurrency, ATPN'07*, volumen 4546 de *Lecture Notes in Computer Science*, páginas 402–422. Springer-Verlag, 2007.

[RVFEA07] F. Rosa-Velardo, D. de Frutos-Escrig y O. M. Alonso. On the expressiveness of mobile synchronizing Petri nets. En *3rd International Workshop on Security Issues in Concurrency, SECCO'05*, volumen 180 de *Electronic Notes in Theoretical Computer Science*. 2007.

[RVFEMA06] F. Rosa-Velardo, D. de Frutos-Escrig y O. Marroquín-Alonso. Replicated ubiquitous nets. En M. L. Gavrilova, O. Gervasi, V. Kumar, C. J. K. Tan, D. Taniar, A. Laganà, Y. Mun y H. Choo, editores, *International Conference on Computational Science and Its Applications, ICCSA'06*, páginas 158–168. 2006.

[RVMAFE06] F. Rosa-Velardo, O. Marroquín-Alonso y D. de Frutos-Escrig. Mobile synchronizing Petri nets: a choreographic approach for coordination in ubiquitous systems. En *First International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems, MTCOORD'05*, volumen 99 de *Electronic Notes in Theoretical Computer Science*. Elsevier, 2006.

[RVSFE05] F. Rosa-Velardo, C. Segura y D. de Frutos-Escrig. Tagged systems: a framework for the specification of history-dependent properties. En S. Lucas, editor, *4th Spanish Conference on Programming and Computer Languages, PROLE'04*, volumen 137 de *Electronic Notes in Theoretical Computer Science*, páginas 151–174. Elsevier, 2005.

[RVSV06] F. Rosa-Velardo, C. Segura y A. Verdejo. Typed mobile ambients in Maude. En *6th International Workshop on Rule-Based Programming, RULE'05*, volumen 147 de *Electronic Notes in Theoretical Computer Science*, páginas 135–161. Elsevier, 2006.

[RWL+03] A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen y K. Jensen. CPN tools for editing, simulating, and analysing coloured Petri nets. En *Proceedings of the 24th International Conference on Applications and Theory of Petri Nets*

(ICATPN 2003), volumen 2679 de *Lecture Notes in Computer Science*, páginas 450–462. Springer-Verlag, 2003.

[Sch05] B. Schneier. Crypto-gram, June 2005. `http://www.schneier.com/crypto-gram.html`, 2005.

[SMÖ01a] M.-O. Stehr, J. Meseguer y P. C. Ölveczky. Rewriting logic as a unifying framework for Petri nets. En H. Ehrig, G. Juhás, J. Padberg y G. Rozenberg, editores, *Unifying Petri Nets*, volumen 2128 de *Lecture Notes in Computer Science*, páginas 250–303. Springer-Verlag, 2001. ISBN 3-540-43067-9.

[Sta02] F. Stajano. *Security for Ubiquitous Computing*. John Wiley and Sons, febrero 2002. ISBN 0-470-84493-0.

[Ste99] B. Sterling. *A Good Old-fashioned Future*. Spectra, 1999.

[SVW01] N. Sidorova, M. Voorhoeve y J. van der Woude. A calculus of Petri net components. En *Proc. Workshop on Modelling of Objects, Components, and Agents, MOCA'01*, páginas 121–132. 2001.

[Tau89] D. Taubner. Representing CCS Programs by Finite Predicate/Transition Nets. *Acta Informaticae*, 27(6):533–565, 1989.

[TSMO02] P. Thati, K. Sen y N. Martí-Oliet. An executable specification of asynchronous pi-calculus semantics and may testing in Maude 2.0. 71, 2002.

[Val98] R. Valk. Petri nets as token objects: An introduction to elementary object nets. En J. Desel y M. Silva, editores, *ICATPN*, volumen 1420 de *Lecture Notes in Computer Science*, páginas 1–25. Springer-Verlag, 1998. ISBN 3-540-64677-9.

[Val01] R. Valk. Concurrency in communicating object Petri nets. En G. Agha, F. de Cindio y G. Rozenberg, editores, *Concurrent Object-Oriented Programming and Petri Nets*, volumen 2001 de *Lecture Notes in Computer Science*, páginas 164–195. Springer-Verlag, 2001. ISBN 3-540-41942-X.

[Val03] R. Valk. Object Petri nets: Using the nets-within-nets paradigm. En J. Desel, W. Reisig y G. Rozenberg, editores, *Lectures on Concurrency and Petri Nets*, volumen 3098 de *Lecture Notes in Computer Science*, páginas 819–848. Springer-Verlag, 2003. ISBN 3-540-22261-8.

[VMO02] A. Verdejo y N. Martí-Oliet. Implementing CCS in Maude 2. En *Workshop on Rewriting Logic and its Applications, WRLA'02*, volumen 71 de *Electronic Notes in Theoretical Computer Science*. Elsevier, 2002.

[VRGFE99] V. Valero-Ruiz, F. C. Gomez y D. de Frutos Escrig. On non-decidability of reachability for timed-arc Petri nets. En *PNPM '99: Proceedings of the The 8th International Workshop on Petri Nets and Performance Models*, página 188. IEEE Computer Society, Washington, DC, USA, 1999. ISBN 0-7695-0331-4.

[Wan04] R. Want. Enabling ubiquitous sensing with RFID. *IEEE Computer*, 37(4):84–86, 2004.

[Wei91] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.

[Wei93] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):74–84, 1993.

[Zim00] P. Zimmer. On the expressiveness of pure mobile ambients. En *7th International Workshop on Expressiveness in Concurrency, EXPRESS'00*, volumen 39 de *Electronic Notes in Theoretical Computer Science*. Elsevier, 2000.

[Zim03] P. Zimmer. On the expressiveness of pure safe ambients. *Mathematical Structures in Computer Science*, 13(5):721–770, 2003.