THOMSON

MANUEL SILVA

Las Redes de Petri:

en la Automática y la Informática

Editorial AC

Las Redes de Petri:
en la Automática y la Informática

Las Redes de Petri:
en la Automática y la Informática

MANUEL SILVA

THOMSON

Australia • Canadá • México • Singapur • España • Reino Unido • Estados Unidos

LAS REDES DE PETRI EN LA AUTOMÁTICA Y LA INFORMÁTICA
por MANUEL SILVA

una compañía de International Thomson Editores Spain Paraninfo, S.A.
Magallanes, 25. 28015 Madrid. España
Teléf.: 91 4463350. Fax: 91 4456218
clientes@paraninfo.es
www.paraninfo.es

ISBN: 84-7288-045-1
Depósito legal: SE-3759-2002 (30)

Fotocomposición: Alfa Centauro, S.A., Víctor de la Serna, 46. 28016 Madrid
Impresión: Publicaciones Digitales, S.A.
www.publidisa.com - (+34) 95.458.34.25. (Sevilla)

A Regi,
por las horas robadas

Prefacio

Durante los años 50-60 se desarrollaron numerosos estudios relativos a la teoría de sistemas continuos y discretos. Los sistemas discretos denominados autómatas finitos, y por este carácter finito descriptibles exhaustivamente, han conducido a la definición de un cierto número de nociones fundamentales, como la de estado o la de secuencia de estados.

La teoría de autómatas se desarrolló motivada por una serie de necesidades industriales culminando con la aparición de los computadores numéricos programables. En estas condiciones, y paralelamente con la teoría, surgieron diferentes métodos de aplicación. Del mismo modo que en otros muchos dominios científicos, las aplicaciones industriales de los sistemas secuenciales no aguardaron al desarrollo de una teoría para su utilización práctica. Revestían un carácter puramente intuitivo. Posteriormente, en el marco de problemas de pequeña dimensión, las representaciones tabulares de los sistemas secuenciales conocieron numerosas aplicaciones. Gracias a su generalidad, éstas constituyen hoy en día una excelente herramienta para la comprensión de las propiedades de los autómatas. No obstante, los problemas secuenciales industriales, al complicarse debido al crecimiento del número de los variables, aparecieron como más simples al nivel de las secuencias recorridas. Después de un cierto número de tanteos, se abordó hace algo más de un decenio la vía de las representaciones gráficas.

Los primeros trabajos de Petri datan de hace unos 20 años. Los informáticos los descubrieron hace un decenio, y los automáticos les siguieron algunos años después. Numerosos trabajos teóricos contribuyeron a la adaptación de las redes de Petri a la descripción de los autómatas. En el plano industrial, el desarrollo de los microprocesadores y de los autómatas programables suministraron inmensas perspectivas de aplicación. Esta gran efervescencia de ideas, a la que el Profesor Silva no es ajeno, ha podido conducir a una metodología bien ordenada de aplicación de las redes de Petri a la síntesis de sistemas secuenciales y concurrentes.

En 1980, los conceptos estaban suficiente y claramente establecidos para que una obra pedagógica pudiera llevarse a cabo. Se debe agradecer al Profesor Silva el hecho de haber realizado el esfuerzo de redactar la primera obra completa acerca de esta temática.

Nosotros quisiéramos destacar, en primer lugar, el interés pedagógico del texto, la ordenada progresión en el estudio de los conceptos abstractos y su paso bien articulado a aplicaciones concretas. Estas cualidades harán de este libro una obra de referencia no sólo para los estudiantes, sino también para los profesionales de la industria interesados por estos temas. De forma relevante, en esta obra se abordan todos los puntos importantes concernientes a las redes de Petri, aspectos de orden conceptual (formalización, simplificación y validación) y de orden práctico (realización programada y cableada). Se dedica, además, un amplio espacio a la realización, lo que le da un gran interés al libro. De este modo, se estudia con detalle la realización mediante autómatas programables y microcomputadores.

Desde una perspectiva clara se analiza, clasifica y expone la mayoría de los trabajos efectuados hasta la fecha sobre redes de Petri. Se debe reconocer al Profesor Silva el mérito de haber sabido recopilar un gran número de trabajos dispersos y de haberlos ordenado en la presentación. Este libro, que dará lugar presumiblemente a muchos otros, tiene el mérito no sólo de existir, sino de hacer penetrar las redes de Petri y los métodos que de ella se desprenden, tales como el Grafcet en Francia, en la práctica industrial.

Para terminar, sólo nos resta desear la rápida traducción a otras lenguas, en particular al inglés y al francés, de una obra tan útil.

JAVIER ARACIL
Catedrático.
Director del Depto. de Automática de la Escuela Técnica Superior de Ingenieros Industriales de la Univ. de Sevilla.

RENÉ PERRET
Professeur.
Ex-Directeur du Laboratoire d'Automatique de l'Institut Nationale Polytechnique de Grenoble.

Enero, 1982

Introducción

Las redes de Petri constituyen la más natural y potente extensión de los grafos de estado que se ha propuesto en la literatura técnica. Con relación a los grafos de estado, las redes de Petri permiten la representación clara y condensada del paralelismo y la sincronización, facilitando con ello la *descripción* o *modelación* de sistemas, así como su posterior realización.

Este texto se situa en los niveles de introducción a las aplicaciones y teoría de las redes de Petri, insistiendo especialmente sobre la obtención de descripciones y técnicas para su realización física. En este sentido queremos indicar que al tratarse de un texto para la enseñanza y consulta se han primado los aspectos didácticos evitando en todo momento el simple enciclopedismo.

La exposición se desarrolla pretendiendo conjugar intuición y rigor, evitando los aspectos meramente recetarios y tratando de evidenciar propiedades básicas. En cualquier caso, se han omitido determinadas generalizaciones que hubieran provocado una pérdida de claridad en la exposición al complicarse los enunciados, notaciones, etc.

Ahondando en las consideraciones anteriores, se han introducido gran cantidad de ejemplos de diversa complejidad, aunque de fácil aprehensión. De este modo, se llega a utilizar en numerosas ocasiones diferentes tipos de evoluciones de conjuntos de carros. Otros ejemplos típicos presentados son los lectores y redactores, los filósofos y los «spaghetti», sistemas del tipo productor-consumidor, etc. En resumen, los diferentes sistemas que se modelan a lo largo del texto han sido extraidos tanto del campo de la Automática como del de la Informática.

La creciente complejidad de los automatismos lógicos, así como de la concepción y utilización de los computadores digitales, etc., hace que aparezcan, cada vez con mayor frecuencia, subsistemas evolucionando e interaccionando simultáneamente. En estos casos, más que hablar de sistemas secuenciales, conviene hablar de *sistemas con evoluciones simultáneas* o *paralelas*, o *sistemas concurrentes**. La dificultad de la mente humana para dominar la concepción de los sistemas concurrentes hace extremadamente interesante la inserción de una etapa de análisis de los modelos obtenidos antes de proceder a su realización. Esta etapa de análisis se aborda en este texto bajo una óptica cualitativa (no cuantitativa) denominada *validación*. En un análi-

sis de validez se comprobará que el modelo del sistema que se concibe está exento de una serie de errores estructurales y dinámicos, aunque no se pretenderá demostrar su corrección. Si un modelo no es válido, se procederá a su comprobación y eventual modificación antes de pasar a realizarlo.

Las redes de Petri son una herramienta matemática aplicable al modelado de sistemas discretos concurrentes que admite una valiosísima *representación gráfica*, que sin lugar a dudas es uno de sus mayores atractivos desde el punto de vista industrial. Por otro lado, la realización de redes de Petri se puede llevar a cabo sin dificultad con cualquier *tecnología* (electrónica, fluídica, etc.). Las dos cualidades anteriores se completan con la potencia de la *teoría de validación* sobre ellas desarrollada, principalmente, a lo largo de la última década.

Las tres propiedades básicas anteriores permiten afirmar que las redes de Petri constituyen una muy interesante herramienta para la concepción de sistemas discretos concurrentes. Además, cabe señalar su capacidad para permitir la construcción de descripciones válidas de sistemas concurrentes mediante *refinamientos sucesivos* (*topdown*). De todo ello trataremos de convencer al lector a lo largo de los capítulos y anexos que siguen. No obstante, es importante considerar que, como en toda actividad de diseño, la utilidad de la herramienta depende sobremanera de la *metodología* con que se emplee, así como de la disponibilidad de *sistemas automáticos de ayuda* (*CAD, Computer Aided Design*). A estos aspectos hemos dedicado una especial atención. En particular, se presentan diversos algoritmos que permiten abordar el análisis, la simplificación, la realización, etc. de los modelos construidos durante el proceso de concepción. La codificación de los algoritmos presentados en el texto no debe plantear mayores dificultades, cualquiera que sea el lenguaje de programación que se adopte.

Antes de entrar en consideraciones sobre prerrequisitos para la lectura del libro o sobre su estructura, queremos señalar que al ser éste un texto de introducción, no han sido abordados algunos temas. Aquéllos relativos a *Complejidad y Decidabilidad*, así como *Lenguajes y redes de Petri*, se introducen en [PETE 81*a*]. El estudio de la evaluación de prestaciones (análisis cuantitativo) con redes de Petri, puede abordarse a partir de [FLOR 81], [RAMA 80] y [SIFA 77]. Por último, es importante resaltar que visando una mayor potencia de descripción o/y la obtención de modelos más compactos se han propuesto en la literatura numerosas extensiones se presentan rápidamente en el capítulo 2. Entre otras extensiones, previsiblemente de gran importancia práctica en los años venideros, están las redes Predicado-Transición [GENR 79] y las redes de Petri coloreadas [JENS 81].

PRERREQUISITOS PARA LA LECTURA

En la redacción del texto ha presidido la idea de que éste sea autocontenido (su lectura necesita unos prerrequisitos mínimos).

En lo concerniente a la realización se ha supuesto que el lector conoce los fundamentos de los sistemas lógicos combinacionales (álgebra de BOOLE, diagramas de KARNAUGH y realización de funciones lógicas con puertas). De este modo, en el capítulo 7 se realiza la definición de las memorias muertas (ROM) y de las matrices lógicas

programables (PLA), así como se presenta su aplicación a la realización de funciones lógicas. También se ha supuesto al lector mínimamente familiarizado con los biestables básicos (aunque éstos se definen en el texto) y los contadores.

En las cuestiones más abstractas se ha supuesto al lector conocedor de los aspectos básicos del álgebra lineal y del ya mencionado álgebra de BOOLE. En cualquier caso, el anexo 3 recuerda los conceptos y resultados del álgebra lineal que se utilizan. El anexo 2 presenta elementos de la teoría de grafos, convenientes para la lectura del capítulo 4.

Siendo casi exhaustivos, diremos que para interpretar con facilidad los algoritmos expuestos en el texto, es interesante que el lector tenga una, aunque mínima, experiencia en la programación de un computador digital. Ello se hace especialmente necesario para la lectura del capítulo 9, dedicado a la programación de redes de Petri en microcomputadores.

ESTRUCTURA DEL LIBRO

Los capítulos 1, 2, 3 y 5 y el anexo 1 abordan, bajo diversos puntos de vista, la descripción o modelación de sistemas discretos concurrentes. Los capítulos 4 y 5 y los anexos 2, 3, 4 y 5 profundizan sobre la validación de los modelos. Por último, los capítulos 6, 7, 8 y 9 presentan un amplio conjunto de técnicas de realización.

El primer capítulo es introductorio, esencialmente motivador. En él se presentan de forma intuitiva y simplificada nociones generales sobre aplicaciones y propiedades de las redes de Petri. En particular, se ha desarrollado con especial cuidado la evolución desde los métodos de descripción tabulares (tablas de estado y de fases) a los grafos de estado reducidos y, por último, a las redes de Petri. Es decir, se establecen conexiones con herramientas clásicas de descripción, exponiéndose algunas de las aportaciones prácticas de las redes de Petri.

El segundo capítulo presenta los principales conceptos relacionados con las redes de Petri. En particular se insiste sobre la diferenciación de las redes como estructuras matemáticas y como herramientas de modelación de sistemas. Para que una red de Petri autónoma (estructura matemática) represente un sistema dinámico hace falta dotar su evolucioń de un significado. Así, una red de Petri dotada de una interpretación adecuada puede representar funcionalmente un sistema lógico, un programa de computador, etc. En estos casos el modelo describirá una interacción entre el sistema y un medio externo; la evolución de la red de Petri será no-autónoma (dependerá del estado del medio externo). Una red autónoma sólo representa restricciones funcionales sobre la evolución (secuenciamientos, compartición de recursos, etc.). En §2.3 se detalla una posible interpretación para la modelación de automatismos lógicos concurrentes. El anexo 1 presenta otra interpretación posible, destinada a la modelación de la estructura de control de programas concurrentes de computadores digitales.

En el capítulo 3, huyendo de planteamientos rígidos que conduzcan a minimizaciones materiales (hoy en día sin gran sentido práctico), se estudia la simplificación de los modelos construidos con redes de Petri. La lectura del capítulo 3 es, esencialmente, una «gimnasia» muy recomendable para el modelado de sistemas con redes

de Petri. Los métodos de simplificación denominados de fusión de lugares (§3.4) y de eliminación de lugares fuente (§3.5) son específicos a la interpretación que permite describir sistemas lógicos.

El capítulo 4 estudia la validación de redes de Petri autónomas. Se trata de un capítulo que resultará ser el de lectura más onerosa, dada su longitud y temática. No obstante, su estudio es absolutamente fundamental para todo aquél que desee comprender las bases del análisis cualitativo de modelos construidos con redes de Petri.

El capítulo 5, redactado de forma muy intuitiva, consta de dos partes claramente diferenciadas. En la primera de ellas se advierte al lector de las diferencias que pueden observarse entre el análisis de las redes autónomas y el de las redes no-autónomas. En la segunda parte se sugieren metodologías para la construcción de modelos, apoyadas en los resultados del análisis. Es decir, a diferencia de la presentación de modelos realizada en los capítulos 1 y 2 y el anexo 1, que es descriptiva-justificativa, en este capítulo se presentan nociones sobre como llegar a la construcción de «buenos» modelos.

Los capítulos 6 y 7 abordan la realización cableada (con puertas, biestables y contadores, SSI y MSI) y la realización con memorias muertas (ROM) y matrices lógicas programables (PLA). En este último capítulo se considera a las ROM y PLA bajo dos ópticas distintas: (1) como sustitutos directos de la lógica aleatoria (puertas), lo que origina las técnicas de realización denominadas de transición directa y (2) como componentes de un sistema microprogramado. En el capítulo 6 se insiste sobre la realización cableada modular, técnica que simplifica al máximo el diseño, puesta a punto, mantenimiento, etc., a costa de conducir a realizaciones no mínimas en componentes. Es decir, salvo para grandes series de productos no modificables, se trata de optimizar el coste de los equipos con respecto a su ciclo de vida (concepción, producción, utilización) y no con respecto al material empleado (número de puertas, biestables, etc.).

Los capítulos 8 y 9 presentan la realización programada de redes de Petri. En particular, el primero de ellos estudia con cierto nivel de detalle los Autómatas Programables. El capítulo 9 esboza la realización de redes de Petri con computadores de propósito general.

La relativa novedad de la temática tratada, ha hecho que nos hayamos visto obligados a introducir nuevos conceptos y a adaptar términos. En este sentido, esperamos que el anexo 6, glosario trilingüe, sea de utilidad.

El texto ha sido concebido para su lectura en secuencia (capítulo por capítulo); no obstante, el lector impaciente por sumergirse en las técnicas de realización puede proceder sin grandes dificultades al estudio de los capítulos 6 a 9 después de haber considerado los capítulos 1 y 2. En cualquier caso, los apartados señalados con un asterisco pueden ser saltados en una primera lectura.

AGRADECIMIENTOS

La génesis de este texto se encuentra en las notas redactadas con motivo de la explicación de un curso de doctorado en la E.T.S. de Ingenieros Industriales de la Universidad de Zaragoza (año académico 78-79) y en el material preparado para impar-

tir, bajo el patrocinio de la E.T.S. de Ingenieros Industriales de la Universidad de Sevilla y el Colegio Oficial de Ingenieros Industriales de Andalucía Occidental y Badajoz, un Seminario de Perfeccionamiento Tecnológico (Octubre, 1979).

Una primera elaboración de las mencionadas notas fué distribuida en la primavera de 1980, habiendo contado con constructivas críticas de E. Fernández (E.T.S.I. Industriales de la Universidad de Sevilla), T. Lang (Facultad de Informática de la Universidad Politécnica de Barcelona), T. Pollán y M. Gatti (E.T.S.I. Industriales de la Universidad de Zaragoza), J.M. Vela y G. León (E.T.S.I. Telecomunicación de la Universidad Politécnica de Madrid).

En la casi diaria tarea de mejora de las sucesivas versiones, incluida la primera, he contado con la inapreciable ayuda de J. Martínez y S. Velilla, ambos colaboradores en las tareas docentes e investigadores del Depto. de Automática de la E.T.S.I. Industriales de la Universidad de Zaragoza. Parte de los resultados de sus tesis doctorales se presentan en este texto. A. Martínez, J. M. Colom, L. Montano y P. Pardos, han revisado la última versión. Diversos alumnos de 5.° curso, cuyos nombres omito por temor a olvidar algunos, han colaborado también con sus constructivas críticas.

Zaragoza
Enero, 1982

M. SILVA

Símbolos más utilizados

1 Generales

(*a*) *Conjuntos*

$\{e_1, e_2, \ldots, e_n\}$	Conjunto formado por los elementos $e_1, e_2, \ldots, e_n$ (notación por extensión)
$\{e \mid r\}$	Conjunto formado por los elementos e que cumplen la(s) propiedad(es) r (notación por comprensión)
$\in$	Pertenencia
$\notin$	No pertenencia
$\exists$	Existe (cuantificador existencial)
$\forall$	Para todo (cuantificador universal)
$\lvert P \rvert$	Cardinal del conjunto P (número de elementos que posee)
$\varnothing$	Conjunto vacío ($\lvert\varnothing\rvert = 0$).
$\mathbb{N}$	Conjunto de los números naturales: $\{0, 1, 2, 3, \ldots\}$
$\mathbb{N}^+$	Conjunto $\{1, 2, 3, \ldots\}$
$\mathbb{Z}$	Conjunto de los números enteros: $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$
$\mathbb{Q}$	Conjunto de los números racionales: $\{n_i/n_j \mid n_i, n_j \in \mathbb{Z}\}$
$\mathbb{Q}^+$	Conjunto de los números racionales positivos
$\mathbb{R}$	Conjunto de los números reales
$A \cup B$	Unión de los conjuntos A y B
$A \cap B$	Intersección de los conjuntos A y B
$A \times B$	Producto cartesiano de A por B [conjunto de los pares (a, b), $a \in A$ y $b \in B$]
$A \subset B$	Inclusión del conjunto A en el conjunto B

(*b*) *Lógica*

$\Rightarrow$	Implicación lógica
$\Leftrightarrow$	Equivalencia Condición necesaria y suficiente (CNS) Si y sólo si (sii)
$\wedge$	Conjunción
$\vee$	Disyunción
$\neg$	Negación

(*c*) *Vectores*

$X > Y$	sii $\forall i\ X(i) > Y(i)$
$X \geqslant Y$	sii $\forall i\ X(i) \geqslant Y(i)$
$X \gneqq Y$	sii $X \geqslant Y$ y $X \neq Y$

(*d*) *Matrices*

I_n	Matriz identidad de rango n
A^T	Matriz traspuesta de la matriz A
A^{-1}	Matriz inversa de la matriz A

(*e*) *Varias*

$\lceil\ \rceil$	Redondeo por exceso
$\lfloor\ \rfloor$	Redondeo por defecto

2 Redes de Petri

P	Conjunto de lugares de una RdP ($\|P\| = n$)
p, p_i	Lugar, i-ésimo lugar
T	Conjunto de transiciones de una RdP ($\|T\| = m$)
t, t_j	Transición, j-ésima transición
$z_j = Z(t_j)$	Duración asociada al disparo de la transición t_j
α	Función de incidencia previa ($\alpha: P \times T \rightarrow \mathbb{N}$)
β	Función de incidencia posterior ($\beta: T \times P \rightarrow \mathbb{N}$)
$\boldsymbol{R}$	Red de Petri ($\boldsymbol{R} = \langle P, T, \alpha, \beta \rangle$)
${}^{\bullet}t$	Conjunto de lugares de entrada a t
$t^{\bullet}$	Conjunto de lugares de salida de t
${}^{\bullet}p$	Conjunto de transiciones de entrada a p
$p^{\bullet}$	Conjunto de transiciones de salida de p
M	Marcado de una RdP
$M(p_i), m_i$	Marcado del i-ésimo lugar
M_0	Marcado inicial

$\langle \boldsymbol{R}, M_0 \rangle$	Red de Petri marcada
$M \xrightarrow{t}$	La transición t es sensibilizada por el marcado M
$M_i \xrightarrow{t} M_j$	El marcado M_j es alcanzado a partir del marcado M_i al disparar la transición t
σ	Secuencia de transiciones ($\sigma = t_i t_j \ldots t_q$)
$\bar{\sigma}$	Vector característico asociado a la secuencia σ [$\dim(\sigma) = m$, número de transiciones]
$\bar{\sigma}(t_i)$	Número de ocurrencias de t_i en la secuencia σ
$\boldsymbol{L}(\boldsymbol{R}, M_0)$	Conjunto de secuencias disparables en $\langle \boldsymbol{R}, M_0 \times$
$\boldsymbol{M}(\boldsymbol{R}, M_0)$	Conjunto de marcados alcanzables en $\langle \boldsymbol{R}, M_0 \langle$
$\bar{\boldsymbol{L}}(\boldsymbol{R}, M_0)$	Conjunto de vectores característicos asociados a $\boldsymbol{L}(\boldsymbol{R}, M_0)$
$\boldsymbol{G}(\boldsymbol{R}, M_0)$	Grafo de marcados generado por $\langle \boldsymbol{R}, M_0 \rangle$
$\boldsymbol{C}^-$	Matriz que representa la función de incidencia previa
$\boldsymbol{C}^+$	Matriz que representa la función de incidencia posterior
$\boldsymbol{C} = \boldsymbol{C}^+ - \boldsymbol{C}^-$	Matriz que representa la estructura de una RdP pura
$\boldsymbol{C}_j$	j-ésima columna de $\boldsymbol{C}$ [$\boldsymbol{c}_j = (c_{1j} c_{2j} \ldots c_{nj})^T$]
$\boldsymbol{I}_i, \boldsymbol{I}(p_i)$	i-ésima línea o fila de $\boldsymbol{C}$ [$\boldsymbol{I}_i = (c_{i1} c_{i1} \ldots c_{im})$]
ME	Máquina de estados
GE	Grafo de estados
GM	Grafo marcado o de sincronización
RLE	Red de Petri libre elección
RS	Red de Petri simple
RdPAI	Red de Petri con arcos inhibidores
RdPG	Red de Petri generalizada. (También se representa por RdP.)
RdPC	Red de Petri con capacidades
RdPT	Red de Petri temporizada
RdPI	Red de Petri interpretada

Contenido

Prefacio v

Introducción vii

Prerrequisitos para la lectura viii
Estructura del libro ix
Agradecimientos x

Símbolos más utilizados xiii

Capítulo 1

Introducción a las Redes de Petri como herramienta de descripción funcional de sistemas secuenciales y concurrentes 1

1.1 Introducción 1
1.2 Separación entre Parte de Control y Parte Operativa 2
1.3 Representación tabular de un sistema secuencial de estados finitos 3
1.3.1 Autómatas de estados finitos 4
1.3.2 Descripción de sistemas lógicos secuenciales asíncronos: la tabla de fases 6
1.3.3 Ejemplo de utilización 8
1.3.4 Crítica a los métodos tabulares de descripción 9
1.4 Representación de un sistema secuencial asíncrono mediante un grafo de estados reducido 10
1.4.1 Presentación intuitiva de los grafos reducidos 10
1.4.2 Obtención de la tabla de fases a partir del grafo reducido. Cantidad de información contenida en la descripción 11
1.4.3 Ejemplos de modelación 12
1.4.4 Crítica al grafo reducido (GR) 15
1.5 Aplicación de las redes de Petri a la modelación funcional de sistemas concurrentes 16
1.5.1 Definición del modelo de descripción 16

1.5.2 Ejemplos de modelación de sistemas concurrentes 17
1.5.3 Configuraciones y propiedades básicas 20
1.5.3.1 Configuraciones en un RdP 20
1.5.3.2 Propiedades básicas 21
1.5.4 Obtención de un grafo reducido a partir de una RdP 23
1.6 Comentarios en torno a la utilización de las RdP en la modelación de sistemas concurrentes 23
Ejercicios 25

Capítulo 2

Redes de Petri: formalización y aplicación a la modelación funcional de sistemas concurrentes 29

2.1 Introducción 29
2.2 Redes de Petri Autónomas 30
2.2.1 Terminología básica 30
2.2.1.1 Conceptos estructurales 30
2.2.1.2 Conceptos dinámico-estructurales 32
2.2.2 Ecuación de estado de una red de Petri 34
2.3 Las Redes de Petri como modelo de descripción de sistemas lógicos concurrentes 35
2.3.1 Interpretación asociada a las RdP 36
2.3.2 Ejemplo de modelación (recurso compartido por dos usuarios) 37
*2.3.3 Transformaciones sobre condiciones externas y eventos que no alteran una descripción 37
2.4 Ejemplos típicos de modelación con Redes de Petri 40
2.4.1 Relación productor-consumidor con exclusión mutua 40
2.4.2 Secuencias alternadas 42
2.4.3 Reutilización de secuencias de funcionamiento (subprogramas) 45
2.4.4 Lectores y redactores 45
2.5 Redes de Petri ordinarias: principales subclases y extensiones 47
2.5.1 Subclases de Redes de Petri ordinarias 48
*2.5.2 Extensiones de las Redes de Petri ordinarias 50
2.5.2.1 Redes de Petri generalizadas 51
2.5.2.2 Redes de petri con capacidad limitada 52
2.5.2.3 Redes de Petri con arcos inhibidores 55
2.5.2.4 Redes de Petri y transiciones no estándar 59
2.5.2.5 Redes de Petri coloreadas 60
2.5.2.6 Redes de Petri temporizadas (RdPT) 61
Ejercicios 62

Capítulo 3

Simplificación de una descripción 63

3.1 Introducción 63

3.2 Clasificación de los tipos de simplificación 64
3.2.1 Simplificaciones estructurales 64
3.2.2 Simplificaciones que tienen en cuenta la interpretación asociada a la RdP 64
3.3 Método de los lugares implícitos 65
3.3.1 Concepto de lugar implícito y simplificación de la RdP 65
*3.3.2 Búsqueda de los lugares implícitos 66
3.3.3 Asignación de las acciones asociadas a un lugar implícito 72
*3.4 Método de fusión de lugares 75
3.4.1 Lugares equivalentes o directamente fusionables 75
3.4.2 Lugares compatibles 76
3.4.2.1 Introducción 76
3.4.2.2 Compatibilidad y conjuntos de lugares fusionables 78
3.4.2.3 Determinación de los eventos y las condiciones sobre las salidas asociadas al lugar de fusión 80
*3.5 Método de los lugares fuente 83
3.5.1 Generalidades y terminología 83
3.5.2 Presentación del método de los lugares fuente 84
3.5.3 Aplicación 85
3.6 Ejemplo y Conclusión 87
Ejercicios 88

Capítulo 4

Validación funcional de una descripción (I): Redes de Petri autónomas 91

4.1 Introducción 91
4.2 Propiedades básicas que caracterizan el funcionamiento de los sistemas con evoluciones simultáneas 92
4.2.1 Vivacidad 92
4.2.2 Ciclicidad 94
4.2.3 Limitación 94
4.2.4 Conflictividad 97
4.2.5 Exclusión mutua 98
4.2.6 Relaciones sincrónicas 98
4.3 Clasificación de los métodos de análisis 100
4.4 Análisis por enumeración: grafo de marcados 102
4.4.1 Grafo de marcados: construcción 102
4.4.2 Análisis a partir del grafo de marcados 106
4.4.2.1 Análisis de la vivacidad 106
4.4.2.2 Análisis de la ciclicidad 109
4.4.2.3 Análisis de la conflictividad 109
4.4.2.4 Otros análisis 109
4.4.3 Crítica de los métodos de análisis por enumeración 110
4.5 Análisis por reducción 110
*4.5.1 Reducción de una subRdP a un lugar ($\mathcal{R}_1$) 111
4.5.1.1 Regla de reducción 111
4.5.1.2 Propiedades que preserva la regla de reducción 114

4.5.1.3 Obtención de los macrolugares 115
*4.5.2 Sustitución de un lugar ($\mathfrak{R}_2$) 118
4.5.2.1 Lugar sustituible y operación de sustitución 118
4.5.2.2 Consecuencias de la regla de reducción 121
*4.5.3 Eliminación de una transición ($\mathfrak{R}_3$) 122
4.5.4 Cuestiones adicionales sobre reducción y limitación de RdP 123
4.5.4.1 Eliminación de lugares implícitos y limitación 123
4.5.4.2 Estrategia para la utilización de las reglas de reducción 124
4.6 Conservatividad y repetitividad: aplicación al análisis global de redes de Petri 128
4.6.1 Definiciones y propiedades básicas 129
4.6.2 Relación entre propiedades estructurales y propiedades dinámico-estructurales 133
4.7 Invariantes de marcado y de disparo: aplicación al análisis local de Redes de Petri 137
4.7.1 Alcanzabilidad y base de invariantes lineales de marcado 138
4.7.1.1 Fundamentos y aplicaciones 138
4.7.1.2 Limitación del método de análisis e interés de los invariantes no negativos 141
4.7.2 Invariantes no negativos: componentes conservativas 142
4.7.2.1 Definiciones y propiedades básicas 143
*4.7.2.2 Obtención de todas las componentes elementales de una RdP 145
4.7.2.3 Aplicaciones de las componentes conservativas al análisis de RdP ... 150
*4.7.3 Invariantes de disparo 157
*4.8 Cerrojos y Trampas 162
4.8.1 Definiciones y propiedades básicas 163
4.8.2 Obtención de los cerrojos y de las trampas de una RdP 166
4.8.2.1 Obtención de los cerrojos 166
4.8.2.2 Obtención de las trampas 169
4.8.2.3 Obtención de los conjuntos de lugares que son simultáneamente, cerrojo y trampa 169
4.9 Conclusiones 169
Ejercicios 170

Capítulo 5

Validación funcional de una descripción (II): Redes de Petri no autónomas. Impacto sobre la descripción 173

5.1 Introducción 173
5.2 Análisis de las RdP temporizadas 173
5.2.1 Estudio de la limitación 173
5.2.2 Estudio de la vivacidad 174
5.2.3 Cuestiones adicionales sobre la vivacidad 176
5.3 Análisis de las RdP interpretadas 177
5.3.1 Estudio de la limitación 178
5.3.2 Estudio de la vivacidad 179
5.3.3 RdP no autónomas: marcados alcanzables y secuencias disparables ... 179
5.4 Validación dinámica de una descripción 180

5.5 Métodos de descripción de sistemas complejos: aspectos básicos 181
5.5.1 Descripción descendente («top-down») 182
5.5.1.1 Descripción descendente y reglas básicas de expansión 183
5.5.1.2 Descripción descendente y subRdP 186
5.5.2 Descripción modular 190
5.6 Conclusión 193
Ejercicios 194

Capítulo 6

Realización cableada 195

6.1 Introducción 195
6.2 Realización asíncrona (I): funcionamiento por transferencia impulsional 198
6.2.1 La célula de memoria 198
6.2.2 Realización por transferencia impulsional 202
*6.2.3 Cuestiones adicionales de índole práctica 204
6.2.3.1 Realización cuando el evento es un flanco 204
6.2.3.2 Realización de una temporización 205
*6.2.4 Análisis del comportamiento dinámico de los circuitos lógicos diseñados y estudio de fenómenos aleatorios 206
*6.3 Realización asíncrona (II): funcionamiento por llamada-respuesta 212
6.3.1 Conceptos básicos 212
6.3.2 Limitación del funcionamiento por llamada-respuesta cuando se utiliza la condición de desactivación simplificada 215
6.3.3 A modo de resumen 216
6.4 Realización síncrona 217
6.4.1 Síntesis utilizando el biestable J-K 220
6.4.2 Síntesis utilizando el biestable D 220
6.4.3 Síntesis utilizando el biestable T 221
6.4.4 A modo de resumen 221
*6.4.5 Realización asíncrona cuando el evento es un flanco 222
6.5 Cuestiones adicionales sobre realización cableada 222
6.5.1 Las acciones (salidas) 222
6.5.2 Interés de la simplificación de la descripción 222
6.5.3 Realizaciones autoverificables («self-checking») 225
6.5.3.1 Método de ponderación 225
6.5.3.2 Utilización de códigos de Hamming 226
6.6 Conclusiones 226
Ejercicios 227

Capítulo 7

Realización con memorias y matrices lógicas programables ... 229

7.1 Introducción 229
7.2 ROM y PLA: definición funcional y síntesis directa de funciones lógicas combinacionales 230

7.2.1 Memorias ROM 230
7.2.1.1 Definición funcional 230
7.2.1.2 Aplicación a la síntesis directa de funciones lógicas combinacionales 231
7.2.2 Matrices lógicas programables: PLA 234
7.2.2.1 Definición funcional y aplicación a la síntesis directa de funciones lógicas combinacionales 234
7.2.2.2 Cuestiones adicionales sobre síntesis directa de funciones lógicas combinacionales 236
7.2.3 Reducción de la longitud de la palabra en ROM o PLA 238
7.2.3.1 Codificación total 239
7.2.3.2 Codificación independiente por campos 241
7.3 Realización de grafos reducidos 243
7.3.1 Métodos lógicos de transición directa 244
7.3.1.1 Síntesis utilizando registros constituídos por biestables D 245
7.3.1.2 Síntesis utilizando registros constituídos por biestables T 247
7.3.2 Reducción del número de variables de entrada a la memoria o matriz lógica en los métodos de transición directa 248
7.3.2.1 Planteamiento básico 248
7.3.2.2 Máquina de decisiones explícitas 2^v-arias 250
7.3.3 Métodos básicos con secuenciador: máquinas de decisiones binarias .. 252
7.3.3.1 Grafo reducido equivalente con alternativas binarias: transformación 254
7.3.3.2 Instrucción única y direcciones explícitas 254
7.3.3.3 Instrucción única y una dirección implítica 257
7.3.3.4 Dos instrucciones y direcciones explícitas 258
7.3.3.5 Dos instrucciones y direcciones implícitas 259
7.3.3.6 Realización asícrona-autosincronizada: ejemplo 260
7.3.4 A modo de resumen 262
7.4 Realización de redes de Petri 263
7.4.1 Métodos lógicos de transición directa 264
7.4.1.1 Codificación total del marcado 264
7.4.1.2 Codificación independiente por campos del marcado 265
7.4.2 Realización con máquinas de decisiones binarias 267
7.4.3 Descomposición de una RdP binaria en GR 269
7.4.3.1 Descomposición en componentes y cobertura de los macrolugares (ML) 269
*7.4.3.2 Descomposición en RdP binarias y vivas 275
7.5 Conclusión 276
Ejercicios 277

Capítulo 8

Realización programada (I): autómatas programables generales 279

8.1 Introducción 279
8.2 Elementos sobre la estructura y el funcionamiento de los autómatas programables 280
8.2.1 Estructura de un AP. Generalidades 281
8.2.2 Unidad de memoria 282
8.2.3 Unidad de entrada/salida 282
8.2.4 Unidad central de procesamiento 283

8.2.5 Periféricos 285
8.2.6 Funcionamiento de un AP 285
8.2.6.1 Desarrollo de una instrucción 285
8.2.6.2 Desarrollo de un programa: ciclo de tratamiento 286
8.3 Lenguajes de Programación 287
8.3.1 Lenguaje basado en los esquemas de contactos (ladder diagrams) 289
8.3.2 Lenguaje basado en los diagramas lógicos 289
8.3.2.1 El lenguaje 290
8.3.2.2 Programación síncrona y no-síncrona de RdP 292
8.3.3 Lenguajes basados en la notación booleana 295
8.3.3.1 Lenguaje exclusivamente basado en la notación algebraica y programación de RdP 296
8.3.3.2 Lenguaje provisto de saltos condicionales y programación de RdP .. 296
*8.4 Presentación simplificada de algunos autómatas programables 297
8.4.1 Presentación general 298
8.4.2 Interpretador de diagramas lógicos (M_1) 299
8.4.3 Interpretador de cadenas postfijas (máquina de pila) (M_2) 301
8.4.4 Máquina basada en el concepto de operación lógica (M_3) 306
8.4.5 Interpretador mixto de operaciones lógica y saltos condicionales (M_4) . 308
8.4.6 Comentarios generales 311
8.5 Conclusión 312
Ejercicios 313

Capítulo 9

Realización programada (II): microcomputadores y autómatas programables especiales 315

9.1 Introducción 315
9.2 Lenguajes de especificación de RdP 316
9.2.1 Lenguajes 316
9.2.2 Representación interna de la RdP y traducción 319
9.3 Cuestiones previas a la realización de sistemas especializados en la simulación de RdP binarias 320
9.3.1 Computador soporte 320
9.3.2 Definición de las entradas y salidas 321
9.3.3 Simulación síncrona/no síncrona 321
9.3.4 Sobre la interpretación asociada a las RdP que se simulan 322
9.4 Método matricial básico para la simulación de RdP binarias 323
9.4.1 Fundamentos 323
9.4.2 Prestaciones 325
9.4.3 Una realización del simulador matricial 326
9.5 Representaciones basadas en listas (I): esquema básico 331
9.5.1 Nociones previas 332
9.5.2 Esquema básico de representación de RdP con listas 333
9.5.3 Prestaciones 336
9.6 Representaciones basadas en listas (II): simulación dirigida por el marcado .. 337
9.6.1 Simulación de grafos reducidos 337
9.6.2 Simulación de RdP dirigida por escrutación del vector de marcado ... 341

9.6.3 Codificación de la estructura de una RdP: lugares representantes y lugares de sincronización 349
9.6.4 Simulación de RdP por escrutación de la lista de lugares representantes marcados 353
9.6.5 Comentarios generales 357
*9.7 Representaciones basadas en listas (III): simulación dirigida por las transiciones sensibilizadas 363
*9.8 Un autómata programable con lenguaje especial 366
9.8.1 Generalidades 366
9.8.2 Estructura de un programa 366
9.8.3 Presentación del lenguaje de programación y funcionamiento global del sistema 367
9.8.3.1 Funciones combinacionales 367
9.8.3.2 Instrucciones especiales para la programación de RdP 368
9.9 Conclusiones 371
Ejercicios 372

Anexo 1

Redes de Petri y programación de computadores 373

Anexo 2

Elementos sobre los grafos 377

A.2.1 Concepto de grafo. Representación y algunas interpretaciones 377
A.2.2 Terminología básica 379
A.2.3 Obtención de las componentes conexas y las fuertemente conexas de un grafo 380

Anexo 3

Sistemas de ecuaciones lineales: obtención de bases de anuladores de una matriz 383

A.3.1 Introducción: sistemas de ecuaciones lineales y homogéneas 383
A.3.2 Transformaciones elementales 384
A.3.3 Obtención de bases de anuladores 384

Anexo 4

Cómputo eficiente de todas las componentes elementales de una RdP generalizada 387

A.4.1 Introducción 387
A.4.2 Caracterización de las componentes conservativas elementales 388
A.4.3 Codificación del algoritmo 389

Anexo 5

Soluciones enteras no negativas de un sistema de ecuaciones e inecuaciones lineales 395

A.5.1 Introducción 395
A.5.2 Sistema homogéneo de ecuaciones e inecuaciones: variables de holgura 395
A.5.3 Sistema no homogéneo de ecuaciones e inecuaciones 397

Anexo 6

Léxico sobre Redes de Petri 399

Bibliografía 407

Índice 421

1

Introducción a las Redes de Petri como herramienta de descripción funcional de sistemas secuenciales y concurrentes

1.1 INTRODUCCIÓN

Describir un sistema es una tarea que consiste en elaborar un modelo del mismo. Según sean los objetivos que se persigan, de un sistema se pueden construir modelos de diferentes tipos. Así, un modelo que enumera las partes del sistema (subsistemas) y sus interconexiones se denomina *estructural*. Un modelo que describe cómo opera o funciona un sistema se denomina *funcional*. En un modelo funcional se especifican las reacciones del sistema frente a eventos o acontecimientos que provienen del exterior.

A lo largo de todo este texto nos preocuparemos de *construir* modelos funcionales, *analizarlos* y *realizarlos físicamente*. En este capítulo introducimos de forma gradual algunos problemas que surgen al modelar, con diversas herramientas de descripción, sistemas (lógicos) secuenciales. Terminamos presentando informalmente las redes de Petri provistas de una interpretación adaptada al modelado de sistemas lógicos. Las redes de Petri (RdP) interpretadas permiten expresar, de forma clara y rigurosa, objetivos de funcionamiento y seguridad funcional de los sistemas objeto de estudio. En particular, las RdP son una herramienta de descripción fácilmente comprensible por el futuro usuario del sistema, mediante la cual le es posible formalizar su diálogo con el diseñador.

El capítulo se desarrolla a partir de la introducción de un concepto básico en el modelado: la separación en *parte de control* y *parte operativa*. Posteriormente se entra de lleno en la definición y utilización de herramientas para el modelado de la parte de control. La figura 1.1 presenta la estructura de esta parte. La descripción de sistemas discretos en los que se identifica un número finito de situaciones o estados se puede llevar a cabo gracias a una herramienta matemática denominada *autómata finito* (§1.3.1), AF. La descripción tabular de un AF cuando se modelan sistemas lógicos asíncronos se denomina *tabla de fases* (§1.3.2). La complejidad de la descripción basada en la tabla de fases lleva a introducir los *grafos reducidos* (§1.4), GR. Un GR puede modelar exactamente la misma clase de sistemas que una tabla de fases, pero las descripciones son, normalmente, más fáciles de establecer (dado

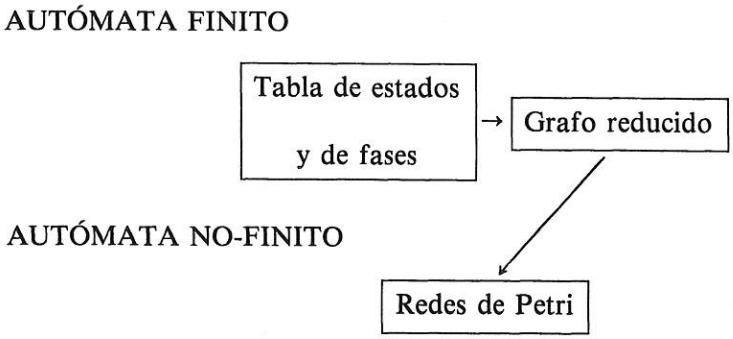


Figura 1.1. Herramientas de modelado de sistemas que se consideran a lo largo del capítulo.

que no se pretende la descripción exhaustiva del sistema, sino sólo recoger información necesaria y suficiente para abordar la realización).

Cuando un sistema se compone de varios subsistemas que evolucionan simultáneamente, se dice que es un sistema *concurrente* o con *evoluciones paralelas*†. La descripción secuencial de un sistema concurrente se complica en gran medida, por lo que los GR son de una utilidad restringida. En este punto se introducen las redes de Petri, RdP, como una herramienta de modelado que facilita la descripción. Además, las RdP tienen mayor potencia descriptiva que las tablas de fases y los grafos reducidos, pues mediante ellas se pueden describir determinados (no todos) sistemas en los que el número de estados no es finito.

1.2 SEPARACIÓN ENTRE PARTE DE CONTROL Y PARTE OPERATIVA

Todo sistema se puede representar y realizar mediante la interconexión de dos subsistemas o partes que cooperan (figura 1.2):

—la *parte operativa* (PO),
—la *parte de control* o *mando* (PC).

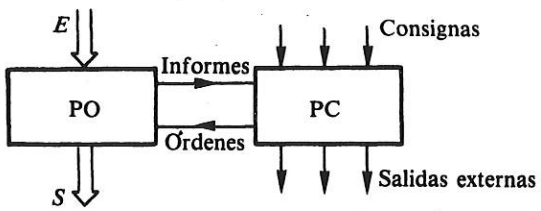


Figura 1.2. Descomposición PC-PO de la descripción de un sistema.

† Se podrían matizar diferencias entre concurrencia y paralelismo, pero en este texto utilizaremos ambos términos como sinónimos.

Una separación PC-PO en un sistema no hace más que establecer una distinción entre un subsistema de *ejecución* (PO) y un subsistema de *dirección* (PC). La PO emite hacia la PC informes sobre su situación y la PC, en función de esta información, emite órdenes hacia la PO.

La separación PC-PO es un concepto clásico que pretende facilitar la descripción aplicando el aforismo «divide y vencerás». En efecto, la descripción por partes simplifica la construcción del modelo total. Tomando como ejemplo un sistema máquina herramienta con control numérico, es posible considerar como PC al equipo de control numérico. Evidentemente, este automatismo numérico puede a su vez ser considerado como un sistema que se puede descomponer en PC-PO (figura 1.3).

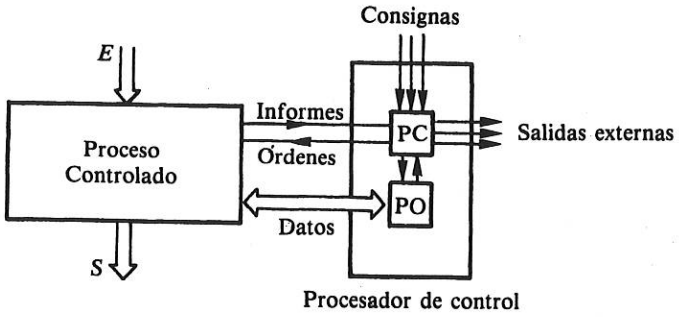


Figura 1.3. Descomposición del procesador de control.

Al describir un sistema, se pueden aplicar sucesivas descomposiciones PC-PO que permitan pasar de una cierta descripción a otra más detallada. La metodología de descripción denominada *descripción por refinamientos sucesivos* consiste en reaplicar el esquema siguiente tantas veces como se juzgue necesario:

1) definir una nueva separación PC-PO sobre la PO original, descomponiendo las acciones en otras más elementales; y
2) llevar a la PC el secuenciamiento de las operaciones en que se descompusieron las acciones habidas anteriormente.

En resumen, la separación PC-PO facilita la descripción. Esto resulta evidente al considerar automatismos numéricos (entre los que se encuentran los computadores digitales programables), puesto que existen subsistemas típicos que forman parte de la PO (sumadores, contadores, etc.). La descripción por refinamientos sucesivos se basa en sucesivas separaciones PC-PO; ésta se abordará en detalle en el capítulo 5. En lo sucesivo se considera esencialmente el modelado de la PC.

1.3 REPRESENTACIÓN TABULAR DE UN SISTEMA SECUENCIAL DE ESTADOS FINITOS

La representación de sistemas secuenciales de estados finitos se basa en una herra-

mienta matemática denominada *autómata finito*. La introduciremos en §1.3.1, mientras que en §1.3.2 estudiaremos su adecuación al modelado de sistemas (secuenciales de estados finitos) asíncronos. La modelación de un automatismo muy simple permitirá ilustrar las ideas básicas, así como realizar una primera crítica de los métodos tabulares de representación.

1.3.1 Autómata finito

Según el diccionario de la Real Academia de la Lengua (19.ª edición, 1970), un *autómata* es:

—[un] instrumento o aparato que encierra dentro de sí el mecanismo que le imprime determinados movimientos;

—[una] máquina que imita la figura y los movimientos de un ser animado.

Es decir, en ambas acepciones se evoca la imagen de movimiento. En lo que sigue nos ocuparemos del aspecto de tratamiento de información subyacente y no del aspecto de simulación del movimiento.

Precisando el significado de «autómata» en nuestro contexto, diremos que:

a) Es un sistema *discreto* tal que:

—*Recibe un número finito de símbolos*. El conjunto de los símbolos que recibe se denomina *alfabeto de entrada*, E.

—*Emite un número finito de símbolos*. El conjunto de los símbolos que emite se denomina *alfabeto de salida*, S.

b) Los símbolos que recibe y emite un autómata evolucionan en el tiempo. La salida en un momento dado es función de los símbolos de entrada recibidos; es decir, existe una *memorización*. El *estado del autómata*, resumen de la evolución sufrida por éste a partir de un estado o situación inicial, permite afirmar que la salida en un momento dado es función del estado y de la entrada. Si Q representa el conjunto de estados en los que se puede encontrar el autómata, la función λ [$\lambda: Q \times E \rightarrow S$] define la salida en un instante dado. Por otra parte, dos estados consecutivos vienen definidos por una función δ [$\delta: Q \times E \rightarrow Q$] que permite obtener el nuevo estado en función del anterior y del último símbolo de entrada recibido.

Si el autómata es tal que el conjunto de los estados, Q, es finito, se dice que el autómata es de estados finito o, simplemente, que es un autómata finito.

En resumen, un *autómata finito* (AF) es una quíntupla $\langle E, S, Q, \lambda, \delta \rangle$ en la que:

1) $E = \{E_i\}$ es el *alfabeto de entrada* (conjunto finito).
2) $S = \{S_j\}$ es el *alfabeto de salida* (conjunto finito).
3) $Q = \{Q_k\}$ es el *conjunto finito de estados* (internos) del autómata.
4) δ es la *función de transición*, $\delta: Q \times E \rightarrow Q$.
5) λ es la *función de lectura o salida*, $\lambda: Q \times E \rightarrow S$.

Se denomina *estado inicial* a aquel en el que se encuentra el autómata cuando aún no ha recibido ningún símbolo de entrada.

Las funciones δ y λ definen el comportamiento del autómata; por lo tanto, la descripción completa de éste se obtiene al especificarlas exhaustivamente. Una primera

		E		E	
		a	b	a	b
Q	Q_a	Q_a	Q_{ab}	y	y
	Q_{ab}	Q_a	Q_{bb}	x	y
	Q_{bb}	Q_a	Q_{bb}	y	y
			δ		λ

NOTAS:
$Q = \{Q_a, Q_{ab}, Q_{bb}\}$.
Q_{ab} [Q_{bb}] es un estado en el que se recuerda que los dos últimos símbolos recibidos son la secuencia *a-b* [*b-b*].
Q_a es un estado en el que se recuerda que el último símbolo recibido es *a*.

Tabla 1.1 Tabla de estados (MEALY).

representación es la *tabular*. En la tabla 1.1 se presenta la definición de un autómata cuyos alfabetos de entrada y salida son $E = \{a, b\}$ y $S = \{x, y\}$. Su salida en un instante k es $S(k) = x$ si y sólo si se han recibido los símbolos de entrada $E(k) = a$, $E(k-1) = b$ y $E(k-2) = a$.

Como se puede comprobar, una *tabla de estados* está compuesta por dos subtablas; en la primera de ellas se define $\delta(Q_i, E_j)$, mientras que en la segunda se define $\lambda(Q_i, E_j)$.

La definición de autómata finito que hemos considerado hasta ahora se denomina de MEALY. Una definición alternativa es la de MOORE, en la que la función λ se simplifica al depender sólo de Q: $S_i = \lambda(Q_i)$; es decir, la salida depende exclusivamente del estado. La tabla 1.2 representa el sistema descrito por la tabla 1.1, pero utilizando un autómata de MOORE. Como a cada estado le corresponde una salida, la subtabla que define las salidas se reduce a un vector.

Un resultado clásico establece que se puede pasar de un autómata de MEALY a uno de MOORE, y viceversa, con lo que se demuestra que *la potencia de descripción de ambos modelos es idéntica*†. El paso de un autómata de MOORE a un autómata de

		E		
		a	b	
Q'	Q'_a	Q'_a	Q'_{ab}	y
	Q'_{ab}	Q'_{aba}	Q'_{bb}	y
	Q'_{aba}	Q'_a	Q'_{ab}	x
	Q'_{bb}	Q'_a	Q'_{bb}	y
			δ	λ

NOTAS:
$Q' = \{Q'_a, Q'_{ab}, Q'_{aba}, Q'_{bb}\}$.
Q'_{aba} [Q'_a] es un estado en el que se recuerda que los tres últimos símbolos recibidos es la secuencia *a-b-a* [diferente de *a-b-a*, pero el último símbolo es *a*; es decir: *a-a-a*, *b-a-a* o *b-b-a*].
Q'_{ab} [Q'_{bb}] es un estado en el que se recuerda que los dos últimos símbolos recibidos es la secuencia *a-b* [*b-b*].

Tabla 1.2 Tabla de estados (MOORE).

† Es decir, todo lo que se pueda describir con un modelo de AF se puede describir con el otro.

MEALY es inmediato, puesto que el segundo comprende al primero. La transformación inversa (MEALY → MOORE) puede llevarse a cabo de forma sistemática asociando a cada par (Q_i, S_j) del autómata de MEALY un estado $\bar{Q}_e$ del autómata de MOORE: $(Q_i, S_j) \equiv \bar{Q}_e$. Evidentemente, el mínimo número de estados de un modelo de MOORE nunca será inferior al de un autómata de MEALY.

La transformación del autómata de MEALY de la tabla 1.1 conduce a un autómata de MOORE con $\bar{Q} = \{\bar{Q}_{ay}, \bar{Q}_{aby}, \bar{Q}_{ax}, \bar{Q}_{bby}\}$.

EJERCICIO. Compruébese que los estados $\bar{Q}$ anteriores se corresponden con los Q' (tabla 1.2) de acuerdo con la siguiente lista: $\bar{Q}_{ay} \equiv Q'_a$, $\bar{Q}_{aby} \equiv Q'_{ab}$, $\bar{Q}_{ax} \equiv Q'_{aba}$ y $\bar{Q}_{bby} \equiv Q'_{bb}$.

1.3.2 Descripción de sistemas lógicos secuenciales asíncronos: la tabla de fases

La evolución del estado interno de un sistema lógico secuencial puede interpretarse y realizarse de dos formas distintas:

1) Evolución *autónoma*, en la que los cambios del estado interno se producen al presentarse un símbolo de entrada.
2) Evolución *controlada*, en la que los cambios del estado interno se producen sólo en determinados instantes. Éstos están definidos por las transiciones de subida, o de bajada, de una (o varias) señal(es) de entrada privilegiada(s).

Un sistema *síncrono* es un sistema cuya evolución está controlada por una señal de entrada denominada *reloj*. El reloj sincroniza la evolución del sistema. Entre dos señales de sincronización el estado del sistema es independiente de la evolución de las entradas.

Un sistema *asíncrono* (*y no pulsado*) es un sistema que está sometido a entradas de nivel y tal que su evolución es autónoma. Dada una determinada entrada, el sistema evoluciona hasta que el estado presente y el estado siguiente sean idénticos. En esas circunstancias se dice que el sistema se encuentra en un *estado estable* o *fase de evolución*. Los estados que no son estables se denominan *transitorios*.

Por el momento nos limitaremos a la consideración de los sistemas asíncronos no pulsados. Los sistemas síncronos se describen o modelan de forma similar. Las peculiaridades de su realización se estudiarán en los capítulos 6 al 9.

Para describir sistemas asíncronos se utiliza de forma clásica una representación tabular semejante a la tabla de estados, denominada *tabla de fases*. En una tabla de fases, una fase de evolución (o estado estable) se representa dentro de una circunferencia (figura 1.4*a*), lo que permite la eliminación de la primera columna. Como se puede observar en la figura 1.4*a*, los diferentes símbolos de entrada son los 2^2 vectores de entrada que se pueden formar con las *variables lógicas de entrada* $\{x_1, x_2\}$. Del mismo modo, se puede observar que las salidas son un subconjunto de los 2^2 vectores de salida que se pueden formar con las *variables lógicas de salida* $\{s_1, s_2\}$. Si existiesen q [r] variables de entrada [de salida], los símbolos de entrada [de salida] serían un subconjunto de los 2^q [2^r] vectores lógicos posibles. Una casilla (q_i, E_j) se rellenará con «—» cuando sea imposible que se presente E_j a partir del estado q_i. Se dice que el par (q_i, E_j) está *no especificado*.

El *diagrama de fases* (figura 1.4*b*) es un diagrama en el que cada estado estable de la máquina está representado por una circunferencia. Las transiciones se repre-

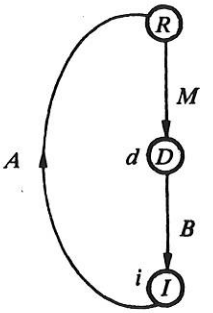


Figura 1.7. Ejemplo del carro que va-y-viene modelado con un grafo reducido.

concepto, denominado *receptividad*: un sistema es *receptivo* a un cierto evento o condición si éste es capaz de hacer evolucionar su estado. La receptividad a un determinado evento es función del estado del sistema.

Insistamos en que la receptividad aparece como un concepto nuevo entre las descripciones funcionales del tipo tabla de fases y grafo reducido, aunque se trate de un concepto clásico en la descripción algorítmica de sistemas.

Un concepto en cierto modo «dual» de la receptividad es el concepto de *sensibilidad*: un sistema lógico es *sensible* a cierta condición lógica si ésta es capaz de hacer que evolucionen sus variables de salida sin que evolucione su estado (interno).

Para ilustrar este concepto, supongamos que en el carro que «va-y-viene» de §1.3.3 se dispone de un conmutador H de inhibición de movimientos. La descripción del sistema es la misma, salvo que d e i (las acciones) estarán condicionadas por H (se representará d/H e i/H). En estas condiciones, si el automatismo se encuentra en los estados D o I, éste es sensible a H.

1.4.2 Obtención de la tabla de fases a partir del grafo reducido. Cantidad de información contenida en la descripción

El problema que abordamos aquí es la obtención de la tabla de fases definida por $\langle E, S, Q, \delta, \lambda \rangle$ a partir del grafo reducido. La comparación de esta tabla con la obtenida directamente a partir del enunciado del problema permitirá extraer algunas conclusiones de interés.

Para obtener la tabla de fases basta con observar que un estado de un GR es estable mientras su receptividad no se verifique. Dicho de otra forma, un estado es estable para el conjunto de todos los vectores de entrada que pertenecen al conjunto complementario de su receptividad. De este modo, el estado R (figura 1.7) es estable para todos los vectores de entrada que contengan $\bar{M} = \{\bar{M}\bar{A}\bar{B}, \bar{M}A\bar{B}, \bar{M}\bar{A}B, \bar{M}AB\}$. La transición $R \rightarrow D$ se realiza cuando $M = 1$. Operando de esta forma se llega sin dificultad a la tabla 1.5.

Al comparar la tabla 1.5 con la obtenida por reducción de la tabla primitiva (tabla 1.4) observamos que:

MAB 000	001	011	010	110	111	101	100	*d i*
Ⓡ	Ⓡ	Ⓡ	Ⓡ	*D*	*D*	*D*	*D*	0 0
Ⓓ	*I*	*I*	Ⓓ	Ⓓ	*I*	*I*	Ⓓ	1 0
Ⓘ	Ⓘ	*R*	*R*	*R*	*R*	Ⓘ	Ⓘ	0 1

Tabla 1.5 Tabla de fases obtenida a partir del GR de la figura 1.7. (Ejemplo del carro que va-y-viene.)

1) La tabla 1.5 está completamente especificada, a diferencia de la tabla 1.4 (obtenida a partir de la tabla primitiva). La tabla 1.4 recubre la tabla obtenida a partir del grafo reducido (tabla 1.5), lo cual se puede comprobar al considerar que $R = \{1\}$, $D = \{2, 3, 4, 5\}$ e $I = \{6, 7, 8, 9\}$.
2) A todo estado estable de la tabla 1.4 le corresponde uno de la tabla 1.5.
3) El estado R en la descripción con GR es estable y transitorio. En efecto, si $MA = 1$, R es un estado por el que pasa transitoriamente el GR. Esto se traduce en la tabla 1.5 en una transición de I a D que no es directa, cuando $MA = 1$. Obsérvese en la tabla 1.4 que dicha transición sí es directa.

En resumen, la tabla obtenida a partir del GR contiene un *mínimo de información*, suficiente para realizar una síntesis del sistema secuencial. Es fácil observar que el esfuerzo necesario para su obtención es mucho menor que el realizado para construir y reducir la tabla primitiva de fases. La economía de esfuerzo que se realiza se manifiesta en que la tabla obtenida a partir del GR especifica pares q_j-E_i que no tienen sentido físico (realmente son inespecificados). Así, por ejemplo, se comprueba que las columnas con $A = B = 1$ están innecesariamente especificadas.

Puesto que la única diferencia obtenida es la especificación de pares q_j-E_i no especificados físicamente, el comportamiento del modelo será correcto para las secuencias de operación previstas. Por otra parte, en estas condiciones interesa resaltar que las tablas obtenidas a partir del GR serán, en general, no reducibles. Este último hecho se comprende fácilmente, pues para obtener el autómata mínimo (mínimo número de estados) hace falta disponer de toda la información posible sobre el funcionamiento del sistema (condición general para efectuar una optimización).

Nota muy importante. La tabla de fases obtenida a partir de un GR que posea una selección (figura 1.8) puede tener casillas en las que coexistan dos o más estados. En el caso de la figura 1.8, los estados q_j y q_k coexistirán en las casillas que pertenecen a las columnas definidas por $x_j x_k = 1$. Para que el sistema esté descrito correctamente, es necesario que $q_i x_j x_k \equiv 0$, pues de lo contrario existiría una ambigüedad (conflicto).

1.4.3 Ejemplos de modelación

1.4.3.1 *Conjunto de carros que van-y-vienen sincronizados* (figura 1.9)

Supongamos que los carros C_1 y C_2 están inicialmente sobre los contactos A y C,

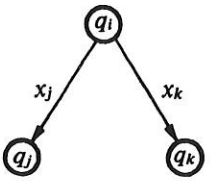


Figura 1.8. Selección en un GR.

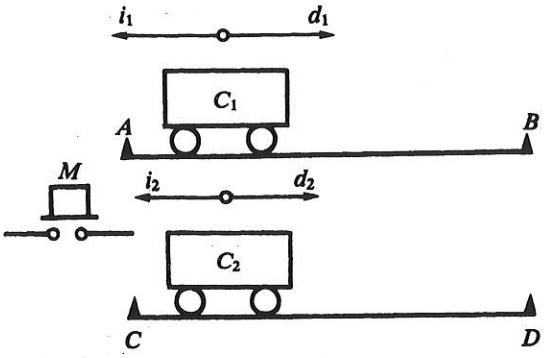


Figura 1.9. Dos carros que van-y-vienen sincronizados.

respectivamente. Al pulsar el botón M, los carros C_1 y C_2 se desplazan simultáneamente hacia la derecha. El inicio del regreso hacia la izquierda lo realizarán simultáneamente cuando ambos carros se encuentren en el extremo derecho (sobre los contactos B y D). Un posible GR que modela el problema propuesto es el representado en la figura 1.10.

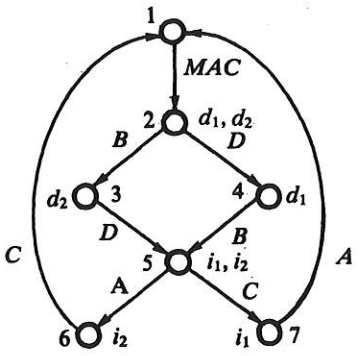


Figura 1.10. Modelación con GR (dos carros).

Si se considera ahora la existencia de tres carros, C_1, C_2, C_3, tendremos el GR de la figura 1.11. (C_3 evoluciona entre los contactos E y F.)

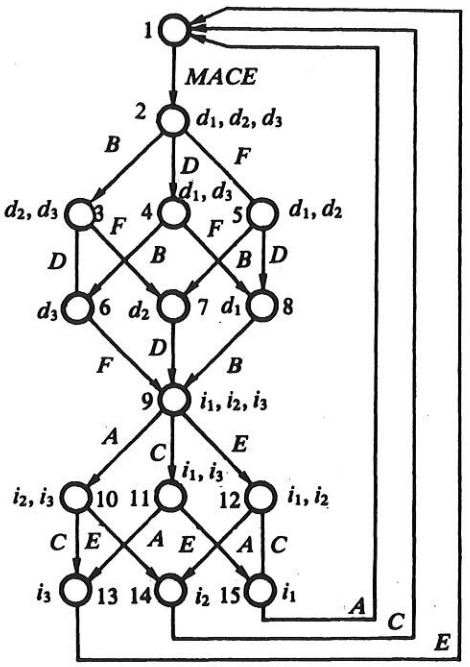


Figura 1.11. Modelación con GR (tres carros). El tercer carro evoluciona entre los contactos E y F.

Si se generaliza a N carros, el GR tendrá $2^{N+1} - 1$ estados, cantidad que, al crecer exponencialmente, hará materialmente imposible la utilización de un único GR. Por ejemplo, con $N = 5$ se tendrán 63 estados, mientras que con $N = 10$ se tendrán 2047.

Otro inconveniente a señalar, además del de la complejidad de la descripción resultante, es el hecho de tener que rehacer enteramente la descripción del sistema cuando se pasa de N a $N + 1$ carros. Por otra parte, si se deseara modificar ligeramente el enunciado, definiendo, por ejemplo, que el primer carro debiera quedar sobre el contacto B un tiempo mínimo, $t_{\min}$, la descripción de la figura 1.10 nos serviría de poco. Sería preciso rehacerla prácticamente en su totalidad (realícese la descripción como ejercicio).

En resumen, el GR no permite hacer fácilmente modificaciones *locales*; se dice que es un modelo *poco flexible*. Esto es fácil de comprender, puesto que el estado de un GR es un estado global del sistema y no un subestado parcial. Como veremos más adelante, con una red de Petri se puede representar el estado de un sistema como conjunto de estados parciales.

1.4.3.2 *Acciones simultáneas*

Supongamos que en cierto momento se han de desarrollar en un sistema informático dos acciones simultáneas A y B, seguidas de una acción C. Se desea modelar esta situación sabiendo que A y B están compuestas a su vez por tres subacciones $\{A_1, A_2, A_3\}$ y $\{B_1, B_2, B_3\}$, respectivamente, las cuales deben realizarse en secuencia.

Una solución es la representada en la figura 1.12.

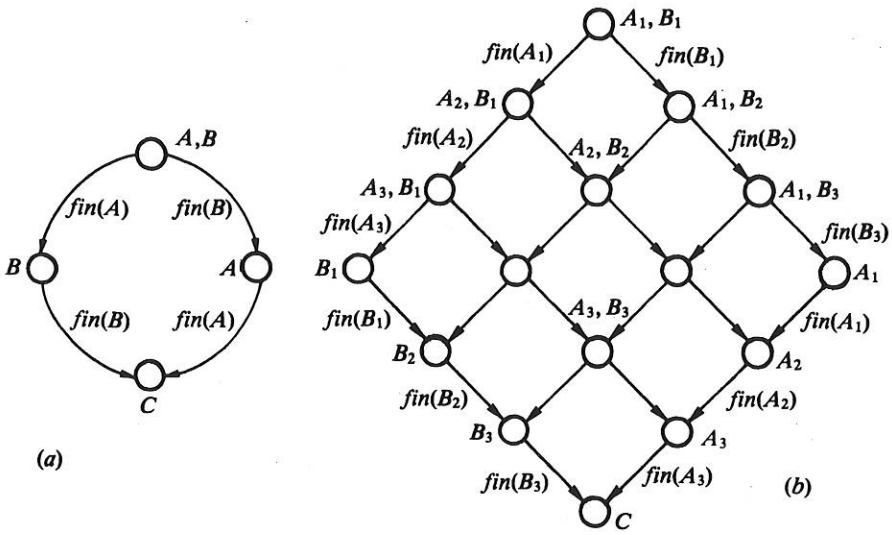


Figura 1.12. Acciones simultáneas. (Debido a su complejidad, no se han indicado en *b* todas las acciones ni todos los eventos.)

Se observa, pues, que el paso de la descripción a nivel de las macroacciones A y B al nivel de las acciones A_i-B_j se realiza destruyendo por completo la descripción original. Es decir, el GR no se adapta a una descripción progresiva (*por refinamientos sucesivos*) cuando el sistema posee evoluciones simultáneas.

1.4.4 Crítica al grafo reducido (GR)

1. Introduce los conceptos de receptividad y sensibilidad, los cuales permiten simplificar en gran medida la descripción de un sistema.
2. La información que aporta la modelación con un GR es mínima, pero suficiente para hacer la síntesis de un sistema.
3. Cuando existen evoluciones paralelas, simultáneas, la modelación de un GR no es eficiente porque:

   —conduce a descripciones complejas (gran número de estados);

—no permite modificaciones locales del comportamiento del sistema sin poner en tela de juicio toda la descripción realizada; es decir, no es flexible;
—no permite una descripción descendente del sistema (por refinamientos sucesivos, *top-down*).

Es precisamente para evitar o disminuir los inconvenientes (punto 3) por lo que se introducen las redes de Petri. Éstas aparecen como una generalización natural de los grafos reducidos que permite la consideración directa de las evoluciones simultáneas.

1.5 APLICACIÓN DE LAS REDES DE PETRI A LA MODELACIÓN FUNCIONAL DE SISTEMAS CONCURRENTES

Una *red de Petri* es una herramienta matemática que puede servir para modelar comportamientos de sistemas de naturaleza muy diferente. En particular, nos vamos a ceñir a su utilización como modelo de descripción del funcionamiento de un sistema discreto concurrente.

1.5.1 Definición del modelo de descripción

Una *red de Petri* (RdP) es un grafo orientado en el que intervienen dos clases de nudos, los *lugares* (representados por circunferencias) y las *transiciones* (representadas por segmentos rectilíneos), unidos alternativamente por *arcos*. Un arco une un lugar con una transición, o viceversa, pero nunca dos transiciones o dos lugares.

Un lugar puede contener un número positivo o nulo de *marcas*. Una marca se representa por un punto en el interior del círculo correspondiente al lugar. El conjunto de marcas asociadas en un instante dado a cada uno de los lugares constituye un *marcado* de la RdP.

Para la descripción funcional de sistemas concurrentes, a los lugares se les asocian *acciones* o *salidas* del sistema que se desea modelar. A las transiciones se les asocian los *eventos* (funciones lógicas de las variables de entrada del sistema) y acciones o salidas.

La figura 1.13 representa una RdP. Se sugiere al lector que dé rienda suelta a su imaginación para que «vea» la estructura de la RdP como un «tablero de damas especial» y las marcas como peones.

Nota. Los lugares estarán representados con la letra p, pues tanto en inglés como en francés se les denomina *place*.

Evolución del marcado (*reglas del «juego de damas especial»*):

—un lugar p es *lugar de entrada* de una transición t si existe un arco orientado de p hacia t;
—un lugar p es *lugar de salida* de una transición t si existe un arco orientado de t hacia p;
—una *transición está sensibilizada* si todos los lugares de entrada están marcados.

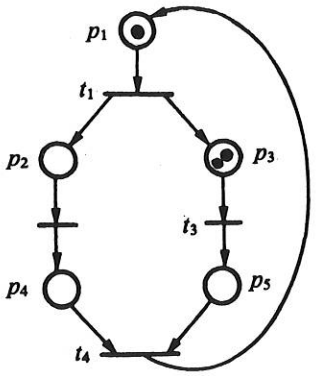


Figura 1.13. Ejemplo de RdP con 5 lugares y 4 transiciones. El lugar p_1 contiene una marca, mientras que el lugar p_3 contiene dos marcas. p_4 y p_5 son lugares de entrada de t_4, mientras que p_2 y p_3 son lugares de salida de t_1.

REGLA. Una transición sensibilizada es *disparada* o *franqueada* si el evento que le está asociado se verifica. El disparo de una transición consiste en quitar una marca a cada uno de los lugares de entrada y en añadir una marca a cada uno de los lugares de salida. □

La figura 1.14 ilustra los conceptos de sensibilización y disparo de una transición.

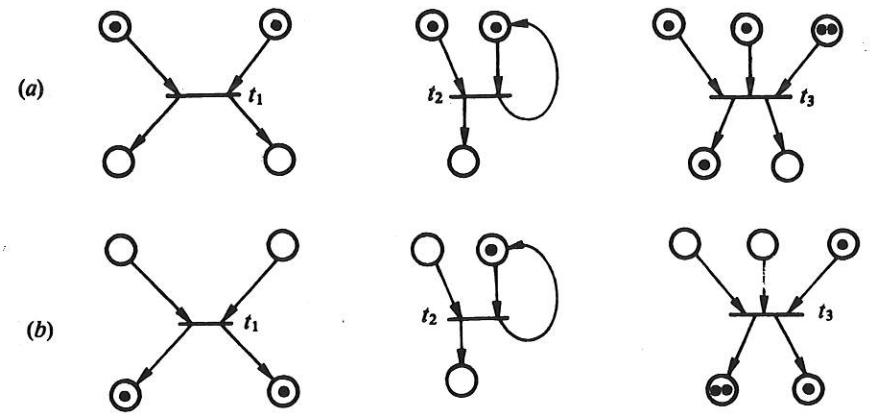


Figura 1.14. (*a*) Ejemplo de transiciones sensibilizadas.
(*b*) Marcados obtenidos después de sus disparos.

1.5.2 Ejemplos de modelación de sistemas concurrentes

En un grafo reducido un sistema se representa mediante el conjunto de estados totales en los que éste se puede encontrar. *En una RdP el estado está representado por*

el marcado. Como el marcado puede contener varios lugares marcados simultáneamente, el estado estará definido por un conjunto de *subestados* (estados locales o parciales) del sistema. A partir de esta consideración, es evidente que una RdP está mejor adaptada que los GR para la descripción de evoluciones simultáneas. Para ilustrar esta afirmación recurriremos al ejemplo de §1.4.3.1, conjunto de carros que van-y-vienen sincronizados. Las descripciones para los casos de dos y tres carros están representadas en la figura 1.15. En ambos casos el marcado representado, *marcado inicial*, corresponde a la situación de partida: todos los carros están sobre los contactos izquierdos de fin de carrera.

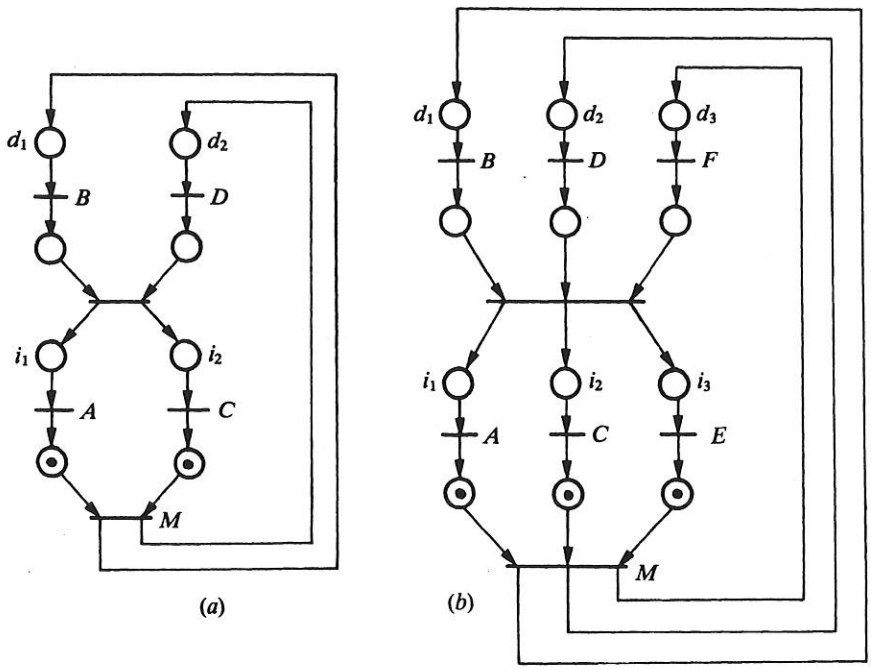
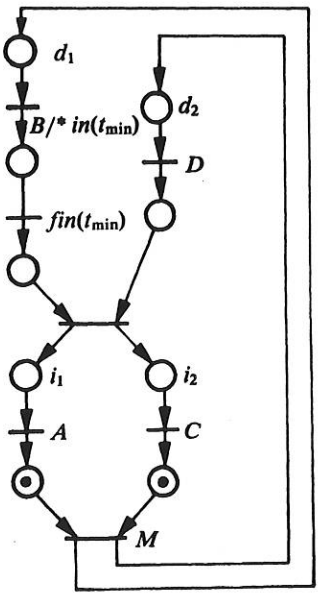


Figura 1.15. Ejemplos de carros que van-y-vienen sincronizados (véanse las figuras 1.10 y 1.11).

Los estados en la evolución de cada carro se encuentran representados por cuatro lugares. El lugar marcado inicialmente se corresponde con el estado R (figura 1.7). Los lugares etiquetados con i_j y d_j se corresponden con los estados I y D, respectivamente. Por último, el cuarto lugar representa el estado del carro en espera de que el (los) otro(s) carro(s) haya(n) alcanzado el extremo derecho de su recorrido.

El estado total del sistema viene definido por los subestados correspondientes a cada uno de los carros.

Se deja a la consideración del lector su comparación con las modelaciones con grafos reducidos y, si se encuentra de buen humor, se le invita a establecer una tabla de fases primitiva.



Nota. «$^*in(t_{min})$ es una acción de tipo impulsional que se ejecuta al dispararse la transición etiquetada con el evento B. Significa iniciar la cuenta de t_{min} unidades de tiempo.»

Figura 1.16. El primer carro debe permanecer al menos t_{min} unidades de tiempo sobre B.

La figura 1.16 ilustra una modificación del enunciado del ejemplo de los dos carros que van-y-vienen. Ésta consiste en que el primer carro debe permanecer sobre el contacto B durante un tiempo mínimo t_{min}.

De los ejemplos ilustrados en la figuras 1.15 y 1.16 se pueden deducir interesantes conclusiones. Podemos afirmar que la representación de evoluciones simultáneas se facilita enormemente con las RdP. En particular, esta facilidad permite:

1) Tener *descripciones menos complejas*. En el caso de los carros que van-y-vienen sincronizados se necesitan $4N$ lugares en vez de los $2^{N+1} - 1$ estados, lo cual, para $N = 10$, nos da 40 lugares frente a 2047 estados.
2) La consideración de *modificaciones locales*, como la introducida en el caso de la figura 1.16, se facilita enormemente (compárese con el modelo que utiliza un grafo reducido). Por todo ello diremos que la RdP es una herramienta de modelación *flexible*.

Para tratar de encontrar otra cualidad interesante en la modelación con RdP, reconsideremos el ejemplo de §1.4.3.2, acciones simultáneas. Su representación con RdP se ilustra en la figura 1.17. A partir de este caso se puede observar que las RdP permiten una *descripción progresiva* (por refinamientos sucesivos) de los sistemas con evoluciones simultáneas.

Observación. Comparando las figuras 1.12*a* y 1.17*a* se puede comprobar que, cuando el paralelismo es muy pequeño, la descripción con grafo reducido puede ser menos compleja que con RdP exhibiendo directamente el paralelismo.

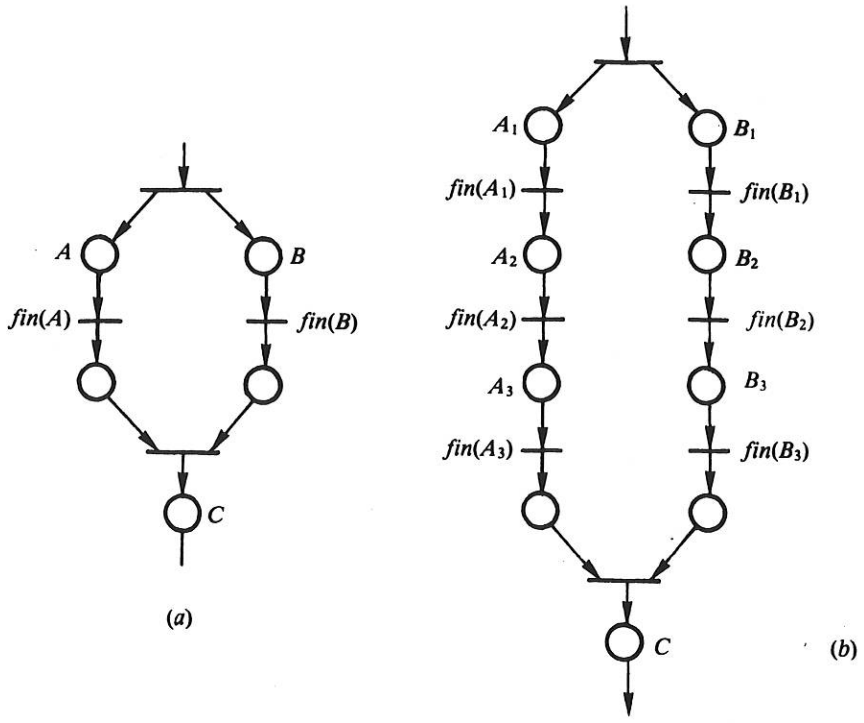


Figura 1.17. Acciones simultáneas.

Antes de abordar una crítica global sobre las aportaciones de las RdP a la modelación, insistiremos en algunas cuestiones de interés.

1.5.3 Configuraciones y propiedades básicas

1.5.3.1 *Configuraciones en una RdP*

Hemos incluido este apartado para presentar cierto vocabulario, que se encuentra con frecuencia en la literatura técnica y que concierne a las estructuras elementales de las RdP (figura 1.18).

1) Un lugar que tiene varios arcos de entrada y/o de salida se denomina *nudo O*.
   Un grafo reducido está formado a base de *nudos O*.
   Dos casos particulares de nudos *O* son:
   - la *selección* (un arco de entrada y varios de salida),
   - la *atribución* (varios arcos de entrada y uno de salida).
2) Una transición que tiene varios arcos de entrada y/o salida se denomina *nudo Y*. Se observará que estos nudos no existen en los grafos reducidos. Son los que permiten la creación y extinción de las evoluciones simultáneas.
   Dos casos particulares de nudos *Y* son:

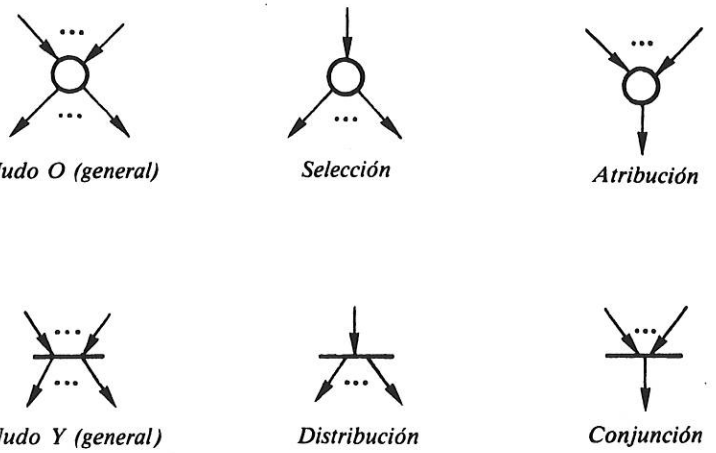


Figura 1.18. Nudos O y nudos Y.

- la *distribución* (un arco de entrada y varios de salida).
- la *conjunción* (varios arcos de entrada y uno de salida).

Una RdP que no contiene nudos *Y* se denomina *máquina* o *grafo de estados*. Obsérvese que esta definición no coincide (según la interpretación dada) con la de grafo reducido, puesto que en un grafo de estados podemos tener simultáneamente varias marcas. Para que un grafo de estados coincida con un grafo reducido hace falta que su marcado se limite a una sola marca.

1.5.3.2 *Propiedades básicas*

Una *transición* es *viva*, para un marcado M_0, si para todo marcado M que se pueda alcanzar a partir del marcado inicial M_0 existe un marcado M' sucesor de M a partir del cual se puede disparar esa transición. Una RdP es *viva*, para un marcado dado, si todas sus transiciones son vivas para ese marcado.

La RdP de la figura 1.19 no es viva. En efecto, el lector puede comprobar que

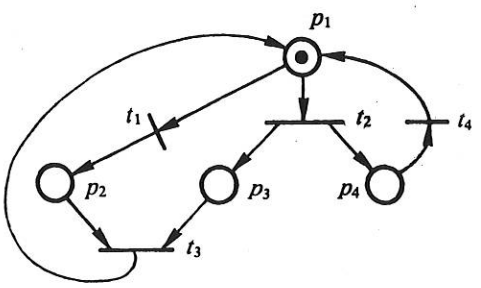


Figura 1.19. RdP no viva y no binaria.

si se aplica la secuencia de disparos t_2-t_4-t_1-t_3, el marcado evoluciona normalmente, pero si se aplica t_1 a partir del marcado inicial, el marcado de la RdP no permite el disparo de ninguna transición más (está bloqueada). Puesto que una *transición no viva* no puede ser disparada a partir de una cierta evolución en la RdP, se puede «sospechar» que el modelo del sistema objeto de estudio es incorrecto.

Para un marcado inicial dado, una RdP es *binaria* si cualquier marcado alcanzable es tal que ningún lugar posee más de una marca. En una RdP binaria todo lugar estará marcado con una marca o no estará marcado.

La RdP de la figura 1.19 es no binaria, puesto que si se repite la secuencia de disparos t_2-t_4, el número de marcas de p_3 crece continuamente. En este caso se dice que la RdP marcada es *no limitada*, puesto que el marcado de p_3 no tiene límite finito. Si la RdP que modela un sistema es no limitada, cabe igualmente «sospechar» que el modelo no es correcto. La limitación caracteriza la finitud del número de estados internos del sistema que se modela. Las redes de Petri pueden modelar *autómatas no finitos*.

Para un marcado inicial dado, se dice que una RdP es *conforme* si es binaria y viva. La RdP de la figura 1.20*a* es conforme. Debido a su marcado inicial, una misma RdP puede ser conforme, no binaria o no viva (figuras 1.20*b* y *c*).

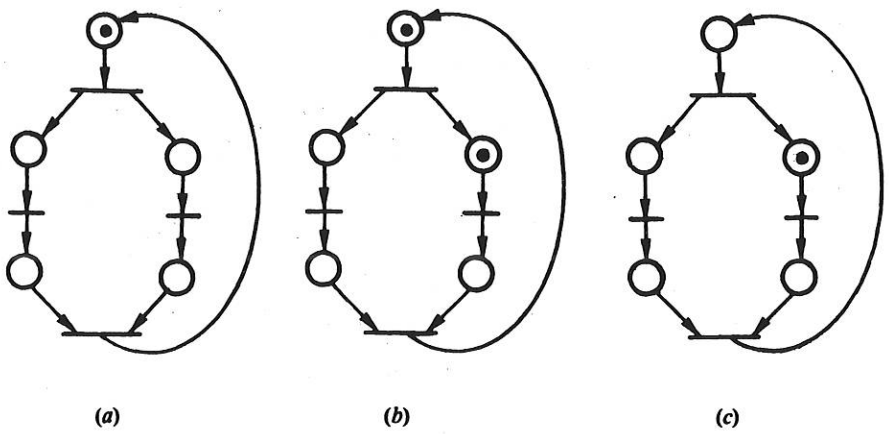


Figura 1.20. Debido a su marcado inicial, una misma RdP puede ser conforme, no binaria o no viva.

Dos o más transiciones simultáneamente sensibilizadas están en *conflicto* (figura 1.21) si descienden de un mismo lugar y éste no dispone de un número de marcas suficientes para dispararlas simultáneamente. Un conflicto se hace *efectivo* si los eventos asociados a las transiciones en conflicto se verifican simultáneamente.

Un conflicto efectivo corresponde a una ambigüedad en la descripción. Ninguna descripción correcta puede poseer conflictos efectivos.

En la modelación de sistemas lógicos concurrentes nos limitaremos fundamentalmente a la utilización de RdP que sean conformes (vivas y binarias) y sin conflicto.

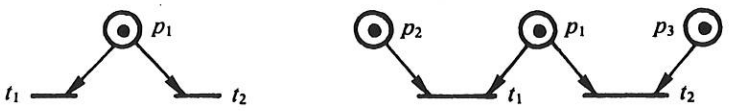


Figura 1.21. En ambos casos t_1 y t_2 están en conflicto.

La condición de que sean binarias no impone una restricción importante a la descripción de sistemas lógicos y, sin embargo, simplifica las realizaciones.

1.5.4 Obtención de un grafo reducido a partir de una RdP

La idea básica para transformar una RdP en un GR consiste en obtener *todos los marcados posibles y las transiciones entre éstos.*

Hemos visto que los marcados de una RdP representan los estados. Para obtener los diferentes estados se parte del marcado inicial de acuerdo con el siguiente procedimiento: A partir de este marcado (estado), se dispara cada transición sensibilizada y se calcula un nuevo marcado (nuevo estado). A partir de cada uno de estos nuevos marcados, se recomienza el proceso hasta que no se cree ningún nuevo marcado.

En otras palabras, el procedimiento se limita a obtener *todos los marcados posibles* y *las transiciones entre éstos.*

Se encomienda al lector como ejercicio la obtención de los GR de las figuras 1.10 y 1.11 a partir de las RdP de la figura 1.15 (supóngase que nunca se disparan simultáneamente dos transiciones).

1.6 COMENTARIOS EN TORNO A LA UTILIZACIÓN DE LAS RdP EN LA MODELACIÓN DE SISTEMAS CONCURRENTES

1. Se trata de una herramienta de modelación clara, fácil de utilizar y no ambigua. En particular, comprende los conceptos de *receptividad* y *sensibilidad*, lo cual permite obtener las descripciones con un *mínimo de información* suficiente para sintetizar los sistemas.

2. Facilita la representación de evoluciones simultáneas. Esto conduce a descripciones más simples que las obtenidas con grafos reducidos. Otras propiedades (consecuencia de su adaptación a la representación de evoluciones simultáneas) son su *flexibilidad* (facilidad de modificaciones locales) y su capacidad para permitir descripciones de forma descendente, es decir, por *refinamientos sucesivos.*

3. Permite una «primera aproximación» al problema de la *validación* del correcto funcionamiento de un sistema. Para ilustrar esta idea podemos reconsiderar el ejemplo (§1.4.3.1) del conjunto de carros que van-y-vienen. Este ejemplo ha sido descrito con un GR y con una RdP. Su descripción con un conjunto de N grafos reducidos (uno por carro) mutuamente dependientes en su evolución es también bastante sencilla y, por lo tanto, cabe pensar que se trate de una alternativa a la modelación con RdP. En el modelo construido con los N grafos reducidos, la sincronización entre los diferentes carros se llevará a cabo mediante la interpretación (semántica) asocia-

da a los grafos. Sin embargo, esta información sobre la sincronización se encuentra explícitamente en la estructura y marcado del modelo realizado con RdP. Por consiguiente, *la estructura y el marcado de la RdP contienen cierta información sobre el funcionamiento del sistema*, la cual es muy superior a la que contiene la estructura y el estado inicial del conjunto de grafos.

El valor de esta información es el aumentar la legibilidad de las descripciones, así como el posibilitar «cierto nivel» de verificación formal de las mismas (estudio de su buen comportamiento). Permite también abordar una mayor protección del equipo realizado frente a algunos problemas de la explotación (detección de averías, etc.). Es decir, las RdP pueden aportar una notable ayuda a la concepción y utilización.

Desde un punto de vista práctico, la estrategia que se utiliza con las RdP consiste en comprobar si la descripción verifica una serie de *propiedades de buen funcionamiento* (por ejemplo, ausencia de bloqueos y conflictos, exclusión mutua, etc.). Estas propiedades son, en gran parte, independientes de las funciones específicas que realiza el sistema, así como del modelo utilizado para describirlo. Un *análisis de validez* de una descripción consiste en la comprobación de un conjunto de estas propiedades sobre el modelo del sistema que se estudia (véanse los capítulos 4 y 5). Este análisis se puede realizar gracias a la información que, independientemente de la interpretación asociada a la RdP, contiene su estructura y su marcado inicial.

Así, por ejemplo, parece poco probable que la RdP de la figura 1.19 corresponda a una descripción correcta de un sistema, puesto que no es viva y, además, el número de marcas que puede contener p_3 es potencialmente ilimitado (basta con iterar la secuencia de disparos t_2-t_4).

Para concluir lo relativo a la validación de una descripción, resaltaremos que otra de las ventajas de las RdP consiste en que hacen posible una fácil traducción de las propiedades de buen funcionamiento del sistema en propiedades específicas de las RdP. De este modo, las propiedades de vivacidad (específicas de las RdP) caracterizan, de forma extremadamente simple, la presencia o ausencia de bloqueos totales (*deadlock*) o parciales del sistema.

4. La *simplificación* que se puede realizar de una descripción es muy limitada. Se reduce a la eliminación de redundancias estructurales en la RdP. Se han desarrollado otros métodos de simplificación basados en un aporte suplementario de información. El capítulo 3 lo dedicaremos a este interesante problema. En cualquier caso, conviene dejar bien sentado que, más que buscar descripciones minimizadas, lo que se persigue es simplificar un poco la descripción original, de forma que su comprensión no se dificulte extraordinariamente.

5. Otra propiedad interesante radica en que la RdP, interpretada de acuerdo con los convenios expuestos, constituye *una herramienta de modelación independiente de cualquier tecnología* (electrónica, fluídica, etc.)

A modo de información final, remitimos al lector impaciente por conocer algunas de las posibilidades de modelado inherentes a las RdP a los apartados 2.3.2 y 2.4, donde encontrará diversos ejemplos clásicos. La primera parte del capítulo 2 se dedica a la presentación formalizada de conceptos básicos utilizados en las redes de Petri.

EJERCICIOS

1.1 Determínese si las RdP de la figura E.1.1 son vivas y/o binarias.

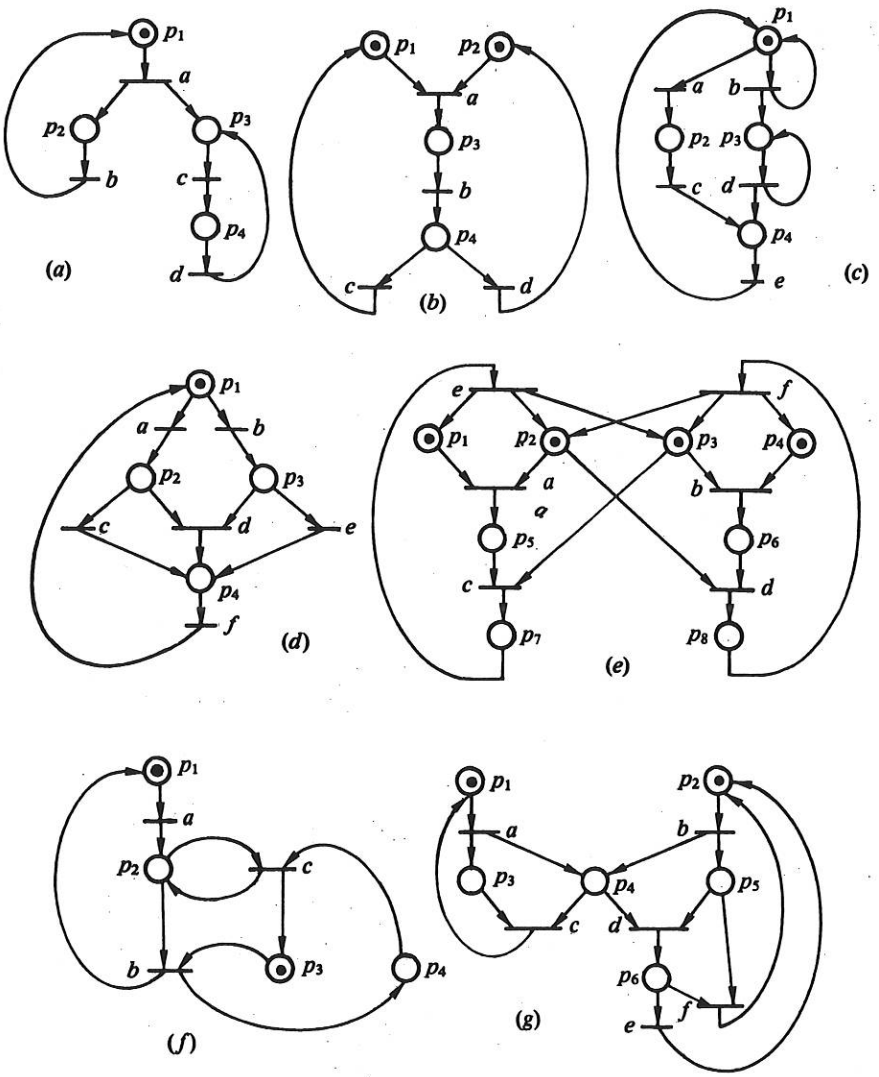


Figura E.1.1. Diversas redes de Petri.

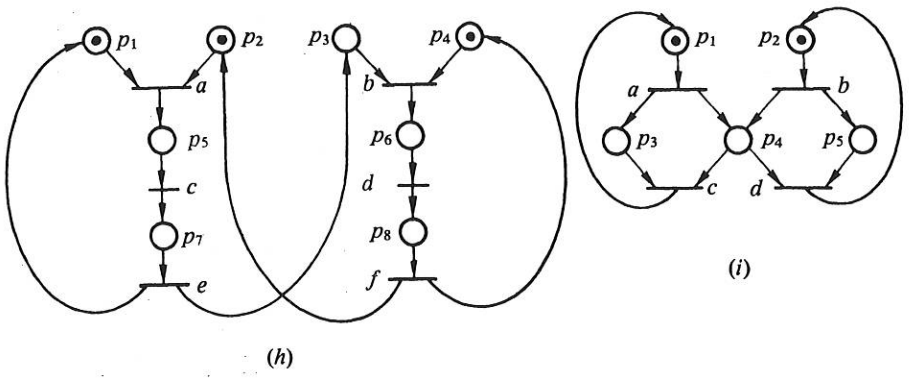


Figura E.1.1. (cont.) Diversas redes de Petri.

1.2 Sean dos carros C_1 y C_2 (véase figura 1.9): C_1 se desplaza entre A y B, y C_2 entre C y D; d_i controla el movimiento del carro C_i hacia la derecha, i_i controla el movimiento hacia la izquierda.

Los dos carros inician la marcha cuando se pulsa M, partiendo hacia la derecha si están sobre A y C. C_1 vuelve hacia la izquierda en cuanto llega a B, mientras que el regreso de C_2 no debe comenzar hasta que C_2 llegue a D y C_1 haya vuelto a A.

a) Obténgase una descripción funcional con RdP que muestre la evolución simultánea de los dos carros.

b) Obténgase una descripción secuencial del sistema (GR).

c) Compárense ambas descripciones. Obténgase el GR equivalente a la descripción realizada en (*a*). ¿Se atreve el lector a escribir una tabla de fases del sistema?

1.3 Se desea realizar el ciclo X, Y, Z, X indicado en la figura E.1.2. El sistema consta de cinco entradas, que son la puesta en marcha del ciclo (pulsador P) y los cuatro contactos fin de carrera: arriba (contacto A), derecha (contacto D), abajo (contacto B) e izquierda (contacto I). Se controlan cuatro variables de salida, $\{a, d, b, i\}$, que corresponden a los movimientos hacia arriba, derecha, abajo e izquierda, respectivamente.
Resuélvanse las mismas cuestiones del ejercicio 1.2.

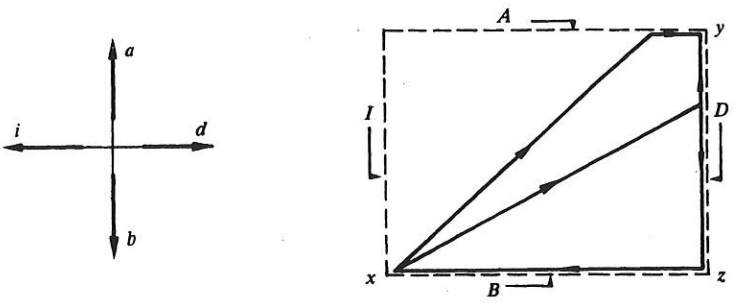


Figura E.1.2. Ciclo que se desea realizar.

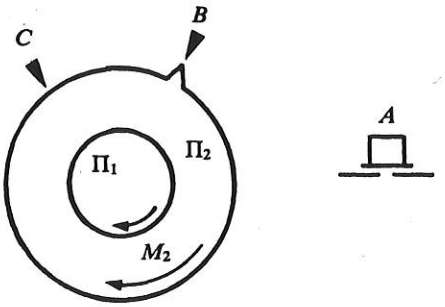


Figura E.1.3. Sistema que se desea automatizar.

1.4 *a*) ¿Se puede obtener un grafo reducido a partir de la RdP de la figura E.1.1(*a*)? ¿Por qué?

b) Obténgase los grafos reducidos que se puedan determinar a partir de las diferentes RdP de la figura E.1.1.

1.5 Modélese con una RdP el funcionamiento del sistema de la figura E.1.3, que consta de dos plataformas, π_1 y π_2, y un botón pulsador, A. La posición de π_2 es detectada por medio de dos fines de carrera B y C. La rotación de las plataformas está asegurada por dos motores: M_1 para π_1 y M_2 para π_2. El funcionamiento deseado es el siguiente:

El sistema parte del reposo (plataforma π_2 en posición B; motores parados).

—De la acción de A resulta que:

- π_2 se desplaza de B a C, donde se detiene;
- π_1 queda en rotación mientras se mantiene pulsado A.
  Al soltar A, π_1 se detiene y cualquier nueva acción sobre A no tiene influencia sobre π_1.

—Después de estar π_2 en la posición C y π_1 en reposo:

- π_2 se desplaza desde C hasta B;
- simultáneamente, la plataforma π_1 se pone en rotación cada vez que se pulsa A y se detiene cuando éste se suelta;
- el sistema vuelve al estado inicial (motores parados) cuando la plataforma π_2 pasa por la posición B, estando A en reposo.

La plataforma π_2 puede, por consiguiente, dar tantos giros como se desee, manipulando el pulsador A.

2

Redes de Petri: formalización y aplicación a la modelación funcional de sistemas concurrentes

2.1 INTRODUCCIÓN

La creciente utilización de las redes de Petri se justifica dado que permiten *modelar* y *analizar* el subsistema de control de sistemas discretos que exhiben evoluciones concurrentes. Dada la adecuación de las redes de Petri a la clase de sistemas definidos, éstas se utilizan en diversos campos.

Para emplear una red de Petri en la modelación de una clase de aplicaciones, se ha de proceder a dotarla de una *interpretación*. Es decir, asociarle una significación «física» a las condiciones de evolución de la red, así como definir las acciones generadas por dicha evolución. De forma esquemática, se puede decir que la red define la estructura de la descripción del sistema, y que su interpretación le asocia una semántica.

Los principales campos de aplicación en los que se han utilizado las redes de Petri, provistas de diversas interpretaciones, son:

1) los sistemas legales [MELD 71];
2) los sistemas operativos y descripción de *software* en general [BAER 73] [DENN 73] [AGER 79] (véase el anexo 1);
3) la descripción de *hardware* de computadores y sistemas discretos de control con evoluciones concurrentes (con tratamientos numéricos, considerando una parte operativa) [PATI 73] [VALE 76] [MOAL 76];
4) los automatismos lógicos [DACL 76] [SILV 82*a*];
5) los lenguajes formales [HACK 75] [CRES 77] [PETE 81*a*];
6) la evaluación de prestaciones (*performances*) [SIFA 77] [RAMC 73] [RAMA 80] [FLOR 81].

Las redes de Petri no constituyen el único modelo de descripción de sistemas discretos capaz de modelar las evoluciones paralelas. Al lector interesado se le remite a las referencias [BAER 73] [PETE 81*a*] [VALE 76], donde podrá encontrar información complementaria al respecto.

El funcionamiento de una red sin interpretación alguna se define como *autónomo*. Una red temporizada (red cuya evolución es función del tiempo) y/o interpreta-

da es una red *no autónoma*. En efecto, en este último caso la evolución de la red depende también del tiempo y/o de la interpretación asociada.

En este capítulo procederemos, en primer lugar, a presentar las redes de Petri como herramienta matemática. Después introduciremos una interpretación que, asociada a la herramienta matemática, permitirá construir modelos de sistemas lógicos concurrentes. En el anexo 1 dotaremos a las redes de Petri de otra interpretación, adecuada a la modelación de la estructura de control de programas concurrentes. El apartado 2.4 ilustra la utilización de redes de Petri interpretadas para la descripción de una serie de sistemas de gran importancia en la automática y la informática. Por último, en §2.5 definiremos una serie de subclases y extensiones de las redes de Petri. Las extensiones pretenden facilitar la modelación de los sistemas o ampliar la capacidad descriptiva, mientras que las subclases permitirán, posteriormente, estudios más simples sobre el comportamiento de los modelos que se construyan.

2.2 REDES DE PETRI AUTÓNOMAS

En este apartado, además de presentar la terminología básica, introducimos la ecuación de estado de una red.

Para simplificar la presentación de la terminología básica partiremos del concepto de red de Petri generalizada. Las redes de Petri, tal y como fueron definidas en el capítulo 1, son redes de Petri *ordinarias* e *interpretadas*.

2.2.1 Terminología básica

2.2.1.1 *Conceptos estructurales*

Definición 2.1. Una *red de Petri generalizada* es una cuádrupla $\boldsymbol{R} = \langle P, T, \alpha, \beta \rangle$ tal que

P es un conjunto finito y no vacío de *lugares*
T es un conjunto finito y no vacío de *transiciones*
$P \cap T = \emptyset$; es decir, lugares y transiciones son conjuntos disjuntos
$\alpha: P \times T \to \mathbb{N}$ es la *función de incidencia previa*
$\beta: T \times P \to \mathbb{N}$ es la *función de incidencia posterior*. □

Representación gráfica. Una RdP se representa gráficamente por un grafo bipartido orientado. Los lugares se representan por circunferencias y las transiciones por barras. Existe un *arco* que va del lugar p_i a la transición t_j sii $\alpha(p_i, t_j) \neq 0$. Análogamente, existe un arco que va de la transición t_k al lugar p_i sii $\beta(t_k, p_i) \neq 0$. Cada arco se etiqueta con un entero natural, $\alpha(p, t)$ o $\beta(t, p)$, que se denomina *peso del arco*. Por convenio, un arco no etiquetado posee un peso unitario. Para facilitar la legibilidad, todo arco cuyo peso sea superior a la unidad se dibuja normalmente con un trazo grueso, o con dos o más trazos paralelos.

Representación matricial. Una red se representa matricialmente por medio de dos matrices. Sea $|P| = n$ (número de lugares de la red) y sea $|T| = m$ (número de transi-

ciones de la red). Se denomina *matriz de incidencia previa* la matriz

$$C^- = [c_{ij}^-]_{n \times m},$$

en la que $c_{ij}^- = \alpha(p_i, t_j)$. Se denomina *matriz de incidencia posterior* la matriz

$$C^+ = [c_{ij}^+]_{n \times m},$$

en la que $c_{ij}^+ = \beta(t_j, p_i)$.

Es decir, en las matrices de incidencia los lugares numeran las filas (i) y las transiciones las columnas (j), y cada elemento (i,j) expresa la incidencia que el lugar i tiene sobre la transición j.

Definición 2.2. Una *red es ordinaria* si sus funciones de incidencia sólo pueden tomar los valores 0 y 1:

$$\begin{cases} \alpha(p, t) \in \{0, 1\} \\ \beta(t, p) \in \{0, 1\}. \end{cases} \qquad \square$$

Definición 2.3. Una *red es pura* si ninguna transición contiene un lugar que sea simultáneamente de entrada y de salida:

$$\forall t_j \in T \quad \forall p_i \in P \quad \alpha(p_i, t_j)\beta(t_j, p_i) = 0. \qquad \square$$

La representación matricial de una red pura se simplifica definiendo una única matriz, C, denominada *matriz de incidencia*:

$$C = C^+ - C^- \Rightarrow c_{ij} = \begin{cases} \beta(t_j, p_i) \text{ si es no nula} \\ -\alpha(p_i, t_j) \text{ si es no nula} \\ 0 \text{ en cualquier otro caso} \end{cases}$$

de esta forma, en la matriz C un elemento positivo indica incidencia posterior y uno negativo señala incidencia previa. Un elemento nulo en C indica que la transición y lugar correspondientes no están conectados directamente a través de un arco.

En la figura 2.1 se muestra una RdP ordinaria y pura.

Definición 2.4. Sea una red $R = \langle P, T, \alpha, \beta \rangle$, $t \in T$ y $p \in P$. Se definen los siguientes conjuntos:

a) Conjunto de *lugares de entrada* de t:

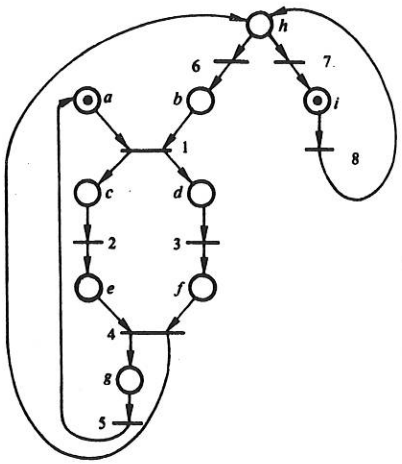
$$^\cdot t = \{ p \in P \mid \alpha(p, t) > 0 \}$$

b) Conjunto de *lugares de salida* de t:

$$t^\cdot = \{ p \in P \mid \beta(t, p) > 0 \}$$

c) Conjunto de *transiciones de entrada* de p:

$$^\cdot p = \{ t \in T \mid \beta(t, p) > 0 \}$$



$$\mathbf{C} = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 & t_8 \end{matrix} \\ \begin{bmatrix} -1 & 0 & 0 & 0 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & +1 & 0 & 0 \\ +1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ +1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & +1 & 0 & -1 & -1 & +1 \\ 0 & 0 & 0 & 0 & 0 & 0 & +1 & -1 \end{bmatrix} \end{array} \begin{matrix} p_a \\ p_b \\ p_c \\ p_d \\ p_e \\ p_f \\ p_g \\ p_h \\ p_i \end{matrix}$$

Figura 2.1. Ejemplo de red de Petri ordinaria y pura. Matriz de incidencia.

d) Conjunto de *transiciones de salida* de p:

$$p^{\cdot} = \{t \in T \mid \alpha(p, t) > 0\}. \qquad \square$$

Si se considera la red de la figura 2.1, se puede escribir, por ejemplo,

$$\begin{aligned} {}^{\cdot}t_1 &= \{p_a, p_b\} & {}^{\cdot}p_h &= \{t_4, t_8\} \\ t_1^{\cdot} &= \{p_c, p_d\} & p_h^{\cdot} &= \{t_6, t_7\}. \end{aligned}$$

De acuerdo con esta notación, una red es pura sii $\forall t \in T \; {}^{\cdot}t \cap t^{\cdot} = \emptyset$.

Definición 2.5. Una *subred* de $\boldsymbol{R} = \langle P, T, \alpha, \beta \rangle$ es una red $\bar{\boldsymbol{R}} = \langle \bar{P}, \bar{T}, \bar{\alpha}, \bar{\beta} \rangle$ tal que $\bar{P} \subseteq P$ y $\bar{T} \subseteq T$. $\bar{\alpha}$ y $\bar{\beta}$ son restricciones de α y β sobre $\bar{P} \times \bar{T}$. $\square$

2.2.1.2 *Conceptos dinámico-estructurales*

Definición 2.6. El *marcado* M de una red $\boldsymbol{R}$ es una aplicación de P en $\mathbb{N}$, o sea, la asignación de un número entero no negativo (número de marcas) a cada lugar. $\square$

En el grafo asociado a $\boldsymbol{R}$, el marcado M se representa por una distribución, en los lugares, de objetos denominados *marcas*. Una marca se representa gráficamente por un punto en el interior de la circunferencia que define el lugar que la contiene.

Si $|P| = n$, entonces un marcado se representa, en forma matricial, por un vector de n elementos: $M(p_i)$. (*Vector de marcado.*)

Definición 2.7. Una *red de Petri marcada* es el par $\langle \boldsymbol{R}, M_0 \rangle$, en el que $\boldsymbol{R}$ es una red de Petri y M_0 es un *marcado inicial.* □

La evolución del marcado le confiere a la RdP marcada un comportamiento dinámico que permite modelar evoluciones simultáneas de sistemas discretos.

Definición 2.8. Una transición $t \in T$ está *sensibilizada* por el marcado M sii cada uno de sus lugares de entrada posee al menos $\alpha(p, t)$ marcas. Es decir, se exige que $\forall p \in {}^{\cdot}t \quad M(p) \geqslant \alpha(p, t)$. □

Regla de evolución del marcado (definición 2.9). *Disparar una transición sensibilizada* t es la operación que consiste en eliminar $\alpha(p, t)$ marcas a cada lugar $p \in {}^{\cdot}t$ y añadir $\beta(t, p)$ marcas a cada lugar $p \in t^{\cdot}$. $M_i \xrightarrow{t} M_j$ significa que t está sensibilizada por M_i y que al disparar t a partir de M_i se alcanza M_j.

Es decir, al disparar t se obtiene:

$$M_j(p) = M_i(p) + \beta(t, p) - \alpha(p, t) \quad \forall p \in P. \qquad \square$$

La figura 2.2 representa el disparo de la transición t.

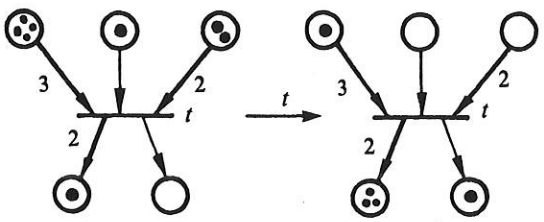


Figura 2.2. Disparo de la transición t: evolución del marcado.

Definición 2.10. Una *secuencia de disparos aplicable a partir del marcado* M_0 se representa por una secuencia de transiciones tal que el disparo de cada transición conduce a un marcado que sensibiliza la transición siguiente de la secuencia.

Si $M_0 \xrightarrow{t_i} M_1 \xrightarrow{t_j} M_2 \rightarrow \cdots \xrightarrow{t_r} M_q$, se dirá que la secuencia $\sigma = t_i t_j \ldots t_r$ es aplicable a partir de M_0. La anterior evolución del marcado se puede condensar escribiendo $M_0 \xrightarrow{\sigma} M_q$.

El *conjunto de secuencias disparables* a partir de M_0 es un *lenguaje*:

$$\boldsymbol{L}(\boldsymbol{R}, M_0) = \{\sigma \mid M_0 \xrightarrow{\sigma} M\}. \qquad \square$$

En la RdP que se observa en la figura 2.1 es aplicable, por ejemplo, la secuencia $\sigma_1 = t_8\, t_6\, t_1\, t_3\, t_2\, t_4\, t_6$. El marcado que se obtiene al disparar la anterior secuencia de 7 disparos es M_7:

$$M_0^T = (1\,0\,0\,0\,0\,0\,0\,0\,1)$$
$$ \quad a\ b\ c\ d\ e\ f\ g\ h\ i$$
$$M_7^T = (0\,1\,0\,0\,0\,0\,1\,0\,0).$$

Definición 2.11. Se llama *vector característico asociado a una secuencia de disparos* σ al vector $\bar{\sigma} \in \mathbb{N}^m(|T| = m)$, cuya i-ésima componente es el número de ocurrencias del disparo de t_i en la secuencia σ.

El *conjunto de vectores característicos* de las secuencias disparables a partir de M_0 es

$$\bar{L}(R, M_0) = \{\bar{\sigma} \mid M_0 \xrightarrow{\sigma} M\}. \qquad \square$$

El vector característico asociado a la secuencia σ_1 es $\bar{\sigma}_1 = (1\,1\,1\,1\,0\,2\,0\,1)^T$.

Definición 2.12. Un marcado M *es alcanzable a partir de un marcado inicial* M_0 sii existe una secuencia de disparos aplicable a partir de M_0 que transforma M_0 en M: $M_0 \xrightarrow{\sigma} M$. El conjunto de los marcados alcanzables a partir de M_0 es:

$$M(R, M_0) = \{M \mid (\exists \sigma \in L(R, M_0)) \wedge (M_0 \xrightarrow{\sigma} M)\}. \qquad \square$$

De acuerdo con lo establecido, por ejemplo, en la RdP de la figura 2.1 el marcado M_7 es alcanzable a partir de M_0.

2.2.2 Ecuación de estado de una red de Petri

Sea C la matriz de incidencia de una red pura y marcada. A partir de la definición de C y de la regla de evolución del marcado se puede escribir $M_k = M_{k-1} + C \cdot U_k$, donde:

—M_k es el marcado obtenido al realizar el k-ésimo disparo;

—U_k es un vector cuyas componentes son nulas salvo la i-ésima si t_i es la transición disparada en k-ésimo lugar: $U_k(i) = 1$, $U_k(j) = 0, \forall j \neq i$.

Razonando por recurrencia se tiene:

$$\begin{aligned} M_k &= M_{k-1} + C \cdot U_k = \\ &= M_{k-2} + C \cdot (U_{k-1} + U_k) = \\ &= M_{k-3} + C \cdot (U_{k-2} + U_{k-1} + U_k) = \ldots = \\ &= M_0 + C \cdot \sum_{j=1}^{k} U_j = M_0 + C \cdot \bar{\sigma} \end{aligned}$$

donde $M_0 \xrightarrow{\sigma} M_k$.

En este punto es importante observar que $\bar{\sigma}$ no puede ser cualquier vector no negativo, pues podría no existir una σ aplicable a partir de M_0. Así, por ejemplo, $\bar{\sigma} = (1\,1\,1\,1\,1\,1\,1\,0)^T$ no tiene sentido en la RdP de la figura 2.1, pues no existe σ aplicable a partir de M_0 tal que su vector característico sea el presentado anteriormente.

EJERCICIO. Calcúlese el marcado que se obtendría si existiese una secuencia disparable σ cuyo vector característico fuese el anterior.

La justificación de la anterior restricción es inmediata dado que un marcado no puede contener componentes negativas y, por lo tanto, los sucesivos vectores U_k deben verificar la relación $M_{k-1} + C \cdot U_k \geqslant 0$; es decir, $\forall M_k \in M(R, M_0)$ debe verificarse que sus componentes sean enteros no negativos.

En conclusión:

$$(1)\quad M_k = M_{k-1} + C \cdot U_k$$

$$(2)\quad (M_0 \xrightarrow{\sigma} M_k) \Rightarrow M_k = M_0 + C \cdot \bar{\sigma}.$$

La primera ecuación tiene la forma de la *ecuación de estado de un sistema dinámico lineal e invariante discretizado en el tiempo*, por lo que a veces se le denomina *ecuación de estado asociada a la red de Petri.* La segunda ecuación integra la evolución desde M_0 hasta M_k.

La importancia de estas ecuaciones es muy grande, pues permiten plantear diversos análisis sobre el comportamiento de la RdP utilizando técnicas derivadas del álgebra lineal. Ahora bien, estos análisis poseen intrínsecamente dos limitaciones:

1) no todo $\bar{\sigma} \in \mathbb{N}^m$ es admisible (como se ha dicho);
2) el vector $\bar{\sigma}$ no define unívocamente la secuencia σ, por lo que se ha perdido una información fundamental para estudiar la actividad de la red.

La teoría de lenguajes formales permite abordar análisis más potentes sobre la evolución de las redes de Petri, pero su utilización práctica es bastante más complicada.

2.3 LAS REDES DE PETRI COMO MODELO DE DESCRIPCIÓN DE SISTEMAS LÓGICOS CONCURRENTES

En el apartado anterior hemos venido considerando a las RdP como una estructura matemática dotada de una propiedad dinámica (el marcado). Para que una RdP pueda representar un sistema, hace falta asociarle una *interpretación*. Interpretar una RdP es establecer un convenio por el cual se define:

—Un significado físico a las condiciones necesarias para el disparo de una transición. Las reglas de evolución son modificadas ligeramente por la interpretación, la cual pasa a ser también función de la evolución del proceso que se desea controlar.

—Las acciones generadas por la evolución del marcado.

El *estado interno* del sistema se representa siempre por el marcado de la RdP.

En lo que concierne a la interpretación de una RdP, se debe señalar que no existe una única interpretación posible para describir una clase de aplicaciones. De este modo, para la descripción de sistemas lógicos se encuentran dos grandes grupos, según que las acciones generadas se asocien a los lugares o a las transiciones.

La interpretación que presentamos aquí comprende ambos tipos.

Sea un sistema con:

—entradas $X = \{x_1, \ldots, x_e\}$
—salidas a nivel $Y = \{y_1, \ldots, y_s\}$
—salidas impulsionales $Z = \{z_1, \ldots, z_q\}$.

Definición 2.13. Una *condición externa*, C_i, es un subconjunto de estados de las variables de entrada. Ésta podrá ser representada siempre por una función combinatoria de las variables de entrada. □

Por ejemplo, sea $X = \{a, b, c\}$ el conjunto de variables de entrada. Una condición externa será $\{a\bar{b}c, a\bar{b}\bar{c}, abc, ab\bar{c}, \bar{a}bc, \bar{a}b\bar{c}\}$ y se podrá representar mediante la función lógica:

$$a + b \equiv a\bar{b}c + a\bar{b}\bar{c} + abc + ab\bar{c} + \bar{a}bc + \bar{a}b\bar{c}.$$

Definición 2.14. Un *evento* o *acontecimiento*, E_i, se define como:

—el cambio de estado lógico de una condición externa,
—la unión de un conjunto de eventos. □

El conjunto de eventos se designa por $E = \{E_i\}$.

2.3.1 Interpretación asociada a las RdP

Por convenio asociaremos:

—*eventos*, *condiciones externas* y *salidas impulsionales* a las transiciones,
—*salidas a nivel* a los lugares.

La activación de una salida puede estar ligada a condiciones externas. En tal caso hablaremos de *salidas condicionales*.

Definición 2.15. Una RdP *interpretada* (RdPI) se define por:

—una RdP marcada $\langle R, M_0 \rangle$;
—una aplicación de T en E que asocia a cada transición un evento, E_i;
—una aplicación de T en C que asocia a cada transición una condición externa, C_i;
—una aplicación de T en Z que a las transiciones les asocia salidas impulsionales;
—una aplicación de $P \times C$ en Y que a los lugares les asocia salidas a nivel, eventualmente condicionadas. □

Reglas de evolución del marcado de una RdPI

REGLA 1: El disparo de una transición t_i sensibilizada sólo se realiza si se verifica C_iE_i. En estas condiciones se dice que t_i es *receptiva* a la condición y al evento (C_iE_i).

REGLA 2: Cuando una transición es disparada, todas las salidas impusionales asociadas a t_i son generadas.

REGLA 3: Para un marcado dado, las salidas a nivel asociadas a los lugares marcados son generadas si se verifican las condiciones externas que eventualmente les pueden estar asociadas. Si Y_j es una salida asociada a un lugar p_i condicionada por C_{ij}, se dirá que la salida es *sensible* a C_{ij}, cuando p_i está marcado.

Observación muy importante. La representación de sistemas *síncronos* puede realizarse de acuerdo con la definición 2.15. En efecto, basta considerar que todos los pares evento-condición se reducen a la intersección entre: (1) los niveles o flancos activos de la variable de entrada privilegiada que es el reloj y (2) la condición externa que se le asocie al disparo de la transición.

2.3.2 Ejemplo de modelación (recurso compartido por dos usuarios)

Dos carros A y B transportan cierto material desde los puntos de carga C_A y C_B, respectivamente, hasta el punto de descarga D (figura 2.3*a*).

Los diferentes movimientos, hacia la izquierda o hacia la derecha, son controlados mediante las acciones i_A, i_B, d_A, d_B.

Si A está en C_A y el pulsador M_A está oprimido, comienza un ciclo C_A-D-C_A con las siguientes características:

—espera eventual en E_A hasta que la zona común a los dos carros esté libre, con el fin de evitar colisiones;

—espera obligatoria en D de $T_A = 100$ s de duración.

El carro B tiene un funcionamiento similar (pulsador M_B, ciclo C_B-D-C_B y espera en D de $T_B = 50$ s) pero, en caso de demanda simultánea de la vía común (recurso compartido), el carro B es prioritario (prioridad fija).

El recorrido E_A-D (respec. E_B-D) se establece gracias al posicionamiento de un cambio de agujas controlado por la acción G (respec. $\bar{G}$). En lo sucesivo admitiremos que E_A (respec. E_B) proporciona un «1» lógico si el eje delantero de A (respec. B) está en la zona E_A-D (respec. E_B-D).

COMENTARIOS

1) La figura 2.3*b* presenta una posible modelación.
2) Es importante considerar la construcción que modela el acceso al recurso. El lugar p_3 representa el recurso en «estado de reposo» (no utilizado). Los lugares p_6 a p_{11} representan la ocupación de recurso común ($\{p_6, p_8, p_{10}\}$ ocupación por el carro A y $\{p_7, p_9, p_{11}\}$ ocupación por el carro B).
3) Los lugares $\{3, 4, 5\}$ plantean un conflicto (el acceso al recurso cuando la demanda es simultánea). La realización debe evitar la ambigüedad que se presentaría si tecnológicamente se pudiera tener $M(p_3)M(p_4)M(p_5)\bar{E}_B E_A E_B = 1$.
4) La transición $p_6 \rightarrow p_8$ es receptiva a D. A ella se le ha asociado la acción impulsional «preseleccionar un temporizador con 100 unidades de tiempo», *temp*(100).
5) La acción d_A asociada al lugar p_4 está condicionada por $\bar{E}_A$. Es decir, si p_4 está marcado, el automatismo será sensible a $\bar{E}_A$.
6) Si se desea que el ciclo de movimientos que corresponde al carro A comience cuando se «pulsa M_A» (y no si M_A está pulsado), la RdP es entonces la misma; bastará con que la transición etiquetada $M_A C_A$ sea etiquetada $M_A \uparrow C_A$ (flanco de subida de M_A y condición externa C_A).

2.3.3 Transformaciones sobre condiciones externas y eventos que no alteran una descripción

La idea que vamos a exponer es muy sencilla: para la descripción de un sistema da-

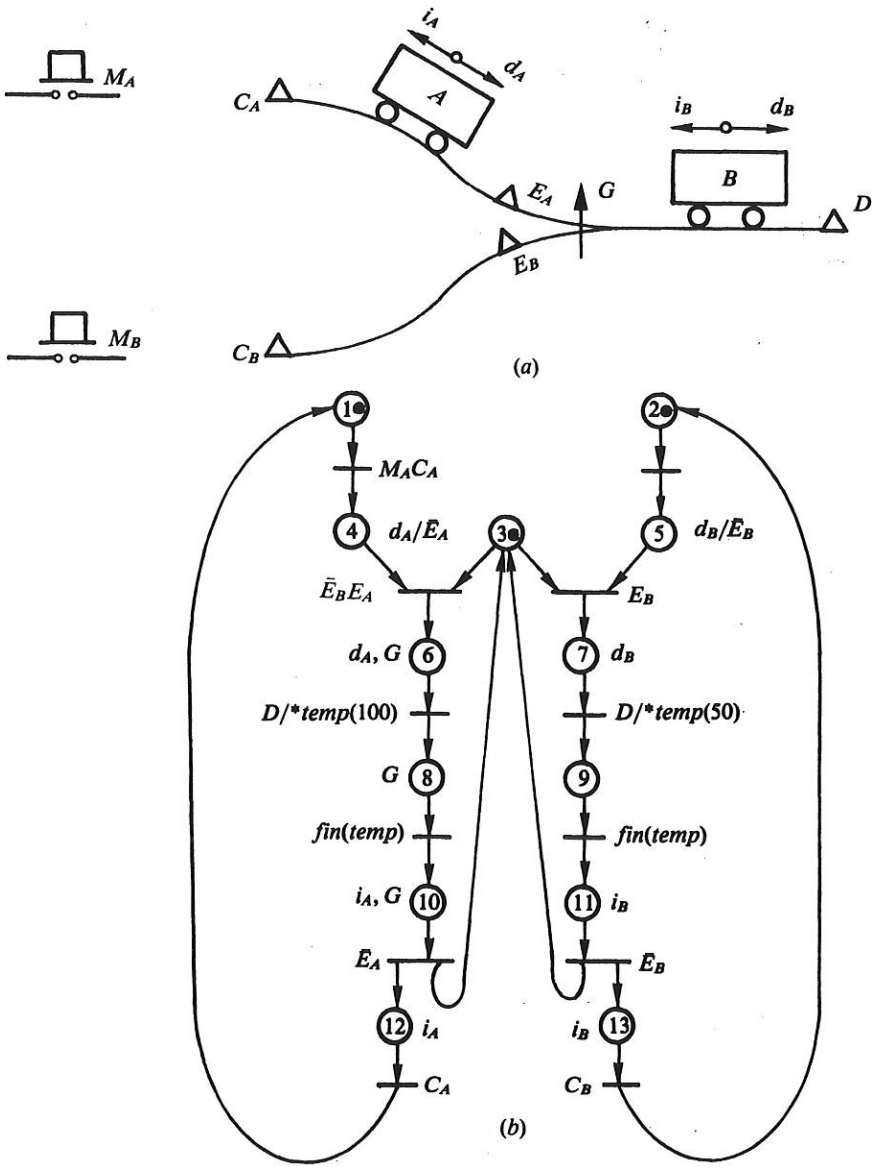


Figura 2.3. Esquema de la instalación y descripción (recurso compartido por dos usuarios).
(*a*) Esquema de la instalación.
(*b*) Descripción funcional con red de Petri.

do, las condiciones externas y los eventos pueden ser modificados en ciertos casos sin que el comportamiento de la descripción se altere. Esta propiedad elemental se puede utilizar, pongamos por caso, para simplificar los eventos o para transformarlos en condiciones externas, lo que eventualmente facilitará la realización física.

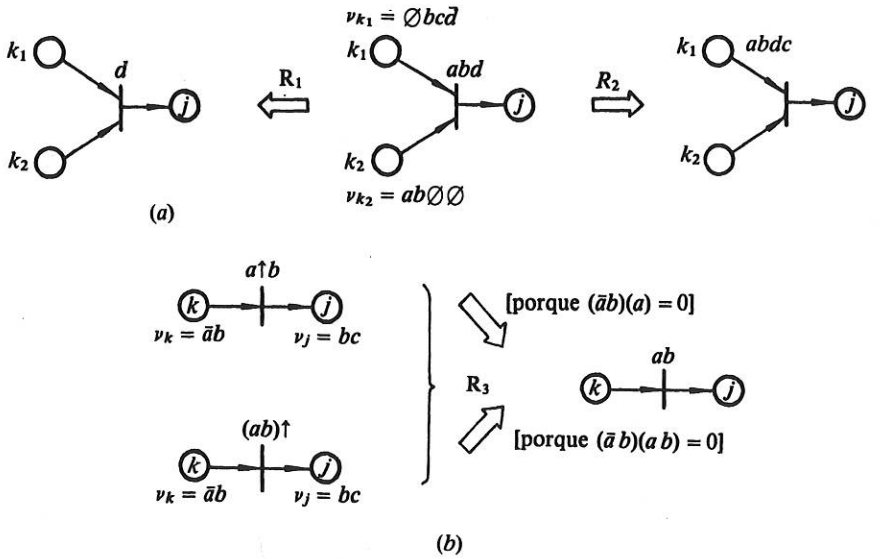


Figura 2.4. Ilustración de las reglas 1, 2 y 3.
(*a*) Transformaciones de condiciones externas.
(*b*) Transformación de un evento en condiciones externas.

Definición 2.16. Se llama *condición envolvente de un lugar* p_i, representada por γ_i, a la intersección lógica de las variables de entrada que tienen un valor definido cuando el lugar p_i está marcado. El estado (complementado o no) de las variables de entrada será tal que, si p_i está marcado, $\gamma_i = 1$. □

Si en p_i, por ejemplo, las entradas de un sistema toman los valores $x_1 = 1$, $x_2 = \emptyset\dagger$ y $x_3 = 0$, entonces $\gamma_i = x_1\bar{x}_3$.

Sea C_jE_j el par condición externa-evento asociado a una de las transiciones de entrada a p_i, que supondremos escrito como una unión: $C_jE_j = \sum_a C_j^a E_j^a$. Ejemplo: $C_jE_j = x_1x_2 + x_1x_3\uparrow$.

En estas condiciones es fácil observar que $C_j^a E_j^a$ puede ser transformado sin modificar funcionalmente el comportamiento que corresponda a la descripción. Para ello se pueden tener en cuenta las siguientes reglas:

REGLA 1. Una variable que pertenezca al menos a una de las condiciones envolventes de los lugares de entrada de la j-ésima transición, $p_k \in {}^{\cdot}t_j$, puede ser eliminada de C_j^a si se encuentra definida en el mismo estado, complementada o no, en C_j^a (figura 2.4*a*).

REGLA 2. Una variable que pertenezca al menos a una de las condiciones envolventes de los lugares $p_k \in {}^{\cdot}t_j$ puede ser añadida a C_j^a si no está definida en C_j^a (figura 2.4*a*).

† $\emptyset$ en álgebra de BOOLE es 0 o 1.

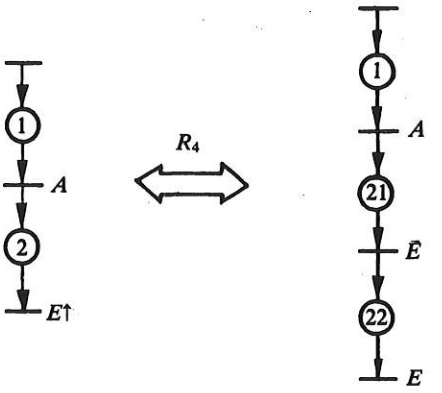


Figura 2.5. Transformación de $E\uparrow$ en E cuando no sea aplicable la regla 3.

REGLA 3. El evento asociado a t_j (E_j^q) puede ser transformado en una condición externa si ésta posee una intersección nula con la condición envolvente de alguno de los lugares de entrada a t_j, $p_k \in {}^{\cdot}t_j$ (figura 2.4*b*).

REGLA 4. Si no se puede aplicar la regla 3, siempre se puede transformar el evento $E\uparrow$ en E realizando una transformación del tipo de la expuesta en la figura 2.5.

Supongamos que inicialmente los carros están sobre C_A y C_B. A modo de ejemplo, se puede destacar que la condición externa $M_A C_A$ se puede transformar en $M_A C_A \bar{E}_A$ (regla 2) puesto que, cuando p_1 está marcado, el carro no puede estar en la zona E_A-D. De la misma forma, puesto que cuando p_1 está marcado se tiene C_A, la condición $M_A C_A$ se puede transformar en M_A (regla 1).

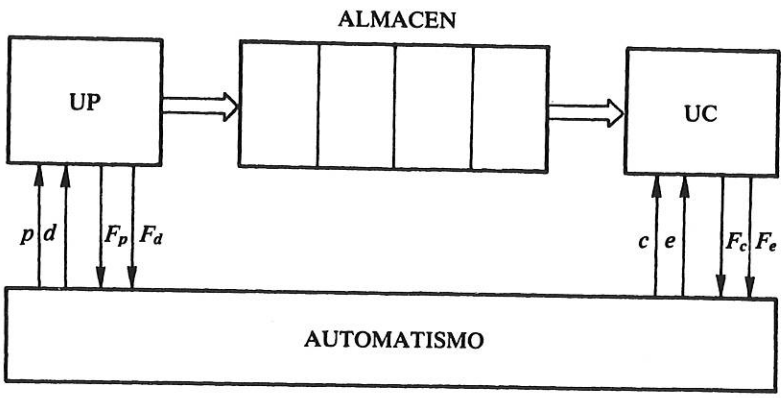
2.4 EJEMPLOS TÍPICOS DE MODELACIÓN CON REDES DE PETRI

Como ya anticipábamos en la introducción del capítulo, incluimos este apartado para ilustrar algunas construcciones clásicas de la modelación†. Pretendemos con ello que el lector se habitúe a «ver» las evoluciones simultáneas en los sistemas. Obsérvese que en modo alguno perseguimos la presentación de descripciones «minimizadas». En el capítulo 3 se estudian técnicas que, eventualmente, permitirán simplificar las descripciones.

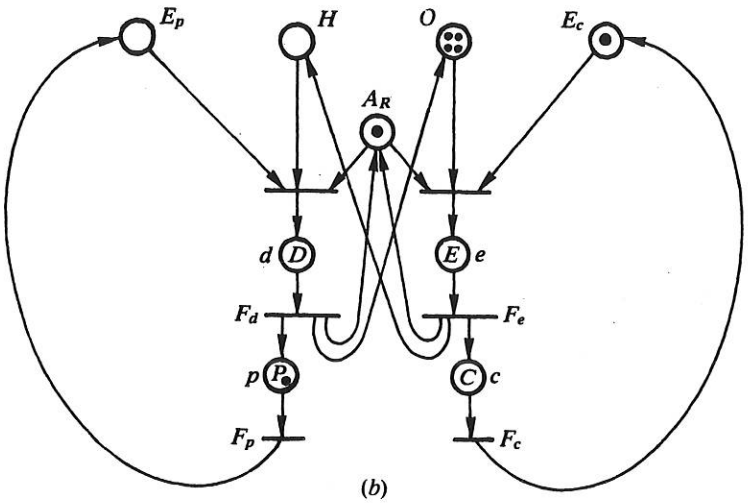
2.4.1 Relación productor-consumidor con exclusión mutua

Enunciado. Una unidad de producción (UP) produce cierta clase de objetos que deposita en un almacén. El almacén tiene una capacidad máxima de $N = 4$ objetos.

† En el apartado 2.5.2 presentaremos ejemplos complementarios en los que se utilizan diversas generalizaciones de las redes de Petri.



(*a*)



(*b*)

Figura 2.6. Esquema de la instalación y descripción del sistema productor-consumidor cuando se accede al almacén en exclusión mutua.
(*a*) Esquema de la instalación (almacén de capacidad 4).
(*b*) Descripción (el acceso al almacén se efectua en exclusión mutua entre el productor y el consumidor).

Una unidad de consumo (UC) retira objetos del almacén para su posterior consumo. Pretendemos describir un sistema que coordine la producción y el consumo, es decir, ordene al productor (consumidor) cuándo debe producir (consumir) y cuándo debe depositar (extraer) un objeto.

Las señales intercambiadas entre el sistema de control y el proceso son las siguientes (figura 2.6*a*):

p = orden de producir un objeto
d = orden de depositar un objeto en el almacén
c = orden de consumir un objeto
e = orden de extraer un objeto del almacén
F_p, F_d, F_c y F_e representan el fin de producción, depósito, consumo y extracción de un objeto.

Se supone que al almacén no se puede acceder simultáneamente en depósito y extracción (imagínese una memoria tampón: depósitos ≡ escritura, extracción ≡ ≡ lectura). Es decir, las operaciones de depósito y extracción deben estar en exclusión mutua.

COMENTARIOS. En la descripción de la figura 2.6*b*, los lugares que se relacionan a continuación tienen las siguientes funciones:

1) A_R representa al almacén en reposo (no se desarrolla operación de depósito ni de extracción).
2) O y H representa el número de objetos depositados y de huecos libres en el almacén, respectivamente.
3) E_p representa las esperas del productor hasta que en el almacén exista un hueco donde depositar la unidad producida y no esté accediendo al almacén el consumidor (esté en reposo). El papel de E_c es similar (esperas del consumidor para que haya objetos en el almacén y no esté accediendo a éste el productor).

A partir de esta información se puede observar que para cualquier marcado alcanzable se tendrá:

a) $M(A_R) + M(D) + M(E) = 1$, lo que significa que el almacén, o está en reposo, o está accediendo a él el productor o el consumidor, pero no ambos a la vez.
b) $M(O) + M(E) + M(H) + M(D) = 4$, lo que significa que el número de objetos depositados, más los que se están retirando, más los huecos libres y más los objetos que se están depositando suman 4, la capacidad del almacén.

EJERCICIOS

1) Determínese el significado del marcado inicial que se ha asignado a la red de la figura 2.6*b*.
2) Determínese el significado de las relaciones siguientes:
   a) $M(E_p) + M(D) + M(P) = 1$.
   b) $M(E_c) + M(E) + M(C) = 1$.
3) Simplifíquese la descripción de la figura 2.6*b*, suponiendo que no existe la restricción sobre exclusión mutua entre las operaciones de depósito y extracción de objetos.

2.4.2 Secuencias alternadas

Enunciado. Se desea realizar el dispositivo de control de una cadena de tratamiento de superficies por inmersión. Las piezas son cogidas por unas pinzas. Para no complicar inútilmente la descripción, no vamos a considerar el sistema de control de las pinzas (cogida de piezas).

El proceso consta de cinco zonas de trabajo (figura 2.7*a*):

—carga
—desengrasado (1 cuba)
—tratamiento (2 cubas)
—lavado (1 cuba)
—descarga

Puesto que el tiempo de tratamiento es superior al tiempo de inmersión en las otras cubas, se han destinado dos cubas idénticas al tratamiento, en lugar de una.

Las pinzas son desplazadas por un carro. El dispositivo de control debe ser tal que cuando el carro va cargado, ordene su desplazamiento horizontalmente sólo en la posición alta (sensor P_A), y cuando va descargado lo haga únicamente en la posición baja (P_B). Se definen los contactos C, D_G, T_1, T_2, L y D_C (figura 2.7*a*) de forma que estarán activados cuando se encuentren entre las ruedas del carro transportador.

Suponiendo que se parte inicialmente de:

(1) todas las cubas llenas, menos la de desengrasado,
(2) el carro sobre el puesto de carga y la pinza abajo (con carga),

un ciclo completo de tratamiento consta de dos subciclos con los siguientes movimientos con transporte de piezas:

(1.1) Carga → desengrasado
(1.2) Lavado → descarga
(1.3) Tratamiento 1 → lavado
(1.4) Desengrasado → tratamiento 1

(2.1) Carga → desengrasado
(2.2) Lavado → descarga
(2.3) Tratamiento 2 → lavado
(2.4) Desengrasado → tratamiento 2

El inicio de cada subciclo requiere la autorización del operador, el cual la otorga mediante un pulsador M.

COMENTARIOS

1) Los dos subciclos son idénticos: en el primero se trabaja con la cuba de tratamiento número 1, mientras que en el segundo se trabaja con la número 2.

2) La descripción más inmediata consiste en repetir dos veces un subciclo: una vez operando con el contacto T_1 y la otra operando con el contacto T_2.

Aunque una descripción como ésta pueda ser correcta, será poco interesante, pues parte de la doble representación de los movimientos de transporte de piezas.

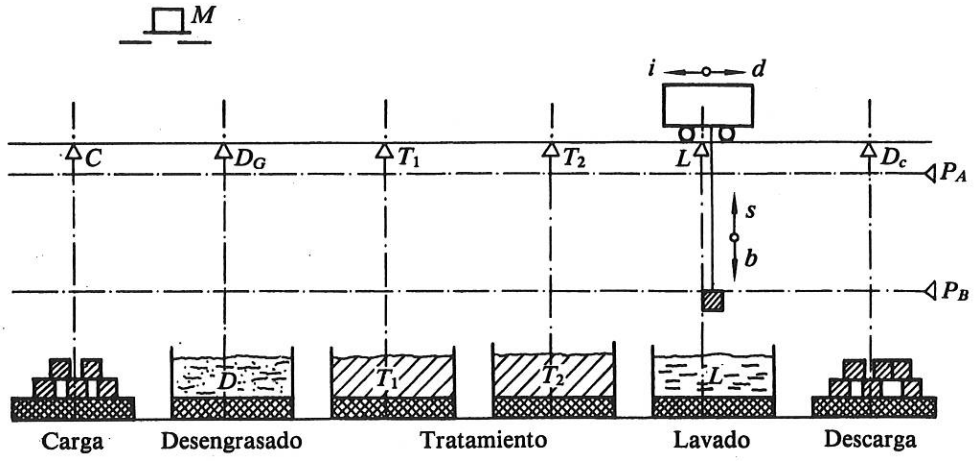
3) La figura 2.7*b* presenta una descripción basada en la definición de un único subciclo. En éste las construcciones realizadas con los lugares 91-92 y 161-162 permiten recordar la cuba objetivo de un desplazamiento (paridad del subciclo).

El marcado de p_0 indica el estado de espera inicial (éste se caracteriza por estar el carro sobre C y la pinza en posición baja, P_B). El marcado de p_{91} (p_{92}) y de p_{161} (p_{162}) indica que se espera alcanzar el contacto T_1 (T_2).

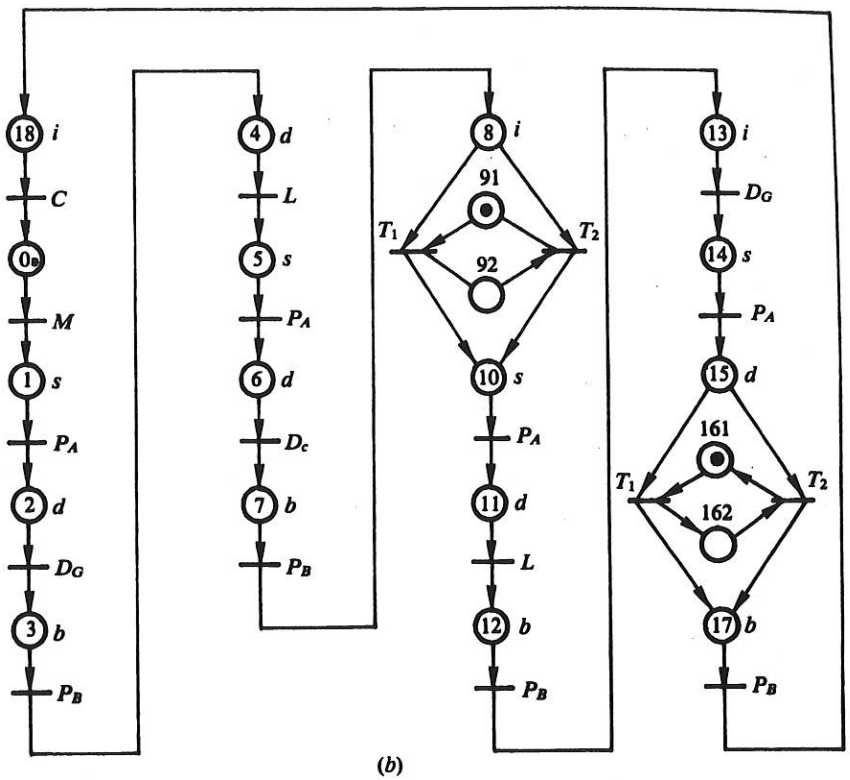
Como es fácil comprobar, este ejemplo muestra que la descripción «paralela» de una evolución secuencial puede conducir a modelos reducidos.

4) A partir de lo expresado anteriormente, se puede constatar que:

- $\{p_1, p_2, p_3\}$ representa el movimiento con carga: carga → desengrasado.
- $\{p_4\}$ representa el movimiento sin carga: desengrasado → lavado.



(*a*)



(*b*)

Figura 2.7. Sistema de tratamiento de superficies.
(*a*) Esquema de la instalación.
(*b*) Modelo construido.

- $\{p_5, p_6, p_7\}$ representa el movimiento con carga: lavado → descarga.
- $\{p_8, p_{91}, p_{92}\}$ representa el movimiento sin carga:

$$\text{descarga} \rightarrow \begin{cases} \text{tratamiento 1} \\ \text{o} \qquad \text{(según la paridad del subciclo)} \\ \text{tratamiento 2} \end{cases}$$

- $\{p_{10}, p_{11}, p_{12}\}$ representa el movimiento con carga:

$$\left.\begin{array}{c} \text{tratamiento 1} \\ \text{o} \\ \text{tratamiento 2} \end{array}\right\} \rightarrow \text{lavado}$$

- $\{p_{13}\}$ representa el movimiento sin carga: lavado → desengrasado
- $\{p_{14}, p_{15}, p_{161}, p_{162}, p_{17}\}$ representa el movimiento con carga:

$$\text{desengrasado} \rightarrow \begin{cases} \text{tratamiento 1} \\ \text{o} \\ \text{tratamiento 2} \end{cases}$$

- $\{p_{18}\}$ representa el movimiento sin carga:

$$\left.\begin{array}{c} \text{tratamiento 1} \\ \text{o} \\ \text{tratamiento 2} \end{array}\right\} \rightarrow \text{carga}$$

2.4.3 Reutilización de secuencias de funcionamiento (subprogramas)

Todo conjunto de secuencias representado por una subRdP puede ser utilizado a partir de diversas situaciones. La figura 2.8 propone un esquema básico para su modelación. Los lugares p_i y p_j recuerdan *el punto de regreso* de la secuencia. Si, como es de esperar, no se ordena la ejecución de una subsecuencia desde dos puntos a la vez, $M(p_i)M(p_j) = 0$, el regreso a la secuencia de llamada no será nunca ambiguo.

Dos implicaciones básicas de esta posibilidad de modelación son:

1) Modularidad en la descripción.
2) Descripciones de tamaño más reducido y realizaciones más económicas. (En este punto se debe señalar que se pueden realizar directamente incluso con técnicas cableadas, capítulo 6.)

La generalización de esta simple idea de reutilización de subsecuencias predefinidas conduce al concepto clásico de *subprograma*. En un subprograma habría que definir la transferencia de argumentos.

EJERCICIO. Descríbase de nuevo el ejemplo de §2.4.2 reutilizando las subsecuencias de funcionamiento que se pueda. Compruébese que se llega a obtener una red de Petri distinta pero de la misma complejidad.

2.4.4 Lectores y redactores

Enunciado. Vamos a considerar un ejemplo de sincronización clásico en la litera-

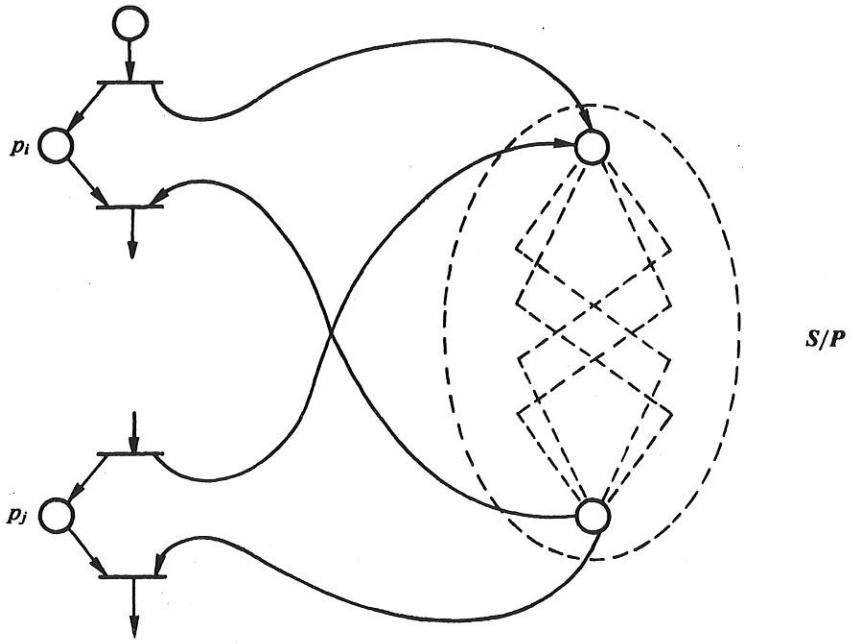


Figura 2.8. Esquema de utilización de subprogramas.

tura informática. Pertenece a la clase de problemas en los que dos conjuntos de usuarios (lectores y redactores) tienen que coordinarse para acceder a unos datos comunes (recurso que comparten). Ignoramos la estructura de los datos compartidos; nos basta con saber que:

—los lectores sólo inspeccionan, y por tanto pueden acceder simultáneamente (concurrencia) a los datos;
—los redactores modifican los datos, y por ello todo redactor debe trabajar en exclusión mutua con el resto de los usuarios (lectores y redactores).

Cada usuario puede encontrarse en uno de los tres estados siguientes: activo (trabaja con el recurso), espera (pendiente de acceder al recurso) y reposo (no necesita el recurso).

COMENTARIO. La figura 2.9 representa una descripción posible para el caso de dos lectores y dos redactores. Desde el punto de vista de la modelación, interesa resaltar el significado de los lugares X_1 y X_2. El lugar X_i implica la exclusión mutua entre los redactores y el i-ésimo lector. Si se tienen en cuenta simultáneamente X_1 y X_2, se llega a la conclusión de que los redactores están en exclusión mutua entre sí y en exclusión mutua con los lectores.

EJERCICIO. Descríbase un sistema similar al anterior pero en el que dos usuarios o procesos sean ambos capaces de leer y escribir.

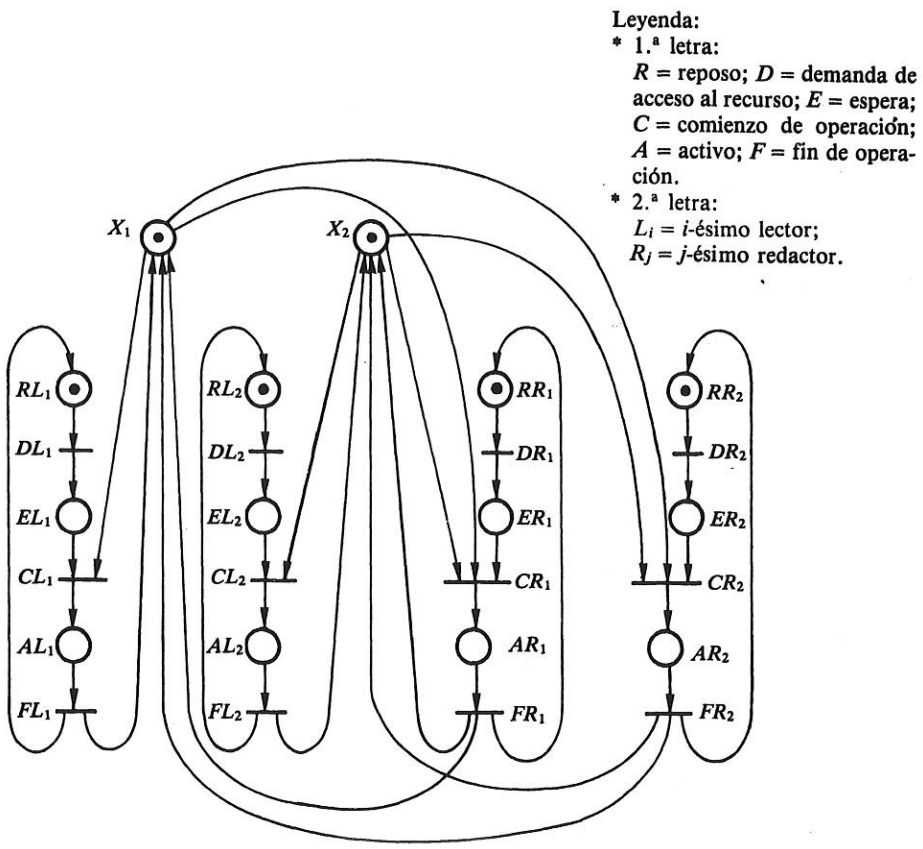


Figura 2.9. Descripción del sistema de dos lectores y dos redactores.

2.5 REDES DE PETRI ORDINARIAS: PRINCIPALES SUBCLASES Y EXTENSIONES

En este apartado, tomando como referencia las redes de Petri ordinarias, presentamos algunas de las principales subclases y extensiones.

La definición de subclases se realizará exclusivamente introduciendo restricciones en la estructura de las RdP. Al restringir la generalidad del modelo, cabe pensar que será más fácil el estudio de su comportamiento dinámico. Como se verá, entre las subclases aparecerá el conocido grafo de estados, subclase a partir de la que se introdujeron de forma intuitiva las RdP en el capítulo 1.

Por otra parte, el deseo de obtener descripciones más condensadas y fáciles de utilizar, o la obtención de una mayor potencia descriptiva, han sido las dos principales razones por las que se han definido diversas generalizaciones de las RdP.

2.5.1 Subclases de redes de Petri ordinarias

Vamos a definir cuatro subclases de redes y a establecer las relaciones entre ellas. Al definir las diferentes subclases mediante restricciones en la estructura de las RdP ordinarias, la determinación de aquélla a la que pertenece una red dada es muy sencilla.

Desde un punto de vista práctico, la utilización de estas subclases es bastante importante, como lo demuestra el hecho de que la mayor parte de las redes presentadas hasta ahora (capítulos 1 y 2 hasta §2.4) pertenezcan a alguna de éstas.

Definición 2.17. Un *grafo de estados* (GE) o *máquina de estados* (ME) es una RdP tal que:

$$\forall t \in T \quad |{}^{\cdot}t| = 1 \text{ y } |t^{\cdot}| = 1,$$

es decir, tal que en ella toda transición tiene un lugar de entrada y uno de salida. □

Un grafo de estados no posee nudos Y.

Es importante observar que el concepto de grafo de estados, considerado como subclase de RdP, es más general que el utilizado de forma clásica, puesto que puede tener más de una marca. No obstante, si se trabaja con RdP binarias, entonces habrá coincidencia entre los dos conceptos: GE en el sentido habitual y GE como subclase de RdP.

Observación. Con el prósito de evitar ambigüedades en el texto, esta desafortunada coincidencia en la nomenclatura es la que nos impulsó a denominar grafo reducido (GR) al grafo de estados binario provisto de la interpretación presentada en §1.4.

Un grafo de estados monomarcado (binario) modela los sistemas como si fuesen secuenciales; es decir, no puede modelar directamente los sistemas concurrentes. Puesto que en general un grafo de estados puede contener varias marcas, es posible modelar la *reapelabilidad*†. En cualquier caso, es evidente que un grafo de estados sólo modela sistemas de estados finitos.

Definición 2.18. Un *grafo marcado* (GM) o *grafo de sincronización* es una RdP tal que:

$$\forall p \in P \quad |{}^{\cdot}p| = 1 \text{ y } |p^{\cdot}| = 1,$$

es decir, tal que en ella todo lugar tiene como máximo una transición de entrada y una transición de salida. □

Un grafo marcado no posee nudos O.

---

† Un programa es *reapelable* (en inglés, *reenterable*) si puede ser compartido por varios usuarios en un sistema de multiprogramación (véase [MEIN 72]).

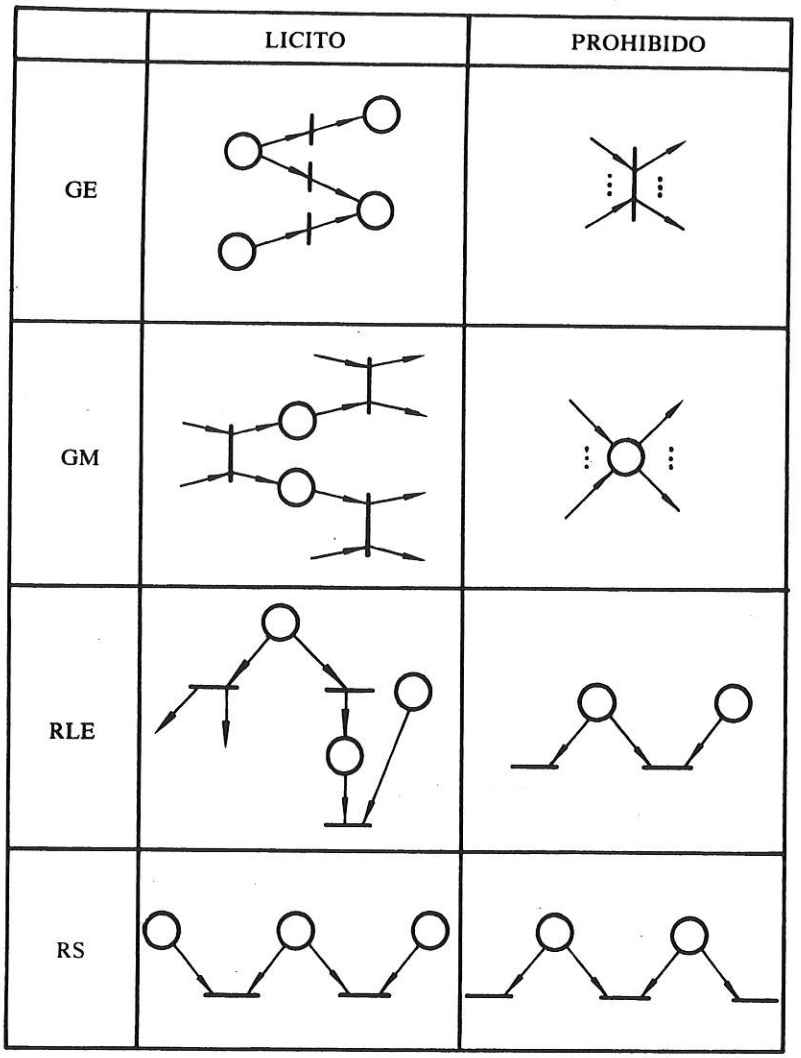


Figura 2.10 Ilustración de las definiciones de subclases de redes de Petri ordinarias.

Las RdP de la figura 1.15 son GM. Los GM pueden modelar sistemas de ordenación de actividades como permiten los grafos PERT (*Program Evaluation and Review Technique*).

En este punto conviene destacar que las capacidades de modelado de los GE y los GM son duales. En efecto, un GE puede modelar alternativas, pero no el paralelismo. Por otra parte, un GM puede modelar la creación o destrucción de marcas o determinadas sincronizaciones entre actividades (figura 1.15), pero no puede expresar alternativas en la evolución. De lo anterior se desprende que un GM puede modelar sistemas en los que el número de estados (marcados) no es finito.

Las subclases que introducimos a continuación contienen a los GE y GM; por lo tanto, pueden modelar las alternativas y la concurrencia, aunque no en todos los casos que permiten las redes de Petri ordinarias.

Definición 2.19. Una RdP *libre elección* (RLE) es una red tal que:

$$\forall p \in P, \text{ si } |p^{\bullet}| > 1, \text{ entonces } \forall t_k \in p^{\bullet}, \ |{}^{\bullet}t_k| = 1.$$

Es decir, si dos transiciones t_i y t_j tienen un lugar de entrada p en común, se deduce que p es el único lugar de entrada de t_i y de t_j. □

Desde un punto de vista funcional, las RLE permiten la construcción de modelos en los que o todas o ninguna de las transiciones de salida de cada lugar están sensibilizadas. En una RLE, siempre que un lugar esté marcado y posea más de una transición de salida, es posible elegir libremente (independientemente del resto del marcado) la transición que se disparará. De ahí su denominación.

Definición 2.20. Una *red de Petri simple* (RS) es una RdP en la que toda transición tiene como máximo un lugar de entrada compartido con otras transiciones. □

Como veremos más adelante (capítulo 5), a pesar de su relativa generalidad, esta subclase de RdP tiene importantes propiedades. Muchos sistemas se modelan con RS. Por citar algún ejemplo, un recurso compartido por dos o más usuarios.

La figura 2.11 ilustra las relaciones de inclusión entre las subclases definidas.

EJERCICIO. Determínense las subclases a las que pertenecen las redes presentadas hasta ahora.

2.5.2 Extensiones de las redes de Petri ordinarias

El objetivo de este subapartado es presentar varias extensiones de las redes de Petri ordinarias y su aplicación al modelado de sistemas discretos concurrentes. Del con-

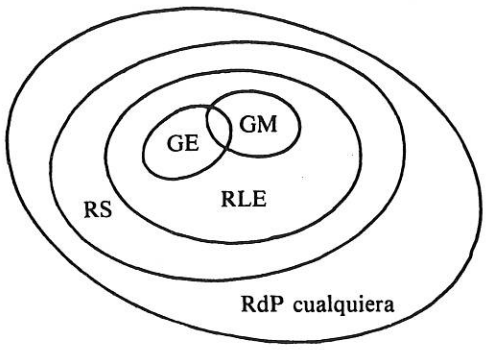


Figura 2.11. Relaciones de inclusión entre las diferentes subclases de redes de Petri ordinarias.

junto de extensiones que consideraremos, la mayor parte sirve para la construcción de modelos funcionales (descripción de *qué* se hace) y otra (§2.5.2.6) sirve para la construcción de modelos comportamentales, de interés para el análisis cuantitativo de prestaciones. La extensión que es utilizable en esta segunda categoría de aplicaciones se denomina *redes de Petri temporizadas*.

2.5.2.1 *Redes de Petri generalizadas*

La primera extensión de las redes de Petri ordinarias (RdP) son las *redes de Petri generalizadas* (RdPG). Las RdPG se utilizan fundamentalmente en la modelación de sistemas a un nivel bastante elevado (grandes bloques y sus principales relaciones).

La definición 2.1 (§2.2.1) presentó el concepto de red de Petri generalizada. Con respecto a las redes ordinarias, las RdPG introducen el concepto de *peso de un arco*. La RdPG de la figura 2.12*a* es tal que el peso de los arcos que van de la transición a a p_2 y de p_3 a a es k: $\beta(a, p_2) = \alpha(p_3, a) = k$.

EJEMPLO. Se desea modelar la sincronización entre un productor y un consumidor. El productor es un disco de un sistema informático que produce bloques de k líneas. El consumidor es una impresora que consume línea por línea. La interconexión productor-consumidor se establece gracias a una memoria tampón FIFO de capacidad k líneas, a la que se puede acceder simultáneamente en lectura y escritura.

En la figura 2.12 se encuentran dos modelos, ambos construidos con RdPG, que difieren en la interpretación asociada a las redes. En el primero de ellos (figura 2.12*a*) las acciones se asocian al disparo de las transiciones. De este modo, el disparo de la transición a representa el depósito en memoria de un bloque (k líneas), mientras que el disparo de b representa la producción de un bloque. De forma simétrica, los disparos de c y d representan la extracción de la memoria de una línea y su (posterior) consumo. Según lo establecido, resulta evidente que p_1 y p_5 representan los estados de espera para acceder a la memoria por parte del productor y el consumidor, respectivamente. El marcado de p_2 representa el número de líneas de la memoria que se encuentran vacías. El marcado de p_4 representa el número de líneas preparadas para ser escritas por la impresora.

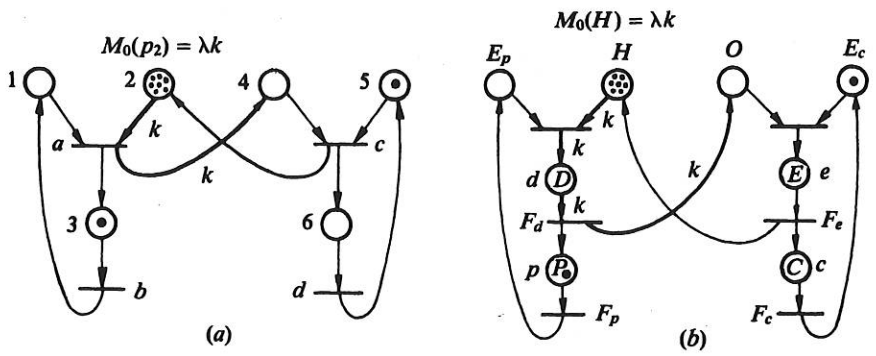


Figura 2.12. Modelación de un sistema productor (por paquetes)-consumidor utilizando redes de Petri generalizadas.

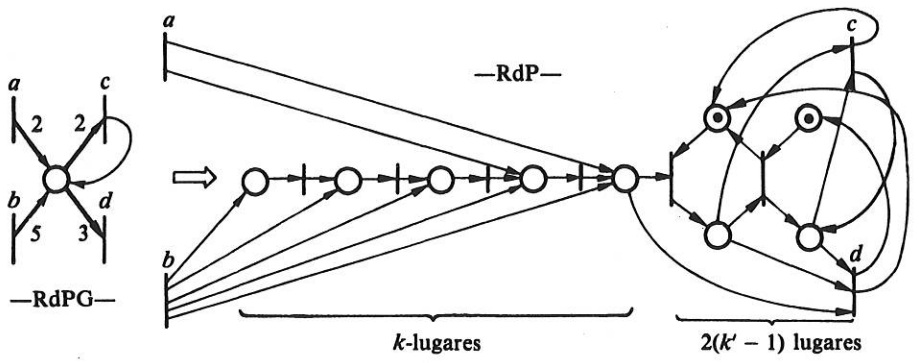


Figura 2.13. Simulacion con una red de Petri ordinaria de una descripción realizada con red de Petri generalizada ($k = 5$, $k' = 3$).

La RdPG de la figura 2.12*b* representa el sistema considerado, pero los eventos se asocian a las transiciones, y las acciones sólo a los lugares. Se trata del tipo de interpretación definido en §2.3 y utilizado, por ejemplo, en el sistema productor-consumidor de la figura 2.6. La notación empleada es la misma que se utilizó en aquel caso.

Las RdPG, como extensión de las RdP, aportan una indiscutible facilidad a la modelación, pero toda descripción realizable con RdPG puede transformarse en una basada en RdP. Es decir, *la potencia de descripción de las* RdP *y de las* RdPG *es idéntica.* En efecto, toda RdPG puede ser simulada mediante una RdP si se respetan las transiciones de la primera y se reemplaza cada lugar p por: (1) una fila de k lugares, donde k es el peso máximo de sus arcos de entrada, y (2) dos filas de $k' - 1$ lugares, donde k' es el peso máximo de sus arcos de salida. Una transformación básica mediante la que una subRdP simula una subRdPG es la ilustrada por la figura 2.13. Como se puede comprobar, si se consideran las transiciones etiquetadas, $\{a, b, c, d\}$, la mencionada transformación permite definir las mismas secuencias de disparo.

EJERCICIO. Obténgase una RdP que simule la RdPG de la figura 2.12*b*. ¿Es la única RdP posible?

2.5.2.2 *Redes de Petri con capacidad limitada*

En este apartado introduciremos una extensión de las RdPG, y por lo tanto de las RdP, con la que pretendemos facilitar la tarea de modelación de sistemas complejos, especialmente si existen numerosos almacenes, memorias, etc. (su capacidad siempre estará limitada físicamente).

Definición 2.21. Una *red de Petri con capacidad limitada* (RdPC) es una quíntupla $\langle P, T, \alpha, \beta, \gamma \rangle$, donde $\boldsymbol{R} = \langle P, T, \alpha, \beta \rangle$ es una RdPG y γ es una función que asocia a cada lugar su capacidad; es decir, el máximo número de marcas que puede contener ($\gamma: P \to \mathbb{N} \cup \{\infty\}$).

$\gamma(p)$ es la *capacidad* del lugar p. □

Teniendo en cuenta que el disparo de una transición t conduce del marcado M_i al marcado M_j definido por (definición 2.9):

$$M_j(p) = M_i(p) + \beta(t,p) - \alpha(p,t) \quad \forall p \in P,$$

resulta que el marcado de una RdPC evoluciona de acuerdo con la regla de sensibilización de transiciones siguiente: una transición t *está sensibilizada* sii:

[1] $\forall p \in {}^{\cdot}t \quad M(p) \geqslant \alpha(p,t)$.
[2] $\forall p' \in t^{\cdot} \quad M(p') + \beta(t,p') - \alpha(p',t) \leqslant \gamma(p')$.

La primera condición es la considerada en la definición 2.8. La segunda condición indica que, para que t esté sensibilizada, es necesario que el marcado que se obtenga después de su disparo no viole las restricciones de capacidad de sus lugares de salida.

A partir de las dos condiciones anteriores, se deduce inmediatamente que para que toda transición pueda estar sensibilizada (sea disparable, pues de lo contrario la transición será superflua) es necesario que:

[1] $\gamma(p) \geqslant \alpha(p,t)$
[2] $\gamma(p) \geqslant \beta(t,p)$.

En efecto, la condición 1 se deduce inmediatamente:

$$\left.\begin{array}{ll} \gamma(p) \geqslant M(p) & \text{[por definición]} \\ M(p) \geqslant \alpha(p,t) & \text{[1]} \end{array}\right\} \Rightarrow \gamma(p) \geqslant \alpha(p,t).$$

La condición 2 se deduce al considerar:

$$\left.\begin{array}{l} \text{[1]} \quad M_i(p) \geqslant \alpha(p,t) \Rightarrow M_i(p) - \alpha(p,t) \geqslant 0 \\ \text{[2]} \quad M_j(p) = M_i(p) - \alpha(p,t) + \beta(t,p) \end{array}\right\} \Rightarrow M_j(p) \geqslant \beta(t,p).$$

Pero, dado que es necesario que $\gamma(p) \geqslant M_j(p)$, se concluye que $\gamma(p) \geqslant \beta(t,p)$.

Por último, para respetar la restricción sobre la capacidad de un lugar, hemos de tener, para el marcado inicial, $M_0(p) \leqslant \gamma(p)$.

EJEMPLO. La figura 2.14 modela el caso considerado en la figura 2.12. La capacidad de p_4 es la de la memoria tampón, $\gamma(p_4) = \lambda k$.

Al igual que ocurrió al definir las RdPG, la extensión de éstas a RdPC sólo facilita la modelación en ciertos casos. En efecto, es obvio que toda RdPG es una RdPC en la que la capacidad de cada uno de sus lugares es infinita. Por otra parte, a toda RdPC se le puede asociar una RdPG y, por consiguiente, una RdP que la simula. Para justificar esta última afirmación vamos a introducir un nuevo concepto.

Definición 2.22. Un lugar $\bar{p}$ *es complementario del lugar* p si $\forall t \in T\ \alpha(p,t) = \beta(t,\bar{p})$ y $\beta(t,p) = \alpha(\bar{p},t)$. Es decir, el disparo de cualquier transición de la red quita o añade tantas marcas de p como añade o quita de $\bar{p}$. □

Una propiedad interesante que se deduce inmediatamente de lo establecido es que el número total de marcas que poseen p y $\bar{p}$ no varía: $M(p) + M(\bar{p}) = k$ (figura

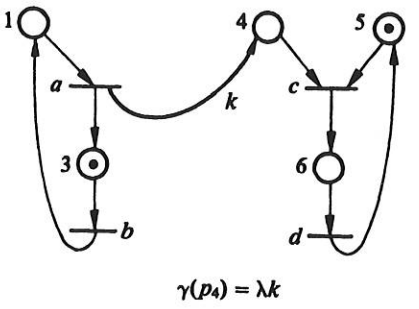


Figura 2.14. Modelación mediante una red de Petri con capacidad del sistema productor-consumidor de la figura 2.12.

2.15). (De acuerdo con las definiciones que se presentarán en §4.7.2.2, p y $\bar{p}$ determinan una componente conservativa.) De este modo, los lugares p_2 y p_4 (figura 2.12) son complementarios: $M(p_2) + M(p_4) = \lambda k$.

A partir de la definición de lugar complementario se puede concebir la regla que permite simular una RdPC con una RdPG.

REGLA. A cada lugar p que posea una capacidad finita, se le asocia su lugar complementario $\bar{p}$ marcado inicialmente de acuerdo con la siguiente expresión: $M_0(\bar{p}) = \gamma(p) - M_0(p)$.

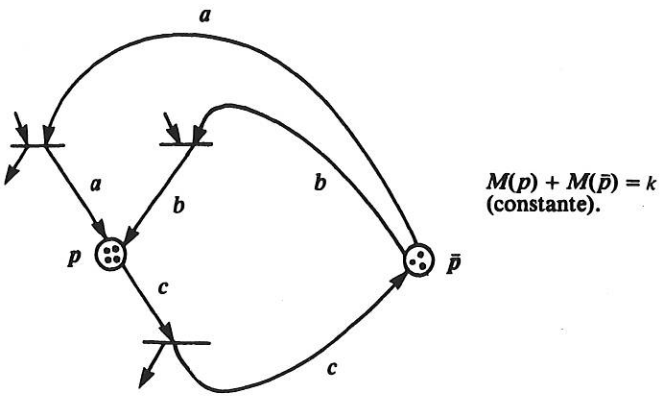


Figura 2.15. Los lugares p y $\bar{p}$ son complementarios ($k = 7$).

2.5.2.3 *Redes de Petri con arcos inhibidores*

Como hemos visto anteriormente, las RdPG (§2.5.2.1) y las RdPC (§2.5.2.2) facilitan el modelado de sistemas discretos concurrentes, pero su potencia descriptiva es idéntica a la de las RdP. En este apartado introducimos una extensión que no sólo facilita la descripción de determinados sistemas, sino que hace posible la descripción de aquellos que no pueden ser modelados con RdP. Partamos de uno de estos casos, en el que consideramos un recurso (un túnel) compartido por dos filas, teóricamente con un número *ilimitado* de usuarios (trenes).

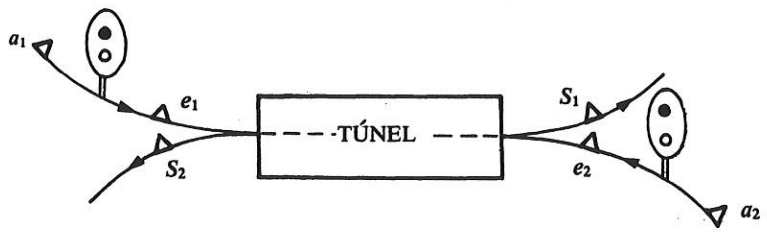


Figura 2.16. Esquema del túnel con sus accesos.

EJEMPLO. Supongamos un túnel con una sola vía férrea que puede ser utilizado por trenes que lleguen en los dos sentidos (figura 2.16), procedentes de dos vías férreas distintas.

Las señales a_i, e_i y s_i indican la aproximación, la entrada y la salida del túnel, respectivamente, de un tren en el sentido «i».

El sistema debe controlar el tráfico de acceso al túnel. Para ello elabora las acciones v_i (verde) y r_i (rojo), que autorizan y prohiben, respectivamente, la entrada al túnel en el sentido «i».

Se desea que las salidas del sistema de control sean tales que $r_1 = r_2 = 1$ en los dos casos siguientes:

a) si no hay tren en espera ni recorriéndolo (el automatismo no sirve más que para autorizar una vía de paso: *seguridad intrínseca*);

b) si hay un tren que recorre el túnel.

Un tren que espera en el sentido 1 tiene preferencia sobre un tren que espera en el sentido 2. Supóngase que puede haber un número ilimitado de trenes que esperan en cada sentido.

La descripción directa del sistema propuesto con una RdP (o RdPG, o RdPC) es imposible. En efecto, supongamos que E_1 es un lugar cuyo marcado representa el número de trenes que circulan en el sentido 1 y que esperan para entrar en el túnel. La asignación del túnel (recurso) a un tren que circula en el sentido 2 tiene que hacerse cuando E_1 está desmarcado. Ahora bien, las RdP no permiten la comprobación directa de la ausencia de marcas en un lugar, puesto que el disparo de una transición θ condicionada por un lugar de entrada E_1 [$E_1 \in {}^{\cdot}\theta$] exige que el arco (E_1, θ) sea de peso no nulo, $\alpha(E_1, \theta) > 0$.

Reformulando la conclusión sobre el modelado del ejemplo anterior, podemos afirmar que la limitación de las RdP reside en que las condiciones que permiten disparar una transición (provocar el cambio de estado o marcado) exigen que el número de marcas de cada lugar de entrada sea estrictamente mayor que cero. Es decir,

no se puede comprobar directamente el que un lugar no disponga de marcas, *test de cero*. A continuación presentamos una extensión de las RdP que, mediante unos arcos especiales denominados «inhibidores», añade la capacidad de comprobar directamente la ausencia de marcas en un lugar. Posteriormente veremos cómo se puede simular el arco inhibidor con una RdP cuando el lugar puede contener como máximo k (acotado) marcas.

Definición 2.23. Una *red de Petri con arcos inhibidores* (RdPAI) es una red a la que se añaden unos arcos que sólo parten de lugares y van a transiciones, denominados arcos inhibidores, $I(p, t)$.

La regla de disparo de una transición en una RdPAI exige que estén desmarcados todos los lugares que se encuentren unidos a la transición mediante un arco inhibidor. □

Desde un punto de vista gráfico, un arco inhibidor se distingue de un arco normal merced a una pequeña circunferencia situada en el extremo que incide sobre la transición. La figura 2.17 ilustra la sensibilización y el disparo de una transición a la que llega un arco inhibidor. Como se puede observar, la notación gráfica del arco inhibidor deriva de las convenciones de representación de los circuitos lógicos, en los que toda circunferencia pequeña significa negación (inhibición).

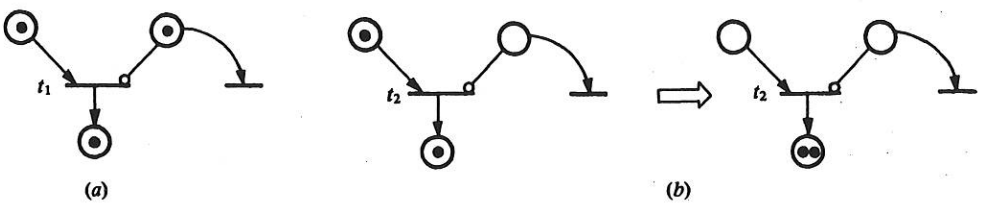


Figura 2.17. RdPAI: (*a*) t_1 no está sensibilizada;
(*b*) disparo de la transición sensibilizada t_2.

COMENTARIOS SOBRE EL EJEMPLO DEL TÚNEL

1) La figura 2.18 modela el caso con una RdPAI interpretada. Cada vez que se produce $a_i\uparrow$, el lugar E_i aumenta su número de marcas en una unidad. Se puede comprobar fácilmente que el marcado de E_i representa el número de trenes que esperan para entrar en el túnel en el sentido «i».

2) Los lugares m_i (figura 2.18) se pueden suprimir (lugares identidad, §3). En esas condiciones, las transiciones etiquetadas a_i se denominan transiciones *fuente* (véase §3.5.1). La regla de evolución del marcado es tal que, si se produce $a_i\uparrow$, se dispara la transición.

3) Al ser (teóricamente) ilimitado el número de trenes que esperan en cada sentido, el sistema no se puede modelar con un autómata de estados finitos. La figura 2.19 propone una modelación alternativa basada en una descomposición PC-PO,

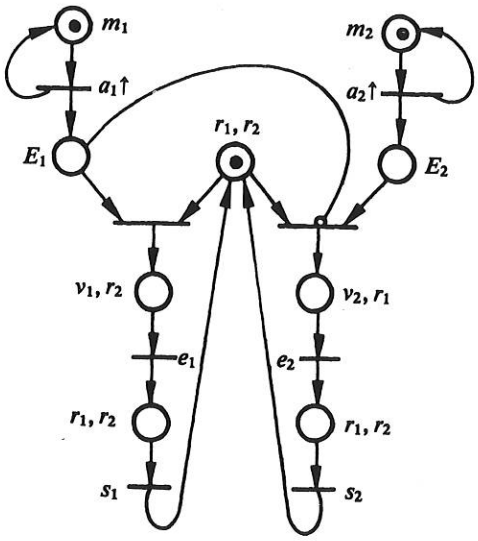


Figura 2.18. Modelación del control del túnel (figura 2.16) con una RdPAI (RdP con arcos inhibidores).

en la que la PC es un GE binario. Evidentemente, se supone que los contadores son de capacidad ilimitada. Obsérvese cómo se realiza el test de cero, mediante la interpretación asociada al GE y no mediante su *estructura*. En la descripción de la figura 2.19 se han «ocultado» las evoluciones simultáneas del sistema al separar los contadores de la PC.

Desde un punto de vista teórico, se puede enunciar que las RdPAI tienen la potencia descriptiva de las máquinas de TURING [PETE 81*a*]. Desde un punto de vista práctico, se puede afirmar que si el número de marcas de los lugares de una RdPAI que son origen de arcos inhibidores está acotado (limitado), existe una RdP que simula la primera. Es decir, en este caso particular las RdPAI no aportarán más que una facilidad para la modelación. Las reglas que siguen permiten simular, en el caso últimamente mencionado, una RdPAI con una RdPG:

REGLA 1. A cada lugar p que posea un arco inhibidor se le asocia un lugar complementario $\bar{p}$ marcado inicialmente de acuerdo con la siguiente expresión: $M_0(\bar{p}) = k(p) - M_0(p)$.

Como p y $\bar{p}$ son complementarios, para cualquier marcado alcanzable, M, se tendrá: $M(\bar{p}) = k(p) - M(p)$. El entero $k(p)$ representa el máximo número de marcas que puede llegar a contener p al evolucionar el marcado en la RdPAI.

REGLA 2. Si t no es transición de entrada de p (figura 2.20), el arco inhibidor $I(p, t)$ se sustituye por dos arcos normales (no inhibidores) tales que $\alpha(\bar{p}, t) = \beta(t, \bar{p}) = k(p)$. En efecto, leer $k(p)$ marcas de $\bar{p}$ es lo mismo que leer 0 marcas de p, habida cuenta que $M(\bar{p}) = k(p) - M(p)$.

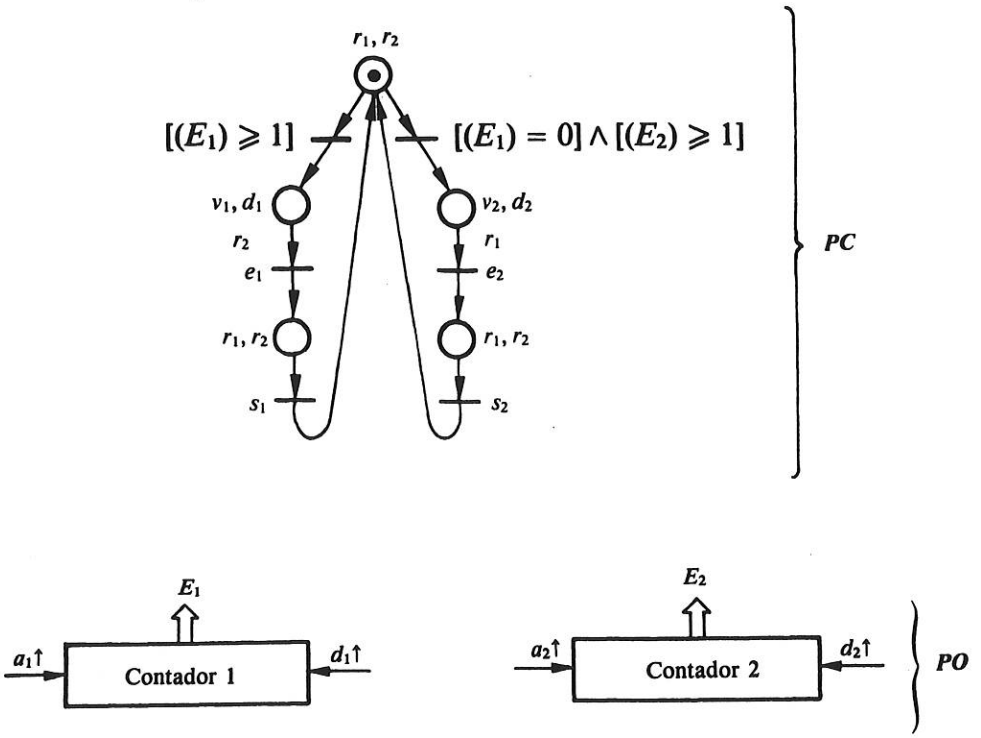


Figura 2.19. Modelación del control del túnel (figura 2.16) mediante descomposición (separación PC-PO). (*Nota*: d_i es una señal de salida de la PC que provoca la decrementación del contador i de la PO.)

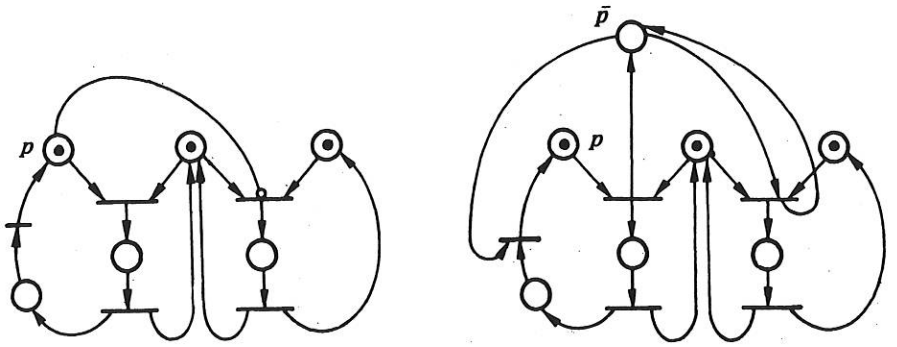


Figura 2.20. Transformación de una RdPAI limitada en una RdP que la simula (que posee idénticas secuencias de disparo).

En caso contrario, si t (T_2 en la figura 2.21*a*) es transición de entrada de p, el arco inhibidor se sustituye tomando $\alpha(\bar{p}, t) = k(p)$ y $\beta(t, \bar{p}) = k(p) - \beta(t, p)$. (Compruébelo el lector.) Es decir, se modifica el peso del arco $(t, \bar{p})$. Eventualmente, si $k(p) = \beta(t, p)$, el arco $(t, \bar{p})$ desaparece [véase la figura 2.21, tomando $k(p) = \beta(t, p) = 1$].

La transformación ilustrada por la figura 2.21 permite comprobar cómo al utilizar las RdPAI se puede simplificar la descripción de la figura 2.7, realizada sin arcos inhibidores.

A modo de observación final, es importante señalar que las reglas de simulación de una RdPAI con una RdPG enunciadas anteriormente *no son válidas si $k(p)$ no es finito*. En este caso, las RdPAI no pueden ser simuladas por RdPG (o RdP, o RdPC); las RdPAI son más potentes.

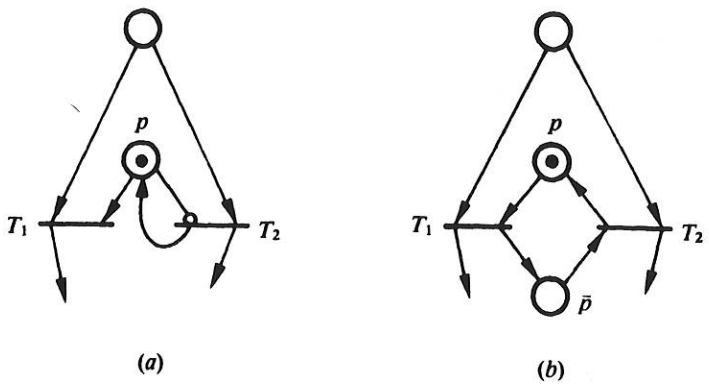


Figura 2.21. Transformación de subred con arco inhibidor cuando T_2 es transición de entrada de p y, simultáneamente, de salida de un arco inhibidor que proviene de p. (*Nota*. La subred *b* aparece dos veces en la red de la figura 2.7.)

2.5.2.4 *Redes de Petri y transiciones no estándar*

En toda red de Petri la función lógica que expresa la condición para que una transición esté sensibilizada (precondición) es una *intersección*. Así, si $\{p_1, p_2, p_3\}$ es el conjunto de lugares de entrada de t, la sensibilización de t se expresa como sigue:

$$[M(p_1) \geqslant \alpha(p_1, t)] \text{ AND } [M(p_2) \geqslant \alpha(p_2, t)] \text{ AND } [M(p_3) \geqslant \alpha(p_3, t)].$$

Del mismo modo, el disparo de una transición provoca la modificación del marcado de *todos* los lugares de salida. Así, si $\{p_4, p_5\}$ es el conjunto de lugares de salida de t, tendremos:

$$[M(p_4) := M(p_4) + \beta(t, p_4)] \text{ AND } [M(p_5) := M(p_5) + \beta(t, p_5)].$$

Según lo anterior, diremos que la lógica del disparo de transiciones en una RdP es del tipo INTERSECCIÓN de *entrada* y de *salida* (AND—*input/output*).

Con el objeto de facilitar el modelado de sistemas, se han propuesto diversas alternativas a la lógica del disparo de una transición en las redes de Petri, resultando otras tantas extensiones. Por ejemplo, la figura 2.22*a* representa una transición t con lógica O-EXCLUSIVO de *entrada*/INTERSECCIÓN de *salida*. Su simulación directa con RdP no es posible. La figura 2.22*b* presenta la simulación con RdPAI. Evidentemente, si los lugares de entrada de t son limitados, la RdPAI de la figura 2.22*b* puede ser simulada con una RdPG (§2.5.2.3).

EJERCICIO. Simúlese con una RdPAI una transición-conmutador t (*switch* [BAER 73]) tal que ${}^{\bullet}t = \{p, s\}$ y $t^{\bullet} = \{p_0, p_1\}$. t se sensibiliza sólo si $M(p) > 0$. El disparo de t coloca una marca en p_0 si $M(s) = 0$, o en p_1 si $M(s) > 0$; además, si s estaba marcado, se desmarca.

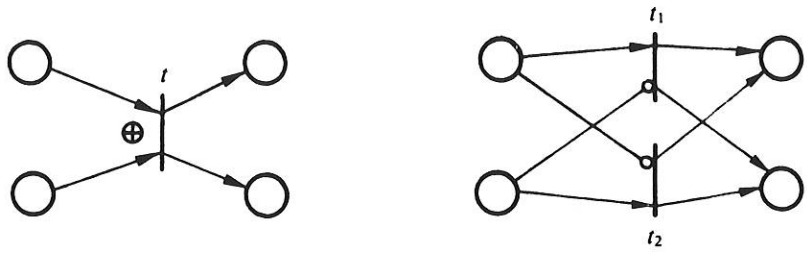


Figura 2.22. Transformación de las transiciones O-EXCLUSIVO en RdPAI.

2.5.2.5 *Redes de Petri coloreadas* †

Las *redes de Petri coloreadas* han sido introducidas para condensar la descripción (y el análisis) de sistemas en los que se identifican diversos subsistemas con estructura y comportamiento similares, pero que trabajan en paralelo. A modo de ejemplo, podemos considerar el caso de los lectores y redactores de §2.4.4.

En una red de Petri coloreada, cada marca puede portar un color que la identifique. A cada lugar y a cada transición se le asigna un conjunto de colores. Una transición puede dispararse respecto a cada uno de sus colores. El disparo de una transición elimina y añade marcas como en las RdP, pero respetando la dependencia funcional especificada entre el color del disparo de la transición y los colores de las marcas. El color de cada marca puede ser cambiado por el disparo de una transición.

Como aplicación elemental, la red de la figura 2.23 representa el sistema de dos lectores, $\{a, b\}$, y dos redactores, $\{c, d\}$. Comparando este modelo con el de la figura 2.9, se comprende inmediatamente que lo que se ha conseguido es una superposi-

† La presentación que realizamos es meramente intuitiva y simplista. La definición formalizada se encuentra en [JENS 81].

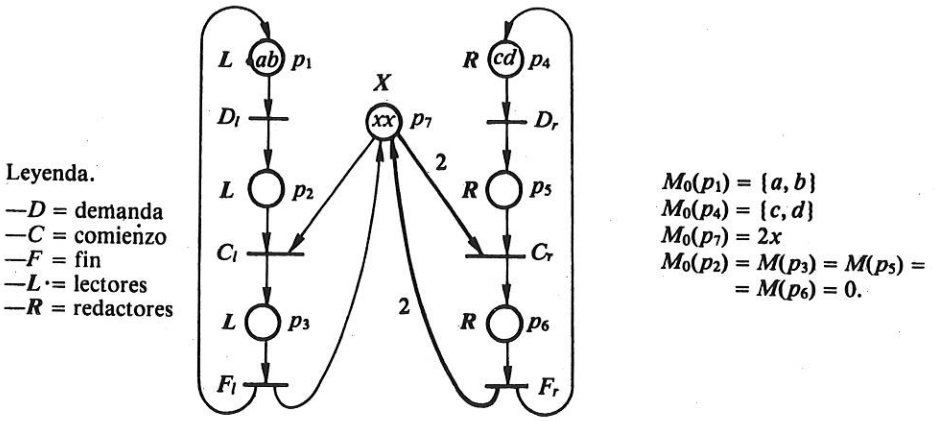


Figura 2.23. Representación parcial del sistema de dos lectores ($L = \{a, b\}$) y dos redactores ($R = \{c, d\}$).

ción de subredes (*folding*) gracias a la diferenciación de las marcas mediante colores.

En la red coloreada de la figura 2.23 se definen los colores siguientes:

—lector, $L = \{a, b\}$;
—redactor, $R = \{c, d\}$;
—accesibilidad de la memoria, $X = \{x\}$.

A los lugares $\{p_1, p_2, p_3\}$ se les asocia los colores L, a los lugares $\{p_4, p_5, p_6\}$ se les asocia los colores R y al lugar p_7 se le asocia el color X.

El disparo de la transición D_l, si es debido a $l = a$, provoca el paso de la marca a al lugar p_2. El disparo de C_l, si es debido a $l = a$, provoca el paso de la marca a al lugar p_3, mientras que p_7 pierde una marca. La presencia de la marca a en p_3 representa la lectura por parte del lector a.

A modo de comentario final, hemos de señalar que las RdP coloreadas no incrementan la potencia descriptiva de las RdP ordinarias [PETE 81*b*], lo cual se comprende de forma intuitiva, puesto que no se puede ejecutar el test de cero.

2.5.2.6 *Redes de Petri temporizadas* (RdPT)

El tiempo ha sido introducido de diversas formas en las RdP. Vamos a dar dos definiciones de RdP temporizadas. Se deja a la consideración del lector la reflexión sobre la relación entre ambas.

Definición 2.24. Una RdP *temporizada* (RdPT) es un par $\langle R, Z\rangle$ tal que $R = \langle P, T, \alpha, \beta\rangle$ y Z es una función que asigna un número real no negativo, z_i, a cada transición de la red: $Z: T \to \mathbb{R}^+$.

$z_i = Z(t_i)$ se denomina *tiempo de disparo* de la transición t_i. □

La regla de evolución del marcado es idéntica a la de una RdPG. La única cuestión a tener en cuenta es que el disparo de t_i dura z_i unidades de tiempo.

Definición 2.24 bis. Una RdPT es un par $\langle R, Z\rangle$ tal que $R = \langle P, T, \alpha, \beta\rangle$ y Z es una función que asigna un número real no negativo, z_i, a cada lugar de la red: $Z: P \rightarrow \mathbb{R}^+$. □

Una marca en una RdPT puede encontrarse en dos estados: *dispuesta* o *indispuesta*. Las reglas de evolución del marcado son idénticas a las de las RdP si las marcas están dispuestas. Si las marcas no están dispuestas, es como si no existieran en lo que se refiere a la evolución en la red. Al pasar a un lugar una marca, entra en estado indispuesto y pasa a estado dispuesto después de $z_i = Z(p_i)$ unidades de tiempo.

Esta clase de modelos se utiliza normalmente para la evaluación de *prestaciones* (*perfomances*).

Las RdPT serán consideradas en el capítulo 5.

En la literatura técnica se describen otras extensiones de las RdP. En general, corresponden a tipos específicos de aplicaciones.

EJERCICIOS

2.1 Estúdiese la RdP de la figura E.2.1. ¿Cuál es el papel funcional de p_3, p_6, p_9 y p_{12}? Evalúese el número de estados de un GR equivalente.

2.2 Obténgase una RdPAI equivalente a la RdP de la figura E.2.1. ¿Qué tipo de sistema describe?

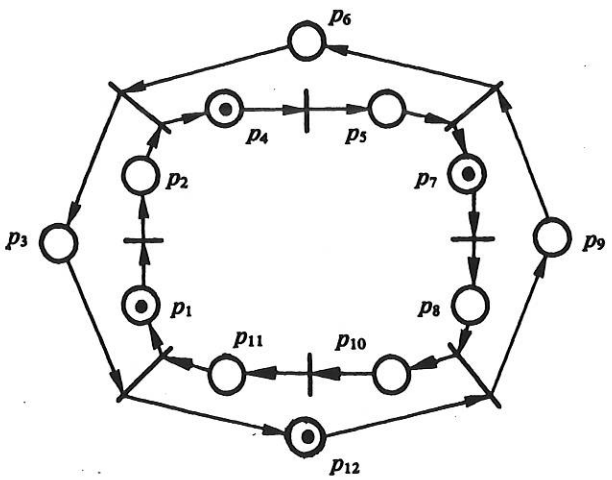


Figura E.2.1

3

Simplificación de una descripción

3.1 INTRODUCCIÓN

La forma en que el diseñador describe un sistema condiciona *en parte* la complejidad del correspondiente modelo. Esta consideración ha generado siempre un enorme interés por los métodos formales de minimización de las descripciones.

Los métodos clásicos de síntesis de sistemas secuenciales (cuyo nacimiento y maduración podemos situar entre las décadas de los 50 y los 60) tienden hacia la obtención de una realización que tenga el mínimo número de componentes (puertas, biestables). Este objetivo fue motivado por la fuerte incidencia del coste de los componentes en el coste total del equipo.

El proceso de síntesis clásico busca la minimización del modelo del sistema; es decir, trata de obtener, en una primera fase, un (el) autómata que modele el sistema y tenga la menor cantidad posible de estados. No obstante, cabe resaltar que esta aproximación no permite garantizar la consecución del objetivo enunciado. Dicho de otro modo, la minimización de la descripción impide *a veces* minimizar la realización posterior.

La distribución de los costes de estudio, fabricación y utilización de los sistemas de control es tan diferente en la actualidad que ya no es cuestión de perseguir la realización mínima (o cuasi-mínima). Los criterios de maximización de la modificabilidad, de la reparabilidad, etc., desempeñan un papel absolutamente preponderante. No obstante, la cuantificación de estos objetivos se hace tan compleja, que sólo podrán definirse las líneas fundamentales en las que se basa la búsqueda de «las buenas realizaciones».

Los métodos de simplificación de una descripción propuestos en este capítulo se centran en dos ideas fundamentales:

1) reducir, ¡no minimizar!, el número de lugares y/o transiciones de la descripción inicial;
2) conservar «el sentido físico» de la descripción inicial, lo que conduce a que se realicen sólo simplificaciones locales.

Al carecer de una definición formal de los criterios para la reducción de la complejidad, la simplificación se deberá realizar en pasos sucesivos, dejando las decisiones al diseñador. Nótese en este punto que se establece de nuevo una diferencia fundamental con los métodos clásicos de síntesis, en los que la minimización se realiza de forma totalmente automática. Esta diferencia de objetivos y técnicas utilizadas justifica el cambio de denominación: *simplificación* en lugar de *minimización*.

A pesar de que la simplificación constituye una etapa optativa dentro del proceso de concepción, pensamos que el conocimiento de las técnicas de simplificación que se presentan le resultará interesante al diseñador, puesto que le permitirá profundizar sobre la modelación con RdP. De esta forma se hace posible el establecimiento directo de descripciones «simples».

A partir de una clasificación de los diferentes tipos de simplificación de RdP, se abordará un estudio detallado de algunas técnicas de simplificación. El capítulo terminará con un ejemplo de descripción al que se aplicarán las diferentes técnicas presentadas.

3.2 CLASIFICACIÓN DE LOS TIPOS DE SIMPLIFICACIÓN

Los diversos tipos de simplificación que vamos a describir tienen por objeto la obtención de una RdP que «equivalga» funcionalmente a la RdP inicial, pero de una complejidad menor. Se clasifican en dos grupos bien diferenciados.

3.2.1 Simplificaciones estructurales

Se trata de simplificaciones independientes de la interpretación que se le asocie a la RdP. En ellas sólo se consideran en principio la *estructura* y el *marcado inicial* de la RdP.

El objetivo que se persigue es la eliminación de redundancias de tipo estructural, consistentes en la definición de falsas evoluciones paralelas. Los lugares que definen las falsas evoluciones se encuentran *implícitos*. En §3.3 presentaremos técnicas eficaces para su detección y eliminación. Estas técnicas de simplificación son específicas de las RdP.

3.2.2 Simplificaciones que tienen en cuenta la interpretación asociada a la RdP

Estas simplificaciones se realizan sobre RdP binarias e interpretadas, a las que se añade como información suplementaria las condiciones envolventes de sus lugares (§2.3.3, definición 2.16).

En este grupo de simplificaciones se pueden incluir diferentes técnicas desarrolladas para los grafos de estado, pero que no expondremos en este texto (diagramas de Girard, [GIRA 73], grafos de proceso [DACL 76], etc.).

Mediante el aporte de información, se pretende extraer funcionamientos no secuenciales en la descripción del sistema. Las dos técnicas de simplificación presentadas son las siguientes:

1) *Fusión de lugares*, que permite simplificar la parte secuencial, pero que, en contrapartida, suele incrementar la complejidad de la parte combinatoria.
2) *Supresión de conexiones entre lugares*, que hace posible la eliminación de los lugares que: (1) debido a la supresión de las mencionadas conexiones, no desempeñen ningún papel secuencial, y (2) no tengan asociada ninguna salida.

La técnica que permite estudiar la supresión de conexiones se denomina *método de los lugares fuente.*

3.3 MÉTODO DE LOS LUGARES IMPLÍCITOS

3.3.1 Concepto de lugar implícito y simplificación de la RdP

De forma intuitiva podemos decir que, en una RdP marcada, un lugar implícito es aquél que cumple las condiciones siguientes:

1) su marcado se puede evaluar en función del marcado de otros lugares;
2) jamás es el único lugar que impide la sensibilización de sus transiciones de salida.

Puesto que la evolución del marcado puede expresarse linealmente (§2.2.2), elegiremos una *función lineal* para calcular el marcado de un lugar implícito.

Antes de abordar una definición formal consideraremos un ejemplo introductorio. Observando la figura 3.1, se puede deducir que p_2 es un lugar implícito. En efecto, es fácil comprobar que:

1) el marcado de p_2, $M(p_2)$, se puede calcular en función del marcado de p_3 y de p_4:

$$M(p_2) = M(p_3) + M(p_4);$$

2) p_2 posee una única transición de salida, que es t_3. Ahora bien, $M(p_2) = M(p_3) + M(p_4) \Rightarrow M(p_2) \geqslant M(p_4)$ y, por consiguiente, no se puede obtener el marca-

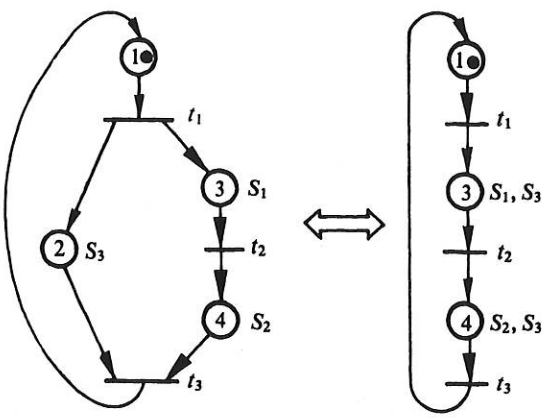


Figura 3.1. Eliminación de p_2 (lugar implícito que está «en paralelo» con p_3-p_4).

do $M(p_2) = 0$ y $M(p_4) = 1$. Este último marcado sería el único en el que p_2 fuese el único lugar que impide la sensibilización de t_3.

En conclusión, la eliminación de p_2 no altera la relación evento-acción definida si asociamos la salida S_3 a los lugares p_3 y p_4.

Podemos afirmar que si un lugar se encuentra en paralelo con una rama de la RdP, éste es implícito siempre que los lugares de la rama no contengan un número de marcas superior al que posee el lugar. *Un lugar implícito define una falsa evolución paralela.*

Formalicemos ahora el concepto de lugar implícito. Es interesante destacar que la definición y todos los tratamientos posteriores son también válidos para simplificar las RdPG.

Sea $|P| = n$.

Definición 3.1. Un *lugar* p_i está *implícito* en la red marcada $\langle \boldsymbol{R}, M_0 \rangle$ si para todo marcado alcanzable se verifica que:

1) $M(p_i) = \sum_{\substack{j=1 \\ i \neq j}}^{n} \lambda_j M(p_j) + \mu \quad \lambda_j \in \mathbb{Q}^+, \mu \in \mathbb{Q}$

2) $\forall p_j \in P - \{p_i\} \quad M(p_j) \geqslant \alpha(p_j, t_k) \Rightarrow M(p_i) \geqslant \alpha(p_i, t_k)$.

Todo conjunto de lugares $\Pi_i = \{p_j \mid \lambda_j > 0\}$ se denomina *conjunto de lugares implicantes de* p_i. □

El conjunto de lugares implicantes de p_2 (figura 3.1) es $\{p_3, p_4\}$.

La simplificación de las RdP por eliminación de los lugares implícitos se desarrollará en dos etapas:

1) la búsqueda de los lugares implícitos;
2) su eliminación, y asignación de sus salidas a los lugares implicantes (véase la figura 3.1).

En los apartados que siguen estudiaremos en detalle ambas etapas.

3.3.2 Búsqueda de los lugares implícitos

En este apartado presentaremos tres proposiciones. La primera establece una condición necesaria para que un lugar pueda estar implícito. Su interés radica en que la selección de lugares *potencialmente implícitos* se realiza con un algoritmo de complejidad lineal en función del número de lugares de la RdP. La segunda proposición permite determinar, en condiciones generales, si un lugar está implícito. La tercera proposición facilita la determinación de los lugares implícitos en un caso particular de gran interés práctico.

Condiciones necesarias para que un lugar pueda estar implícito (proposición 3.1). Para que p_i esté implícito es necesario que sus transiciones de entrada tengan más de un lugar de salida y sus transiciones de salida tengan más de un lugar de entrada.

Es decir, si p_i está implícito se cumplirá que:

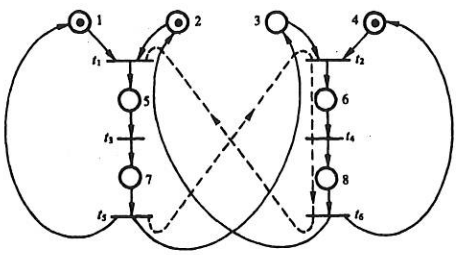
1) $\forall t_j \in {}^{\cdot}p_i \quad |t_j^{\cdot}| > 1.$
2) $\forall t_k \in p_i^{\cdot} \quad |{}^{\cdot}t_k| > 1.$ □

DEMOSTRACIÓN. Si $t_j \in {}^{\cdot}p_i$ y $|t_j^{\cdot}| = 1$, su disparo aumenta el marcado de p_i en $\beta(t_j, p_i)$ unidades sin que por ello se aumente el resto del marcado. Por lo tanto, no se puede evaluar el marcado de p_i en función del resto del marcado.

Análogamente, si $t_k \in p_i^{\cdot}$ y $|{}^{\cdot}t_k| = 1$, el disparo de t_k disminuye el marcado de p_i en $\alpha(p_i, t_k)$ unidades sin disminuir el resto del marcado. □

Esta proposición permite seleccionar un subconjunto de lugares de la RdP entre los que se encontrarán los lugares implícitos, si los hubiere. De esta manera se evita la aplicación indiscriminada a todos los lugares de la red del análisis que sigue.

EJEMPLO. La aplicación de la proposición 3.1 a la RdP de la figura 3.2 indica que los lugares $\{p_5, p_6, p_7, p_8\}$ no pueden estar implícitos. Los lugares *potencialmente implícitos* son $\{p_1, p_2, p_3, p_4\}$.



$$\tilde{C} = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{matrix} \\ \begin{bmatrix} -1 & 0 & 0 & 0 & +1 & 0 \\ -1 & 0 & 0 & 0 & 0 & +1 \\ 0 & -1 & 0 & 0 & +1 & 0 \\ 0 & -1 & 0 & 0 & 0 & +1 \\ +1 & 0 & -1 & 0 & 0 & 0 \\ 0 & +1 & 0 & -1 & 0 & 0 \\ 0 & 0 & +1 & 0 & -1 & 0 \\ 0 & 0 & 0 & +1 & 0 & -1 \end{bmatrix} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ p_7 \\ p_8 \end{matrix} \end{array}$$

Figura 3.2 p_1 y p_4 son implícitos (p_1 está «en paralelo» con $\{p_3, p_6, p_8, p_2\}$).

Para abordar la determinación de lugares implícitos conviene introducir las siguientes notaciones. Sea $\tilde{c}_{ij} = \beta(t_j, p_i) - \alpha(p_i, t_j)$. La matriz $\tilde{C} = [\tilde{c}_{ij}]_{n \times m}$ se denominará *matriz de flujo de marcas*. $\tilde{I}(p_i)$ es la i-ésima fila de $\tilde{C}_{n \times m}$. Si σ es aplicable a partir de M_0, $M_k = M_0 + \tilde{C} \cdot \bar{\sigma}$ (incluso si la red no es pura). Evidentemente, si la RdP es pura, $\tilde{C} = C$ y $\tilde{I}(p_i) = I(p_i)$; es decir, las matrices de flujo de marcas y de incidencia coinciden.

Determinación de lugares implícitos (proposición 3.2). Un conjunto de lugares Π_i implica al lugar p_i si se verifican las condiciones siguientes:

$$\left.\begin{array}{ll} C_1) & M_0(p_i) = \sum\limits_{p_j \in \Pi_i} \lambda_j M_0(p_j) + \mu \\ C_2) & \tilde{I}(p_i) = \sum\limits_{p_j \in \Pi_i} \lambda_j \tilde{I}(p_j) \\ C_3) & \forall t_k \in p_i^{\cdot} \quad \alpha(p_i, t_k) \leqslant \sum\limits_{p_j \in \Pi_i} \lambda_j \alpha(p_j, t_k) + \mu \end{array}\right\} \lambda_j \in \mathbb{Q}^+, \ \mu \in \mathbb{Q}. \qquad \square$$

DEMOSTRACIÓN. Las primera y segunda condiciones son necesarias y suficientes para que se cumpla la primera parte de la definición de lugar implícito. En efecto, la primera condición resulta de la particularización de la primera parte de la definición de lugar implícito para el marcado inicial. La segunda condición se deduce al escribir la ecuación que establece la evolución del marcado del lugar p_i (particularización para p_i de la ecuación $M = M_0 + \tilde{C} \cdot \bar{\sigma}$):

$$M(p_i) = M_0(p_i) + \tilde{I}(p_i) \cdot \bar{\sigma} = \sum_j \lambda_j M_0(p_j) + \mu + \tilde{I}(p_i) \cdot \bar{\sigma}.$$

Ahora bien, si p_i es implícito se podrá escribir:

$$M(p_i) = \sum_j \lambda_j M(p_j) + \mu = \sum_j \lambda_j [M_0(p_j) + \tilde{I}(p_j) \cdot \bar{\sigma}] + \mu.$$

Igualando las dos expresiones que definen $M(p_i)$ se obtiene:

$$\tilde{I}(p_i) \cdot \bar{\sigma} = \sum_j \lambda_j \tilde{I}(p_j) \cdot \bar{\sigma}.$$

Por último, como la identidad anterior es cierta para cualquier $\bar{\sigma}$, se concluye que

$$\tilde{I}(p_i) = \sum_j \lambda_j \tilde{I}(p_j).$$

A la vista del proceso de demostración, se puede observar que la condición 1 de la definición de lugar implícito se descompone directamente en una *condición inicial* C_1 y otra *condición estructural* C_2. Razonando en sentido inverso, estas condiciones se componen para dar la condición 1 de la definición 3.1.

La tercera condición de la proposición garantiza que p_i no sea nunca el único lugar que impida la sensibilización de una transición (segunda parte de la definición de lugar implícito). En efecto, si se cumple:

$$\begin{cases} \alpha(p_i, t_k) \leqslant \sum\limits_{p_j \in \Pi_i} \lambda_j \alpha(p_j, t_k) + \mu \quad \forall t_k \in p_i^{\cdot} \\ M(p_j) \geqslant \alpha(p_j, t_k) \qquad \text{(es decir, los } p_j \text{ sensibilizan } t_k\text{)}, \end{cases}$$

entonces se verifica la expresión:

$$\alpha(p_i, t_k) \leqslant \sum_{p_j \in \Pi_i} \lambda_j M(p_j) + \mu = M(p_i)$$

En conclusión, si $M(p_j) \geqslant \alpha(p_j, t_k)$, p_i sensibiliza t_k. $\square$

La proposición 3.2 comprende tres condiciones que permiten garantizar el que un lugar sea implícito. De su observación se deduce que todo proceso de determinación de lugares implícitos debe comenzar por el cálculo de un conjunto de lugares y de

λ_j que verifique la condición C_2. Acto seguido, utilizando la condición C_1, se determina el valor de μ y, por último, se comprueba si se cumple la condición C_3.

EJEMPLO. Para la RdP de la figura 3.2 tenemos:

$$C_2)\quad I(p_1) = I(p_3) + I(p_6) + I(p_8) + I(p_2)$$
$$C_1)\quad M_0(p_1) = M_0(p_3) + M_0(p_6) + M_0(p_8) + M_0(p_2) = 1 \Rightarrow \mu = 0$$
$$C_3)\quad \alpha(p_1, t_1) = \cancel{\alpha(p_3, t_1)} + \cancel{\alpha(p_6, t_1)} + \cancel{\alpha(p_8, t_1)} + \alpha(p_2, t_1) = 1,$$

luego p_1 es un lugar implícito y $\Pi_1 = \{p_3, p_6, p_8, p_2\}$ es un conjunto de lugares implicantes.

A continuación demostraremos que las tres condiciones que permiten determinar lugares implícitos (proposición 3.2) se pueden reducir a las dos primeras si el lugar p_i no es simultáneamente lugar de entrada y de salida de una misma transición ($^{\cdot}p_i \cap p_i^{\cdot} = \emptyset$) y $\mu \geqslant 0$.

Método simplificado para la determinación de lugares implicitos (proposición 3.3). Si $^{\cdot}p_i \cap p_i^{\cdot} = \emptyset$ y $\mu \geqslant 0$, entonces la tercera condición de la proposición 3.2 es redundante con la segunda condición de la misma proposición, y bastará con verificar las dos primeras. □

DEMOSTRACIÓN. Si $^{\cdot}p_i \cap p_i^{\cdot} = \emptyset$, entonces $\forall t_k \in p_i^{\cdot}$ se tendrá $\beta(t_k, p_i) = 0$. Esto implica que $\tilde{c}_{ik} = -\alpha(p_i, t_k)$, y por la segunda condición de la proposición 3.2 se podrá escribir:

$$-\alpha(p_i, t_k) = \sum_{p_j \in \Pi_i} \lambda_j [\beta(t_k, p_j) - \alpha(p_j, t_k)] \Rightarrow$$
$$\Rightarrow \alpha(p_i, t_k) = \sum_{p_j \in \Pi_i} \lambda_j [\alpha(p_j, t_k) - \beta(t_k, p_j)] \leqslant \sum_{p_j \in \Pi_i} \lambda_j \alpha(p_j, t_k)$$

Por consiguiente, si $\mu \geqslant 0$ se cumplirá que (condición 3 de la proposición 3.2):

$$\alpha(p_i, t_k) \leqslant \sum_{p_j \in \Pi_i} \lambda_j \alpha(p_j, t_k) + \mu. \qquad \square$$

El resultado de la proposición 3.3 permite garantizar que p_1 (figura 3.2) es un lugar implícito, sin necesidad de comprobar la verificación de la tercera condición de la proposición 3.2.

El problema fundamental que queda por resolver es la determinación del conjunto de lugares implicantes y de los λ_j correspondientes. Para ello vamos a considerar el *lugar complementario* $\bar{p}_i$. Siguiendo la definición 2.22 se tendrá $\tilde{I}(p_i) = -\tilde{I}(\bar{p}_i)$, de donde:

$$\tilde{I}(p_i) = \sum_j \lambda_j \tilde{I}(p_j) \Rightarrow \tilde{I}(\bar{p}_i) + \sum_j \lambda_j \tilde{I}(p_j) = 0.$$

Es decir, si consideramos $\bar{p}_i$ en vez de p_i, hemos de encontrar un conjunto de lugares, $\{p_j\}$, y de coeficientes no negativos, $\{\lambda_j\}$, que cumplan la ecuación anterior†.

† Volveremos a este problema en el capítulo 4 (§4.7.1) al estudiar las *componentes conservativas* de una RdP.

La matriz en la que sustituiremos $\tilde{l}(p_i)$ por $\tilde{l}(\bar{p}_i)$ será representada por $\bar{C}_{pi}$.

Para determinar si existe una o varias sumas ponderadas con coeficientes positivos de las filas asociadas a $\bar{p}_i$ y $\{p_j\}$ que sean nulas, aplicaremos el algoritmo que presentaremos a continuación†. Si no existiera ninguna solución, de acuerdo con la proposición 3.2, el lugar p_i no podría estar implícito.

El algoritmo que se considera calcula todas las soluciones no negativas elementales del sistema de ecuaciones $D_\varphi \cdot \bar{C}_{p_i} = 0$ en que aparece $\tilde{l}(\bar{p}_i)$; es decir, todas las soluciones de la forma $D_{\varphi i} \cdot \tilde{l}(\bar{p}_i) + \sum_j D_{\varphi j} \cdot \tilde{l}(p_j) = 0$, donde $\forall j \neq i \quad D_{\varphi j} \geqslant 0$ y $D_{\varphi i} > 0$. A partir de las soluciones elementales se puede obtener cualquiera otra no negativa mediante sumas, exclusivamente. Desde un punto de vista operativo se procede de forma iterativa anulando columnas de una cierta matriz. La anulación de una columna se realiza sumando pares de filas y eliminando otras filas. La ejecución de la iteración termina cuando se han anulado todas las columnas de las combinaciones en que aparece como sumando la fila asociada a $\bar{p}_i$.

Sean $[D^j \vdots A^j]$ e $[I_n]$ la matriz obtenida a partir de $[D^0 \vdots A^0]$ después de j iteraciones y la matriz unitaria de dimensión n, respectivamente.

ALGORITMO PARA DETERMINAR LOS CONJUNTOS DE LUGARES POTENCIALMENTE IMPLICANTES DE p_i

(1) $j := 0$; $[D^j \vdots A^j] := [I_n \vdots \bar{C}_{pi}]$;

(2) **Mientras que** exista $D^j_{\varphi i} \neq 0$ y $A^j_{\varphi k} \neq 0$

hacer 2.1 *Añadir* a la matriz $[D^j \vdots A^j]$ todas las filas que resulten como combinación lineal positiva de pares de filas de $[D^j \vdots A^j]$ y que anulen la k-ésima columna de A^j.

2.2 *Eliminar* las filas en las que la k-ésima columna de A^j sea no nula.

2.3 $j := j + 1$;

(3) Las filas φ de la matriz D final en las que $D_{\varphi i} \neq 0$, representan soluciones de la forma buscada $(D_{\varphi i} \cdot \tilde{l}(\bar{p}_i) + \sum_j D_{\varphi j} \cdot \tilde{l}(p_j) = 0)$.

Nota. El papel que desempeña la fila φ de la matriz D^j es el de recordar la combinación de filas de $\bar{C}_{p_i}$ que representa la φ-ésima fila de A^j. Por consiguiente, en una aplicación manual se puede sustituir la matriz D^j por la expresión directa de la suma de las filas originales de $A^0 \equiv \bar{C}_{p_i}$. De esta forma se procede en el ejemplo siguiente.

EJEMPLO. Para ilustrar el algoritmo anterior, vamos a determinar si el lugar potencialmente implícito p_1 (figura 3.2) puede ser implícito. En caso afirmativo determinaremos los conjuntos de lugares implicantes.

† Una justificación formal del algoritmo se deduce directamente de lo expuesto en §4.7.2.2*a*.

Paso 1. La matriz $[D^0 \vdots A^0]$ es $[I_8 \vdots \bar{C}_{p_1}]$:

							k	=	1	2	3	4	5	6	n.º de fila
+1	0	0	0	0	0	0	0		+1	0	0	0	−1	0	(1)
0	+1	0	0	0	0	0	0		−1	0	0	0	0	+1	(2)
0	0	+1	0	0	0	0	0		0	−1	0	0	+1	0	(3)
0	0	0	+1	0	0	0	0		0	−1	0	0	0	+1	(4)
0	0	0	0	+1	0	0	0		+1	0	−1	0	0	0	(5)
0	0	0	0	0	+1	0	0		0	+1	0	−1	0	0	(6)
0	0	0	0	0	0	+1	0		0	0	+1	0	−1	0	(7)
0	0	0	0	0	0	0	+1		0	0	0	+1	0	−1	(8)

Paso 2.

• *Iteración* n.º 1: φ (único) = 1. $k = 1$ o $k = 5$. Sea, por ejemplo, $k = 1$.

(2.1) Las filas que se deben añadir son:

+1	+1	0	0	0	0	0	0		0	0	0	0	−1	+1	(9) ≡ (1) + (2)
0	+1	0	0	+1	0	0	0		0	0	−1	0	0	+1	(10) ≡ (2) + (5)

(2.2) Las filas que han de eliminarse son la (1), la (2) y la (5) (puesto que $A^0_{11} \neq 0$, $A^0_{21} \neq 0$ y $A^0_{51} \neq 0$).

• *Iteración* n.º 2: φ (único) = 9. $k = 5$ o $k = 6$. Sea, por ejemplo, $k = 5$.

(2.1) Las filas que se deben añadir son:

0	0	+1	0	0	0	+1	0		0	−1	+1	0	0	0	(11) ≡ (3) + (7)
+1	+1	+1	0	0	0	0	0		0	−1	0	0	0	+1	(12) ≡ (3) + (9) ≡
															≡ (3) + (1) + (2).

(2.2) Las filas que se deben eliminar son la (3), la (7) y la (9) (puesto que $A^1_{35} \neq 0$, $A^1_{75} \neq 0$ y $A^1_{95} \neq 0$). La matriz que se tiene en estos momentos es: $[D^2 \vdots A^2]$:

0	0	0	+1	0	0	0	0		0	−1	0	0	0	+1	(4)
0	0	0	0	0	+1	0	0		0	+1	0	−1	0	0	(6)
0	0	0	0	0	0	0	+1		0	0	0	+1	0	−1	(8)
0	+1	0	0	+1	0	0	0		0	0	−1	0	0	+1	(10) ≡ (2) + (5)
0	0	+1	0	0	0	+1	0		0	−1	+1	0	0	0	(11) ≡ (3) + (7)
+1	+1	+1	0	0	0	0	0		0	−1	0	0	0	+1	(12) ≡ (1) + (2) + (3)

• *Iteración* n.º 3: φ (único) = 12. $k = 2$ o $k = 6$. Sea, por ejemplo, $k = 2$.

(2.1) Las filas que se deben añadir son:

0	0	0	+1	0	+1	0	0		0	0	0	−1	0	+1	(13) ≡ (4) + (6)
0	0	+1	0	0	+1	+1	0		0	0	+1	−1	0	0	(14) ≡ (6) + (11) ≡ (6) + (3) + (7)
+1	+1	+1	0	0	+1	0	0		0	0	0	−1	0	+1	(15) ≡ (6) + (12) ≡ (6) + (3) + (9)
															≡ (6) + (1) + (2) + (3)

(2.2) Las filas que se deben eliminar son la (4), la (6), la (11) y la (12).

• *Iteración* n.º 4: φ (único) = 15. $k = 4$ o $k = 6$. Sea, por ejemplo, $k = 4$.

(2.1) Las filas que se deben añadir son:

0	0	0	+1	0	+1	0	+1		0	0	0	0	0	0	(16) ≡ (8) + (13) ≡ (8) + (4) + (6)
0	0	+1	0	0	+1	+1	+1		0	0	+1	0	0	−1	(17) ≡ (8) + (14) ≡ (8) + (3) + (6) + (7)
+1	+1	+1	0	0	+1	0	+1		0	0	0	0	0	0	(18) ≡ (8) + (15) ≡ (8) + (6) + (3) + (9)
															≡ (8) + (6) + (1) + (2) + (3)

(2.2) Las filas que se deben eliminar son la (8), la (13), la (14) y la (15). La matriz que se tiene es $[D^4 \vdots A^4]$:

$$\left[\begin{array}{cccccccc|cccccc|l}
0 & +1 & 0 & 0 & +1 & 0 & 0 & 0 & 0 & 0 & +1 & 0 & 0 & =1 & (10) \equiv (2) + (5) \\
0 & 0 & 0 & +1 & 0 & +1 & 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 & (16) \equiv (4) + (6) + (8) \\
0 & 0 & +1 & 0 & 0 & +1 & +1 & +1 & 0 & 0 & +1 & 0 & 0 & -1 & (17) \equiv (3) + (6) + (7) + (8) \\
+1 & +1 & +1 & 0 & 0 & +1 & 0 & +1 & 0 & 0 & 0 & 0 & 0 & 0 & (18) \equiv (1) + (2) + (3) + (6) + (8)
\end{array}\right]$$

• *Iteración* n.º 5: φ (único) = 18. Como no existe k tal que $A^4_{18k} \neq 0$, la iteración ha terminado (no se reejecuta).

Paso 3. La fila etiquetada 18 en D^4 (4.ª en la última matriz, $[D^4 \vdots A^4]$), indica que $\bar{I}(\bar{p}_1) + \bar{I}(p_2) + \bar{I}(p_3) + \bar{I}(p_6) + \bar{I}(p_8) = 0$.

Según la proposición 3.1, inicialmente sólo $\{p_1, p_2, p_3, p_4\}$ (figura 3.2) pueden estar implícitos. Si por estar implícito se elimina p_1, al reaplicar la proposición 3.1 sólo aparece p_4 como potencialmente implícito. Un estudio detallado (ejercicio que se propone al lector) indica que p_4 también está implícito.

EJERCICIO. Demuéstrese la proposición 3.1 a partir de la proposición 3.2. (*Sugerencia*: aplíquese el algoritmo anterior y compruébese si se verifica la condición estructural.)

EJERCICIO. Compruébense los resultados anunciados en la figura 3.3. ¿Se puede simplificar aún más alguna de las RdP marcadas? (*Sugerencia*: determínense de antemano los lugares potencialmente implícitos.)

EJERCICIO. Determínense los lugares implícitos (figura 3.3) si se realizan los cambios siguientes en los marcados iniciales:

a) añadir una marca a cada lugar p_2,
b) añadir una marca a cada lugar p_1 y p_2.

Antes de abordar el estudio de la asignación de las salidas asociadas a los lugares implícitos, consideremos un caso particular de especial interés. Se trata del estudio de un lugar cuya fila $\bar{I}(p_i)$ sea inicialmente nula. En ese caso p_i no tiene ningún lugar implicante, $|\Pi_i| = 0$. Diremos que p_i es un *lugar identidad*.

Las condiciones 1 y 3 de la proposición 3.2 pueden escribirse del siguiente modo:

$$\left.\begin{array}{ll} C'_1) & M_0(p_i) = \mu \\ C'_3) & \alpha(p_i, t) \leqslant \mu \end{array}\right\} \Rightarrow M_0(p_i) \geqslant \alpha(p_i, t) \quad \forall t \in p_i.$$

De aquí se deduce que la eliminación de p_i sólo es posible si inicialmente contiene suficientes marcas para disparar cualquiera de sus transiciones de salida. El lugar p (figura 3.4) sólo puede ser eliminado si inicialmente contiene un número de marcas superior o igual a máx(α_1, α_2).

3.3.3 Asignación de las acciones asociadas a un lugar implícito

En el apartado anterior hemos estudiado la búsqueda de los lugares implícitos. Si éstos no poseen ninguna acción (salida) asociada, se pueden eliminar sin mayor reparo. Ahora bien, si un lugar implícito tiene asociadas acciones, hemos de asignarlas a otros lugares de forma tal que la coordinación evento-acción se mantenga.

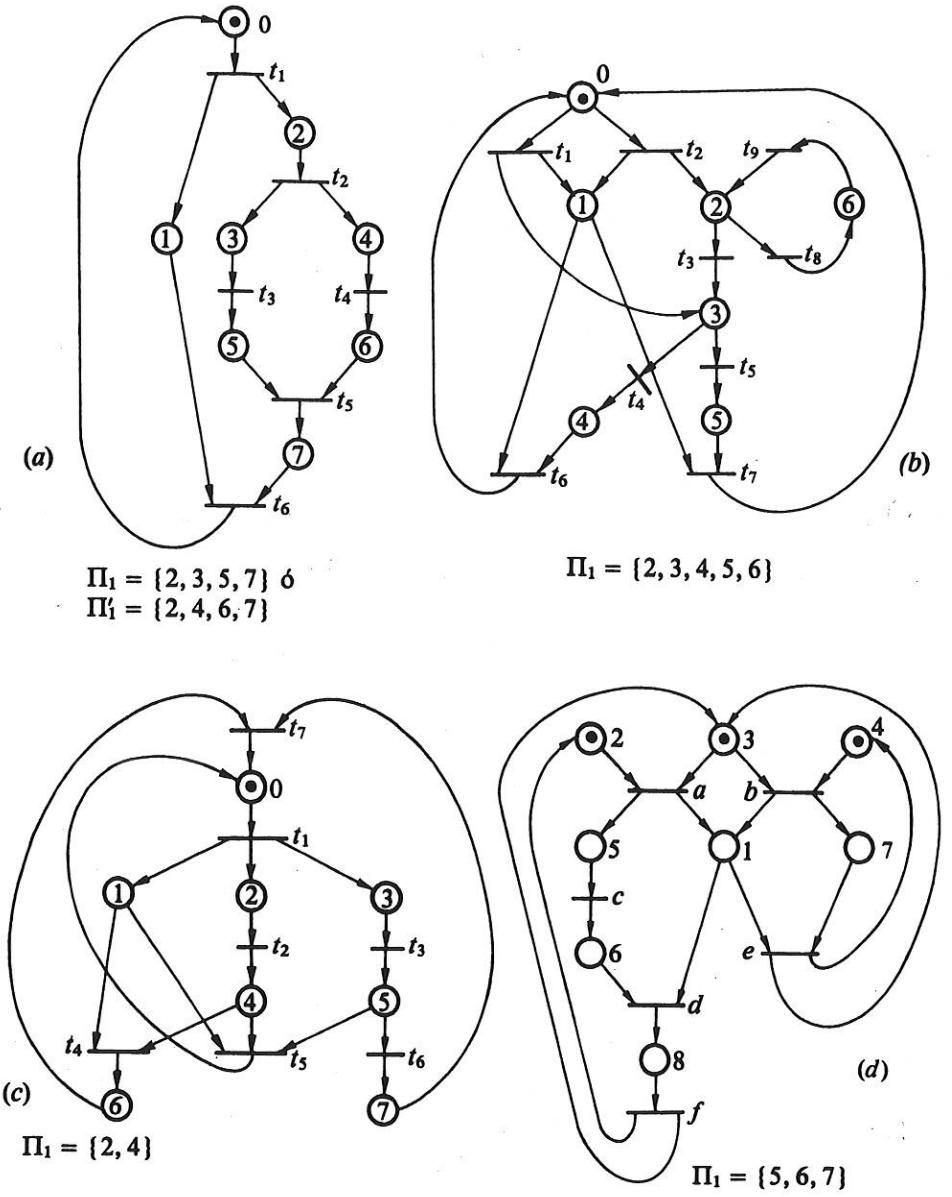


Figura 3.3. Ejemplos en los que p_1 está siempre implícito.

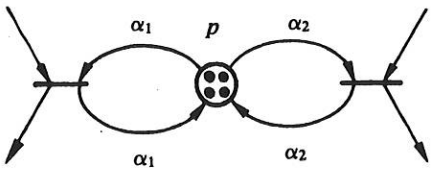


Figura 3.4. Si $M_0(p) \geqslant \max(\alpha_1, \alpha_2)$, p es un lugar identidad en la subRdPG.

Si un lugar p_i está implícito, sabemos que:

$$M(p_i) = \sum_j \lambda_j M(p_j) + \mu,$$

es decir, se puede calcular el marcado de p_i como suma de una serie de términos. En estas condiciones generales, resulta evidente que se ha avanzado poco al eliminar un lugar implícito, puesto que para la realización de la red habrá que introducir la capacidad de sumar y restar. Si considerásemos una realización cableada† se eliminaría un dispositivo lógico (memoria) a cambio de un dispositivo de cálculo aritmético.

Supongamos que p_i sólo puede estar o marcado o no marcado (p_i es *binario*) y $\mu = 0$. En este caso, siempre que algún lugar implicante esté marcado, también lo estará p_i y, por consiguiente, será posible sustituir la *suma* aritmética por la *unión lógica*. En conclusión, si p_i es *binario* y $\mu = 0$, asignaremos el conjunto de acciones asociadas al lugar implícito a cada uno de los lugares implicantes.

El caso anterior es el de tratamiento más sencillo y, afortunadamente, el que se presenta con mayor frecuencia. Cuando p_i es binario y $\mu < 0$ o $\mu > 1$, es posible, en cada caso particular, expresar el marcado de p_i como *función lógica* del marcado de sus lugares implicantes.

En todos los ejemplos tratados hasta ahora (figuras 3.1, 3.2 y 3.3) hemos encontrado RdP binarias, y $\mu = 0$. En estos modelos el problema de la asignación de acciones (salidas) no plantea dificultad alguna. En la RdP de la figura 3.5, el lugar p_9 está implícito, pero $\mu = -1$. De este modo, s_9 no se puede calcular como *unión* de un conjunto de marcados, pese a que p_9 es *binario*. No obstante, podemos transformar la expresión aritmética (suma) en una expresión lógica. En efecto, es fácil comprobar‡ que, por ejemplo, la siguiente expresión *lógica* define el marcado de p_9:

$$M(p_9) = M(p_7)[M(p_4) + M(p_5)].$$

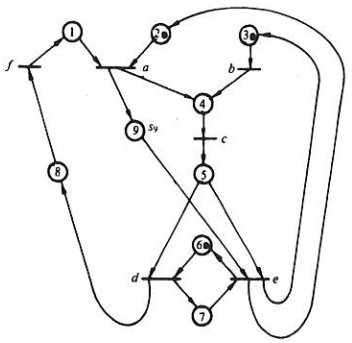
EJERCICIO. Compruébese que en la RdP de la figura 3.5:

1) después de eliminar p_9, es posible eliminar p_2;
2) en vez de eliminar $\{p_2, p_9\}$ se puede eliminar $\{p_2, p_7\}$, obteniéndose siempre $\mu = 0$.

---

† Véase el capítulo 6.

‡ Para ello se puede obtener, por ejemplo, el grafo reducido (GR) equivalente (§1.5.4) y analizar los marcados alcanzables.



1) $\tilde{I}(p_9) = \tilde{I}(p_3) + \tilde{I}(p_4) + \tilde{I}(p_5) + \tilde{I}(p_7)$

2) $M_0(p_9) = \underbrace{M_0(p_3)}_{1} + \underbrace{M_0(p_4) + M_0(p_5) + M_0(p_7)}_{0} + \mu = 0$

luego $\mu = -1$

3) $\underbrace{\alpha(p_6, e)}_{1} = \underbrace{\alpha(p_3, e)}_{0} + \underbrace{\alpha(p_4, e)}_{0} + \underbrace{\alpha(p_5, e)}_{1} + \underbrace{\alpha(p_7, e)}_{1} - 1 = 1$

Figura 3.5. RdP marcada y conforme (binaria y viva). El lugar p_9 está implícito, pero $\mu \neq 0$ ($\mu = -1$).

3.4 MÉTODO DE FUSIÓN DE LUGARES

En este método de simplificación aprovecharemos la interpretación asociada a la RdP para detectar lugares *equivalentes* y lugares *compatibles*; posteriormente procederemos a fusionar dichos lugares. Hemos de resaltar que este método de fusión no sólo se basa en propiedades estructurales de la red, sino que se encuentra estrechamente ligado a la interpretación a ella asociada.

Para evitar cambios importantes en la estructura de la red que traigan como consecuencia una pérdida ostensible de la significación del modelo, vamos a *limitar voluntariamente las posibles fusiones entre lugares*. Dada una red, restringiremos las fusiones entre lugares a aquéllas que sean posibles dentro de cada subRdP obtenida al eliminar los nudos Y (transiciones con más de un lugar de entrada o/y de salida) en la red original. Ello implica en todas las subRdP que cada lugar será el único lugar de entrada [salida] de sus transiciones de salida [entrada] que no hayan sido eliminadas.

3.4.1 Lugares equivalentes o directamente fusionables

Definición 3.2. Dos *lugares* p_i y p_j son *directamente fusionables* o *equivalentes*, si:

1) Coinciden los conjuntos de acciones a ellos asociados.
2) Para cada transición de salida de p_i existe otra de p_j que tiene el mismo lugar de salida y a ambas se les ha asociado idéntico par evento-condición externa. □

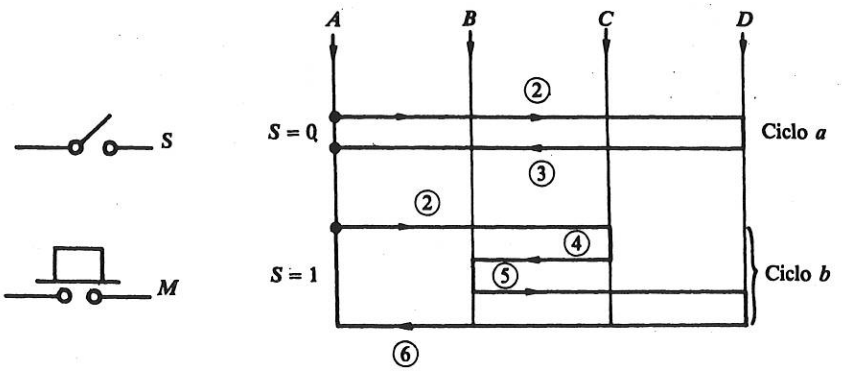


Figura 3.6. Diagrama de evoluciones previstas para el sistema descrito en el ejemplo.

En efecto, en tal caso no es necesario distinguir cuándo una marca corresponde a p_i y cuándo a p_j. Por desgracia, este tipo de fusión suele ser poco frecuente.

Introduzcamos ahora un ejemplo que nos servirá para ilustrar este tipo de fusión, así como para ilustrar el que presentaremos en el apartado siguiente.

EJEMPLO. Sea un órgano móvil, con un movimiento lineal, que puede accionar cuatro contactos, A, B, C y D, como se muestra en el esquema de la figura 3.6.

El contacto A se cierra ($A \leftarrow 1$) cuando el órgano móvil lo sobrepasa hacia la izquierda. El mismo contacto se abre ($A \leftarrow 0$) cuando el órgano móvil lo sobrepasa hacia la derecha. La definición del comportamiento de los contactos B, C y D es la inversa. Sus aperturas y cierres se llevan a cabo cuando el órgano móvil los sobrepasa hacia la izquierda y la derecha, respectivamente.

Si el órgano móvil se encuentra sobre A estando M pulsado, comienza un ciclo a o un ciclo b, según que el selector S esté abierto o cerrado. Supongamos que S toma su valor antes de iniciarse un ciclo y lo conserva hasta que el ciclo termina.

La figura 3.7 presenta una posible descripción del automatismo.

Sobre la descripción del automatismo (figura 3.7*a*) podemos realizar una primera fusión entre p_3 y p_6. En efecto, ambos lugares cumplen las condiciones de la definición 3.2:

1) la acción asociada a ellos es la misma: i;
2) la etiqueta asociada a las transiciones $p_3 \rightarrow p_1$ y $p_6 \rightarrow p_1$ es la misma: A.

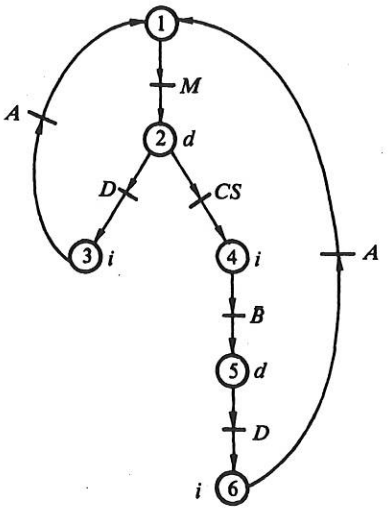
En conclusión, podemos fusionar p_3 y p_6 *aunque no* seamos capaces de determinar si es uno u otro el lugar que está marcado en la descripción inicial cuando el lugar de fusión p_{36} esté marcado.

Por razones didácticas no realizaremos esta fusión inmediata. En el próximo apartado presentaremos otra condición suficiente para que dos lugares cualesquiera puedan ser fusionados.

3.4.2 Lugares compatibles

3.4.2.1 *Introducción*

Para que sea posible la fusión de dos o más lugares en una red binaria es suficiente



(*a*) Grafo reducido

variables de entrada

lugares	A	B	C	D	M	S
p_1	A	$\bar{B}$	$\bar{C}$	$\bar{D}$	—	—
p_2	—	—	—	$\bar{D}$	—	—
p_3	$\bar{A}$	—	—	—	—	$\bar{S}$
p_4	$\bar{A}$	B	—	$\bar{D}$	—	S
p_5	$\bar{A}$	—	—	$\bar{D}$	—	S
p_6	$\bar{A}$	—	—	—	—	S

(*b*) Estado de las variables de entrada

Figura 3.7 Descripción del automatismo que posee los dos ciclos de trabajo (figura 3.6).

con que seamos capaces de determinar, cuando el lugar de fusión está marcado, cuál de los lugares fusionados es el marcado en la descripción inicial. Para ello se requiere *añadir* una información complementaria a la que posee la RdP inicial interpretada. Recuérdese (§1.4.2) que un modelo construido con una RdP interpretada contiene una información mínima.

El aporte complementario de información lo realizaremos especificando, para cada lugar, el estado de las variables de entrada cuando éste se encuentre marcado. Dicho de otra forma, determinaremos las *condiciones envolventes* (§2.3.3) de los distintos lugares de la RdP.

Volviendo al sistema de la figura 3.6, detallemos cómo se aporta la información complementaria para simplificar la descripción de la figura 3.7*a*.

En la *tabla de estado de las variables de entrada* (figura 3.7*b*) asociamos una fila a cada lugar y una columna a cada variable de entrada. En la casilla definida por el lugar p_i y la variable de entrada e_j colocamos e_j o $\bar{e}_j$ dependiendo de si, cuando p_i está marcado, la variable e_j sólo toma el valor «1» o el «0», respectivamente. Si e_j puede tomar los dos valores cuando p_j está marcado, en la casilla correspondiente colocamos «—». Así es fácil observar que, cuando p_5 esté marcado, los contactos A y D estarán abiertos y el selector S estará cerrado.

Para rellenar la tabla de estado de las variables de entrada se puede razonar también a partir de las columnas. Por ejemplo, el pulsador M puede ser accionado intempestivamente en cualquier momento y, por lo tanto, se colocará «—» en toda la columna.

Partiendo de la tabla de estado de las variables de entrada, se pueden escribir inmediatamente las condiciones envolventes asociadas a los lugares, v_i (§2.3.3).

Este método de simplificación por fusión de lugares implica una disminución de la parte secuencial de la descripción (puesto que al realizar las fusiones existirán menos lugares), pero incrementa la complejidad de la parte combinatoria (puesto que se complicarán las funciones que definen los eventos, las condiciones externas y las salidas).

Desde un punto de vista operativo, se pueden distinguir dos fases en la aplicación de este método de simplificación:

1) estudio de condiciones suficientes para efectuar la fusión propiamente dicha;
2) cálculo de los eventos † y de las condiciones que permitan asegurar la coordinación evento-acción de la descripción inicial.

3.4.2.2 *Compatibilidad y conjuntos de lugares fusionables*

Definición 3.3. Se dice que *dos lugares distintos son compatibles* si la intersección de sus condiciones envolventes es nula. Es decir, p_i, p_j son compatibles si $v_i v_j = 0$. Por otra parte, todo lugar es compatible consigo mismo. □

Esto significa que, si el lugar obtenido tras la fusión de dos lugares compatibles se encuentra marcado, a partir del conocimiento del estado de las variables de entrada se puede discernir, sobre la descripción simplificada, cuál de los lugares de la descripción primitiva estará marcado.

Es fácil verificar que la «fusionabilidad no inmediata» es una relación de compatibilidad, puesto que la transitividad no se asegura. En efecto, en el caso considerado (figura 3.7*b*) se puede observar que $v_1 v_4 = 0$ y $v_1 v_5 = 0$, pero $v_4 v_5 \neq 0$.

En conclusión, nos encontramos frente a un estudio de compatibilidad, el cual no depende de las acciones asociadas a los lugares ni de las transiciones de salida de éstos.

El problema de máxima simplificación se reduce a obtener la (una) descripción que posea el mínimo número de lugares. Su tratamiento se puede realizar a partir de la determinación de los conjuntos máximos fusionables, o sea, de los *compatibles máximos.* ‡

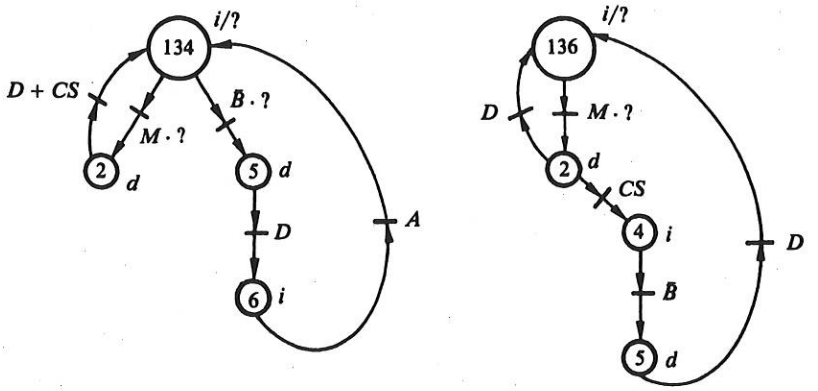
Para facilitar la determinación de los compatibles máximos vamos a introducir una representación tabular de la relación de compatibilidad. Ésta se denomina *tabla de pares compatibles.*

Una tabla de pares compatibles especifica, para cada par de elementos (lugares), si son o no compatibles. Si la relación está definida sobre un conjunto de n elementos, y dado que la compatibilidad es una relación reflexiva y simétrica, la tabla poseerá sólo $n(n-1)/2$ casillas (tabla 3.1). La casilla ij contiene la intersección de las condiciones envolventes de los lugares correspondientes, $v_i v_j$.

---

† Las definiciones 2.13 y 2.14 precisan los conceptos de condición *externa* y de *evento.* Para simplificar la terminología, en lo sucesivo hablaremos de «evento» en un sentido amplio de la palabra, refiriéndonos al par (o pares) evento-condición asociado a una transición.

‡ Sea P un conjunto (de lugares) sobre el que se define una *relación de compatibilidad.* Una *clase de compatibilidad* es un subconjunto de P tal que todos sus elementos son compatibles dos a dos. Se llama *compatible máximo* a una clase de compatibilidad que no está contenida en ninguna otra. En [DACL 76] o [TORN 72] se encontrará un método para la obtención de los compatibles máximos basado en la construcción de los incompatibles máximos.



(*a*) Fusión de {1, 3, 4} (*b*) Fusión de {1, 3, 6}

Figura 3.8 Grafos reducidos simplificados.

$\nu_1 = A\bar{B}\bar{C}\bar{D}$						
$\nu_2 = \bar{D}$	2	~~$A\bar{B}\bar{C}\bar{D}$~~				
$\nu_3 = \bar{A}\bar{S}$	3	0	~~$\bar{A}\bar{D}\bar{S}$~~			
$\nu_4 = \bar{A}B\bar{D}S$	4	0	~~$\bar{A}B\bar{D}S$~~	0		
$\nu_5 = \bar{A}\bar{D}S$	5	0	~~$\bar{A}\bar{D}S$~~	0	~~$\bar{A}B\bar{D}S$~~	
$\nu_6 = \bar{A}S$	6	0	~~$\bar{A}\bar{D}S$~~	0	~~$\bar{A}B\bar{D}S$~~	~~$\bar{A}\bar{D}S$~~
		1	2	3	4	5

Nota.
Si $\nu_i\nu_j = 0$ p_i y p_j son incompatibles

Tabla 3.1 Condiciones envolventes y tabla de pares compatibles (fusionables).

Sea $\mathcal{C}_i$ el subconjunto de elementos j (lugares p_j) definido por las casillas nulas de la columna i (subconjunto de lugares compatibles con p_i). En la tabla 3.1 se tiene $\mathcal{C}_1 = \{3, 4, 5, 6\}$, $\mathcal{C}_3 = \{4, 5, 6\}$ y $\mathcal{C}_2 = \mathcal{C}_4 = \mathcal{C}_5 = \emptyset$. Por otro lado, sea k el mayor índice de elemento (lugar) cuya columna en la tabla contenga al menos un par compatible ($k = 3$ en la tabla 3.1 puesto que $\mathcal{C}_5 = \mathcal{C}_4 = \emptyset$ y $\mathcal{C}_3 \neq \emptyset$).

ALGORITMO PARA LA OBTENCIÓN DE LOS COMPATIBLES MÁXIMOS

(1) *Crear* una lista de compatibles ($\mathcal{L}$) con los pares de elementos compatibles definidos por la columna k (la que está más a la derecha con al menos un par compatible).

(2) **Para** $i := k - 1$ **hasta** $i = 1$
Si $\mathcal{C}_i \neq \emptyset$ **entonces**
2.1 Realizar la intersección entre $\mathcal{C}_i$ y los elementos de $\mathcal{L}$.
2.2 *Añadir* a $\mathcal{L}$ los compatibles formados por el resultado de las intersecciones no nulas más el elemento i, así como los pares de compatibles definidos por la columna i-ésima.
2.3 *Suprimir* de $\mathcal{L}$ los conjuntos contenidos en otros.
(3) *Añadir* a $\mathcal{L}$ todos los elementos del conjunto de partida que no están contenidos en ningún elemento de $\mathcal{L}$.

Al aplicar el algoritmo anterior, cuya justificación dejamos al lector como ejercicio, la lista $\mathcal{L}$ final contiene todos los compatibles máximos.

EJEMPLO. Aplicando el algoritmo a la tabla 3.1, se tienen los pasos siguientes:

Paso 1: $k = 3$, de donde $\mathcal{L} := \{\{3,4\}, \{3,5\}, \{3,6\}\}$
Paso 2 (1.ª aplicación): $\mathcal{C}_2 = \emptyset$, luego no se hace nada.
Paso 2 (2.ª aplicación): $\mathcal{C}_1 = \{3,4,5,6\}$.
2.1 $\mathcal{L} \cap \mathcal{C}_1 := \{\{3,4\}, \{3,5\}, \{3,6\}\}$.
2.2 $\mathcal{L} := \mathcal{L} \cup \{\{1,3\}, \ldots, \{1,6\}, \{1,3,4\},\}, \{1,3,5\}, \{1,3,6\}\}$.
2.3 $\mathcal{L} := \{\{1,3,4\}, \{1,3,5\}, \{1,3,6\}\}$.
Paso 3: $\mathcal{L} := \{\{1,3,4\}, \{1,3,5\}, \{1,3,6\}, \{2\}\}$.

Luego los compatibles máximos son $\{1,3,4\}$, $\{1,3,5\}$, $\{1,3,6\}$ y $\{2\}$. Vemos, por tanto, que no es posible fusionar p_2 con ningún otro lugar.

De acuerdo con lo anterior, por ejemplo, el subconjunto de lugares $\{p_1, p_3, p_5\}$ es fusionable en uno solo, puesto que los marcados de:

— p_1 y p_3 se pueden distinguir gracias a la variable de entrada A,
— p_1 y p_5 se pueden distinguir gracias a la variable de entrada A,
— p_3 y p_5 se pueden distinguir gracias a la variable de entrada S.

A la vista de los conjuntos compatibles, el diseñador podrá elegir las fusiones que más le interesen en función de los criterios por él seleccionados (sentido físico de la descripción simplificada, seguridad, etc.).

Para concluir el estudio de los conjuntos de lugares fusionables, consideremos el caso en el que sólo se deseen fusionar los lugares cuyas acciones sean idénticas. De esta forma, el diseño conduce a una máquina de MOORE. De acuerdo con la figura 3.7*a* tenemos $s_2 = s_5 \neq s_3 = s_4 = s_6 \neq s_1$ y $s_1 \neq s_2$, y, por lo tanto, los conjuntos máximos fusionables compatibles con las salidas son $\{3,4\}$ y $\{3,6\}$ (véase tabla 3.2).

3.4.2.3 *Determinación de los eventos y las condiciones sobre las salidas asociadas al lugar de fusión*

Una vez fusionados varios lugares compatibles en uno solo, se plantea el problema de determinar los eventos y las condiciones de las salidas que permiten mantener la descripción funcional de partida. Volviendo al ejemplo que estamos tratando, hemos obtenido los siguientes conjuntos fusionables:

$\{1,3,4\}$ con 2 salidas que distinguir ($s_3 = s_4 = i, s_1 = \emptyset$).

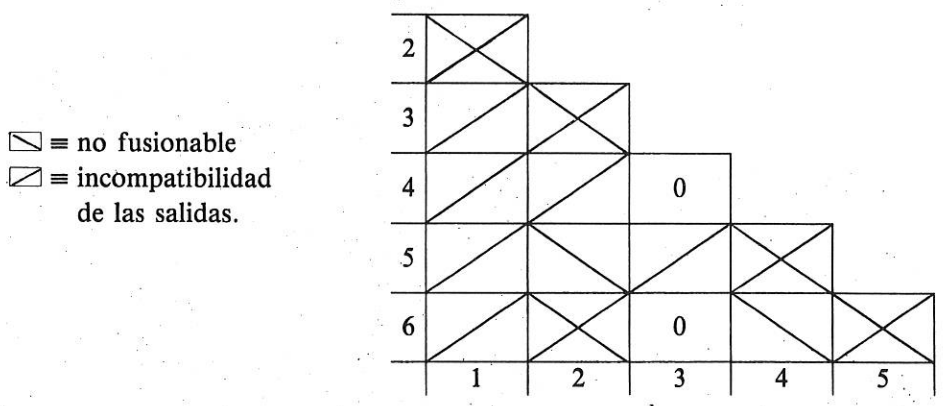


Tabla 3.2 Superposición de la compatibilidad de las salidas y de la fusión para obtener los conjuntos máximos fusionables que proporcionen una máquina cuyas acciones sean incondicionales (máquina de MOORE).

$\{1, 3, 5\}$ con 3 salidas que distinguir ($s_1 = \emptyset, s_3 = i, s_5 = d$).
$\{1, 3, 6\}$ con 2 salidas que distinguir ($s_3 = s_6 = i, s_1 = \emptyset$).

La figura 3.8 muestra los GR que obtenemos al realizar las fusiones $\{1, 3, 4\}$ y $\{1, 3, 6\}$. Estas fusiones son las que proporcionan un menor número de salidas a distinguir. En ellas hemos marcado con «?» los eventos y/o condiciones sobre las salidas en los que hay que introducir modificaciones.

Si tuviésemos que realizar una síntesis, es posible que la solución (*b*) fuera la mejor, puesto que es la más próxima a la descripción inicial (tiene un mayor sentido físico). Por razones didácticas vamos a desarrollar la solución (*a*).

a) *Cálculo de los nuevos eventos*

La transición $134 \rightarrow 2$ se debería realizar sólo si el sistema se encontrara en el estado inicial 1. Por consiguiente, estudiaremos cómo diferenciar $\{1\}$ de $\{3, 4\}$; es decir, cómo realizar esa partición del lugar fusión $\{1, 3, 4\}$.

Puesto que $\{1\}$ y $\{3, 4\}$ son fusionables, tendremos:

$$\nu_1(\nu_3 + \nu_4) = 0$$

Ahora bien,

$\nu_1\nu_3 = 0$ se debe a la variable A,
$\nu_1\nu_4 = 0$ se debe indistintamente a la variable A o a la B.

La condición suplementaria que habremos de asociar al evento destinado a etiquetar la transición $134 \rightarrow 2$ vendrá dada por una función *lógica* en la que se considere el estado de las variables de entrada en el lugar 1 (puesto que en la red original sólo existe la transición $1 \rightarrow 2$):

$$f^1_{134\rightarrow 2} = \underbrace{A(A + \bar{B})}_{\text{permite distinguir } \{1\} \text{ y } \{4\}} = A \text{ permite distinguir } \{1\} \text{ y } \{3, 4\}$$

A permite distinguir $\{1\}$ y $\{3\}$

El evento completo será $M? = Mf^1_{134\to 2} = MA$.

Nota. En general, la función lógica $f^q_{i\to j}$ podrá ser desarrollada en suma de productos, $f^q_{i\to j} = \sum_k \prod_k$. Es obvio que cada uno de los productos $\prod_k$ puede ser utilizado independientemente para definir el nuevo evento: $e_{\text{nuevo}\,k} = e_{\text{antiguo}\,k} \prod_k$.

La transición $134 \to 5$ se debe realizar sólo si el sistema se encuentra en el estado 4. Procediendo de la misma forma que antes, obtenemos que para el lugar 4 y la mencionada transición:

$$f^4_{134\to 5} = (\underbrace{\bar{A} + B}_{\text{permite distinguir 4 y 1}})\underbrace{S}_{\text{permite distinguir 4 y 3}} = \bar{A}S + BS$$

El evento completo podrá ser:

1) $\bar{B}? = \bar{B}\bar{A}S$
2) $\bar{B}? = \bar{B}BS = 0$ (¡la transición no se disparará nunca!).

Como la segunda opción ha conducido a una incongruencia, se eligirá la primera; es decir, el evento asociado a la transición $134 \to 5$ será $\bar{A}\bar{B}S$.

La incongruencia a la que llegamos anteriormente tiene una fácil explicación, que permite presentar una limitación de *este* método de fusión de lugares. En efecto, la mencionada limitación se presenta cuando, dado un lugar p_i, la intersección de su condición envolvente ν_i con algún evento asociado a sus transiciones de salida puede ser nula. Así, es posible encontrar:

$$\nu_i = \nu_i^* \bar{V} \qquad e_{ik} = e^*_{ik} V,$$

donde V es una variable de entrada. Esta situación corresponde, por ejemplo, al caso en que p_i represente un estado de espera de V ($e^*_{ik} = 1$). En estas circunstancias, si la variable V es la que diferencia el lugar p_i de los otros con los que se pretende fusionarlo, la fusión es imposible.

b) Cálculo de las condiciones suplementarias que es preciso asociar a las acciones

El problema que abordamos es análogo al que hemos tratado anteriormente. En el ejemplo desarrollado tenemos $s_3 = s_4 = i$ y $s_1 = \emptyset$, luego hemos de particionar el lugar de fusión $\{1, 3, 4\}$ en $\{1\}$ y $\{3, 4\}$. Este caso ya ha sido estudiado; como vimos, la variable A basta para asegurar la partición. En $\{3, 4\}$ la variable A está en su estado complementario, luego la acción i, asociada a $\{1, 3, 4\}$, deberá estar condicionada por $\bar{A}{:}i \mid \bar{A}$.

En resumen, el problema tratado en este apartado (§3.4.2.3) no es más que la búsqueda de condiciones suplementarias que permitan la partición del lugar de fusión. Esta partición se elige de forma que se respete la coordinación evento-acción contenida en la descripción inicial.

Es muy importante observar que la fusión de lugares permite obtener un GR simplificado cuyo comportamiento (relación evento-acción que representa) no es

idéntico al que exhibe el GR original. Así pues, la secuencia de eventos $M\uparrow D\uparrow \uparrow A\downarrow A\uparrow A\downarrow A\uparrow$ genera en los GR inicial y simplificado (figuras 3.7*a* y 3.8*a*) diferentes secuencias de acciones. Por ello se advierte al diseñador para que prevea los posibles *efectos secundarios* de sus simplificaciones.

EJERCICIO

a) Complétese el GR simplificado de la figura 3.8*b*.
b) Obténgase el GR simplificado que corresponda a la fusión $\{1, 3, 5\}$.

Antes de estudiar la simplificación de las RdP por el método de los lugares fuente, recordemos que, para mantener el sentido físico del modelo, la simplificación por fusión de lugares se *debe* realizar siempre en subRdP que no posean nudos *Y* (distribución y conjunción).

3.5 MÉTODO DE LOS LUGARES FUENTE

3.5.1 Generalidades y terminología

Como ya hemos visto anteriormente, el método de fusión de lugares compatibles no hace más que simplificar una RdP mediante la extracción de funcionamientos combinatorios locales (que conciernen a un subconjunto de lugares). En cambio, el método de los lugares fuente trata de simplificar la RdP extrayendo funcionamientos combinatorios globales es decir, que conciernen a todo el sistema†.

La característica principal de este método es que la simplificación de la parte secuencial de una RdP no conlleva, a diferencia del método de fusión de lugares, una complicación de la parte combinatoria.

Antes de llevar a cabo la presentación del método, introduciremos algunos conceptos.

Definición 3.4. Se llama *condición envolvente de un marcado M*, representado por μ_M, a la intersección lógica de las condiciones envolventes de los lugares marcados por M: $\mu_M = \prod_{M(p_i)=1} \nu_i$. □

Definición 3.5. Desde un punto de vista estructural, una transición t es *fuente* [*sumidero*] si no posee lugares de entrada, ${}^{\cdot}t = \emptyset$ [de salida, $t^{\cdot} = \emptyset$]. □

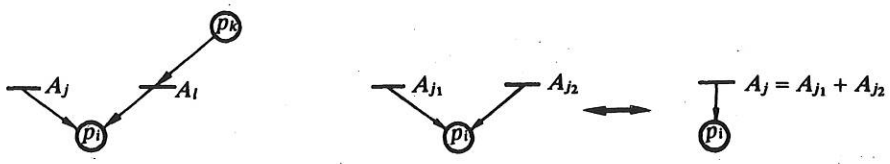
Definición 3.6. Un lugar p_i es *fuente* con respecto a un evento e_k si la ocurrencia de e_k implica el marcado de p_i. □

Definición 3.7. Un lugar es *fuente* [*sumidero*] si todas sus transiciones de entrada [de salida] son transiciones fuente [sumidero]. □

---

† En una tabla de fases, un funcionamiento combinatorio global está caracterizado por un subconjunto de columnas en las que existe un único estado estable por columna.

A partir de estas definiciones, presentaremos (§3.5.2) una condición suficiente para que un lugar sea fuente con respecto a un evento. Posteriormente enunciaremos dos reglas de simplificación de un modelo. Por último (§3.5.3), aplicaremos el método de simplificación a un ejemplo.



(*a*) p_i es fuente con respecto a A_j (*b*) p_i es fuente

Figura 3.9. Lugar fuente con respecto a un evento y lugar fuente.

3.5.2 Presentación del método de los lugares fuente

Sean: $\{M\}_i$ el conjunto de marcados de la red $\langle R, M_0\rangle$ para el cual el lugar p_i está marcado, M_k un marcado alcanzable a partir de M_0 y A_j un evento que etiqueta una transición de entrada de p_i.

Condición suficiente para que un lugar sea fuente con respecto a un evento (proposición 3.4). Si $\forall M_k \in M(\boldsymbol{R}, M_0) - \{M\}_i$ se tiene $A_j\mu_{M_k} = 0$, entonces p_i es fuente con respecto a A_j. □

Dicho de otra forma, p_i es fuente con respecto a A_j si la intersección de A_j con las condiciones envolventes de los marcados es nula cuando se consideran aquéllos para los cuales p_i no está marcado; sólo en caso contrario *puede* no ser nula.

DEMOSTRACIÓN. Por construcción de $\{M\}_i$, si $\forall M_k \in M(\boldsymbol{R}, M_0) - \{M\}_i$ se tiene $A_j\mu_{M_k} = 0$ cuando $A_j = 1$, el marcado no puede pertenecer a $M(\boldsymbol{R}, M_0) - \{M\}_i$, luego tiene que pertenecer a $\{M\}_i$ y, por lo tanto, p_i estará marcado. □

Caso particular. Si el modelo de un sistema es un grafo de estados binario e interpretado (GR) y si $\forall k \neq i$ se tiene que $A_j\nu_k = 0$, entonces p_i es fuente con respecto a A_j.

REGLAS DE SIMPLIFICACIÓN

REGLA 1: Si un conjunto de lugares es fuente con respecto a un evento que etiqueta una transición de entrada común, los lugares de entrada y de salida de la transición pueden ser desconectados (figura 3.10*b*): la transición se desdobla en dos, una fuente y otra sumidero, independientes.

REGLA 2: Si un lugar sumidero no tiene asociada ninguna acción y sus transiciones de salida no poseen ningún otro lugar de entrada, aquél puede ser suprimido (figura 3.10*c*).

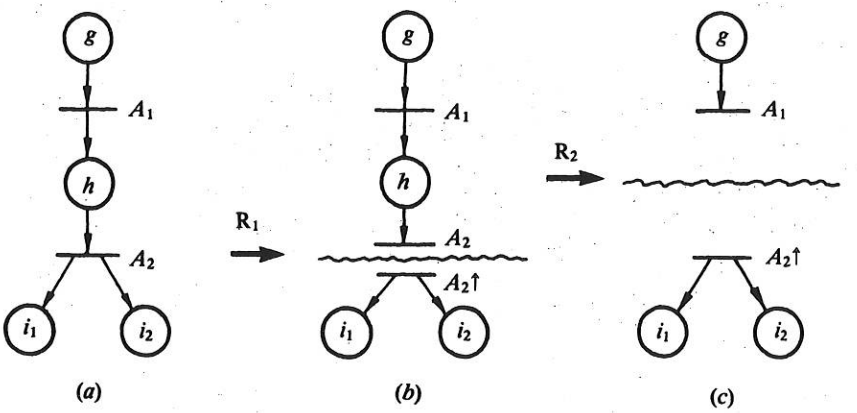


Figura 3.10. Simplificación si $\{p_{i_1}, p_{i_2}\}$ son fuentes con respecto a A_2 (h es eliminado).

3.5.3 Aplicación

Consideremos el ejercicio 1.2. En la figura 3.11*a* se presenta una posible descripción del sistema. Un análisis detallado del problema da como resultado el estado de las variables de entrada asociadas a los lugares (figura 3.11*b*).

A partir de la figura 3.11 obtenemos sin dificultad lo siguiente:

$$\mu_1 = \nu_1 = \bar{M}A\bar{B}C\bar{D}$$
$$\mu_2 = \nu_2\nu_3 = \bar{B}\bar{D}$$
$$\mu_3 = \nu_2\nu_5 = \bar{A}\bar{B}\bar{C}D$$
$$\mu_4 = \nu_3\nu_4 = \bar{A}\bar{C}\bar{D}$$
$$\mu_5 = \nu_3\nu_6 = A\bar{B}\bar{C}\bar{D}$$
$$\mu_6 = \nu_4\nu_5 = \bar{A}\bar{C}D$$
$$\mu_7 = \nu_5\nu_6 = 0 \ (M_7 = M(p_5) + M(p_6) \text{ es un marcado transitorio}\dagger)$$
$$\mu_8 = \nu_7 = A\bar{B}\bar{C}$$

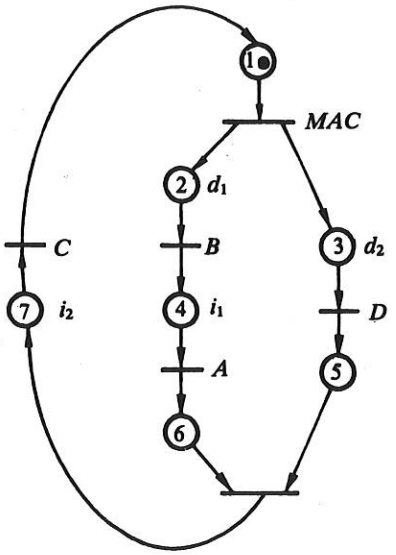
Evaluados los μ_i, es fácil comprobar que $\forall i \neq 2 \ \mu_i MAC = 0$, luego la transición etiquetada MAC podrá ser desdoblada (regla 1). Puesto que p_2 y p_3 son fuentes con respecto al evento MAC y como, además, p_1 no tiene asociada ninguna acción, es posible eliminarlo (regla 2).

La figura 3.12 presenta la RdP simplificada. Puesto que la regla 3 sobre transformación de condiciones externas y eventos (§2.3.3) es aplicable, aunque físicamente se haya eliminado p_1, $(MAC)\uparrow$ se puede transformar en MAC.

EJERCICIO. Aplicando la regla 1 de transformación (§2.3.3), podemos reducir MAC a MC o incluso a M en la red original. Compruébese que MC permite aún la eliminación de p_1, pero ésta no es posible si la transición $1 \rightarrow 23$ se etiqueta sólo con M.

---

† Esto se puede comprender fácilmente porque la transición de salida de p_5 y p_6 no está etiquetada.



(*a*)

p_j	M	A	B	C	D
1	$\bar{M}$	A	$\bar{B}$	C	D
2	—	—	$\bar{B}$	—	—
3	—	—	—	—	$\bar{D}$
4	—	$\bar{A}$	—	$\bar{C}$	—
5	—	$\bar{A}$	—	$\bar{C}$	D
6	—	A	$\bar{B}$	$\bar{C}$	—
7	—	A	$\bar{B}$	$\bar{C}$	—

(*b*)

Figura 3.11. Aplicación al ejercicio 1.2 (figura 1.9).

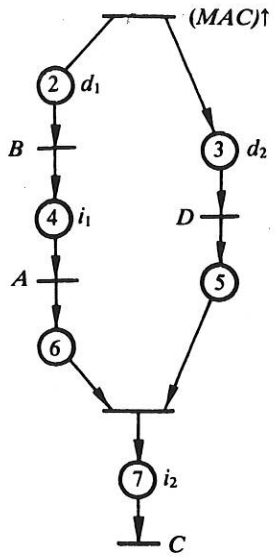


Figura 3.12. Primera simplificación de la red de la figura 3.11.

3.6 EJEMPLO Y CONCLUSIÓN

Para ilustrar la potencia de los métodos de simplificación que hemos presentado continuaremos con el ejercicio 1.2. En el estado de simplificación en que se encuentra la RdP de la figura 3.12 no existen ni lugares implícitos ni lugares fuente. (Verifíquelo el lector.)

Aplicando el método de fusión de lugares, al eliminar los nudos Y aparecen los subconjuntos de lugares conexos $\{2, 4, 6\}$, $\{3, 5\}$ y $\{7\}$.

Es fácil comprobar que $v_4 v_6 = 0$ y $v_3 v_5 = 0$, luego son fusionables. La figura 3.13*a* presenta el resultado de estas fusiones. (Compruébelo el lector.)

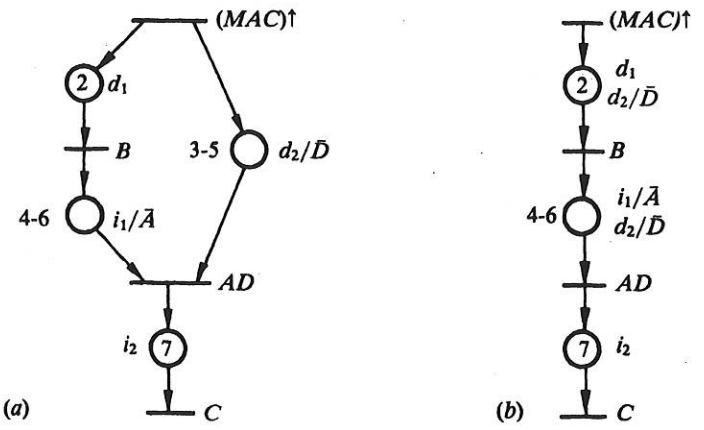


Figura 3.13. Segunda y tercera simplificaciones de la red de la figura 3.11.

El lugar {3-5} (figura 3.13*a*) es implícito (está implicado por {2} y {4-6}). La figura 3.13*b* muestra el resultado de la simplificación.

Como conclusión de lo tratado en este capítulo podemos decir que:

1) La simplificación se realiza de forma *iterativa* e interactiva. El diseñador es quien debe tomar las decisiones con respecto a las simplificaciones que se realicen.
2) El método estructural de simplificación denominado «de los *lugares implícitos*» (§3.3) no necesita información «extra» para ser aplicado. Tampoco introduce secuencias no previstas. Elimina sólo redundancias estructurales. Este método es directamente aplicable a las RdPG.
3) Los métodos de simplificación denominados *fusión de lugares* (§3.4) y *lugares fuente* (§3.5) son aplicables a las RdP binarias e interpretadas de acuerdo con el convenio adoptado en §2.4. Así pues, estos métodos no tienen sentido si se consideran RdP interpretadas de acuerdo con el convenio establecido para representar el flujo del control de los programas de un computador digital (Anexo 1).

Los métodos de simplificación considerados introducen en la descripción secuencias de acciones que, en caso de funcionamientos no previstos (averías, etc.), pueden acarrear problemas.

4) Por razones de modificabilidad, reparabilidad, etc., es conveniente que las descripciones simplificadas tengan un *sentido físico*, lo cual impone restricciones a las simplificaciones, aun cuando éstas sean posibles.
5) El método de simplificación por *fusión de lugares* disminuye la parte secuencial de la descripción (número de lugares), normalmente a costa de complicar los eventos y las condiciones asociadas a las acciones. Es decir, este método complica la parte combinatoria. Se basa en la extracción de funcionamientos combinatorios locales.
6) El método de simplificación de los *lugares fuente* disminuye la parte secuencial de la descripción, sin complicar los eventos y las condiciones asociadas a las acciones. Habida cuenta de que este último utiliza funcionamientos combinatorios globales, su aplicación no será rentable más que para automatismos de dimensión relativamente pequeña.

Por último, cabe señalar que el haber profundizado sobre la modelación de sistemas con RdP interpretadas es seguramente una de las mayores aportaciones de este capítulo.

EJERCICIOS

3.1 Para las RdP de la figura E.3.1, determínese si existen lugares implícitos y obténganse conjuntos de lugares implicantes.

3.2 Simplifíquense los modelos obtenidos en los capítulos 1 y 2.

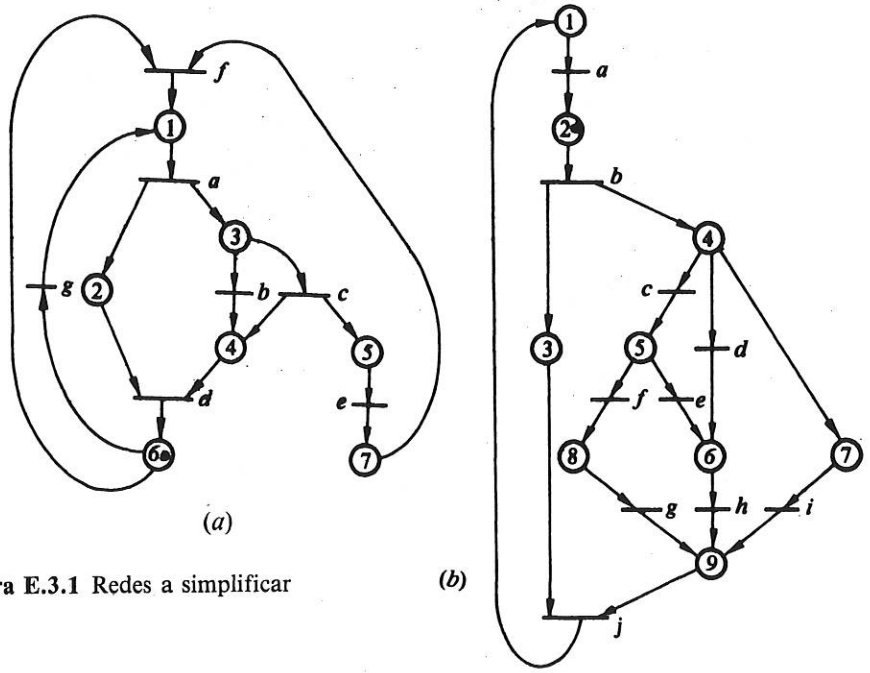
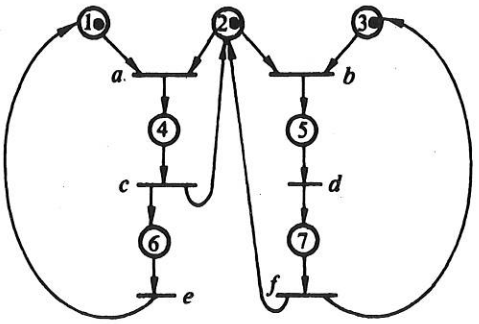
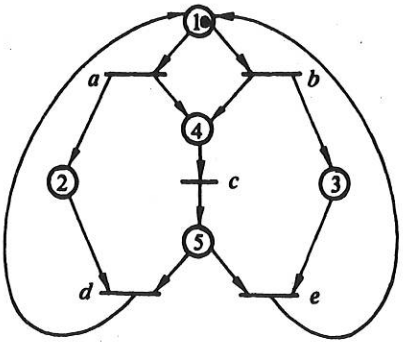


Figura E.3.1 Redes a simplificar



(*c*)



(*d*)

Figura E.3.1 (cont.). Redes a simplificar

4

Validación funcional de una descripción (I): (I): redes de Petri autónomas

4.1 INTRODUCCIÓN

En los capítulos anteriores hemos abordado la modelación de sistemas discretos con actividades simultáneas. En este capítulo y en el siguiente estudiaremos la validación funcional del modelo obtenido.

En la introducción general a esta obra hicimos referencia al hecho de que la creciente complejidad de los sistemas hace que éstos resulten difícilmente comprensibles para el diseñador (o equipo de diseño). Con el fin de evitar que en el proceso de concepción de un sistema se pase a la fase de realización con un modelo erróneo, se introducirá una fase en la que se realice un *estudio cualitativo del modelo funcional* que se ha diseñado.

La *validación funcional* y la *verificación funcional* son dos tipos de estudios cualitativos que pueden ser objeto de nuestro interés.

En un análisis de validez, o *validación*, se determina si el modelo del sistema que ha sido diseñado, es decir, su descripción, cumple una serie de propiedades que caracterizarán su buen funcionamiento. Estas propiedades son, en gran parte, independientes de las funciones específicas que realiza el sistema, así como de la herramienta de modelación (en este caso, redes de Petri). Entre ellas cabe citar la ausencia de bloqueos, la finitud del conjunto de estados, la ausencia de conflictos, etc.

En un estudio de *verificación* se examina la descripción del sistema a fin de comprobar si ésta cumple las especificaciones del sistema. El estudio de verificación tiene que considerar la semántica, la significacción, de cada una de las operaciones o acciones elementales. Ello es necesario para que se pueda llegar a la conclusión de que la descripción del sistema que se ha concebido es capaz de cumplir todos los requisitos previstos. El estudio de verificación exige una definición formalizada de los objetivos, cosa que, desafortunadamente, no es fácil de conseguir.

La gran cantidad de información que sobre el comportamiento del sistema modelado posee la estructura de la RdP y su marcado inicial, permite abordar un primer análisis de la validez del modelo concebido. Se trata de una *validación funcional de la RdP autónoma*; ésta es la temática sobre la que versará este extenso capítulo. Re-

saltemos que, salvo el material presentado en §4.8, toda la teoría que vamos a desarrollar será directamente aplicable a las RdP generalizadas.

En el próximo capítulo abordaremos el estudio de las consecuencias que sobre la validación de un modelo tiene la incorporación del tiempo y/o la interpretación. Por último, propondremos dos métodos de modelación. El método *descendente* pretende que *por construcción* el modelo sea válido. El método que denominamos *modular* parte de la descripción aislada de subsistemas, para establecer a continuación sus interconexiones. Ambos se pueden combinar en la descripción de un sistema.

La estructura de este importante capítulo es la siguiente: comienza por la enumeración de las «propiedades de buen funcionamiento» más importantes; a continuación se clasifican los métodos disponibles para analizar la validez de un modelo realizado con RdP y, por último, se presentan los métodos básicos de análisis.

4.2 PROPIEDADES BÁSICAS QUE CARACTERIZAN EL FUNCIONAMIENTO DE LOS SISTEMAS CON EVOLUCIONES SIMULTÁNEAS

En este apartado definiremos una serie de propiedades básicas y comunes a todos los sistemas con evoluciones simultáneas. La traducción de estas propiedades generales a otras específicas de las RdP nos permitirá abordar el análisis de validez de los modelos construidos con RdP.

La serie de propiedades que vamos a definir no es exhaustiva, pues sólo comprende las que han sido consideradas más importantes en una introducción a estos temas.

Entre las propiedades características que todo sistema debe poseer están, por una parte, la *ausencia de bloqueos* —totales o parciales— y, por otra, la *finitud del conjunto de estados* en los que se puede encontrar el sistema. En la terminología específica de las RdP, la ausencia de bloqueos se traduce en propiedades de *vivacidad*. La finitud del conjunto de estados se traduce en propiedades de *limitación*. Ambas clases de propiedades fueron introducidas de forma intuitiva y muy simplificada en el capítulo 1 (§1.5.3.2).

Para estudiar las situaciones de ambigüedad ante una decisión entre diversas alternativas en la evolución del sistema (indeterminación), se definen las propiedades de *conflictividad*.

Otra propiedad interesante es, por ejemplo, la exclusión mutua entre estados parciales. En la terminología propia de las RdP, la *exclusión mutua* se traduce en restricciones sobre los marcados alcanzables a partir del marcado inicial. Dos lugares están en exclusión mutua si, a partir del marcado inicial, nunca pueden estar marcados simultáneamente.

4.2.1 Vivacidad

Definición 4.1. Una transición t es *viva para un marcado inicial dado* M_0 sii existe una secuencia de disparos a partir de cualquier marcado M, sucesor de M_0, que comprenda a t:

$$\forall M \in M(R, M_0) \quad \exists \sigma : M \xrightarrow{\sigma} M' \text{ tal que } t \subset \sigma. \qquad \square$$

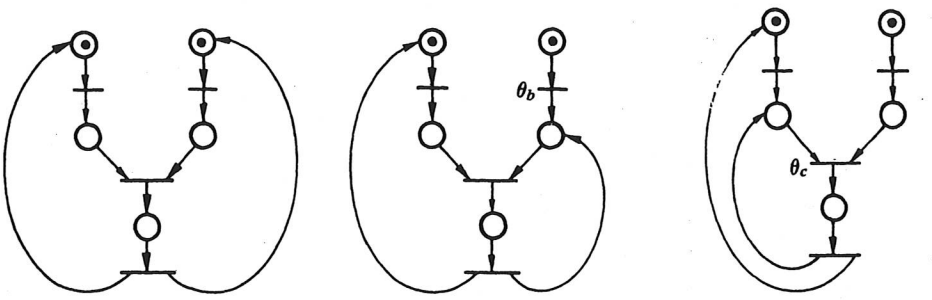
Definición 4.2. Una RdP marcada es *viva para* M_0 sii todas sus transiciones son vivas para M_0. □

La importancia de la propiedad de vivacidad estriba en su capacidad para caracterizar el bloqueo de un sistema. En efecto, si una RdP es viva, el sistema no puede bloquearse, puesto que todas las transiciones pueden llegar a dispararse. Evidentemente, la proposición contraria no es cierta. Puede suceder que una RdP marcada no viva no se bloquee. Para caracterizar esta última situación se define la noción de RdP marcada parcialmente viva.

Definición 4.3. Se dice que una RdP marcada es *parcialmente viva para* M_0 si, tomando como punto de partida cualquier marcado alcanzable a partir de M_0, existe al menos una transición disparable y otra transición no viva. □

Toda RdP marcada parcialmente viva tiene la posibilidad de evolución global, independientemente de que existan transiciones que no puedan ser disparadas.

La figura 4.1 ilustra estas nociones. La RdP de la figura 4.1*b* puede evolucionar indefinidamente, aunque la transición θ_b sólo pueda ser disparada una vez (a partir de M_0). La RdP de la figura 4.1*c* se bloquea globalmente al no poderse disparar por segunda vez la transición θ_c. Si se analiza detenidamente esta última RdP, se observará que para cualquier marcado inicial finito se obtiene una RdP marcada no viva. La caracterización de esta propiedad se hace a través del concepto de vivacidad estructural.



(*a*) RdP viva para M_0

(*b*) RdP parcialmente-viva para M_0

(*c*) RdP no-viva (se bloquea totalmente)

Figura 4.1 Vivacidad en las RdP.

Definición 4.4. Una RdP R es *estructuralmente viva* si existe un M_0 finito para el cual la RdP marcada es viva. □

Evidentemente, la vivacidad estructural es una condición necesaria pero no suficiente para la vivacidad.

4.2.2 Ciclicidad

Definición 4.5. Se dice que una RdP posee un *comportamiento globalmente cíclico para* M_0 si existe una secuencia de disparos que permite alcanzar el marcado inicial M_0 a partir de cualquier marcado M alcanzable a partir de M_0:

$$\forall M \in M(\boldsymbol{R}, M_0), \qquad \exists \sigma \text{ tal que } M \xrightarrow{\sigma} M_0. \qquad \square$$

Para abreviar las referencias, hablaremos de RdP *cíclica para* M_0. La ciclicidad de una RdP marcada garantiza que no existen *subconjuntos finales de estados* (marcados). Un subconjunto final de estados (marcados) contiene estados (marcados) mutuamente alcanzables entre sí y tales que el estado inicial (marcado inicial) no es alcanzable a partir de ninguno de ellos. La justificación del calificativo *final* es ahora evidente: si el sistema evoluciona hacia un estado perteneciente a un subconjunto final, entonces su estado pertenecerá «in aeternum» al mencionado subconjunto, lo cual es, normalmente, inadmisible desde un punto de vista práctico. La RdP marcada de la figura 4.1*a* es cíclica.

La ciclicidad, así como la vivacidad parcial, caracterizan la existencia de evoluciones globales, independientemente de las transiciones disparables. Obviamente la ciclicidad implica la vivacidad parcial si existe al menos un marcado sucesor de M_0.

4.2.3 Limitación

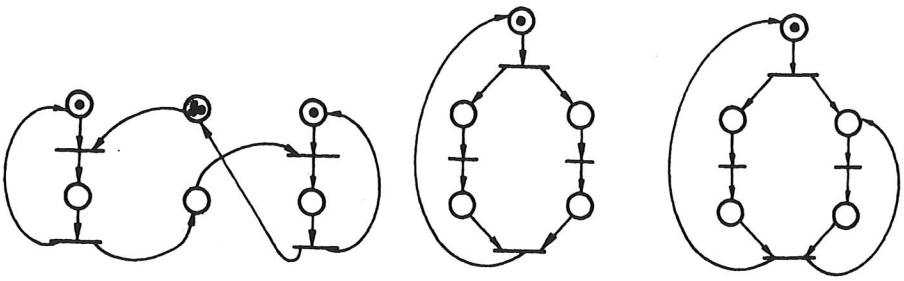
Definición 4.6. Un lugar p es *k-limitado para* M_0 sii existe un número entero k tal que $M(p) \leqslant k$ para cualquier marcado $M \in M(\boldsymbol{R}, M_0)$. Se denomina *límite del lugar p* al menor entero k que verifica la desigualdad anterior. $\square$

Definición 4.7. Una RdP marcada es *k-limitada para* M_0 sii todos sus lugares son k-limitados para M_0: $\forall p \in P$ y $\forall M \in M(\boldsymbol{R}, M_0)$, $M(p) \leqslant k$. $\square$

La propiedad de limitación determina la finitud del número de estados del sistema representado por una RdP. Desde un punto de vista práctico, esta propiedad debe verificarse, puesto que los lugares se realizarán con memorias de capacidad finita.

Para la representación de multitud de sistemas, merece una consideración especial la 1-limitación. Si una RdP es 1-limitada para M_0, su marcado es *binario* (un lugar está o no está marcado) y se dirá que *la RdP es binaria para* M_0. El interés de la RdP marcadas binarias —en lo sucesivo RdP binarias— reside en la simplicidad de su realización. Por otra parte, es importante observar que una RdP binaria está perfectamente adaptada a un gran número de problemas en los que hay que distinguir sólo dos estados para un elemento o subsistema (ocupado/no ocupado, activado/no activado, etc.). La figura 4.2 ilustra la noción de limitación.

Un estudio detallado de las RdP de las figuras 4.2*a* y *b* permite deducir que ambas RdP son limitadas para cualquier marcado inicial finito. La caracterización de esta propiedad se hace a través del concepto de limitación estructural.



(*a*) RdP 3-limitada para M_0 (*b*) RdP binaria para M_0 (*c*) RdP no limitada para M_0

Figura 4.2. Limitación en las RdP.

Definición 4.8. Una RdP es *estructuralmente limitada* si es limitada para cualquier marcado inicial y finito. □

La limitación estructural es una condición suficiente para la limitación. La RdP de la figura 4.2*c* no es estructuralmente limitada. La RdP de la figura 4.3 demuestra que la limitación estructural no es una condición necesaria para la limitación.

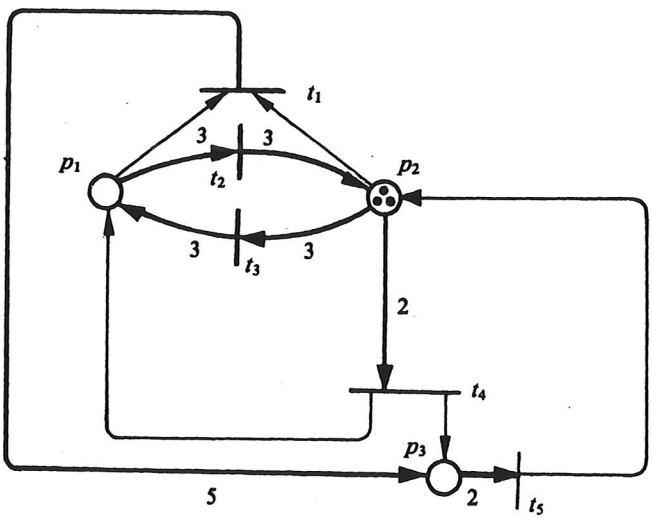


Figura 4.3. RdP marcada limitada y viva pero no estructuralmente limitada (para $M_0' = (0\,4\,0)^T$ es no-limitada).

La figura 4.4 permite concluir sobre la independencia de los conceptos de vivacidad, ciclicidad y limitación.

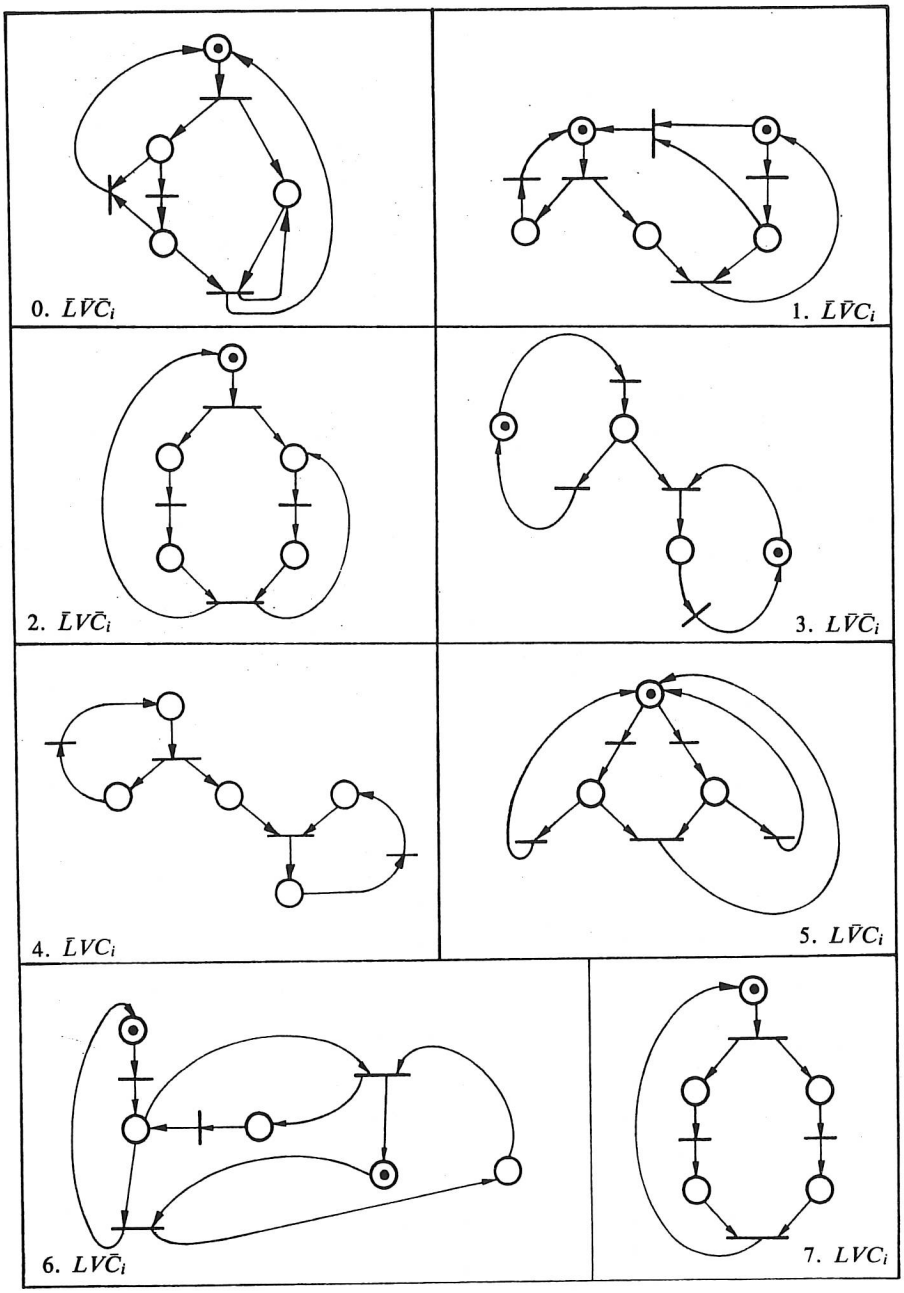


Figura 4.4. Independencia entre limitación, vivacidad y ciclicidad.

4.2.4 Conflictividad

Definición 4.9. Se dice que en una RdP existe *conflicto estructural* cuando un lugar posee más de una transición de salida. □

La figura 4.5 exhibe un conflicto estructural, puesto que t_1 y t_2 son transiciones de salida de un mismo lugar, p.

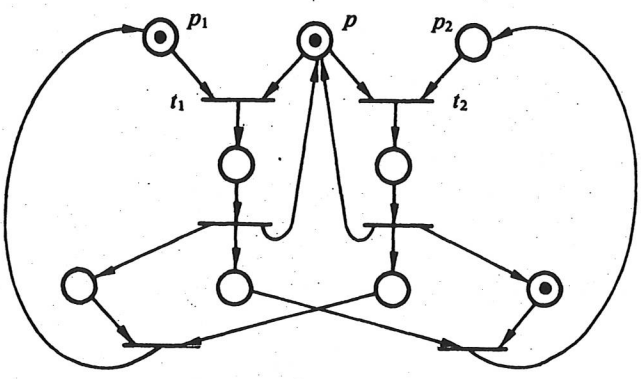


Figura 4.5. Conflicto estructural pero no efectivo.

Definición 4.10. Se dice que dos transiciones, t_i y t_j, están en *conflicto efectivo* para M_0 sii:

—existe $M \in M(R, M_0)$ que sensibiliza a t_i y t_j;
—al disparar t_i (t_j) el marcado obtenido no sensibiliza la transición t_j (t_i). □

Dicho de otro modo, existe conflicto efectivo entre t_i y t_j cuando las dos transiciones tienen al menos un lugar de entrada común y éste no posee suficientes marcas para permitir el disparo simultáneo de ambas transiciones. Por ejemplo, la RdP de la figura 2.1 presenta un conflicto efectivo pues, para el marcado inicial que exhibe, el lugar h no puede contener más de una marca. La RdP de la figura 4.5 no presenta conflicto efectivo para el marcado inicial indicado, puesto que p_1 y p_2 no llegan a estar marcados simultáneamente.

La situación de conflicto efectivo es inaceptable para cualquier descripción de un sistema, dado que será ambigüa. El conflicto efectivo se resuelve, normalmente, mediante la interpretación asociada a la red: estableciendo una exclusión mutua entre los eventos asociados a las transiciones. Si, pese a la interpretación asociada a la RdP, persiste la situación de conflicto efectivo, entonces es obvio que la descripción del sistema que se pretende modelar es deficiente: no se define con claridad una decisión (ambigüedad).

Un caso típico de descripción deficiente que genera conflicto efectivo se presenta cuando un recurso (R) debe ser compartido por dos usuarios (U_1 y U_2) y no se ha definido con claridad una regla de prioridad para el acceso a él (figura 4.6).

Llegados a este punto, conviene destacar que, independientemente de la interpretación asociada a una RdP marcada, la existencia de un conflicto estructural no es condición suficiente para que haya conflicto efectivo (figura 4.5). Ahora bien, para que exista conflicto efectivo es necesario que haya conflicto estructural.

En una red interpretada, insistimos, los conflictos efectivos deben ser resueltos mediante los eventos asociados a las transiciones.

4.2.5 Exclusión mutua

Definición 4.11. Se dice que dos lugares de una RdP están en *exclusión mutua para* M_0 si no pueden estar marcados simultáneamente en los marcados alcanzables a partir de M_0. □

De acuerdo con esta definición, se dirá que los lugares 4 y 5 de la figura 4.6 están en exclusión mutua. Esto se puede verificar fácilmente considerando los marcados alcanzables a partir de M_0.

Un ejemplo típico de utilización del concepto de exclusión mutua se encuentra en el análisis de sistemas con recursos compartidos por dos o más usuarios (véanse ejemplos en el capítulo 2): si la utilización de un recurso por el i-ésimo usuario se representa por $M(p_i) = 1$, entonces como máximo un único p_i estará marcado, $\sum_i M(p_i) \leqslant 1$. En caso contrario, dos o más usuarios utilizarán el recurso simultáneamente. Ésto es lo que ocurriría si los dos vagones se encontraran en el tramo de vía común de la figura 2.11.

Π_i = petición del recurso por U_i.
R_i = recurso asignado a θ_i
FU_i = fin de utilización del recurso por U_i.

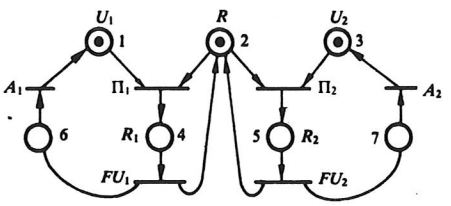


Figura 4.6. Si $\Pi_1 \cap \Pi_2 \neq \emptyset$ existe conflicto efectivo.

4.2.6 Relaciones sincrónicas

En este apartado presentaremos algunos conceptos que nos permitirán abordar un primer análisis de la interdependencia entre los disparos de las transiciones de una RdP marcada. Sólo introduciremos los conceptos básicos de avance sincrónico y avance sincrónico ponderado.

Sea la red marcada $\langle \boldsymbol{R}, M_0 \rangle$ y sea $\boldsymbol{L}(\boldsymbol{R}, M_0)$ el conjunto de todas las secuencias de disparo de transiciones aplicables. El número de disparos de la transición t_i en la secuencia $\sigma \in \boldsymbol{L}(\boldsymbol{R}, M_0)$ se representará por $\bar{\sigma}(t_i)$.

Definición 4.12. El *avance sincrónico* de la transición t_i con respecto a la transición t_j en la red $\langle R, M_0 \rangle$ es el valor máximo que, considerando todas las secuencias de disparo posibles, puede tomar la diferencia entre el número de disparos de t_i y el de t_j. Simbólicamente escribiremos:

$$AV(R, M_0; t_i, t_j) = \max_{\sigma \in L(R, M_0)} \{\bar{\sigma}(t_i) - \bar{\sigma}(t_j)\}. \qquad \square$$

El avance sincrónico es una cantidad no negativa, puesto que siempre existirá una secuencia, eventualmente vacía, que no dispare la transición t_j. Por consiguiente, el avance sincrónico será nulo, como mínimo.

La definición de avance sincrónico se puede generalizar a subconjuntos disjuntos de transiciones, $T_i \cap T_j = \emptyset$. Si $\bar{T}_i$ representa el *vector característico* del subconjunto de transiciones T_i, $\bar{T}_i(k) = 1$ si y sólo si $t_k \in T_i$. Si $\bar{T}_1$ y $\bar{T}_2$ representan los vectores característicos de los subconjuntos disjuntos T_1 y T_2, y $\bar{\sigma}$ representa el vector característico asociado a la secuencia de disparos σ, entonces el avance sincrónico del conjunto de transiciones T_1 con respecto al conjunto de transiciones T_2 en la red $\langle R, M_0 \rangle$ viene dado por:

$$AV(R, M_0; T_1, T_2) = \max_{\sigma \in L} \{(\bar{T}_1 - \bar{T}_2)^T \cdot \bar{\sigma}\}.$$

Si el avance de la transición t_i con respecto a t_j es nulo, entonces t_i no se puede disparar antes que t_j para el marcado inicial dado. Si, por el contrario, el avance es infinito, entonces existe una (o varias) secuencia disparable, σ, en la que el número de disparos de t_i es superior al de t_j en una cantidad no acotable. De este modo, si consideramos la RdP de la figura 4.6, podemos escribir $AV(R, M_0; \Pi_1, \Pi_2) = \infty$. De la misma forma, si consideramos la RdP de la figura 4.7, podremos escribir $AV(R, M_0; t_1, t_2) = \infty$.

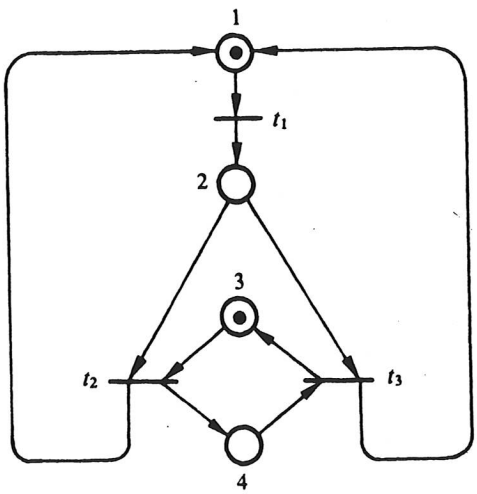


Figura 4.7. El avance de t_1 con respecto a t_2 no es limitado.

Un análisis más detallado de estos ejemplos demuestra que entre ambos existe una diferencia radical. En el primer caso (figura 4.6), podemos disparar una infinidad de veces consecutivas la transición etiquetada $\prod_1$ antes de disparar la transición etiquetada $\prod_2$. En el segundo caso (figura 4.7), podemos observar que existe en realidad una única secuencia de disparos de longitud infinita. En ésta se repite periódicamente la subsecuencia $t_1\ t_2\ t_1\ t_3$, y, por lo tanto, se puede afirmar que para poder disparar t_2 una vez hacen falta dos disparos de t_1. A partir de ahí, se deduce sin dificultad que, al repetirse indefinidamente la mencionada subsecuencia, se obtiene un avance sincrónico ilimitado.

Para distinguir ambos casos, tan diferentes desde el punto de vista de la interdependencia entre los disparos de las transiciones, generalizaremos el concepto de avance sincrónico. La generalización se obtiene al ponderar los disparos de las transiciones. De esta forma resulta obvio que, para la RdP de la figura 4.7, si $T_1 = \{t_1\}$ y $T_2 = \{t_2\}$ entonces $\max_{\sigma \in L} \{(\bar{T}_1{}^T - 2\bar{T}_2{}^T) \cdot \bar{\sigma}\} = 1$; su interpretación es inmediata: el número de disparos de t_1 menos el doble del número de disparos de t_2 es inferior o igual a la unidad.

Definición 4.13. Sean θ_1 y $\theta_2 \in \mathbb{N}^m$ dos vectores que expresan las ponderaciones que se asocian a las transiciones de los subconjuntos disjuntos T_1 y T_2 [$T_1 \cap T_2 = \emptyset \Rightarrow \theta_1(i)\,\theta_2(i) = 0$]. Se define el *avance sincrónico ponderado* de θ_1 con respecto a θ_2 en una RdP marcada $\langle \boldsymbol{R}, M_0 \rangle$ por la expresión:

$$AV(\boldsymbol{R}, M_0; \theta_1, \theta_2) = \max_{\sigma \in L} \{(\theta_1 - \theta_2)^T \cdot \bar{\sigma}\}. \qquad \square$$

De acuerdo con la definición 4.13, no existen θ_1 y θ_2 finitos tales que hagan que el avance ponderado entre t_1 y t_2 (figura 4.6) sea acotado.

Más adelante (§4.7.2) veremos cómo se puede aplicar esta noción al análisis de RdP; también estableceremos su relación con el concepto de lugar implícito, cuya presentación hicimos en §3.2 y 3.3.

En el próximo apartado clasificamos los métodos que permiten analizar la validez de una RdP, para luego estudiarlos en los apartados subsiguientes.

4.3 CLASIFICACIÓN DE LOS MÉTODOS DE ANÁLISIS

En general, los métodos de análisis de la validez de la RdP se pueden clasificar en los siguientes grupos:

1) análisis por enumeración,
2) análisis por transformación,
3) análisis estructural,
4) análisis por simulación.

Los métodos correspondientes a los tres primeros grupos se denominan *métodos estáticos*. Su aplicación a las RdP consideradas como grafos conduce a *resultados exactos*. Los métodos de simulación se denominan *métodos dinámicos* y permiten una «cierta confianza» en la descripción, pero *no demuestran* propiedades.

Estos últimos métodos tienen gran utilidad cuando a la evolución de la RdP se le asocian duraciones (RdP temporizadas), o bien cuando se pretende conocer la respuesta de un sistema descrito con RdP en un cierto ámbito definido también por simulación.

En este capítulo estudiaremos sólo métodos estáticos aplicados a redes autónomas. En el capítulo 5 haremos una breve alusión a los métodos dinámicos.

Los métodos de *enumeración* (§4.4) se basan en la construcción de un grafo que represente *individualizadamente* los marcados de la RdP y el disparo de sus transiciones. Si la RdP es limitada, el grafo es finito y se pueden verificar fácilmente las diferentes propiedades definidas en §4.2. Si la RdP no es limitada, sucede que el grafo no es finito y, por consiguiente, es imposible construirlo. En este caso, se pueden construir grafos finitos denominados de *alcanzabilidad o de cobertura* ([KARP 69], [BERT 78]).

A pesar de su potencia, este método es a veces difícilmente aplicable, incluso a RdP con pocos lugares, a causa de su fuerte naturaleza combinatoria.

El *análisis por transformación* se basa en la idea siguiente: dada una RdP marcada $\langle \boldsymbol{R}, M_0 \rangle$ sobre la que se desea verificar el conjunto de propiedades Π, se procede transformándola en la red $\langle \boldsymbol{R}', M_0' \rangle$ de forma que:

1) $\langle \boldsymbol{R}', M_0' \rangle$ satisfaga las propiedades Π sii $\langle \boldsymbol{R}, M_0 \rangle$ las satisface.
2) Sea más fácil verificar las propiedades Π sobre $\langle \boldsymbol{R}', M_0' \rangle$ que sobre $\langle \boldsymbol{R}, M_0 \rangle$.

Los métodos de reducción (§4.5) aparecen como caso particular de los métodos por transformación. En los métodos de reducción se va construyendo una secuencia de RdP marcadas que preservan las propiedades a estudiar. Esto se realiza de forma que la red $\langle \boldsymbol{R}^{i+1}, M_0^{i+1} \rangle$ sea más fácil de analizar que la red anterior en la secuencia, $\langle \boldsymbol{R}^i, M_0^i \rangle$, y tenga menos lugares o/y transiciones.

La existencia de redes irreducibles limita la aplicabilidad de estos últimos métodos. Desde un punto de vista práctico, las reducciones que normalmente se obtienen son sustanciales, permitiendo verificaciones directas de las propiedades de interés. No obstante, la existencia de RdP irreducibles hace necesaria su complementación con otros métodos de análisis.

Por último, los métodos de *análisis estructural* permitirán demostrar una serie de propiedades de la RdP, casi independientemente del marcado inicial de ésta. Es decir, lo que consideran fundamentalmente es la estructura de la RdP; de ahí su nombre.

La importancia de este tercer grupo de técnicas de análisis es muy grande puesto que, con frecuencia, en la práctica interesa que la red satisfaga determinadas propiedades estructurales, y no sólo las ligadas a un marcado inicial particular. Dicho de otro modo, estas técnicas permiten de forma eficiente el análisis de una RdP para diferentes marcados iniciales.

En este último tipo de análisis podemos distinguir dos subgrupos:

1) *Métodos basados en el álgebra lineal* (§4.6 y 4.7), en los que se parte de la ecuación de estado de la RdP. Son métodos que, en ciertos análisis, hacen posible un diagnóstico sin recurrir a la enumeración. Son aplicables directamente a las RdP generalizadas.

El material presentado en §4.7.2 también tiene gran importancia para el estudio de los métodos de realización cableada (capítulo 6) y microprogramada (capítulo 7) de las RdP.

2) *Métodos de cerrojos y trampas* (§4.8), en los que se determina una serie de subRdP de la RdP que se está analizando, de forma que se puede estudiar un conjunto de propiedades sin tener que recurrir a la enumeración de los marcados. Estos métodos son especialmente eficaces en el análisis de la vivacidad y de la limitación de subclases de RdP ordinarias.

Los tres grupos de métodos de análisis que hemos esbozado no son excluyentes, sino complementarios. Normalmente el diseñador los utilizará según las necesidades del proceso de concepción.

4.4 ANÁLISIS POR ENUMERACIÓN: GRAFO DE MARCADOS

Los métodos de esta clase se basan en la *simulación exhaustiva* de las evoluciones posibles del marcado de la RdP. La exposición que sigue se ceñirá fundamentalmente al caso en que la RdP sea limitada, con lo que el conjunto de los marcados alcanzables a partir de M_0 será finito.

4.4.1 Grafo de marcados: construcción

Definición 4.14. El *grafo de marcados* asociados a la RdP marcada $\langle \boldsymbol{R}, M_0 \rangle$ es un grafo $\boldsymbol{G}(\boldsymbol{R}, M_0)$ en el que cada nudo representa un marcado alcanzable a partir de M_0 y cada arco el disparo de una transición. Existe un arco, etiquetado t_k, que va desde el nudo que representa M_i al que representa M_j sii al disparar t_k a partir de M_i se alcanza $M_j : M_i \xrightarrow{t_k} M_j$. □

Si la RdP marcada es limitada y viva, el proceso de construcción del grafo de marcados es elemental. Culmina cuando se han considerado todas las evoluciones posibles a partir de los marcados alcanzables. Si consideramos la RdP de la figura 4.3, a partir de $M_0 = (0\,3\,0)^T$ podremos disparar t_3 o t_4. Si disparamos t_3, obtendremos $M_1 = (3\,0\,0)^T$. Si en vez de t_3 disparamos t_4, obtendremos $M_2 = (1\,1\,1)^T$. El proceso continúa de forma evidente. En la figura 4.8 se ha representado el grafo de marcados. (*Nota*: hemos representado los marcados en forma de productos; el exponente de un factor indica el número de marcas que posee el lugar representado por ese factor.)

En general, una RdP marcada puede ser no limitada, por lo que el proceso de construcción del grafo no acabaría nunca. Para evitar esta situación, durante el proceso de obtención del grafo se deberá tener en cuenta una condición de abandono. La generalización de esta idea nos ahorrará la molestia de completar la construcción del grafo correspondiente a ciertas redes anómalas (entre las que se encuentran las redes no limitadas). Sea $M \in \boldsymbol{M}(\boldsymbol{R}, M_0)$.

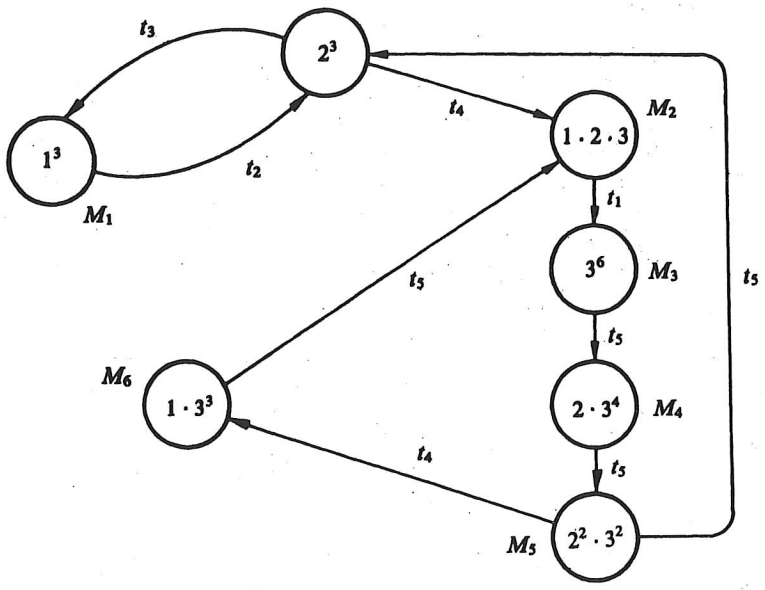


Figura 4.8. Grafo de marcados asociado a la $\langle \mathbf{R}, M_0 \rangle$ de la figura 4.3

Condición de abandono en la construcción de un grafo de marcados (proposición 4.1). Se puede abandonar la construcción del grafo de marcados de una RdP en los dos casos siguientes:

1) Si M no sensibiliza ninguna transición, porque la RdP no es viva (en este caso se bloquea totalmente).
2) Si existen marcados $M, M_1 \in \mathbf{M}(\mathbf{R}, M_0)$ tales que $M_1 \gtrdot M$ (M_1 *superior a* M), porque la RdP será no limitada estructuralmente o no viva. Además, se puede afirmar que si existe una secuencia de disparos σ, tal que $M \xrightarrow{\sigma} M_1$, la RdP es no limitada para el marcado considerado. □

DEMOSTRACIÓN. La demostración de la primera afirmación es inmediata. La figura 4.9 presenta un ejemplo en el que, a partir del marcado $M = (0\,1\,0)^T$, la RdP se bloquea. Es decir, a partir de M no se puede disparar ninguna transición.

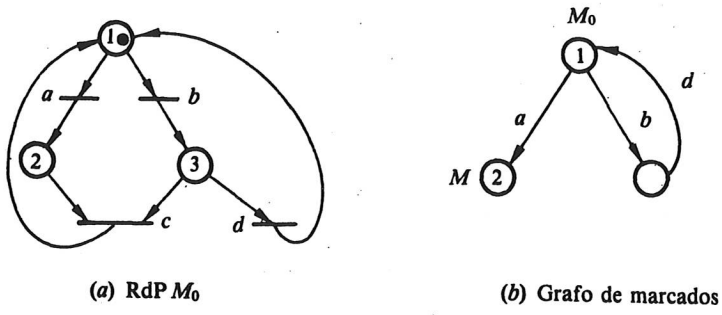
La demostración de la segunda afirmación es algo más complicada. Abordémosla primero para el caso particular (aunque frecuente) en el que se tiene $M_1 \gtrdot M$ y $M \xrightarrow{\sigma_1} M_1$.

a) *Caso particular.* Existe σ_1 tal que $M \xrightarrow{\sigma_1} M_1$.

Al repetir la secuencia σ_1 se evidencia que:

$$M \xrightarrow{\sigma_1} M_1 \xrightarrow{\sigma_1} M_2 \xrightarrow{\sigma_1} M_3 \xrightarrow{\sigma_1} \ldots, \text{ con } M_{i+1} \gtrdot M_i,$$

de donde al tener una serie de marcados estrictamente creciente, se deduce que la RdP no es limitada. (Si consideramos la RdP marcada de la figura 4.10, por ejem-



(*a*) RdP M_0 (*b*) Grafo de marcados

Figura 4.9. Red de Petri no-viva.

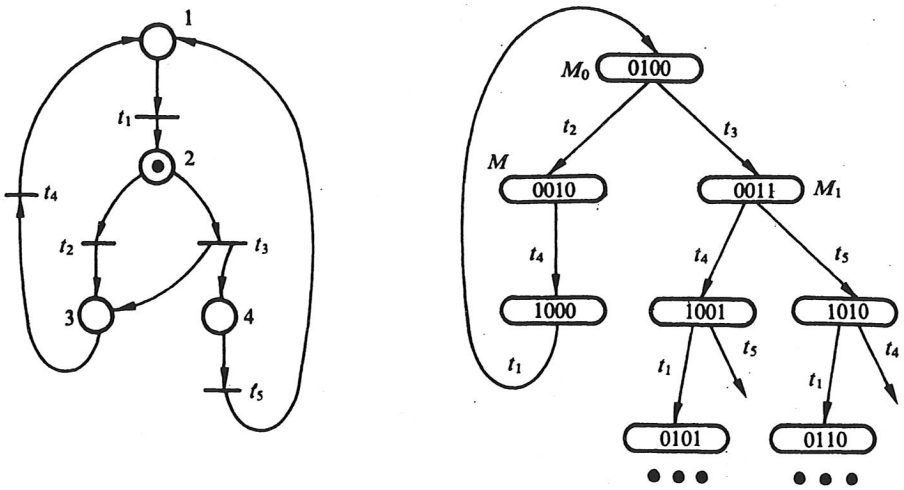


Figura 4.10 RdP marcada no-limitada. El grafo de marcados no es finito ($M_1 > M$).

plo, las secuencias $\sigma_1 = t_4\, t_1\, t_3$ o $\sigma_1' = t_3\, t_5\, t_1$ permiten afirmar que la mencionada red no es limitada, puesto que se alcanzan marcados estrictamente superiores a M y M_0, respectivamente.)

b) *Caso general.* No se sabe si existe un σ_1 o un σ_2 tales que $M \xrightarrow{\sigma_1} M_1$, o $M_1 \xrightarrow{\sigma_2} M$.

En este caso, de acuerdo con el enunciado, se puede afirmar que la RdP es no limitada estructuralmente o no viva. La demostración de esta afirmación necesita del conocimiento de un concepto que introduciremos en §4.6 (la conservatividad), y su lectura puede omitirse sin que por ello se dificulte la comprensión de los conceptos y resultados que se presentan posteriormente. No obstante, a modo de ilustración, podemos señalar que en las RdP de las figuras 4.10 y 4.11 se tiene $M_1 \gneqq M$. La primera de ellas es claramente no limitada (existe $\sigma = t_4\, t_1 t_3$ tal que $M \xrightarrow{\sigma} M_1$), mientras que la segunda es claramente no viva (no existe evolución posible a partir de M').

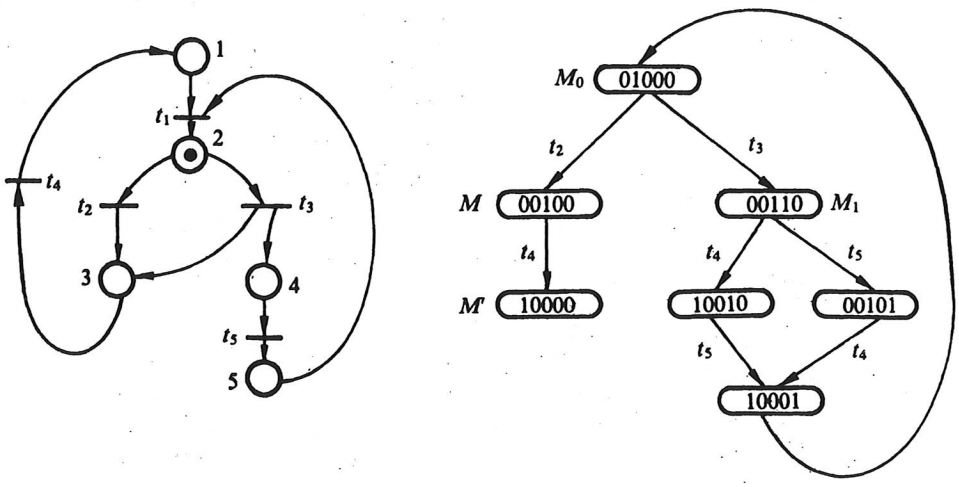


Figura 4.11. RdP marcada no viva y grafo de marcados (completo), ($M_1 > M$).

La proposición 4.9 (véase la tabla 4.2) establece lo siguiente:

[limitación estructural] ∧ [vivacidad estructural] ⇒ [conservatividad].

Ahora bien, $M_1 \gneqq M$ implica que la RdP no es conservativa, pues, si $Y \in (\mathbb{N}^+)^n$, $Y^T \cdot M_1 \neq Y^T \cdot M$. Invirtiendo la anterior implicación, la RdP será no limitada estructuralmente o/y no viva estructuralmente. Como [vivacidad] ⇒ [vivacidad estructural], la RdP será o no limitada estructuralmente o no viva, o ambas cosas a la vez. □

Si durante la construcción del grafo de marcados, $G(\boldsymbol{R}, M_0)$, no aparece nunca $M_1 \gneqq M$, se puede afirmar que el número de marcados alcanzables, y por lo tanto el número de nudos de $G(\boldsymbol{R}, M_0)$, es *finito*. En efecto, ello se puede inferir dado que el número de vectores no negativos (marcados) no superiores a los de un conjunto dado es finito.

EJERCICIO. Demuéstrese que una red $\langle \boldsymbol{R}, M_0 \rangle$ no puede ser simultáneamente limitada y cíclica si existen marcados M y M_1 alcanzables a partir de M_0 y tales que $M_1 \gneqq M$. (*Sugerencia*: hágase la hipótesis de que la red es cíclica, de donde existirá $\sigma = \sigma_1\sigma_2$ tal que $M \xrightarrow{\sigma_1} M_0 \xrightarrow{\sigma_2} M_1$, y conclúyase por contradicción.)

EJERCICIO. Compruébese que la RdP de la figura 4.3 no es limitada para $M_0' = (0\,4\,0)^T$.

A partir del grafo de marcados $G(\boldsymbol{R}, M_0)$ es conceptualmente muy fácil determinar las propiedades de la red marcada $\langle \boldsymbol{R}, M_0 \rangle$. En el próximo apartado abordamos el análisis de las propiedades más importantes.

4.4.2 Análisis a partir del grafo de marcados

En el apartado anterior hemos visto que el propio proceso de construcción del grafo de marcados suministra información sobre algunas propiedades de la RdP (en particular sobre la limitación).

En este apartado presentamos algunos resultados aplicables al análisis de la validez, cuya interpretación es inmediata. La sencillez de los análisis se debe a que el grafo de marcados contiene *toda* la información acerca de las posibles evoluciones de una RdP limitada.

4.4.2.1 *Análisis de la vivacidad*

Proposición 4.2. Una RdP limitada para M_0 es viva sii se cumple que en su grafo de marcados†:

1) no existe *nudo terminal*; es decir, no existe ningún marcado que no sensibilice ninguna transición;
2) toda *componente fuertemente conexa final* (es decir, tal que el conjunto de sus nudos sucesores esté incluido en la propia componente) tiene etiquetado el conjunto de sus arcos con todas las transiciones de la RdP. □

Antes de justificar la proposición 4.2 veamos en un ejemplo el significado de la segunda condición. Sea el grafo de marcados de la figura 4.12. Aplicando el algoritmo de obtención de componentes fuertemente conexas de un grafo (véase el Anexo 2, §A.2.3), se obtienen las componentes siguientes (ejercicio):

$$C_1 = \{10103, 01102, 01013, 10012\}$$
$$C_2 = \{10101, 01100, 01011, 10010\}.$$

Si sustituimos cada componente por un único nudo, se obtiene un grafo con dos nudos y un arco que une C_1 a C_2. Dado que C_2 no posee otros nudos sucesores, si la evolución de la RdP es tal que se alcanza uno de los marcados que pertenecen a C_2, las secuencias de evolución posibles harán que el marcado de la red pertenezca indefectiblemente a C_2. En estas condiciones, para que la RdP sea viva es necesario y suficiente que C_2 comprenda todas las transiciones, puesto que si faltase alguna, ésta no sería disparable y, por lo tanto, no viva. Dicho de otro modo, habida cuenta que los nudos sucesores de los que pertenecen a C_2 están contenidos en C_2, esta componente fuertemente conexa del grafo de marcados representa un *subconjunto final de marcados*, y la vivacidad de la red se garantiza al poderse disparar todas las transiciones.

Se debe observar que el razonamiento anterior es independiente de las transiciones que se disparan dentro de la componente fuertemente conexa C_1. En efecto, esto es posible porque para cualquier marcado perteneciente a C_1 existe una secuencia de disparo de transiciones que hace que el marcado alcanzado pertenezca a C_2.

---

† La terminología básica sobre grafos utilizada en este capítulo se presenta en el anexo 2.

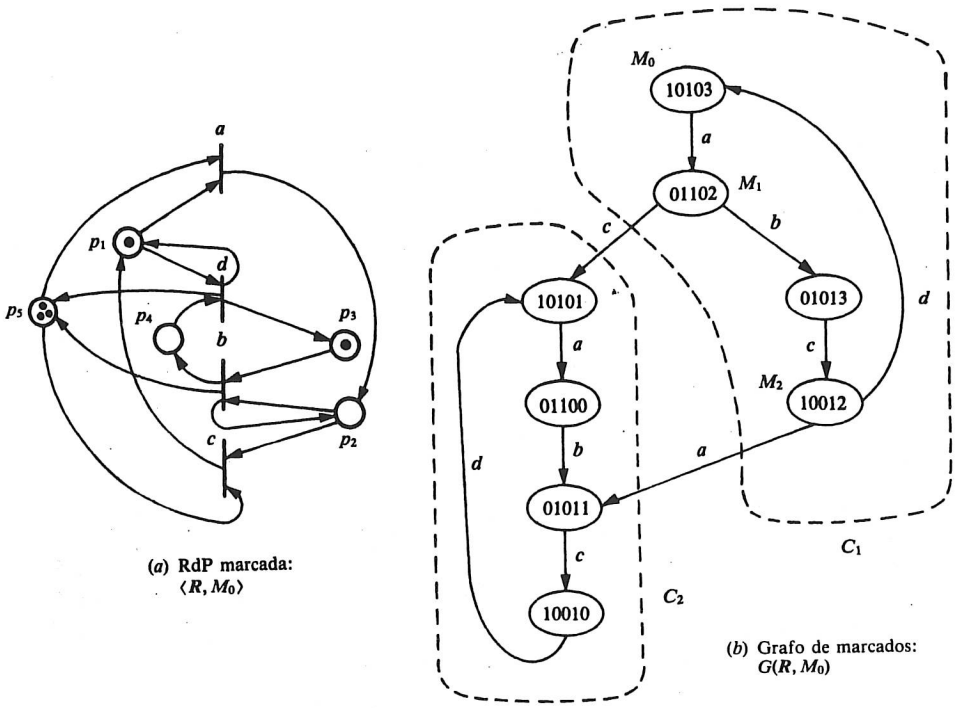


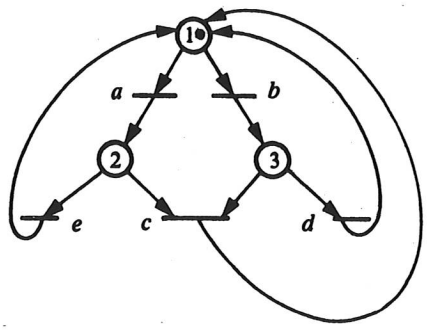
Figura 4.12. RdP marcada (viva y no cíclica) y grafo de marcados.

DEMOSTRACIÓN (proposición 4.2). La justificación del primer enunciado ya la hemos visto antes (proposición 4.1.1): corresponde al caso en que la red presenta un bloqueo total (figuras 4.9 y 4.11). La justificación del segundo enunciado es también inmediata. Si existiera una componente fuertemente conexa de $G(R, M_0)$ que no estuviese etiquetada con t_k, y si la evolución en el grafo no permitiera abandonar dicha componente, t_k no podría ser disparada; entonces t_k sería no viva. El razonamiento inverso es evidente. □

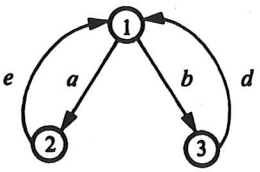
La RdP de la figura 4.13 no es viva ya que $G(R, M_0)$, que es una componente fuertemente conexa, no contiene la transición c. Además, como no existe nudo terminal, $\langle R, M_0 \rangle$ es parcialmente viva (no se bloquea).

Para proponer un algoritmo eficiente que determine la vivacidad de una red se deben considerar las dos observaciones siguientes:

1) El cálculo de la *componente fuertemente conexa* (CFC) con el nudo inicial, etiquetado con M_0, se puede realizar sin más que propagar, a partir del mencionado nudo, las marcas « – » del algoritmo enunciado en §A.2.3. La CFC con el nudo etiquetado M_0 es el conjunto de nudos marcados « – ».



(*a*) $\langle R, M_0 \rangle$



(*b*) $G(R, M_0)$

Figura 4.13. RdP parcialmente viva y cíclica.

La anterior afirmación se justifica dado que al construirse $G(R, M_0)$ a partir de M_0, todos los nudos del grafo son alcanzables a partir de M_0. Dicho de otro modo, de acuerdo con la notación del §A.2.3, las marcas « + » alcanzarán a todos los nudos de $G(R, M_0)$.

2) Si $G(R, M_0)$ posee más de una CFC, entonces la CFC con el nudo inicial, etiquetado M_0, no puede representar un conjunto final de marcados y, por lo tanto, no es necesario considerarla para estudiar la vivacidad $\langle R, M_0 \rangle$.

La utilización de estas dos observaciones nos permite enunciar el siguiente algoritmo.

ALGORITMO: ANÁLISIS DE LA VIVACIDAD A PARTIR DEL GRAFO DE MARCADOS

a) Marcar « – » el nudo inicial (el etiquetado con M_0).
b) Marcar « – » todo nudo predecesor aún no marcado con « – » de un nudo previamente marcado « – ».
c) **si** todos los nudos del grafo están marcados « – »
  entonces • el grafo de marcados es una única CFC y, por consiguiente, la RdP será viva sii se han disparado todas las transiciones;
  si no • eliminar los nudos del grafo marcados con « – » (CFC con el nudo inicial):
    • **mientras que** exista algún nudo en el grafo **hacer**
      • calcular la CFC con un nudo (véase el algoritmo de §A.3.3);
      • **si** la CFC calculada, $\mathcal{C}_i$, contiene todos sus nudos sucesores (su espectro a cierre) y existe al menos una transición no disparada dentro de la CFC, **entonces** la RdP no es viva debido a $\mathcal{C}_i$
      • eliminar del grafo la CFC calculada.

EJERCICIO. Dedúzcase la vivacidad de la red de Petri de la figura 4.12*a* aplicando el algoritmo anterior a su grafo de marcados (figura 4.12*b*).

EJERCICIO. Estúdiese la vivacidad de las RdP que modelan los ejemplos del capítulo 2, §2.5.

4.4.2.2 *Análisis de la ciclicidad*

Proposición 4.3. Una RdP limitada para M_0 es cíclica para ese marcado inicial sii el grafo de marcados es fuertemente conexo. □

La justificación de este enunciado es evidente puesto que, por definición, en todo grafo fuertemente conexo se puede alcanzar cualquier nudo (en particular, el etiquetado por M_0) a partir de cualquier otro nudo (cualquier marcado alcanzable a partir de M_0).

La RdP de la figura 4.13 es cíclica porque su grafo de marcados es fuertemente conexo. La RdP de la figura 4.12 no es cíclica porque el grafo de marcados no es fuertemente conexo.

4.4.2.3 *Análisis de la conflictividad*

El estudio de la aparición de conflictos efectivos es inmediato. En el ejemplo de la figura 4.12 observaremos dos conflictos efectivos. Éstos corresponden a las decisiones de disparo entre:

1) b o c, estando p_2 marcada con una única marca [marcado $M_1 = (0\,1\,1\,0\,2)^T$].
2) a o d, estando p_1 marcada con una única marca [marcado $M_2 = (1\,0\,0\,1\,2)^T$].

En este punto es importante resaltar que la presencia de una decisión en $G(R, M_0)$ no implica la existencia de un conflicto efectivo. Así, se puede observar que incluso el grafo de marcados de un grafo de sincronización puede contener selecciones.

EJERCICIO. ¿Qué significa entonces una selección en $G(R, M_0)$? (*Sugerencia*: ténganse en cuenta las RdP de la figura 1.15 y sus grafos de marcados, figuras 1.10 y 1.11.)

4.4.2.4 *Otros análisis*

El análisis de la k-limitación de un lugar o de la exclusión mutua entre lugares se realiza sin más que considerar el conjunto de marcados alcanzables a partir de M_0 (etiquetas asociadas a los nudos del grafo de marcados).

En el ejemplo de la figura 4.12 se ve que

$$M(p_1) \leqslant 1, \quad M(p_2) \leqslant 1, \quad M(p_3) \leqslant 1, \quad M(p_4) \leqslant 1, \quad M(p_5) \leqslant 3,$$

luego la red marcada es 3-limitada.

Considerando el mismo ejemplo, podemos representar tabularmente las exclusiones mutuas (tabla 4.1). La exclusión mutua entre el marcado de dos lugares es una relación de *incompatibilidad*.

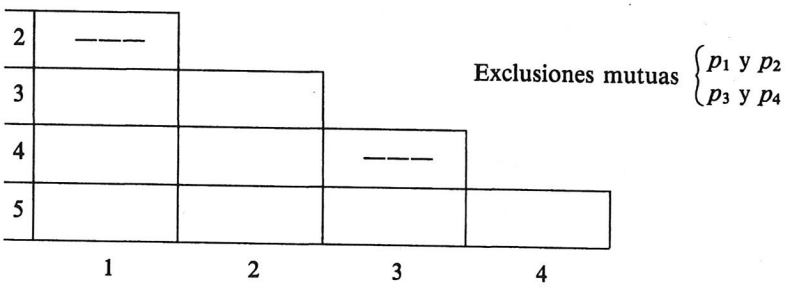


Tabla 4.1 Exclusiones mutuas entre los marcados de la RdP de la figura 4.12.

También se puede realizar el estudio de los avances sincrónicos, aunque su presentación es más compleja. Éste será abordado por métodos más adecuados en §4.7 No obstante, podemos extraer algunas conclusiones inmediatamente; por ejemplo, en el $G(R, M_0)$ (figura 4.13) existen dos circuitos que permiten afirmar que [siendo $AV(R, M_0; x, y) = AV(x, y)$]:

$$a)\ AV(a,b) = AV(a,d) = AV(b,a) = AV(b,e) = AV(d,a) = AV(d,e) = AV(e,b) = AV(e,d) = \infty$$
$$b)\ AV(a,e) = AV(b,d) = 1$$
$$c)\ AV(e,a) = AV(d,b) = 0.$$

4.4.3 Crítica de los métodos de análisis por enumeración

La naturaleza fuertemente combinatoria de estos métodos de análisis puede restringir su utilización práctica, incluso si las redes poseen pocos lugares y/o transiciones.

En resumen, los métodos por enumeración son potentes porque permiten analizar, más o menos directamente, todas las propiedades características del buen funcionamiento de una RdP. El problema fundamental que plantean reside en su complejidad operativa, lo que los hace a veces difícilmente utilizables. Para atenuar este inconveniente se han desarrollado las técnicas de reducción.

4.5 ANÁLISIS POR REDUCCIÓN

Los métodos de análisis que consideramos a continuación se basan en la idea de reducir la complejidad de la red inicial, pero de forma tal que, al llevar a cabo la reducción, se preserven las propiedades que se desea analizar.

Las técnicas de reducción permiten eliminar o sustituir transiciones o/y lugares de manera que no resulten afectadas las propiedades objeto del análisis del comportamiento dinámico. Evidentemente, las reducciones que vamos a estudiar no preservan la relación funcional evento-acción que describe la red interpretada inicial. El estudio de reducciones que preservan la relación evento-acción fue abordado en el capítulo 3, dedicado a la simplificación de una descripción.

Desde un punto de vista conceptual, el proceso de reducción de una RdP marcada permite agrupar secuencias de operaciones en el modelo del sistema concebido y sus-

tituirlas por una única «macroacción» en el modelo reducido. Está claro, pues, que el proceso de reducción de un modelo es inverso del proceso de descripción descendente (mediante refinamientos sucesivos). En efecto, en un proceso de descripción descendente (*top-down*) se parte de una descripción general (a nivel de macroacciones) que se va refinando mediante la descomposición de macroacciones en acciones más simples.

La íntima relación entre reducción y refinamiento hace que los resultados de ambos métodos sean prácticamente «trasvasables» de uno a otro.

En este apartado consideramos tres reglas de reducción. Una de ellas tiene carácter no estrictamente local, puesto que permite reducir a un lugar ciertas subRdP. Las otras dos son reglas de aplicación local (sustitución de un lugar y eliminación de una transición). Estas tres reglas de reducción se complementan con la regla sobre *eliminación de un lugar implícito* (§3.3). En cualquier caso, siempre es posible definir nuevas reglas de reducción. El conjunto de reglas que presentaremos pretende establecer un compromiso entre *completitud* y *utilidad*.

El lector que en una primera lectura desee considerar directamente los casos más sencillos de las reglas de reducción de redes ordinarias puede pasar al §4.5.4.2*b*. En éste y en la figura 4.26 se presentan casos particulares de las reglas que se estudian a continuación. Como es fácil intuir sobre estos casos particulares, tanto la vivacidad como la limitación se preservan al aplicar las reglas de reducción.

4.5.1 Reducción de una subRdP a un lugar ($\mathcal{R}_1$)

La regla de reducción que estudiamos en este apartado contiene ciertas condiciones suficientes que, si son satisfechas por la estructura de una subRdP, garantizan que ésta es reducible a un lugar preservando las propiedades de vivacidad y limitación. La aplicación de esta regla es especialmente eficaz en el caso de que las evoluciones secuenciales, incluso con alternativas, sean numerosas dentro de la evolución total de la RdP.

4.5.1.1 *Regla de reducción*

La reducción de una subRdP a un lugar es posible si la subRdP presenta un comportamiento global «idéntico» al del lugar. Así, de forma intuitiva podremos enunciar que una subRdP podrá ser reducida a un lugar si:

1) no crea ni destruye marcas;
2) la vivacidad de las transiciones de la subRdP está implicada por la vivacidad de otras transiciones externas a la subRdP;
3) todas las marcas existentes en la subRdP se pueden utilizar en el disparo de cualquiera de las transiciones que definen la frontera con el resto de la RdP.

Para formalizar estas condiciones vamos a introducir una serie de conceptos.

Sea la RdP $\bar{\boldsymbol{R}} = \langle \bar{P}, \bar{T}, \bar{\alpha}, \bar{\beta} \rangle$ una subRdP de $\boldsymbol{R} = \langle P, T, \alpha, \beta \rangle$. En la definición que sigue se expresa una condición suficiente para que una subRdP no cree ni destruya marcas.

Definición 4.15. La subRdP $\bar{\boldsymbol{R}}$ es *potencialmente reducible a un lugar* si el peso de sus arcos es la unidad y sus transiciones poseen, un único lugar de entrada y único lugar de salida. □

Una subRdP potencialmente reducible a un lugar es una máquina o grafo de estados en la que ciertos lugares pueden no tener transiciones de entrada o de salida.

Definición 4.16. Los lugares $\bar{p} \in \bar{P}$ que en R poseen transiciones de entrada [de salida] que no pertenecen a la subRdP, $t \notin \bar{T}$, serán denominados *lugares ascendientes* [*descendientes*] de la subRdP $\bar{R}$. □

En la figura 4.14 se presenta un ejemplo típico de subRdP potencialmente reducible a un lugar. Se puede observar que la intersección de los conjuntos de lugares ascendientes, $\{p_1, p_2\}$, y descendientes, $\{p_2, p_5, p_6\}$, no tiene por qué ser nula (p_2 en la figura 4.14).

Observación. Si se consideran RdP ordinarias, todo componente conexo que no posea transiciones con más de un lugar de entrada y/o de salida es potencialmente reducible a un lugar.

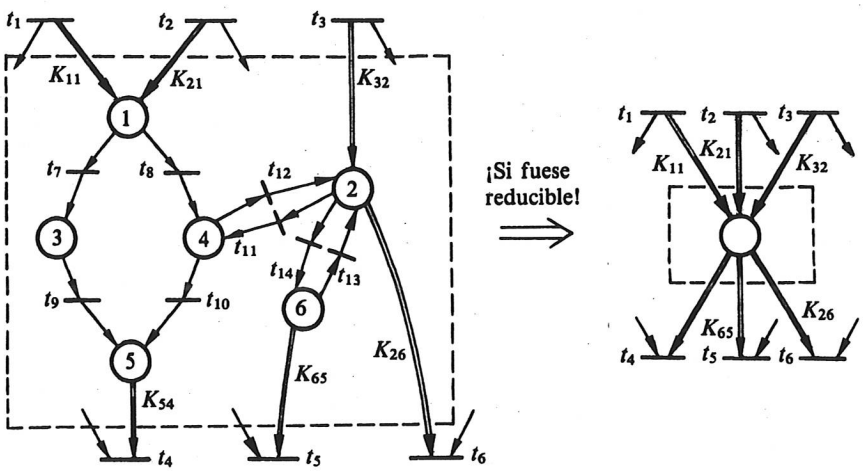


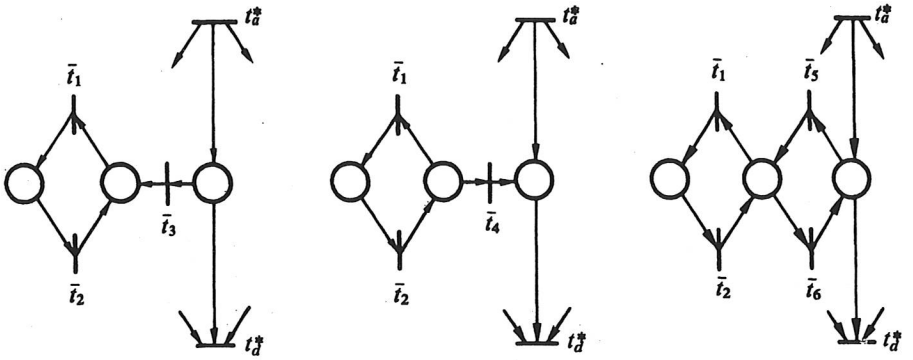
Figura 4.14. La subRdP $\bar{R} = (\{p_1, \ldots, p_6\}, \{t_7, \ldots, t_{14}\})$ es potencialmente reducible a un lugar. Si $\bar{R}$ fuese reducible a un lugar, se obtendría la red de la derecha.

Para que una subRdP sea reducible a un lugar, debe cumplir las condiciones que permiten preservar la vivacidad mediante la operación de reducción (condiciones 2 y 3 enunciadas al comienzo de §4.5.1.1). A continuación caracterizaremos una clase de subRdP reducibles a un lugar. El lugar resultante de la reducción se define como *macrolugar*.

Definición 4.17. Se llama *macrolugar*, $P_{\bar{R}}$, al lugar que se obtiene al reducir una subRdP, $\bar{R}$, que verifique las condiciones siguientes:

c_1) $\bar{R}$ es potencialmente reducible a un lugar;
c_2) $\forall \bar{p} \in \bar{P}$ existe al menos un camino que parte de un lugar ascendiente de $\bar{R}$ y llega a $\bar{p}$;
c_3) $\forall \bar{p} \in \bar{P}$ existen caminos en $\bar{R}$ que lo unen a los diferentes lugares descendientes de $\bar{R}$. □

Las tres condiciones enunciadas en la definición 4.17 permiten verificar las condiciones intuitivas presentadas en el primer párrafo de §4.5.1.1. Según esta definición, la subRdP potencialmente reducible a un lugar de la figura 4.14 no es reducible porque no existe un camino que vaya desde p_5 hasta $\{p_2, p_6\}$. Las subRdP de la figura 4.15*a* y *b* tampoco son reducibles a un lugar. (Estúdiese la causa.)



(*a*) $\{c_1, c_2\}$ (*b*) $\{c_1, c_3\}$ (*c*) $\{c_1, c_2, c_3\}$

Figura 4.15. Conjunto de subRdP que facilita la comprensión del significado de las tres condiciones de la definición 4.17. (*Nota*: se indican las condiciones que verifica cada subRdP.)

El macrolugar $P_{\bar{R}}$ que representa la reducción de $\bar{R}$, subRdP reducible a un lugar, se determina de acuerdo con las reglas que siguen:

1) $\forall t_j \in T \begin{cases} \alpha(P_{\bar{R}}, t_j) = \sum\limits_{\bar{p}_i \in \bar{P}} \alpha(\bar{p}_i, t_j) \\ \beta(t_j, P_{\bar{R}}) = \sum\limits_{\bar{p}_i \in \bar{P}} \beta(t_j, \bar{p}_i), \end{cases}$

es decir, las transiciones de entrada y de salida de $P_{\bar{R}}$ son aquellas que no pertenecen a $\bar{R}$, pero que, sin embargo, son transiciones de entrada y de salida de $\bar{R}$.

2) $M_0(P_{\bar{R}}) = \sum\limits_{\bar{p}_i \in \bar{P}} M_0(\bar{p}_i)$.

es decir, $P_{\bar{R}}$ posee como marcado inicial la suma de las marcas que poseen los lugares de $\bar{R}$, $\bar{p}_i \in \bar{P}$.

En estas condiciones es fácil verificar que, si a la red original $\langle \boldsymbol{R}, M_0 \rangle$ se le añade el lugar $P_{\bar{R}}$,ésta se transforma en la red $\langle \boldsymbol{R}^*, M_0^* \rangle$ equivalente a la anterior, puesto que $P_{\bar{R}}$ será un lugar implícito (§3.3).

La subRdP de la figura 4.15*c* es reducible a un lugar, el cual tendrá sólo una transición de entrada, t_a^*, y una transición de salida, t_d^*. Las transiciones $\bar{t}_1$, $\bar{t}_2$, $\bar{t}_5$ y $\bar{t}_6$ no figurarán en la red reducida puesto que pertenecen a $\bar{R}$.

A continuación presentaremos las propiedades que preserva la regla de reducción considerada y expondremos algoritmos que permiten determinar si, de acuerdo con las condiciones dadas, una subRdP es reducible a un lugar o no lo es.

4.5.1.2 *Propiedades que preserva la regla de reducción*

Sea la red marcada original $\langle R, M_0 \rangle$ y sea $\langle R_r, M_{0r} \rangle$ la red marcada obtenida mediante la reducción a un lugar de una subRdP reducible.

Propiedad fundamental de la reducción de una subRdP a un lugar (proposición 4.4). $\langle R, M_0 \rangle$ es viva y limitada sii $\langle R_r, M_{0r} \rangle$ es viva y limitada. □

Una justificación de este resultado está implícita en el razonamiento realizado al comienzo de §4.5.1.1, por lo que se puede omitir la lectura de la demostración.

DEMOSTRACIÓN. Para demostrar la proposición 4.4 basta comprobar que la red reducida $\langle R_r, M_{0r} \rangle$ es viva y limitada sii la red $\langle R^*, M_0^* \rangle$, obtenida al añadir a $\langle R, M_0 \rangle$ el macrolugar (que será un lugar implícito), es viva y limitada.

(1) $\langle R^*, M_0^* \rangle$ viva $\Leftrightarrow$ $\langle R_r, M_{0r} \rangle$ viva.

(1.1) $\langle R^*, M_0^* \rangle$ viva $\Rightarrow$ $\langle R_r, M_{0r} \rangle$ viva.

La transformación de $\langle R^*, M_0^* \rangle$ en $\langle R_r, M_{0r} \rangle$ se realiza eliminando $\bar{R}$ en $\langle R^*, M_0^* \rangle$. Hay que demostrar que la eliminación de $\bar{R}$, prescindiendo de las transiciones $\bar{t} \in \bar{T}$, no crea nuevas secuencias de disparos. Esto se puede garantizar gracias a las condiciones 1 y 3 de la definición de macrolugar. En efecto, éstas permiten que en $\bar{R}$ todas las marcas alojadas, y sólo éstas, puedan ser encaminadas hacia cualquiera de sus transiciones de salida, comportamiento que es idéntico al del macrolugar $P_{\bar{R}}$. Por consiguiente, la eliminación de $\bar{R}$ no crea nuevas secuencias de disparos.

(1.2) $\langle R_r, M_{0r} \rangle$ viva $\Rightarrow$ $\langle R^*, M_0^* \rangle$ viva.

La transformación de $\langle R_r, M_{0r} \rangle$ en $\langle R^*, M_0^* \rangle$ se realiza insertando $\bar{R}$ en la primera. Para que la implicación anterior se verifique basta con que:

a) La vivacidad de las transiciones $\bar{t} \in \bar{T}$ sea implicada por la vivacidad de las transiciones externas a $\bar{R}$, $t_e \in T_e = T - \bar{T}$. Esta condición se cumple gracias a las condiciones 1 y 2 de la definición de macrolugar. En efecto, de acuerdo con dichas condiciones, si las transiciones de entrada de los lugares ascendientes de $\bar{R}$ son vivas, todas las transiciones de $\bar{R}$ son vivas.

b) La inserción de $\bar{R}$ no introduzca restricciones sobre el disparo de las transiciones de salida de sus lugares descendientes. El razonamiento es similar al realizado en el punto 1.1.

En resumen, $\langle R, M_0 \rangle$ viva $\Leftrightarrow$ $\langle R^*, M_0^* \rangle$ viva $\Leftrightarrow$ $\langle R_r, M_{0r} \rangle$ viva.

(2) $\langle R^*, M_0^* \rangle$ limitada $\Leftrightarrow$ $\langle R_r, M_{0r} \rangle$ limitada.

Puesto que los macrolugares, son lugares implícitos en $\langle R^*, M_0^* \rangle$, se puede establecer:

$$\langle R^*, M_0^* \rangle \text{ } k\text{-limitada} \Rightarrow \langle R_r, M_{0r} \rangle \text{ } k\text{-limitada.}$$

Ahora bien, dada la condición c_3 (def. 4.17), el límite del marcado de $P_{\bar{R}}$ y de cualquier $\bar{p}_i$ descendiente de $\bar{R}$ coinciden. Es decir, el límite de $\langle R, M_0 \rangle$ será idéntico al de la red $\langle R^*, M_0^* \rangle$. En conclusión, podemos afirmar:

$$\langle R, M_0 \rangle \text{ } k\text{-limitada} \Leftrightarrow \langle R_r, M_{0r} \rangle \text{ } k\text{-limitada.} \quad \square$$

Corolario 4.1. Si R es una RdP ordinaria, $\langle R, M_0 \rangle$ es viva y binaria sii $\langle R_r, M_{0r} \rangle$ es viva y binaria. $\square$

Si se desea estudiar el límite de un lugar, las exclusiones mutuas entre lugares y la ciclicidad de una RdP marcada, basta con reducir la red original sin eliminar los lugares cuya consideración interese (en el caso de la ciclicidad, los lugares marcados para M_0).

4.5.1.3 *Obtención de los macrolugares*

Para obtener los macrolugares se puede proceder siguiendo tres etapas:

1) *Eliminación* de las *transiciones* Y de la red:

Esta operación consiste en la supresión en $\langle R, M_0 \rangle$ de todas las transiciones de tipo Y con todos sus arcos de entrada y de salida (figura 4.16). Tras esta operación, se obtienen varias componentes conexas, según la terminología específica de los grafos (Anexo 2).

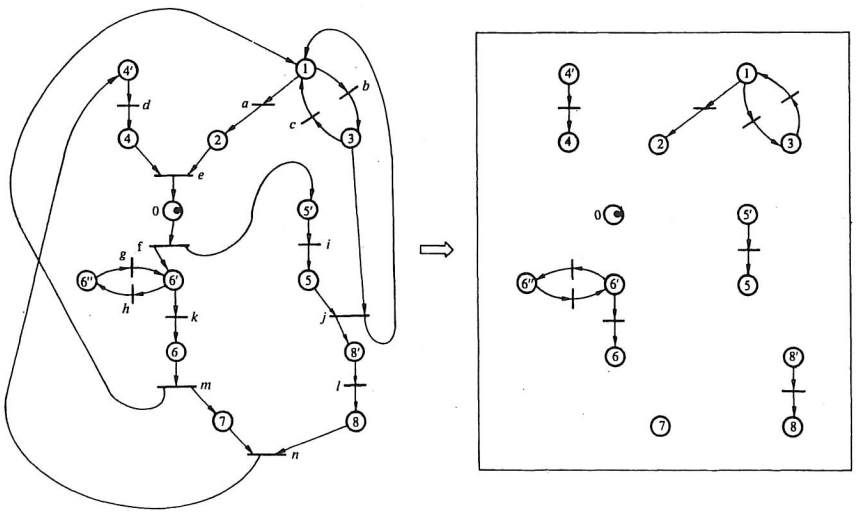


Figura 4.16. Supresión de los nudos Y.

2) *Partición* de cada componente conexo en máquinas de estado:

Esta operación sólo es necesaria si $\boldsymbol{R}$ es una RdP generalizada. Si $\boldsymbol{R}$ es una RdP ordinaria, todos los componentes conexos son máquinas de estado y, por consiguiente, subRdP potencialmente reducibles a un lugar. (*Nota*: recuérdese que las ME no tienen por qué estar monomarcadas.)

3) *Partición* de cada máquina de estados en subRdP reducibles a un macrolugar.

Desde un punto de vista algorítmico, ésta es quizás la etapa más interesante.

Puesto que toda partición de una máquina de estados permite obtener un conjunto de máquinas de estados, se puede proceder a la partición según las condiciones 2 y 3 de la definición 4.17.

ALGORITMO PARA LA VERIFICACIÓN DE LA CONDICIÓN 2 DE LA DEFINICIÓN 4.17

a) Marcar « + » los lugares ascendientes de un componente conexo $\bar{R}$.
b) **Mientras que** exista un lugar que pueda ser marcado « + »
hacer marcar « + » todo lugar aún no marcado « + » que sea lugar de salida de una transición de salida de un lugar previamente marcado « + ».
c) **Si** existe al menos un lugar $\bar{p} \in \bar{P}$ no marcado « + »
entonces $\bar{R}$ no es estructuralmente viva
si no se verifica la condición 2 de la definición 4.17.

Como fácilmente se comprueba, este algoritmo sólo sigue directamente la condición que pretende determinar, empleando una técnica clásica en grafos (véase el anexo 2, §A.2.3).

ALGORITMO PARA LA VERIFICACIÓN DE LA CONDICIÓN 3 DE LA DEFINICIÓN 4.17

a) **Para todo** lugar $\bar{p}_i$ que sea descendiente de $\bar{R}$
hacer marcar el lugar con « $-i$» (es decir, con « − » su indicativo o subíndice)
b) **mientras que** algún lugar pueda ser marcado negativamente
hacer marcar « $-j$» todo lugar aún no marcado « $-j$» que sea lugar de entrada de una transición de entrada de un lugar previamente marcado « $-j$».
c) Cada una de las subRdP cuyos lugares han sido etiquetados por un mismo conjunto de marcas negativas verifica la condición 3 de la definición 4.17.

Como se puede comprobar, este segundo algoritmo no hace más que determinar los conjuntos de lugares que pueden enviar marcas a un mismo subconjunto de lugares descendientes de $\bar{R}$, $\{\bar{p}_i\}$. A modo de observación, es interesante señalar que la condición 3 permite establecer una relación de equivalencia entre los lugares de $\bar{R}$: dos lugares son *equivalentes* sii a partir de ellos se puede alcanzar el mismo subconjunto de lugares descendientes de $\bar{R}$.

EJEMPLO. Sea la subRdP potencialmente reducible de la figura 4.14:

- la aplicación del primer algoritmo permite marcar «+» todos los lugares, y, por consiguiente, se verifica la condición 2;
- la aplicación del segundo algoritmo permite:
  —*Paso a*: marcar p_2 con «−2», p_5 con «−5» y p_6 con «−6».
  —*Paso b*: marcar $\{p_1, p_2, p_4, p_6\}$ con {«−2», «−5», «−6»} y $\{p_3, p_5\}$ con {«−5»}.
  En la figura 4.17 se representa la reducción posible.

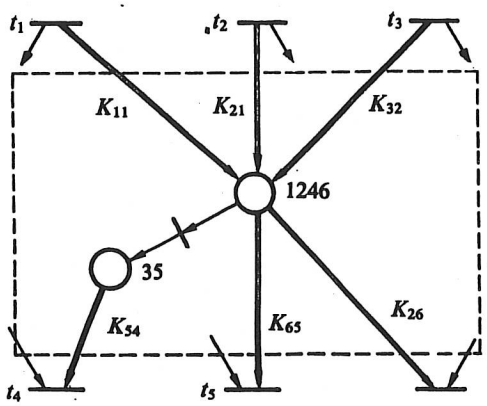
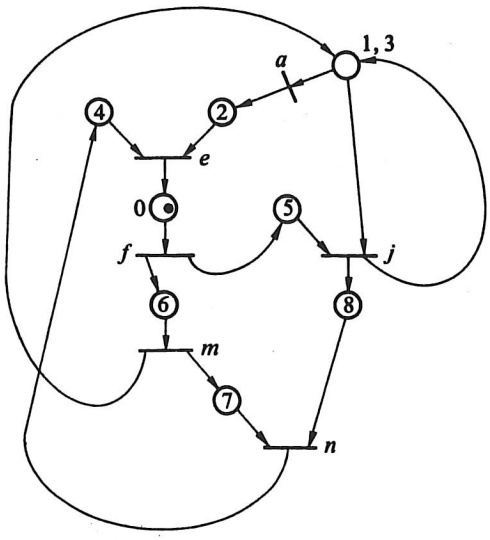
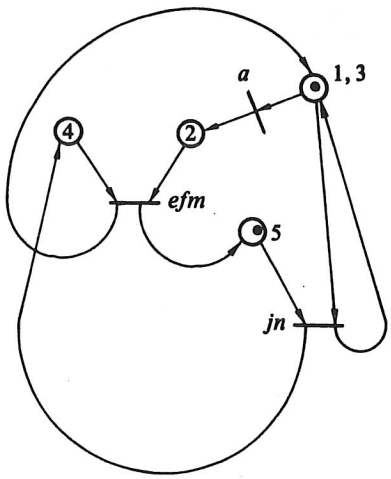


Figura 4.17. La subRdP (figura 4.14) no es reducible a un único lugar.



(*a*) Reducción de las subRdP a Macrolugares.



(*b*) Reducciones posteriores, incluyendo sustituciones de lugares y eliminación de un lugar implícito.

Figura 4.18. Reducciones de la figura 4.16.

EJERCICIO. Compruébese que al reducir la red de la figura 4.16 se obtiene la de la figura 4.18*a*.

EJERCICIO. Compruébese que la red de la figura 1.15 es viva y binaria. (*Nota*: independientemente de N, número de carros, se obtiene un grafo de marcados con dos nudos al aplicar la regla de reducción de subRdP a lugares y la de eliminación de lugares implícitos.)

4.5.2 Sustitución de un lugar ($\mathfrak{R}_2$)

La aplicación de esta nueva regla de reducción nos conducirá a la eliminación de lugares que no están implícitos. En cualquier caso, esta regla preserva las propiedades de vivacidad y de limitación. Como veremos, es una regla de aplicación local.

A partir de la definición de lo que denominaremos lugar sustituible, presentaremos la operación de sustitución. Por último, enunciaremos las propiedades que se mantienen.

4.5.2.1 *Lugar sustituible y operación de sustitución*

El objetivo de la sustitución de un lugar es eliminarlo de la RdP, aunque manteniendo de forma condensada todas las secuencias de disparo de la RdP original. Un lugar, p, más el conjunto de sus transiciones de entrada y de salida, será sustituido por un conjunto de transiciones. Para que se mantengan todas las secuencias de disparo de la red original, cada una de las transiciones que se introduzca deberá representar una secuencia (o un conjunto de secuencias) de disparo de la red original. Estas secuencias estarán formadas exclusivamente por transiciones de entrada y de salida del lugar que se sustituye. Por otro lado, las secuencias serán tales que cualquier marcado en el que p contenga marcas deberá ser un marcado intermedio entre dos en los que p estará desmarcado. Si consideramos la figura 4.19, observaremos que p puede ser marcado al disparar t_1 o t_2 y, además, puede ser desmarcado al disparar t_3 o t_4. La eliminación de p y de $\{t_1, t_2, t_3, t_4\}$ exige la introducción de transiciones que representen las secuencias de marcado-desmarcado de p: $\{t_1t_3, t_1t_4, t_2t_3, t_2t_4\}$.

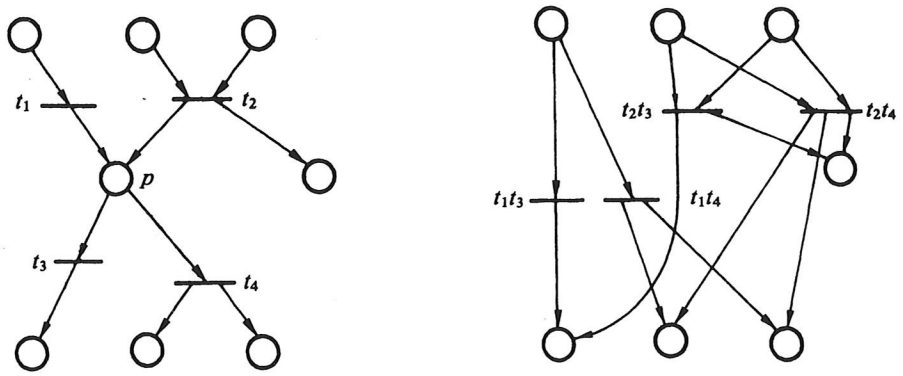


Figura 4.19. Sustitución del lugar p.

El razonamiento anterior es válido sólo si se pueden determinar las secuencias de marcado-desmarcado de p, independientemente de la evolución del marcado en el resto de la RdP (localidad de la transformación). Para garantizar esta propiedad, se introducen las condiciones 1 y 2 de la definición de lugar sustituible (definición 4.18). La condición 3 se introduce para preservar la limitación.

Definición 4.18. El lugar p es *sustituible* en la red marcada $\langle \boldsymbol{R}, M_0 \rangle$ si:

1) p es el único lugar de entrada de sus transiciones de salida y, además, el peso del arco correspondiente es la unidad; es decir, $\forall t_k \in p^{\cdot}$ se verifica:

   (1.1) ${}^{\cdot}t_k = p$

   (1.2) $\alpha(p, t_k) = 1$;

2) no existe transición que sea simultáneamente transición de entrada y de salida de p: ${}^{\cdot}p \cap p^{\cdot} = \emptyset$.

3) al menos una transición de salida de p, $t_k \in p^{\cdot}$, posee uno o más lugares de salida: $\exists t_k \in p^{\cdot}$ tal que $t_k^{\cdot} \neq \emptyset$. □

De acuerdo con esta definición, la figura 4.20 presenta el esquema general de un lugar sustituible. En la RdP de la figura 4.18*a* sólo se pueden sustituir los lugares p_0 y p_6. Los restantes no cumplen la condición 1; además, el lugar p_{13} tampoco cumple la condición 2.

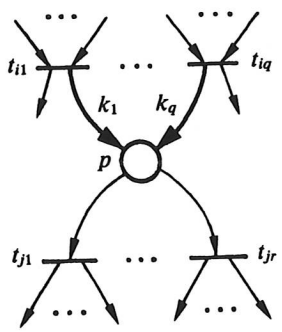
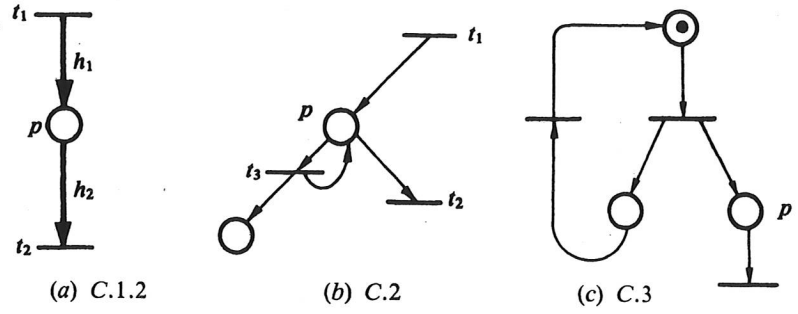


Figura 4.20. Esquema general de un lugar sustituible.

Antes de continuar con la operación de sustitución comentaremos las condiciones de la definición 4.18. La condición 1.1 es necesaria porque, de lo contrario, sería válida la eliminación de p_2 en la red de la figura 4.18. La mencionada red marcada es no viva pero, si por error se sustituye p_2, se convierte en una red viva; es decir, no se preservaría la vivacidad.

Veamos intuitivamente el interés de la condición 1.2. Consideremos la subRdP de la figura 4.21*a*. Si h_1 no es múltiplo de h_2 ($\nexists a \in \mathbb{N}$ tal que $h_1 = ah_2$) resulta imposible encontrar un número de disparos de t_2 que desmarque p después de un disparo de t_1, y, por lo tanto, el lugar p no podrá ser sustituido localmente. Evidentemente la condición 1.2 (suficiente) resuelve el problema al tomar $h_2 = 1$.



(*a*) *C*.1.2 (*b*) *C*.2 (*c*) *C*.3

Figura 4.21. Estudio de las condiciones de la definición de lugar sustituible.

La condición 2 de la definición 4.18 impide la sustitución de lugares como el de la figura 4.21*b*. En efecto, si t_1 es disparada, existirá un número infinito de secuencias de disparo posibles que corresponderán a otros tantos disparos de t_3. Por último, la condición 3 de la definición 4.18 impide la sustitución de lugares como el de la figura 4.21*c*. En efecto, p es el único lugar no limitado (la RdP es no limitada) y, por consiguiente, su eliminación hace que la RdP reducida sea limitada (binaria).

Como anunciábamos anteriormente, la operación de sustitución del lugar p permite su eliminación y, además, reemplaza las transiciones $\cdot p$ y $p\cdot$ por un conjunto de transiciones, S. Cada transición $s_j \in S$ representará una secuencia elemental de marcado-desmarcado del lugar p. Es fácil comprobar que al proceder de esta manera conservamos en el grafo de marcados de la red reducida, $G(R_r, M_{0r})$, «los caminos» del grafo de marcados de la red original, $G(R, M_0)$. Evidentemente, en $G(R_r, M_{0r})$ no aparecerán los marcados en los que p contenía, al menos, una marca.

La figura 4.22 presenta la aplicación de la operación de sustitución al lugar p. La sustitución de p_0 y p_6 (figura 4.18*a*) conduce a una RdP en la que se confunden las transiciones e, f y m (figura 4.18*b*).

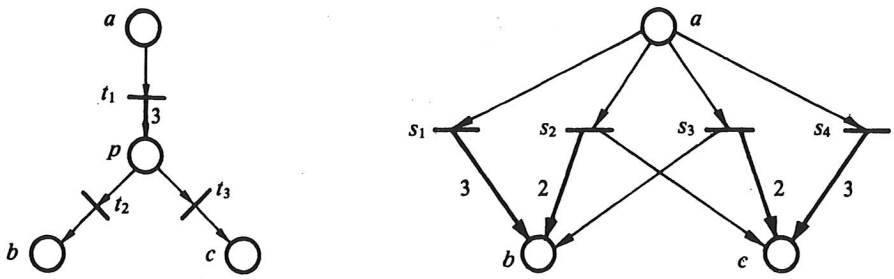


Figura 4.22. Sustitución del lugar p.

El número de transiciones que hemos de crear después de una operación de sustitución, $|S|$, se calcula fácilmente. Supongamos que existe una única transición de

entrada al lugar p que le introduce h marcas. Supongamos también que p tiene r transiciones de salida, $r = |p\cdot|$. El número de transiciones que tenemos que crear corresponde al número de formas en que se pueden distribuir h bolas en r casillas (combinaciones con repetición), es decir,

$$|S| = W_r^h = \frac{(h + r - 1)!}{h!(r - 1)!} \qquad \begin{cases} h \in \mathbb{N}^+ \\ r \in \mathbb{N}^+ \end{cases}$$

Observación.

$$W_r^1 = r,\ W_1^h = 1 \text{ y } W_2^h = h + 1. \qquad \square$$

Por último, si existen q transiciones de entrada a p, se han de crear

$$|S| = \sum_{i=1}^{q} W_r^{h_i} \quad \text{transiciones.}$$

En el proceso de sustitución del lugar p éste queda eliminado, así como las $q + r$ transiciones originales, $\cdot p \cup p\cdot$. De este modo, si los h_i son unitarios, $h_i = 1$ (en toda RdP ordinaria ocurre esto), se introducen qr transiciones. Puesto que se eliminan $q + r$, globalmente se añaden $qr - (q + r) = q(r - 1) - r = (q - 1)r - q$, cantidad que para $q = 1$ o/y $r = 1$ es, evidentemente, negativa. Es decir, en estos casos (los más frecuentes en la práctica), al eliminar un lugar también se reduce el número de transiciones de la red.

En conclusión, la aplicación de esta regla de reducción disminuye el número de lugares, aunque puede aumentar el número de transiciones de la red. No obstante, esto no resulta ser un inconveniente si se consideran las distintas secuencias de disparo posibles (caminos del grafo de marcados). Así pues, si en el ejemplo de la figura 4.22 el lugar «a» contiene una única marca, existen ocho secuencias de disparo posibles en la RdP original, y sólo cuatro en la RdP reducida. En el cuadro siguiente establecemos su correspondencia:

$\{\sigma_j\} \equiv$ secuencias en $\langle \mathbf{R}, M_0 \rangle$		$\{S_j\} \equiv$ representación de las $\{\sigma_j\}$ en $\langle \mathbf{R}_r, M_{0r} \rangle$
$\sigma_1 = t_1t_2t_2t_2$	⟶	S_1
$\sigma_2 = t_1t_2t_2t_3$ $\sigma_3 = t_1t_2t_3t_2$ $\sigma_4 = t_1t_3t_2t_2$	⟶	S_2
$\sigma_5 = t_1t_2t_3t_3$ $\sigma_6 = t_1t_3t_2t_3$ $\sigma_7 = t_1t_3t_3t_2$	⟶	S_3
$\sigma_8 = t_1t_3t_3t_3$	⟶	S_4

4.5.2.2 *Consecuencias de la regla de reducción*

Sea $\langle \boldsymbol{R}_r, M_{0r} \rangle$ la red obtenida al sustituir p en $\langle \boldsymbol{R}, M_0 \rangle$.

La definición de un lugar sustituible conduce, naturalmente, a enunciar que:

$\langle \boldsymbol{R}, M_0 \rangle$ k-limitada $\Rightarrow$ $\langle \boldsymbol{R}_r, M_{0r} \rangle$ k-limitada
$\langle \boldsymbol{R}_r, M_{0r} \rangle$ k-limitada $\Rightarrow$ $\langle \boldsymbol{R}, M_0 \rangle$ k-limitada.

La primera implicación es evidente. La segunda se puede demostrar, puesto que el lugar que se sustituye, p, tiene un límite inferior o igual al de todos los lugares de salida de sus transiciones de salida (de acuerdo con la condición 3 de la definición 4.18, existe al menos uno).

Por otro lado, de acuerdo con la definición de la operación de sustitución, toda transición de $\boldsymbol{R}_r$ representa:

—la misma transición de la red original $\boldsymbol{R}$, o bien
—una secuencia de disparos de transiciones de $\boldsymbol{R}$ que marca y desmarca al lugar sustituido,

de donde $\langle \boldsymbol{R}, M_0 \rangle$ viva $\Leftrightarrow$ $\langle \boldsymbol{R}_r, M_{0r} \rangle$ viva.

En resumen (**proposición 4.5**), $\langle \boldsymbol{R}, M_0 \rangle$ es k-limitada y viva sii $\langle \boldsymbol{R}_r, M_{0r} \rangle$ es k-limitada y viva. □

Corolario 4.2. $\langle \boldsymbol{R}, M_0 \rangle$ es conforme sii $\langle \boldsymbol{R}_r, M_{0r} \rangle$ es conforme. □

Antes de ofrecer un ejemplo de reducción de una red debemos señalar que hasta ahora hemos supuesto que los lugares que se trataba de sustituir no estaban marcados. Si p estuviera marcado, $M(p) = u$, y se fuera a sustituir, en lo sucesivo habría que considerar los W_r^u marcados iniciales, resultantes del disparo de las diferentes secuencias que desensibilizan las r transiciones de salida de p. Desde un punto de vista práctico, interesa resaltar que si p posee una única transición de salida, $r = 1$, entonces $W_1^u = 1$ y sólo habremos de considerar un marcado inicial único: el obtenido al disparar u veces consecutivas la transición de salida de p.

EJEMPLO. Sea de nuevo la RdP de la figura 4.18*a*. La sustitución de p_6 no plantea problemas; se obtiene la transición *fm*. La sustitución de p_0 exige que se dispare la transición *fm*, con lo que $M_{0r} = \{p_5, p_7, p_{13}\}$; se obtiene la transición *efm* (figura 4.18*b*). En esta situación, p_7 es un lugar implícito (está implicado por p_5 y p_8) y, por consiguiente, podemos proceder a su eliminación. Ésta nos permite sustituir p_8, con lo que se obtiene la transición *jn*. En la RdP obtenida (figura 4.18*b*) basta con disparar la transición *a* para comprobar que es no viva. En conclusión, la RdP de la figura 4.16 es no viva.

EJERCICIO. Considérese la RdP de la figura 4.23. ¿Qué reglas de reducción se pueden aplicar? ¿La red final es ordinaria o generalizada? ¿Se puede afirmar que, en determinados casos, las reglas $\mathfrak{R}_1$ (reducción de una subRdP a un lugar) y $\mathfrak{R}_2$ (sustitución de un lugar) no son aplicables indistintamente?

4.5.3 Eliminación de una transición ($\mathfrak{R}_3$)

Con las reglas de eliminación que estudiamos en este apartado, se preservan también la vivacidad y la k-limitación. Son reglas cuyo enunciado basta para comprender la afirmación anterior.

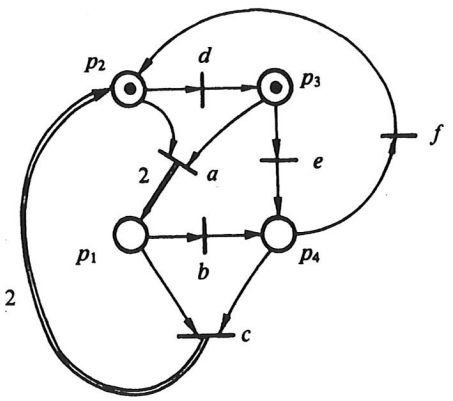


Figura 4.23. ¿Es reducible esta RdP?

Definición 4.19. Las *transiciones* t y t' son *idénticas* sii sus funciones de incidencia previa y posterior coinciden:

$$\forall p \in P \quad \begin{cases} \alpha(p,t) = \alpha(p,t') \\ \beta(t,p) = \beta(t',p). \end{cases} \qquad \square$$

La eliminación de una cualquiera de las dos transiciones, t o t', no alterará ni la vivacidad ni la limitación de la RdP.

Definición 4.20. La transición t es una *transición identidad* sii $\forall p \in P$ se verifica $\alpha(p,t) = \beta(t,p)$. $\square$

El disparo de una transición identidad no modifica el marcado de la RdP. La eliminación de una transición identidad, exige que se verifique que ésta no sea la única transición no viva. Para ello basta con encontrar una $t' \in T$; $t' \neq t$, tal que $\forall p \in P \quad \alpha(p,t') \geqslant \alpha(p,t)$ o/y $\beta(t',p) \geqslant \beta(t,p)$. En efecto, de acuerdo con la condición anterior si t' es viva, entonces también t será viva. En efecto, el disparo de t' requerirá más marcas que el de t $[\alpha(p,t') \geqslant \alpha(p,t)]$ o/y producirá un número de marcas suficiente para disparar t $[\beta(t',p) \geqslant \beta(t,p) = \alpha(p,t)]$.

La figura 4.24*c* presenta un caso en el que, si se desea preservar la vivacidad, no se puede eliminar la transición identidad, t.

4.5.4 Cuestiones adicionales sobre reducción y limitación de RdP

4.5.4.1 *Eliminación de lugares implícitos y limitación*

La eliminación de un lugar implícito no afecta la vivacidad de una RdP, pero puede ocurrir que el lugar implícito que eliminemos sea el lugar de mayor límite en la red y, por consiguiente, defina su k-limitación (véase la figura 4.25).

Sea k_0 el límite de $\langle \boldsymbol{R}_r, M_{0r} \rangle$ y $k_i (i \neq 0)$ el límite de i-ésimo lugar implícito eliminado en todo el proceso de reducción. El límite de $\langle \boldsymbol{R}, M_0 \rangle$ es $k = \max\{k_0, k_1, \ldots, k_q\}$.

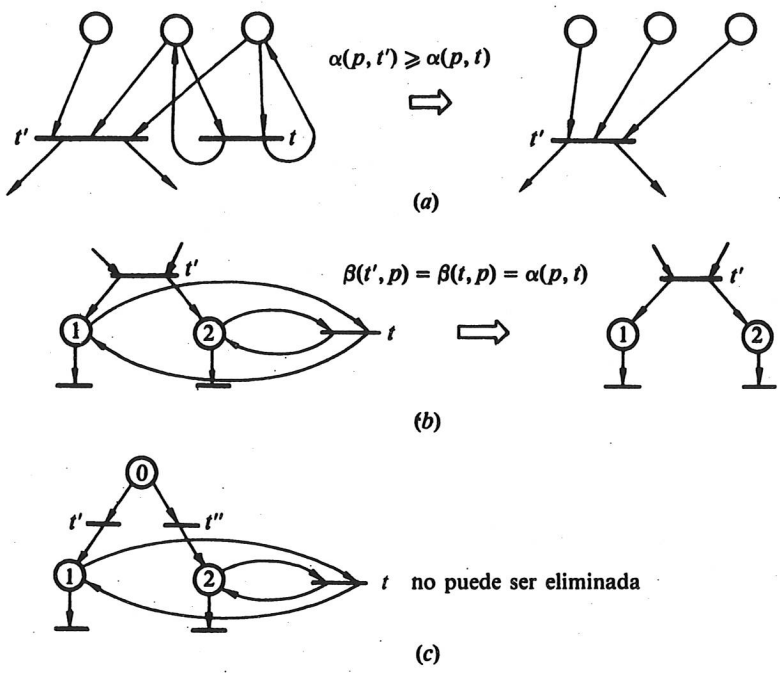
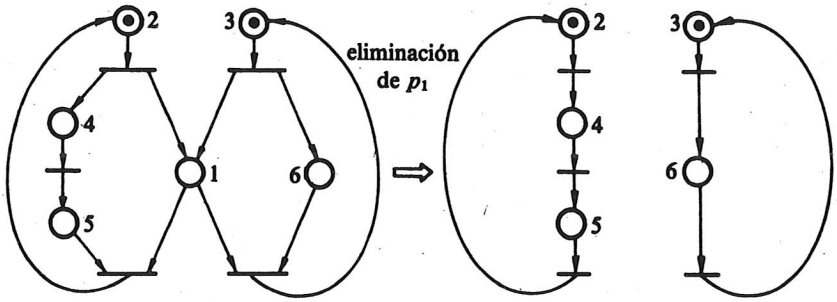


Figura 4.24. La transición identidad t sólo puede ser eliminada en los casos *a* y *b*.



(*a*) RdP no-binaria (2-limitada) (*b*) RdP binaria

Figura 4.25. La eliminación de un lugar implícito no preserva la k-limitación de la red marcada.

4.5.4.2 *Estrategia para la utilización de las reglas de reducción*

A lo largo del apartado 4.5 hemos estudiado tres reglas de reducción ($\mathfrak{R}_1$, $\mathfrak{R}_2$ y $\mathfrak{R}_3$). En el capítulo 3 (§3.3) estudiamos la eliminación de un *lugar implícito* ($\mathfrak{R}_0$). Estas cuatro reglas de reducción permiten conservar *la vivacidad* y *la limitación*. También hemos anunciado que si se desea estudiar el límite de un lugar, las exclusiones mutuas o la ciclicidad, conviene no hacer desaparecer los lugares considerados. A continuación presentamos dos tipos de estrategias de utilización de las cuatro reglas de reducción, así como casos particulares de las reglas citadas.

a) Estrategias básicas para la reducción: predefinidas o específicas

La primera de ellas, predefinida, consiste en definir «a priori» secuencias de aplicación de las reglas para reducir cualquier RdP. Desde un punto de vista práctico, hemos comprobado que normalmente conviene aplicar primero la regla $\mathcal{R}_1$ (la cual, al ser una regla no estrictamente local, suele producir fuertes reducciones en una sola aplicación) e iterar posteriormente con las otras reglas.

Si representamos mediante $\mathcal{R}_i^*$ la aplicación reiterada de la regla $\mathcal{R}_i$ hasta que no se pueda conseguir una nueva reducción, el siguiente esquema propone una estrategia (*predefinida*):

$\mathcal{R}_1$;
repetir
$\mathcal{R}_2^*$; $\mathcal{R}_3^*$; $\mathcal{R}_6^*$;
hasta que en una iteración no haya reducción alguna.

Como alternativa a las estrategias de reducción predefinidas, hay que señalar el interés de las estrategias específicas, generadas por el diseñador durante el proceso de reducción de la red analizada. Si el conjunto de reglas de reducción está implementado en un paquete de programas, el diálogo diseñador-computador será *interactivo*.

b) Casos particulares de las reglas de reducción

En este punto cabe señalar que a veces sólo se utiliza un conjunto de *casos particulares* de las reglas antes presentadas, $\mathcal{R}_0$ a $\mathcal{R}_3$. Este conjunto puede ser elegido en función de criterios tales como:

1) utilidad de cara a reducciones eficaces.
2) fácil implementación en computador.

En la figura 4.26 se presenta el conjunto de reglas elegido en el sistema de *ayuda a la concepción con computador* (CAD) denominado CAVIAR (realizado por Electricité de France) [BOUS 78]. (*Nota*. Obsérvese la analogía entre las reglas que se encuentran al mismo nivel; posteriormente introduciremos el concepto de dualidad en redes.) Este sistema fue diseñado para la concepción de automatismos lógicos destinados a las estaciones de distribución de energía eléctrica de alta tensión, etc.

EJEMPLO. Sea la RdP de la figura 4.27. Una aplicación de las reglas básicas de reducción conduce a la figura 4.28*a*. Los pasos seguidos son:

Paso 1. Aplicación de RA1 a t_3-t_5 y t_4-t_6.
Paso 2. Aplicación de RC1 a A_R con t_{35} y t_{46}.
Paso 3. Aplicación de RA1 a t_1-t_{35} y t_2-t_{46}.

Aplicando de nuevo las reglas de reducción se llega a la figura 4.29. Los pasos seguidos son:

Paso 4. Aplicación de RA1 a t_{246}-t_8 y t_{135}-t_7.
Paso 5. Aplicación de RC1 a p_1 y p_4.

EJERCICIO. Determínese, aplicando los métodos de reducción, cuales de las RdP del §2.4 son totalmente reducibles (es decir, tales que no hace falta aplicar posteriormente la enumeración). ¿Son vivas?, ¿son limitadas?

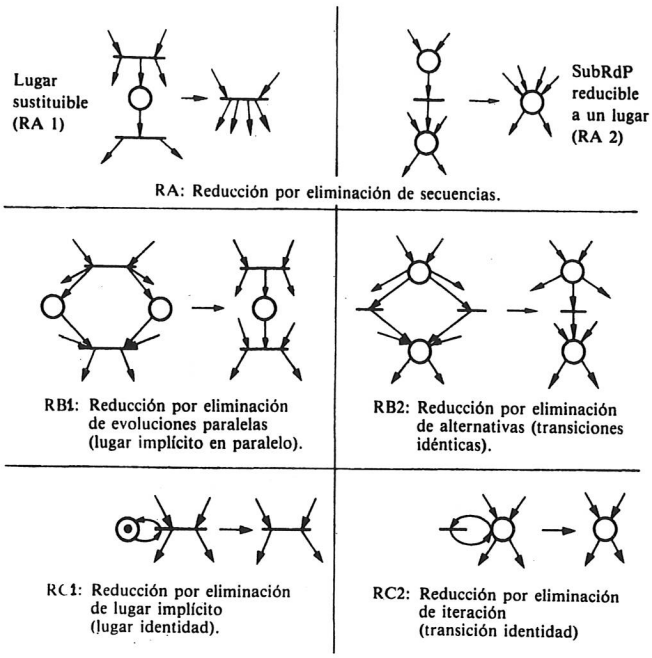


Figura 4.26. Reglas simples para la reducción de RdP ordinarias.

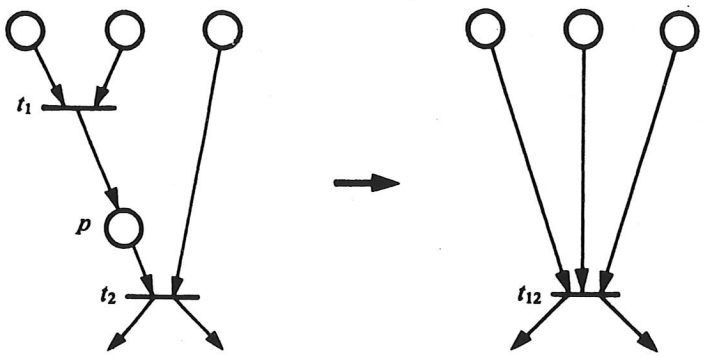


Figura 4.26 bis. Regla simple de reducción complementaria.

EJERCICIO. (1) Demuéstrese que la regla de reducción ilustrada por la figura 4.26*bis* permite preservar la k-limitación y la vivacidad. (*Sugerencia*: tengáse en cuenta que el lugar p posee sólo una transición de salida; además, p es el único lugar de salida de su transición de entrada.) (2) ¿Se puede generalizar esta regla de reducción?

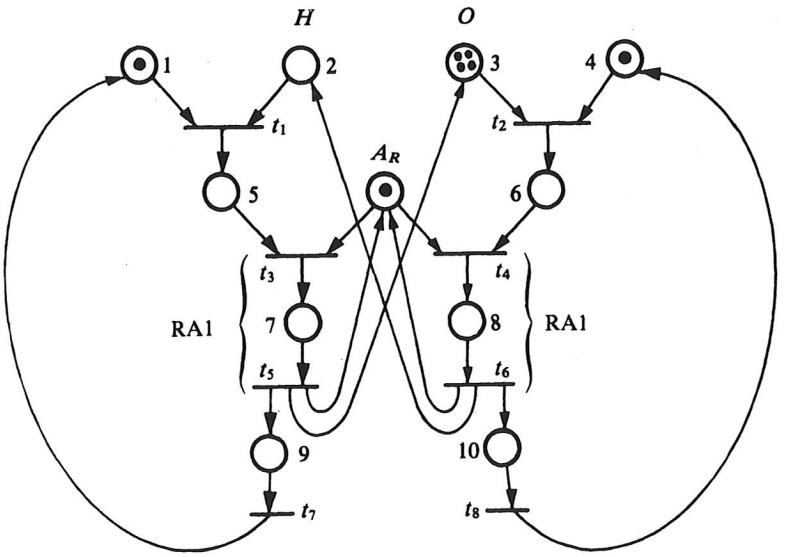


Figura 4.27. RdP a reducir (relación productor-consumidor). (*Nota:* los lugares $\{p_1, p_5\}$ y $\{p_4, p_6\}$ representan esperas del productor y del consumidor, respectivamente).

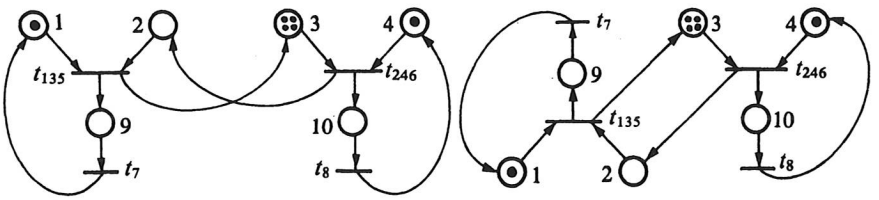


Figura 4.28. RdP que se obtiene después de los tres primeros pasos de reducción (dibujada en dos posiciones).

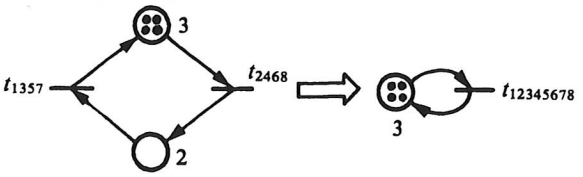


Figura 4.29. La RdP es viva y 4-limitada.

En los apartados que siguen abordaremos el estudio de los métodos que se conocen en la literatura como «estructurales». Antes resaltaremos que, en un sentido am-

plio, los métodos de reducción que hemos presentado son también estructurales, ya que proceden transformando la estructura de la RdP original.

4.6 CONSERVATIVIDAD Y REPETITIVIDAD: APLICACIÓN AL ANÁLISIS GLOBAL DE REDES DE PETRI

La conservatividad y la repetitividad son dos propiedades estructurales fáciles de estudiar, que están relacionadas con ciertas propiedades dinámico-estructurales como son la vivacidad estructural y la limitación estructural.

Los resultados que exponemos en este apartado, así como en el siguiente (§4.7), se obtienen al aplicar técnicas derivadas del álgebra lineal a la *ecuación de estado* de una RdP (§2.2.2):

$$M_k = M_{k-1} + \boldsymbol{C} \cdot U_k \quad \Rightarrow \quad M_k = M_0 + \boldsymbol{C} \cdot \bar{\sigma},$$

donde $\boldsymbol{C}$ representa la matriz de incidencia de una RdP pura. Si la RdP no es pura, es decir, no posee transiciones para las que ${}^{\cdot}t \cap t^{\cdot} \neq \emptyset$, entonces no se puede representar su estructura con una matriz. No obstante, la ecuación de estado es correcta si se sustituye la matriz de incidencia, $\boldsymbol{C}$, por la matriz de flujos de marcas, $\tilde{\boldsymbol{C}}$†. Siempre que exista una secuencia σ aplicable a partir de M_0: $M_k = M_0 + \tilde{\boldsymbol{C}} \cdot \bar{\sigma}$.

Antes de comenzar el estudio de definiciones y propiedades, conviene resaltar que las conclusiones a las que podamos llegar tienen «a priori» una serie de limitaciones:

1) La matriz $\boldsymbol{C}$ no representa la estructura de la RdP más que para las redes puras. De este modo, a toda red pura a la que se añadan lecturas de lugares le corresponde la misma ecuación de estado. Evidentemente, esto lleva consigo el que el estudio de la actividad (vivacidad, ciclicidad,...) de las redes no puras esté limitado de partida.

2) El vector característico $\bar{\sigma}$ condensa parte de la información sobre la secuencia de disparos σ, pero no la representa unívocamente, puesto que no informa sobre el orden en que se realizan los disparos de las transiciones. Esto también limita el estudio de la actividad para toda red de Petri.

3) El que $\bar{\sigma}$ y M sean vectores no negativos no permite garantizar la existencia de una secuencia σ aplicable a partir de M_0. Así, por ejemplo, a la RdP pura de la figura 4.30 no se le puede aplicar ninguna secuencia de disparos si inicialmente está marcado sólo p_3; sin embargo, existen soluciones de la ecuación de estado con $\bar{\sigma} \gneqq 0$ y $M \gneqq 0$:

$$\begin{pmatrix}1\\0\\0\\0\\0\\0\end{pmatrix} = \begin{pmatrix}0\\0\\1\\0\\0\\0\end{pmatrix} + \boldsymbol{C} \cdot \begin{pmatrix}0\\0\\1\\0\\1\\1\end{pmatrix} \qquad [M = M_0 + \boldsymbol{C} \cdot \bar{\sigma}]$$

† Ésta se introdujo en §3.3.2: $\tilde{\boldsymbol{C}} = [\tilde{c}_{ij}]_{n \times m}$ y $\tilde{c}_{ij} = \beta(t_j, p_i) - \alpha(p_i, t_j)$

A pesar de estas tres restricciones, el interés del estudio de las RdP mediante la ecuación de estado es enorme, puesto que permite:

1) Extraer importantes conclusiones sobre el comportamiento de la red mediante la aplicación de algoritmos de ejecución muy rápida.
2) Analizar fácilmente el comportamiento de la red para diversos marcados iniciales.

En este apartado presentamos definiciones básicas y resultados *globales* sobre el análisis estructural de una RdP. En el siguiente abordaremos fundamentalmente el estudio de propiedades *locales* del comportamiento (exclusiones mutuas entre marcados, límite de un lugar, avances sincrónicos, etc.).

4.6.1 Definiciones y propiedades básicas

Sea la RdP $\boldsymbol{R} = \langle P, T, \alpha, \beta \rangle$ y su matriz de incidencia $\boldsymbol{C}$. (*Nota*: si la red no es pura se utilizará $\tilde{\boldsymbol{C}}$.)

Definición 4.21. $\boldsymbol{R}$ es *repetitiva* sii existe un vector $X \in (\mathbb{N}^+)^m$ tal que $\boldsymbol{C} \cdot X = 0$. □

Es decir, una RdP es repetitiva sii su matriz de incidencia admite un vector anulador derecho tal que todos sus elementos sean positivos. Como $\boldsymbol{C} \cdot X = 0$ es un sistema de ecuaciones lineales con coeficientes enteros, éste admite soluciones racionales. Si una solución X_0, $X_0(i) = n(i)/d(i)$, se multiplica por el mínimo común múltiplo (mcm) de los denominadores, $d = \text{mcm}(d(i))$, se obtiene una solución entera, $\bar{X}_0$: $\bar{X}_0(i) = X_0(i)d$. Si todas las componentes son positivas, la RdP es repetitiva.

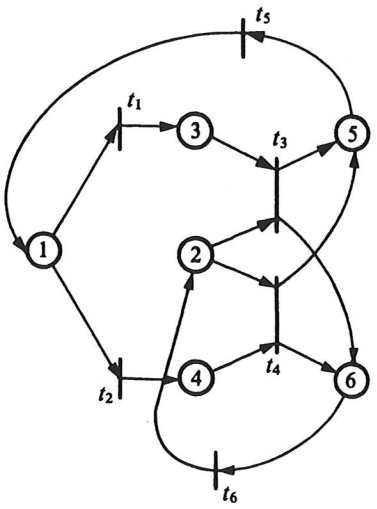
Para escribir directamente las n ecuaciones $\boldsymbol{C} \cdot X = 0$, basta con establecer el balance de marcas en cada lugar. En efecto, el disparo de una transición, t, implica la eliminación de $\alpha(p, t)$—peso del arco que une p a t— marcas de p y la adición de $\beta(t, p')$ —peso del arco que une t a p'— marcas a p'. La figura 4.30 muestra un ejemplo para el que $X^T = (\lambda\ \mu\ \lambda\ \mu\ \lambda+\mu\ \lambda+\mu)$ es solución. Tomando $\lambda = \mu = 1$, $X^T = (1\,1\,1\,1\,2\,2)$ y la RdP es, por tanto, repetitiva. La independencia entre λ y μ se puede explicar fácilmente, dado que a partir de p_1 es posible elegir libremente entre disparar t_1 o t_2. Más adelante abordaremos el cálculo sistemático de una solución positiva de la ecuación $\boldsymbol{C} \cdot X = 0$.

Observación. Establecer el balance de marcas en cada lugar de una RdP ordinaria es semejante a aplicar la primera ley de KIRCHOFF (la suma de intensidades en un nudo es nula) a un circuito eléctrico en el que:

- los nudos son los lugares,
- las ramas son las transiciones.

Si se trata de una RdP generalizada, se debería realizar la aplicación de la primera ley de KIRCHOFF, considerando que a cada nudo (lugar) llega (o sale) una intensidad, la asociada a la rama (transición), multiplicada por el peso del arco correspondiente (el arco que une la transición al lugar o el lugar a la transición).

Una RdP es repetitiva si admite una «asignación de intensidades orientada de acuerdo con sus arcos» en la que todas las intensidades son positivas.



p_1: $x_5 - x_1 - x_2 = 0$
p_2: $x_6 - x_3 - x_4 = 0$
p_3: $x_1 - x_3 = 0$
p_4: $x_2 - x_4 = 0$
p_5: $x_3 + x_4 - x_5 = 0$
p_6: $x_3 + x_4 - x_6 = 0$

Tomando:
$x_1 = x_3 = \lambda$
$x_2 = x_4 = \mu$
se obtiene
$x_5 = x_6 = \lambda + \mu$

Figura 4.30. RdP repetitiva (y conservativa).

Significación de la repetitividad de una RdP (proposición 4.6). La RdP $\boldsymbol{R}$ es repetitiva sii existe una secuencia de disparos, σ, que contiene todas las transiciones de la RdP y un marcado M, suficientemente grande, tal que $M \xrightarrow{\sigma} M$. □

DEMOSTRACIÓN

1) $\exists\sigma$ y $\exists M \Rightarrow \boldsymbol{R}$ repetitiva.
   Si σ contiene todas las transiciones de $\boldsymbol{R}$, entonces $X = \bar{\sigma} \geqslant {}' \mathbb{I}^m$. Como $M = M_0 + \boldsymbol{C} \cdot \bar{\sigma}$ y $M_0 = M$, tendremos $\boldsymbol{C} \cdot \bar{\sigma} = 0$, de donde $\boldsymbol{C} \cdot X = 0$ y, por tanto, $\boldsymbol{R}$ es repetitiva.
2) $\boldsymbol{R}$ repetitiva $\Rightarrow \exists\bar{\sigma} \geqslant {}' \mathbb{I}^m$ y $\exists M$ tal que $M \xrightarrow{\sigma} M$.
   $\boldsymbol{R}$ repetitiva $\Rightarrow \exists X \in (\mathbb{N}^+)^m$ tal que $\boldsymbol{C} \cdot X = 0$. Construyamos un marcado M tal que $\forall p \in P$, $M(p) \geqslant \sum_{t_i \in T} X(t_i)\alpha(p, t_i)$. A partir de M puede aplicarse tantas veces como se desee la secuencia siguiente para la que $\bar{\sigma} = X$:

$$\sigma = \underbrace{t_1 \ldots t_1}_{X(t_1) \text{ veces}} \; \underbrace{t_2 \ldots t_2}_{X(t_2) \text{ veces}} \; \ldots \; \underbrace{t_m \ldots t_m}_{X(t_m) \text{ veces}}$$

   por consiguiente, existe $\bar{\sigma} \geqslant {}' \mathbb{I}^m$ y $M \xrightarrow{\sigma} M$. □

Definición 4.22. Una RdP es *conservativa* sii existe un vector $Y \in (\mathbb{N}^+)^n$ tal que $Y^T \cdot \boldsymbol{C} = 0$. □

Es decir, una RdP es conservativa sii su matriz de incidencia admite un vector anulador izquierdo tal que todos sus elementos sean positivos. Razonando de forma similar a como hicimos anteriormente, llegamos a la conclusión de que a partir de cualquier solución, Y_0, de la ecuación $Y^T \cdot \boldsymbol{C} = 0$ se puede obtener una solución canónica en la que todos sus componentes sean enteros positivos.

El cálculo de un vector Y, solución de la ecuación $Y^T \cdot C = 0$, puede realizarse sin tener que escribir la matriz C. En efecto, basta recordar las reglas de evolución del marcado. Para escribir directamente las m ecuaciones $Y^T \cdot C = 0$, estableceremos el balance de marcas en cada transición. Reconsiderando la RdP de la figura 4.30 se puede escribir lo siguiente:

$$\left.\begin{array}{l} t_1: -Y_1 + Y_3 = 0 \\ t_2: -Y_1 + Y_4 = 0 \\ t_3: -Y_2 - Y_3 + Y_5 + Y_6 = 0 \\ t_4: -Y_2 - Y_4 + Y_5 + Y_6 = 0 \\ t_5: -Y_5 + Y_1 = 0 \\ t_6: -Y_6 + Y_2 = 0 \end{array}\right\} \Rightarrow \begin{array}{l} Y_1 = Y_3 = Y_4 = Y_5 = \gamma \text{ (relaciones 1, 2 y 5)} \\ Y_2 = Y_6 = \delta \text{ (relación 6). (}\textit{Nota.}\text{ las relaciones 3 y 4 son combinación lineal de las anteriores.)} \end{array}$$

En resumen, $Y^T = (\gamma\, \delta\, \gamma\, \gamma\, \gamma\, \delta)$. Si $\gamma = \delta = 1$, entonces $Y^T = (1\,1\,1\,1\,1\,1)$ y la RdP es conservativa.

Observación. La determinación de un vector Y solución de la ecuación $Y^T \cdot C = 0$ puede interpretarse también en términos electrotécnicos: el balance de marcas en cada transición equivale a la aplicación de la primera ley de KIRCHOFF a un circuito eléctrico en el que:

- los nudos son las transiciones,
- las ramas son los lugares.

Una RdP es conservativa si admite una «asignación de intensidades orientada de acuerdo con sus arcos» en la que todas las intensidades sean positivas.

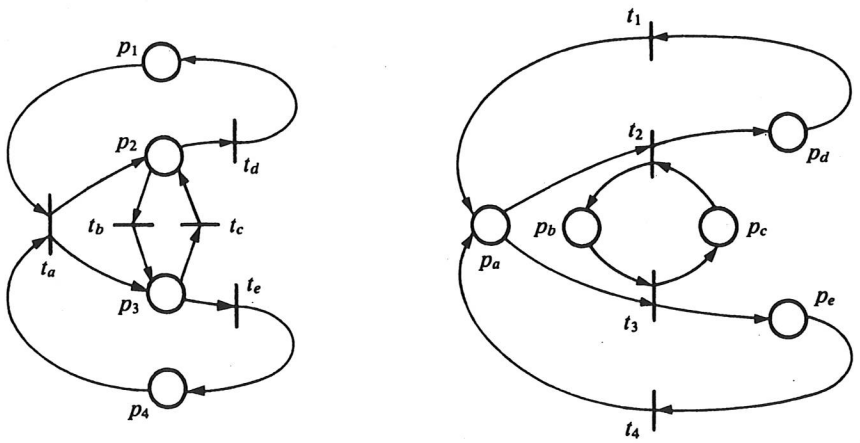
Volviendo al ejemplo anterior, figura 4.30, podemos explicar fácilmente la independencia entre γ y δ puesto que el disparo de t_3 o el de t_4 (únicas transiciones compartidas por los dos subconjuntos de lugares) elimina de $\{p_1, p_3, p_4, p_5\}$ un número de marcas igual al que le añade. Del mismo modo se puede razonar para el subconjunto de lugares $\{p_2, p_6\}$. Dicho de otra forma, el número de marcas en cada uno de los subconjuntos de lugares es *invariante*. Este razonamiento será formalizado y generalizado posteriormente (§4.7). En la proposición 4.7 se demostrará que, si la RdP es conservativa, el marcado se conservará acotado (de ahí que se llame *conservatividad* a la propiedad estructural).

Como se puede observar, existe gran similitud entre las definiciones de RdP repetitiva y RdP conservativa. Esta similitud puede ser formalizada introduciendo el concepto de RdP dual. Sea $R = \langle P, T, \alpha, \beta \rangle$.

Definición 4.23. Se dice que R_d es *la RdP dual* de R si $R_d = \langle T, P, \beta, \alpha \rangle$. Es decir, R_d es la dual de R si, preservando su estructura, se cambian los lugares por transiciones y viceversa (figuras 4.31 y 4.32). □

La matriz de incidencia de R_d, C_d, se calcula fácilmente a partir de C: $C_d = -C^T$ (¡atención al signo!). La inversión de signo es necesaria para preservar la orientación de los arcos, porque:

$$c_{ij} = (-\alpha(p_i, t_j) + \beta(t_j, p_i)) \text{ y } (c_d)_{ji} = ((-\beta(p_j, t_i) + \alpha(t_i, p_j)).$$



(*a*) Estructuralmente limitada. Estructuralmente no-viva.

(*b*) Estructuralmente limitada. Estructuralmente viva.

Figura 4.31. Ejemplo de redes de Petri duales (ambas son repetitivas y conservativas).

La consideración de la dualidad en RdP nos permitirá comprender mejor algunas propiedades estructurales y razonar de forma más económica. Por ejemplo, resulta evidente que toda *máquina de estados* (ME) es una RdP dual de un *grafo marcado* (GM), y viceversa. Es fácil demostrar, por otra parte, que toda ME es conservativa y, si es fuertemente conexa, también es repetitiva. Por consiguiente, se puede afirmar por dualidad que todo GM es repetitivo y, si es fuertemente conexo, también será conservativo.

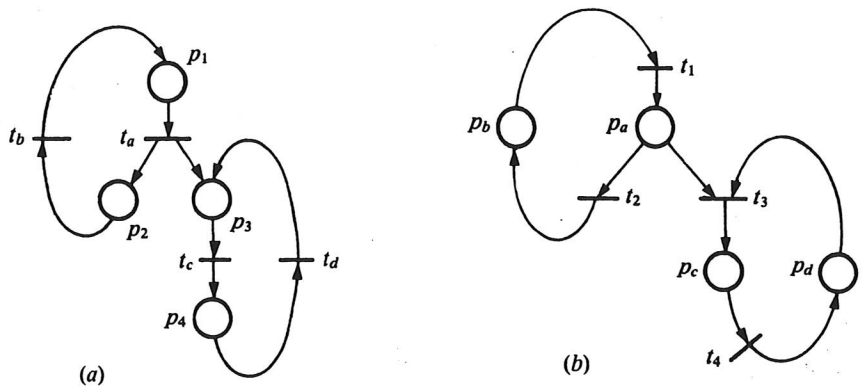


Figura 4.32. R_a es $\{\bar{R}, \bar{C}, \bar{L}, V\}$ y $R_b = (R_a)_d$ es $\{\bar{R}, \bar{C}, L, \bar{V}\}$.

4.6.2 Relación entre propiedades estructurales y propiedades dinámico-estructurales

En este apartado vamos a presentar los resultados básicos que permiten establecer un diagnóstico sobre la limitación, la vivacidad y la ciclicidad de una RdP. Veremos que la caracterización de la limitación es muy satisfactoria. Sin embargo, no ocurre lo mismo con la actividad de la RdP. En particular, no existe propiedad general que proporcione una condición suficiente para la vivacidad. Esta «laguna» puede comprenderse fácilmente, puesto que, como se ha anunciado, el vector característico de la secuencia de disparos, $\bar{\sigma}$, no dispone de una información que es esencial para caracterizar la vivacidad: el orden en que se disparan las transiciones en σ.

Los resultados básicos que exponemos se estructuran en cuatro proposiciones y un conjunto de corolarios de interés práctico. Todos los corolarios se deducen directamente de las proposiciones principales mediante simples transformaciones lógicas. En la tabla 4.2 se resume el conjunto de resultados. A ella remitimos al lector no interesado en los detalles de su demostración.

Conservatividad y limitación (proposición 4.7). Sea $\boldsymbol{R}$ una red, $\boldsymbol{C}$ su matriz de incidencia e $Y \in (\mathbb{N}^+)^n$. Si $Y^T \cdot \boldsymbol{C} = 0$ (RdP conservativa), $\boldsymbol{R}$ es estructuralmente limitada (el marcado se conserva acotado). □

DEMOSTRACIÓN. $Y^T \cdot \boldsymbol{C} = 0 \Rightarrow Y^T \cdot M = Y^T \cdot M_0$. Ahora bien, como $Y > 0$ y $M(p) \geqslant 0$, se puede escribir

$$Y^T \cdot M_0 = Y^T \cdot M = \sum_{p \in P} Y(p)M(p) \geqslant Y(p)M(p) \Rightarrow M(p) \leqslant \left\lfloor \frac{Y^T \cdot M_0}{Y(p)} \right\rfloor .$$

Por consiguiente, si M_0 es finito, el marcado de cualquier lugar está limitado por un valor finito, de donde se deduce que la red es estructuralmente limitada. □

La inversión de la proposición 4.7 no es posible, porque una RdP puede ser estructuralmente limitada y no ser conservativa (figura 4.32*b*).

EJERCICIO. Compruébese que las RdP de la figura 1.15 son estructuralmente limitadas. (*Sugerencia*: demuéstrese que son conservativas.)

Condición suficiente para que una RdP sea conservativa (proposición 4.8). Toda RdP estructuralmente limitada y repetitiva es conservativa. □

DEMOSTRACIÓN. Descompondremos ésta en las dos fases siguientes:

a) Si $\boldsymbol{R}$ es estructuralmente limitada y repetitiva, no existe $X \in \mathbb{Z}^m$ tal que $\boldsymbol{C} \cdot X \gneqq 0$.

b) Si no existe $X \in \mathbb{Z}^m$ tal que $\boldsymbol{C} \cdot X \gneqq 0$, existe $Y \in (\mathbb{N}^+)^n$ tal que $Y^T \cdot \boldsymbol{C} = 0$ y, por consiguiente, $\boldsymbol{R}$ es conservativa.

Fase a. Supóngase que existe $X \in \mathbb{Z}^m$ tal que $C \cdot X \gneqq 0$. Si R es repetitiva, existe $X_0 \in (\mathbb{N}^+)^m$ tal que $C \cdot X_0 = 0$. En estas condiciones podemos construir una infinidad de vectores $X_1 = X + \lambda X_0$ ($\lambda > 0$) tales que $X_1 > 0$. Considerando una secuencia σ tal que $\bar{\sigma} = X_1$, podremos construir un marcado inicial M_0 suficientemente grande como para que σ sea aplicable. Por ejemplo: $M_0(p) \geqslant \sum_{t_i \in T} X_1(t_i)\alpha(p, t_i)$. Por lo tanto, $M_0 \xrightarrow{\sigma} M_1$ y $M_1 = M_0 + C \cdot \bar{\sigma}$, luego, si repetimos la secuencia σ, tendremos $M_1 \xrightarrow{\sigma} M_2 \xrightarrow{\sigma} M_3 \xrightarrow{\sigma} \ldots$ con $M_{i+1} \gneqq M_i$, de donde $\langle R, M_0 \rangle$ no será limitada. Si $\langle R, M_0 \rangle$ es no limitada, R no será estructuralmente limitada y, por lo tanto, incurriremos en contradicción. En conclusión, no existe $X \in \mathbb{Z}^m$ tal que $C \cdot X \gneqq 0$.

Fase b. Esta fase resulta directamente de la aplicación del teorema de STIEMKE (véase, por ejemplo, [DANT 68]). En efecto, este teorema establece que el sistema de inecuaciones $Y^T \cdot C = 0$ e $Y > 0$ tiene solución sii el sistema $C \cdot X \gneqq 0$ no tiene. Por consiguiente, existe $Y > 0$ tal que $Y^T \cdot C = 0$, de donde se deduce que la RdP es conservativa. □

El interés de la detección de los lugares (respec. transiciones) que hacen que una RdP no sea conservativa (respec. no repetitiva) se considerará en este mismo apartado, cuando se establezcan los diferentes resultados básicos. Ahora presentaremos un resultado fundamental.

Condición necesaria para la vivacidad y la limitación estructural (proposición 4.9). Toda RdP estructuralmente limitada y viva es repetitiva y conservativa. □

Esta proposición es de capital importancia. Desafortunadamente, la proposición inversa no es cierta. Así, la RdP de la figura 4.31*a* es repetitiva, conservativa y limitada, pero no es estructuralmente viva. Por el contrario la red dual, figura 4.31*b*, sí es estructuralmente viva.

Por otro lado, tampoco se puede afirmar que una RdP no repetitiva y no conservativa sea estructuralmente no limitada y no viva (figura 4.32).

DEMOSTRACIÓN (de la proposición 4.9):

a) Si R es estructuralmente limitada y viva, es también repetitiva. Su justificación es evidente, incluso si se procede a un enunciado más restrictivo: toda RdP limitada y viva es repetitiva.

En efecto, para que R sea repetitiva, basta con demostrar la existencia de una secuencia de disparos, σ, tal que $M \xrightarrow{\sigma} M$ y que contenga todas las transiciones. Si $\langle R, M_0 \rangle$ es limitada y viva, el grafo de marcados es finito y toda componente fuertemente conexa, contiene todas las transiciones. Sea M' un marcado de una de esas componentes fuertemente conexas de $G(R, M_0)$; puesto que M' pertenece a una componente fuertemente conexa que contiene todas las transiciones de la red, se puede disparar una secuencia que contenga todas las transiciones y tal que su aplica-

ción conduzca de nuevo a M'. σ es una secuencia repetitiva y, por lo tanto, R será también repetitiva.

b) Si $\boldsymbol{R}$ es estructuralmente limitada y viva, también es conservativa. Esto se demuestra combinando la proposición 4.8 y la propia proposición 4.9*a*:

$$\left.\begin{array}{l} \text{4.9}a\text{: [Estructuralmente limitada]} \wedge \text{[Viva]} \Rightarrow \text{[Repetitiva]} \\ \text{4.8: [Estructuralmente limitada]} \wedge \text{[Repetitiva]} \Rightarrow \text{[Conservativa]} \end{array}\right\} \Rightarrow$$

$$\text{[Estructuralmente limitada]} \wedge \text{[Viva]} \Rightarrow \text{[Conservativa]}. \quad \square$$

Observación muy importante. Para la verificación de las propiedades dinámico-estructurales de vivacidad y de limitación, la proposición 4.9*a* presenta dos condiciones necesarias: la repetitividad y la conservatividad. Ahora bien, es importante recordar que existen RdP marcadas limitadas y vivas, pero no estructuralmente limitadas (recuérdese que la vivacidad implica la vivacidad estructural y que la limitación estructural implica la limitación; las implicaciones inversas no son ciertas) (véase la figura 4.3).

El último de los resultados básicos de este apartado establece una relación entre la vivacidad, la ciclicidad y la repetitividad.

Condición suficiente para la repetitividad (proposición 4.10). Toda RdP viva y cíclica es repetitiva. □

Como se puede observar, en esta última proposición se infiere una propiedad estructural (la repetitividad) a partir de dos propiedades puramente dinámicas.

DEMOSTRACIÓN (de la proposición 4.10). Es trivial, puesto que en cualquier RdP viva y cíclica se puede considerar una secuencia que dispare todas las transiciones volviendo al marcado inicial. Es decir, existe al menos una secuencia σ tal que $\bar{\sigma} > 0$ y $\boldsymbol{C} \cdot \bar{\sigma} = 0$; por lo tanto, la RdP será repetitiva. □

La inversión de la proposición 4.10 no es posible. En efecto, ni siquiera se puede garantizar que una RdP repetitiva y viva sea cíclica (véase en la figura 4.4. la red número 6) ya que el marcado inicial para el que se define la secuencia repetitiva no tiene por qué ser un marcado determinado.

En la tabla 4.2 resumimos, por un lado, las diferentes proposiciones y, por otro, presentamos sus corolarios, que son simples reformulaciones lógicas del resultado básico del que se deducen. Obsérvese que todos los resultados presentados en la tabla 4.2 son independientes del marcado inicial de la red y sólo dependen de su estructura.

Observación. Ninguno de los resultados del análisis estructural global (tabla 4.2) permite demostrar que una red es viva o cíclica. Ello se debe a la falta de información existente en

Denominación	Enunciado	Fuente para su deducción
(*) Proposición 4.7	$C \Rightarrow L_e$	------
Proposición 4.8	$L_e \wedge R \Rightarrow C$	------
(*) Corolario 4.3	$R \wedge \bar{C} \Rightarrow \bar{L}_e$	Proposición 4.8
Proposición 4.9	$L_e \wedge V_e \Rightarrow C \wedge R$	------
(*) Corolario 4.4	$\bar{C} \vee \bar{R} \Rightarrow \bar{L}_e \vee \bar{V}_e$	Proposición 4.9
Corolario 4.5	$\bar{R} \wedge L_e \Rightarrow \bar{V}_e$	Corolario 4.4
Corolario 4.6	$\bar{C} \wedge L_e \Rightarrow \bar{V}_e$	,,
Corolario 4.7	$\bar{R} \wedge V_e \Rightarrow \bar{L}_e$	,,
Corolario 4.8	$\bar{C} \wedge V_e \Rightarrow \bar{L}_e$	,,
(*) Corolario 4.9	$\bar{R} \wedge C \Rightarrow \bar{V}_e \wedge L_e$	Proposición 4.7 y Corolario 4.5
Proposición 4.10	$V_e \wedge C_i \Rightarrow R$	------
Corolario 4.10	$\bar{R} \wedge V_e \Rightarrow C_i$	Proposición 4.10
Corolario 4.11	$\bar{R} \wedge C_i \Rightarrow \bar{V}_e$	,,

Leyenda

C = Conservativa
R = Repetitiva
V_e = Estructuralmente viva
L_e = Estructuralmente limitada
C_i = Cíclica

Nota: Se han marcado con (*) aquellas implicaciones cuyo primer miembro contiene sólo propiedades estructurales.

Tabla 4.2. Resumen de la relación entre propiedades estructurales y propiedades dinámico-estructurales.

la ecuación del marcado (ya comentado) y a que los resultados anteriores son independientes del marcado inicial.

Observación. Los corolarios 4.3 y 4.9 exhiben cierta asimetría. En efecto, el corolario 4.3 no puede afirmar la vivacidad o no vivacidad de una RdP (figura 4.33). Sin embargo, se puede comprobar fácilmente que las RdP duales de las redes de la figura 4.33 son estructuralmente no vivas y limitadas.

Antes de concluir este apartado, interesa considerar la existencia de los lugares (respec. transiciones) que hacen que una RdP no sea conservativa (respec. repetitiva). Estos lugares (respec. transiciones) están representados por variables $Y(i)$ (respec. $X(j)$) de valor *nulo* para cualquiera de las soluciones de $Y^T \cdot C = 0$ e $Y \gneqq 0$ (respec. $C \cdot X = 0$, $X \gneqq 0$).

Si la red es viva, el que $Y(i)$ sea una *variable nula* significa que existirá al menos un marcado M_0 para el que p_i es no limitado. Así pues, la RdP de la figura 4.33*b* no es conservativa porque $Y(3) = 0$ para cualquier solución del problema $Y^T \cdot C = 0$ e $Y \gneqq 0$. La red es viva y p_3 es un lugar no limitado. Por dualidad, podemos argumentar que, si t_j hace que una RdP no sea repetitiva y la RdP es limitada, t_j será una transición no viva.

EJERCICIO. Compruébese que p_3 y p_4 (figura 4.32*a*) hacen que la RdP sea no conservativa (su supresión permite obtener una red conservativa y, por lo tanto, limitada). Compruébese que t_3 y t_4 (figura 4.32*b*) hacen que la RdP sea no repetitiva (su supresión permite obtener una red repetitiva y, en este caso, viva).

En conclusión, la verificación de propiedades estructurales, como son la repetitividad y la conservatividad, permite establecer un diagnóstico previo sobre las propiedades dinámicas (dinámico-estructurales) de una RdP. El interés de estos resultados reside fundamentalmente en que la determinación de la repetitividad y de la conservatividad no exige la enumeración de los marcados de la RdP. En §4.7.2 se presenta un algoritmo que permite calcular unos objetos (las componentes conservativas elementales) a partir de los cuales se puede determinar, mediante sumas, si la RdP es conservativa. La determinación de la repetitividad es similar.

EJERCICIO. Determínese si las RdP de la figura 4.4 son conservativas o/y repetitivas. ¿Qué propiedades dinámicas se pueden deducir sin enumerar los marcados alcanzables?

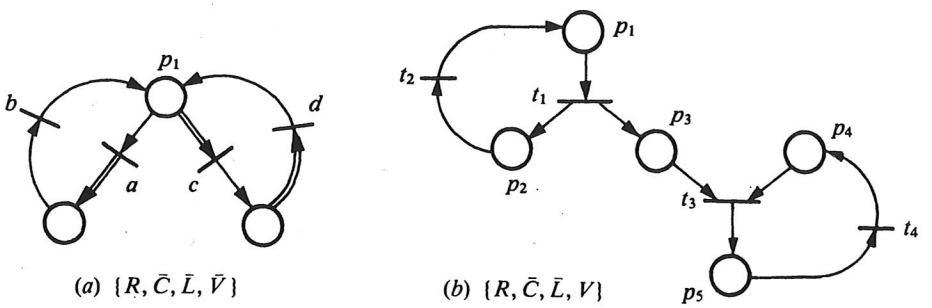


Figura 4.33. El corolario 4.3 no permite afirmar nada sobre la vivacidad de una RdP (proposición «simétrica» a la del corolario 4.9).

4.7 INVARIANTES DE MARCADO Y DE DISPARO: APLICACIÓN AL ANÁLISIS LOCAL DE REDES DE PETRI

En el apartado anterior hemos estudiado la conservatividad y la repetitividad, así como su aplicación al *análisis global* de redes; es decir, a un tipo de análisis que considera la red como un todo y permite estudiar propiedades generales de la misma, tales como la limitación, la vivacidad y la ciclicidad. En este apartado presentaremos dos grupos de técnicas de análisis estructural basadas en la determinación de relaciones válidas independientemente de la evolución de la RdP; es decir, de relaciones invariantes. Estas últimas permitirán un análisis detallado de algunas propiedades locales o/y específicas (exclusiones mutuas, límite de un lugar, etc.)

Un *invariante de marcado* es cualquier relación satisfecha por todos los marcados alcanzables a partir del marcado inicial. Análogamente, un *invariante de disparo* es cualquier relación satisfecha por todas las secuencias de disparos aplicables a partir del marcado inicial, o bien por sus vectores característicos asociados.

Los vectores anuladores izquierdos de la matriz de incidencia (o de flujo de marcas) permiten obtener invariantes de marcado. El apartado 4.7.1 se dedica a estable-

cer y aplicar una relación lineal básica que determina una condición necesaria para que un marcado M sea alcanzable a partir de un marcado inicial M_0. Los vectores no negativos anuladores izquierdos de la matriz de incidencia son especialmente interesantes desde un punto de vista práctico. A la definición, obtención y utilización de estos anuladores particulares se dedicará el apartado 4.7.2. Por último, el §4.7.3 se dedica al estudio de invariantes de disparo deducibles directamente de la estructura de la red, mediante una simple aplicación de la regla de evolución de una RdP a lugares de la misma.

4.7.1 Alcanzabilidad y base de invariantes lineales de marcado

4.7.1.1 *Fundamentos y aplicaciones*

La ecuación de estado de una RdP establece que, si M es alcanzable a partir de M_0 mediante la aplicación de la secuencia de disparos σ, entonces $M = M_0 + C \cdot \bar{\sigma}$. El problema fundamental que se plantea a continuación, *problema de alcanzabilidad*, consiste en determinar si un marcado, M, o conjunto de marcados, $\{M_i\}$, es alcanzable en la RdP definida mediante C y M_0.

Si la RdP posee n lugares y m transiciones, C será de dimensión $n \times m$. Sea rango(C) = r y sea B_J una base de anuladores izquierdos de C: la matriz B_J será de dimensión $n \times (n - r)$†. Dado que B_J es una base, cualquier vector J anulador izquierdo de C [$J^T \cdot C = 0$ y $J \in \mathbb{Z}^n$] será combinación lineal de sus $n - r$ columnas. Por último, sea $B_J^T \cdot M_0 = b$.

Condición necesaria para la alcanzabilidad (proposición 4.11). Para que M sea alcanzable a partir de M_0 es necesario que $B_J^T \cdot M = B_J^T \cdot M_0 = b$ y $M \in \mathbb{N}^n$. □

DEMOSTRACIÓN. Es trivial. Premultiplicando la ecuación de estado por B_J^T: $B_J^T \cdot M = B_J^T \cdot M_0 + B_J^T \cdot C \cdot \bar{\sigma} = B_J^T \cdot M_0 = b$. Luego, para que sea alcanzable M, es necesario que $B_J^T \cdot M = b$. Por definición: $M \in \mathbb{N}^n$. □

Lamentablemente no se puede afirmar que todo vector M que cumpla $B_J^T \cdot M = b$ será un marcado alcanzable a partir de M_0. La figura 4.34 muestra la relación de inclusión existente entre los vectores M solución del sistema de ecuaciones $B_J^T \cdot M = b$, los vectores solución del anterior sistema y de las restricciones $M \in \mathbb{N}^n$, y los vectores solución del problema de alcanzabilidad (existencia de una secuencia de disparos σ aplicable a partir de M_0 que lleve a M). El siguiente ejercicio es suficientemente ilustrativo.

EJERCICIO. Calcúlese una base de anuladores izquierdos, B_J, para la matriz C de la RdP de la figura 4.30 (*Sugerencia*: utilícese, por ejemplo, el algoritmo del Anexo 3.) Compruébese que para $M_0^T = (0\,1\,0\,0\,0)$ se tiene:

1) $M^T = (0\,0\,-1\,0\,1\,1)$ como solución de $B_J^T \cdot M = B_J^T \cdot M_0 = b$ y $M \in \mathbb{Z}^n$;
2) $M^T = (0\,0\,0\,0\,0\,1)$ como solución de $B_J^T \cdot M = b$ y $M \in \mathbb{N}^n$;
3) $M(R, M_0) = M_0$; es decir, no existe ningún otro marcado alcanzable a partir de M_0.

---

† Los razonamientos que siguen utilizan el álgebra lineal, por lo que, si fuese necesario, aconsejamos la lectura del anexo 3.

Como hemos visto, no se puede afirmar que todo vector no negativo M que cumple $B_J^T \cdot M = b$ será alcanzable a partir de M_0. No obstante, es evidente que todo marcado alcanzable es un vector no negativo que cumple el sistema de ecuaciones (restricciones) $B_J^T \cdot M = b$. Cada una de las $n - r$ ecuaciones del sistema anterior expresa un *invariante de marcado*. Cualquier combinación lineal de los $n - r$ invariantes es a su vez un nuevo invariante.

Desde un punto de vista práctico, más que determinar si un determinado marcado es alcanzable, interesa saber si el conjunto de marcados alcanzables verifica una determinada propiedad. Así, por ejemplo, si se desea demostrar la exclusión mutua entre los marcados de dos lugares p_4 y p_5, lo que interesa demostrar es que, para el conjunto de marcados alcanzables, se tiene $M(p_4)M(p_5) = 0$.

De acuerdo con lo anterior, el planteamiento básico del problema de la alcanzabilidad se generalizará hasta la formulación de *aserciones invariantes sobre los marcados alcanzables.* Sea $A(M)$ el conjunto de aserciones invariantes sobre los marcados alcanzables que se desea comprobar para la red $\langle R, M_0 \rangle$.

Condición suficiente para la verificación de aserciones (proposición 4.12). Si no existe $M \in \mathbb{N}^n$ que satisfaga el sistema de ecuaciones $B_J^T \cdot M = B_J^T \cdot M_0 = b$ y la negación de las aserciones $A(M)$, $\neg A(M)$, entonces todos los marcados alcanzables en $\langle R, M_0 \rangle$ verifican las aserciones $A(M)$. □

DEMOSTRACIÓN. Como se ha dicho anteriormente (figura 4.34), el conjunto de soluciones del sistema de ecuaciones $B_J^T \cdot M = b$ y $M \in \mathbb{N}^n$ contiene al conjunto de marcados alcanzables. Por lo tanto no existirá marcado alcanzable que verifique $\neg A(M)$, luego todo marcado alcanzable verificará $A(M)$. □

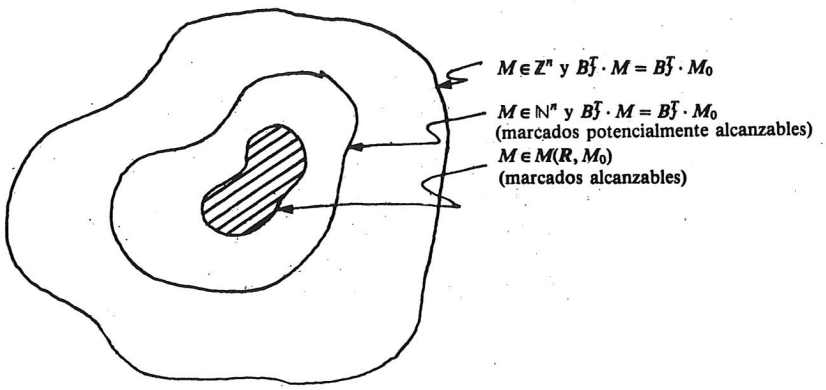


Figura 4.34. Los marcados M solución de $B_J^T \cdot M = B_J^T \cdot M_0$ y $M \in \mathbb{N}^n$ se denominarán marcados *potencialmente alcanzables.*

Si la violación de aserciones se expresa linealmente, el estudio de una condición suficiente para la validación se reduce a demostrar que el siguiente *sistema lineal* no posee solución:

$$\begin{cases} B_J^T \cdot M = b & \{n - r \text{ ecuaciones: sistema generador de invariantes de marcado}\} \\ \neg A(M) & \{q \text{ inecuaciones}\} \\ M \in \mathbb{N}^n & \{n \text{ restricciones}\} \end{cases}$$

EJEMPLO. Compruébese si el marcado de los lugares p_4 y p_5 está en exclusión mutua en la red de la figura 4.35.

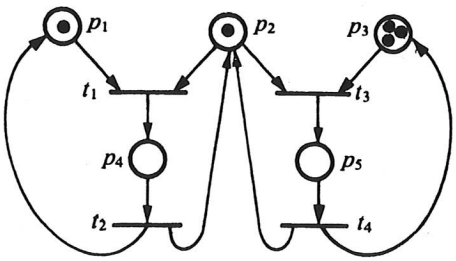


Figura 4.35. Salvo p_3 todos los lugares están 1-limitados. Además p_4 y p_5 están en exclusión mutua.

La aserción invariante que se desea comprobar es

$$A(M) \equiv [M(p_4) = 0] \vee [M(p_5) = 0] \Rightarrow \neg A(M) = [M(p_4) \geqslant 1] \wedge [M(p_5) \geqslant 1].$$

Escribiendo la matriz de incidencia de la RdP y aplicando las directrices del anexo 3, se puede calcular una base de anuladores izquierdos de aquélla. Por ejemplo:

$$B_J^T = \begin{pmatrix} -1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 2 \\ 0 & 1 & -1 & 1 & 0 \end{pmatrix}$$

Por consiguiente, el sistema que se ha de considerar es (sea $M(p_i) = m_i$):

$$\begin{cases} B_J^T \cdot M = B_J^T \cdot M_0 = b \Rightarrow \begin{cases} -m_1 + m_2 + m_5 = 0 & (1) \\ m_2 + m_3 + m_4 + 2m_5 = 4 & (2) \\ m_2 - m_3 + m_4 = -2 & (3) \end{cases} \\ \neg A(M) \Rightarrow \begin{cases} m_4 \geqslant 1 & (4) \\ m_5 \geqslant 1 & (5) \end{cases} \\ M \in \mathbb{N}^n \Rightarrow \{m_i \geqslant 0 \quad \forall i \in \{1,2,3,4,5\} & (6\text{-}10) \end{cases}$$

Este sistema no tiene solución puesto que sumando (2) y (3) se obtiene: $2m_2 + 2m_4 + 2m_5 = 2$, ecuación que no se puede verificar si $m_2 \geqslant 0$ (7), $m_4 \geqslant 1$ (4) y $m_5 \geqslant 1$ (5). Luego los marcados de p_4 y de p_5 están en exclusión mutua.

EJEMPLO. Comprobemos si p_4 es 1-limitado en la red de la figura 4.35.

La aserción invariante que se desea comprobar es:

$$A(M) \equiv [M(p_4) \leqslant 1] \Rightarrow \neg A(M) = [M(p_4) \geqslant 2].$$

Utilizando los cálculos del ejemplo anterior podemos reescribir $2m_2 + 2m_4 + 2m_5 = 2$. Si se hace $m_4 \geqslant 2$, se tendrá que $m_2 + m_5 \leqslant -1$, lo que es imposible puesto que el marcado de todo lugar tiene que ser positivo, $m_i \geqslant 0$. En conclusión, p_4 es 1-limitado (binario).

Los ejemplos anteriores muestran el interés de disponer de algún método sistemático para la obtención de las soluciones (o subconjunto de éstas que permita generar todas las demás) no negativas de un sistema de ecuaciones e inecuaciones lineales. En el anexo 5 se presenta un método basado en la transformación en un sistema de ecuaciones mediante la introducción de *variables de holgura*. La resolución del sistema de ecuaciones se basa en resultados que presentaremos en §4.7.2.

4.7.1.2 *Limitación del método de análisis e interés de los invariantes no negativos*

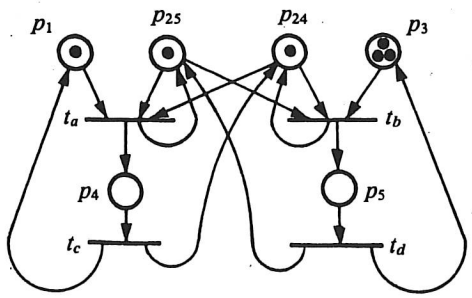
A lo largo del apartado 4.7.1.1, hemos introducido los fundamentos de un método de análisis basado en una condición necesaria para la alcanzabilidad de un marcado (proposición 4.11). En éste presentaremos, en primer lugar, un ejemplo que muestra la limitación del método de análisis considerado. Posteriormente, centraremos la atención en un subconjunto de invariantes de marcado que tienen un interés especial.

a) Limitación del método de análisis

Al construir el grafo de marcados asociado a la red de la figura 4.36, se puede concluir inmediatamente acerca de la exclusión mutua entre los marcados de p_4 y de p_5. Sin embargo, la aplicación del método de análisis esbozado por la proposición 4.12 conduce a la escritura de sistemas (equivalentes) de ecuaciones e inecuaciones lineales que admiten como solución:

$$\begin{cases} m_4 = 1 \Rightarrow m_1 = m_{24} = 0 \\ m_5 = 1 \Rightarrow m_{25} = 0 \text{ y } m_3 = 2. \end{cases}$$

De lo anterior se desprende que el análisis de alcanzabilidad derivado de la ecuación de estado de una RdP no permite obtener conclusiones en determinados casos para los que efectivamente se verifica la propiedad buscada.



$$\tilde{\mathbf{C}} = \begin{array}{c} \\ p_1 \\ p_{24} \\ p_{25} \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{cccc} t_a & t_b & t_c & t_d \end{array} \\ \begin{bmatrix} -1 & 0 & +1 & 0 \\ -1 & 0 & +1 & 0 \\ 0 & -1 & 0 & +1 \\ 0 & -1 & 0 & +1 \\ +1 & 0 & -1 & 0 \\ 0 & +1 & 0 & -1 \end{bmatrix} \end{array}$$

Figura 4.36. p_4 y p_5 están en exclusión mutua.

EJERCICIO. Compruébense las afirmaciones anteriores sobre la RdP de la figura 4.36. Estúdiese, apoyándose en la red considerada, el origen de la limitación encontrada en el método de análisis basado en la ecuación de estado.

b) Interés de los anuladores no negativos

Cuando la propiedad que se desea verificar es del tipo límite de un lugar o límite simultáneo de un subconjunto de lugares, se puede simplificar el tratamiento presentado en §4.7.1.1 y 2, reduciendo la atención a los anuladores izquierdos de C (o $\tilde{C}$) no negativos. En efecto, si reconsideramos la RdP de la figura 4.35, se puede observar que el anulador izquierdo no negativo de C, $J_1^T = (0\,1\,0\,1\,1)$, conduce al invariante de marcado $m_2 + m_4 + m_5 = 1$.

Ahora bien, dado que todo m_i es no negativo, $m_i \geqslant 0$, el anterior invariante permite concluir directamente que:

1) $m_2 \leqslant 1$, $m_4 \leqslant 1$ y $m_5 \leqslant 1$; es decir, el límite del marcado de p_2, p_4 y p_5 es 1 (son binarios)
2) $m_2 m_4 = 0$, el marcado de p_2 y p_4 está en exclusión mutua
   $m_2 m_5 = 0$, el marcado de p_2 y p_5 está en exclusión mutua
   $m_4 m_5 = 0$, el marcado de p_4 y p_5 está en exclusión mutua

Los anuladores $J_2^T = (1\,0\,0\,1\,0)$ y $J_3^T = (0\,0\,1\,0\,1)$ de la misma red permiten deducir lo siguiente:

$$m_1 + m_4 = 1 \Rightarrow \begin{cases} m_1 \leqslant 1, m_4 \leqslant 1 \\ m_1 m_4 = 0 \end{cases}$$

$$m_3 + m_5 = 3 \Rightarrow m_3 \leqslant 3, m_5 \leqslant 3.$$

Considerando simultáneamente todas las relaciones obtenidas se tiene:

- $m_1 \leqslant 1$, $m_2 \leqslant 1$, $2 \leqslant m_3 \leqslant 3$, $m_4 \leqslant 1$ y $m_5 \leqslant 1$
- exclusión mutua entre los marcados de los pares de lugares $\{p_2, p_4\}$, $\{p_2, p_5\}$, $\{p_4, p_5\}$, $\{p_1, p_4\}$. Como se puede comprobar, el análisis de una RdP basado en sus invariantes no negativos es bastante directo para determinadas propiedades. Dado su interés, el próximo apartado se dedica al estudio y obtención de los mencionados invariantes así como a la presentación de sus aplicaciones básicas en el análisis.

4.7.2 Invariantes no negativos: componentes conservativas

Los conceptos y técnicas relacionados con los invariantes de marcado no negativos los estructuraremos en tres puntos:

1) definiciones y propiedades básicas,
2) obtención del conjunto generador de todos ellos,
3) propiedades útiles para el análisis del comportamiento de sistemas descritos con RdP.

4.7.2.1 *Definiciones y propiedades básicas*

Definición 4.24. El vector $Y \in \mathbb{N}^n$ es una *componente conservativa* de una RdP cuya matriz de incidencia (o flujo) es C sii $Y^T \cdot C = 0$. □

Es decir, una componente conservativa es un vector no negativo anulador izquierdo de la matriz C.

Observación muy importante. Podemos definir de forma dual el concepto de *componente repetitiva*: vector $X \in \mathbb{N}^m$ tal que $C \cdot X = 0$.

Definición 4.25. Se denomina *soporte* de una componente Y al conjunto de lugares asociados a los elementos no nulos de Y. $||Y||$ representará el soporte de la componente Y. □

Los vectores $Y_1^T = (1\,0\,1\,1\,1\,0)$ e $Y_2^T = (0\,1\,0\,0\,0\,1)$ son componentes conservativas de la RdP de la figura 4.30. Sus soportes respectivos son: $||Y_1|| = \{p_1, p_3, p_4, p_5\}$ e $||Y_2|| = \{p_2, p_6\}$

Resulta evidente que si Y es una componente conservativa de una RdP, existe una infinidad de escalares $k \in \mathbb{N}^+$ tales que kY es también una componente de la RdP. De ahí la necesidad de su normalización.

Definición 4.26. Y es una *componente canónica* si el máximo común divisor (m.c.d.) de sus elementos no nulos es la unidad. □

De acuerdo con la definición anterior:

- $Y^T = (15\ 25\ 0\ 0\ 5)$ no es una componente canónica puesto que mcd($Y(1)$, $Y(2)$, $Y(5)$) = $5 \neq 1$,
- $Y^T = (3\ 5\ 0\ 0\ 1)$ es una componente canónica puesto que mcd($Y(1)$, $Y(2)$, $Y(5)$) = 1.

El conjunto de componentes conservativas canónicas de una RdP puede ser infinito, puesto que la suma ponderada con coeficientes no negativos de dos componentes es también una componente. A continuación vamos a definir un subconjunto (se demostrará que es finito) de componentes que tiene dos propiedades fundamentales:

1) permite generar todas las componentes (por definición);
2) su consideración es suficiente para extraer el *máximo de información* que se puede obtener a partir de todas las componentes (cf. §4.7.2.3).

Definición 4.27. Se llama *conjunto fundamental* de componentes conservativas de una RdP, $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_q\}$, al conjunto formado por el menor número posible de éllas que permita, mediante sumas, generar cualquier componente conservativa de la red: $Y = \sum_{Y_j \in \mathcal{Y}} k_j Y_j$, donde $k_j \in \mathbb{N}$ e $Y_j \in \mathcal{Y}$.

Las componentes $Y_j \in \mathcal{Y}$ se denominan *elementales*. □

Según esta definición, toda componente elemental debe ser canónica, puesto que, de lo contrario, el cardinal del conjunto fundamental no sería mínimo. Antes de continuar vamos a demostrar que el conjunto fundamental de las componentes conservativas de una RdP es efectivamente único y de cardinal finito.

Componentes elementales e inclusión de soportes (proposición 4.13). Una componente es elemental si y sólo si es canónica y su soporte no contiene al soporte de ninguna otra componente canónica. □

DEMOSTRACIÓN.

(1) Para que una componente sea elemental es necesario que su soporte no contenga ningún otro soporte de componente. Razonemos por reducción al absurdo: sean Y_i e Y_j dos componentes elementales e $||Y_i|| \supset ||Y_j||$. A partir de Y_i e Y_j se puede obtener al menos una relación $k_q Y_q = k_i Y_i - k_j Y_j$, siendo $k_i > 0$, $k_j > 0$, $k_q \geqslant 0$, e Y_q una componente. Por consiguiente, $k_i Y_i = k_j Y_j + k_q Y_q$, de donde, si k_q no es nulo ($k_q > 0$), Y_i no es elemental (contradicción). Ahora bien, si $k_q = 0$, ello implica $k_i Y_i = k_j Y_j$, lo cual significa que al menos una de las dos componentes Y_i o Y_j no es canónica y, por lo tanto, elemental (nueva contradicción).

(2) Para que una componente canónica Y_l sea elemental, es suficiente que su soporte no contenga ningún otro soporte de componente. En efecto, el soporte de la suma de componentes siempre contiene los soportes de las componentes que se suman, por lo que la única forma de representar la componente canónica Y_l (cuyo soporte no contiene ningún otro soporte de componente) es añadiéndola al conjunto generador. □

La *unicidad* del conjunto fundamental se puede inferir directamente de la proposición anterior, puesto que una componente elemental no se puede obtener sumando otras componentes elementales (su soporte incluirá los de las componentes que lo generan). Por otro lado, la proposición se puede reformular vectorialmente enunciando que los soportes de las componentes elementales de una RdP son vectores (booleanos) no comparables. Por consiguiente, el conjunto fundamental de las componentes conservativas de toda red posee menos de $2^n - 1$ componentes (n es el número de lugares y $2^n - 1$ es el número de vectores booleanos no nulos de n elementos); es decir, es finito.

Observación importante. El que los soportes de las componentes elementales de una RdP sean vectores (booleanos) incomparables, permite afirmar que el número de componentes elementales es no superior a $C_x^n = \binom{n}{x}$ donde $x = \left\lfloor \frac{n}{2} \right\rfloor$ o $x = \left\lceil \frac{n}{2} \right\rceil$. En efecto, consideremos, para simplificar, $n = 4$. La tabla 4.3 recoge los $2^4 - 1$ vectores booleanos no nulos. Los subconjuntos φ_i (i «unos» entre los 4 elementos) están formados por C_i^4 vectores incomparables. Así, $\varphi_2 = C_2^4 = 6$.

Veamos que no se puede aumentar el número de vectores incomparables de φ_2. En efecto,

1) si se añade un elemento de φ_1, por ejemplo b, se han de eliminar tres elementos de φ_2 (f, g, i) para que todos los vectores sigan siendo incomparables;

Tabla 4.3 Los φ_i son subconjuntos de vectores incomparables.

a	— — —	i	1 0 0 1
b	0 0 0 1	j	1 0 1 0
c	0 0 1 0	k	1 1 0 0
d	0 1 0 0	l	0 1 1 1
e	1 0 0 0	m	1 0 1 1
f	0 0 1 1	n	1 1 0 1
g	0 1 0 1	o	1 1 1 0
h	0 1 1 0	p	1 1 1 1

$\varphi_1 = \{b, c, d, e\}$

$\varphi_2 = \{f, g, h, i, j, k\}$

$\varphi_3 = \{l, m, n, o\}$

$\varphi_4 = \{p\}$

2) si se añade un elemento de φ_3, por ejemplo l, se han de eliminar tres elementos de φ_2 (f, g, h) para que todos los vectores sigan siendo incomparables;
3) si se añade un elemento (el único) de φ_4, se han de eliminar todos los elementos de φ_2 puesto que p los contiene todos.

De lo anterior se deduce que el cardinal del subconjunto de vectores booleanos incomparables φ_2 no puede ser incrementado. Para concluir que el máximo número de vectores incomparables posibles es el de φ_2 basta con considerar un subconjunto cualquiera de vectores incomparables, θ. Los elementos de θ que no pertenecen a φ_2 pueden ser sustituidos por al menos tantos elementos de φ_2, de forma que el conjunto resultante esté compuesto por vectores incomparables. De este modo, si $\theta = \{b, o\}$, b puede ser sustituido por $\{f, g, i\}$, mientras que o puede ser sustituido por $\{h, j, k\}$. Es decir, se llega a $\theta' = \{h, g, h, i, J, k\} = \varphi_2$, o en general a un $\theta' \subset \varphi_2$. El cardinal de φ_2 es máximo.

En general, si existen n elementos, los subconjuntos φ_i poseen C_i^n componentes incomparables, valor que se maximiza para $i = \left\lfloor \frac{n}{2} \right\rfloor$ o $i = \left\lceil \frac{n}{2} \right\rceil$.

En resumen, la importancia del conjunto fundamental de componentes reside en que, mediante sumas, permite generar todas las componentes y sólo éstas. Por otro lado, en §4.7.2.3 se comprobará cómo todo análisis basado en componentes puede realizarse de forma óptima considerando sólo las elementales (subconjunto finito).

4.7.2.2 *Obtención de todas las componentes elementales de una RdP*

En el párrafo anterior, §4.7.2.1, se introdujeron los conceptos de componente conservativa, soporte de una componente conservativa y conjunto fundamental de las

componentes conservativas. De forma dual se pueden definir las componentes repetitivas, sus soportes y conjunto fundamental.

Como resultado básico se dedujo que una componente canónica es elemental sii su soporte no contiene ningún otro soporte de componente (proposición 4.13). Este resultado permitió establecer que a toda RdP va asociado un único conjunto finito de componentes elementales a partir de las que, mediante sumas, se puede obtener cualquier otra componente. A continuación presentamos un algoritmo que calcula todas las componentes conservativas elementales de una RdP (su conjunto fundamental de componentes conservativas). Posteriormente se verá que el número de componentes elementales de una RdP puede ser inferior, igual o superior a la dimensión de cualquiera de las infinitas bases que permiten generar todos los anuladores (incluidos los no negativos o componentes) izquierdos de la matriz C. Si se opera con C^T, la aplicación del algoritmo que sigue generará todas las componentes repetitivas elementales.

a) Algoritmo

La estructura del algoritmo que se presenta a continuación es similar a la del que se presentó en §3.3.2. En aquel caso se pretendía determinar si un lugar cumplía la condición estructural para ser implícito: $\tilde{I}(p_i) = \sum_j \lambda_j \tilde{I}(p_j)$ y $\lambda_j \geqslant 0$. El cálculo de los λ_j que autorizan la escritura de $\tilde{I}(\bar{p}_i) + \sum_j \lambda_j \tilde{I}(p_j) = 0$, es el mismo que permite obtener las componentes conservativas cuyo soporte contiene al lugar $\bar{p}_i$ [recuérdese que $\tilde{I}(\bar{p}_i) = -\tilde{I}(p_i)$].

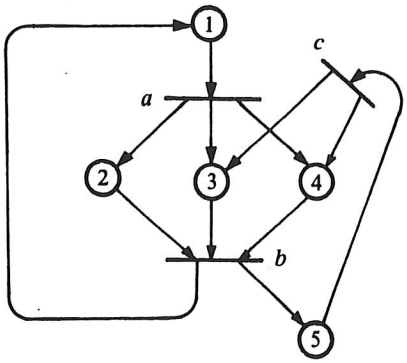
Antes de abordar la presentación y justificación del algoritmo es importante subrayar que no es óptimo, puesto que, además de las componentes conservativas elementales, se pueden generar otras componentes conservativas no elementales. En el anexo 4 se presentan las bases teóricas para mejorarlo y se incluye el listado de una codificación en PASCAL del algoritmo modificado.

ALGORITMO PARA LA OBTENCIÓN DE TODAS LAS COMPONENTES CONSERVATIVAS ELEMENTALES

(1) $A := C, D := I_n$ {matriz unitaria de dimensión n}
(2) **Para** $i := 1$ **hasta** $i = m$ {número de transiciones} **hacer**
  2.1 Añadir a la matriz $[D \vdots A]$ todas las filas que resulten como combinación lineal positiva de pares de filas de $[D \vdots A]$ y que anulen la i-ésima columna de A.
  2.2 Eliminar de $[D \vdots A]$ las filas en las que la i-ésima columna de A sea no nula.
(3) Las filas de la matriz D final representan componentes conservativas de la RdP. Entre éstas se encontrarán, tras una eventual normalización, todas las componentes elementales.

Antes de demostrar que este algoritmo genera todas las componentes conservativas elementales de una RdP, vamos a aplicarlo a una red.

EJEMPLO. Sea la RdP de la figura 4.37.



$$C = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{array} \begin{array}{c} \begin{array}{ccc} t_a & t_b & t_c \end{array} \\ \begin{bmatrix} -1 & +1 & 0 \\ +1 & -1 & 0 \\ +1 & -1 & +1 \\ +1 & -1 & +1 \\ 0 & +1 & -1 \end{bmatrix} \end{array}$$

$r = \text{rango}(C) = 3 \Rightarrow \text{rango}(B_r) = n - r = 5 - 3 = 2$

Figura 4.37. El número de componentes conservativas elementales (1) es inferior a la dimensión de la base de anuladores izquierdos de C, B_Y.

Paso 1:

$$[D^0 \vdots A^0] = \left[\begin{array}{ccccc:ccc} +1 & 0 & 0 & 0 & 0 & -1 & +1 & 0 \\ 0 & +1 & 0 & 0 & 0 & +1 & -1 & 0 \\ 0 & 0 & +1 & 0 & 0 & +1 & -1 & +1 \\ 0 & 0 & 0 & +1 & 0 & +1 & -1 & +1 \\ 0 & 0 & 0 & 0 & +1 & 0 & +1 & -1 \end{array}\right] \quad \begin{array}{l} \text{n.° de fila} \\ (1) \\ (2) \\ (3) \\ (4) \\ (5) \end{array}$$

Paso 2:

- *Iteración* n.° 1:

(2.1) Las filas que se deben añadir son:

$$\left[\begin{array}{ccccc:ccc} +1 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ +1 & 0 & +1 & 0 & 0 & 0 & 0 & +1 \\ +1 & 0 & 0 & +1 & 0 & 0 & 0 & +1 \end{array}\right] \quad \begin{array}{l} (1)+(2) \\ (1)+(3) \\ (1)+(4) \end{array}$$

(2.2) Las filas que han de eliminarse son las cuatro primeras de $[D^0 \vdots A^0]$; es decir (1), (2), (3), (4).

En resumen, la matriz que resulta es:

$$[D \vdots A] = \left[\begin{array}{ccccc:ccc} 0 & 0 & 0 & 0 & +1 & 0 & +1 & -1 \\ +1 & +1 & 0 & 0 & 0 & 0 & 0 & 0 \\ +1 & 0 & +1 & 0 & 0 & 0 & 0 & +1 \\ +1 & 0 & 0 & +1 & 0 & 0 & 0 & +1 \end{array}\right] \quad \begin{array}{l} (5) \\ (1)+(2) \\ (1)+(3) \\ (1)+(4) \end{array}$$

- *Iteración* n.° 2:

(2.1) No se puede anular la 2ª columna de A_1 sumando filas, por lo cual no se añade ninguna fila nueva.

(2.2) Se ha de eliminar la primera fila de $[D^1 \vdots A^1]$: (5).

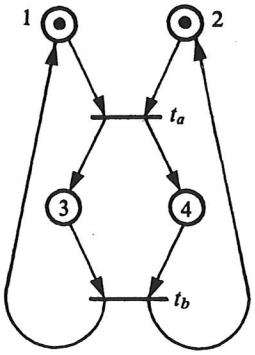
- *Iteración* n.° 3:

(2.1) No se puede añadir ninguna fila nueva.

(2.2) Han de eliminarse las filas 2 y 3 de la matriz que quedaba; es decir, (1) + (3), (1) + (4).

Paso 3 (fin del algoritmo): Queda la fila (1 1 0 0 0 0 0 0), luego se deduce que (1 1 0 0 0) es la única componente conservativa elemental de la RdP.

EJERCICIO. Obténganse las componentes conservativas elementales de las RdP de las figuras 4.35, 4.38 y 3.5 (obsérvese cómo en el último caso se genera, además de las componentes elementales, alguna no elemental).



$$C = \begin{array}{c} \\ p_1 \\ p_2 \\ p_3 \\ p_4 \end{array} \begin{array}{c} \begin{array}{cc} t_a & t_b \end{array} \\ \begin{bmatrix} -1 & +1 \\ -1 & +1 \\ +1 & -1 \\ +1 & -1 \end{bmatrix} \end{array}$$

$$r = \text{rango}(C) = 1 \Rightarrow \text{rango}(B_Y) = n - r = 4 - 1 = 3$$

Figura 4.38. El número de componentes conservativas elementales (4) es superior a la dimensión de la base B_Y (3).

Sea $\boldsymbol{R}^i$ la RdP resultante de eliminar en R las transiciones $i+1, i+2, \ldots, m$. Para demostrar que el algoritmo anterior genera todas las componentes conservativas elementales, se razonará por inducción:

1) Inicialmente las filas de la matriz D ($D^0 \equiv I_n$) contienen las componentes elementales de $\boldsymbol{R}^0$, red sin transición alguna. En efecto, en $\boldsymbol{R}^0$ cada lugar define una componente elemental.
2) Partiendo de las componentes elementales de $\boldsymbol{R}^i$, el algoritmo genera al menos todas las componentes elementales de $\boldsymbol{R}^{i+1}$. Esta afirmación se establece dado que:
   a) Toda componente de $\boldsymbol{R}^{i+1}$, Y_j^{i+1}, será también componente de $\boldsymbol{R}^i$ y las filas de D^i (por hipótesis en la inducción) contienen al conjunto fundamental de $\boldsymbol{R}^i$.
   b) Cualquier componente Y_j^{i+1} generada como combinación lineal positiva de más de dos componentes elementales de $\boldsymbol{R}^i$ no será elemental. En efecto, esto se cumplirá ya que su soporte será superior o igual a alguno de los obtenidos al combinar los pares de componentes de $\boldsymbol{R}^i$ que anulan la $(i+1)$-ésima columna de A y se incurre en contradicción con el resultado anunciado por la proposición 4.13. □

b) Conjunto fundamental y bases de anuladores

La aplicación del algoritmo anterior a la RdP de la figura 4.37 demostró que la mencionada red posee una única componente conservativa elemental: $Y^T = (1\,1\,0\,0\,0)$.

Por otro lado, procediendo según el algoritmo presentado en el anexo 3, se obtiene una (entre las infinitas) base de anuladores izquierdos de la matriz C (de dimensión dos):

$$B_Y^T = \begin{pmatrix} 0 & 0 & +1 & -1 & 0 \\ +1 & +1 & 0 & 0 & 0 \end{pmatrix}.$$

Este ejemplo pone de relieve que el cardinal del conjunto de componentes conservativas elementales puede ser *inferior* a la dimensión del espacio de vectores anuladores izquierdos de la matriz C. Dicho de otra forma, a veces, a partir del conjunto fundamental no se puede generar todo el espacio de las soluciones del sistema $Y^T \cdot C = 0$, o lo que es lo mismo todo el conjunto de invariantes de marcado de la forma: $Y^T \cdot M = Y^T \cdot M_0 = b$.

El aspecto negativo que, de cara al análisis utilizando exclusivamente componentes conservativas, establece la observación anterior es la insuficiencia del método para afirmar o negar ciertas aserciones, sobre las cuales sí se puede concluir utilizando todo tipo de invariantes lineales de marcado (§4.7.1). A modo de ejemplo basta con considerar $A(M) \equiv M(p_3) - M(p_4) = 0$ y la RdP de la figura 4.37 con $M_0^T = (1\,0\,0\,0\,0)$. El invariante no negativo, $M(p_1) + M(p_2) = 1$, no permite afirmar o negar $A(M)$. Si se considera el primer vector de B_Y^T, $Y_1^T = (0\,0\,1\,-1\,0)$, se deduce inmediatamente que $A(M)$ se verifica: $Y_1^T \cdot M = M(p_3) - M(p_4) = Y_1^T \cdot M_0 = 0$.

Si estudiamos ahora la RdP de la figura 4.38, se obtienen cuatro componentes conservativas elementales: $Y = (1\,0\,0\,1)^T, (1\,0\,1\,0)^T, (0\,1\,0\,1)^T, (0\,1\,1\,0)^T$. Por otro lado, toda base de anuladores izquierdos de C (figura 4.38) es de dimensión tres. Este nuevo ejemplo, permite constatar que el número de componentes conservativas elementales puede ser *superior* a la dimensión de la base del espacio de anuladores izquierdos de C. De donde se deduce, naturalmente, que existen componentes conservativas elementales que son linealmente dependientes. Dicho de otro modo, la representación de una componente como suma de componentes elementales no tiene que ser única. Así, la componente $(1\,1\,1\,1)^T$ admite dos representaciones distintas:

$$(1\,0\,0\,1)^T + (0\,1\,1\,0)^T$$
$$(1\,0\,1\,0)^T + (0\,1\,0\,1)^T.$$

Después de lo anterior cabe preguntarse ¿en qué casos el conjunto fundamental de componentes conservativas puede, mediante sumas y restas, generar todos los anuladores izquierdos de C?

Bases de anuladores y componentes (proposición 4.14). Si y sólo si una RdP es conservativa, se puede obtener una base de anuladores formada exclusivamente por componentes conservativas elementales. □

El interés del resultado anterior reside en que enuncia una condición necesaria y suficiente para que todo análisis que utilice una base de invariantes (§4.7.1.1) pueda ser realizado (de forma «más agradable» como se verá a continuación en §4.7.2.3) utilizando sólo invariantes de marcado derivados de las componentes conservativas

(posteriormente se verá como sólo es necesario considerar los invariantes derivados de las componentes conservativas elementales, del conjunto fundamental de componentes conservativas).

EJERCICIO. Demuéstrese la proposición 4.14. (*Sugerencia*: constrúyase en primer lugar una base con vectores no negativos, componentes. Posteriormente redúzcanse las componentes de la base a otras elementales.)

4.7.2.3 *Aplicaciones de las componentes conservativas al análisis de RdP*

Sea $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_q\}$ el conjunto fundamental de componentes conservativas de una RdP. Cada una de las componentes conservativas elementales, Y_i, define un invariante elemental de marcado (se obtiene premultiplicando la ecuación de estado): $Y_i^T \cdot M = Y_i^T \cdot M_0 = b$. Un *invariante elemental* de marcado expresa una ley de conservación de marcas que no es deducible mediante sumas a partir de otros invariantes elementales.

En este apartado se presenta una serie de resultados que a partir de las componentes conservativas permiten analizar las RdP. En especial se insistirá sobre la idea de que todo análisis realizado utilizando componentes conservativas (su número es infinito) puede llevarse a cabo de forma *óptima* considerando sólo las componentes conservativas elementales (subconjunto finito). Dicho de otro modo, toda propiedad sobre el comportamiento de una RdP deducible a partir de un invariante no elemental puede deducirse, e incluso estudiarse mejor, a partir de los invariantes elementales que lo generan. Este importante resultado tiene lugar habida cuenta que, como se ha repetido varias veces, toda componente (o el invariante que representa) se obtiene mediante sumas de componentes elementales.

A continuación se analizan la limitación, la exclusión mutua y la vivacidad. También se considera la detección de lugares implícitos.

A partir de las componentes conservativas elementales cuyos soportes contienen al lugar p, puede determinarse un límite superior del marcado del mismo.

Componentes conservativas y limite de un lugar (proposición 4.15). La mejor acotación superior del marcado de un lugar p que puede obtenerse a partir de todas las componentes conservativas, es calculable a partir del conjunto de componentes conservativas elementales:

$$M(p) \leqslant \min_{Y_i \in \mathcal{Y}} \left\lfloor \frac{Y_i^T \cdot M_0}{Y_i(p)} \right\rfloor . \quad \square$$

DEMOSTRACIÓN. Una justificación de que el valor de la expresión anterior es un límite superior del marcado del lugar p, se puede obtener razonando de la misma forma que en la demostración de la proposición 4.7.

Para demostrar que el límite obtenido es el mejor que se puede determinar a partir de las componentes conservativas, considérese la componente conservativa no elemental: $Y = \sum_i f_i Y_i$, $f_i \in \mathbb{N}$. Por otro lado, sea

$$\frac{Y_i^T \cdot M_0}{Y_i(p)} = \frac{a_i}{b_i} \quad \text{y} \quad \min_{Y_i \in \mathcal{Y}} \left\lfloor \frac{Y_i^T \cdot M_0}{Y_i(p)} \right\rfloor = \frac{\bar{a}}{\bar{b}} \leqslant \frac{a_i}{b_i}.$$

De la anterior desigualdad se desprende que $\bar{a}b_i \leqslant \bar{b}a_i$; multiplicando por f_i y sumando, dado que todo $f_i \geqslant 0$, se puede escribir:

$$\bar{a}\sum f_i b_i \leqslant \bar{b}\sum f_i a_i \Rightarrow \frac{\bar{a}}{\bar{b}} \leqslant \frac{\sum_i f_i a_i}{\sum_i f_i b_i} \Rightarrow \left\lfloor \frac{\bar{a}}{\bar{b}} \right\rfloor \leqslant \left\lfloor \frac{\sum_i f_i a_i}{\sum_i f_i b_i} \right\rfloor,$$

luego el límite obtenido por $\lfloor \bar{a}/\bar{b} \rfloor$ es mínimo; la acotación del marcado de p no se puede mejorar considerando componentes no elementales. □

Teniendo en cuenta la RdP de la figura 4.35 obtendremos (todos los límites se alcanzan en este caso):

$$M(p_1) \leqslant \frac{(1\,0\,0\,1\,0)M_0}{1} = 1$$

$$M(p_2) \leqslant \frac{(0\,1\,0\,1\,1)M_0}{1} = 1$$

$$M(p_3) \leqslant \frac{(0\,0\,1\,0\,1)M_0}{1} = 3$$

$$M(p_4) \leqslant \min\left\{\frac{(1\,0\,0\,1\,0)M_0}{1}, \frac{(0\,1\,0\,1\,1)M}{1}\right\} = \min\{1, 1\} = 1$$

$$M(p_5) \leqslant \min\left\{\frac{(0\,0\,1\,0\,1)M_0}{1}, \frac{(0\,1\,0\,1\,1)M}{1}\right\} = \min\{3, 1\} = 1.$$

El establecimiento de una condición suficiente, con el fin de garantizar la exclusión mutua entre los marcados de dos lugares pertenecientes a una (varias) componente conservativa elemental es inmediato.

Exclusión mutua entre dos lugares (proposición 4.16).

1) Sean los lugares $p_i \in ||Y||$ y $p_j \in ||Y||$. Si $Y(i) + Y(j) > Y^T \cdot M_0$, entonces los marcados de p_i y p_j estarán en exclusión mutua.
2) Toda exclusión mutua deducible a partir de un invariante Y puede obtenerse a partir de los invariantes elementales que lo generan. □

DEMOSTRACIÓN

1) $Y^T \cdot M_0 = \sum_{k=1}^{n} Y(k)M_0(p_k) \geqslant Y(i)M(p_i) + Y(j)M(p_j)$.
   Ahora bien, si $Y(i) + Y(j) > Y^T \cdot M_0$, de acuerdo con la expresión anterior, tendremos necesariamente $M(p_i)M(p_j) = 0$. Es decir, p_i y p_j no pueden estar simultáneamente marcados.
2) Una demostración similar a la considerada en la proposición 4.15 permite concluir que los resultados del análisis realizado con los invariantes elementales no pueden mejorarse al considerar otros invariantes no elementales □.

Volviendo sobre la RdP de la figura 4.35 y considerando $Y^T = (0\,1\,0\,1\,1)$, se demuestra que p_2, p_4 y p_5 están en exclusión mutua.

Desafortunadamente, la proposición 4.16 no permite determinar todas las exclusiones mutuas posibles entre los marcados de los lugares de una red. Así, las compo-

rifica la segunda condición de la proposición 3.2 y, por consiguiente, p_i es un lugar potencialmente implícito (véase el primer ejercicio después del algoritmo, §3.3.2). □

Si estudiamos detenidamente el anterior razonamiento, se observará que el soporte de la componente conservativa definida por el vector $k\bar{Y} - Y$, es un conjunto de lugares que podrían ser implicantes de $p_i{:}Q = ||k\bar{Y} - Y||$.

Esta última es una información fundamental, puesto que si $\bar{R}$ es pura, de acuerdo con la proposición 3.3, para determinar si p_i es un lugar implícito sólo se tiene que comprobar la primera condición de la proposición 3.2 con $\mu \geqslant 0$. En el ejemplo que sigue se presenta una de las posibles formas de utilización del resultado anterior.

EJEMPLO. La red de Petri de la figura 3.3*d* es conservativa, y se puede escribir $Y_0^T = (1\,1\,1\,1\,1\,1\,1\,2)$. Si suprimimos p_1, lugar potencialmente implícito, tendremos que R_1 es también conservativa. Por ejemplo: $Y_1 = (0\,1\,1\,1\,2\,2\,2\,2)$. A partir de esto se tendrá:

$$k_1 = \max_{j>1}\left[\frac{Y_0(j)}{Y_1(j)}\right] = 1, \text{ luego } Y_1^T - Y_0^T = (-1\,0\,0\,0\,1\,1\,1\,0),$$

es decir, $I(p_1) = I(p_5) + I(p_6) + I(p_7)$.

Dado que $M_0(p_1) = M_0(p_5) + M_0(p_6) + M_0(p_7)$ y que la red es pura, se puede concluir que p_1 es implícito (lo implica $Q = \{p_5, p_6, p_7\}$).

La subRdP que se obtiene al eliminar p_1 (lugar implícito) y p_2 (potencialmente implícito), es también conservativa. Por ejemplo, $Y_2 = (0\,0\,1\,1\,1\,1\,2\,1)$, de donde:

$$k_2 = \max_{j>2}\left[\frac{Y_1(j)}{Y_2(j)}\right] = 2, \text{ luego } 2Y_2 - Y_1 = (0\,-1\,1\,1\,0\,0\,2\,0),$$

es decir, $I(p_2) = I(p_3) + I(p_4) + 2I(p_7)$. Puesto que se verifica la condición sobre el marcado inicial y la red es pura, p_2 es implícito. (*Nota.* También $I(p_2) = I(p_3) + I(p_7)$.)

La subRdP que se obtiene al eliminar p_1 y p_2 (lugares implícitos) y p_4 (potencialmente implícito) es también conservativa. Por ejemplo:

$$Y_3 = (0\,0\,1\,0\,1\,1\,1\,1) = j \in \{3, 5, 6, 7, 8\}$$

$$k_3 = \max_{j}\left[\frac{Y_2(j)}{Y_3(j)}\right] = 2, \text{ luego } 2Y_3 - Y_2 = (0\,0\,1\,-1\,1\,1\,0\,1),$$

es decir, $I(p_4) = I(p_3) + I(p_5) + I(p_6) + I(p_8)$. Dado que se verifica la condición sobre el marcado inicial y la red es pura, p_4 es implícito. En conclusión, en la RdP de la figura 3.3*d* se pueden eliminar $\{p_1, p_2, p_4\}$.

Observación. Procediendo de esta forma, no se pueden eliminar los lugares implícitos que no pertencen a ninguna componente conservativa. Por ejemplo, porque sean lugares no limitados. Tal es el caso de p_3 o p_4 en la RdP de la figura 4.37.

A modo de conclusión, la estrategia de validación de una descripción basada en la utilización de invariantes de marcado se puede llevar a cabo en tres fases:

1) Formulación de las propiedades que se desean validar.
2) Extracción de los invariantes elementales de marcado de la RdP.

3) Utilización de aquellos invariantes que permitan demostrar las propiedades formuladas en la primera fase.

Desde un punto de vista práctico, estos métodos suelen ser bastante eficientes, especialmente si se combinan con métodos de reducción de redes (§4.5). Ahora bien, la reducción que se realice en la RdP no puede ser total puesto que la red deberá dejarse como irreducible cuando la aplicación de alguna regla de reducción obligue a eliminar algún lugar o transición que aparezca en las propiedades que se deseen verificar. Esta forma de proceder es la adoptada en el ejemplo de lectores y redactores que consideramos a continuación.

EJEMPLO. LECTORES Y REDACTORES. Sea el caso presentado en §2.4.4. Pretendemos verificar que el modelo construido (figura 2.9) cumple las propiedades siguientes:

a) es vivo y binario,
b) existe exclusión mutua entre redactores,
c) existe exclusión mutua entre cada redactor y el conjunto de lectores (no se puede leer y escribir al mismo tiempo).

Para abordar la validación, vamos a utilizar técnicas de reducción de redes, así como técnicas derivadas de la conservatividad. En primer lugar, reduciremos la RdP de la figura 2.9 hasta que sin sustituir o eliminar los lugares AL_i y AL_j, la red sea irreducible. Sobre la RdP reducida (figura 4.39), estudiaremos las exclusiones mutuas. Por último, reduciremos completamente la RdP de la figura 4.39 y concluiremos sobre la vivacidad y la k-limitación de la RdP original.

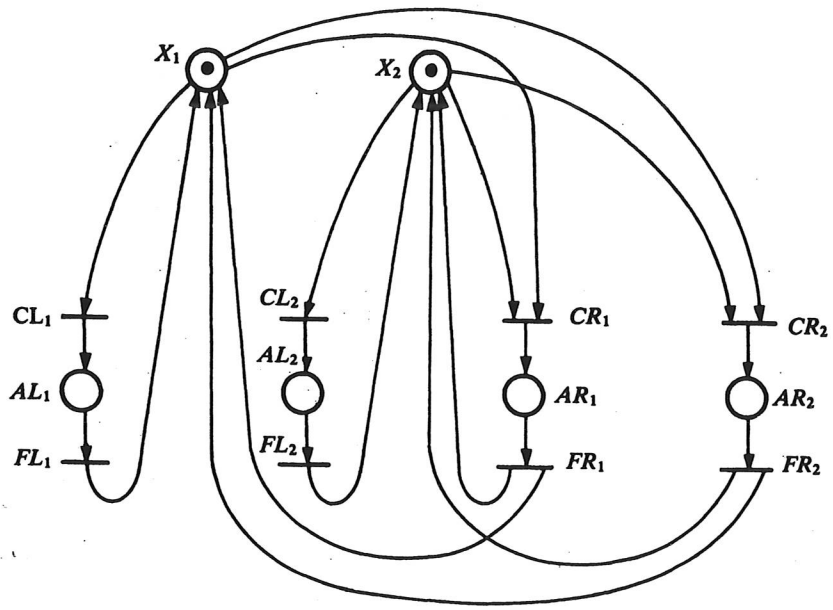


Figura 4.39. Una primera reducción de la RdP de la figura 2.29.

1) Reducción previa de la RdP de la figura 2.9:

a) Se eliminan (regla $\mathfrak{R}_1$) los lugares EL_1, EL_2, ER_1 y ER_2.

b) Se eliminan (regla $\mathfrak{R}_3$) los lugares implícitos RL_1, RL_2, RR_1 y RR_2.

Si para verificar las propiedades de exclusión mutua enunciadas no se pueden eliminar los lugares AL_1, AL_2, AR_1 y AR_2, la RdP obtenida (figura 4.39) es irreducible.

2) Estudio de las exclusiones mutuas:

A partir de la RdP de la figura 4.39 obtendremos los invariantes elementales de marcado siguientes:

$$M(AL_1) + M(AR_1) + M(AR_2) + M(X_1) = 1$$
$$M(AL_2) + M(AR_1) + M(AR_2) + M(X_2) = 1.$$

Aplicando la proposición 4.16 podemos concluir que:

$$-M(AR_1)M(AR_2) = 0$$
$$-M(AL_i)M(AR_j) = 0 \quad \{i = 1, 2; \quad j = 1, 2\}.$$

La primera de ambas expresiones indica que los redactores están en exclusión mutua. La segunda de las expresiones nos permite escribir $(M(AL_1) + M(AL_2)) \cdot (M(AR_1) + M(AR_2)) = 0$; es decir, lectores y redactores están en exclusión mutua.

En resumen, se verifican las propiedades *b* y *c*.

3) Reducción final. Estudio de la vivacidad y de la limitación:

a) Se sustituyen (eliminan) los lugares AL_i y AR_i $(i = 1, 2)$.

b) Se eliminan las transiciones CL_i-FL_i. La RdP que se obtiene es la de la figura 4.40.

c) X_1 (o X_2) es implícito, por lo que la RdP original es viva y 1-limitada (binaria).

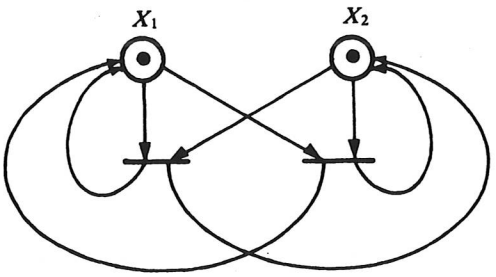


Figura 4.40. Reducción de la RdP de la figura 4.39.

EJEMPLO. DOS CARROS QUE VAN Y VIENEN. Se desea comprobar que la descripción del sistema de dos carros que van y vienen (figura 1.15*a*) posee las propiedades siguientes:

a) es viva y binaria;

b) nunca se pueden encontrar dos carros circulando en sentidos opuestos.

Al igual que en el ejemplo anterior se combinan técnicas de reducción y las basadas en la conservatividad.

1) Reducción previa de la RdP de la figura 1.15*a*:

Aplicando la regla de reducción $\mathfrak{R}_1$ se obtiene la RdP de la figura 4.38. Para comprobar que dos carros nunca pueden circular en sentidos opuestos, basta con demostrar que, en la RdP de la figura 4.38, los lugares p_1 y p_2 se encuentran en exclusión mutua con los lugares p_3 y p_4. Por esta razón, no se puede continuar aplicando reglas de reducción.

2) Estudio de las exclusiones mutuas (figura 4.38):

Sus componentes conservativas elementales fueron obtenidas con anterioridad. Los invariantes de marcado correspondiente son:

$$\left.\begin{aligned} M(p_1) + M(p_3) &= 1 \\ M(p_1) + M(p_4) &= 1 \end{aligned}\right\} \Rightarrow M(p_1)(M(p_3) + M(p_4)) = 0$$

$$\left.\begin{aligned} M(p_2) + M(p_3) &= 1 \\ M(p_2) + M(p_4) &= 1 \end{aligned}\right\} \Rightarrow M(p_2)(M(p_3) + M(p_4)) = 0$$

en resumen, se podrá escribir $(M(p_1) + M(p_2)) \cdot (M(p_3) + M(p_4)) = 0$, ecuación que demuestra la propiedad buscada.

3) Reducción final. Estudio de la vivacidad y de la limitación: En la RdP de la figura 4.38, p_1 o p_2 y p_3 o p_4 son implícitos. Eliminando, pongamos por caso, p_2 y p_4, se obtiene una RdP en la que p_3 es sustituible. La sustitución de p_3 permite obtener una RdP con un único lugar, p_1, marcado con una marca y una transición. La RdP es viva y 1-limitada (binaria).

EJERCICIO. Demostrar que las reglas de reducción presentadas en §4.5 son tales que la red reducida es conservativa sii lo es la red original.

4.7.3 Invariantes de disparo

Hemos visto que en el marco de la teoría de RdP, los lugares y las transiciones representan papeles duales. En este apartado, se introducen los *invariantes de disparo* (asociados al disparo de las transiciones), los cuales vienen a añadirse los ya considerados invariantes de marcado (asociados al marcado de los lugares).

El material que presentamos no constituye más que una somera introducción. En ésta, estableceremos algunas relaciones entre conceptos aparentemente tan dispares como los de avance sincrónico, lugar implícito, componentes conservativas y componentes repetitivas de una RdP.

El invariante de disparo básico es el lugar. En efecto, si para simplificar los razonamientos considerásemos RdP *puras*, de acuerdo con la ecuación de estado, es posible escribir:

$$M = M_0 + \boldsymbol{C} \cdot \bar{\sigma} \geqslant 0 \Rightarrow \boxed{\forall \bar{\sigma} \in \bar{L}(\boldsymbol{R}, M_0) \qquad M_0(p) + \boldsymbol{I}(p) \cdot \bar{\sigma} \geqslant 0}.$$

Ahora bien, si $\boldsymbol{R}$ es una RdP *ordinaria* y si se definen los subconjuntos de transiciones $T_1 = p^{\cdot}$ y $T_2 = {}^{\cdot}p$, se puede establecer:

$$\forall \bar{\sigma} \in \bar{L}(\boldsymbol{R}, M_0) \qquad M_0(p) \geqslant -\boldsymbol{I}(p) \cdot \bar{\sigma} = (\bar{T}_1 - \bar{T}_2)^T \cdot \bar{\sigma} \Rightarrow$$
$$M_0(p) \geqslant \max_{\sigma} [(\bar{T}_1 - \bar{T}_2)^T \cdot \bar{\sigma}] = AV(\boldsymbol{R}, M_0; T_1, T_2) \Rightarrow$$
$$\boxed{M_0(p) \geqslant AV(\boldsymbol{R}, M_0; p^{\cdot}, {}^{\cdot}p)}.$$

De esta forma, cualquier RdP ordinaria y pura puede ser descrita por un conjunto de n invariantes del tipo $AV(\boldsymbol{R}, M_0; p\cdot, \cdot p) \leqslant z$, donde cada lugar define un *invariante de disparo.*

EJEMPLO. La RdP de la figura 4.35 puede ser descrita por los invariantes que siguen (sea $AV(\boldsymbol{R}, M_0; T_1, T_2) = AV(T_1, T_2)$):

$$
\begin{aligned}
&p_1 : AV(t_1, t_2) \leqslant 1\\
&p_2 : AV(\{t_1, t_3\}, \{t_2, t_4\}) \leqslant 1\\
&p_3 : AV(t_3, t_4) \leqslant 3\\
&p_4 : AV(t_2, t_1) = 0\\
&p_5 : AV(t_4, t_3) = 0.
\end{aligned}
$$

La potencia de estos sencillos razonamientos se puede constatar con el siguiente ejemplo, en el que se demuestra el «Teorema del semáforo». (Véase, por ejemplo, [CROC 75].)

EJEMPLO. SEMÁFORO. Por similitud con el semáforo de tráfico (urbano, ferroviario, etc.), un semáforo (informático) es un mecanismo de sincronización entre procesos (programas de ejecución).

El lugar s de la figura 4.41 representa un *semáforo*. El disparo de la transición t_v corresponde a la ejecución de una primitiva *señalar* (primitiva V_s). El disparo de la transición t_p corresponde a la ejecución de una primitiva *esperar* (primitiva P_s). La transición t_f se dispara al sensibilizarse. El lugar e_s representa la *cola de espera* asociada al semáforo.

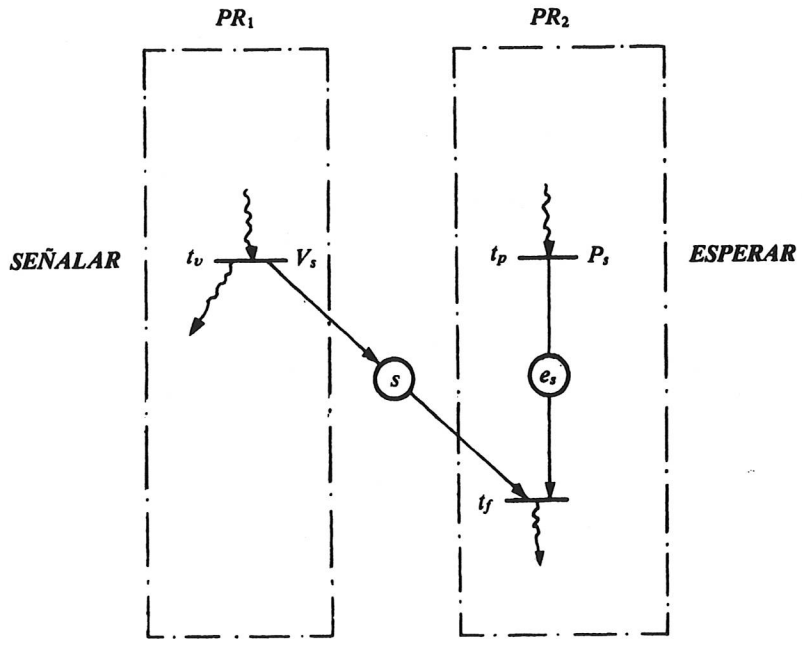


Figura 4.41. SubRdP que representa el semáforo S.

El funcionamiento del semáforo está definido por la subRdP de la figura 4.41, no obstante, puede definirse informalmente, y en pocas palabras, diciendo que el proceso PR_2 es bloqueado en su evolución si el proceso PR_1 no ha ejecutado un número suficiente de primitivas señalar (primitiva V_s). El semáforo está inicializado con $M_0(s)$ marcas. Cada una de esas marcas representa una autorización de evolución que se le otorga al proceso PR_2, sin que necesite la previa ejecución de ninguna primitiva *señalar*.

Una vez presentado el mecanismo de sincronización denominado semáforo, escribimos los invariantes de disparo asociados a los dos lugares, el semáforo y su cola de espera:

$$P_s{:}AV(\boldsymbol{R}, M_0; t_f, t_v) \leqslant M_0(s) \Rightarrow \bar{\sigma}(t_f) \leqslant \bar{\sigma}(t_v) + M_0(s)$$
$$P_e{:}AV(\boldsymbol{R}, M_0; t_f, t_p) = 0 \Rightarrow \bar{\sigma}(t_f) \leqslant \bar{\sigma}(t_p).$$

A partir de las dos inecuaciones anteriores, y teniendo en cuenta que la transición t_f se dispara al sensibilizarse, se obtiene:

$$\bar{\sigma}(t_f) = \min\{\bar{\sigma}(t_p), M_0(s) + \bar{\sigma}(t_v)\},$$

expresión que se conoce como *teorema del semáforo*.

A continuación, introducimos un resultado que permite calcular el avance sincrónico. Como corolario se establece una relación entre avances sincrónicos y lugares implícitos.

Sea $\langle \boldsymbol{R}, M_0 \rangle$ una RdP ordinaria marcada y T_1 y T_2 dos subconjuntos disjuntos de transiciones de $\boldsymbol{R}$ [$T_1 \subset T, T_2 \subset T, T_1 \cap T_2 = \emptyset$]. Se define el lugar $p_{T_1T_2}$ de acuerdo con las expresiones: ${}^{\cdot}p_{T_1T_2} = T_2$ y $p_{T_1T_2}^{\cdot} = T_1$; es decir, $p_{T_1T_2}$ (figura 4.42) es un lugar cuyo conjunto de transiciones de entrada (respec. de salida) coincide con T_2 (respec. T_1). Por otro lado, sea $\langle \boldsymbol{R}^*, M_0^* \rangle$ la red marcada obtenida al añadir el lugar $p_{T_1T_2}$ a $\langle \boldsymbol{R}, M_0 \rangle$.

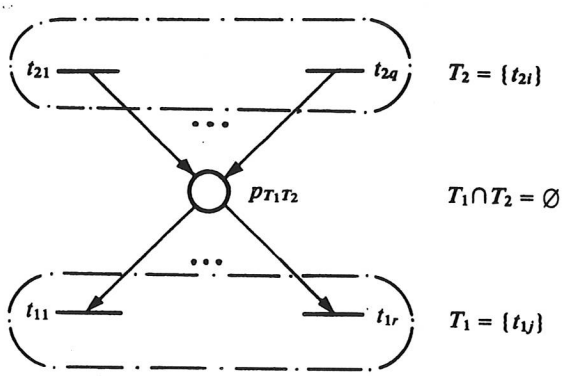


Figura 4.42. Representación gráfica del lugar $p_{T_1T_2}$.

Avance sincrónico y secuencias de disparos (proposición 4.19). El avance sincrónico entre T_1 y T_2, $AV(\boldsymbol{R}, M_0; T_1, T_2)$, es igual al mínimo número de marcas que inicialmente debe poseer el lugar $p_{T_1T_2}$ para que $\langle \boldsymbol{R}, M_0 \rangle$ y $\langle \boldsymbol{R}^*, M_0^* \rangle$ admitan el mismo conjunto de secuencias de disparos, $L(\boldsymbol{R}, M_0) = L(\boldsymbol{R}^*, M_0^*)$. □

DEMOSTRACIÓN. Sea τ el avance de T_1 con respecto a T_2. Según la definición 4.12:

$$\tau = \max_{\forall \sigma \in L(R, M_0)} \{(\bar{T}_1 - \bar{T}_2)^T \cdot \bar{\sigma}\} \geqslant \{(\bar{T}_1 - \bar{T}_2)^T \cdot \bar{\sigma}\}$$

De la anterior expresión se deduce que τ es el menor entero tal que $\forall \sigma \in \boldsymbol{L}(\boldsymbol{R}, M_0)$ se puede escribir: $\tau + (\bar{T}_2 - \bar{T}_1)^T \cdot \bar{\sigma} \geqslant 0$

Si T_1 y T_2 son disjuntos, $T_1 \cap T_2 = \emptyset$, entonces, $\forall i = 1, \ldots, m$, $\bar{T}_1(i) \cdot \bar{T}_2(i) = 0$ y, por consiguiente:

$$\forall \bar{\sigma} \in \bar{L}(R, M_0) \qquad \tau + (\bar{T}_2 - \bar{T}_1)^T \cdot \bar{\sigma} \geqslant 0 \Leftrightarrow \tau - \bar{T}_1^T \cdot \bar{\sigma} \geqslant 0$$

Dado que $\boldsymbol{R}$ es una RdP ordinaria, la última inecuación significa que, para cualquier secuencia perteneciente a $\langle \boldsymbol{R}, M_0 \rangle$, si $M_0^*(p_{T_1T_2}) = \tau$, el lugar $p_{T_1T_2}$ no es nunca la única restricción para el disparo de sus transiciones de salida. En conclusión, la introducción de $p_{T_1T_2}$ con $M_0(p_{T_1T_2}) = \tau$ no restringe el conjunto de secuencias disparables. Puesto que la introducción de un nuevo lugar sólo puede reducir el conjunto de secuencias disparables, se infiere que $\boldsymbol{L}(\boldsymbol{R}, M_0) = \boldsymbol{L}(\boldsymbol{R}^*, M_0^*)$. □

Avance sincrónico y lugares implícitos (corolario 4.12). Sea el lugar $p_{T_1T_2}$. Si $M_0^*(p_{T_1T_2}) = \tau$ es el menor entero que le hace ser implícito en $\langle \boldsymbol{R}^*, M_0^* \rangle$, entonces $AV(\boldsymbol{R}, M_0; T_1, T_2) = \tau$. □

La justificación de este corolario es inmediata, habida cuenta que, para que un lugar sea implícito, es necesario que no introduzca restricciones sobre las secuencias de disparos.

Antes de presentar un ejemplo en el que apliquemos estos resultados, es importante mencionar que, tanto la proposición 4.19 como el corolario 4.12 mantienen su vigencia si se habla de RdP Generalizadas y de avances ponderados.

Cuando se pretende determinar si existe un avance (ponderado) entre dos conjuntos disjuntos de transiciones, T_1 y T_2, puede utilizarse el corolario anterior. Un procedimiento de trabajo consiste en:

1) Definir un lugar $p_{T_1T_2}$ ($^{\cdot}p_{T_1T_2} = T_2$ y $p^{\cdot}_{T_1T_2} = T_1$) tal que:

$$\alpha(p_{T_1T_2}, t_{1i}) = \alpha_i \text{ y } \beta(t_{2j}, p_{T_1T_2}) = \beta_j$$

2) El *avance sincrónico ponderado* viene dado por el mínimo valor $M_0(p_{T_1T_2})$ que hace implícito a $p_{T_1T_2}$ (corolario 4.12).

Como es sabido, para que $p_{T_1T_2}$ sea implícito, es condición necesaria que $\boldsymbol{l}(p_{T_1T_2})$ sea combinación lineal (con coeficientes positivos) de las demás filas de la matriz de incidencia de la RdP, $\boldsymbol{C}$. Esto implica que si $\boldsymbol{C}^*$ es la matriz de incidencia de la RdP obtenida al añadirle $p_{T_1T_2}$ a la red original, entonces rango($\boldsymbol{C}$) = rango($\boldsymbol{C}^*$). Esta ecuación impone condiciones sobre los α_i y β_j.

EJEMPLO. Se pretende determinar si existe un avance ponderado finito entre las transiciones t_1 y t_2 de la RdP de la figura 4.7.

Aplicando el método anteriormente esbozado:

$$C^* = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 \end{matrix} \\ \left(\begin{array}{ccc} -1 & +1 & +1 \\ +1 & -1 & -1 \\ 0 & -1 & +1 \\ 0 & +1 & -1 \\ \hline -\alpha & \beta & 0 \end{array}\right) \end{array} \begin{array}{l} p_1 \\ p_2 \\ p_3 \\ p_4 \\ \hline p_{12} \end{array}$$

$$\left[\begin{array}{l} \left.\begin{array}{l} \text{rango}(C) = 2 \\ \text{rango}(C^*) = 2 \end{array}\right\} \Rightarrow \\ \Rightarrow \begin{vmatrix} -\alpha & +1 & +1 \\ 0 & -1 & +1 \\ -\alpha & \beta & 0 \end{vmatrix} = -\alpha - \alpha + \beta = 0 \Rightarrow \\ \qquad\qquad \Rightarrow \beta = 2\alpha \end{array}\right.$$

Para simplificar los vectores de ponderación, α y β serán enteros naturales y primos entre sí: $\alpha = 1$, $\beta = 2$. Es decir, $\alpha(p_{12}, t_1) = 1$ y $\beta(t_2, p_{12}) = 2$.

A continuación, se estudian condiciones suficientes para que p_i sea un lugar implícito. Procediendo de acuerdo con los métodos explicados (§3.3. o proposición 4.18), se obtiene:

$$I(p_{12}) = I(p_1) + I(p_4).$$

En conclusión, puesto que la RdP es pura y $M_0(p_1) + M_0(p_4) = 1$, el avance ponderado que se ha calculado (θ_1, θ_2) es la unidad:

$$AV(R, M_0; \underbrace{(2\,0\,0)^T}_{\theta_1^T}, \underbrace{(0\,1\,0)^T}_{\theta_2^T}) = 1.$$

EJEMPLO. Determínese el avance ponderado entre las transiciones t_1 y t_2 de la RdP de la figura 4.43.

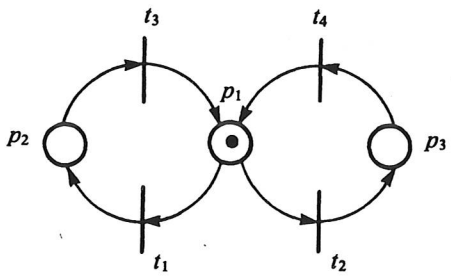


Figura 4.43. $AV(R, M_0; t_1, t_2) = \infty$ y $AV(R, M_0; t_2, t_1) = \infty$.

Procediendo de acuerdo con el método propuesto se tiene:

$$C^* = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 \end{matrix} \\ \left(\begin{array}{cccc} -1 & -1 & +1 & +1 \\ +1 & 0 & -1 & 0 \\ 0 & +1 & 0 & -1 \\ \hline -\alpha & \beta & 0 & 0 \end{array}\right) \end{array} \begin{array}{l} p_1 \\ p_2 \\ p_3 \\ \hline p_{12} \end{array}$$

$$\left[\begin{array}{l} \text{rango}(C) = 2 \\ \text{rango}(C^*) = 2 \\ \begin{vmatrix} -1 & -1 & +1 \\ +1 & 0 & -1 \\ -\alpha & \beta & 0 \end{vmatrix} = \begin{vmatrix} -1 & -1 & +1 \\ +1 & 0 & 0 \\ -\alpha & \beta & 0 \end{vmatrix} = 0 \Rightarrow \\ -\alpha + \beta - \beta = \beta = 0 \Rightarrow \alpha = \beta = 0 \end{array}\right.$$

Es decir, p_{12} no puede ser un lugar implícito a no ser que esté desconectado del resto de la RdP. Si se hace $\beta \neq 0$, para que p_{12} no restrinja las secuencias de disparo, éste debe poseer inicialmente una infinidad de marcas, de donde $AV(\boldsymbol{R}, M_0; t_1, t_2) = \infty$

Si se calcula $AV(\boldsymbol{R}^*, M_0; t_2, t_1)$ se obtiene idéntico resultado. Habida cuenta que la red es viva y estructuralmente limitada, existirán dos o más secuencias repetitivas, de las cuales, al menos una contendrá t_1 y otra contendrá t_2. En este caso, $\sigma_1 = t_1 t_3$ y $\sigma_2 = t_2 t_4$. Éstas se ejecutan debido a la presencia de las *componentes repetitivas elementales* $(1\,0\,1\,0)^T$ y $(0\,1\,0\,1)^T$.

EJEMPLO. LECTORES Y REDACTORES. Se trata de comprobar que la descripción de la figura 4.39 y, por tanto, la de la figura 2.9 cumplen las restricciones ya expuestas con anterioridad:

a) existe exclusión mutua entre redactores,
b) no se puede leer y escribir al mismo tiempo.

Por razones de tipo didáctico, procederemos a verficar independientemente cada una de las restricciones. La exclusión mutua entre redactores se expresa en términos de disparos de transiciones, diciendo que el número de «comienzo de redacción» ejecutados, $\bar{\sigma}(CR_1) + \bar{\sigma}(CR_2)$, menos el número de «fin de redacción» ejecutados, $\bar{\sigma}(FR_1) + \bar{\sigma}(FR_2)$, es menor o igual a la unidad:

$$\bar{\sigma}(CR_1) + \bar{\sigma}(CR_2) - \bar{\sigma}(FR_1) - \bar{\sigma}(FR_2) \leqslant 1 \Rightarrow$$
$$\Rightarrow AV(\boldsymbol{R}^*, M_0; \quad \{CR_1, CR_2\}, \ \{FR_1, FR_2\}) \leqslant 1.$$

Definiendo $p_{T_2T_2} = \{CR_1, CR_2\}$ y $\cdot p_{T_1T_2} = \{FR_1, FR_2\}$ obtendremos que $p_{T_1T_2}$ es un lugar implícito si $M_0(p_{T_1T_2}) = 1$ y, por consiguiente, según el corolario 4.12 se verifica la restricción sobre el avance: los redactores están en exclusión mutua.

La restricción sobre la lectura y escritura simultánea la podemos estudiar fácilmente, al considerar cada lectura con todas las escrituras. Procediendo de esta forma, garantizamos la restricción «*b*» si tenemos:

$$i = 1, 2 \qquad \bar{\sigma}(CL_i) + \bar{\sigma}(CR_1) + \bar{\sigma}(CR_2) - \bar{\sigma}(FL_i) - \bar{\sigma}(FR_1) - \bar{\sigma}(FR_2) \leqslant 1$$

(*Nota*. Puesto que el invariante de disparo asociado al lugar AL_i exige que $\bar{\sigma}(CL_i) - \bar{\sigma}(FL_i) \leqslant 1$, la inecuación anterior comprende a la que se consideró para demostrar la exclusión mutua entre redactores).

Procediendo como hicimos anteriormente para demostrar que los redactores estaban en exclusión mutua, se llega a la conclusión de que $\forall\, i = 1, 2$ el lugar $(p_{T_1T_2})_i$ es idéntico al existente X_i, de donde se puede afirmar que lectores y redactores están en exclusión mutua.

4.8 CERROJOS Y TRAMPAS

Introducimos en este apartado otro grupo de técnicas de análisis estructural. Las técnicas que agrupamos bajo este epígrafe se basan en la detección de ciertos subconjuntos de nudos de la RdP original. La ventaja de esta aproximación es la de permitir el estudio de la vivacidad de una red sin tener que recurrir a la enumeración de los marcados (§4.4). Antes de continuar con la presentación de los métodos basados en cerrojos y trampas, conviene recordar que el problema del estudio de la vivacidad no ha sido completamente resuelto ni por los métodos de reducción ni por los métodos de análisis estructural basado en los conceptos de conservatividad y repetitividad. En efecto, los primeros están limitados por la existencia de redes irreduci-

bles para el conjunto de reglas de reducción presentadas; los segundos permiten el establecimiento de condiciones necesarias, pero no suficientes.

Para poder garantizar la vivacidad de una red sin recurrir a la enumeración, hemos de proceder a restringir las clases de redes que consideramos. De este modo, en lo que concierne a este apartado, nos ceñiremos a las RdP ordinarias. Posteriormente, iremos restringiendo nuestra atención a subclases de RdP ordinarias (§2.3.1). Evidentemente, cuanto más particular sea una subclase, más potentes serán los resultados para su análisis†.

En §4.8.1 presentamos las definiciones y propiedades básicas relacionadas con los conceptos de cerrojo y de trampa. En el apartado 4.8.2 exponemos métodos que permiten determinar sistemáticamente los cerrojos y/o trampas de una red. Su formulación se basa en el álgebra de BOOLE y el álgebra lineal.

4.8.1 Definiciones y propiedades básicas

Sea una RdP $\boldsymbol{R} = \langle P, T, \alpha, \beta \rangle$.

Definición 4.28. Un *cerrojo* es un subconjunto de lugares de una RdP tal que el conjunto de sus transiciones de entrada está contenido en el conjunto de sus transiciones de salida. □

Es decir, todo cerrojo $\Gamma = \{p_i\} \subseteq P$ verifica $\{{}^{\cdot}p_i\} \subseteq \{p_i^{\cdot}\}$, lo cual lo expresaremos como ${}^{\cdot}\Gamma \subseteq \Gamma^{\cdot}$.

Definición 4.29. Una *trampa* es un subconjunto de lugares de una RdP tal que el conjunto de sus transiciones de salida está contenido en el conjunto de sus transiciones de entrada. □

Considerando $\boldsymbol{R}$, $\theta \subseteq P$ es una trampa sii $\theta^{\cdot} \subseteq {}^{\cdot}\theta$.

Como podrá deducirse fácilmente, en las definiciones de cerrojos y trampas, se considera sólo la estructura de la RdP, haciéndose abstracción del marcado inicial. La figura 4.44 presenta sendos ejemplos de cerrojo y de trampa.

Partiendo de estas definiciones se pueden establecer inmediatamente las propiedades siguientes (justifíquelas el lector).

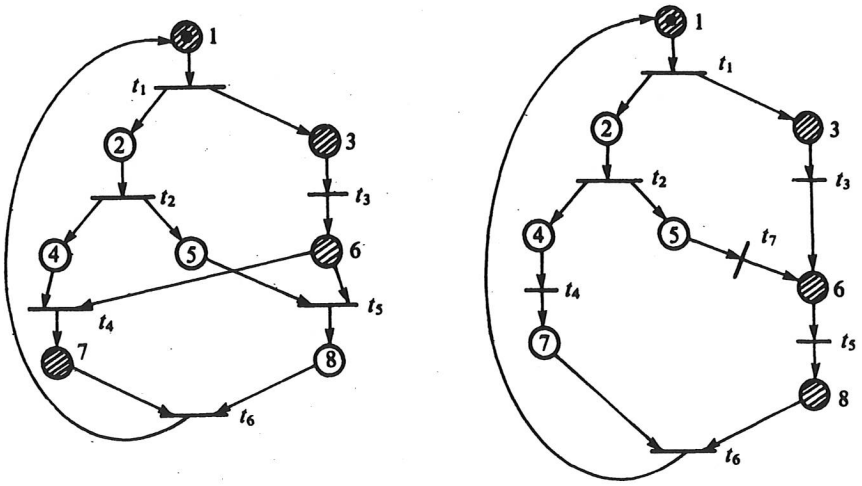
Propiedades básicas de los cerrojos y de las trampas (proposición 4.20).

a) Un cerrojo inicialmente desmarcado o que se desmarca debido a una cierta secuencia de disparos, permanecerá desmarcado para cualquier evolución posible de la red. Es decir, al menos todas sus transiciones de salida y, por lo tanto, las de entrada no son vivas, y, por consiguiente, la RdP no es viva.

b) Una trampa, marcada para M_0, permanecerá siempre marcada; es decir, independientemente de la evolución de la RdP. □

---

† Esta afirmación viene a decir algo que es bien conocido en todas las ramas del saber: cuanto más simple es un problema más fácilmente se puede estudiar.



(a)
$\Gamma = \{p_1, p_3, p_6, p_7\}$
${}^{\bullet}\Gamma = \{6, 1, 3, 4\}$
$\Gamma^{\bullet} = \{1, 3, 4, 5, 6\}$
$\Rightarrow {}^{\bullet}\Gamma \subset \Gamma^{\bullet}$

(b)
$\theta = \{p_1, p_3, p_6, p_8\}$
${}^{\bullet}\theta = \{6, 1, 3, 7, 5\}$
$\theta^{\bullet} = \{1, 3, 5, 6\}$
$\Rightarrow \theta^{\bullet} \subset {}^{\bullet}\theta$

Figura 4.44. Ejemplos de cerrojo y de trampa.

Si consideramos la RdP de la figura 4.44*a*, se observará que $\Gamma = \{p_1, p_3, p_6, p_7\}$ es un cerrojo que se vacía debido a la secuencia de disparos $\sigma = t_1 t_2 t_3 t_5$. Por consiguiente, la RdP es no viva. Por otro lado, si consideramos la RdP de la figura 4.44*b*, se observará que $\theta = \{p_1, p_3, p_6, p_8\}$ no puede perder todas sus marcas, independientemente de la evolución del marcado.

Para que una RdP sea viva es necesario que los cerrojos no se desmarquen. Esto se puede garantizar, gracias a la estructura de la RdP, si todo cerrojo contienen trampas marcadas. Para formalizar esta idea definiremos una nueva propiedad estructural de las RdP.

Definición 4.30. La red marcada $\langle \boldsymbol{R}, M_0 \rangle$ posee la *propiedad cerrojo-trampa*, propiedad CT, si cada cerrojo contiene una trampa marcada para M_0. □

Caracterización de la vivacidad-parcial (proposición 4.21). Si $\langle \boldsymbol{R}, M_0 \rangle$ posee la propiedad CT, es parcialmente viva (no se bloquea). □

DEMOSTRACIÓN. La propiedad CT garantiza que cada cerrojo de $\boldsymbol{R}$ esté marcado $\forall M \in M(\boldsymbol{R}, M_0)$. Supóngase que $\boldsymbol{R}$ se bloquea con el marcado M'. El subconjunto de lugares desmarcados en M' es un cerrojo, puesto que se trata de un subconjunto de lu-

gares que no puede ser marcado después de estar desmarcado. Al existir un cerrojo desmarcado, no se cumple la propiedad CT (contradicción). □

La RdP de la figura 4.45 pone de manifiesto que no se puede asegurar la vivacidad total, pese a que se verifique la propiedad CT. Para obtener resultados más potentes, es necesario restringir la clase de RdP ordinarias que se considera. De este modo se pueden establecer muchos resultados de interés práctico. En este texto nos limitaremos a presentar uno de los más clásicos, conocido como teorema de COMMONER.

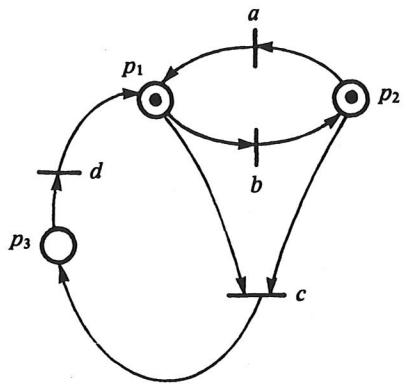


Figura 4.45. RdP marcada parcialmente-viva.

RLE y vivacidad (proposición 4.22). Una RdP libre elección $\langle R, M_0 \rangle$ es viva sii posee la propiedad CT. □

La demostración de este resultado escapa a los objetivos de este texto. Puede encontrarse, por ejemplo, en [HACK 72] o en [JANT 79].

Grafos marcados y vivacidad (corolario 4.13). Un *grafo marcado fuertemente conexo* es vivo sii cada circuito contiene, al menos, una marca en el marcado inicial. □

En efecto, cualquier circuito de un GM fuertemente conexo es simultáneamente un cerrojo y una trampa. De lo anterior, también se puede concluir que el número de marcas que contiene un circuito es invariante. Por consiguiente, si un GM fuertemente conexo es binario para M_0, será también binario para cualquier marcado alcanzable a partir de M_0.

El razonamiento anterior permite anunciar que un grafo marcado fuertemente conexo es *conforme* sii cada circuito contiene inicialmente una marca.

EJEMPLO. Sea la RdP de la figura 4.46. Los circuitos elementales son:

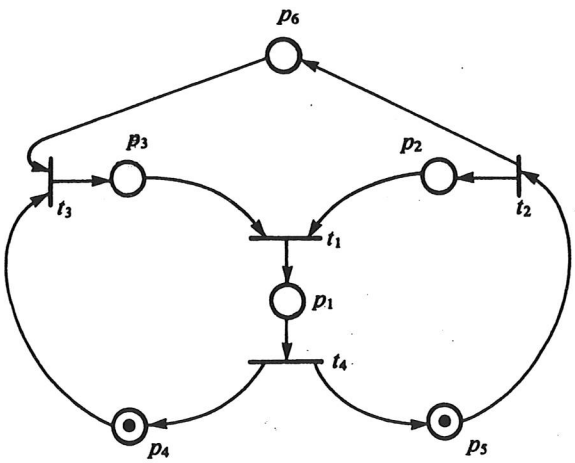


Figura 4.46. Grafo Marcado fuertemente conexo.

$\{p_2, p_1, p_5\} \rightarrow 1$ marca en $M_0{:}p_5$
$\{p_3, p_1, p_4\} \rightarrow 1$ marca en $M_0{:}p_4$
$\{p_3, p_1, p_5, p_6\} \rightarrow 1$ marca en $M_0{:}p_5$

Luego la RdP de la figura 4.46 es viva y binaria.

4.8.2 Obtención de los cerrojos y de las trampas de una RdP

4.8.2.1 *Obtención de los cerrojos*

Asociemos a cada lugar p_i de la RdP una variable γ_i que represente «la pertenencia de p_i a un cerrojo Γ»:

$$\gamma_i = 1 \quad \text{sii} \quad p_i \in \Gamma.$$

De acuerdo con la definición 4.28, el lugar p_i pertenece a un cerrojo Γ si se cumple que para cada una de sus transiciones de entrada, $\forall t_k \in {}^{\cdot}p_i$, al menos un lugar de entrada a cada una de ellas pertenece también al cerrojo, $\exists p_j \in {}^{\cdot}t_k$. La condición anterior se puede expresar lógicamente mediante la implicación $\gamma_i \Rightarrow u_i$ (es decir, $\gamma_i = 1$ es condición suficiente para $u_i = 1$), donde:

$$u_i = \bigwedge_{t_k \in {}^{\cdot}p_i} \left[\bigvee_{p_j \in {}^{\cdot}t_k} \gamma_j \right].$$

Si la condición anterior se cumple simultáneamente para un conjunto de lugares, éstos definen un cerrojo. Por consiguiente, todo cerrojo se obtiene como solución del conjunto de las n implicaciones siguientes (una por lugar de la red):

$$\forall i = 1, \ldots, n \qquad [\gamma_i \Rightarrow u_i] \equiv \bigwedge_{i=1,n} (\bar{\gamma}_i \vee u_i) = 1.$$

EJEMPLO. Sea la RdP de la figura 4.44*a*.

$$\bigwedge_{i=1,8} (\bar{\gamma}_i \vee u_i) = (\bar{\gamma}_1 \vee \gamma_7 \vee \gamma_8) \wedge (\bar{\gamma}_2 \vee \gamma_1) \wedge (\bar{\gamma}_3 \vee \gamma_1) \wedge (\bar{\gamma}_4 \vee \gamma_2) \wedge (\bar{\gamma}_5 \vee \gamma_2) \wedge (\bar{\gamma}_6 \vee \gamma_3) \wedge (\bar{\gamma}_7 \vee \gamma_4 \vee \gamma_6) \wedge (\bar{\gamma}_8 \vee \gamma_5 \vee \gamma_6) = 1.$$

Desarrollando en unión de intersecciones se obtiene (se omite la «∧»):

$$\underbrace{\gamma_1\gamma_3\gamma_6\gamma_7\bar{\gamma}_4\bar{\gamma}_5}_{a} \vee \underbrace{\gamma_1\gamma_3\gamma_6\gamma_8\bar{\gamma}_4\bar{\gamma}_5}_{b} \vee \underbrace{\bar{\gamma}_1\bar{\gamma}_2\bar{\gamma}_3\bar{\gamma}_4\bar{\gamma}_5\bar{\gamma}_6\bar{\gamma}_7\bar{\gamma}_8}_{c} \vee$$

$$\vee \underbrace{\gamma_1\gamma_2\gamma_4\gamma_7\bar{\gamma}_6\bar{\gamma}_8}_{d} \vee \underbrace{\gamma_1\gamma_2\gamma_4\gamma_5\gamma_7\bar{\gamma}_6}_{e} \vee \underbrace{\gamma_1\gamma_2\gamma_3\gamma_4\gamma_7\bar{\gamma}_8}_{f} \vee$$

$$\vee \underbrace{\gamma_1\gamma_2\gamma_3\gamma_4\gamma_5\gamma_7}_{g} \vee \underbrace{\gamma_1\gamma_2\gamma_3\gamma_6\gamma_7}_{h} \vee \underbrace{\gamma_1\gamma_2\gamma_5\gamma_8\bar{\gamma}_6\bar{\gamma}_7}_{i} \vee$$

$$\vee \underbrace{\gamma_1\gamma_2\gamma_4\gamma_5\gamma_8\bar{\gamma}_6}_{j} \vee \underbrace{\gamma_1\gamma_2\gamma_3\gamma_5\gamma_8}_{k} \vee \underbrace{\gamma_1\gamma_2\gamma_3\gamma_4\gamma_5\gamma_8}_{l} \vee$$

$$\vee \underbrace{\gamma_1\gamma_2\gamma_3\gamma_6\gamma_8}_{m} = 1.$$

Considerando el término «*a*» (por ejemplo), todos los conjuntos de lugares comprendidos entre $\{p_1, p_3, p_6, p_7\}$ y $\{p_1, p_2, p_3, p_6, p_7, p_8\}$ son cerrojos. Es decir, según el término «*a*», son cerrojos los conjuntos de lugares siguientes:

$$\{p_1, p_3, p_6, p_7\} \qquad \{p_1, p_2, p_3, p_6, p_7\}$$
$$\{p_1, p_3, p_6, p_7, p_8\} \qquad \{p_1, p_2, p_3, p_6, p_7, p_8\}.$$

Procediendo de la forma indicada, tras la eliminación de repeticiones, se obtienen todos los cerrojos de la RdP. Los cerrojos no incluidos en otros se denominan *cerrojos mínimos*. Éstos se obtienen directamente como un subconjunto de los cerrojos menores definidos por cada uno de los términos del desarrollo en unión de intersecciones. Reconsiderando el ejemplo anterior se tiene:

$$a = \{p_1, p_3, p_6, p_7\} \qquad b = \{p_1, p_3, p_6, p_8\} \qquad c = \emptyset$$
$$d = \{p_1, p_2, p_4, p_7\} \qquad e = \{p_1, p_2, p_4, p_5, p_7\} \qquad f = \{p_1, p_2, p_3, p_4, p_7\}$$
$$g = \{p_1, p_2, p_3, p_4, p_5, p_7\} \qquad h = \{p_1, p_2, p_3, p_6, p_7\} \qquad i = \{p_1, p_2, p_5, p_8\}$$
$$j = \{p_1, p_2, p_4, p_5, p_8\} \qquad k = \{p_1, p_2, p_3, p_5, p_8\} \qquad l = \{p_1, p_2, p_3, p_4, p_5 p_8\}$$
$$m = \{p_1, p_2, p_3, p_6, p_8\}.$$

Ahora bien, $a \subset h$ (el conjunto *a* está incluido en *h*), $b \subset m$, $d \subset e$, $d \subset f$, $d \subset g$, $i \subset j$, $i \subset k$ e $i \subset l$, luego los cerrojos mínimos del ejemplo son $\{a, b, d, i\}$.

Observación. Como se puede comprobar, a veces un cerrojo no mínimo no puede expresarse como la unión de cerrojos mínimos. Ejemplos: $e = d \cup p_5$, $f = d \cup p_3$, etc.

El método de búsqueda de cerrojos basado en el algebra de BOOLE tiene la ventaja de su fácil comprensión pero computacionalmente no suele ser muy eficiente. A continuación lo traducimos en un problema algebraico-lineal. Sustituyendo las operaciones de intersección y de unión por las de multiplicar y sumar, respectivamente, la implicación básica $\gamma_i \Rightarrow u_i$ se puede reescribir como sigue:

$$[\gamma_i = 1] \Rightarrow \left[\prod_{t_k \in \cdot p_i} \left(\sum_{p_j \in \cdot t_k} \gamma_j\right) \geqslant 1\right],$$

o, lo que es lo mismo,

$$\forall t_k \in \cdot p_i : [\gamma_i = 1] \Rightarrow \left[\sum_{p_j \in \cdot t_k} \gamma_j \geqslant 1\right].$$

Por último, puesto que $\gamma_i \in \{0, 1\}$, la condición de pertenencia de p_i a un cerrojo se puede expresar de la siguiente forma:

$$\forall t_k \in \cdot p_i : \sum_{p_j \in \cdot t_k} \gamma_j - \gamma_i \geqslant 0.$$

Es decir, asociado a cada lugar se obtiene un sistema de inecuaciones lineales que debe ser satisfecho.

EJEMPLO. Inecuaciones asociadas a p_6 en la RdP de la figura:

(1) 4.44*a*: $\cdot p_6 = \{t_3\}$. El lugar p_6 con respecto a t_3 genera la inecuación: $\gamma_3 - \gamma_6 \geqslant 0$.
(2) 4.44*b*: $\cdot p_6 = \{t_3, t_7\}$. El lugar p_6 con respecto a t_3 genera la inecuación: $\gamma_3 - \gamma_6 \geqslant 0$. El lugar p_6 con respecto a t_7 genera la inecuación: $\gamma_5 - \gamma_6 \geqslant 0$.

Para calcular los cerrojos de una red de Petri basta con resolver el sistema de inecuaciones que se obtiene al escribir todas las correspondientes a cada lugar de la red.

EJEMPLO. Sea la RdP de la figura 4.44*a*. El sistema global de inecuaciones es:

$$\left.\begin{array}{lll}
(1)\ p_1\colon & \cdot p_1 = \{t_6\} & \Rightarrow\ \gamma_7 + \gamma_8 - \gamma_1 \geqslant 0 \\
(2)\ p_2\colon & \cdot p_2 = \{t_1\} & \Rightarrow\ \gamma_1 - \gamma_2 \geqslant 0 \\
(3)\ p_3\colon & \cdot p_3 = \{t_1\} & \Rightarrow\ \gamma_1 - \gamma_3 \geqslant 0 \\
(4)\ p_1\colon & \cdot p_4 = \{t_2\} & \Rightarrow\ \gamma_2 - \gamma_4 \geqslant 0 \\
(5)\ p_5\colon & \cdot p_5 = \{t_2\} & \Rightarrow\ \gamma_2 - \gamma_5 \geqslant 0 \\
(6)\ p_6\colon & \cdot p_6 = \{t_3\} & \Rightarrow\ \gamma_3 - \gamma_6 \geqslant 0 \\
(7)\ p_7\colon & \cdot p_7 = \{t_4\} & \Rightarrow\ \gamma_4 + \gamma_6 - \gamma_7 \geqslant 0 \\
(8)\ p_8\colon & \cdot p_8 = \{t_5\} & \Rightarrow\ \gamma_5 + \gamma_6 - \gamma_8 \geqslant 0
\end{array}\right\} \Gamma^T \cdot \mathcal{C} \geqslant 0$$

Cualquier solución *binaria* ($\gamma_i \in \{0, 1\}$) del anterior sistema es un cerrojo. Ahora bien, dada la peculiar forma de las inecuaciones, resulta inmediato comprobar que cualquier solución no negativa ($\gamma_j^* \in \mathbb{N}$) describe una solución binaria que consiste en tomar $\gamma_i = 1$ si y sólo si $\gamma_j^* > 0$. De acuerdo con esta observación, la resolución de los sistemas de inecuaciones se puede llevar a cabo utilizando un método como el sugerido en el anexo 5. La obtención de los cerrojos mínimos se realizará buscando directamente entre todos los cerrojos aquellos que no están incluidos en otros.

EJERCICIO. Compruébese que si un lugar p es simultáneamente lugar de entrada y de salida de una transición t, es inútil la escritura del término booleano (o inecuación lineal) asociado a t con respecto a p.

4.8.2.2 *Obtención de las trampas*

Una trampa es, por definición, toda solución del conjunto de las n implicaciones siguientes (compruébese):

$$\forall i = 1, \ldots, n \quad [\gamma_i \Rightarrow v_i], \text{ donde } v_i = \bigwedge_{t_k \in p_i} \left(\bigvee_{p_j \in t_k} \gamma_j \right).$$

El tratamiento para la obtención de las *trampas mínimas* (no incluidas en otras) es similar al de la obtención de los cerrojos, por lo que no se presenta.

EJERCICIO. Obténganse las trampas mínimas (que no contienen a otras) de la RdP de la figura 4.44*b*.

EJERCICIO. Desarróllese un método algebraico-lineal para obtener las trampas de una RdP.

4.8.2.3 *Obtención de los conjuntos de lugares que son simultáneamente, cerrojo y trampa*

Admitiendo los resultados §4.8.2.1 y §4.8.2.2, todo conjunto de lugares es cerrojo y trampa simultáneamente si satisface las n implicaciones siguientes:

$$\forall i = 1, \ldots, n \quad [\gamma_i \Rightarrow (u_i \wedge v_i)].$$

EJERCICIO. Demuéstrese que los conjuntos de lugares de la red de la figura 4.44*a*:

$$\{p_1, p_2, p_4, p_7\}, \{p_1, p_2, p_5, p_8\} \text{ y } \{p_1, p_3, p_6, p_7, p_8\}$$

son cerrojos y trampas simultáneamente y no contienen a ningún otro.

4.9 CONCLUSIÓN

Se ha realizado una aproximación a las técnicas de análisis de RdP marcadas y autónomas; es decir, desprovistas de interpretación.

Desde un punto de vista metodológico, es importante tener en cuenta la aplicación de las técnicas de reducción, antes de proceder a un análisis por enumeración o estructural. Con ello, se suele disminuir sustancialmente la complejidad de la ejecución de los algoritmos de análisis.

El carácter autónomo de las RdP analizadas, restringe la utilidad de algunos de los resultados presentados, dado que «las RdP, que describen sistemas, son evidentemente no-autónomas». A pesar de ello, el análisis de las RdP autónomas es muy importante puesto que:

1) Existen subclases de RdP provistas de interpretación, para las que los resultados del análisis de la RdP autónoma mantienen su vigencia al asociársele una interpretación (se considerará en el próximo capítulo).
2) Permite evidenciar estructuras de RdP «susceptibles de introducir errores» y, además, simplificar la verificación semántica de la descripción (no abordada en este texto).

3) Aunque el comportamiento de la RdP interpretada sea correcto, es conveniente eliminar las incoherencias estructurales de la red, con el objeto de facilitar la legibilidad y la modificabilidad de la descripción.

Como puede constatarse, esta última reflexión está en relación con el tipo de argumentos esgrimidos en favor de la «programación estructurada» (véase por ejemplo [DIKJ 76], [TABO 75], [LING 79] [WIRT 76]).

En resumen, se ha estudiado un conjunto de técnicas que permiten alcanzar un cierto grado de confianza en el modelo.

En el próximo capítulo se analizará el impacto, que sobre el análisis de las RdP, tiene la introducción del tiempo o/y de la interpretación de las mismas. Posteriormente, se sugerirán metodologías de modelación.

EJERCICIOS

4.1 ¿Puede determinarse un marcado inicial que haga que la RdP de la figura 4.31*a* sea viva? ¿Por qué?
Repítase el mismo estudio con la RdP de la figura 4.31*b*.

4.2 Aplicando el método de enumeración de los marcados, estúdiese la vivacidad, la ciclicidad y la limitación de las RdP ilustradas por la figura 4.44.

4.3 Demuéstrese a partir del concepto de conservatividad que la red de la figura 4.3 es estructuralmente no limitada.

4.4 ¿Es completamente reducible la red de la figura 4.5? ¿Es viva y binaria? ¿Cuáles son sus componentes conservativas elementales?

4.5 Supóngase que la red de la figura 4.2*a* es una red con capacidad (definición 2.17) uniforme en todos sus lugares e igual a 3. Transfórmese en una red ordinaria y simplificada. ¿Sería lógico que se obtuviese como resultado final la misma red pero considerada como ordinaria? ¿Por qué?

4.6 Determínense los cerrojos y las trampas mínimos de las redes ilustradas por las figuras 4.10 y 4.11. ¿Se puede deducir directamente la vivacidad?

4.7 Redúzcase al máximo posible, de acuerdo con las reglas presentadas en §4.5, la RdP de la figura E.2.1.

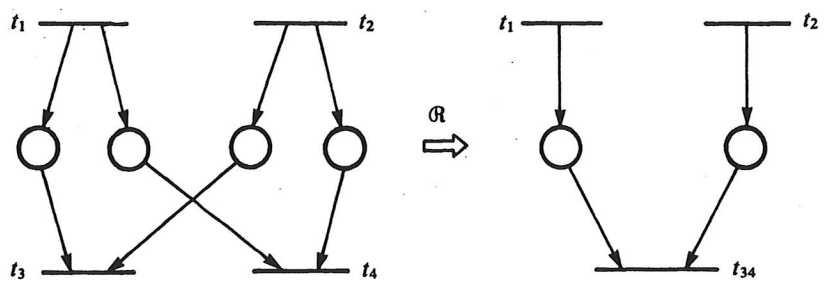


Figura E.4.1 Doble semáforo que genera una espera mutua.

4.8 Considérese la regla de reducción ilustrada en la figura E.4.1. ¿Preserva la vivacidad, la limitación y la ciclicidad?

4.9 Transfórmese la negación de la aserción sobre el marcado $\neg A(M) = A \cdot M \geqslant B$ (A y B son matrices) en una aserción sobre las secuencias disparables, $\bar{\sigma}$. ¿Es posible establecer siempre la transformación inversa: aserción sobre $\bar{\sigma}$ en aserción sobre M?

5

Validación funcional de una descripción (III): redes de Petri no autónomas. Impacto sobre la descripción

5.1 INTRODUCCIÓN

Los resultados que se refieren al análisis de RdP autónomas no se pueden utilizar directamente cuando a la evolución de la RdP se asocia un tiempo, u otra interpretación que permita modelar un sistema. En este capítulo se estudiarán, en un tono deliberadamente informal, algunas de estas cuestiones y se presentará su impacto sobre la descripción de sistemas. Ésto nos conducirá a ideas conocidas en el marco de la programación de computadores, como son, por ejemplo, la estructuración y la descripción por refinamientos sucesivos.

5.2 ANÁLISIS DE LAS RdP TEMPORIZADAS

Una RdP temporizada (RdPT, §2.5.2.6) es un par $\langle \boldsymbol{R}, Z \rangle$ tal que $\boldsymbol{R}$ es una RdP y Z es una función que asigna un número real no negativo, z_i, a cada transición de la RdP. $z_i = Z(t_i)$ recibe el nombre de tiempo de disparo de la transición t_i.

Una vez recordada una de las definiciones de RdPT, vamos a estudiar las implicaciones que sobre el análisis tiene la temporización de una RdP.

5.2.1 Estudio de la limitación

Una RdPT es limitada si la RdP correspondiente también lo es. Sin embargo, ocurre a veces que determinadas sincronizaciones no son necesarias si se conocen los tiempos de disparo. Considérese, por ejemplo, la conocida relación productor-consumidor, según la figura 5.1.

La RdP de la figura 5.1 no es limitada. Ahora bien, si el ciclo de producción dura más que el de consumo, la RdPT sí es limitada; es decir:

si $z_1 + z_2 + z_3 \geqslant z_4 + z_5 + z_6$, entonces la RdPT es limitada.

En conclusión:

RdPT y limitación (proposición 5.1). Para que una RdPT sea limitada (o binaria), no es necesario que la RdP lo sea. Si una RdP es limitada (o binaria), cualquier RdPT construída sobre ella será limitada (o binaria). □

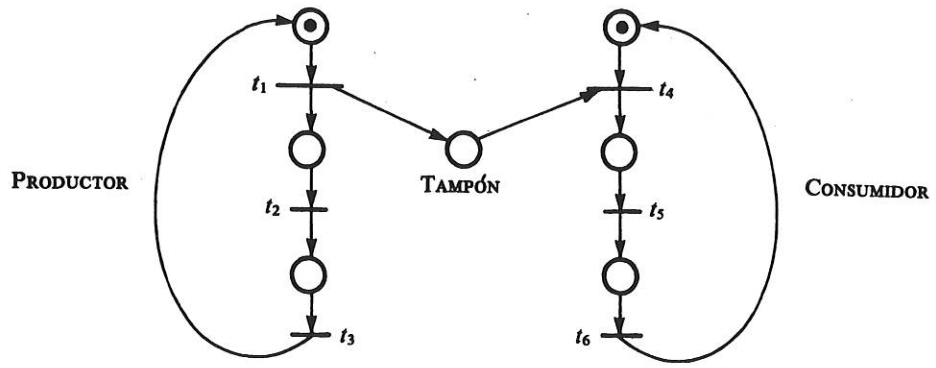


Figura 5.1 Relación productor-consumidor (la RdP autónoma no es limitada).

5.2.2 Estudio de la vivacidad

El estudio de la vivacidad conduce a problemas delicados, puesto que la vivacidad de una RdPT no podrá ser inferida, en modo alguno, partiendo de la vivacidad o no-vivacidad de la RdP autónoma.

Proposición 5.2. Una RdPT marcada puede ser viva aunque la RdP no lo sea. □

Su justificación es inmediata al comprobar la evolución de la RdPT de la figura 5.2*a*. Obsérvese que, debido a las restricciones temporales, el grafo de marcados de la RdPT es un *subgrafo* del grafo de marcados de la RdP. En este caso, las mencionadas restricciones impiden el disparo de las secuencias $\sigma_1' = t_3t_4t_3t_4$ y $\sigma_2' = t_2t_1t_2t_1$, entre otras. A partir de σ_1' o σ_2' es imposible disparar otra transición puesto que los marcados alcanzables serían C^2FH y B^2FH, respectivamente.

Proposición 5.3. Una RdPT marcada puede ser no viva a pesar de que la RdP sea viva. □

Una justificación de esta última proposición la presentamos al estudiar un ejemplo clásico de sincronización: el problema de *los filósofos y los «spaghetti»* [DIJK 71]. Lo particularizaremos para cuatro filósofos.

Enunciado. Cuatro filósofos se reunen para elucubrar y cenar. La cena se compone de un plato único a base de «spaghetti». Según un protocolo establecido (lejos de lo habitualmente considerado como norma de urbanidad), los «spaghetti» deben comerse con dos tenedores. Como quiera que la mesa ha sido puesta únicamente con un tenedor por cubierto, al iniciar la comida se les plantea un problema práctico. Después de reflexionar, los filósofos adoptan el siguiente ritual:

1) Cada filósofo se sentará delante de un cubierto.
2) Para comer, un filósofo podrá utilizar su tenedor y el del comensal que se encuentre a su derecha. Por consiguiente, nunca podrán comer simultáneamente dos filósofos que utilicen cubiertos contiguos.

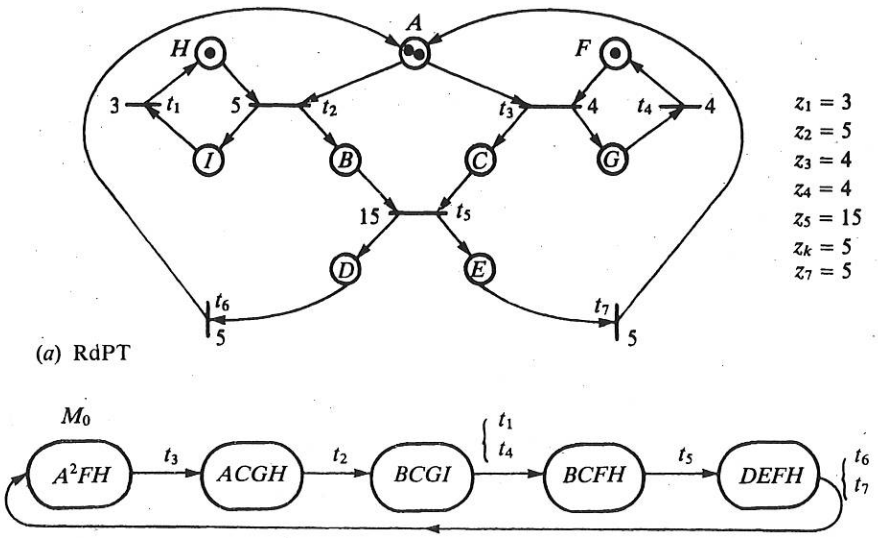


Figura 5.2. RdPT viva (la RdP autónoma no lo es) y su grafo de marcados (subgrafo del grafo de marcados de la RdP).

3) Cada uno de los cuatro filósofos se puede encontrar en una de las tres situaciones (estados) siguientes:
   - *comiendo.*
   - *esperando* para comer, por no disponer de los dos tenedores que necesita;
   - *pensando* y, por cortesía, no utiliza mientras tanto ningún tenedor.
4) Inicialmente todos los filósofos están pensando.

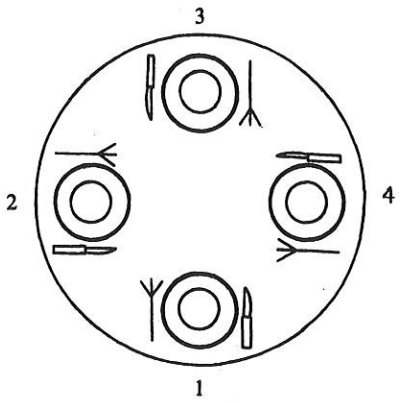
De acuerdo con lo enunciado, el comportamiento de cada uno de los comensales se reduce a una sucesión de intervalos de reflexión y de ingestión.

Al margen del problema que plantee el hecho de que dos filósofos intenten coger un mismo tenedor (*conflicto*), la RdP de la figura 5.3*b* es viva y binaria. (Compruébese aplicando las reglas de reducción, §4.5.4.2, figura 4.26.) La mencionada RdP representa una conducta de los filósofos (usuarios) que se basa en la adquisición simultánea de ambos tenedores (recursos).

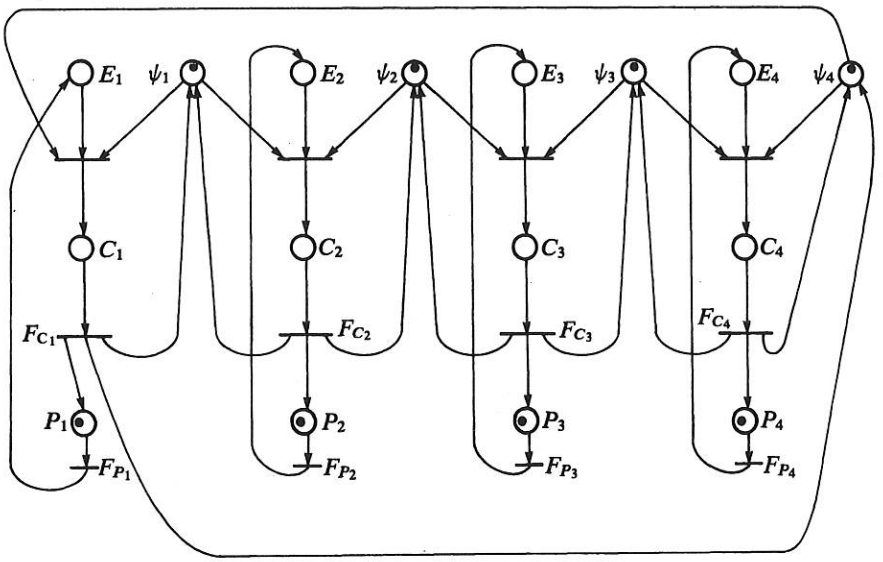
Supóngase, ahora, que la transición etiquetada F_{P_1} es disparada en el instante inicial, y que la transición F_{P_3} es disparada una unidad y media de tiempo más tarde. Si la duración de cada intervalo de comida es fija (2 unidades) y la duración de reflexión también es fija (1 unidad) los filósofos 1 y 3 pueden *monopolizar* los tenedores y, por consiguiente, los filósofos 2 y 4 «no prueban bocado». En estas condiciones, la RdPT de la figura 5.3 es no viva porque existe un bloqueo parcial.

El anterior ejemplo justifica el enunciado de la proposición 5.3.

Resumiendo los enunciados de las proposiciones 5.2 y 5.3 en uno sólo, se puede afirmar que *para que una RdPT sea viva no es ni necesario ni suficiente que la RdP marcada lo sea.*



(*a*) Disposiciones de los comensales.



(*b*) Una descripción con RdP.

Figura 5.3. Problema de los filósofos y los spaghetti. RdP viva y binaria. (*Nota*: E = espera; C = come; P = piensa; F = fin de actividad; ψ = tenedor.)

5.2.3 Cuestiones adicionales sobre la vivacidad

Según lo considerado anteriormente, conviene definir una nueva propiedad que caracterice la situación de presencia o ausencia de posibles monopolios en una RdPT.

Definición 5.1. Una RdP marcada y viva es *libre de monopolio* cuando, independientemente de las duraciones de disparo que se asocien a las transiciones, ésta es viva. □

Como se ha visto en el párrafo anterior (§5.2.2), la propiedad de vivacidad y de ausencia de monopolio no coinciden para las RdP en general. A continuación se considerará una subclase de RdP para la cual la vivacidad implica la ausencia de monopolios.

Monopolios y Subclases de RdP (proposición 5.4). Toda RdP simple y viva para M_0 es libre de monopolio. □

DEMOSTRACIÓN. Para justificarla, se observará cuál es el origen del monopolio. El monopolio aparece en RdP vivas cuando se cumplen las condiciones siguientes:

1) existe una configuración como la ilustrada en la figura 5.4.
2) la temporización de la RdP hace que una marca llegada a p_1 (respec. p_2) desaparezca antes de que p_2 (respec. p_1) sea marcado.

Esto supone que p_1 y p_2 sean selecciones, de donde se concluye que la RdP no puede ser simple, puesto que t_i es un nudo Y (${}^{\bullet}t_i = \{p_1, p_2\}$). □

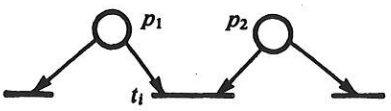


Figura 5.4. t_i puede ser viva en la RdP autónoma pero no en la RdPT.

La RdP de la figura 5.5 representa una conducta diferente de nuestros cuatro filósofos. Habida cuenta de que la RdP es viva y simple, con esta nueva conducta ninguno de ellos puede llegar a un estado de inanición.

EJERCICIO.

a) Determínese la conducta descrita por la RdP de la figura 5.5.
b) Compruébese que la RdP de la figura 5.5 es conforme.

En conclusión, para asegurar la ausencia de bloqueos (parciales o totales) en una RdPT hay que verificar la vivacidad y la ausencia de monopolio. Si la RdPT estuviese definida a partir de una RdP simple, bastaría con verificar la vivacidad para garantizar la ausencia de monopolios.

5.3 ANÁLISIS DE LAS RdP INTERPRETADAS

Los resultados que se presentan son esencialmente análogos a los presentados anteriormente para las RdPT.

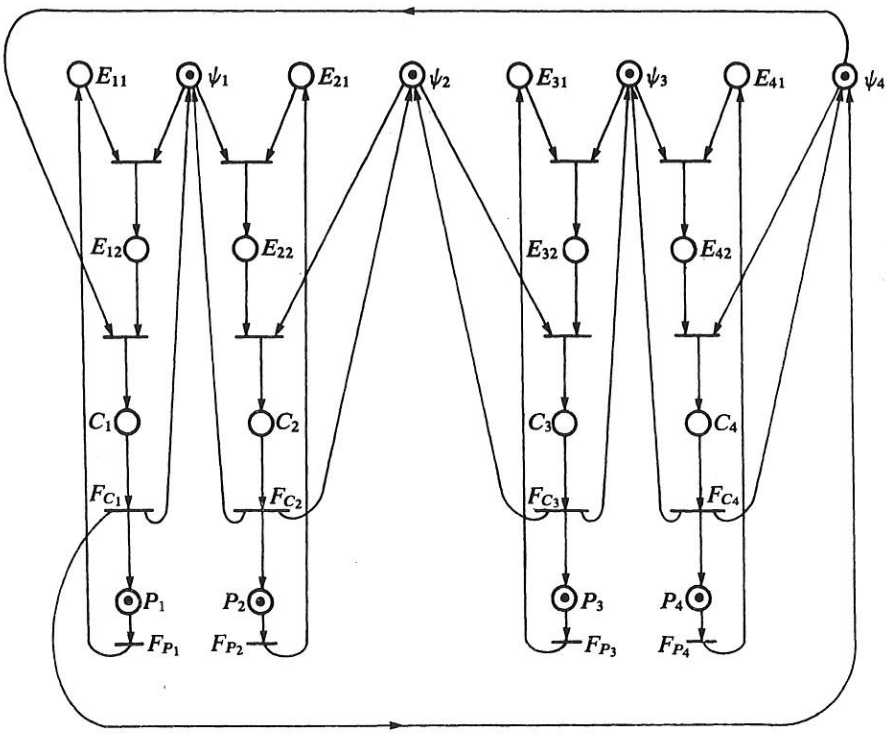


Figura 5.5. RdP marcada conforme y simple implica RdPT libre de monopolio.

5.3.1 Estudio de la limitación

Proposición 5.5. Si una RdP marcada es limitada, cualquier RdP interpretada construída sobre ella lo es. Una RdP no limitada puede ser limitada al asociársele una interpretación (Figura 5.6). □

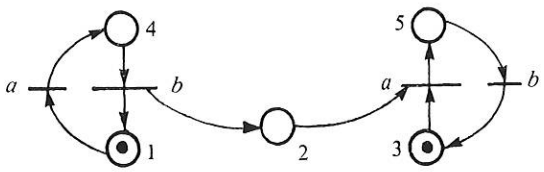


Figura 5.6. La RdP autónoma es no-limitada mientras que la RdP interpretada si es limitada.

Observando detenidamente la RdP interpretada de la figura 5.6, se deduce que en la construcción del grafo de marcados (§4.4) no es aplicable el criterio de parada (proposición 4.1). En efecto, se recordará que cuando existe una secuencia de dispa-

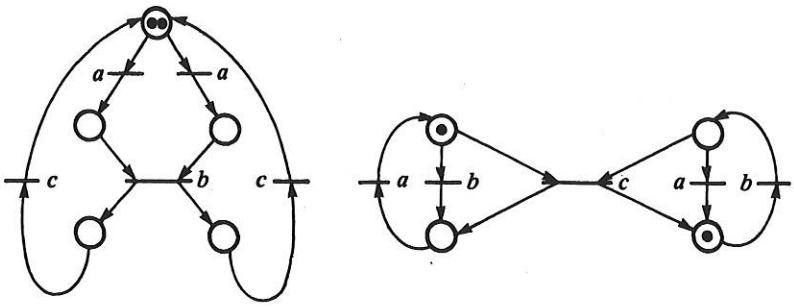
ros σ que lleva desde M_i a M_j, $M_i \xrightarrow{\sigma} M_j$, si $M_i \not\leqslant M_j$, se puede concluir para una red autónoma diciendo que es no limitada. Este criterio no es aplicable a las RdP interpretadas, puesto que en la figura 5.6 se puede alcanzar el marcado $M_2^T = (1\,1\,1\,0\,0)$ a partir de $M_0^T = (1\,0\,1\,0\,0)$, siendo la red limitada.

Esta última observación es válida también para las RdPT.

5.3.2 Estudio de la vivacidad

Vivacidad y RdP interpretadas (proposición 5.6). La vivacidad de una RdP marcada no es necesaria ni suficiente para que la RdP interpretada construida sobre la anterior sea viva. □

Su justificación puede establecerse a partir de las redes ilustradas en la figura 5.7.



(*a*) RdP interpretada viva (la red autónoma no lo es)

(*b*) RdP interpretada no viva (la red autónoma si lo es)

Figura 5.7. Vivacidad y redes interpretadas.

Como resultado análogo a la proposición 5.4 se puede enunciar lo siguiente:

Proposición 5.7. Si una RdP simple es viva para un marcado M_0, toda RdP interpretada construida a partir de la red es viva para M_0, si todos los eventos asociados a las transiciones pueden producirse. □

5.3.3 RdP no autónoma: marcados alcanzables y secuencias disparables

Después de haber realizado una presentación informal de los resultados básicos que establecen las conexiones entre el análisis de las RdP autónomas y el análisis de las RdP no-autónomas, es conveniente reflexionar globalmente sobre el origen de las diferencias aparecidas.

Si el lector determina el conjunto de marcados alcanzables de cualquiera de las redes presentadas en este capítulo, observará que éste se reduce a un *subconjunto* de los marcados alcanzables de la correspondiente red autónoma. Esto puede com-

prenderse fácilmente, puesto que la temporización y la interpretación asociada a una red introducen *restricciones* adicionales sobre las secuencias de disparo posibles en la red autónoma. Es decir *el grafo de marcados de la* RdPT *o* RdPI *es un subgrafo del grafo de marcados de la red autónoma.*

Habida cuenta de estas observaciones, se deducen las tres propiedades siguientes, las cuales constituyen un resumen de los resultados presentados a lo largo del capítulo:

1) La limitación de la RdP autónoma es suficiente, pero no necesaria, para la limitación de la RdP no autónoma. En efecto, si el subconjunto de marcados alcanzables en la red no autónoma es finito, ésta será limitada aunque la red autónoma sea no limitada.
2) Una RdP no autónoma puede ser limitada a pesar de que exista una evolución desde un marcado M_i a otro marcado superior M_j. En efecto, si $M_j \geqslant M_i$ y σ es la secuencia que provoca la evolución, $M_i \xrightarrow{\sigma} M_j$, la red no autónoma podrá ser limitada si la temporización o la interpretación impiden la aplicación de σ un número no acotable de veces.
3) La vivacidad de la RdP autónoma no es necesaria ni suficiente para la vivacidad de la RdP no autónoma. Esta propiedad se comprende también con facilidad, puesto que el conjunto de secuencias disparables en la red no autónoma es un subconjunto de las secuencias disparables en la red autónoma, de donde se deduce que la vivacidad de la red autónoma no es suficiente para la vivacidad de la red no autónoma. Por otro lado, el subconjunto de secuencias de disparo de la red no autónoma puede conducir a que ésta sea viva, aunque la red autónoma no lo fuera.

Además de los resultados anteriores, puede deducirse que la *exclusión mutua* del marcado de dos lugares de una red autónoma es condición suficiente, pero no necesaria, para la exclusión mutua del marcado de los mismos en la red no autónoma. En efecto, esta propiedad es consecuencia de que el conjunto de marcados alcanzables en la red no autónoma está contenido en el conjunto de marcados alcanzables en la red autónoma.

En conclusión, la limitación y la exclusión mutua se conservan al hacer que la red evolucione de forma no autónoma. De forma más general se puede afirmar que *todas las relaciones invariantes sobre los marcados* (§4.7.7) *siguen siendo válidas a pesar de que la red no sea autónoma.* Los invariantes de disparo serán en general válidos, aunque algunos de ellos puedan ser afectados por la interpretación que se le asocie a la red. Desgraciadamente, la vivacidad no se conserva, pero su estudio sobre las redes autónomas es interesante, puesto que es razonable admitir que una «buena» descripción debe poseer una estructura tal que la red autónoma sea viva.

EJERCICIO. Para la ciclicidad de una red no autónoma ¿es necesaria y/o suficiente la ciclicidad de la red autónoma correspondiente?

5.4 VALIDACIÓN DINÁMICA DE UNA DESCRIPCIÓN

En este apartado consideramos brevemente lo que se denominan métodos dinámicos

de validación y que, fundamentalmente, consisten en la utilización de técnicas de simulación.

Desde un punto de vista conceptual, cuando los métodos de análisis estático (véase el capítulo 4) son aplicados a RdP autónomas, dan resultados exactos, absolutos. Normalmente los métodos de análisis dinámico no producirán nunca resultados absolutos, dado que no se basan en la demostración de una propiedad, sino que persiguen una comprobación parcial, relativa al ámbito específico del funcionamiento simulado. De forma más precisa, se puede decir que el análisis por simulación permite obtener una cierta confianza sobre la descripción o bien evidenciar determinadas situaciones mal descritas (incompatibilidades, olvidos, etc.).

La simulación del funcionamiento de un sistema descrito con una RdP interpretada implica la simulación de los sistemas con los cuales interacciona. Para ello, existen sistemas de ayuda al diseño (*Computer Aided Design*, CAD) concebidos bajo esa óptica.

La verificación por simulación de las propiedades características del buen funcionamiento puede abordarse directamente, a través del estudio de las evoluciones del marcado inducidas por las secuencias normales de funcionamiento (éstas las debe introducir explícitamente el diseñador o emplear un subsistema para su generación). Se concluirá sobre la calidad del comportamiento obtenido, al observar, en cada paso de simulación (o cada k-pasos), las propiedades de buen funcionamiento objeto de verificación. Una mejora sustancial de la calidad de las conclusiones que se puedan extraer de la simulación, se obtiene al realizar, previa o simultáneamente, un análisis de la RdP autónoma. De esta forma, se podrán caracterizar con eficiencia algunas situaciones de violación de las propiedades de buen funcionamiento. Por ejemplo, se puede examinar el marcado de un cerrojo durante la simulación. Evidentemente, si un cerrojo se vacía, la RdP será no viva, independientemente de la interpretación asociada.

Esta aproximación híbrida a la simulación permite la detección de funcionamientos defectuosos, incluso si éstos son muy difíciles de observar a partir de una simulación directa, como es el caso de la existencia de bloqueos parciales (RdP interpretada parcialmente viva).

En resumen, la simulación de una RdP interpretada constituye una ayuda más para el difícil problema que nos ocupa: la validación del modelo de un sistema construido con RdP interpretadas. Por otro lado, la simulación es hoy en día una herramienta muy importante para el estudio de prestaciones del sistema modelado, pues permite obtener valores sobre tiempos de respuesta a determinadas solicitaciones, tasas de actividad de procesos o de utilización de recursos, etc. Es decir, un simulador adecuadamente diseñado permite estudios *funcionales* y *comportamentales* sobre el modelo o modelos del sistema que se considere.

5.5 MÉTODOS DE DESCRIPCIÓN DE SISTEMAS COMPLEJOS: ASPECTOS BÁSICOS

Los diferentes conceptos y resultados expuestos a lo largo del presente texto, aconsejan la definición de algunas líneas directrices que permitan abordar con eficacia la

descripción y la validación de sistemas, especialmente si existe un alto grado de concurrencia.

Todo método de descripción de un sistema complejo basado en una iteración sobre las fases de descripción y validación adolece de algunas dificultades importantes puesto que el proceso de descripción-validación ha de ser reiterado tantas veces como la validación arroje resultados insatisfactorios. Este proceso iterativo conlleva dos inconvenientes básicos:

1) la *falta de criterios* generales para proceder a la modificación (corrección) del modelo que no ha cumplido los requisistos exigidos en la validación.
2) la *dificultad operativa* intrínseca a la fase de validación. Obviamente esta segunda dificultad se hallará atenuada si se dispone de un sistema de concepción asistida por computador (*Computer Aided Design,* CAD).

Partiendo de estas reflexiones, parece deseable enunciar unas recomendaciones generales para la descripción de sistemas complejos. Estas recomendaciones deben guiar la especificación y modelación de los sistemas, así como facilitar la validación de los modelos resultantes.

Los apartados que siguen presentan dos aproximaciones complementarias a la descripción: la *descendente* y la *modular.*

La descripción descendente permite simultanear la modelación de un sistema y su validación. Todo proceso de descripción descendente opera detallando de forma progresiva las acciones que se deban realizar; es decir, *refinando* el modelo. Se trata del proceso inverso al de reducción de un modelo (§4.5). Normalmente se pretenderá que, *por construcción, el modelo sea válido.*

Cuando un sistema es muy complejo, a veces resulta difícil su descripción empleando sólo un razonamiento de tipo deductivo (descripción descendente). En estos casos puede resultar más adecuado elaborar una descripción en dos etapas:

1) Descripción independiente de diferentes *subsistemas.* La descomposición inicial del sistema es tarea del diseñador.
2) Establecimiento de las diferentes *relaciones* existente entre subsistemas. Estas relaciones pueden clasificarse en dos grandes grupos que son las sincronizaciones entre actividades que persiguen directamente un objetivo común (relaciones de *cooperación*) y las sincronizaciones derivadas de la utilización compartida de recursos (relaciones de *competencia*)

A esta última forma de proceder en la construcción de un modelo para un sistema, se denomina *método de descripción modular.*

Desde un punto de vista conceptual, ambas aproximaciones permiten aplicar, según principios diferentes y complementarios, el célebre aforismo «divide y vencerás». En efecto, en una descripción descendente totalmente estructurada se parte de un modelo inicial que sitúa las fases principales del funcionamiento para, posteriormente, desarrollar de forma *independiente* la descripción de cada fase.

5.5.1 Descripción descendente (*top-down*)

En todo proceso de descripción descendente de un sistema se parte de modelos condensados e incompletos del mismo (*descripciones abstractas*). Posteriormente, me-

diante sucesivas transformaciones locales del modelo original, se van obteniendo descripciones más detalladas (*concretas*). Las sucesivas transformaciones se denominan con el nombre genérico de *refinamientos*. Básicamente, en un refinamiento se expresa una macroacción en función de acciones y eventos más elementales. Dado que una macroacción puede estar asociada al disparo de una transición o al marcado de un lugar, hablaremos de refinamientos de transiciones y de lugares. En términos estructurales, el refinamiento de una transición o de un lugar se realizará sustituyendo la transición o el lugar por una determinada subRdP.

Además de los refinamientos de lugares y transiciones, en el proceso de descripción se podrán introducir lugares implícitos, los cuales crearán nuevas componentes conservativas.

5.5.1.1 *Descripción descendente y reglas básicas de expansión*

En una primera aproximación, podemos considerar las reglas de reducción (§4.5 y §3.3) tomadas en sentido inverso, como reglas de expansión (refinamiento). Actuando de esta forma, es posible construir una descripción detallada, preservando las propiedades que se deseen validar (vivacidad, limitación, exclusión mutua,...). Así se podrá:

(1) *Sustituir*: *a*) lugares por subRdP reducibles a un lugar (§4.5.1);
b) conjuntos de transiciones que representen secuencias de eventos/ /acciones por un lugar y otro conjunto de transiciones (§4.5.2).

(2) *Añadir*: *a*) lugares implícitos (§3.3);
b) transiciones idénticas y transiciones identidad (§4.5.3).

Desde un punto de vista práctico, conviene considerar especialmente los casos particulares de reglas de expansión definidos por la figura 4.26 así como la regla de la figura 4.26*bis*. Ciñámosnos de momento a las reglas ilustradas por la figura 4.26:

(1.*a*) En su forma más elemental, la sustitución de un lugar por una subRdP reducible (RA.2) permite expresar, en función de acciones más elementales, una macroacción secuencial asociada a un lugar.

(1.*b*) En su caso más simple, la sustitución de un conjunto de transiciones por un lugar y otro conjunto de transiciones (RA.1) permite expresar, en función de acciones más elementales, una macroacción secuencial asociada a una transición. También permite asociar a un lugar (el que se añade) la acción que estaba asociada a la transición.

(2.*a*) La adición de lugares implícitos puede tener dos objetivos básicos:
1) Posibilitar la posterior definición de evoluciones paralelas en las que se sincroniza su comienzo y su terminación (RB.1).
2) Posibilitar la posterior definición de la actividad de un subsistema que evoluciona concurrentemente (RC.1). La introducción de un lugar identidad permitirá la imposición de ciertas restricciones a la evolución de la RdP. Por ejemplo, para garantizar la exclusión mutua en el acceso a un recurso.

(2.*b*) La adición de transiciones idénticas (RB.2) permitirá la expresión de alternativas (selecciones en la evolución). En su forma más elemental, la adición

de transiciones identidad (RC.2) posibilitará la expresión de iteraciones en la evolución.

Como puede comprobarse, las reglas RB.2 y RC.2 parten de una asignación de las macroacciones a las transiciones. Aplicando posteriormente la regla RA.2 a las mencionadas transiciones, se pueden asociar a lugares las acciones alternativas o la acción a iterar.

Para ilustrar las ideas expuestas vamos a construir por refinamientos sucesivos, por un lado, un modelo para el sistema productor-consumidor con accesos excluyentes al almacén (§2.5.2) y, por otro, un modelo para el sistema de los cuatro filósofos (§5.2.2).

EJEMPLO. SISTEMA PRODUCTOR-CONSUMIDOR CON EXCLUSIÓN MUTUA. Puesto que el enunciado es ya conocido (§2.4.1), procedemos a establecer una descripción (modelo). La figura 5.8 ilustra las diferentes fases de un proceso de refinamiento. El almacén es el subsistema sobre el que se desarrollará la iteración entre los subsistemas de producción y de consumo (figura 5.8*a*). O y H representan el número de objetos y de huecos en el mismo, respectivamente. En la figura 5.8*b* se han añadido tres lugares implícitos (identidad). Sus funciones son las siguientes:

1) Representar la actividad del agente productor (P_r).
2) Representar la actividad del agente consumidor (C_s).
3) Garantizar la exclusión mutua en los accesos al almacén (recurso) (A_R).

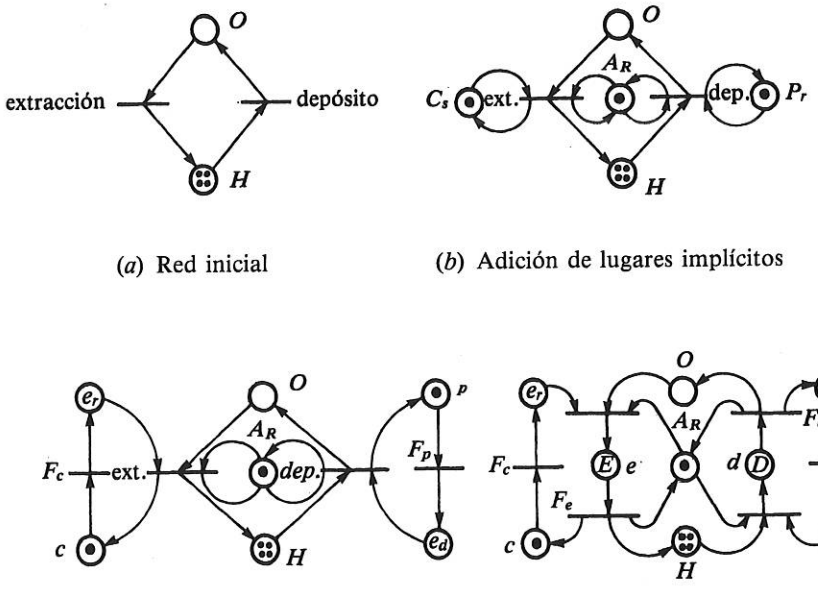


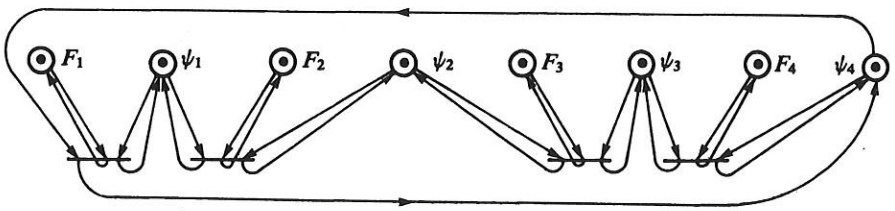
Figura 5.8. Construcción por refinamientos sucesivos de una descripción del sistema productor-almacén-consumidor con accesos excluyentes al almacén.

Gracias al lugar monomarcado A_R no se pueden disparar simultáneamente las transiciones etiquetadas por *extracción* y *depósito*. Es decir, ambas operaciones están en exclusión mutua.

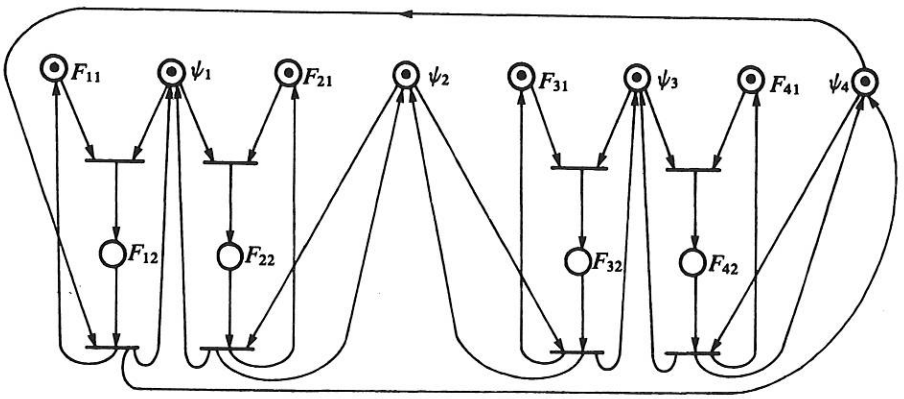
En la figura 5.8*c* se ha detallado la actividad de los agentes productor y consumidor. Ésta se compone de dos estados, que son la producción (respec. consumo) y la espera para acceder al almacén. Por último, la RdP de la figura 5.8*d* refleja en detalle el acceso al almacén. Si comparamos las figuras 2.21 y 5.8, observaremos que, independientemente de la disposición de los elementos, ambas redes describen la misma relación evento-acción.

Es importante observar, a modo de comentario final, el papel desempeñado por las componentes conservativas elementales. En efecto, todas se encuentran en la figura 5.8*b* y su significación ya ha sido comentada. Por otro lado, la RdP final es una red simple, lo cual garantiza que el sistema esté libre de monopolio.

EJEMPLO. PROBLEMA DE LOS FILÓSOFOS. Su enunciado fue presentado en §5.2.2. En la figura 5.9*a* se establecen las relaciones fundamentales entre usuarios (filósofos) y recursos (tenedores). Como podrá observarse con facilidad, la RdP es no simple. Para que la descripción no posea monopolio, se ha procedido a una primera expansión (figura 5.9*b*). La regla de expansión utilizada es una generalización de la ilustrada por la figura 4.26*bis*. El significado de la descripción obtenida es evidente, así como el posterior proceso de refinamiento (éste conduce al modelo de la figura 5.5).



(*a*) RdP no simple en la que se establece la relación general entre usuarios y recursos (filósofos y tenedores respectivamente).



(*b*) RdP simple obtenida al refinar la red de la figura anterior.

Figura 5.9. Refinamientos y problema de los filósofos.

Todo método de descripción basado en la aplicación de las reglas básicas de expansión (figuras 4.26 y 4.26*bis*) implica la interesante propiedad que enunciamos a continuación.

Proposición 5.8

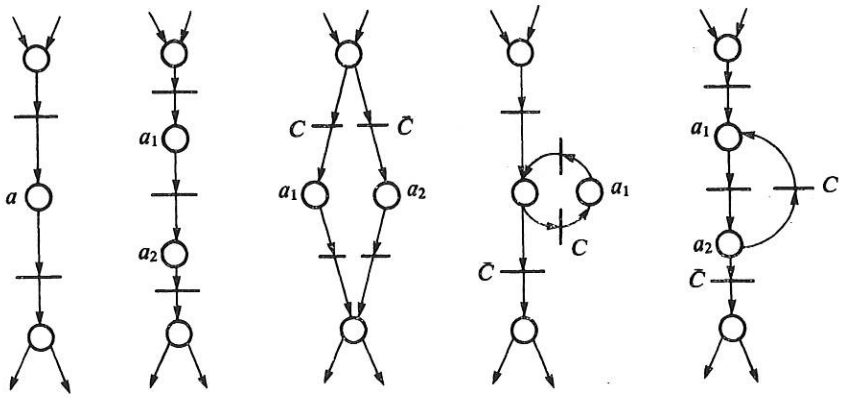
1) La red marcada final, $\langle \boldsymbol{R}, M_0 \rangle$, es viva, sii la red inicial $\langle \boldsymbol{R}_{\text{in}}, M_{\text{in}_0} \rangle$ es viva.
2) Si k_j representa el número de marcas que posee el j-ésimo lugar implícito añadido durante el proceso de construcción del modelo, y k_0 es el límite de la red inicial, $\langle \boldsymbol{R}_{\text{in}}, M_{\text{in}_0} \rangle$, el límite de la red marcada final es $k = \text{máx}(k_0, k_1, \ldots)$. □

DEMOSTRACIÓN. La vivacidad se garantiza dado que toda red construida de acuerdo con esta aproximación es reducible a la red inicial. La k-limitación se puede demostrar observando que al añadir los lugares implícitos se introducen componentes conservativas elementales cuyos lugares poseen como máximo k_j marcas. □

5.5.1.2 *Descripción descendente y subRdP*

En el apartado anterior se consideró el proceso de refinamiento utilizando exclusivamente reglas básicas. Desde un punto de vista práctico, interesa con frecuencia la sustitución directa de una transición o de un lugar por una subRdP capaz de representar un determinado conjunto de evoluciones. Así, surge de forma natural la idea de construir *catálogos* de subRdP. En éstos se incluirán aquellas subredes de uso más frecuente para la clase de aplicaciones que se modele.

Si en el marco de una *evolución puramente secuencial*, una determinada macroacción «*a*» asociada a un lugar debe ser expresada en función de acciones más elementales, se recomienda la utilización de subredes obtenidas por composición entre las reglas de expansión estructurada. En la figura 5.10.1 se representa la macroacción



(1) Red original (2) Secuencia (3) Discriminación condicional (4) Iteración (5) Repetición

Figura 5.10 Reglas de expansión secuencial estructurada cuando las acciones están asociadas a los lugares. (La repetición puede obtenerse a partir de otras reglas; es redundante.)

que se desea refinar. Las figuras 5.10.2, 3 y 4 representan las reglas básicas de expansión estructurada *secuencial*:

- *secuencia*: $a = a_1; a_2$;
- *discriminación condicional*: $a =$ **si** C **entonces** a_1 **si no** a_2;
- *iteración*: $a =$ **mientras que** C **hacer** a_1;

La figura 5.10.5 representa un ejemplo de regla de expansión estructurada no básica pero de gran utilidad práctica. Esta regla se puede obtener por composición de las reglas básicas de iteración y de secuenciación:

- *Repetición*: $a =$ **repetir** $[a_1; a_2]$ **hasta** $\bar{C}$; $= \begin{cases} [a_1; a_2]; \\ \textbf{mientras que } C \textbf{ hacer } [a_1; a_2]; \end{cases}$

EJERCICIO. Obténgase la regla de repetición a través de un proceso de composición de las reglas básicas y una posterior simplificación de la subRdP. (*Sugerencia*: la regla de simplificación que se debe utilizar es la de *fusión de lugares equivalentes*, §3.4.1.)

La experiencia acumulada utilizando las anteriores subredes en el proceso de refinamiento ha demostrado que se cometen pocos errores, así como se mejora ostensiblemente la legibilidad, la comprensibilidad y la modificabilidad de los modelos. Indudablemente, la autodisciplina en la modelación de las evoluciones secuenciales que conlleva la utilización exclusiva de las anteriores reglas de expansión puede tener como contrapartida que el modelo no sea mínimo en número de lugares o/y transiciones. Ahora bien, como se dijo en el capítulo 3, optimizar una descripción con respecto a un criterio puede llevarnos a un mal modelo con respecto a otros criterios.

Observación importante. Si la *iteración* o *repetición* en la ejecución de una acción *a* tiene el sentido de mantenerla y no el de reejecutarla (por ejemplo, mantener abierta una válvula hasta que un nivel sea el adecuado), las subredes de las figuras 5.10.4 y 5 degeneran en la de la figura 5.10.1. (Compruébese.) Ahora bien, en esta última subred, la transición de salida del lugar que tiene asociada la acción estará condicionada por la negación de C (condición de mantenimiento de la acción *a*). Para evitar ambigüedades, en este caso se especificará «**mantener** *a* **hasta que** $\bar{C}$». Evidentemente esta regla es fácilmente generalizable tanto si se ejecutan varias acciones como si se esperan diversas condiciones que conduzcan a diferentes estados.

Dado que todas las subRdP de la figura 5.10 son reducibles a un lugar (§4.5.1, $\mathfrak{R}_1$), si la RdP inicial es viva, binaria y cíclica, también lo será la red que se obtiene después de la expansión (independientemente de que la red inicial exprese evoluciones simultáneas).

Además de las reglas de expansión secuencial estructurada (figura 5.10) se puede definir una regla de expansión estructurada que cree ramas con evoluciones paralelas. Ésta consiste simplemente en desdoblar el lugar al que se asocian las macroacciones en otros tantos lugares (implícitos) que posean idénticos conjuntos de transiciones de entrada y de salida. A cada uno de los lugares resultantes se les asocia una o un conjunto de acciones o macroacciones.

La utilización exclusiva de las reglas de expansión estructurada a partir de un lugar conduce a subRdP que pueden representar evoluciones paralelas y que son reducibles a un único lugar. Aquellas subRdP compuestas de mayor utilidad serán candidatas a ser *catalogadas* (eventualmente tras una simplificación) para su reutilización inmediata.

EJERCICIO. Exprésese de forma algorítimica la macroacción a en función de a_1, a_2 y a_3 (figura 5.11). ¿Cuál es el comportamiento global de la red?

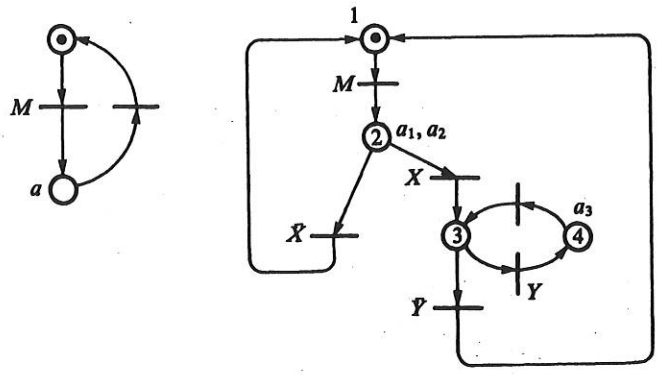


Figura 5.11. Expansión de la acción (estructurada) «a».

Antes de abordar brevemente el refinamiento de transiciones, interesa resaltar que la utilización de las reglas de expansión estructurada debe completarse con otras reglas (adición de lugares, etc.) de forma que se puedan insertar restricciones sobre las evoluciones simultáneas en el sistema objeto de modelación. Dicho de otro modo, la consideración exclusiva en un ámbito concurrente de las reglas de expansión estructurada puede ser insuficiente.

A lo largo de este apartado se ha comentado la sustitución de lugares por subRdP (su expansión). De forma dual, es posible hablar de la *sustitución de transiciones por subredes*. La figura 5.12 es suficientemente explicativa en este sentido. Si la subRdP es reducible a una transición, al sustituir la transición se preservará la vivacidad, la limitación y la ciclicidad. Así, la subRdP de la figura 5.12*a* es reducible a una transición; puesto que la red de la figura 5.12*b* es viva y binaria, la red que se obtiene tras la sustitución (figura 5.12*c*) es viva y binaria.

Si la subRdP es irreducible, se pueden plantear dificultades al sustituir una transición, puesto que la red resultante puede hacerse no viva y/o no limitada y/o no cíclica. En este sentido es altamente recomendable el estudio detallado del ejemplo siguiente.

EJEMPLO. La RdP marcada de la figura 5.13*a* es viva y cíclica. La subRdP de la figura 5.13*b* no es reducible a una transición. La red que se obtiene al añadirle el lugar monomarcado O' es viva y cíclica. Del mismo modo, la red que se obtiene al añadirle el lugar O' con dos

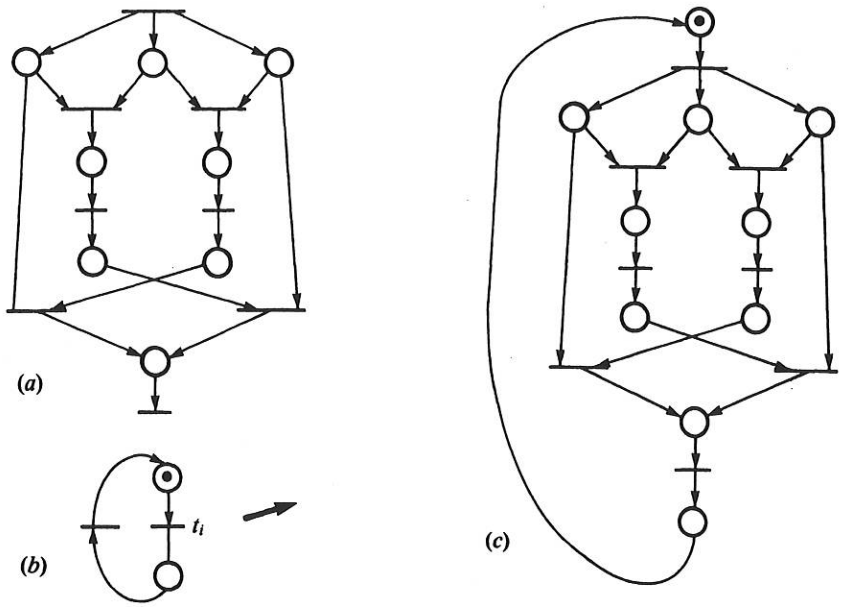


Figura 5.12. Sustitución de t_i por una subRdP.

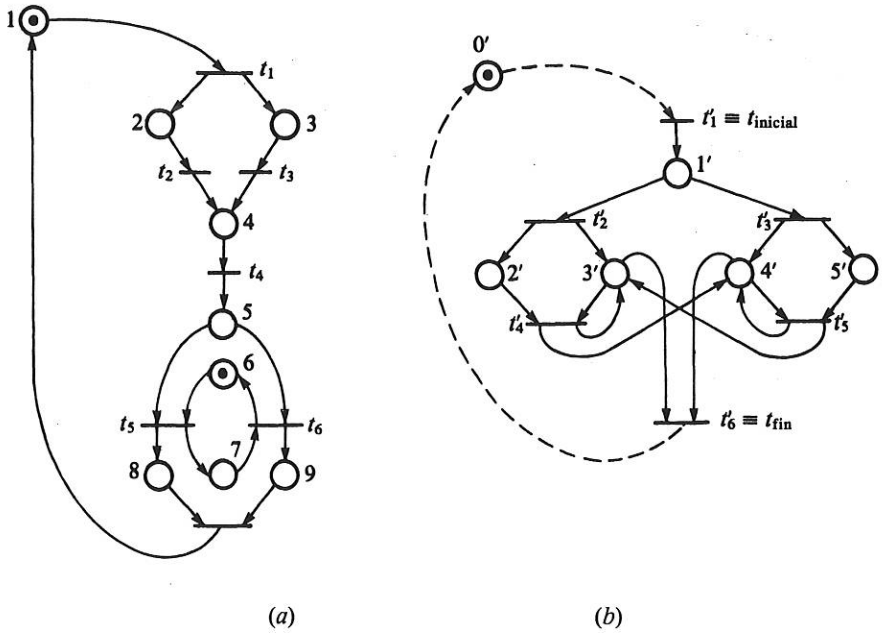


Figura 5.13. La sustitución de la transición t_4 (2-sensibilizable) por la subred conduce a una red no viva.

marcas es también viva y cíclica. (Compruébese.) Sin embargo, la sustitución de la transición t_4 de la figura 5.13*a* ($\exists M_k$ tal que $M_k(p_4) = 2\alpha(p_4, t_4) = 2$; es decir, para M_0, t_4 es 2-*sensibilizable*) por la mencionada subred conduce a una red global que no es ni viva ni cíclica, circunstancia que es fácil de comprobar aplicando, por ejemplo, la secuencia de disparos $\sigma = t_1t_2t_3t_1't_2't_1't_3't_6't_5$. De forma intuitiva se puede comprender la no vivacidad de la red final al constatarse en la figura 5.13*b* que si $M_0(O') = 2$, existen secuencias para las que el disparar dos veces la transición t_6' requiere más de dos disparos previos de t_1', lo que es imposible en la red de la figura 5.13*a*. Por otro lado, la red considerada con $M_0(O') = 1$ es tal que toda secuencia de disparos que comienza disparando una vez t_1' permite disparar una vez t_6' sin que se tenga que volver a disparar t_1'. En este caso, la sustitución de cualquier transición que no sea 2-sensibilizable (todas las de la figura 5.13*a* menos t_4) por la subRdP de la figura 5.13*b* conduciría a una RdP viva y cíclica.

Los estudios generales sobre condiciones suficientes para sustituir una transición por una subRdP se basan en la *teoría de lenguajes formales* (véase, por ejemplo, [ANDR 80], [SUZU 80]). Su consideración escapa a los objetivos básicos de este texto. En cualquier caso, debe indicarse que, desde un punto de vista práctico, es muy razonable pensar que las subredes más utilizadas son reducibles, con lo que el refinamiento no planteará problema alguno.

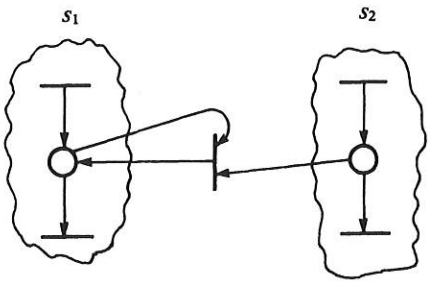
Para concluir este apartado sobre descripción descendente (§5.5.1) hemos de significar que el proceso de modelación del sistema aparece como una serie de transformaciones sucesivas de descripciones abstractas. Cada transformación (*refinamiento*) acerca el modelo abstracto a una descripción completa. Como recurso básico del diseñador ha aparecido el concepto de *catálogo* de subRdP, que no es más que una colección de soluciones a problemas típicos encontrados en la (su) práctica. En cualquier caso, es evidente que la existencia de estos catálogos permite una gran economía de esfuerzos en el modelado de sistemas complejos.

5.5.2 Descripción modular

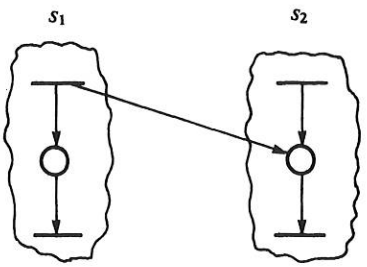
La comprensión del funcionamiento de un sistema complejo es tanto más fácil cuanto su descripción se aborde de forma más estructurada y progresiva. De este modo, en gran número de aplicaciones complejas, los sistemas concurrentes se describen sincronizando las descripciones (realizadas independientemente) de diferentes subsistemas o módulos. A continuación introducimos algunos de los esquemas elementales de sincronización más frecuentes, señalando posibles inconvenientes que pueden plantearse de cara a la validación del sistema completo.

La figura 5.14 presenta algunos esquemas de interconexión entre subsistemas (*sincronizaciones*). La significación de cada uno de ellos es suficientemente clara. A modo de comentario sólo indicaremos que el esquema de autorización memorizada de evolución mutua (doble semáforo, figura 5.14*e*) es, de acuerdo con la regla de reducción ilustrada por el ejercicio 4.8, equivalente a los esquemas de sincronización mutua con evolución simultánea. (Compruébese.) Para ilustrar esta forma de proceder, vamos a modelar un sistema ya conocido, aunque su complejidad no sea importante.

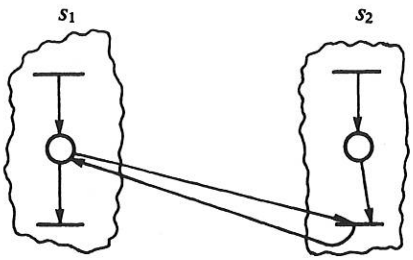
EJEMPLO. Reconsideremos una vez más el sistema productor-consumidor con accesos excluyentes a almacén (§2.4.1 figura 2.6; §5.5.1.1, figura 5.8). En este caso, los dos subsistemas



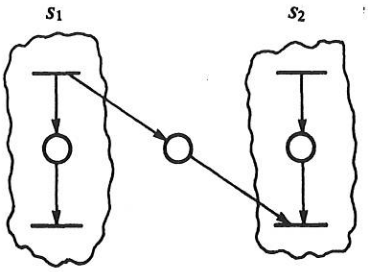
(*a*) S_1 desactiva a S_2



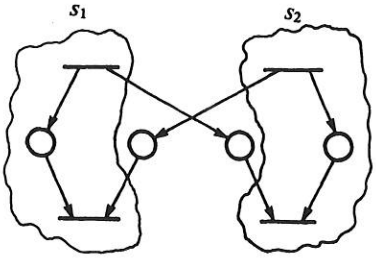
(*b*) S_1 activa a S_2



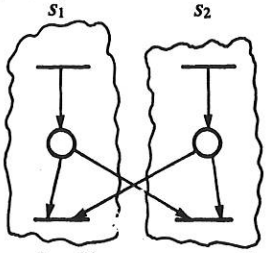
(*c*) S_1 autoriza sin memorizar la evolución de S_2



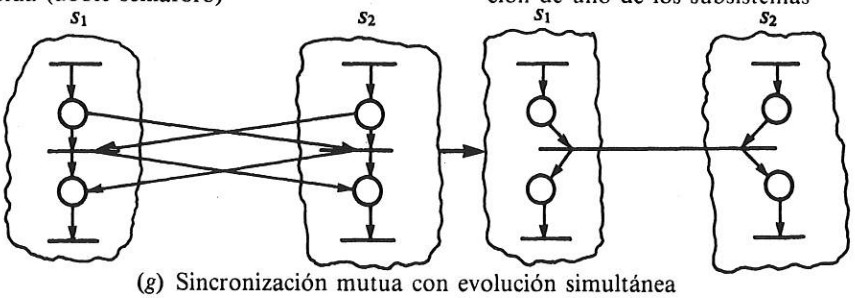
(*d*) S_1 otorga a S_2 una autorización memorizada de evolución (semáforo, figura 4.41)



(*e*) Autorización memorizada de evolución mutua (doble semáforo)



(*f*) Sincronización mutua con desactivación de uno de los subsistemas



(*g*) Sincronización mutua con evolución simultánea

Figura 5.14. Algunos esquemas típicos de sincronización entre módulos. (*Nota.* «Activación» y «desactivación» tienen pleno sentido, desde un punto de vista más formal, si los subsistemas son, por ejemplo, grafos de estado monomarcados.)

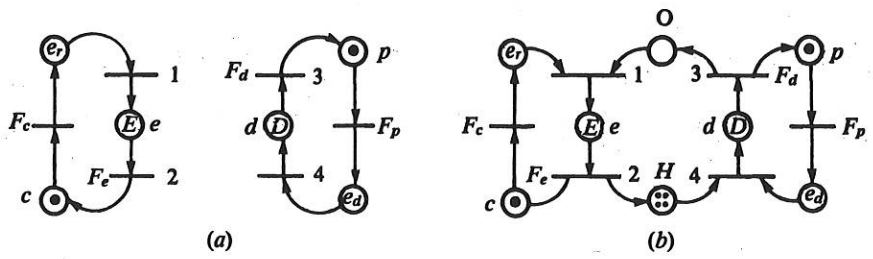


Figura 5.15. Construcción por interconexión de descripciones parciales de una descripción del sistema productor-almacén-consumidor.

activos tienen un funcionamiento similar, descrito por la figura 5.15*a*. Como puede apreciarse, se distinguen un estado de trabajo (producción o consumo), uno de eventual espera para acceder al recurso (almacén) y uno de trabajo con el recurso (depósito o extracción). El funcionamiento descrito por la figura 5.15*a* es incompleto puesto que se permite retirar objetos, incluso si no los hay (¡!). En efecto, al ser independiente el funcionamiento del productor y del consumidor, nada impide que el proceso de consumo sea más rápido que el de producción. Para resolver este problema se introduce un *semáforo* (figura 5.14*d*) por el que el subsistema de producción otorga una autorización (al haberse producido un objeto) de evolución al subsistema de consumo (operación de retirar el objeto producido). Este semáforo está representado por el lugar O (figura 5.15*b*), su función es, en resumen, memorizar los objetos depositados y no retirados aún. Es decir, cuenta los objetos existentes en el almacén si no están en curso operaciones de depósito o extracción. El lugar O impone un avance sincrónico nulo entre las transiciones $\{1,2\}$ y $\{3,4\}$.

La adición del lugar O, hasta ahora la única restricción entre las evoluciones del productor y el consumidor, no impide que si el productor es más rápido que el consumidor, se intente depositar un número no limitado de objetos en el almacén (la RdP es no limitada si sólo se añade el lugar O). Dado que todo almacén será de capacidad finita, si éste está lleno, es importante no autorizar una operación de depósito hasta que se haya producido una de extracción. Esto es, hay que limitar el avance de las transiciones $\{3,4\}$ con respecto a $\{1,2\}$. Este avance se limita añadiendo un lugar (semáforo) el cual vendrá a contar el número de huecos existentes en el almacén. Si hay huecos, se autoriza una operación de depósito. Si se supone que el sistema arranca con el almacén vacío, el lugar H tendrá tantas marcas como capacidad el almacén. En este punto es interesante observar que los lugares añadidos, O y H, pertenecen a una componente conservativa, lo que garantiza la limitación de la RdP obtenida: $M(O) + M(H) + M(E) + M(D) = 4$.

La RdP de la figura 5.15*b* no cumple todas las especificaciones del problema puesto que nada impide un acceso simultáneo al almacén por parte del productor y el consumidor. Para garantizar la exclusión mutua entre los lugares E y D, $M(E)M(D) = 0$, se puede crear otra componente conservativa añadiendo un lugar A_R. Es decir, A_R debe ser tal que $M(E) + M(D) + M(A_R) = 1$, lo cual exige que $I(E) + I(D) + I(A_R) = 0$. Haciendo el cálculo se obtiene A_R (figura 5.8*d*) y la descripción cumple las especificaciones. Como es fácil comprobar, el subconjunto de lugares que determinan la componente conservativa, $\{E, D, A_R\}$, representa los tres estados de utilización en que se puede encontrar el almacén (recurso). El marcado de A_R representa la inactividad del recurso; A_R es un *semáforo de exclusión mutua*, relación de sincronización no ilustrada en la figura 5.14.

Observación. El razonamiento anterior permite construir una RdP en la que de forma estructural se fuerza la exclusión mutua a partir del marcado inicial. Este método de construcción de la exclusión mutua no es el único, puesto que se podía haber razonado de forma que se obtuviese un esquema como el basado en la red de la figura 4.36. No obstante, calificaremos de mala aproximación a esta última pues, además de obtenerse una red más compleja, la demostración de la exclusión mutua es netamente más complicada (no existe una componente conservativa que describa los estados del recurso).

El ejemplo anterior ha permitido esbozar una aproximación a la construcción de modelos cuando se parte de la descripción independiente de subsistemas. Su desarrollo ha mostrado que, dada la forma de proceder, al irse introduciendo restricciones sobre los disparos de transiciones (es decir, al introducir lugares) se pueden obtener comportamientos anómalos (no limitación,...). A diferencia de un método de descripción descendente en el que *por construcción* el modelo es válido, en este caso hay que tener en cuenta el análisis del modelo obtenido. Así, por ejemplo, las relaciones *activar* (figura 5.14*a*) o *desactivar* (figura 5.14*b*) no pueden ser las únicas establecidas entre los dos subsistemas secuenciales de un determinado sistema, puesto que, de lo contrario, el modelo sería estructuralmente no limitado o no vivo, respectivamente.

EJERCICIO. Supóngase que se tienen descritos dos subsistemas secuenciales, siendo sus modelos vivos y binarios. Determínese cuales son las relaciones de sincronización que pueden utilizarse aisladamente y preservan la vivacidad y la binariedad.

5.6 CONCLUSIÓN

Los resultados fundamentales de este capítulo han sido los siguientes:

1) No se pueden trasladar directamente y de forma general los resultados del análisis de las RdP autónomas sobre las RdP no-autónomas (temporizadas o interpretadas).
2) Toda RdP simple marcada y viva es libre de monopolio.
3) El análisis por simulación no garantiza ninguna propiedad de la descripción, salvo si éste es exhaustivo.
4) La descripción descendente permite obtener modelos que, por construcción, son válidos.
5) La descripción modular es necesaria para describir sistemas muy complejos. No obstante, ésta puede acarrear algunos problemas al modelo construido que, de forma intuitiva, han sido resaltados.

Por último, es importante señalar que el tratamiento de los aspectos metodológicos que se han presentado no constituye más que una primera aproximación al problema de la modelación de sistemas complejos. Interesa resaltar, en cualquier caso, que las buenas metodologías de descripción estarán siempre definidas en función de los resultados del análisis.

En resumen, no sólo conviene disponer de una potente herramienta de descripción (las redes de Petri), sino que es fundamental el utilizarla convenientemente. Estable-

ciendo un símil automovilístico, para ganar una competición no sólo es interesante disponer de un coche potente, sino que es fundamental el conducirlo de forma adecuada.

EJERCICIOS

5.1 Descríbase mediante refinamientos sucesivos el sistema compuesto por dos lectores y dos redactores (§2.5.5).

5.2 Repítase el ejercicio anterior construyendo una descripción a partir de la interconexión de las descripciones parciales de cada lector y cada redactor.

6

Realización cableada

6.1 INTRODUCCIÓN

Después de haber utilizado las redes de Petri para la descripción y validación de sistemas con evoluciones concurrentes, vamos a presentar las técnicas básicas que permiten *realizar* los sistemas descritos. Es decir, vamos a abordar el estudio de técnicas que posibiliten la construcción de dispositivos físicos capaces de *simular*, con mayor o menor precisión, el comportamiento del modelo funcional construido para el sistema. La realización aparece como la última fase del proceso de síntesis.

El estudio de técnicas de realización lo llevaremos a cabo en cuatro capítulos. El primero de ellos, éste que nos ocupa, aborda la realización mediante el conexionado directo de biestables y puertas lógicas (realización cableada); el segundo capítulo, capítulo 7, presenta técnicas para la realización de RdP utilizando las memorias muertas (ROM) y las matrices lógicas programables (PLA); el capítulo 8 estudia en profundidad los autómatas programables de uso general y su aplicación a la simulación de RdP; por último, el capítulo 9 está dedicado exclusivamente a la simulación de RdP con computadores de propósito general y a los autómatas programables especializados en la simulación de RdP.

El conjunto de métodos de realización que presentaremos a lo largo de los capítulos citados se centra esencialmente en las RdP *binarias*, clase de modelos que, como se recordará (§1.5 y 6, §2.4 y 5), es bastante adecuada para la descripción de sistemas lógicos. No obstante, observaremos que gran parte de los métodos de realización de RdP binarias que se estudien son inmediatamente generalizables para realizar redes de Petri ordinarias k-limitadas.

Las técnicas de realización que se presentan en este capítulo permiten la obtención *directa* del circuito lógico a partir de la RdP, con lo que se suprimen dos de las más delicadas y fastidiosas etapas de la síntesis clásica de los sistemas secuenciales†: *la codificación del estado* y *la escritura de ecuaciones lógicas*. Por otro lado, es importante señalar que, además de las reglas de conexionado de los dispositivos lógicos

† Véase, por ejemplo, [HILL 78].

básicos (puertas y biestables), se presenta una metodología extremadamente simple para el estudio de *funcionamientos secuenciales aleatorios*. En particular, se proponen técnicas de conexionado que garantizan la ausencia de los mencionados funcionamientos aleatorios.

La idea fundamental que preside las técnicas de realización cableada que se proponen consiste en materializar cada lugar (binario) por un biestable que memorice si el lugar está o no marcado. Diremos que se trata de *realizaciones modulares* de las RdP. La realización modular de RdP no binarias utilizará por cada lugar que pueda contener hasta k marcas, un *contador módulo* $k + 1$ o mayor. (*Nota*. El $+1$ es necesario para memorizar la ausencia de marcas, $M(p) = 0$.)

La realización cableada de RdP utilizando las ideas antes esbozadas conducirá a realizaciones en las que el número de memorias (biestables y contadores) no será normalmente mínimo†. No obstante es importante señalar que actualmente el coste de los materiales decrece continuamente y que el esfuerzo de economización debe llevarse a cabo sobre otros aspectos como son *el diseño*, *la puesta a punto* y *el mantenimiento del circuito*. En este sentido, es de destacar que la realización de RdP utilizando las técnicas clásicas de síntesis de máquinas secuenciales es doblemente complicada. En efecto, antes de aplicar las mencionadas técnicas, se ha de transformar la RdP en uno o varios sistemas secuenciales. Esto puede hacerse de diversas formas, por ejemplo:

1) *transformando* la RdP en *un grafo reducido* de acuerdo con el método expuesto en §1.5.4 (atención al posible número de estados).
2) *descomponiendo* la RdP en *un conjunto de grafos reducidos interdependientes*, problema que se abordará en el próximo capítulo (§7.4) al considerar las realizaciones con ROM y PLA.

Por otro lado y dejando aparte la no despreciable complejidad operatoria de las técnicas clásicas de síntesis de máquinas secuenciales, se ha de mencionar que los circuitos que se obtienen se caracterizan normalmente por su *dificultad de comprensión* y de *modificación*.

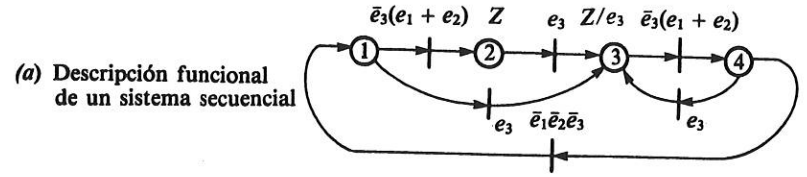
La dificultad de comprensión es fruto de la enorme diferencia que suele existir entre la *descripción funcional* del circuito (qué es lo que hace) y la *descripción estructural* del mismo (cómo está hecho). Como botón de muestra, el lector puede constatar en la figura 6.1 un sistema secuencial descrito funcionalmente mediante una RdP (GR) y dos posibles realizaciones. El origen de la diferencia entre la descripción funcional (figura 6.1*a*) y las estructurales (figuras 6.1*b* y *c*) reside en la fuerte codificación del estado (2 variables internas codifican los 4 estados). Además, con frecuencia, la significación funcional de las variables internas es muy difícil de establecer. Entre otros problemas, la dificultad de comprensión hace que *la puesta a punto* y *el mantenimiento* del circuito se compliquen.

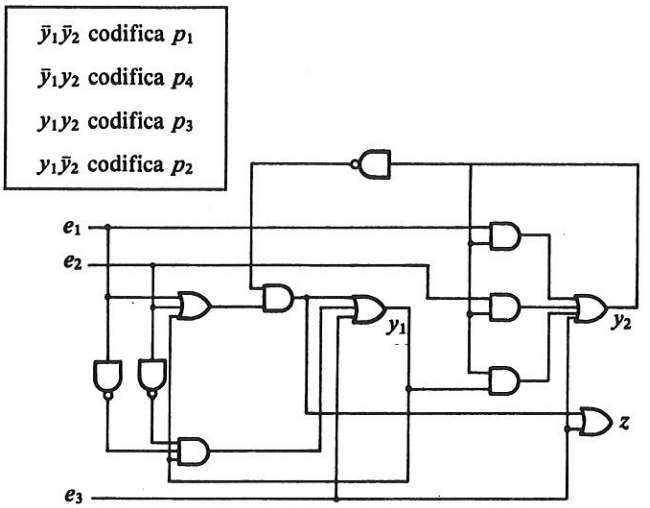
La dificultad de modificación proviene también de la fuerte codificación del estado puesto que, en estos casos, el circuito tiene sentido funcionalmente al considerarlo como un todo y no en sus partes. Dicho de otro modo, resulta muy complejo

---

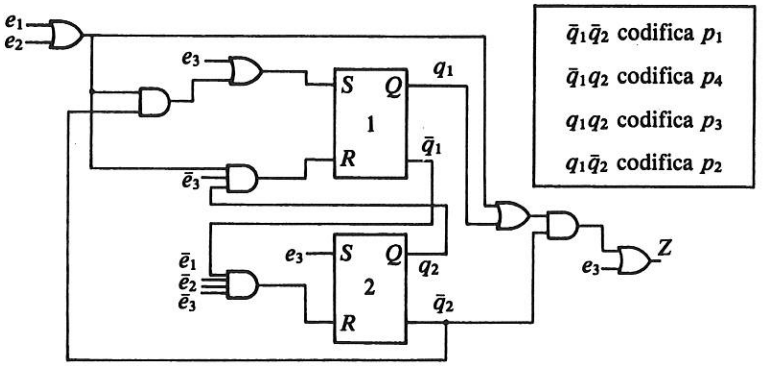
† Esto va en contra de un objetivo fundamental de los métodos clásicos de síntesis de máquinas secuenciales: minimizar el número de componentes a emplear en el circuito.

o incluso imposible proceder a *modificaciones locales*, lo cual implica un difícil mantenimiento en las instalaciones en las que evolucionen sus especificiaciones funcionales.





(*b*) Una realización lógica *sin* biestables (variables internas de memorización: y_1, y_2)



(*c*) Una realización lógica *con* biestables (variables internas de memorización: q_1, q_2)

Figura 6.1 Descripción funcional de un sistema secuencial y dos realizaciones lógicas obtenidas al codificar el estado con dos variables internas. (*Nota*. En la segunda realización Z puede generarse también como $Z = q_1(\bar{q}_2 + e_3)$.)

La realización *modular*, al establecer una relación directa entre la descripción funcional (RdP) y la estructural (circuito), facilita la comprensión y la modificabilidad del circuito, además de reducir enormemente el tiempo necesario en la fase de diseño.

En los apartados 6.2 y 6.3 se presentan realizaciones *asíncronas*, mientras que en el apartado 6.4 se estudian realizaciones *síncronas*. Por último (§6.5), se comentan rápidamente algunas cuestiones adicionales de gran interés práctico.

6.2 REALIZACIÓN ASÍNCRONA (I): FUNCIONAMIENTO POR TRANSFERENCIA IMPUSIONAL

Este método de realización de RdP binarias se basa en las dos ideas siguientes:

1) A cada lugar (binario) se le asocia un biestable especial, al cual denominaremos genéricamente *célula* (de memoria).
2) La activación de una célula se realiza al disparar alguna de las transiciones de entrada del lugar que materializa; análogamente, la desactivación se realiza al disparar alguna de los transiciones de salida de dicho lugar. Es decir, se sigue directamente la regla de evolución del marcado de una RdP binaria.

Puesto que la simulación del disparo de una transición es un pulso† y éste es el que provoca directamente toda la evolución del marcado, el presente modo de funcionamiento se denomina por *transferencia impulsional*. En lo sucesivo, se supondrá que la RdP no presenta conflictos efectivos.

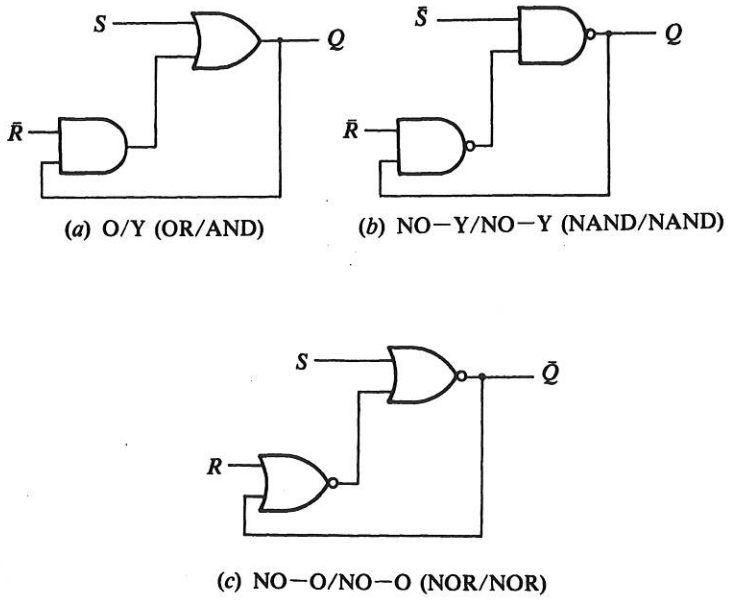
6.2.1 La célula de memoria

Antes de introducir la célula, consideramos rápidamente el biestable R-S con activación prioritaria. La figura 6.2 presenta su esquema lógico básico así como otras dos realizaciones alternativas que utilizan sólo puertas NOR y NAND, respectivamente. Con la ayuda de cualquiera de esos esquemas, se puede comprobar su comportamiento lógico (comportamiento ideal):

R	S	Q	Estado
0	0	Q	memorización
0	1	1	activación
1	1	1	activación (prioritaria)
1	0	0	desactivación

A modo de observación, es importante subrayar que el comportamiento expresado por la tabla anterior se circunscribe a los momentos en que no hay puertas lógicas

† Como veremos más adelante (§6.2.4), su duración es del orden de varias veces la de la conmutación de una puerta lógica.



(*a*) O/Y (OR/AND) (*b*) NO−Y/NO−Y (NAND/NAND)

(*c*) NO−O/NO−O (NOR/NOR)

Figura 6.2 Esquemas lógicos de un biestable R-S con activación prioritaria: $Q := S + Q\bar{R}$.

conmutando; es decir, cuando el circuito está en *régimen estático*. La importancia de esta precisión radica en que, merced a los retrasos en la conmutación de las puertas, cuando en un circuito se interconectan diversos componentes lógicos, pueden sobrevenir diferencias entre el comportamiento esperado y el observado. Estas diferencias se denominan con el nombre genérico de *fenómenos aleatorios*; en §6.2.4 se estudiarán condiciones que permiten garantizar su ausencia.

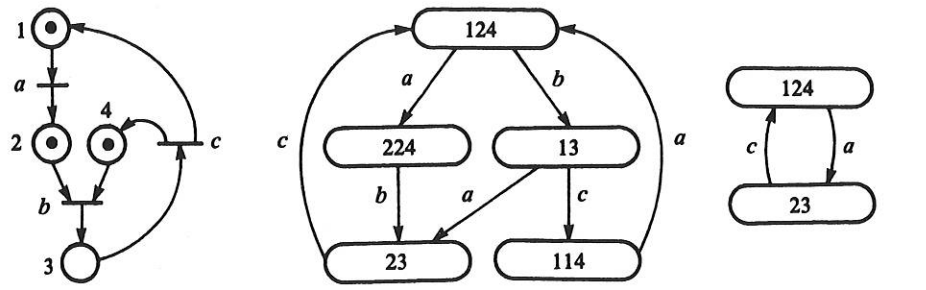
EJERCICIO. Compruébese que los biestables de la figura 6.2 pueden ser utilizados como biestables con desactivación prioritaria sin más que cambiar la denominación de sus entradas y salidas de acuerdo con la regla siguiente: R por S, S por R y complementar lógicamente la definición de la salida, $\bar{Q}$ por Q.

La célula de memoria necesaria para realizar modularmente RdP binarias es un biestable R-S (*Reset-Set*) *con activación prioritaria* (activación cuando $RS = 1$) y *generalizado* (con varias condiciones de activación, $\{C_{A_i}\}$, y varias condiciones de desactivación, $\{C_{D_j}\}$). En efecto, para realizar un lugar binario (1-limitado) hace falta una memoria que se active al dispararse una de las transiciones de entrada del lugar que materializa. Además, la memoria debe desactivarse al dispararse alguna de las transiciones de salida del lugar que realiza. Es decir, si denominamos C_{A_i} y C_{D_j} las condiciones de disparo de una transición de entrada t_i (condición de activación) y de otra de salida t_j (condición de desactivación) del lugar p, podremos escribir:

$$S = C_A = \sum_i C_{Ai} \qquad \text{donde } {}^{\cdot}p = \{t_i\}$$
$$R = C_D = \sum_j C_{Dj} \qquad \text{donde } p^{\cdot} = \{t_j\}.$$

El interés de la *activación prioritaria* se pone de manifiesto en el caso de que para algún lugar se disparen «simultáneamente» una transición de entrada y otra de salida; en este caso el lugar debe permanecer marcado. Esta situación se ilustra, si $a = b$, mediante p_2 en la figura 6.3.

Observación sobre validación y realización. La RdP de la figura 6.3 con $a = b$ es un caso típico en el que un adecuado estudio de validación (capítulos 4 y 5), hubiera detectado la siguiente anomalía: la RdP con $a = b$ es binaria pero si fuese $a \neq b$, no lo sería puesto que p_1 y p_2 podrían contener hasta dos marcas. En ambos casos, la RdP es viva para el marcado inicial que exhibe.



(*a*) RdP

(*b*) Grafo de marcados (p_1 y p_2 pueden tener 2 marcas)

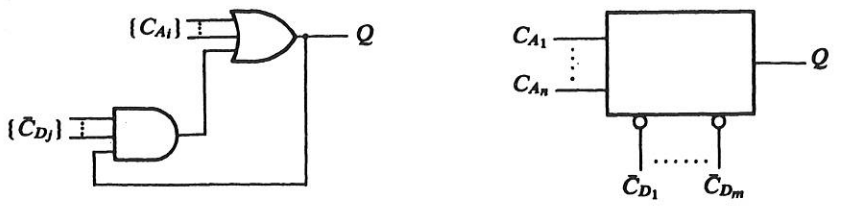
(*c*) Grafo de marcados cuando $a \equiv b$ (binario)

Figura 6.3 Si $a \equiv b$, p_2 se marca y desmarca simultáneamente.

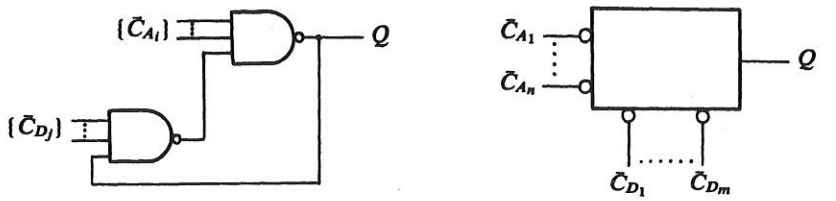
Además del argumento que, sobre el interés de la activación prioritaria en la célula, ofrecen redes como la de la figura 6.3 con $a = b$, más adelante se presentarán otros adicionales. En efecto, en §6.2.4 se argumentará en base a razonamientos *tecnológicos*, considerando los efectos de los retrasos en las conmutaciones de los estados lógicos de las puertas del circuito. Por otro lado, se ha de señalar que al estudiar otra técnica de realización que utiliza la misma célula (conexionado por *llamada-respuesta*, §6.3) se justificará también la necesidad de la activación prioritaria argumentando desde un punto de vista conceptual.

De acuerdo con lo anteriormente expresado, la célula deber ser un *biestable* R-S *generalizado* y *con activación prioritaria*. En la figura 6.4 se presentan tres realizaciones posibles de la misma. Como puede observarse, la única modificación introducida con respecto a los esquemas de la figura 6.2 son las sustituciones de R y S por sus valores:

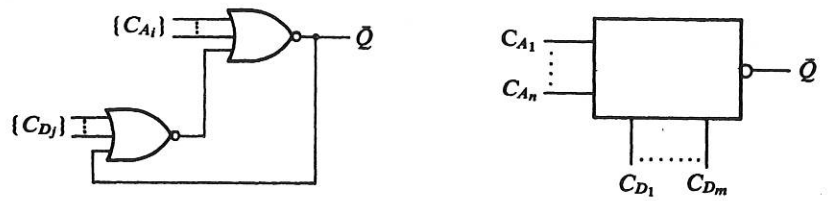
$$S = C_A = \sum_i C_{Ai} \qquad R = C_D = \sum_j C_{Dj}.$$



(*a*) O/Y (OR/AND)



(*b*) NO—Y/NO—Y (NAND/NAND)



(*c*) NO—O/NO—O (NOR/NOR)

Figura 6.4. Esquemas lógicos de una célula y representación simbólica.

Desde un punto de vista práctico, hay que señalar que bajo la denominación CUSA (*Cellule Universelle pour Séquences Asynchrones*) se encuentran disponibles dos células del tipo definido. Estas están realizadas en tecnología TTL y su esquema lógico es del tipo NAND-NAND [SESC 75]. La SFC 607 posee 6 condiciones de activación (denominadas entradas *primarias*) y 7 condiciones de desactivación (denominadas entradas *secundarias*). La SFC 608 (2 células en un *chip*) posee 3 condiciones de cada tipo. En cualquier caso, la célula puede ser realizada mediante un biestable R-S con activación prioritaria (figura 6.5), aunque conviene resaltar que la realización directa con puertas NAND es más económica en número de componentes y utiliza un idéntico número de conexiones (compárense los esquemas de las figuras 6.4*b* y 6.5). De este modo, el 7420 (doble puerta NAND de 4 entradas) permite realizar una célula con 3 condiciones de activación y otras 3 de desactivación.

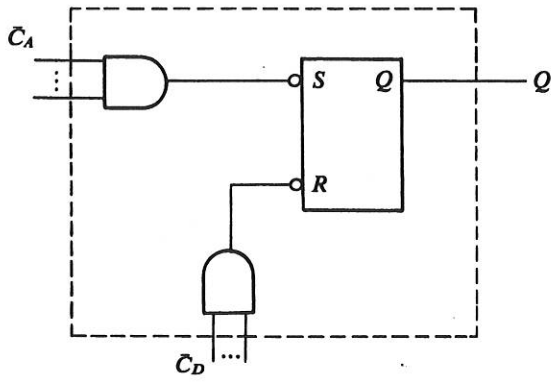


Figura 6.5. Una posible realización de la célula a partir de un biestable R-S construido con puertas NAND.

La utilización conjunta de un 7410 (triple puerta NAND con 3 entradas) y un 7430 (puerta NAND con 8 entradas) permite obtener una célula con 2 condiciones de activación y 2 de desactivación y otra con 2 (o 7) condiciones de activación y 7 (o 2) de desactivación.

6.2.2 Realización por transferencia impulsional

Sea Q_k la salida de la célula que realiza p_k, $Q_k = M(p_k)$. Para poder realizar una RdP quedan por definir las conexiones entre las diferentes células. Estas se realizan a través de la materialización de las condiciones de activación y de desactivación.

La *activación* de una célula se realizará al dispararse alguna de las transiciones de entrada del lugar que materializa. Considerando la figura 6.6, podremos escribir la condición lógica de activación de la célula j:

$$C_A^j = \sum_i C^j{}_{A_i} = \sum_i A_{ij} \prod_\alpha Q_{i\alpha}.$$

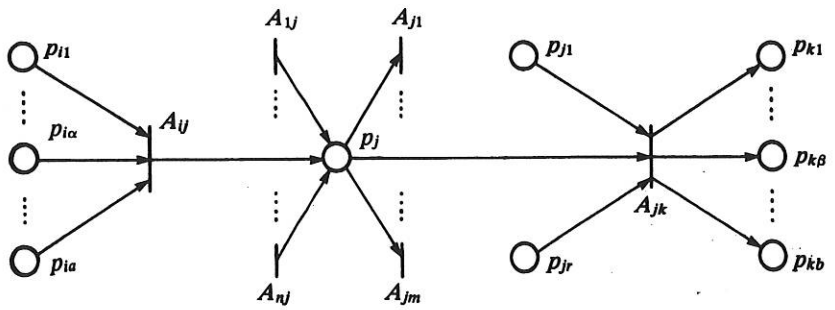
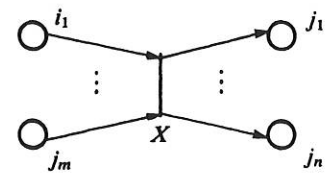


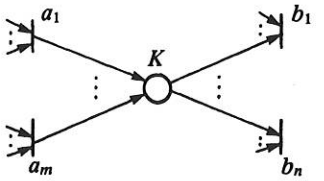
Figura 6.6. Configuración genérica alrededor de p_j (*Nota.* El marcado de p_j se representará por Q_j, salida de la célula asociada al lugar.)

La *desactivación* de una célula se realizará al dispararse alguna de las transiciones de salida del lugar que materializa. De acuerdo con la notación de la figura 6.6, tendremos que la condición lógica de desactivación de la j-ésima célula es la siguiente:

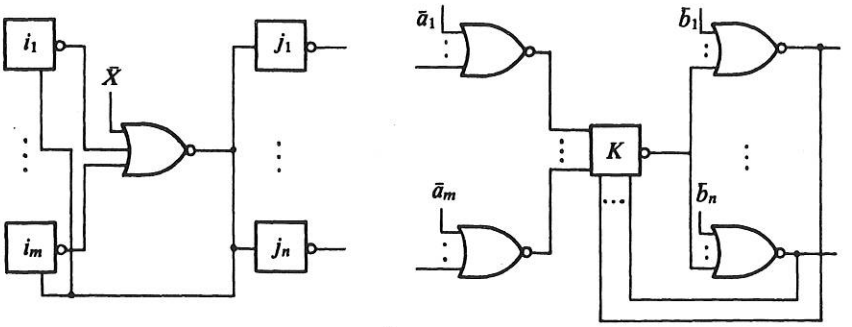
$$C_D^j = \sum_k C_{Dk}^j = \sum_k A_{jk} \prod_\rho Q_{j\rho}.$$



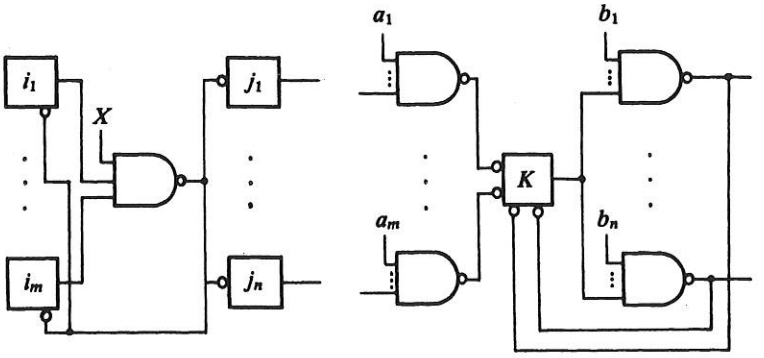
(*a*) *Nudo Y*



(*b*) *Nudo O*



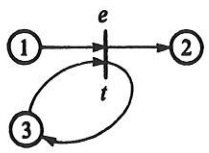
(*c*) Realizaciones utilizando la célula NOR/NOR



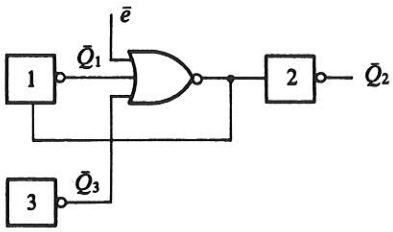
(*d*) Realizaciones utilizando la célula NAND-NAND.

Figura 6.7. Realizaciones de los nudos *O* e *Y* por transferencia impulsional. (Obsérvese que son duales.)

Cada término de C_A^j y C_D^j representa el disparo de una transición y, por lo tanto, es utilizable para construir las condiciones de activación o de desactivación de otras células. Por ejemplo, el término $A_{jk} \prod Q_{jp}$ sirve para *desactivar* las células asociadas a los r lugares de entrada de la transición etiquetada con A_{jk} y para *activar* las células asociadas a los b lugares de salida de la mencionada transición. A partir de esta consideración y de las dos ecuaciones anteriores se pueden establecer sin dificultad las reglas para realizar la conexión de las células. La figura 6.7 presenta dos construcciones locales típicas en RdP (*nudo O* y *nudo Y*) y dos cableados posibles para la realización de cada una de ellas. En el caso de la *lectura de un lugar*, el conexionado puede adoptar la forma simplificada que se muestra en la figura 6.8. En ésta se puede observar que la señal que representa el disparo de t no activa y desactiva simultáneamente al lugar p_3, puesto que su efecto sería inobservable.



(*a*) Lectura del marcado de p_3



(*b*) Cableado utilizando la célula NOR-NOR

Figura 6.8. Lectura de un lugar y esquema práctico de cableado.

Una vez definida la conexión de las células, la realización de las salidas es inmediata. La figura 6.9 presenta un ejemplo suficientemente ilustrativo. En éste sólo hay que señalar que si las especificaciones exigen una determinada duración de *PULSO*, se ha de añadir un monoestable ajustado al valor deseado. Para terminar los comentarios básicos sobre la realización, hemos de indicar que toda señal de inicialización general, I, debe activar las células asociadas a los lugares marcados inicialmente (p_1 en la RdP de la figura 6.9*a*) y debe desactivar las memorias asociadas a los lugares no marcados inicialmente. (*Nota.* La desactivación inicial no ha sido cableada en la figura 6.9*b*.)

6.2.3 Cuestiones adicionales de índole práctica

6.2.3.1 *Realización cuando el evento es un flanco*

En este caso, el método de realización expuesto no puede aplicarse directamente dado que éste es válido sólo con señales de nivel. Una solución indirecta consiste en modificar previamente la descripción del sistema que se desea realizar aplicando las reglas 3 o 4 de transformación de eventos (§2.3.3, figuras 2.15 y 2.16). Eliminados los eventos de tipo flanco de subida o bajada, la realización puede llevarse a cabo utilizando el método expuesto a lo largo del apartado 6.2.2.

En 6.4.5 se volverá sobre la realización cuando el evento es un flanco, pero utilizando biestables síncronos.

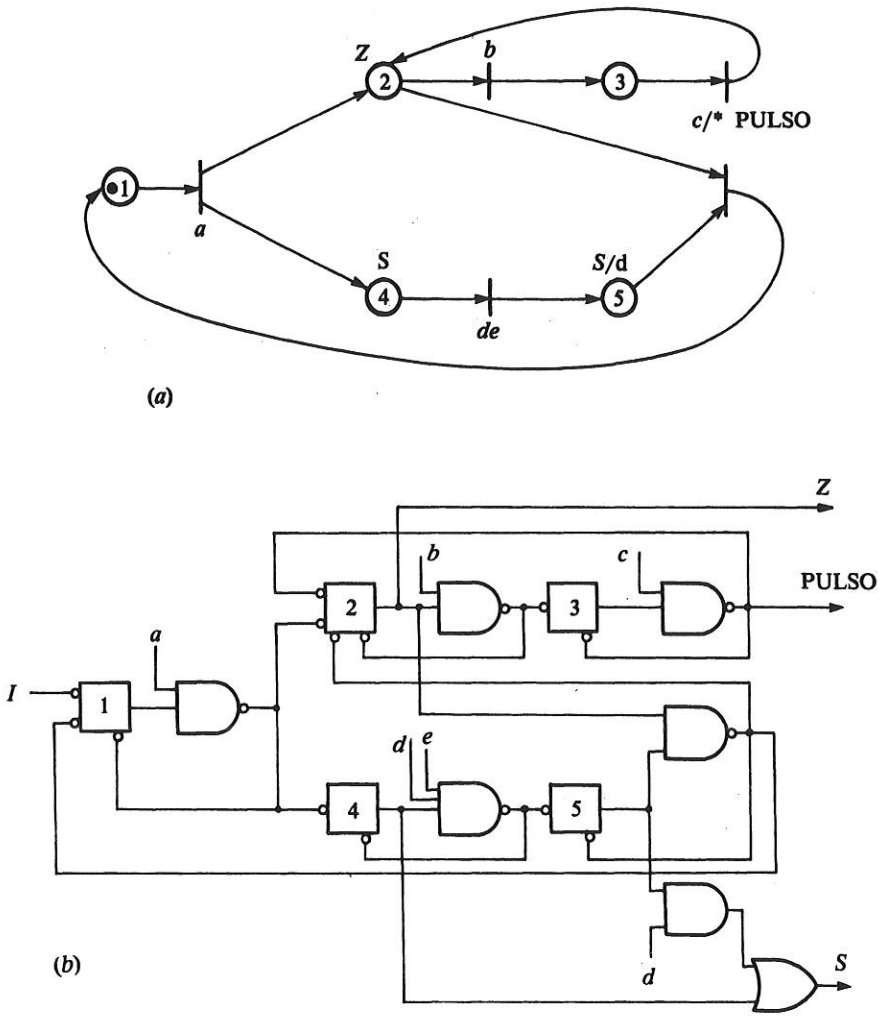


Figura 6.9. RdP y esquema de su realización por transferencia impulsional utilizando la célula NAND-NAND. (*Nota.* PULSO es una acción impulsional asociado al disparo de la transición que etiqueta.)

6.2.3.2 *Realización de una temporización*

La figura 6.10 presenta un esquema. El único punto que merece la pena considerar en éste es el retraso Δ_0 (realizable con puertas en serie). En efecto, el retraso Δ_0 debe

impedir el disparo de la transición de salida de A al activarse la célula y no haberse disparado aún el monoestable. Para Δ_0 se puede tomar como valor δ (figura 6.10*b*) o ligeramente superior.

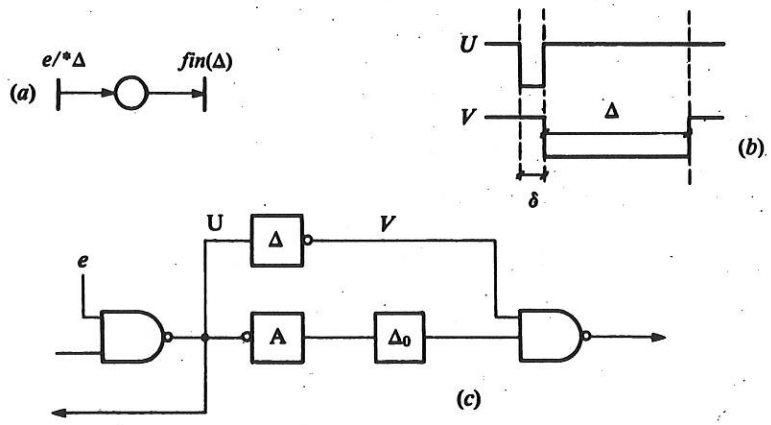


Figura 6.10. Temporización. (*Nota.* El pulso U dura δ unidades de tiempo.)

6.2.4 Análisis del comportamiento dinámico de los circuitos lógicos diseñados y estudio de fenómenos aleatorios

Los circuitos diseñados hasta ahora tienen un comportamiento lógico adecuado si se supone que todas las puertas pueden conmutar su estado instantáneamente.

Ahora bien, ninguna realización física de las puertas lógicas exhibe un comportamiento *dinámico* idéntico al de la puerta considerada como *elemento ideal*. En efecto, siempre existen retrasos en la conmutación del estado de la variable de salida con respecto al instante en que conmutaron las entradas. La figura 6.11 ilustra la obser-

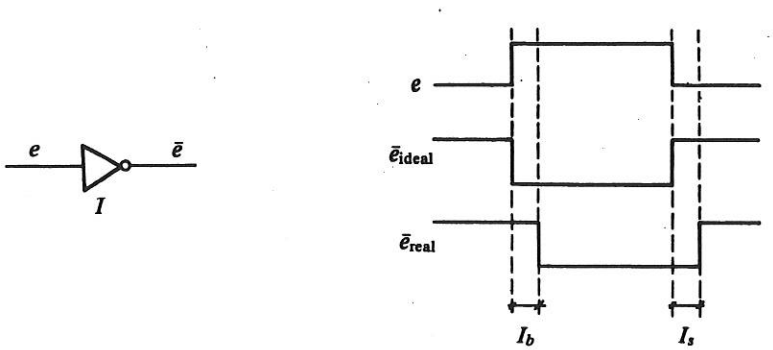


Figura 6.11. Todo dispositivo físico presenta retrasos a la conmutación.

vación anterior en un caso elemental: la puerta es un inversor. Con los subíndices s y b añadidos a la denominación de la puerta, I, se definen los retrasos a la *subida* y a la *bajada* de la variable de salida. *La presencia generalizada de estos retrasos constituye la principal limitación del álgebra de* BOOLE *en el análisis y síntesis de sistemas lógicos.*

El objetivo de este apartado es el de estudiar en que medida los retrasos en la conmutación de las puertas afectan al buen comportamiento de la realización de una RdP por transferencia impulsional.

Metodológicamente, procederemos, en primer lugar, estableciendo condiciones que garanticen que la simulación del disparo de una transición cumpla los tres puntos siguientes:

1) La activación de los lugares de salida.
2) La desactivación de los lugares de entrada.
3) La continuidad de una eventual salida asociada a dos lugares, uno de entrada y otro de salida de la transición.

Una vez considerados estos tres puntos, se impondrán ciertas restricciones sobre la duración mínima de los eventos y el solape máximo entre dos eventos complementarios, e y $\bar{e}$. Por último, se comentará brevemente el problema de los conflictos.

6.2.4.1 *Estudio de la simulación del disparo de una transición*

La figura 6.12*b* presenta un circuito lógico que realiza una transición con un lugar de entrada y otro de salida (figura 6.12*a*). El cronograma detalla la evolución provocada por la transición de e desde 0 a 1 (gráficamente se ha considerado que $A_s = B_s = T_s = T_b = D_s$). En éste se ha supuesto que e no vuelve a cero hasta que, por lo menos, no se haya terminado de realizar la evolución. Esta hipótesis simplificadora la matendremos a lo largo de todo este apartado. Como se ha anunciado, con posterioridad (§6.2.4.2) se calculará una duración mínima de e a 1 para que el disparo de una transición sea simulado correctamente.

A continuación establecemos con precisión las tres condiciones que, en general, permitirán afirmar si se ha simulado de forma adecuada el disparo de la transición:

1) La *activación* de la célula 2 estará asegurada siempre que la duración del pulso θ sea superior al tiempo necesario para que la segunda entrada de la puerta D se anule. Con ello se garantiza que Q_2 permanecerá a 1. Algebraicamente se puede escribir que si $A_s + B_b + T_s > D_s + C_b$, se activará la célula 2.
2) La *desactivación* de la célula 1 estará garantizada siempre. En efecto, la bajada de Q_1, es la que provoca la subida de θ, por lo cual se tendrá que, al menos una entrada de la puerta A estará a cero.
3) La *continuidad* de la salida S se cumple siempre que la activación de la salida de la célula 2 preceda a la desactivación de la salida de la célula 1. Algebraicamente, si $D_s < A_s + B_b$, se garantiza la continuidad de S.

Si, como es de esperar, el circuito se realiza utilizando componentes de una única tecnología y dado que las puertas que realizan las células y sus conexiones son del mismo tipo, se puede afirmar que aproximadamente $A_s = B_s = C_s = D_s = T_s = \delta$

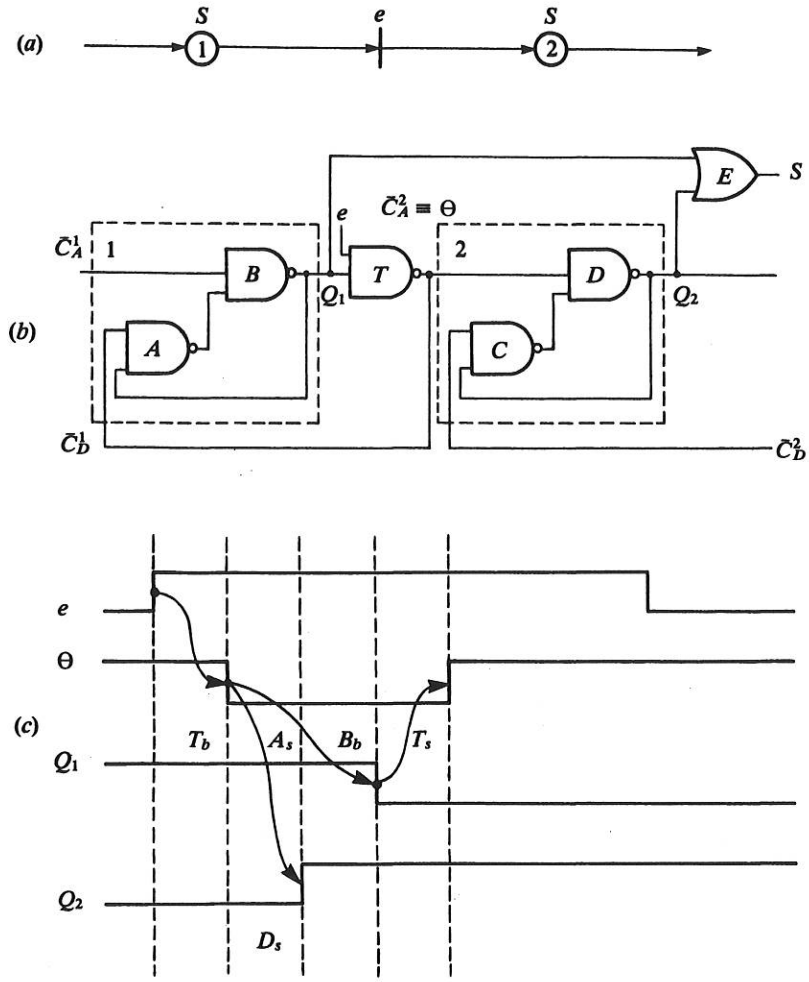


Figura 6.12. Realización y cronograma del disparo de una transición. (*Nota.* Subíndices: *s*, subida; *b*, bajada.)

y $A_b = B_b = C_b = D_b = T_b = \epsilon$. Simplificando las condiciones algebraicas anteriores se puede concluir que:

—la célula 2 se activa, dado que $\delta > 0$;
—la salida S se genera con continuidad, dado que $\epsilon > 0$.

En resumen, la transición se simula correctamente. Si una transición t posee varios lugares de entrada y varios lugares de salida ($^{\cdot}t = \{p_i\}, t^{\cdot} = \{p_j\}$), las células asociadas a éstos se desactivarán y activarán en paralelo. El disparo de la transición t durará $\delta_\theta = \min_i (A_{si} + B_{bi}) + T_s$ y se realizará correctamente si se cumple que el pulso θ dura más que:

1) la activación de la célula más lenta entre las asociadas a t':

$$\delta_\theta > \max_j (D_{sj} + C_{bj});$$

2) la desactivación de la célula más lenta entre las asociadas a $'t$:

$$\delta_\theta > \max_i (A_{si} + B_{bi}).$$

De acuerdo con las condiciones anteriores, normalmente se puede admitir el correcto desarrollo de la simulación del disparo de la transición t. Para «garantizarla» basta con incrementar δ_θ. Esto se puede hacer introduciendo un retraso en la salida (entre Q_1 y la puerta T) o en la realimentación desde Q_1 a la puerta A.

Nota. La célula denominada CUSA posee incorporados unos retrasos sobre la salida que permiten eliminar las aleatoriedades sobre la desactivación y activación de células y sobre la continuidad de las salidas [DAVI 80].

La utilización de un conexionado que conduzca a un funcionamiento por *llamada-respuesta* (§6.3) permite, por principio, garantizar la activación de las células asociadas a los lugares de salida de la transición; la desactivación de todas las células asociadas a los lugares de entrada de la transición no puede ser garantizada «a priori», salvo bajo ciertas condiciones (normalmente verificadas) que serán presentadas.

Observación muy importante. El solapamiento entre Q_1 y Q_2 es interesante cuando las acciones asociadas a los lugares son niveles; es decir, lo importante es su continuidad en la generación. Si por el contrario, como suele ocurrir en los sistemas digitales de los computadores y su periferia, interesa considerar las acciones como asociadas a las *activaciones* de los lugares, es importante que no exista el solapamiento en la acción generada. Una solución lógica inmediata a este problema consiste en inhibir la acción asociada a un lugar con la condición de disparo de la transición que marca el siguiente lugar al que se le asocia la misma acción. Así, en la figura 6.12*b* podríamos tener $S = Q_1\bar{C}_A^2 + Q_2$ en vez de $S = Q_1 + Q_2$.

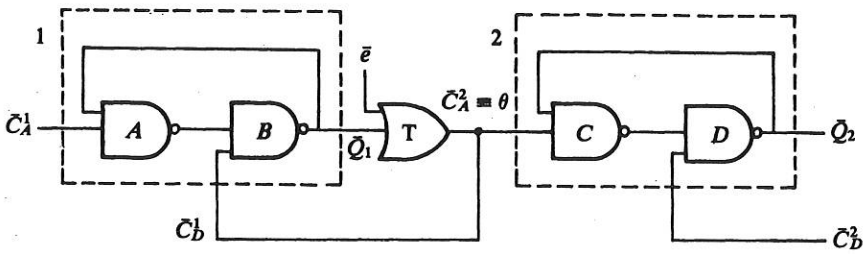


Figura 6.13. Realización de una transición utilizando una célula con desactivación prioritaria.

EJERCICIO. La figura 6.13 presenta otra realización lógicamente correcta del esquema de la figura 6.12*a*. En ésta se utilizan biestables R-S conexionados de forma que el comportamiento que exhiben es de desactivación prioritaria. Compruébese que:

1) Si $B_s + T_s > C_s + D_b$, se activará la célula 2.
2) Si $T_s > A_b$, se desactivará la célula 1.
3) No existe continuidad en la salida S (aleatoriedad en la salida).

Aceptando como válidos los resultados enunciados en el ejercicio anterior, se observará que el cableado de la figura 6.13 puede llegar a exhibir funcionamientos aleatorios del tipo no activar la célula 2 y, si se tiene $\epsilon \geqslant \delta$, también del tipo no desactivar la célula 1. En cualquier caso, no se asegura la continuidad de la salida S. Estas son las *razones de tipo tecnológico* que, como se anunció en §6.2.2, nos inclinan a preferir el conexionado que establece un comportamiento de activación prioritaria para los biestables R-S generalizados que definen la célula.

La técnica utilizada para analizar la simulación del disparo de una transición se puede emplear con otras realizaciones de la célula. No obstante, se puede afirmar que el análisis del comportamiento de los circuitos que utilizan células realizadas con dos puertas NOR arrojará resultados idénticos a los encontrados, puesto que serán circuitos *isomorfos* a los considerados (figura 6.7).

6.2.4.2 *Condiciones complementarias para una buena simulación*

Volvamos sobre el análisis del circuito lógico de la figura 6.12*b*. En este apartado vamos a considerar, como cuestiones complementarias, la duración mínima de e a 1, las aleatoriedades esenciales y los conflictos. El ejercicio 6.7 presenta una condición adicional.

a) *Duración mínima de* e a 1

Hasta ahora se ha venido considerando que la duración de e a 1 era suficientemente grande. Una cuestión que surge inmediatamente es determinar la *mínima duración* de e a 1 que permite una correcta simulación del disparo de la transición. Para responderla basta con observar que (figura 6.12) la mínima duración de θ a 0 que provoca:

—la activación de la célula 2 es: $D_s + C_b$;
—la desactivación de la célula 1 es: $A_s + B_b$.

Por consiguiente, la mínima duración del pulso θ a 0 tiene que ser:

$$\text{máx}\,(D_s + C_b, A_s + B_b) = \delta + \epsilon.$$

Por otro lado, la duración de θ a 0 se puede expresar en función de la duración de e a 1, E:

$$E - T_b + T_s = E - \epsilon + \delta.$$

En conclusión, la condición buscada está dada por $E - \epsilon + \delta > \delta + \epsilon \Rightarrow E > 2\epsilon$; es decir, e debe permanecer a 1 como mínimo *dos veces el retraso de la conmutación a* 0 *de una puerta*. Evidentemente, desde un punto de vista práctico, conviene considerar un valor mayor para compensar las posibles fluctuaciones en los valores de los δ y los ϵ.

b) Aleatoriedades esenciales†

Existe una *aleatoriedad esencial* en una descripción realizada con una RdP marcada receptiva a e, cuando difieren los marcados obtenidos al realizar uno o tres cambios consecutivos en el estado de e (véase la figura 6.14). Una aleatoriedad esencial existe debido a las especificaciones del sistema que se desea realizar. Es decir, la existencia de aleatoriedades esenciales es independiente de la técnica de realización que se vaya a emplear. Si existe tal aleatoriedad al realizar una RdP, el origen tecnológico de los funcionamientos anómalos que pueden sobrevenir, de acuerdo con la técnica de realización considerada (§6.2.2), es el solapamiento entre e y $\bar{e}$. En efecto, supongamos que a partir del marcado de la figura 6.14 se tiene físicamente $e\bar{e} = 1$ durante un lapso de tiempo significativo; en esas condiciones se producirá una errónea simulación del comportamiento de la RdP dado que se llegará a marcar el lugar k.

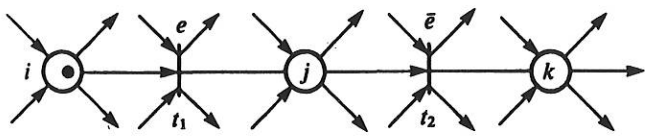


Figura 6.14. Aleatoriedad esencial.

La cuestión que se plantea ahora es saber si, considerando que el solape máximo entre e y $\bar{e}$ viene dado por I_b (figura 6.11), la técnica de realización de RdP binarias que se ha propuesto conduce a circuitos que pueden exhibir funcionamientos anómalos. La respuesta al problema es inmediata. El retraso a la activación de Q_2 (figura 6.12) es $T_b + D_s$. Como se puede admitir $T_b = I_b$, entonces $T_b + D_s > I_b$ y, en conclusión, el solapamiento admitido entre e y $\bar{e}$ no provocará una simulación incorrecta.

c) Conflictos

Considérese el caso en que dos transiciones están en conflicto efectivo (t_1 y t_2 en la figura 1.21; Π_1 y Π_2 en la figura 4.6) (Definición 4.10). Si X_1 y X_2 son dos eventos asociados a t_1 y t_2 respectivamente (figura 1.21), el conflicto se resuelve *lógicamente* haciendo que X_1 y X_2 se excluyan mutuamente. Ahora bien, si por razones tecnológicas existieran solapamientos entre X_1 y X_2 cuando el lugar estuviese marcado, se produciría un funcionamiento anómalo. En efecto, las células asociadas a los lugares de salida de t_1 y t_2 resultarían activadas simultáneamente y, por tanto, se violarían las reglas de evolución de las RdP.

El problema planteado anteriormente *no tiene una solución puramente lógica*. No obstante, puesto que los automatismos industriales evolucionan muy lentamente

† Las aleatoriedades esenciales generan una de las principales clases de problemas que plantea la síntesis de máquinas secuenciales por métodos clásicos [HILL 78] [UNGE 69]. Como se verá en este apartado, estas aleatoriedades no plantean mayor problema con las técnicas de realización presentadas.

con respecto a la dinámica del circuito lógico (segundos frente a nanosegundos si en la realización se emplea tecnología TTL), su influencia es muy escasa en aquéllos.

En los sistemas de muy alta velocidad de trabajo (fundamentalmente de tipo informático) su consideración debe ser imperativa. En este caso, los conflictos se resuelven (tratan de resolver) con unos dispositivos especiales denominados *árbitros*. (Véase [PLUM 73].) Un *árbitro* es un dispositivo diseñado específicamente para la resolución de conflictos tal que sus decisiones deben estar exentas de aleatoriedades. La principal aplicación de los árbitros se encuentra en los sistemas informáticos, en los que hay que resolver conflictos cuando varios usuarios pueden acceder a un mismo recurso.

6.3 REALIZACIÓN ASINCRONA (II): FUNCIONAMIENTO POR LLAMADA-RESPUESTA

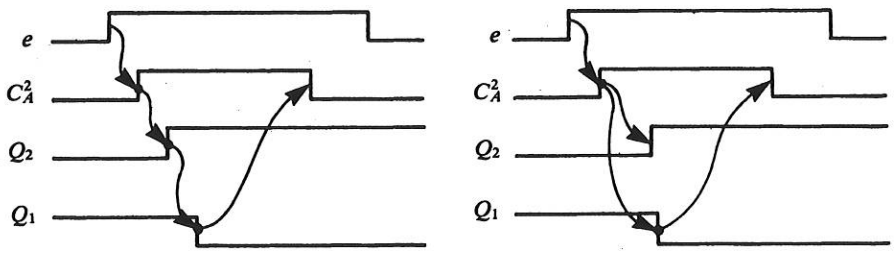
En este apartado presentamos otro tipo de realización de RdP que emplea la misma célula anterior (§6.2.1) pero utilizando una técnica de conexionado diferente lo que le confiere al circuito un modo de funcionamiento distinto.

La utilización de técnicas de conexionado entre células que conduzcan a un funcionamiento por *llamada/respuesta* tienen una gran aceptación industrial. A continuación (§6.3.1) presentamos el principio en que se basa dicho funcionamiento, así como esquemas básicos de cableado que permiten un tipo de realización simplificada. Por último (§6.3.2), se presenta una limitación del método simplificado de cableado y una condición que, si es cumplida por la RdP, permite utilizarlo sin dificultad.

6.3.1 Conceptos básicos

Desde un punto de vista conceptual, el funcionamiento por llamada-respuesta es tal que el disparo de una transición se realiza en dos fases perfectamente definidas:

1) *La llamada*, en la que se activan los lugares de salida de la transición disparada.
2) *La respuesta*, en la que la activación de los lugares de salida de la transición disparada posibilita la desactivación de los lugares de entrada de la misma.



(*a*) Funcionamiento por llamada-respuesta (ciclo cerrado $C_A\uparrow \Rightarrow Q_2\uparrow \Rightarrow Q_1\downarrow \Rightarrow C_A\downarrow$)

(*b*) Funcionamiento por transferencia impulsional (figura 6.12) ($Q_2\uparrow$ no se garantiza por el principio de funcionamiento)

Figura 6.15. Comparación de las relaciones causa-efecto en la simulación del disparo de una transición.

El funcionamiento por llamada-respuesta establece un ciclo *cerrado* de relaciones causa-efecto entre C_A^2, Q_2 y Q_1 en el proceso de simulación del disparo de una transición. La figura 6.15 compara conceptualmente el funcionamiento por llamada-respuesta con el que se obtiene por transferencia impulsional. En este último modo de funcionamiento, la activación de Q_2 se realiza, o no, independientemente del ciclo cerrado de relaciones causales establecido entre C_A^2 y Q_1.

En el funcionamiento por llamada-respuesta, la condición de activación de una célula sigue siendo la unión de las condiciones que simulan el disparo de las transiciones de entrada del lugar que representa (*llamadas*). Utilizando la nomenclatura de la figura 6.6, la condición general de desactivación de una célula (*respuesta*) se expresa de la forma siguiente:

$$C_D^j = \sum_k C_{Dk}^j \text{ donde } C_{Dk}^j = \left(\prod_\beta Q_{k\beta} \right) \cdot \left(A_{jk} \prod_\rho Q_{j\rho} \right).$$

Esto quiere decir que el lugar p_j se desactiva merced al disparo de la transición etiquetada con A_{jk} (condición C_D^j) si:

1) todas las células asociadas a los lugares de salida de la transición etiquetada con A_{jk} están activadas;
2) la transición etiquetada con A_{jk} es la disparada.

A primera vista, la segunda condición puede parecer redundante, por lo cual posteriormente justificaremos, de forma intuitiva, su necesidad (§6.3.2). Desde un punto de vista práctico se suele utilizar la condición de desactivación *simplificada* siguiente (sólo la condición 1 anterior):

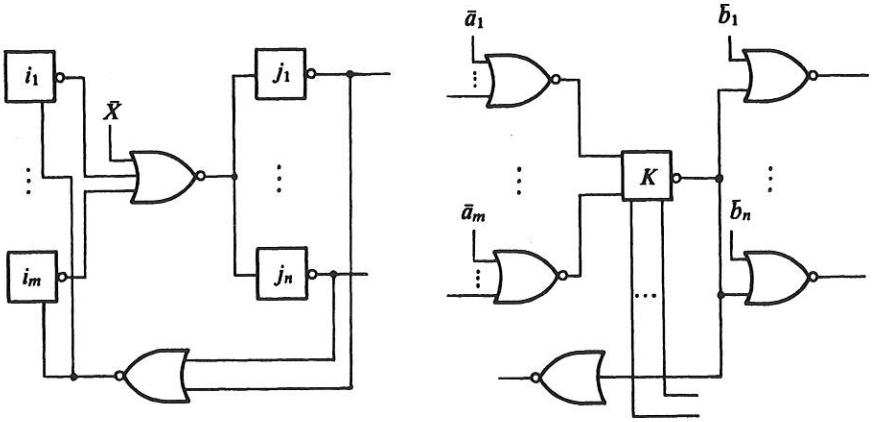
$$C_D^j = \sum_k \prod_\beta Q_{k\beta}.$$

Si el marcado de los lugares se mantiene durante lapsos de tiempo significativamente superiores a los de la simulación del disparo de una transición, con la condición simplificada de desactivación se puede garantizar la desactivación de todos los lugares de entrada de la transición disparada (la respuesta simplificada es un nivel). (Comprúebese.) Sin embargo, la condición completa de desactivación permanecerá a «1» sólo hasta que se desactive una célula de las asociadas a los lugares de entrada de la transición disparada (la respuesta es sólo un pulso). En este caso no se puede garantizar más que la desactivación de un lugar y no la de todos.

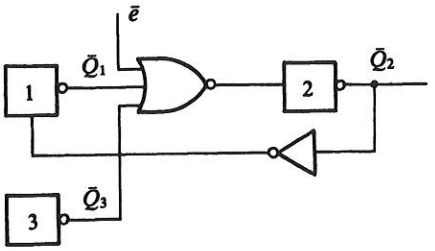
La *principal ventaja* de que el disparo de una transición se simule por llamada-respuesta reside en que si e (figura 6.12*a*) se mantiene a «1» durante un tiempo suficiente, la activación del (de los) lugar(es) de salida de la transición y la continuidad de la(s) salida(s) están garantizadas. Además, si la señal de respuesta representa la condición simplificada de desactivación y si los lugares permanecen marcados durante lapsos de tiempo importantes frente a la duración del disparo de una transición, la desactivación de todos los lugares de entrada de la transición estará también garantizada.

La figura 6.16 presenta esquemas de cableado básicos en los que se ha empleado la condición de desactivación simplificada. Como puede comprenderse fácilmente, la realización de la lectura de un lugar (figura 6.8*a*) *sólo* puede efectuarse de acuerdo

con un esquema como el de la figura 6.16*b* puesto que de lo contrario, si se cablearan las condiciones de activación y de desactivación de la célula 3 debidas a *t* (figura 6.8*a*), la actividad de la propia célula 3 terminaría desactivándola.



(*a*) **Realización de los nudos *Y* e *O* (figura 6.7*a*) utilizando la célula NOR-NOR.**



(*b*) **Realización de la lectura del marcado de p_3 (figura 6.8*a*) utilizando la célula NOR-NOR.**

Figura 6.16. Conexionado por llamada/respuesta que utiliza la condición de desactivación simplificada.

EJERCICIO. ¿Qué ocurriría al realizar por llamada-respuesta la red de la figura 6.17 si se utilizara un biestable con desactivación prioritaria? ¿Y si el biestable es de activación prioritaria (célula)?

EJERCICIO. La realización por llamada-respuesta necesita más puertas lógicas que la realización por transferencia-impulsional siempre que se utilicen las células realizadas sólo con puertas NOR o NAND. ¿Por qué? ¿Es válido el resultado anterior para cualquier tipo de realización de la célula?

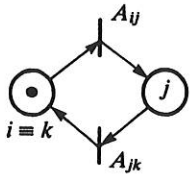


Figura 6.17. Ciclo de longitud 2.

6.3.2 Limitación del funcionamiento por llamada-respuesta cuando se utiliza la condición de desactivación simplificada

Considérese la RdP de la figura 6.18. Si se dispara la transición etiquetada b, se marcará p_4 y, por lo tanto, de acuerdo con las reglas simplificadas del funcionamiento por llamada-respuesta, se debe desactivar p_2, lo cual es incorrecto. En conclusión, *la condición simplificada de respuesta* (desactivación) *no se puede utilizar con cualquier red marcada viva y binaria*. Obsérvese que utilizando la condición de desactivación $C_D^j = \sum_k A_{jk} \prod_\beta Q_{k\beta}$ (notación de la figura 6.6) no se resuelve tampoco el problema, como se demuestra si tomamos $a = b$ en la figura 6.18. El origen del problema planteado está claro: la condición simplificada de respuesta desactiva indiscriminadamente a *todas* las células asociadas a los lugares de entrada de las transi-

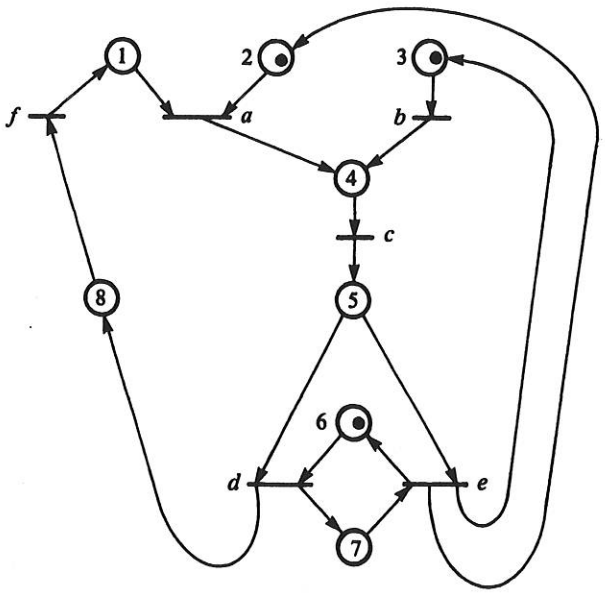


Figura 6.18. RdP conforme que no puede ser realizada por llamada-respuesta utilizando la condición simplificada de desactivación.

ciones de entrada de p_4 (transiciones etiquetadas *a* y *b*). A la condición simplificada de respuesta le falta información para definir *exclusivamente* la transición disparada y desactivar (responder) sólo las células asociadas a los lugares de entrada de la mencionada transición. Precisamente ésta es la información que, al definir la condición simplificada, se eliminó de la definición de la condición general de desactivación (§6.3.1).

La utilización de la condición simplificada no es posible en aquellas RdP en las que se pueda encontrar un marcado tal que *el conjunto de lugares de salida de una transición esté marcado simultáneamente con alguno de sus lugares de entrada*†. A partir de este enunciado, y empleando el concepto de componente conservativa (§4.7.2), se puede formular una condición que no necesita del cálculo de los marcados de la red:

> Todo lugar p_i que pertenezca a una componente conservativa monomarcada (de la forma $\sum_i M(p_i) = 1$) puede ser realizado utilizando la condición simplificada de desactivación.

En efecto, su demostración es trivial: toda componente conservativa en la que aparezca un lugar que pueda estar marcado simultáneamente con todos los lugares de salida de una de sus transiciones de salida, es forzosamente de la forma $\sum_j M(p_j) \geqslant 2$, de donde el resultado enunciado.

En la RdP de la figura 6.18 se tiene: $M(p_2) + M(p_3) + M(p_4) + M(p_5) + M(p_7) = 2$, luego es posible que se pueda producir una simulación errónea de la evolución del marcado de uno de esos lugares (ésta se produce para p_2).

La aplicación de la condición sobre componentes conservativas permite concluir afirmativamente sobre la utilización de la condición simplificada en la gran mayoría de las redes marcadas binarias de este texto (dado que éstas pueden ser, normalmente, descompuestas en un conjunto de grafos de estado monomarcados).

Observaciones:

1) En la red de la figura 6.18, p_2 es implícito. Su supresión elimina el problema considerado.
2) Si a la red de la misma figura se le añade un lugar implícito, p_9, definido entre las transiciones *a* y *e* ($^{\cdot}p_9 = \{a\}$, $p_9^{\cdot} = \{e\}$) y desmarcado inicialmente, ¡la RdP obtenida (la red de la figura 3.5) es realizable correctamente utilizando la condición de desactivación simplificada!

6.3.3 A modo de resumen

La simulación de RdP binarias basada en un funcionamiento por llamada-respuesta emplea la célula con activación prioritaria (§6.2.1). Además, utiliza la misma condición de activación que el conexionado que conduce a un funcionamiento por transferencia impulsional. Por consiguiente, las dos cuestiones adicionales de índole práctica (§6.2.3) así como las condiciones complementarias para una buena simulación (§6.2.4.2), tienen total vigencia. De la misma forma, toda señal de inicializa-

† Afortunadamente, en la práctica esta condición no se verifica casi nunca, lo cual permite comprender en parte la amplia difusión de las técnicas simplificadas de llamada-respuesta en ámbito industrial.

ción general debe activar las células asociadas a los lugares marcados inicialmente y debe desactivar las células asociadas a los restantes lugares. Las diferencias entre los conexionados que conducen a un funcionamiento por transferencia impulsional o por llamada-respuesta estriba en la condición de desactivación que se emplee. El segundo método conduce a una simulación más «robusta» (puesto que se establece un ciclo cerrado de relaciones causa-efecto entre las señales que intervienen en la simulación del disparo transición) pero más lenta y en la que el solapamiento entre Q_2 y Q_1 es más importante (véase el ejercicio 6.6).

A partir de la condición general de desactivación, se ha introducido una condición simplificada y se han caracterizado RdP binarias en las que se puede utilizar sin dificultad. En este sentido, es importante señalar que la generalización a RdP no binarias del conexionado que conduce a un funcionamiento por llamada-respuesta es muy complicada (dado que debe detectarse el incremento de valor de todos los contadores asociados a los lugares de salida de la transición disparada). Por último, si se puede utilizar la condición de desactivación simplificada y si el marcado de los lugares dura bastante más que la simulación del disparo de una transición, la simulación será correcta.

6.4 REALIZACIÓN SINCRONA

Las técnicas que se presentan en este apartado utilizan los principios básicos enunciados anteriormente:

1) Realización *modular*. La célula tendrá la misma definición funcional básica (R-S *generalizado con activación prioritaria*).
2) Conexionado de células de acuerdo con los métodos de *transferencia impulsional* o de *llamada-respuesta*.

A modo de observación preliminar, conviene destacar que como concepto, las RdP están orientadas hacia la construcción de modelos *asíncronos*†, aunque evidentemente se puede asociar una interpretación síncrona a su evolución.

Las técnicas utilizables para el conexionado de células pueden ser las presentadas anteriormente en los apartados 6.2 y 6.3: *transferencia impulsional* y *llamada-respuesta*. No obstante, hay que señalar que el conexionado que conduce a un funcionamiento por llamada-respuesta no tiene interés puesto que no aporta nada a la eliminación de aleatoriedades; además, durante todo un período de reloj habrá solapamiento en la actividad de las células asociadas a los lugares de entrada y de salida de una transición disparada.

El *diseño de células síncronas* (en definitiva biestables síncronos) se va a realizar a partir de biestables síncronos que se encuentran disponibles en el mercado. Es decir, el diseño se reducirá al problema clásico de la transformación de un biestable dado en un biestable de otro tipo, y consiste en determinar la *lógica de transformación* (figura 6.19).

---

† En su acepción comportamental, de evolución no controlada.

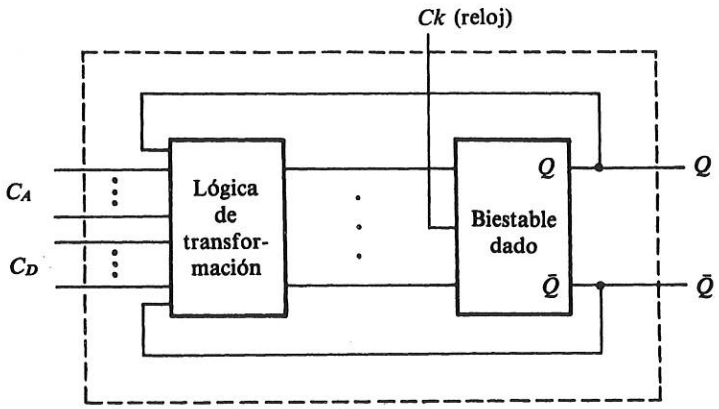


Figura 6.19. Modelo para la transformación de un biestable síncrono en otro.

Representación gráfica	Tabla de estados (resumida)	Ecuación característica
Ck *J* *Q* *K* $\bar{Q}$	J K | Q^+ 0 0 | Q 0 1 | 0 1 1 | $\bar{Q}$ 1 0 | 1	$Q^+ = Q\bar{J}\bar{K} + 0\bar{J}\bar{K} + \bar{Q}JK + 1J\bar{K} = \bar{Q}J + Q\bar{K}$
Ck *Q* *D* $\bar{Q}$	D | Q^+ 0 | 0 1 | 1	$Q^+ = 0\bar{D} + 1D = D$
Ck *Q* *T* $\bar{Q}$	T | Q^+ 0 | Q 1 | $\bar{Q}$	$Q^+ = Q\bar{T} + \bar{Q}T$

Tabla 6.1 Definición y representación de los biestables *J-K*, *D* y *T*. (*Notas.* (1) Q y Q^+ representan el estado interno actual y en el siguiente período, respectivamente; (2) el *preset* y el *clear* no se han considerado.)

En la tabla 6.1 se definen los biestables síncronos J-K, D y T. En la misma se indica el cálculo de la *ecuación característica* de cada biestable a partir de su tabla de estados resumida.

EJERCICIO. Compruébese que los biestables D y T pueden obtenerse a partir del biestable J-K haciendo $J = \bar{K} = D$ y $J = K = T$, respectivamente.

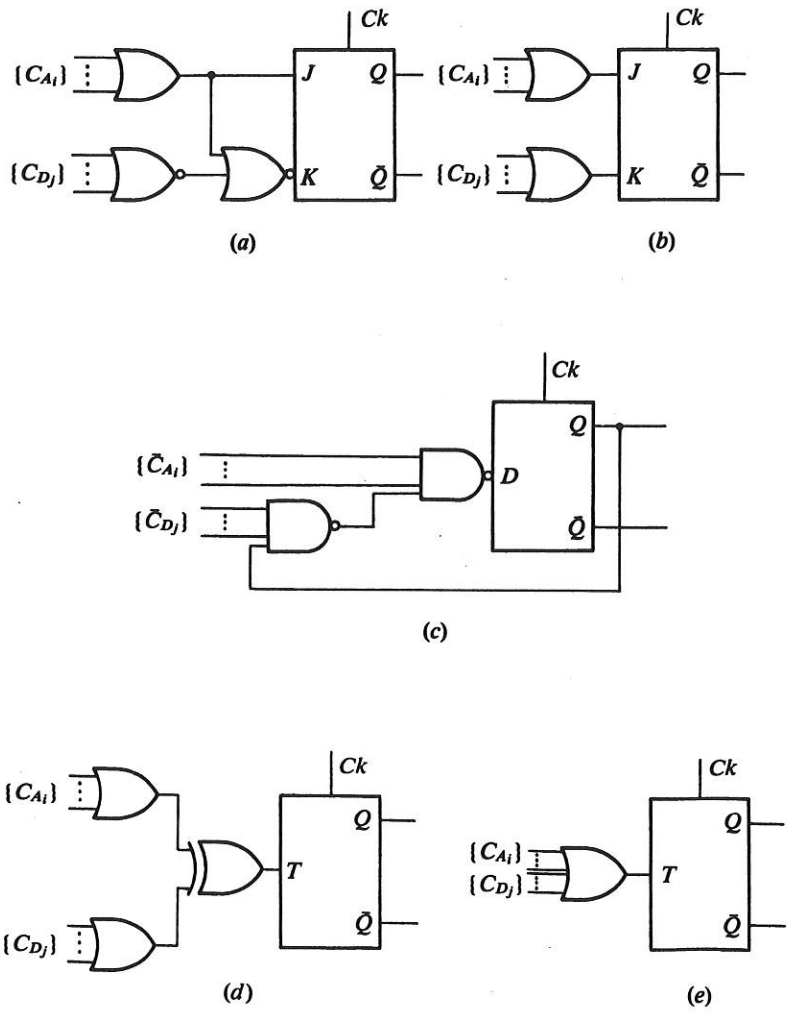


Figura 6.21. Diversas realizaciones de la célula síncrona *con* (casos *a*, *c* y *d*) y *sin* (casos *b* y *e*) activación prioritaria.

6.4.1 Síntesis utilizando el biestable J-K

Ésta se puede realizar de forma inmediata, dado que el biestable J-K tiene un comportamiento idéntico al del biestable R-S (§6.2.1) salvo para $J \equiv K = 1$. Por consiguiente, basta con hacer (figura 6.20*a*):

$$J = C_A \text{ y } K = \bar{C}_A C_D.$$

Si $C_A C_D \equiv 0$ (C_A y C_D nunca se verifican simultáneamente), $K = \bar{C}_A C_D + C_A C_D = C_D$ (figura 6.20*b*). Es decir, si no se presentan simultáneamente la activación y la desactivación, es innecesaria la condición sobre la activación prioritaria.

Desde un punto de vista práctico, si $C_A C_D \equiv 0$, se pueden utilizar como módulos (célula + lógica de elaboración de las condiciones de activación) las denominaciones 74H71 (J-K *maestro/esclavo con preset*) y 74101 (J-K *disparable por flanco negativo con preset*). En ambos casos se tiene $C_A = J = J_{11}J_{12} + J_{21}J_{22}$ y $C_D = K = K_{11}K_{12} + K_{21}K_{22}$.

6.4.2 Síntesis utilizando el biestable D

La ecuación característica del biestable R-S con activación prioritaria es $Q^+ = S + \bar{R}Q$. Igualando ésta con la del biestable D se determina inmediatamente: $D = S + \bar{R}Q$. Sustituyendo S por C_A y R por C_D se obtiene la célula: $D = C_A + \bar{C}_D Q$ (la figura 6.20*c* ilustra una realización).

EJERCICIO. Compruébese que la célula obtenida no se puede simplificar más, aunque no se exija la activación prioritaria (por cumplirse $C_A C_D = 0$).

La única posibilidad de simplificar la célula se da cuando $\bar{C}_D Q = 0$. En este caso, la elaboración de D se simplifica enormemente: $D = C_A$. ¿En qué condiciones se tiene $\bar{C}_D Q \equiv 0$ y, por lo tanto, se puede simplificar la realización? Dado que partimos del estado de activación de la célula, $Q = 1$, la única solución posible es que $\bar{C}_D = 0 \Rightarrow C_D = \sum_j C_{D_j} = 1$; es decir, cuando al estar activa la célula, la unión de las condiciones de desactivación sea cierta. Por consiguiente, cada activación de la célula durará un único ciclo de reloj.

Esta simplificación, que en un principio parece tener poco interés práctico, en realidad si lo tiene. En particular, es muy importante detectar el caso en que p, lugar que se realiza, es el único lugar de entrada de sus transiciones de salida ($p^\cdot = \{t_k\}$ y $\forall k\ {}^\cdot t_k = \{p\}$). En este caso $C_D = Q\sum_k A_k = \sum_k A_k = 1$ ($Q = 1$ habida cuenta que la célula estaba activa); es decir, basta con que la unión de los eventos (condiciones) asociados a sus transiciones de salida sea cierta. Considerando la RdP de la figura 5.11, los lugares $p_2(X + \bar{X} = 1)$, $p_3(Y + \bar{Y} = 1)$ y p_4 (ausencia de evento asociado a su transición de salida) cumplen la condición enunciada, por lo que su realización no necesita más que *células simplificadas*: sólo biestables de tipo D más la lógica de elaboración de las condiciones de activación.

A modo de comentario final, queremos insistir sobre la importancia de la simplificación expuesta cuya utilidad cubre un amplísimo espectro de aplicaciones (secuenciadores de computadores, etc.).

EJERCICIO. Realícese la RdP de la figura 5.11.

6.4.3 Síntesis utilizando el biestable *T*

Para obtener una realización económica de la célula, hay que emplear un razonamiento ligeramente más sutil que los anteriores. La entrada del biestable T debe valer 1 sólo cuando se conmute el estado de la célula. La tabla de KARNAUGH siguiente (tabla 6.2) define T en función de C_A, C_D y Q, considerando exclusivamente un conexionado que conduzca a un funcionamiento por transferencia impulsional.

		C_AC_D			
	T	00	01	11	10
Q	0	0	∅	∅	1
	1	0	1	0	∅

a) $T\ (\emptyset, 1, 0) = \emptyset$, porque no se puede tener $C_D = 1$ si la célula está desactivada, $Q = 0$.
b) $T\ (1, 0, 1) = \emptyset$, porque la RdP es binaria.
c) $T\ (1, 0, 0) = 1$, es la activación de la célula.
d) $T\ (0, 1, 1) = 1$, es la desactivación de la célula.

Tabla 6.2. Definición de $T(C_A, C_D, Q)$.

Considerando la anterior tabla se tiene: $T = C_A \oplus C_D$ (figura 6.20*d*). Si $C_AC_D \equiv 0$, no ha lugar considerar la activación prioritaria. En este caso se puede utilizar el término C_AC_D para simplificar la realización de T: $T' = (C_A \oplus C_D) + C_AC_D = C_A + C_D$ (figura 6.20*e*).

6.4.4 A modo de resumen

Una rápida observación de la figura 6.20 pone de manifiesto que la realización de la célula con *biestables síncronos* es más simple si no se impone la activación prioritaria†. La ausencia de ésta exigencia supone que $C_AC_D \equiv 0$ (C_A y C_D nunca se verifican simultáneamente), lo cual se cumple para sistemas síncronos si:

1) Sólo se emplea el conexionado de células que conduce al funcionamiento por transferencia impulsional. Con ello se impide el intentar desmarcar un lugar no marcado; es decir, $\bar{Q}C_D = 0$ en los instantes definidos por el reloj.
2) La RdP marcada y sin interpretación (la red como grafo) es binaria. En efecto, la condición (1) impide tener $\bar{Q}C_AC_D = 1$. Por otro lado, si la RdP marcada es binaria, no se puede tener $QC_AC_D = 1$ porque al disparar la transición de entrada antes que la de salida se tendrían dos marcas en un lugar, lo que va contra la hipótesis de binariedad.

Para evitar aleatoriedades si las variables de entrada no están naturalmente sincronizadas con el reloj, se debe proceder a su sincronización. En su forma más simple, ésta se puede hacer enviando al circuito la salida de un biestable D alimentado por la variable externa. En [HILL 78] se estudia en detalle un sincronizador más elaborado.

---

† Recuérdese (§6.2.1) que en función de la definición de las variables de entrada y de salida, *una misma* célula asíncrona puede exhibir un comportamiento de activación prioritaria o de desactivación prioritaria.

6.4.5 Realización asíncrona cuando el evento es un flanco

En este apartado se sugiere una solución alternativa a la considerada en §6.2.3.1, que se basa en el empleo de un biestable síncrono activable por flancos. Las ideas fundamentales son:

1) La variable o función que define el flanco entra a la célula como reloj.
2) El biestable debe memorizar la condición de activación, eliminada la variable o función que define el flanco.
3) La desactivación de la célula se realizará a través del borrado (*clear*) del biestable utilizando la condición simplificada que conduce a un funcionamiento por llamada-respuesta.

EJEMPLO. Si en la figura 6.12*a* sustituimos *e* por $a\uparrow b$ y se emplea un biestable J-K disparable por flanco de subida, tendremos:

$$Ck = a; \qquad J = Q_1 b; \qquad K = 0; \qquad clear = Q_3$$

EJERCICIO. Compruébese que si utilizamos un biestable D, se deberá tener $D = Q_1 b + Q_2$, $Ck = a$, $clear = Q_3$.

6.5 CUESTIONES ADICIONALES SOBRE REALIZACIÓN CABLEADA

6.5.1 Las acciones (salidas)

Desde un punto de vista industrial, interesa disponer de una célula (módulo) que permita la fácil integración de ciertas seguridades funcionales o/y de los modos de marcha del sistema. De este modo han sido propuestas, por ejemplo, las células de acción [TELE 77]. Una *célula de acción* facilita el control de los accionadores ofreciendo una lógica diseñada específicamente (figura 6.21).

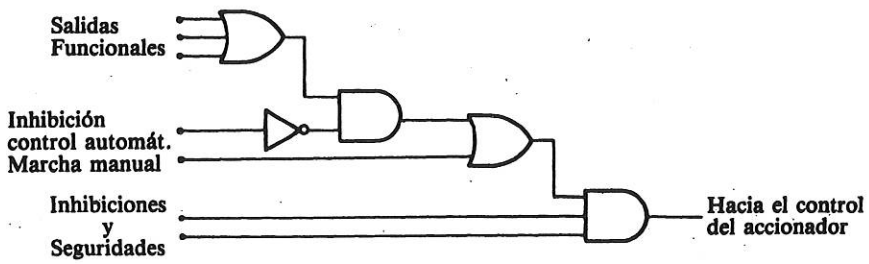


Figura 6.21. Célula de acción

6.5.2 Interés de la simplificación de la descripción

La simplificación de una RdP fué abordada en el capítulo 3. A continuación vamos a presentar brevemente algunas de las consideraciones, que, después de haber de-

cidido proceder a una realización cableada, se deben tener en cuenta al simplificar una RdP.

a) ¿Merece la pena simplificar el lugar implícito, p_i, de la figura 6.22? Obsérvese que en la RdP simplificada, para realizar la salida S_i hace falta la unión lógica de m señales y, por lo tanto, como mínimo otras tantas conexiones. Es decir, la eliminación del lugar implícito puede conducir a una realización cableada más compleja.

b) La obtención de transiciones fuente puede conducir a eventos del tipo de cambio de estado (flanco de subida o de bajada $e\uparrow$ o $e\downarrow$), lo cual suele complicar la realización final.

c) Los circuitos combinatorios que generan las salidas tienen que estar diseñados de forma que no presenten aleatoriedades. Puede suceder que la consideración de simplificaciones del tipo fusión de lugares, llegue a no ser interesante en ciertos casos.

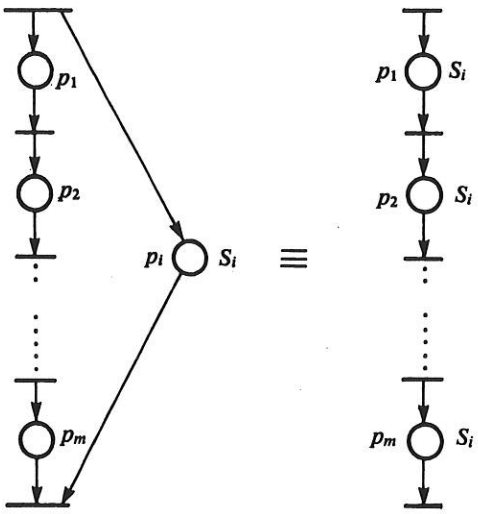
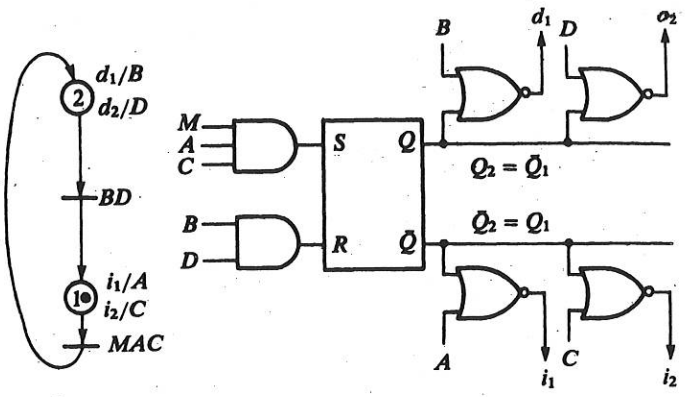


Figura 6.22. Lugar implícito: interés de su eliminación si la realización es cableada-modular.

En conclusión, volvemos sobre la idea de base expresada en el capítulo dedicado a la simplificación. Dada la ausencia de criterios bien definidos, en cada caso el diseñador debe evaluar la conveniencia de simplificar o no, una descripción. En cualquier caso, es fundamental que, para tomar las decisiones pertinentes durante el proceso de simplificación, se tenga muy presente el método de realización que se va a emplear.

Para evitar que se concluya demasiado rápida o drásticamente sobre la relativa inutilidad de la simplificación de cara a una realización, reconsideremos una vez más el ejemplo de los dos carros que van y vienen (figuras 1.9 y 1.15*a*). Si el lector trabajó el ejercicio 3.2, observaría que la RdP de la figura 1.15*a* (8 lugares y 6 transi-



(*a*) RdP

(*b*) Realización

Figura 6.23. Descripción simplificada y realización del sistema de dos carros que van y vienen.

ciones) se podría reducir a un GR con 2 lugares y dos transiciones (figura 6.23*a*). La realización modular de la red de la figura 1.15*a* necesita 8 células y 6 puertas externas. La realización modular de la red de la figura 6.23*a* necesita 2 células y 6 puertas externas. (Compruébese.) Por otro lado, si $Q_1(Q_2)$ representa la salida de la célula asociada a $p_1(p_2)$, dado que $M(p_1) + M(p_2) = 1$, se tendrá $Q_1 = \bar{Q}_2$. Dicho de otro modo, el sistema se puede realizar con una única célula (o biestable R-S) y 6 puertas lógicas (figura 6.23*b*). En definitiva, con respecto a la realización inicial se ahorrarán 7 células y, lo que es también muy importante, 21 conexiones. (Compruébese.) Por último, es interesante señalar que el circuito final se puede realizar con sólo dos integrados:

1. Un cuádruple de puertas NOR de dos entradas ($S_i = \bar{E}_{i_1}\bar{E}_{i_2}$), denominación 7402 (TTL) o 4001 (CMOS) (o similar).
2. *a*) En versión *síncrona*: un biestable R-S o J-K precedido por puertas AND. (Obsérvese que la sustitución del biestable R-S por el J-K es válida puesto que $AB = 0$ y $CD = 0$, lo que implica $JK = (MAC)(BD) = 0$.) Se pueden utilizar las denominaciones 7470 ó 74L71 (TTL) ó 4095 (CMOS).
   b) En versión *asíncrona:* un doble AND-NOR (denominación 74L51 o similar) conexionado como indica la figura 6.24.

6.5.3 Realizaciones autoverificables («*self-checking*»)

La protección contra las averías en los sistemas lógicos se basa en el empleo de redundancias. Estas pueden ser de dos tipos:

—*redundancias funcionales* (a nivel de la descripción funcional),
—*redundancias estructurales* (a nivel de la realización).

Este comentario lo vamos a centrar en las últimas. Con él se pretende unicamente dar una orientación sobre tan importante tema.

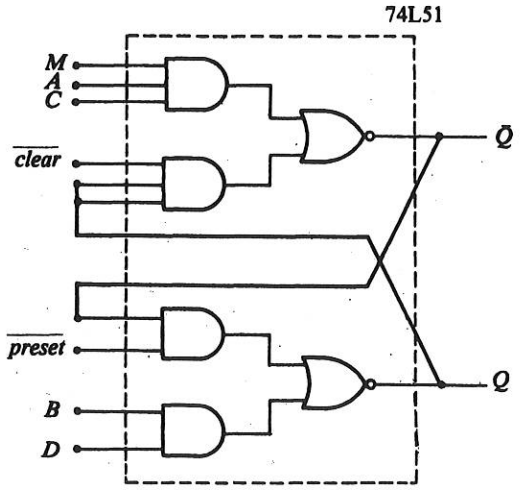


Figura 6.24. Módulo asíncrono con una condición de activación ($C_A = MAC$) y otra de desactivación ($C_D = Bd$). (*Nota. Preset* y *clear* permiten forzar el marcado inicial cuando $C_A = C_D = 0$.)

De forma intuitiva, un sistema autoverificable es un sistema capaz de detectar su mal funcionamiento cuando éste es ocasionado por una avería perteneciente a un conjunto predeterminado.

Para realizar un sistema autoverificable se pueden considerar dos tipos de redundancias estructurales:

1) *Redundancias separables*, en las que la detección de la falta se obtiene al añadir al bloque en funcionamiento normal, otro(s) bloque(s) que trabaja(n) paralelamente. Las salidas (y eventualmente estados internos) son comparados posteriormente mediante un circuito de votación o comparación.

2) *Redundancias no separables*, en las que se utiliza una codificación redundante de la información (por ejemplo, una palabra perteneciente al código 1 entre n). La detección de las averías se realiza cuando la información no pertenece al código preestablecido (por ejemplo, si se obtiene una palabra perteneciente al código 2 entre n).

Una vez hechas estas consideraciones preliminares, vamos a presentar algunas ideas básicas en la realización autoverificable de RdP con redundancia no separable. Se supone que las averías que puedan sobrevenir se traducen finalmente en la aparición o desaparición intempestiva de marcas. Ello conduce a considerar la codificación del marcado.

6.5.3.1 *Método de ponderación* [MARI 75]

Sea $\boldsymbol{R}$ una RdP conservativa (§4.6); es decir, existe un vector $Y \in \mathbb{N}^n$ denominado *vector de pesos* tal que $Y^T \cdot M_k = Y^T \cdot M_0 = A$. El entero A se denomina *peso activo* de la RdP marcada $\langle \boldsymbol{R}, M_0 \rangle$.

Se define el *peso total* de la RdP, π, como la suma ponderada siguiente:

$$\pi = \sum_{i=1}^{n} Y(p_i) \max_k (M_k(p_i)),$$

es decir, se pondera el peso de cada lugar por el máximo número de marcas que puede contener a partir del marcado inicial. Si la RdP es conforme, entonces π será simplemente la suma de los pesos de todos los lugares.

A partir de los conceptos expuestos, el método de autoverificación denominado de ponderación se basa en la construcción de un código «A entre π». Si en un momento determinado, el circuito de verificación detecta que el circuito que realiza la RdP envía un peso activo diferente de A, entonces genera una señal que indica la existencia de un fallo.

6.5.3.2 *Utilización de códigos de* HAMMING [SIFA 78]

Se transforma una RdP marcada $\langle \boldsymbol{R}, M_0 \rangle$ en una RdP equivalente $\langle \boldsymbol{R}', M_0' \rangle$ añadiendo lugares implícitos. $\langle \boldsymbol{R}', M_0' \rangle$ es tal que $\boldsymbol{M}(\boldsymbol{R}', M_0')$ conjunto de marcados sucesores de M_0', posee una *distancia* de HAMMING dada: d. En estas condiciones, es posible detectar todos los errores de multiplicidad $d - 1$ y corregir todos los de multiplicidad $(d - 1)/2$ (resultados clásicos en la teoría de códigos detectores y correctores de error). Un error de multiplicidad μ corresponde a la modificación de un marcado correcto en otro incorrecto de forma que difieran en μ componentes. Un error simple ($\mu = 1$) traduce la generación o la desaparición imprevisible de marcas en un lugar de la RdP.

Tanto este método como el anterior se basan en la existencia de componentes conservativas en la RdP.

Por último debemos plantearnos una pregunta básica: ¿en qué medida es válido aceptar la hipótesis de que los fallos se traducen por la generación o la destrucción intempestiva de marcas? Esta es una cuestión muy difícil de responder. No obstante, es de resaltar que la decisión de realizar los sistemas «calcando» la descripción funcional permite dar mayor grado de verosimilitud a las hipótesis de manifestación de averías establecidas (generación y desaparición de marcas).

6.6 CONCLUSIONES

El hilo conductor sobre el que se ha desarrollado este capítulo ha sido la obtención de un circuito lógico isomorfo a la RdP que describe el comportamiento funcional del sistema. Es decir, el sistema es realizado tal y como es descrito (eventualmente utilizando una descripción simplificada). Las motivaciones por las que se adoptó este criterio de realización han sido comentadas ampliamente: fácil comprensibilidad, modificabilidad, etc. Además en §6.5.3 se han sugerido conceptos y técnicas para abordar realizaciones autoverificables o autocorrectoras.

A lo largo del capítulo, hemos presentado técnicas de síntesis asíncrona y síncrona, mostrándose como los dispositivos definidos (células, etc.) podrían ser realizados con circuitos de fácil adquisición. La conexión simplificada que conduce a un

funcionamiento por llamada/respuesta plantea problemas con ciertas redes; éstos pueden ser eliminados utilizando la conexión por transferencia impulsional (que sigue la definición de la evolución del marcado de las RdP) o la conexión completa por llamada/respuesta. Esta última realiza el disparo de una transición en base a un ciclo cerrado de relaciones causa-efecto. Uno de sus mayores inconvenientes es su difícil empleo con RdP no binarias.

La crítica mayor que se puede hacer a la modularidad en la realización cableada puede provenir del número de circuitos integrados y conexiones a realizar cuando la RdP tiene muchos lugares (a título indicativo digamos que del orden de una docena), puesto que el coste material crece linealmente con la complejidad de la red. Por otro lado, la fiabilidad del equipo decrece con el aumento del número de circuitos y conexiones.

Evidentemente, estas críticas tendrán tanto más sentido cuanto: (1) mayor sea la serie de ejemplares que se desea realizar y (2) menores sean las expectativas de modificación o reparación de estos productos. En estos casos se podrían adoptar, entre otras intermedias, dos soluciones diferentes:

1) Cambiar el nivel de integración de los componentes lógicos que se utilizarán en la realización. Los próximos capítulos abordan esta opción cada día más importante en la práctica.
2) Mantener el nivel de integración (pequeña escala de integración, SSI, y elementos de la mediana, MSI) pero romper la modularidad para tratar de economizar puertas, biestables, etc. Esta vía podría desarrollarse buscando una fuerte codificación de los marcados, lo que conduciría a circuitos difícilmente comprensibles, y modificables, etc; además, en las realizaciones asíncronas sería extremadamente complicado el garantizar la ausencia de aleatoriedades. Por otro lado, normalmente la fuerte codificación del marcado conduciría a circuitos combinatorios complejos por lo que convendría sustituirlos más o menos directamente por memorias o matrices lógicas programables. Es decir, se deberían utilizar macrocomponentes (de gran escala de integración, LSI). En §7.3 y 7.5 se abordará la realización de GR y RdP con memorias muertas (ROM) y/o matrices programables (PLA). La realización de RdP utilizando las técnicas propias a los GR exige su transformación previa, lo que se abordará en §7.4.

En resumen, salvo en casos excepcionales, si la realización modular no conviene, deben utilizarse componentes de un mayor nivel de integración. Ello cambia radicalmente las técnicas de realización, que culminarán con la utilización de computadores más o menos especializados (capítulos 8 y 9).

EJERCICIOS

6.1 Obténgase un circuito asíncrono que realice la RdP de la figura 6.1*a* (*Sugerencia*. Obsérvese que p_3 es fuente con respecto a e_3.)

6.2 Propóngase una realización asíncrona de la RdP de la figura 1.15*a* tras cambiar M por $M\uparrow$.

6.3 Compruébese, construyendo un ejemplo, que existen RdP no-simples y conformes realizables por llamada-respuesta.

6.4 Determínese el efecto de introducir un retraso entre Q_1 y la puerta T (figura 6.12*a*) o bien entre Q_1 y la puerta A. (*Sugerencia*. Trácense los cronogramas correspondientes al disparo de la transición.)

6.5 El ejercicio anterior permite constatar que la introducción de retrasos en los puntos especificados no introduce aleatoriedades. Compruébese que, si un circuito asíncrono funciona por transferencia impulsional, la inserción de retrasos, entre la puerta que simula el disparo de la transición y la activación de las células asociadas a sus lugares de salida puede introducir aleatoriedades sobre las salidas.

6.6 Calcúlese el tiempo que tarda en desarrollarse totalmente la simulación del disparo de una transición (figura 6.12*a*) si el circuito realizado funciona por llamada-respuesta. ¿Cuánto tiempo están solapados Q_1 y Q_2? Compárense estos valores con los obtenidos para el circuito funcionando por transferencia impulsional (figura 6.12*b*).
NOTA. Este resultado demuestra que las técnicas de realización expuestas en §6.2 y 3 admiten que los pulsos lleguen a solaparse.

6.7 Supóngase que todas las variables de entrada de un circuito asíncrono (transferencia impulsional o llamada respuesta) son pulsos. Demuéstrese que si su duración es la mínima (§6.2.4*a*), el circuito funciona correctamente si los flancos de subida de dos pulsos consecutivos distan más de $\epsilon + \delta$ unidades de tiempo. Compruébese que al aumentar la anchura de los pulsos, se disminuye en idéntica cantidad la distancia $\epsilon + \delta$. ¿Qué significaría un valor negativo en la distancia entre dos pulsos consecutivos?

6.8 El circuito integrado 7425 posee dos puertas NOR de 4 entradas con habilitación (*strobe*). La función lógica de la puerta i es $\overline{H_i(E_{i_1} + E_{i_2} + E_{i_3} + E_{i_4})}$. Diséñese una célula asíncrona con tres entradas de activación, otras tres de desactivación, y la inicialización (marcado o desmarcado inicial).

7

Realización con memorias y matrices lógicas programables

7.1 INTRODUCCIÓN

La evolución tecnológica ha hecho posible que el diseñador de sistemas digitales disponga de una amplia gama de componentes programables de *gran escala de integración*†, *macrocomponentes*. En este capítulo expondremos un conjunto de técnicas para la realización de sistemas lógicos basadas en dos clases de macrocomponentes ampliamente difundidos:

1) Las *memorias lógicas con decodificador fijo*, que pueden ser de:
   a) *Sólo lectura* y por ello también denominadas *memorias muertas*. Entre ellas se encuentran las ROM (*Read Only Memory*) y sus variantes tecnológicas [*Programable* ROM: PROM; *Erasable* PROM: EPROM; . . .].
   b) *Lectura y escritura* o RAM (*Random Access Memory*).
2) Las *memorias lógicas con decodificador programable*, comúnmente denominadas *matrices lógicas programables*. Entre éstas se encuentran las PLA (*Programmable Logic Array*) y sus variantes [FPLA, *Field* PLA; SPLA, *Sequential* PLA, . . .].

En lo sucesivo utilizaremos las memorias sólo en modo de lectura, por lo que el primer grupo de macrocomponentes se designará, a veces, de forma abreviada con el nombre de su más típico representante, ROM. Del mismo modo, el segundo grupo de macrocomponentes será designado de forma abreviada por PLA.

Para sistemas de una cierta complejidad, será más interesante, normalmente, una realización con macrocomponentes que otra construida exclusivamente con puertas lógicas y biestables, componentes de *pequeña escala de integración* (*Small Scale Integration*, SSI). En efecto, al utilizar componentes altamente integrados se observa que:

1) el coste de los componentes por función que se desea realizar disminuye;
2) el conexionado entre componentes se reduce, lo cual implica una importante disminución del coste de montaje y de prueba, incrementándose la fiabilidad del equipo;

† LSI, *Large Scale Integration*.

3) la modificabilidad del equipo aumenta debido a las posibilidades inherentes a la programación;
4) el tamaño del equipo y su consumo energético se reducen de forma ostensible;
5) debido a la alta regularidad del diseño, éste y su posterior prueba de correcto funcionamiento son, en gran parte, automatizables.

La realización de RdP con memorias y matrices lógicas programables se basa en los métodos utilizados para realizar sistemas *secuenciales* (GR) y éstos a su vez en los empleados para realizar sistemas *combinacionales*. Por ello, el plan de exposición adoptado en este capítulo parte de la definición funcional de las ROM y PLA, y su aplicación directa a la realización de funciones lógicas combinacionales. Posteriormente, y de forma progresiva, se abordan técnicas de realización de sistemas secuenciales (representados con grafos reducidos, GR) y concurrentes (representados con redes de Petri, RdP). La dificultad o ineficiencia de la realización directa de RdP conduce a su descomposición en sistemas secuenciales interconectados que simulan su comportamiento.

7.2 ROM Y PLA: DEFINICIÓN FUNCIONAL Y SÍNTESIS DIRECTA DE FUNCIONES LÓGICAS COMBINACIONALES

La definición funcional de las memorias ROM, PROM y EPROM es idéntica. Su diferencia esencial estriba en que la programación de la ROM la realiza el fabricante del macrocomponente, mientras que la PROM y la EPROM pueden ser grabadas por el usuario (la primera de ellas una sola vez). La diferencia establecida entre la ROM y la PROM es la misma que permite distinguir la PLA y la FPLA.

La utilización de una ROM para realizar de forma directa las funciones lógicas combinacionales se basa en las *representaciones tabulares* de éstas. La materialización con PLA puede llevarse a cabo utilizando representaciones tabulares abreviadas de las funciones o bien representaciones algebraicas (álgebra de BOOLE). En §7.2.3 se presentarán técnicas para un tipo de optimización de las realizaciones basadas en ROM o PLA.

7.2.1 Memorias ROM

7.2.1.1 *Definición funcional*

Un *registro* es un subsistema digital capaz de almacenar información. En lo sucesivo, los registros se designarán mediante letras versalitas (A, B, C, ...) y su contenido por las letras que lo identifican escritas entre paréntesis. Por ejemplo, (RI) significa «el contenido del registro RI».

Una memoria ROM (figura 7.1) está compuesta por:

1) *Un conjunto de C registros numerados de 0 a C*-1. A cada uno de los registros se denomina *palabra* o *posición de memoria*. El número que identifica unívocamente a una palabra se denomina *dirección* de la palabra.

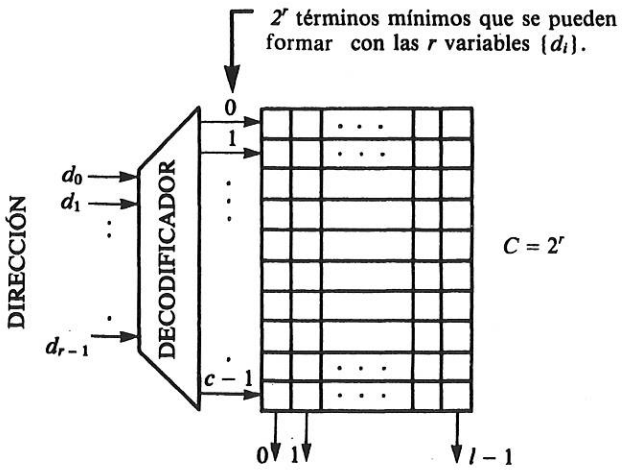


Figura 7.1 Esquema funcional de una memoria de sólo lectura (ROM/PROM/EPROM). (*Nota.* el decodificador es exhaustivo: r a 2^r.)

La *capacidad de una memoria*, C, es el número de registros que contiene. La *longitud* de una memoria, l, es el número de *dígitos binarios* (bits, *binary digits*) que componen cada palabra.

2) *Un decodificador*, dispositivo que a partir de una dirección selecciona la palabra correspondiente. El decodificador genera los 2^r términos mínimos que se pueden formar con las r variables lógicas que codifican la dirección en base binaria, $\{d_i\}$. Evidentemente: $C = 2^r$.

En una ROM (PROM, ...), el acceso a cada palabra se puede efectuar sólo para leer su contenido y de forma directa (también denominada aleatoria), por lo que el tiempo de acceso es constante.

Desde un punto de vista cuantitativo, una memoria ROM puede caracterizarse por el par $\langle C, l \rangle$, siendo Cl el número de puntos elementales de memoria que contiene. La figura 7.2 presenta diversas interconexiones posibles para cambiar el par $\langle C, l \rangle$ en $\langle C', l' \rangle$. En la figura 7.2*a* puede observarse la utilización de un *selector de circuito* (*chip select*, CS), entrada cuya misión consiste en permitir la extensión de la capacidad de direccionamiento global. Una memoria no estará direccionada si su CS no se encuentra activado. El direccionamiento de la figura 7.2*b* se denomina, a veces, a *dos niveles* (decodificador/multiplexor).

7.2.1.2 *Aplicación a la síntesis directa de funciones lógicas combinacionales*

La realización directa de funciones lógicas combinacionales con memorias ROM, se reduce a la realización de la *tabla de verdad* del conjunto de dichas funciones (figura 7.3).

Las variables de entrada se utilizan para definir una dirección. Cada palabra contiene el valor que toma el conjunto de las funciones que se realizan, para el argumento definido por la dirección. De acuerdo con esta convención, se puede comprobar

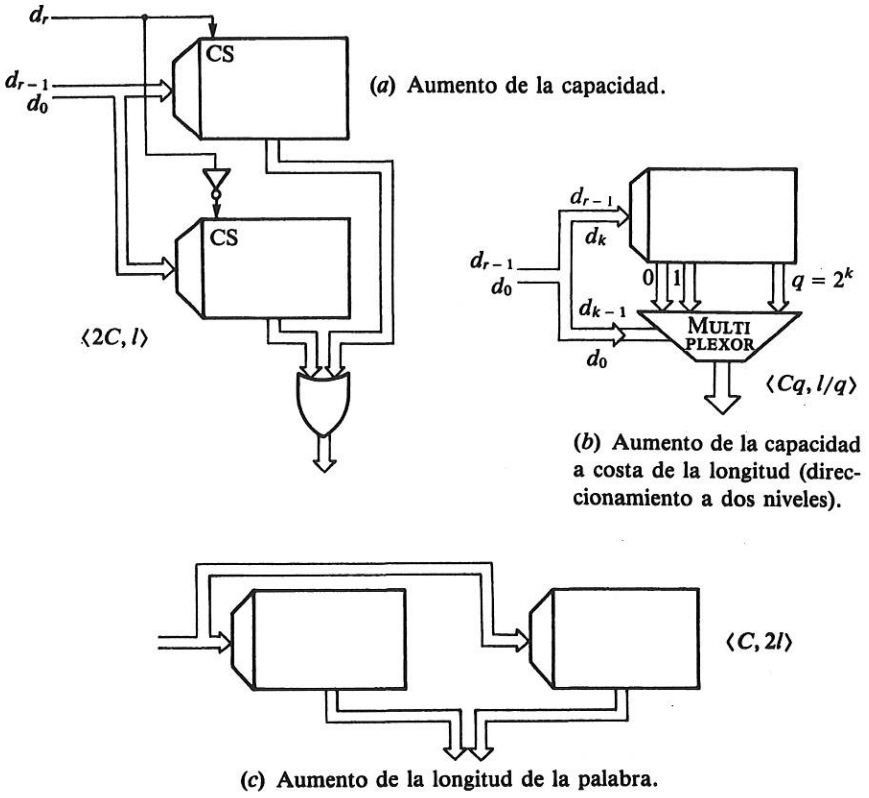


Figura 7.2. Modificación de capacidad y/o longitud.

que la palabra número 1001 ($A = D = 1, B = C = 0$) registra la activación de las funciones F_1 y F_2, de donde se deduce que:

$$F_1(A\bar{B}\bar{C}D) = F_2(A\bar{B}\bar{C}D) = 1 \quad \text{y} \quad F_3(A\bar{B}\bar{C}D) = F_4(A\bar{B}\bar{C}D) = 0$$

Un rápido análisis del método sugiere las observaciones siguientes:

1) La longitud de la palabra ha de ser igual o mayor que s, número de salidas o funciones que se desean realizar: $l \geqslant s$.
2) La capacidad de la memoria no ha de ser inferior a 2^e, donde e es el número de variables de entrada, con las que se define el conjunto de las s salidas: $C \geqslant 2^e$ (¡es decir, el crecimiento de C es exponencial!).
3) Las expresiones algebraicas son innecesarias.

Como se puede comprender fácilmente, la principal limitación del método de síntesis expuesto reside en el *crecimiento exponencial* de la capacidad de la memoria. Ello se debe a la propia definición del decodificador de la ROM que es fijo y completo. En el próximo párrafo se considera la PLA, macrocomponente que, considerado como memoria, posee un decodificador programable, por lo que permite disponer de un mayor número de variables de entrada para la misma capacidad.

$F_1 = A\bar{B}\bar{C}D$

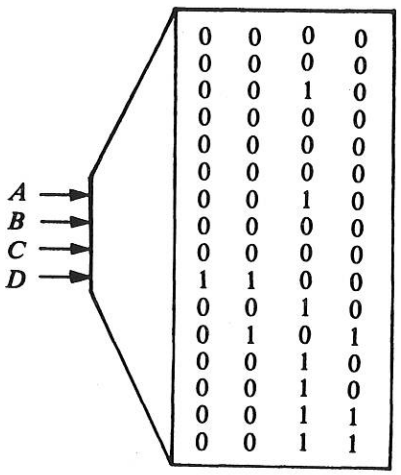
$F_2 = A\bar{B}D$

$F_3 = AB + C\bar{D}$

$F_4 = ABC + ACD$

F_i (CD \ AB)	00	01	11	10
00	○	○	F_3	○
01	○	○	F_3	F_1, F_2
11	○	○	F_3, F_4	F_2, F_4
10	F_3	F_3	F_3, F_4	F_3

a) Expresiones algebraicas *b*) Tabla de KARNAUGH



(*d*) Inscripción en la memoria realizando F_1, F_2, F_3 y F_4.

A	B	C	D	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	1	0	0	0	1	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	0	0	0	1	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	1	0	0
1	0	1	0	0	0	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	0	1	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	1	1
1	1	1	1	0	0	1	1

(*c*) Tabla de verdad de las funciones F_1, F_2, F_3 y F_4.

	A	B	C	S_1	S_2
(π_1)	0	1	1	1	1
(π_2)	1	X	0	1	0
(π_3)	1	0	X	0	1

MATRIZ-Y MATRIZ-O

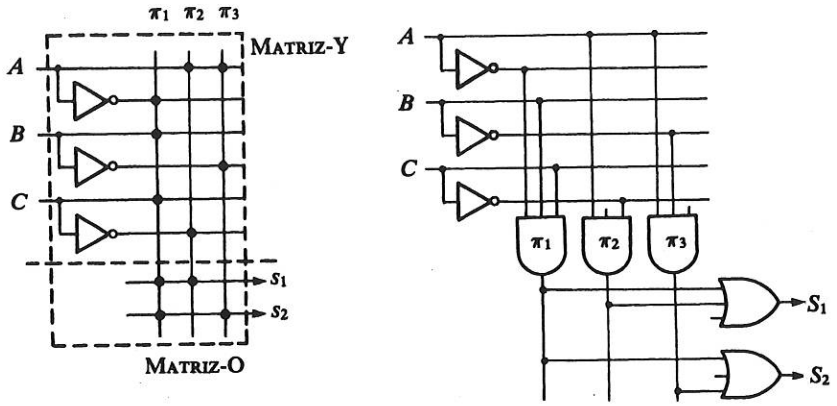
(*c*) Tabla de programación (la no participación de una variable en un término producto se marca con X).

Figura 7.3 Idea básica para la realización de funciones lógicas combinacionales con ROM.

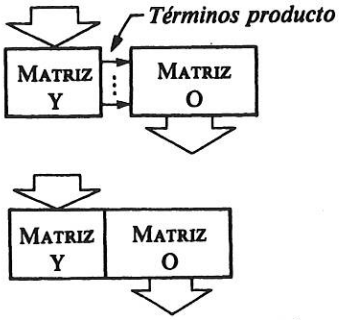
7.2.2 Matrices lógicas programables: PLA

7.2.2.1 *Definición funcional y aplicación a la síntesis directa de funciones lógicas combinacionales*

Una PLA está compuesta básicamente por *dos matrices lógicas programables,* la MATRIZ-Y (AND-ARRAY) y la MATRIZ-O (OR-ARRAY) (figura 7.4). La MATRIZ-Y realiza *intersecciones* (función lógica Y) de subconjuntos seleccionados de variables de entrada o de sus complementos†. Para simplificar las referencias, normalmente supon-



(*a*) Diagrama lógico esquemático (*b*) Diagrama lógico equivalente



(*d*) Representaciones de una PLA.

Figura 7.4. Matrices lógicas programables (PLA).

† Toda selección se representa gráficamente con un punto: ●.

dremos incluida en la MATRIZ-Y la generación del complemento de las variables de entrada. Cada intersección realizada por la MATRIZ-Y se denomina *término producto* (*p-term*). La MATRIZ-O *suma lógicamente* subconjuntos de términos producto seleccionados, generando las salidas.

Para reducir el número de términos producto utilizados al realizar una (varias) función(es), ciertas PLA permiten, mediante programación, *invertir* funciones generadas. Así, por ejemplo, la función $F = (a + \bar{b})(c + d)$ necesita para su generación 4 términos producto (*implicantes*), mientras que su inversa sólo necesita 2, $\bar{F} = \bar{a}b + \bar{c}\bar{d}$.

Una PLA se caracteriza cuantitativamente por el triple $\langle e, s, \pi \rangle$ que representa el *número de entradas*, el de *salidas* y el de *términos producto* (*p-terms*). La PLA de la figura 7.4 se caracteriza por el triple $\langle 3, 2, 3 \rangle$. Valores típicos son $e = 14$ y 16, $s = 8$ y $\pi = 96$ (obsérvese que $\pi \ll 2^e$). En una FPLA, normalmente, $\pi = 48$.

Cuando las funciones que se desean realizar son demasiado complejas y, por lo tanto, desbordan la capacidad de una PLA (a pesar de eventuales simplificaciones u optimizaciones), puede procederse a la definición de estructuras multi-PLA. Como se puede observar en la figura 7.5, la extensión del número de *p*-términos o de salidas es muy simple, sin embargo la extensión del número de entradas conlleva dificultades. En el ejemplo de la figura 7.5*c* se exhiben las restricciones sobre las funciones generadas.

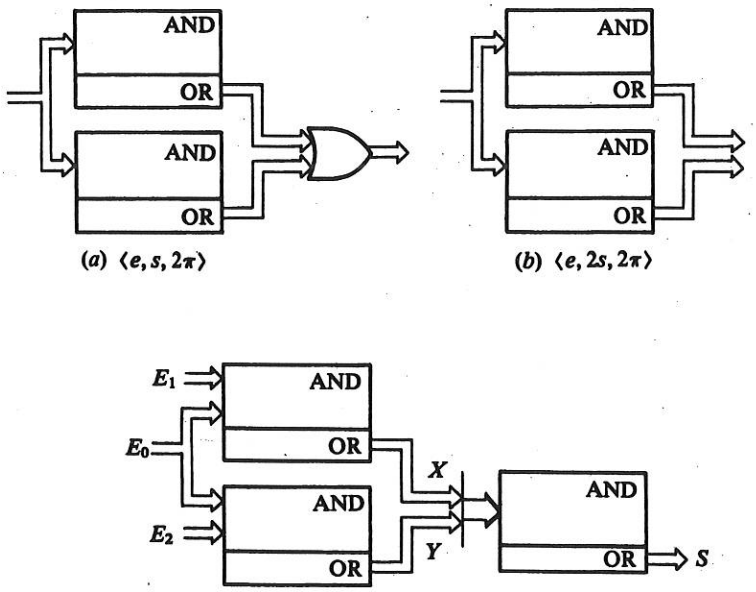


Figura 7.5. Algunas estructuras multi-PLA.

Las PLA *secuenciales* (SPLA) son PLA a las que se les ha añadido un *conjunto de biestables síncronos* (*registro*) con las conexiones internas para su excitación (figura 7.6). Su caracterización necesita un cuarto parámetro, *b*, que indique el *número de biestables integrados*, así como su *tipo* (J-K, D, T, ...). El registro de una SPLA está destinado al almacenamiento del *código del marcado* (*estado interno*) de un sistema lógico secuencial. A modo de ejemplo, la TMS2000 y la TMS2200 se caracterizan por ⟨17, 18, 60, 8 J-K⟩ y ⟨13, 10, 72, 10 J-K⟩, respectivamente.

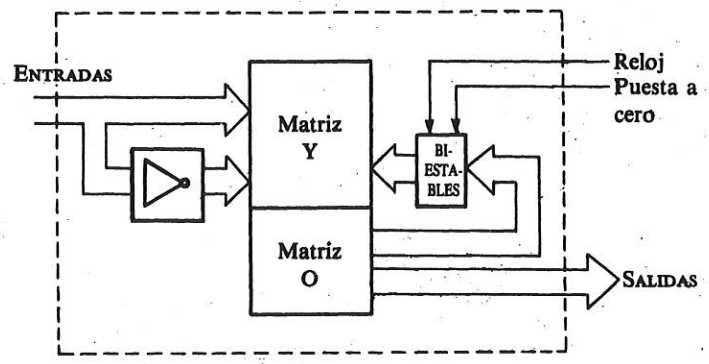


Figura 7.6. PLA-secuencial (SPLA).

7.2.2.2 *Cuestiones adicionales sobre síntesis directa de funciones lógicas combinacionales*

Según la convención lógica utilizada†, una PLA realiza las funciones combinacionales bajo una de sus dos formas canónicas:

1) *Lógica positiva.* 1.ª forma canónica: suma de productos ($\Pi\ \Sigma$, AND-OR, NAND-NAND).
2) *Lógica negativa.* 2.ª forma canónica: producto de sumas ($\Pi\ \Sigma$, OR-AND, NOR-NOR).

Utilizando la primera convención, las PLA de la figura 7.7 realizan de dos maneras las funciones lógicas de la figura 7.3. En la correspondiente a la figura 7.7a se ha buscado la minimización del número de términos-producto (5). Esta realización se puede obtener utilizando los métodos clásicos de síntesis de funciones booleanas (Quine-McCluskey [FLET 80], o de los consensos o de Tison [TISO 71]). Como puede observarse fácilmente, si $A = B = C = 1$ y $D = 0$, el segundo, tercero y cuarto término-producto estarán activados simultáneamente.

La realización de la figura 7.7*b* es tal que todos los términos-producto son *disjuntos*, por lo que nunca puede haber dos activados simultáneamente. Esta pro-

† *Lógica positiva*: 1-estado de actividad. Lógica negativa: 0-estado de actividad.

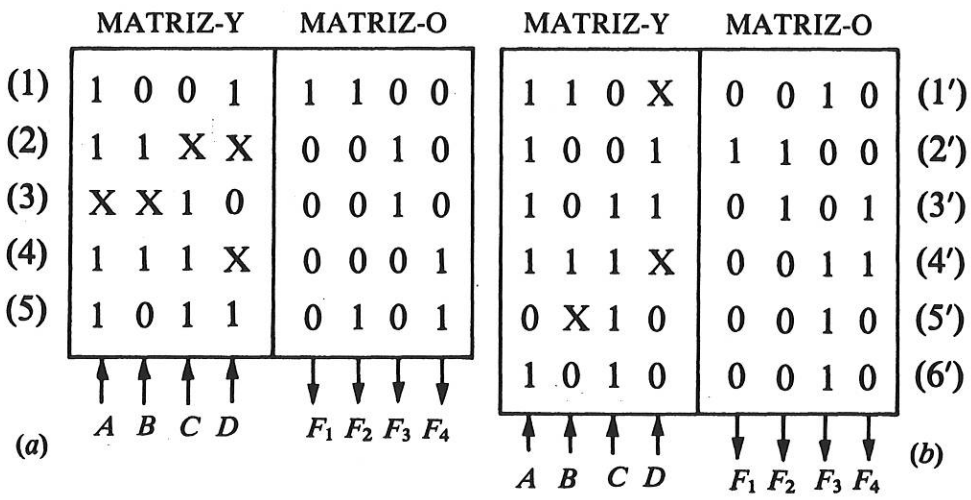


Figura 7.7. Relaciones de las funciones lógicas definidas en la figura 7.3. (*Nota*: en el caso *b* los términos producto son disjuntos. La PLA puede considerarse como una memoria asociativa).

piedad permite contemplar la PLA como una *memoria asociativa* (*Content Address Memory*, CAM); es decir, una memoria en la que se selecciona una palabra por el contenido de uno de sus campos y no por una dirección. Observando la PLA desde esta perspectiva, toda línea se lee como una proposición lógica «**si** condición **entonces** acción». (Por ejemplo, la primera línea se leerá: **si** $AB\bar{C}$ **entonces** $F_1 := F_2 := F_4 := 0$ y $F_3 := 1$;).

De acuerdo con lo expresado anteriormente, al contemplar una PLA como memoria asociativa, se puede afirmar que:

1) La MATRIZ-O cumple idéntica función que el conjunto de palabras que componen una ROM. Dicho de otro modo, la MATRIZ-O cumple la función de «*memoria*» en una PLA. La capacidad de esta memoria es el número de términos-producto que posee.
2) La MATRIZ-Y cumple la función de *decodificador programable*.

En resumen, una ROM es un caso particular de PLA en el que el decodificador es fijo (no programable) y genera los 2^e términos mínimos, en vez de π términos producto. La realización de funciones lógicas con ROM se lleva a cabo expresándolas como suma de *términos mínimos*; con la utilización de PLA, la realización puede hacerse sumando *términos producto* (éstos también se denominan en la literatura *implicantes* o *monomios lógicos*).

A modo de observaciones finales sobre optimización, puede establecerse que:

1) Si un sistema combinacional se define como una memoria asociativa, normalmente es posible reducir el número de términos-producto necesarios utilizando las técnicas clásicas de minimización de funciones lógicas.

2) El problema de la extensión del número de entradas a una PLA conduce al estudio de una partición de éstas (figura 7.5*c*) o al establecimiento de un procesamiento previo a la PLA. Así, por ejemplo, resulta evidente que si hemos de realizar las funciones: $F_1 = ABC(D\bar{E} + EF), F_2 = D(ABC + EF)$ y $F_3 = E(\bar{A} + \bar{B} + \bar{C} + \bar{D})$, al calcular previamente $Z = ABC$, pueden obtenerse las F_i con una PLA de sólo cuatro entradas (Z, D, E, F).

Desafortunadamente, no existen métodos generales eficaces para particionar las variables de entrada o realizar procesamientos previos. El lector interesado puede orientarse sobre la optimización de PLA a partir de las referencias [ROTH 78], [AUGI 78], [FLET 75] y [PERE 80].

7.2.3 Reducción de la longitud de la palabra en ROM o PLA

En lo expresado anteriormente se ha supuesto que a cada función se le asignaba un dígito binario (bit) de las palabras de la ROM o de la PLA. La primera reducción que puede sugerirse es, evidentemente, la eliminación de funciones *idénticas* (se eliminarán todas menos una). Las funciones *a* y *b* (tabla 7.1) son idénticas. Cuando las funciones son *incompletamente especificadas*, se dice que son *pseudo-idénticas* si son idénticas para alguna especificación de las indeterminaciones. La pseudo-identidad es una relación de *compatibilidad*†. Todo conjunto de funciones pseudo-idénticas dos a dos (clase compatible) puede reducirse, tras las correspondientes especificaciones, a una única función.

Funciones que se generan

Configuraciones diferentes	*a*	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*
1	1	1	1	1	1	1	0	0	0	0	0
2	0	0	1	0	0	0	1	1	1	0	0
3	1	1	0	0	0	0	0	1	1	1	0
4	0	0	0	1	0	0	0	1	0	0	1
5	0	0	0	0	0	1	0	1	0	0	0

Tabla 7.1 Conjunto de las 5 diferentes configuraciones que aparecen en la MATRIZ-O de la figura 7.8.

† *Reflexiva* y *simétrica*. En §3.4.2.1 se introdujeron los conceptos de *clase de compatibilidad* y *compatible máximo*; además se presentó un algoritmo para la obtención de los *compatibles máximos*.

Al margen de estas reducciones más o menos inmediatas, a continuación se consideran dos esquemas clásicos de codificación. Éstos se basan en la generación de unas *funciones intermedias* que, tras una cierta decodificación, permiten obtener las funciones deseadas. En cualquier caso, es importante señalar que al insertar un nuevo nivel de decodificación, el tiempo de respuesta del sistema se verá incrementado; es decir, la realización será más lenta.

7.2.3.1 *Codificación total*

Sea D el número de *configuraciones diferentes* que aparecen al definir el contenido de una ROM o PLA (en la tabla 7.1, $D = 5$). Considerando la máxima codificación posible, las D configuraciones pueden representarse con $\Delta = \lceil \log_2 D \rceil$ variables. El método de compactación consiste en sustituir cada configuración por un código intermedio y, posteriormente, decodificar éste (figura 7.8*a*). De este modo, las 5 configuraciones de la tabla 7.1 pueden codificarse con 3 bits ($\lceil \log_2 5 \rceil = 3$).

Estimando por $\gamma D\Delta$ el coste de la MATRIZ-Y, donde γ es el coste relativo de un bit de la MATRIZ-Y con respecto a uno de la MATRIZ-O, el esquema de codificación total puede ser interesante si:

$$Cl > C\Delta + \gamma D\Delta + Dl.$$

En el caso de la tabla 7.1: $C \cdot 10 > C \cdot 3 + \gamma \cdot 5 \cdot 3 + 5 \cdot 10 \Rightarrow 7C > 15\gamma + 50$. Admitiendo $\gamma = 2$†, el esquema de codificación total puede interesar si $C > 12$.

Los dos problemas principales que plantea este método de reducción son:

1) Su poca flexibilidad (la introducción de alguna variación puede implicar grandes cambios).
2) La relativa complejidad del decodificador final (ROM o PLA).

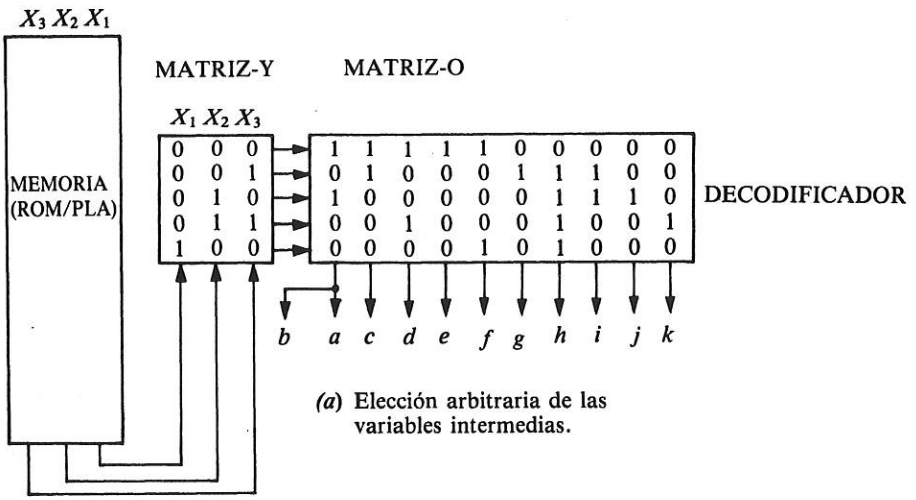
En este punto interesa señalar que una adecuada estrategia en la elección de las Δ *variables intermedias* que codifican las D configuraciones puede reducir significativamente la longitud de la palabra del decodificador. En efecto, basta con utilizar directamente como variables intermedias cuantas funciones sea posible de entre las que *se desean generar* (figura 7.8*b*).

Para tratar sistemáticamente la anterior reducción se puede proceder como sigue. Sea X un *subconjunto de las funciones que se desea generar*. Se define r_x como el mayor número de veces que se repite el contenido de una fila en la matriz formada por el subconjunto de las columnas indicado por X. De acuerdo con esta definición, si $X = \{a, c\}$ (tabla 7.1), entonces $r_x = 2$ («00» se repite en la 4.ª y 5.ª fila); del mismo modo, si $X' = \{a, e\}$, $r_{x'} = 3$ («00» se repite en la 2.ª, 4.ª y 5.ª fila).

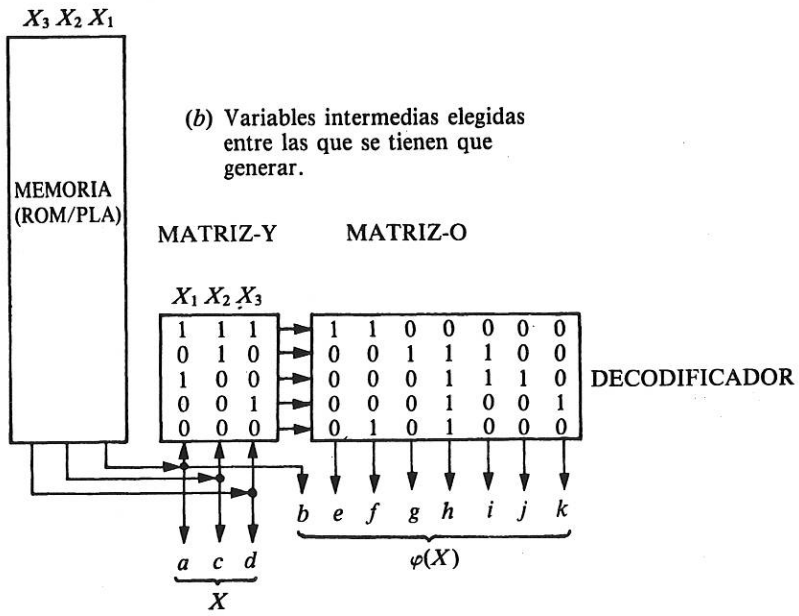
Supóngase que de las Δ variables intermedias necesarias para codificar las D configuraciones, $|X|$ son simultáneamente funciones que se tienen que generar. Veamos acto seguido que condición deben cumplir las funciones $X_i \in X$ para que se puedan utilizar como variables intermedias, sin incrementar el total de Δ. Dado que r_X expresa el máximo número de filas que, codificadas con las variables de X, poseen idéntica representación para que todas las filas puedan tener distinto código, es ne-

† Por cada variable de entrada e existen dos líneas: e y $\bar{e}$.

cesario que las $\Delta - |X|$ variables restantes puedan diferenciar esos grupos de r_X filas. Como $\Delta - |X|$ variables pueden diferenciar como máximo $2^{\Delta - |X|}$ posibilidades, se deduce que es necesario que $r_X \leqslant 2^{\Delta - |X|}$.



(*a*) Elección arbitraria de las variables intermedias.



(*b*) Variables intermedias elegidas entre las que se tienen que generar.

Figura 7.8. Reducción de la longitud de la palabra mediante codificación total y decodificación posterior.

En el ejemplo que venimos considerando, a y c pueden ser utilizadas como variables intermedias puesto que $r_{X'} = 2 \leqslant 2^{3-2} = 2$. Sin embargo, a y e no pueden ser utilizadas habida cuenta que $r_{X'} = 3 > 2^{3-2} = 2$. Entre otras muchas ternas, $\bar{X} = \{a, c, d\}$ puede ser utilizada para codificar las 5 configuraciones porque $r_{\bar{X}} = 1 \leqslant 2^{3-3} = 1$ (figura 7.8*b*).

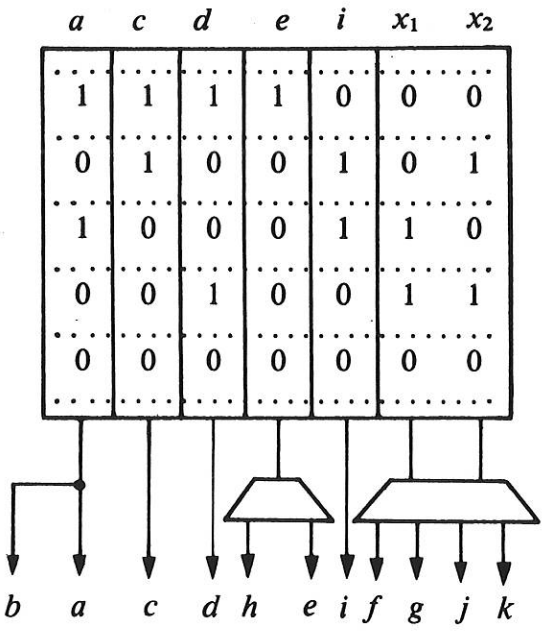
En general, desde un punto de vista algorítmico, se puede proceder construyendo conjuntos de $|X| = 1, 2, \ldots$ vectores mientras que se vaya cumpliendo la condición $r_X \leqslant 2^{\Delta - |x|}$.

EJERCICIO. Compruébese que en determinados casos es imposible utilizar alguna función de las que se desean generar para codificar las D configuraciones. (*Sugerencia.* Sea $j > 1$ y tómense, por ejemplo, las 2^j configuraciones de vectores de longitud 2^j y un único 1.)

7.2.3.2 *Codificación independiente por campos*

Entre el esquema de *codificación total* (§7.2.3.1) y el de partida (completamente decodificado: 1 bit por función) existe un esquema intermedio denominado *codificación por campos* (figura 7.9). La idea básica en este nuevo esquema de codificación consiste en:

1) agrupar las funciones en subconjuntos mutuamente excluyentes,
2) codificar de forma independiente cada uno de los subconjuntos obtenidos.



x_1	x_2	acción codificada
0	0	f
0	1	g
1	0	j
1	1	k

Figura 7.9. Codificación independiente por campos.

Dos funciones son *mutuamente excluyentes* si nunca están activas simultáneamente. (Dicho de otro modo, dos funciones se excluyen mutuamente si son disjuntas, su intersección es nula; es decir, no poseen en común ningún término producto o implicante). La relación de exclusión mutua es una relación de *compatibilidad*, de donde los subconjuntos mutuamente excluyentes son *clases de compatibilidad* (cfr. §3.4.2.1).

Cuando la unión de las B_i funciones de una clase de compatibilidad es la *unidad*, la clase puede codificarse con $\lceil \log_2 B_i \rceil$ variables booleanas; en caso contrario, la codificación de la clase necesita $\lceil \log_2(B_i + 1) \rceil$, donde se añade el «+ 1» para codificar la ausencia de función activa. Para facilitar el razonamiento que sigue, se dirá que una clase de compatibilidad es *principal* si la unión de las funciones que considera es la unidad o bien $B_i \neq 2^j (j \in \mathbb{N})$. Es decir, una clase es principal si no es necesario codificar la «ausencia de función activa» o bien, si siendo necesario codificarla, su consideración no incrementa el coste de la codificación de la clase. En estas condiciones es fácil demostrar que toda clase no principal puede dividirse en dos subclases sin incrementar el coste de la realización. (Demuéstrese.)

El problema de optimización (reducción) que se plantea naturalmente es el de encontrar conjuntos de funciones mutuamente excluyentes (compatibles) que minimicen la longitud de la palabra y realicen todas las funciones. No es difícil comprobar que siempre existe solución *óptima* que considera sólo clases principales disjuntas (su intersección es nula). Por otro lado, es evidente que el número de clases que forma cualquier solución debe ser igual o superior al cardinal del mayor *incompatible máximo*. En el ejemplo de la tabla 7.1 (no considerando *b* por ser idéntica a *a*) existen incompatibles de 5 elementos (por ejemplo, $\{a, c, d, e, f\}$, puesto que en la configuración 1 se tiene $a = c = d = e = f = 1$).

La primera columna de la tabla 7.2 presenta la lista de los subconjuntos máximos de funciones de la tabla 7.1 mutuamente excluyentes (compatibles máximos)†. En

	Lista de compatibles máximos	Lista tras seleccionar 6 y 8	Solución	Coste
1	*agk*	*a*	*a*	1
2	*cjk*	*c*	*c*	1
3	*dgj*	*d*	*d*	1
4	*di*	*di*	—	—
5	*egjk*	—	—	—
6	*eh*	*eh*	*eh*	1
7	*eik*	*i*	*i*	1
8	*fgjk*	*fgjk*	*fgjk*	2
9	*fik*	*i*	—	—
			Coste total	7

Tabla 7.2 Obtención de una solución compactada por campos.

† En §3.4.2.1 se presentó un algoritmo para calcular los *compatibles máximos*.

esta lista todas son principales salvo las clases 4 y 5. Los pares de clases principales disjuntas son: $\{1,6\}$, $\{2,6\}$, $\{3,6\}$, $\{3,7\}$, $\{3,9\}$, $\{6,8\}$ y $\{6,9\}$. Ahora bien, las clases 6 y 8 no necesitan codificar la ausencia de activación puesto que $e + h = 1$ y $f + g + j + k = 1$. Tomándolas para la solución final y eliminando las variables que contiene en el resto de compatibles, se obtiene la segunda columna de la tabla 7.2. Se eligen para la solución las clases principales disjuntas: a, c, d e i. (Obsérvese que la clase $\{d, i\}$ no es principal y por lo tanto puede dividirse en dos sin que el coste de una solución se incremente. En este caso se reobtiene: d e i.) La figura 7.9 presenta el resultado final, en el que f, g, j, y k son codificados mediante las variables intermedias x_1 y x_2.

Resumiendo desde un punto de vista metodológico, el proceso de codificación por campos independientes esbozado, procede en dos fases netamente diferenciadas:

1) La obtención de una *partición de las funciones*, en la que cada elemento de la partición definirá un *campo*.
2) La *codificación independiente de cada campo* con variables intermedias.

A modo de comentario final sobre la codificación por campos, se puede afirmar que es un esquema:

1) razonablemente flexible (modificaciones locales) y eficiente
2) que utiliza sólo decodificadores estándar (decodificador $x - a - 2^x$), para los que sólo una salida puede estar activa en un momento dado.

7.3 REALIZACIÓN DE GRAFOS REDUCIDOS

Dadas las variables de entrada (E) y las de salida (S) de un sistema secuencial proceder a su realización consistirá en materializar:

(1) el estado (Q);
(2) la función de transición entre estados ($\delta: E \times Q \rightarrow Q$);
(3) la función de salida ($\lambda_1: Q \rightarrow S$, si es una máquina de MOORE ó $\lambda_2: E \times Q \rightarrow S$, si es una máquina de MEALY).

En el capítulo 6 se vió que la realización cableada utiliza: (1) un conjunto de biestables, que pueden ser globalmente considerados como un registro donde se almacena el estado (marcado) actual del sistema; (2) un circuito combinatorio (SSI) que calcula el próximo estado (δ) y (3) un circuito combinatorio (SSI) que calcula las salidas (λ).

Las realizaciones que se presentan en este apartado poseen análogamente un *registro de estado* (donde se almacena el estado actual). El circuito combinatorio discreto que determina las salidas y parte del que determina el próximo estado son sustituidos por una *memoria* (ROM) o/y *una matriz lógica programable* (PLA). El registro de estado atacará la ROM o PLA, conviertiéndose en un registro de dirección, al menos parcialmente.

Para sistematizar su estudio, clasificamos los métodos de realización que siguen en dos grandes grupos:

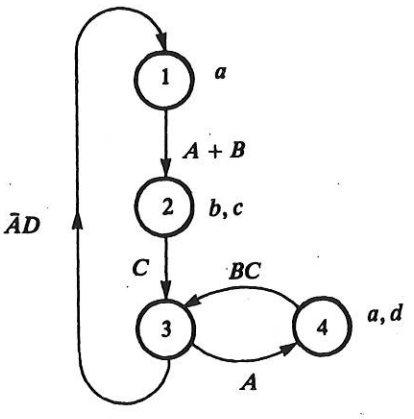
1) Métodos lógicos de transición directa (§7.3.1).
2) Métodos con secuenciador (también denominado métodos con interpretador o microprogramados) (§7.3.3).

En §7.3.2 se presentan dos métodos pertenecientes al primer grupo pero que permiten establecer con cierta continuidad la evolución conceptual entre éstos y los del segundo grupo.

Los primeros, en sus esquemas básicos, utilizan exclusivamente un *registro de estado* y la ROM o PLA. En esquemas más elaborados aparecen circuitos *decodificadores* que reducen la longitud de la palabra y circuitos *selectores* (esencialmente multiplexores) destinados a reducir el número de variables de entrada a la memoria o matriz lógica. Las técnicas de síntesis se basan en una codificación del estado seguida de una cumplimentación (escritura) más o menos inmediata, de la tabla de verdad en la ROM o la PLA. El estado sucesor de uno dado se alcanza en un único ciclo de reloj (una única lectura de la ROM o PLA).

En los métodos del segundo grupo emerge el concepto de *programa* y, por consiguiente, la noción de dispositivo *secuenciador* que lo decodifica y ordena su ejecución. Dada la sencillez de las instrucciones (microinstrucciones), el secuenciador suele ser extremadamente rudimentario. Las técnicas de síntesis terminan siendo de programación. Considerando la descripción original, un estado sucesor de uno dado se obtendrá, en general, a través de diversas transiciones intermedias.

Para aligerar la presentación de las diferentes técnicas, nos ceñiremos a la realización de GR con *salidas incondicionales* (máquinas de MOORE). (En particular, utilizaremos el GR de la figura 7.10a.) La transformación de máquinas de MOORE en máquinas de MEALY se presentó en §1.3.1.



(*a*) Grafo reducido.

p_i	v_1 v_2	v_i
p_1	$\{A, B\}$	2
p_2	$\{C\}$	1
p_3	$\{A, D\}$	2
p_4	$\{B, C\}$	2

$$v = \max_i (v_i) = 2$$

(*b*) Variables relevantes para cada estado.

Figura 7.10. Sistema secuencial tipo MOORE que se desea realizar.

7.3.1 Métodos lógicos de transición directa

Se parte de una codificación previa de los estados del GR que se desea realizar. El código que identifica al estado presente se almacena en un registro, denominado *registro de estado*. Cada palabra de la ROM o PLA, representa una transición en el GR

o el mantenimiento del estado presente (cuando éste coincide con el estado sucesor; es decir, no existe evolución). Para ello una palabra posee dos campos:

1) *siguiente estado* o *estado sucesor* (lugar de salida de la transición), que en condiciones de no evolución (mantenimiento del estado) permite volver a obtener el estado actual;
2) *salidas*, campo a partir del que se pueden calcular las acciones o salidas.

Desde un punto de vista estructural, la realización consta de una *memoria* o *matriz lógica* que tiene como entradas las *variables de entrada* del sistema y las del *registro de estado*; la memoria o matriz lógica alimenta al registro de estado (figura 7.6, SPLA) y, eventualmente, a un *registro de salidas*. En lo sucesivo, supondremos que siempre existe el registro de salidas, por lo que debido a su presencia, el campo «salidas» de cada palabra contendrá información destinada al cálculo de las salidas asociadas al estado sucesor y no a las del estado presente. La ausencia del registro puede conducir a aleatoriedades sobre las salidas. (Obsérvese que las variables de entrada al sistema forman parte de la dirección de la memoria, aún cuando realizemos máquinas de MOORE.)

Para facilitar la comprensión de la síntesis de las funciones de transición y de lectura, «conviene» considerar la matriz lógica programable como una *memoria asociativa*. De esta forma: (1) la realización con ROM se deduce por simple expansión de cada término producto en los correspondientes términos mínimos y (2) salvo para la realización «optimizada» con PLA, el proceso de síntesis no tiene que recurrir al cálculo de ecuaciones lógicas, con frecuencia engorroso.

A continuación, sintetizaremos el GR de la figura 7.10*a* utilizando PLAS. Los registros de estado y de salidas estarán formados por biestables que utilicen una única variable de entrada, D o T, con lo que se reducirá la longitud de la palabra. Para minimizar el campo de estado, en vez de utilizar un código 1 entre n (método modular, capítulo 6) se utilizará la máxima codificación posible; es decir, el registro de estado será de $\lceil \log_2 n \rceil$ biestables. De esta forma, la codificación de los cuatro estados del GR de la figura 7.10*a* puede llevarse a cabo con sólo dos biestables, $\{q_2, q_1\}$. En este punto, a modo de importantísima observación, ha de señalarse que la minimización de la longitud del registro de estado implica la reducción del número de variables de entrada a la ROM o la PLA, lo que es extraordinariamente interesante puesto que:

1) la capacidad de la ROM crece exponencialmente con el número de sus variables de entrada (§7.2.1.1):
2) la expansión de las entradas en estructuras multi-PLAS es extremadamente difícil (§7.2.2.1).

7.3.1.1 *Síntesis utilizando registros constituidos por biestables* D

Un biestable D memoriza el valor de su entrada D†. Por ello los campos de estado y de salidas sucesores contendrán directamente el código del estado y las salidas asociadas. La definición de una tabla de programación necesita sólo la previa elección del *código* que representará a cada estado.

---

† La evolución $Q \rightarrow Q^+$ requiere, simplemente, $D = Q^+$ (ecuación de excitación del biestable D).

La elección de un «buen» código puede reducir la longitud de la palabra al hacer que aparezcan funciones *idénticas* en los campos de estado y de salidas. Así, habida cuenta que a (figura 7.10*a*) está asociada a 2 de los 4 estados, se puede utilizar ésta como una de las dos variables intermedias o de estado necesarias, $\{q_2, q_1\}$. Sea por ejemplo, $q_1 = a$. Adoptando la codificación: $p_1 = \bar{q}_2 q_1$, $p_2 = \bar{q}_2 \bar{q}_1$, $p_3 = q_2 \bar{q}_1$ y $p_4 = q_2 q_1$, la tabla 7.3 presenta la *tabla de programación* de una PLA, la cual es directamente una *tabla de verdad compactada*. En ésta puede observarse que a cada estado se le asocia, como mínimo, un término para representar la *condición de mantenimiento*; esta condición se obtiene al complementar la unión de los eventos asociados a las transiciones de salida del lugar. Las condiciones de mantenimiento en el ejemplo son:

$$p_1: \overline{(A + B)} = \bar{A}\bar{B} \qquad p_3: \overline{(A + \bar{A}D)} = \bar{A}\bar{D}$$
$$p_2: \bar{C} = \bar{C} \qquad p_4: \overline{BC} = \bar{B} + \bar{C} = \bar{B} + B\bar{C},$$

de donde, a p_4 han de asociársele dos términos producto. Para que la PLA pueda observarse como memoria asociativa, todos los términos producto se han hecho disjuntos.

A	B	C	D	q_2	q_1	q_2^+	q_1^+	b	d	Significación	
1	X	X	X	0	1	0	0	1	0	$Ap_1 \rightarrow p_2$	σ_2 (transición)
0	1	X	X	0	1	0	0	1	0	$\bar{A}Bp_1 \rightarrow p_2$	σ_2 (transición)
0	0	X	X	0	1	0	1	0	0	$\bar{A}\bar{B}p_1 \rightarrow p_1$	σ_1 (mantenimiento)
X	X	1	X	0	0	1	0	0	0	$Cp_2 \rightarrow p_3$	σ_3 (transición)
X	X	0	X	0	0	0	0	1	0	$\bar{C}p_2 \rightarrow p_2$	σ_2 (mantenimiento)
0	X	X	1	1	0	0	1	0	0	$\bar{A}Dp_3 \rightarrow p_1$	σ_1 (transición)
1	X	X	X	1	0	1	1	0	1	$Ap_3 \rightarrow p_4$	σ_4 (transición)
0	X	X	0	1	0	1	0	0	0	$\bar{A}\bar{D}p_3 \rightarrow p_3$	σ_3 (mantenimiento)
X	1	1	X	1	1	1	0	0	0	$BCp_4 \rightarrow p_3$	σ_3 (transición)
X	0	X	X	1	1	1	1	0	1	$\bar{B}p_4 \rightarrow p_4$	σ_4 (mantenimiento)
X	1	0	X	1	1	1	1	0	1	$B\bar{C}p_4 \rightarrow p_4$	σ_4 (mantenimiento)

Tabla 7.3 Tabla de verdad y de programación de una PLA para realizar con biestables D el GR de la figura 7.10*a*. (*Nota*: $q_1^+ \equiv a$ y $b = c$.) σ_i representa las salidas asociadas a p_i.

EJERCICIO. Compruébese que el cálculo de las funciones lógicas realizadas arroja el siguiente resultado (pueden utilizarse los diagramas de KARNAUGH o los diagramas de KARNAUGH con *variables introducidas* [CLAR 73] [FLET 80]):

$$q_1^+ = \bar{A}Bq_1 + A\bar{B}q_2 + Dq_2\bar{q}_1 + \bar{C}q_2q_1 + Aq_2\bar{q}_1;$$
$$q_2^+ = C\bar{q}_2\bar{q}_1 + \bar{D}q_2 + q_2q_1 + Aq_2;$$
$$b = c = \bar{q}_2^+\,\bar{q}_1^+;$$
$$d = q_2^+\,q_1^+.$$

EJERCICIO. Redúzcase la longitud de la palabra utilizando la técnica expuesta en §7.2.3.1. (*Sugerencia*. Compruébese que las cuatro configuraciones distintas que existen pueden codificarse a partir de q_2^+ y q_1^+.)

7.3.1.2 *Síntesis utilizando registros constituidos por biestables* T †

La utilización de biestables D conduce a una síntesis extraordinariamente simple puesto que la tabla de verdad y la de programación coinciden. No obstante, hay que resaltar que si la realización se lleva a cabo con PLA, un número importante de términos producto corresponden a condiciones de mantenimiento (5 sobre 11 en la tabla 7.3). La utilización del biestable T permitirá la programación sin que se tengan que especificar las condiciones de mantenimiento. En efecto, en toda condición de mantenimiento, el estado y salidas actuales coinciden con el estado y salidas sucesoras, de donde todas las variables de entrada a los registros de estado y salidas deben ser nulas.

De acuerdo con la ecuación de excitación de los biestables T, sus entradas deben calcularse realizando el O-EXCLUSIVO entre el estado y salidas actuales y el estado y salidas sucesoras. Así, por ejemplo, la primera línea de la tabla de verdad que corresponde al GR de la figura 7.10*a* (tabla 7.3) representa una transición de p_1 a p_2. Luego la excitación de los biestables T en una evolución de p_1 a p_2 (o p_2 a p_1) debe ser:

q_2	q_1	b	d		q_2	q_1	b	d		θ_2	θ_1	θ_b	θ_d
(0	0	1	0)	$\oplus$	(0	1	0	0)	=	(0	1	1	0)
información asociada a p_2					información asociada a p_1					excitación de los biestables T			

El vector obtenido es el contenido en las dos primeras transiciones de la tabla 7.4 (ambas transiciones son desde p_1 a p_2).

En resumen, la utilización de biestables T permite reducir significativamente el número de términos producto, por lo que puede ser muy atractiva con PLA. (Habida cuenta que el número de términos mínimos no se reduce, la utilización de los biestables T con ROM no es interesante.) Como dificultad mayor, es importante resaltar que la adopción de biestables T hace que las tablas de verdad (tabla 7.3) y de programación (tabla 7.4) difieran.

A	B	C	D	q_2	q_1	θ_2	θ_1	θ_b	θ_d	Significación (sólo transiciones)
1	X	X	X	0	1	0	1	1	0	$Ap_1 \rightarrow p_2 \mid\mid \sigma_2$
0	1	X	X	0	1	0	1	1	0	$\bar{A}Bp_1 \rightarrow p_2 \mid\mid \sigma_2$
X	X	1	X	0	0	1	0	1	0	$Cp_2 \rightarrow p_3 \mid\mid \sigma_3$
0	X	X	1	1	0	1	1	0	0	$\bar{A}Dp_3 \rightarrow p_3 \mid\mid \sigma_1$
1	X	X	X	1	0	0	1	0	1	$Ap_3 \rightarrow p_4 \mid\mid \sigma_4$
X	1	1	X	1	1	0	1	0	1	$BCp_4 \rightarrow p_3 \mid\mid \sigma_3$

Tabla 7.4 Tabla de programación de una PLA para realizar con biestables T el GR de la figura 7.10*a*. (*Nota*: $\theta_1 \equiv \theta_a$.)

† Como se recordará (figura 6.20) la salida de un biestable T (*toggle*) se complementa cada vez que el pulso de reloj encuentre $T = 1$. En caso contrario, si $T = 0$ el biestable memoriza el estado anterior. Dicho de otro modo, la evolución $Q \rightarrow Q^+$ requiere $T = Q \oplus Q^+$ (*ecuación de excitación*).

A modo de conclusión, es interesante resaltar que desde el punto de vista de las técnicas de diseño, no se han utilizado más que las *tablas de verdad*, que especifican las funciones que se desean realizar, y las *ecuaciones de excitación* de los biestables que se utilizan. Las *tablas de programación* se obtienen fácilmente a partir de la información anterior.

EJERCICIO. Compruébese, sobre el ejemplo considerado (figura 7.10*a* y tabla 7.4), que al utilizar biestables T, no es posible, en general, reducir la longitud de la palabra (§7.2.3.1) utilizando como variables intermedias aquellas que permiten obtener el estado sucesor (θ_2, θ_1). ¿Por qué?

EJERCICIO. Propónganse soluciones para resolver el problema de la inicialización del sistema. ¿Facilita la tarea el asignar el código 0...0 al estado inicial?

7.3.2 Reducción del número de variables de entrada a la memoria o matriz lógica en los métodos de transición directa

Las ideas que se presentan en este apartado permiten establecer de forma gradual la evolución de las máquinas lógicas de transición directa hacia las que utilizan un secuenciador. En todo momento se persigue reducir la ocupación total de memoria. En §7.3.2.1 se aborda una optimización, válida en gran número de casos, de los esquemas propuestos en §7.3.1. En §7.3.2.2 se presenta, dentro de una continuidad en la evolución conceptual, una máquina de transición directa, pero que admite una inmediata interpretación en términos algorítmicos (no lógicos).

7.3.2.1 *Planteamiento básico*

Hasta ahora, el número de variables de entrada a la PLA o ROM utilizadas es $\lceil \log_2 n \rceil$, para codificar el estado (número mínimo), más una variable por entrada al sistema. En el ejemplo que se viene considerando (figura 7.10*a*) se tienen $2 + 4 = 6$ variables de entrada. Ello quiere decir que, si por ejemplo, se utiliza una ROM, son necesarias $2^6 = 64$ palabras de 4 bits (dos para el estado, y otros dos para b y d, puesto que $a = q_1^+$ y $c = b$).

A tenor de lo enunciado anteriormente, es fácil comprender que para grandes sistemas sea muy importante comprimir el número de variables de entrada a la memoria (ROM o PLA). La técnica que se propone a continuación se basa en la importantísima observación siguiente: el número de variables de entrada a un sistema secuencial que, a partir de un estado, intervienen en la elaboración de la función de transición † es, normalmente, muy inferior al número total de variables de entrada al sistema.

Así, la figura 7.10*b* representa las variables de entrada relevantes para cada estado. El valor máximo, v, es 2, mientras que el número de variables de entrada es 4, $\{A, B, C, D\}$.

A partir de la observación anterior, la idea que se persigue es, mediante *circuitos de selección*, extraer el subconjunto de variables de entrada que interesa a partir de un estado dado. Un circuito estándar de seleccción es el conocido *multiplexor*, MUX.

---

† Si se realizan directamente máquinas de MEALY, hay que considerar también la *función de lectura* o *salida*.

La figura 7.11 presenta un posible esquema general basado en multiplexores. Las salidas de los multiplexores atacan a la ROM, mientras que la variable seleccionada por cada MUX es especificada mediante un campo añadido a la palabra.

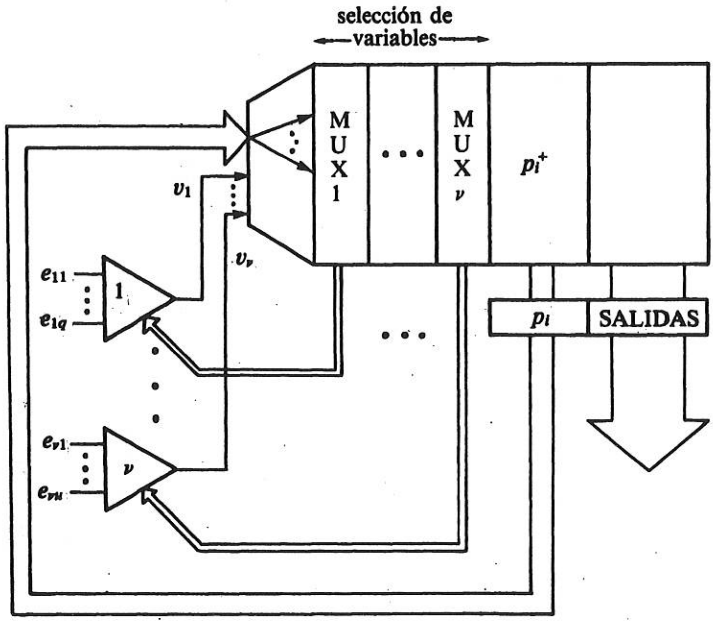


Figura 7.11. Un primer esquema para reducir el número de variables de entrada a la memoria (o matriz lógica).

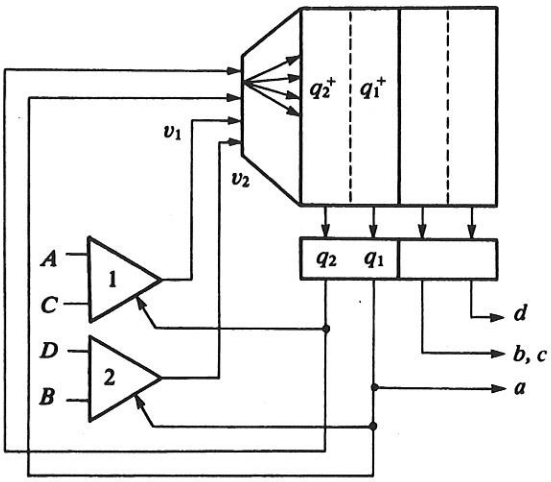
En resumen, con el esquema propuesto se reduce el número de variables de entrada a la ROM o PLA a costa de incrementar, en general, la longitud de la palabra y de utilizar circuitos selectores (por ejemplo, multiplexores). Si, como se indica en la figura 7.11, el código asociado a p_i forma la parte alta de la dirección, las ν variables definen 2^ν posiciones consecutivas, en las que se representan todas las transiciones posibles a partir de p_i.

La realización con multiplexores, conectando a cada uno de ellos todas las variables de entrada, es una solución viable. Ahora bien, no todas las variables tienen que ir a cada multiplexor. La minimización del número de variables de entrada por multiplexor es doblemente interesante puesto que de este modo:

1) se minimiza su complejidad (multiplexores más pequeños) y,
2) se reduce la longitud de la palabra, a través de la reducción de los campos selectores (MUX1, . . . , MUXν, en la figura 7.11).

Toda solución aceptable para la distribución de las variables de entrada debe permitir que las ν_i variables relevantes para el i-ésimo estado puedan ser seleccionadas simultáneamente. La solución ideal, si es posible, es aquella en la que cada variable

se asocia a un único multiplexor. Considerando el ejemplo de la figura 7.10, si conectamos A a MUX1, las variables B y D deben ir a MUX2. Afortunadamente B y D no deben considerarse simultáneamente. Por otro lado, dado que B va a MUX2, C debe ir a MUX1†. La figura 7.12 presenta una posibilidad de conexión. Ésta, unida a la codificación del estado que se ha elegido, permiten fusionar las funciones q_2^+ y MUX1, por una parte, y q_1^+, MUX2 y a, por la otra. La figura 7.12*b* muestra el contenido de la ROM, suponiendo que los registros de entrada y salida están realizados con biestables D.



(*a*) Esquema de la realización.

DIRECCIÓN		CONTENIDO	
q_2q_1	v_1v_2	$q_2^+ q_1^+$	b d
0 0	0 0	0 0	0 0
0 0	0 1	0 1	0 0
0 0	1 0	1 1	0 1
0 0	1 1	1 1	0 1
0 1	0 0	0 1	0 0
0 1	0 1	1 0	1 0
0 1	1 0	1 0	1 0
0 1	1 1	1 0	1 0
1 0	0 0	1 0	1 0
1 0	0 1	1 0	1 0
1 0	1 0	0 0	0 0
1 0	1 1	0 0	0 0
1 1	0 0	1 1	0 1
1 1	0 1	1 1	0 1
1 1	1 0	1 1	0 1
1 1	1 1	0 0	0 0

ESTADO PRESENTE — ENTRADAS PRESENTES — PRÓXIMO ESTADO — PRÓXIMAS SALIDAS

(*b*) Programa almacenado en la memoria.

Figura 7.12. Realización «optimizada» del GR de la figura 7.10. (*Notas*: (1) $p_1 = 01$, $p_2 = 10$, $p_3 = 00$ y $p_4 = 11$; (2) el estado presente selecciona, en este caso, a las variables v_1 y v_2. Además $a \equiv q_1$.)

EJERCICIO. Obténgase una tabla de programación de una PLA si los registros estuviesen realizados con biestables T.

EJERCICIO. Propóngase un método general para distribuir las variables de entrada entre los diferentes multiplexores. Aplíquese el método desarrollado al caso de la figura 7.10*a* suponiendo que el evento que provoca la evolución de p_2 a p_3 es AC. (*Nota:* este último caso demostrará que, a veces, es necesario repetir variables en 2 o más multiplexores.)

7.3.2.2 *Máquina de decisiones explícitas 2^v-arias*

Sea $\mathcal{E}$ el número de bits que codifica el estado ($\mathcal{E} = \lceil \log_2 n \rceil$), σ el número de salidas a realizar y θ la longitud total de los v campos selectores (campo de test). El esquema

† Obsérvese que nunca dos variables que interesan en un estado deben conectarse sólo a un mismo MUX. Esta prohibición es simétrica pero no transitiva, por lo que una vez más se define una relación de *compatibilidad*.

general de la figura 7.11 indica que la ocupación total de memoria para codificar un GR es: $2^{\mathcal{E}+\nu}(\theta + \mathcal{E} + \sigma)$. Utilizando este esquema se emplean 2^{ν} palabras para definir una dirección en la que se encontrará un estado sucesor.

Si se desea reducir la ocupación total de memoria, una forma de proceder es asignar a cada estado una *única palabra*. En este caso, para determinar la dirección asociada al estado sucesor basta con expresar explícitamente en cada palabra las 2^{ν} alternativas. La figura 7.13 presenta la idea en un caso en el que $\nu = 2$, cuando se pretende realizar el GR de la figura 7.10. Como puede observarse, esta máquina necesita, además de los elementos ya considerados en las anteriores, un *multiplexor* de 2^{ν} vías de $\mathcal{E}$ bits cada una. La ocupación memoria es, naturalmente, siempre inferior a la de los esquemas anteriores, ya que se tiene una ocupación:

$$2^{\mathcal{E}}(\theta + 2^{\nu}\mathcal{E} + \sigma) < 2^{\mathcal{E}+\nu}(\theta + \mathcal{E} + \sigma).$$

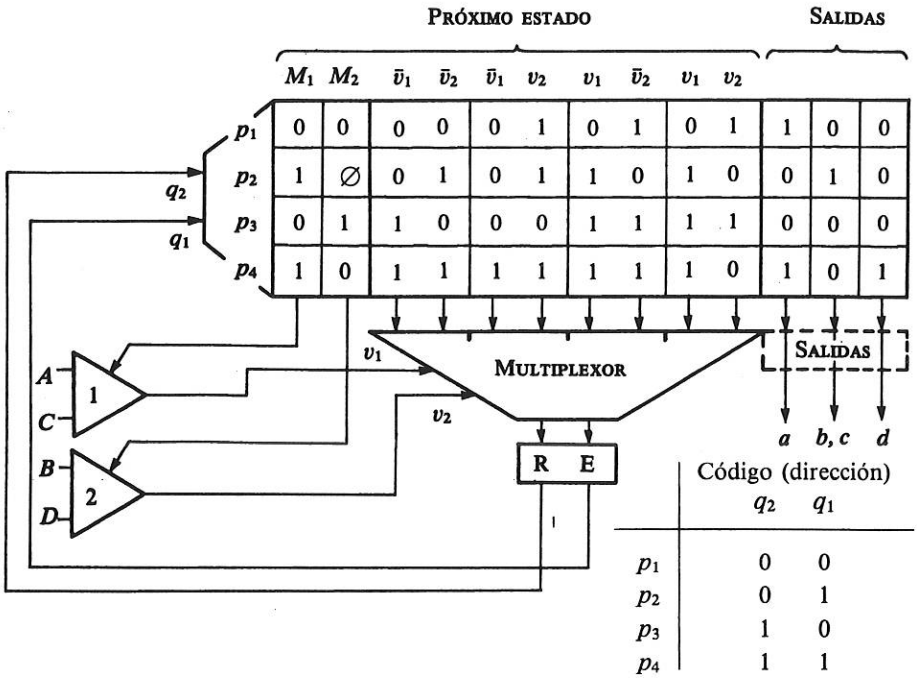


Figura 7.13 Máquina de decisiones explícitas cuaternarias (la existencia de un registro de salidas provocaría el retraso de éstas con respecto al estado en un ciclo de reloj). (*Nota.* ∅ = 0 o 1, indiferente.)

Antes de continuar, conviene resaltar que al adoptar la estructura de la figura 7.13, en una palabra coexisten las direcciones (códigos) de los estados sucesores con las acciones del estado actual. Dicho de otro modo, en esta máquina, la existencia del registro de salidas provoca un retraso de éstas de un ciclo de reloj. Una solución válida, pero muy costosa, para evitar el retraso de las salidas podría ser la utilización de 2^{ν} campos de salidas (asociados a cada uno de los estados sucesores) más el co-

rrespondiente multiplexor. Otra solución, mucho más económica, consiste simplemente en eliminar el registro de salidas, lo cual se puede hacer ahora sin problemas mayores ya que las variables de entrada al sistema *no forman parte de la dirección de la memoria.*

Una cuestión interesante en la máquina que nos ocupa es la interpretación algorítmica que se le puede asignar a cada una de las palabras (estados). Si consideramos la figura 7.13, la primera palabra asociada a p_1, significa (el símbolo «||» quiere decir «en paralelo»):

p_1: **caso de** v_1-v_2 **donde** $v_1 = A$ y $v_2 = B$
0-0: **ir a** p_1
0-1: **ir a** p_2
1-0: **ir a** p_2
1-1: **ir a** p_2
fin del caso || **hacer** $a := 1$, $b := 0$, $d := 0$;

En resumen, con la máquina de decisiones explícitas 2^v-arias se observa que:

1) Se necesita una por cada 2^v palabras de las máquinas del §7.3.2.1, pero éstas son más largas.
2) Cada palabra representa un estado del GR. Dicho de otra forma, el código del estado es, simultáneamente, la *dirección* de la palabra donde se encuentra definido.
3) Toda palabra se interpreta como un **caso de** (define un salto múltiple) y un **hacer** (define las acciones) a ejecutar en paralelo.
4) El estado siguiente se obtiene en un único ciclo de reloj, por lo que es un método de realización de transición directa.

EJERCICIO. (1) Cámbiese la conexión de B y D a MUX1 (figura 7.13). Compruébese que la longitud de la palabra, en este caso, puede reducirse inmediatamente en 4 bits. Además, la utilización de dos inversores permite eliminar otros dos bits más.

(2) Asígnesele otro código (dirección) a los estados, p_i, de forma que el campo de test (M_1-M_2) se pueda fusionar con el $\bar{v}_1\bar{v}_2$. ¿Por qué es posible? ¿Qué otras reducciones son posibles en este caso?

7.3.3 Métodos básicos con secuenciador: Máquinas de decisiones binarias

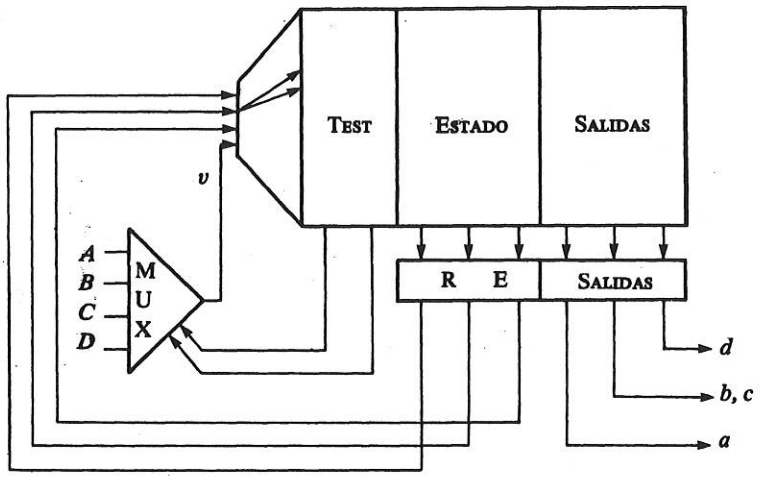
A lo largo de §7.3.2 se adoptó un tipo de realización que emplea multiplexores con lo que se pueden seleccionar simultáneamente todas las variables que interesen, en cualquier estado. A continuación se aborda la realización cuando se utiliza un único multiplexor, por lo que básicamente se ejecutarán *decisiones binarias* (**si** v **entonces/si no**).

Como se verá, las *máquinas de decisiones binarias* pueden realizar cualquier GR aunque para evaluar funciones complejas se requieran varias instrucciones de decisión. Las máquinas de decisiones binarias, dadas su simplicidad de realización y potencia, constituyen un estándar de amplia utilización. Normalmente, la ocupación memoria total es reducida.

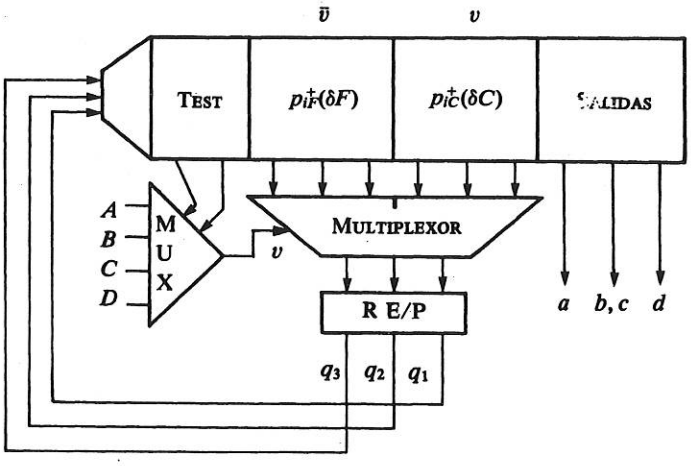
Estructuralmente, las máquinas de decisiones binarias básicas comprenden: (1) el *multiplexor* para las entradas, (2) la *memoria* o *matriz lógica* y (3) el *secuenciador.*

El secuenciador es un conjunto de elementos lógicos con los que se calcula la dirección de la próxima palabra (instrucción) que debe ser leida (determina la secuencia de instrucciones). Básicamente suele comprender un *registro* o *contador de programa* más cierta lógica en la que, a veces, se encuentra un *selector* (multiplexor) de direcciones.

El GR de la figura 7.10*a* ($v = 2$) no puede realizarse directamente con la máquina de la figura 7.14*a* (ésta es estructuralmente análoga a las de las figuras 7.11 y 12, pero con un único multiplexor). En §7.3.3.1 se considera la necesaria transformación. Los apartados que siguen presentan diversos modelos de máquinas de decisio-



(*a*) *v* permite definir dos direcciones consecutivas



(*b*) Máquina básica de decisiones binarias

Figura 7.14. Máquinas con un único multiplexor.

nes binarias. En §7.3.3.2 se considera el modelo básico, a partir del que se pueden obtener otros, mediante transformaciones que tienden a reducir la longitud de la palabra. En cualquier caso, si para el GR que se desea realizar $v > 1$, la realización con cualquier máquina de decisiones binarias ha de proceder en secuencia; es decir, la transición entre estados no será directa sino que se hará en pasos sucesivos (lecturas de la memoria o matriz lógica).

7.3.3.1 *Grafo reducido equivalente con alternativas binarias: transformación*

Para que un GR pueda realizarse directamente con la máquina de la figura 7.14*a*, es necesario que el GR se transforme en uno *equivalente*, $\overline{\text{GR}}$, para el que $v = 1$. Si $v = 1$, el $\overline{\text{GR}}$ debe ser tal que:

1) Todo lugar posea dos transiciones de salida. Es decir, todo lugar sea una selección binaria.
2) Los eventos asociados a las dos transiciones de una selección sean una variable y su complemento: v y $\bar{v}$.

De acuerdo con lo establecido, la transformación de GR en $\overline{\text{GR}}$ debe realizarse de modo que:

1) si un estado es una selección de orden superior a dos, se descomponga en cadenas de selecciones binarias.
2) todo evento no elemental se descomponga en una cadena de decisiones binarias.

Más que proceder a la presentación de técnicas formales de transformación†, apelamos a la intuición del lector quien podrá constatar fácilmente que los GR de las figuras 7.10*a* y 7.15*a* son equivalentes y el segundo de ellos cumple las condiciones anteriores (para éste, $v = 1$). La figura 7.15*b* presenta un organigrama «*equivalente*» al GR de la figura 7.15*a*. En cualquier caso, en ésta se puede observar que los estados p_1, p_3 y p_4 del GR original han sido descompuestos en dos cada uno (p_{11} y p_{12}; p_{31} y p_{32}; p_{41} y p_{42}). Evidentemente, siempre se puede afirmar que las transformaciones anteriores *nunca disminuyen* el número de estados.

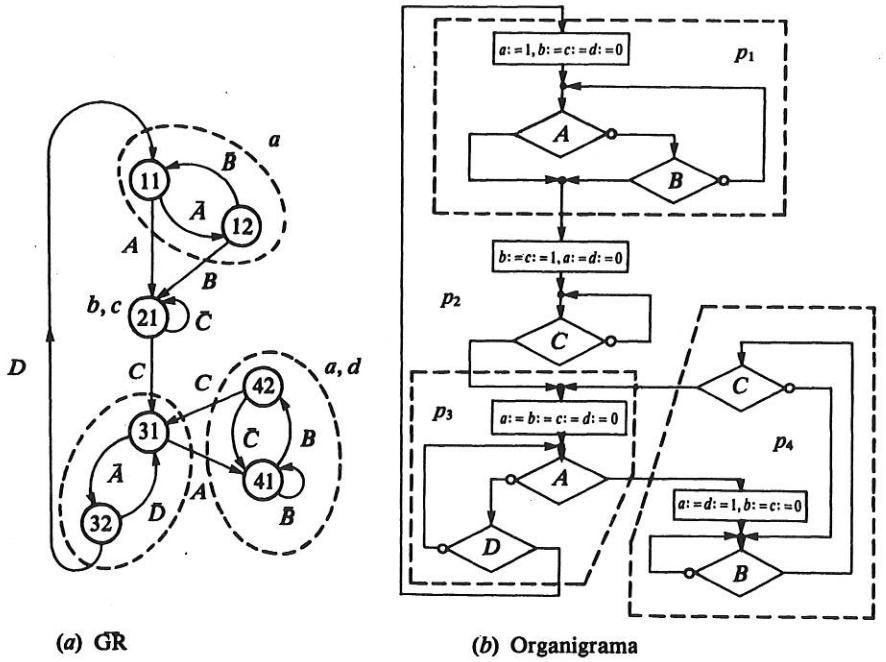
La realización del $\overline{\text{GR}}$ con la máquina de la figura 7.14*a* es inmediata, puesto que $v = 1$. (EJERCICIO.) Obsérvese en este punto que la descomposición de los estados de GR conduce a evaluaciones en secuencia de las condiciones de evolución. Es decir, en general las transiciones no serán directas, sino que se realizarán tras una *serie de ciclos de reloj*.

7.3.3.2 *Instrucción única y direcciones explícitas*

La eliminación de v (figura 7.14*a*) en el direccionamiento de la memoria, utilizando la aproximación presentada en §7.3.2.2, conduce a la máquina de la figura 7.14*b*.

---

† En definitiva, el problema que subyace es la realización de conjuntos de funciones booleanas (para cada estado se han de realizar los eventos asociados a sus transiciones de salida) mediante programas de decisiones binarias. Las referencias [SILV 78, 79*b*, 82*b*], [MANG 82], [THAY 81] consideran el mencionado problema, cuyo tratamiento formal y optimizado escapa a los objetivos de este texto.



(*a*) GR (*b*) Organigrama

Figura 7.15. Grafo y organigrama «equivalentes» al GR de la figura 7.10a.

En ésta, cada palabra (microinstrucción) se interpreta como la ejecución en paralelo de:

p_i: **si** $\bar{v}$ **entonces ir a** p_{iF}^{+} **si no ir a** p_{iC}^{+} || **hacer** «salidas»

Representando la anterior instrucción de forma abreviada por «p_i:$v|p_{iF}^{+}|p_{iC}^{+}$|| salidas activas; », el $\overline{\text{GR}}$ de la figura 7.15*a* (figura 7.10*a*) se podría programar como sigue:

p_{11}:$A|p_{12}|p_{21}||a; c$;
p_{12}:$B|p_{11}|p_{21}||a$;
p_{21}:$C|p_{21}|p_{31}||b, c$;
p_{31}:$A|p_{32}|p_{41}$;
p_{32}:$D|p_{31}|p_{11}$;
p_{41}:$B|p_{41}|p_{42}||a, d$;
p_{42}:$C|p_{41}|p_{31}||a, d$;

Evidentemente, toda ordenación de las instrucciones es correcta, puesto que siempre se designa explícitamente la instrucción que se debe ejecutar a continuación.

Para terminar con el comentario sobre la máquina de la figura 7.14*b*, basta señalar que el *registro de estado* (RE/P) juega el papel de *puntero del microprograma*, puesto que designa la próxima microinstrucción a ejecutar. Dicho de otro modo, los conceptos de *estado* del $\overline{\text{GR}}$ y de *dirección del microprograma* convergen. El *registro de estado* se convierte en, *registro de programa* o *microprograma*.

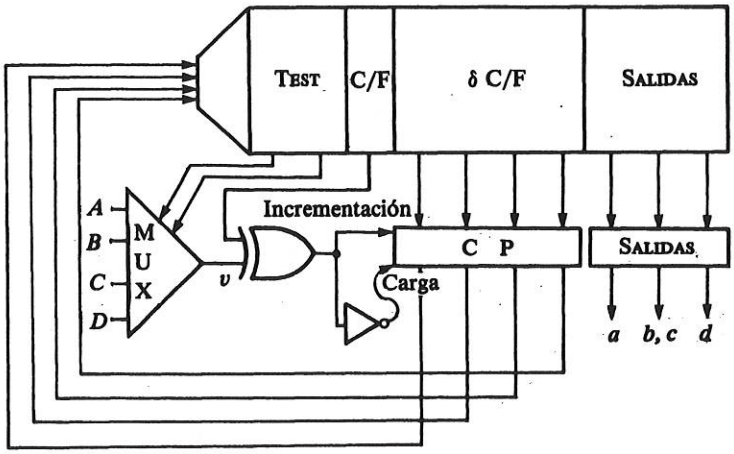


Figura 7.16. Instrucción única y una dirección implícita. (*Nota*. CP es un contador up/down con carga paralela.)

Con objeto de reducir la longitud de la palabra de que se dispone, se pueden adoptar dos transformaciones. Éstas conducen a las máquinas de las figuras 7.16 y 7.17. La aplicación conjunta de ambas transformaciones conduce a la máquina de la figura 7.18. La reducción de la longitud de la palabra supondrá siempre un incremento (aunque con frecuencia relativamente menor) en la longitud del microprograma.

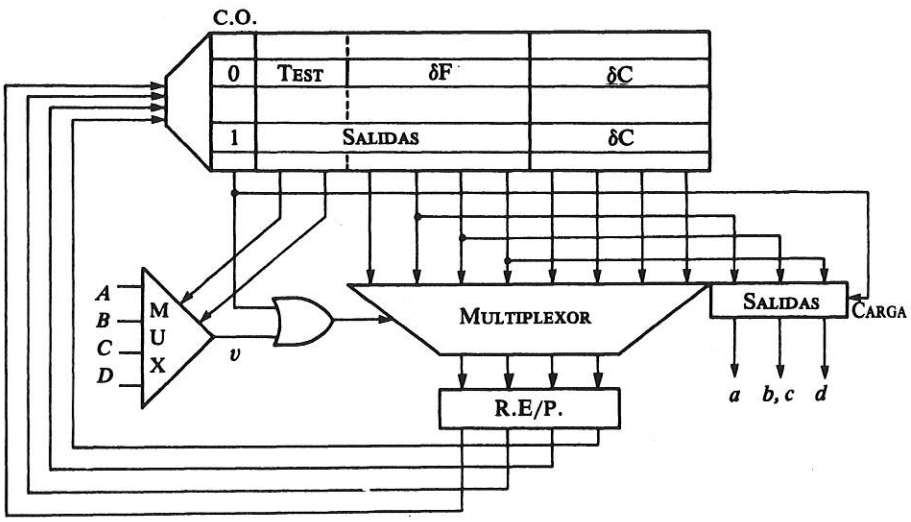


Figura 7.17. Dos instrucciones y direcciones explícitas.

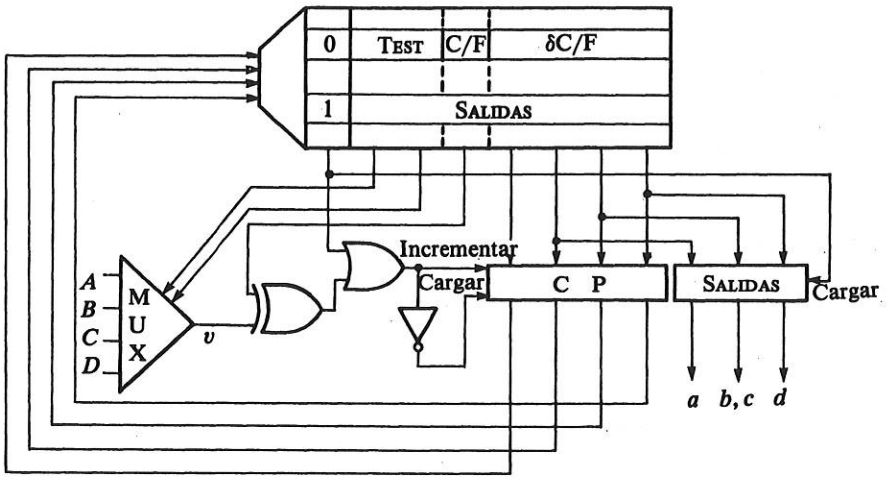


Figura 7.18. Dos instrucciones y dirección implícita.

7.3.3.3 *Instrucción única y una dirección implícita*

Una forma posible de reducir la longitud de la palabra usada en la máquina de la figura 7.14*b* consiste en predefinir una de las dos direcciones (*cierto*, δC, o *falso*, δ*F*) como la que sigue a la actual. Esta reducción de la palabra es tanto más interesante cuanto que en la práctica se suele constatar que los $\overline{\text{GR}}$ contienen partes lineales importantes. En este caso, el registro de estado o programa se convierte realmente en un *contador de programa*, sobre el que se podrán realizar operaciones de *carga* o *incrementación*.

La figura 7.16 presenta una posible estructura de máquina en la que sólo es digno de especial comentario la existencia de un campo adicional, (CIERTO/FALSO), que completa el de test y que permite definir el test sobre v (bit C/F = 1) ó $\bar{v}$ (bit C/F = 0). La restringida capacidad de salto en este tipo de máquina hará que aparezca la necesidad de un *salto incondicional*. Dicho de otra forma, las instrucciones de salto incondicional aparecen como una penalización debida a la eliminación de uno de los dos campos de salto explícito existentes en la máquina de la figura 7.14*b*.

El $\overline{\text{GR}}$ de la figura 7.15*a* puede codificarse como sigue:

p_{11}:$A|p_{21}||a$; p_{33}:$\bar{D}|p_{31}$; {simula un salto incondicional}
p_{12}:$\bar{B}|p_{11}||a$; p_{41}:$\bar{B}|p_{41}||a$, d;
p_{21}:$\bar{C}|p_{21}||b, c$; p_{42}:$C|p_{31}||a, d$;
p_{31}:$A|p_{41}$; p_{43}:$\bar{C}|p_{41}||a, d$; {simula un salto incondicional}.
p_{32}:$D|p_{11}$;

En este programa se ha de resaltar la introducción de las instrucciones (estados) p_{33} y p_{43} que simulan un salto incondicional al comprobar el estado del complemento de las variables consideradas en p_{32} y p_{42}, respectivamente.

Observación. El «truco» utilizado en el programa anterior para simular el salto incondicional no es formalmente correcto puesto que la variable de entrada designada puede cambiar efectivamente su estado lógico al irse a ejecutar la segunda de las instrucciones. Para evitar este problema, normalmente se añade al multiplexor una entrada con la constante «1» lógico. La designación de la entrada correspondiente provocará indefectiblemente el salto.

Dado que nunca se designan explícitamente las instrucciones sucesoras, a diferencia del programa presentado en §7.3.2, en este caso no es válida cualquier ordenación de las microinstrucciones. Por otro lado, es importante señalar que el número de saltos incondicionales a insertar depende de la ordenación de estados elegida al escribir el microprograma. De este modo, se puede comprobar fácilmente (EJERCICIO) que la ordenación:

1) p_{32}-p_{11}-p_{12}-p_{21}-p_{31}-p_{41}-p_{42}, requiere un único salto incondicional al final del programa (p_{43}:$\bar{C}|p_{41}||a, d$;).
2) p_{11}-p_{12}-p_{31}-p_{32}-p_{21}-p_{41}-p_{42}, requiere cuatro saltos incondicionales que son:

   p_{13}:$B|p_{21}||a$; {después de p_{12}}
   p_{22}:$C|p_{31}||b, c$; {después de p_2}
   p_{33}:$\bar{D}|p_{31}$; {después de p_{32}}
   p_{43}:$\bar{C}|p_{41}||a, d$; {después de p_{42}}.

El análisis de las diferentes ordenaciones anteriores permite concluir fácilmente sobre el número de saltos incondicionales que hay que insertar. Éste es igual al *número de subsecuencias que la ordenación considerada define en* GR. Así, por ejemplo, p_{11}-p_{12}-p_{21}-p_{31}-p_{32} forman una subsecuencia (camino según la terminología de la teoría de grafos, anexo 2) que se acaba al aparecer p_{41}, puesto que éste no es descendiente directo de p_{32}; por ello se introduce el salto incondicional p_{33}. Habida cuenta que en GR (figura 7.15*a*), p_{32}-p_{11}-p_{12}-p_{21}-p_{31}-p_{41}-p_{42} forman una única secuencia, sólo se necesita un salto incondicional.

En función de lo establecido, la *minimización* del número de saltos incondicionales se resuelve obteniendo una ordenación en la que exista un mínimo de subsecuencias disjuntas que cubran todos los estados†. La obtención de un resultado óptimo es factible, pero costosa. Afortunadamente, no es difícil definir heurísticas que conduzcan a soluciones *subóptimas*.

7.3.3.4 *Dos instrucciones y direcciones explícitas*

Otra forma de reducir la longitud de la palabra usada en la máquina de la figura 7.14*b*, alternativa a la presentada en §7.3.3.3, consiste en descomponer la microinstrucción en dos (figura 7.17):

1) microinstrucción de *salto condicional*:

si $\bar{v}$ **entonces ir a** δF **si no ir a** δC;

† En términos de la *teoría de grafos*, este problema se enuncia como: obtención de una partición de cardinalidad mínima de los nudos del grafo (estados) en caminos.

2) microinstrucción de *asignación y salto incondicional*:

hacer «salidas»|| **ir a** δC;

Esta descomposición suele ser útil, porque en la práctica se observa que, con frecuencia, las acciones o salidas se mantienen en varias microinstrucciones ejecutadas en secuencia (por ejemplo p_{31} y p_{32} en el microprograma de §7.3.3.2).

La diferenciación entre los dos tipos de microinstrucciones se puede realizar gracias al contenido de un bit. Surge el concepto de *código de operación*.

El GR de la figura 7.15a se puede programar de la forma siguiente (**H** = microinstrucción de asignación, **S** = microinstrucción de salto):

p_{10}:**H**(a||p_{11}); p_{30}:**H**(||p_{31});
p_{11}:**S**(A|p_{12}|p_{20}); p_{31}:**S**(A|p_{32}|p_{40});
p_{12}:**S**(B|p_{11}|p_{20}); p_{32}:**S**(D|p_{31}|p_{10});
p_{20}:**H**(b, c||p_{21}); p_{40}:**H**(a, d||p_{41});
p_{21}:**S**(C|p_{21}|p_{30}); p_{41}:**S**(B|p_{41}|p_{42});
p_{42}:**S**(C|p_{41}|p_{30}).

Como se puede observar, se han añadido microinstrucciones (estados) p_{i0} en las que se posicionan las salidas. Este programa realiza directamente el organigrama de la figura 7.15*b*. Habida cuenta que cada microinstrucción indica explícitamente su sucesora(s), cualquier ordenación de las microinstrucciones es válida.

Para concluir los comentarios sobre esta máquina, es preciso poner de relieve que la descomposición de la microinstrucción primitiva en una de *decisión* (**si**) y otra de *acción* (**hacer**), permite separar netamente las fases de evaluación de funciones y de asignación de valores. De este modo, se facilita la realización secuencial de funciones lógicas combinatorias o de máquinas de MEALY.

7.3.3.5 *Dos instrucciones y direcciones implícitas*

Aplicando las transformaciones de *secuencialización* (§7.3.3.3) y *descomposición* (§7.3.3.4) a la microinstrucción básica (§7.3.3.2), se obtienen las microinstrucciones que siguen:

1) microinstrucción de *salto condicional*:

Δ: **si** $v \oplus b_{CF}$ **entonces ir a** δC/F **si no ir a** $\Delta + 1$;

2) microinstrucción de *asignación*:

Δ: **hacer** «salidas»;

La figura 7.18 presenta una máquina capaz de ejecutar las microinstrucciones presentadas.

EJERCICIO. Escríbase un microprograma que realice con la máquina de la figura 7.18 el GR de la figura 7.15*a*. (*Observación*. Dado que se trata de una máquina con direcciones implícitas, es importante elegir una ordenación de microinstrucciones válida y económica.)

EJERCICIO. Simplifíquese la máquina de la figura 7.18 si se supone que se elimina la posibilidad de saltar cuando $v = 0$ (salto si falso).

EJERCICIO. Compruébese que la máquina simplificada conduce, normalmente, a microprogramas más largos. (*Sugerencia.* Escríbase un microprograma que realice el GR de la figura 7.15*a*.)

7.3.3.6 *Realización asíncrona-autosincronizada: ejemplo*

Las diferentes máquinas para la realización de sistemas secuenciales presentadas a lo largo de §7.3 son *síncronas*. Es decir, se ha supuesto que el *reloj* que acompasa el desarrollo de las instrucciones es de *origen externo* y frecuencia fija. La máquina que se presenta en este apartado posee un funcionamiento síncrono, pero el reloj es generado por ella misma, a partir de los eventos que provienen del sistema que se desea controlar. Este tipo de funcionamiento se suele denominar *asíncrono-autosincronizado*, o simplemente *autosincronizado*.

La figura 7.19 presenta un ejemplo de máquina. Sus elementos principales son: (1) *la memoria* ROM, (2) *la matriz de receptividad* y (3) *el secuenciador* (registro, generador del reloj y lógica adicional).

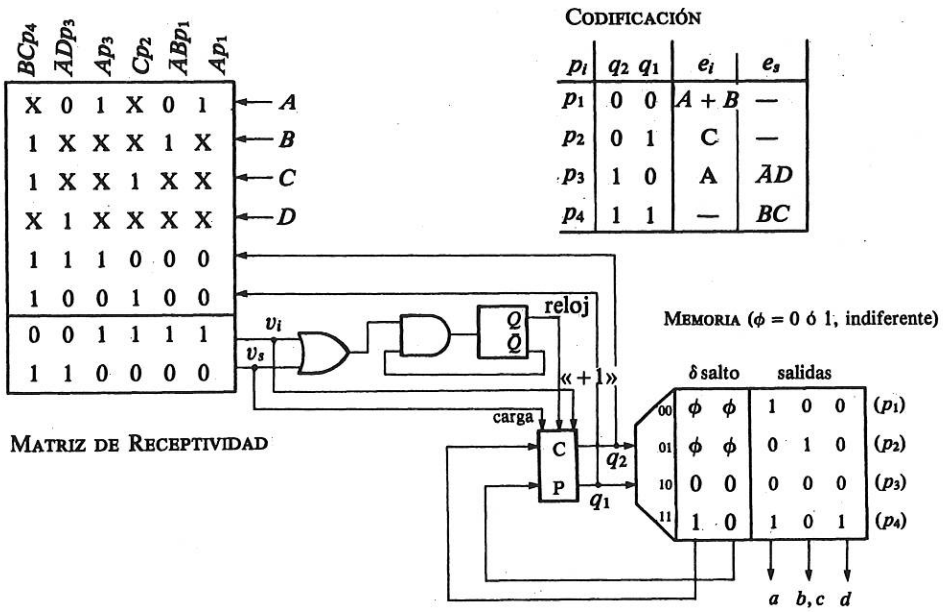


Figura 7.19. Máquina asíncrona-autosincronizada, programada para realizar el GR de la figura 7.10.

La programación de los dos primeros elementos define el GR que se realiza. La matriz de receptividad (normalmente realizada con una PLA), tiene como misión el generar dos variables lógicas disjuntas a partir del estado, q, y de las variables de entrada: (1) variable de *incremento*, v_i, y (2) variable de *salto*, v_s ($v_i v_s = 0$).

Las variables v_i y v_s vienen a sustituir a la variable v que aparecía en los sistemas anteriores; v_i y v_s representan los resultados de dos tests disjuntos e independientes. Si $v_s = 1$, la próxima microinstrucción que se debe ejecutar estará definida por un campo de la memoria, denominado *dirección de salto*, δ. Si $v_i = 1$ la próxima microinstrucción que se ejecutará es la siguiente en la memoria. Las salidas emitidas son las correspondientes a la microinstrucción que se ejecuta (estado actual). La activación de v_i o de v_s indica al secuenciador la necesidad de proceder a una evolución hasta que se alcance el estado (microinstrucción) final correspondiente. Cuando $v_i + v_s = 0$, el secuenciador permanece en reposo. Es decir, la máquina no evoluciona en permanencia como ocurre con todos los métodos presentados anteriormente. Ésta es una diferencia importante con las máquinas síncronas.

De acuerdo con lo enunciado, «se puede» resumir la significación de una microinstrucción de la forma siguiente ($v_i v_s = 0$):

si v_s **entonces** CP : = δ
si v_i **entonces** CP : = CP + 1
si $\bar{v}_s \bar{v}_i$ **entonces** CP : = CP.

} ||**hacer** «salidas»;

El formato de la q-ésima microinstrucción, programada sobre dos componentes independientes, puede «interpretarse» de esta forma:

Campo de:	test «s»	test «i»	dirección de salto	acciones
	qe_s	qe_i	δ	SALIDAS
	v_s	v_i		
	MATRIZ DE RECEPTIVIDAD		MEMORIA	

Como se puede comprobar, este es un esquema del tipo «*instrucción única y una dirección implícita*» (§7.3.3.3), aunque bastante más elaborado.

Para microprogramar un GR, la única transformación previa necesaria (figura 7.20) es la que conduce a otro equivalente, GR, tal que las selecciones sean de orden no superior a dos (sólo se puede ir a la siguiente microinstrucción o a la de salto). Dado que una de las direcciones de la decisión es implícita, la ordenación de los estados tiene su importancia a la hora de reducir el número de saltos incondicionales†.

La figura 7.19 muestra una forma de realizar el GR de la figura 7.10.

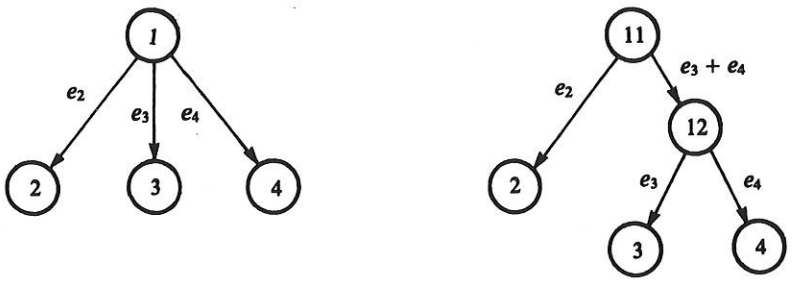
En resumen, la realización presentada en este apartado:

(1) es asíncrona-autosincronizada.

---

† En [AND 76] se presentan algoritmos para la ordenación. Éstos, dada la mayor flexibilidad de la presente máquina con respecto a la del §7.3.3.3, son netamente más simples.

(2) trabaja en base a selecciones binarias en las que existen dos condiciones de test que pueden ser funciones complejas (la matriz de receptividad las calcula): v_i y v_s. El GR debe ser transformado (figura 7.20).
(3) adopta el principio de utilizar una dirección implícita, por lo que interesa determinar una ordenación adecuada para las microinstrucciones, que reduzca los saltos incondicionales. (*Nota.* Las dos consideraciones anteriores llevan a la conclusión de que este método no es, en general, de transición directa.)
(4) difiere, desde el punto de vista material, de la realización de la figura 7.16 en la presencia del generador interno del reloj y en la matriz de receptividad (ésta sustituye al multiplexor).



(*a*) GR (*b*) $\overline{\overline{\text{GR}}}$

Figura 7.20. Transformación de un grafo, GR, en otro cuyas selecciones sean de orden no superior a dos.

EJERCICIO. La matriz de receptividad y la memoria (figura 7.19) pueden fusionarse en una única matriz lógica programable, puesto que la entrada a la memoria ($q_2 - q_1$) ataca también a la matriz de receptividad. Obténgase, para el GR de la figura 7.10*a*, el contenido de la PLA, cuando se hace desaparecer la memoria. Compárese este último tipo de realización con los presentados en §7.3.1 (tablas 7.3 y 4).

7.3.4 A modo de resumen

La realización de grafos reducidos se ha ido desarrollando a partir de métodos puramente *lógicos* (§7.3.1) hasta llegar a métodos en los que el planteamiento más natural es de tipo *algorítimico* (§7.3.3).

Desde el punto de vista de las técnicas de diseño, en §7.3.1 no se utilizan fundamentalmente más que las *tablas de verdad*, las *ecuaciones de excitación de los biestables* y, sólo a veces, el *álgebra de* BOOLE. El paso esencial desde los métodos cableados (capítulo 6) reside en la implementación directa de las tablas, no considerándose el nivel de puertas lógicas.

En §7.3.2 se modificaron los esquemas básicos para disminuir el número de variables de entrada al macrocomponente programable (memoria o matriz lógica). Conceptualmente, la modificación de los esquemas del §7.3.1 posibilitan una aproxima-

ción, dentro de una cierta continuidad, a los métodos más informáticos de realización, con frecuencia denominados *microprogramados*.

El proceso seguido en el estudio de máquinas básicas de decisiones binarias (§7.3.3) se resume en la figura 7.21. La rápida presentación de una máquina autosincronizada, sirve como contundente ejemplo de que en este dominio, la principal limitación a la hora de concebir nuevas máquinas es la imaginación del diseñador. Para «aplicaciones especiales», es seguro que «máquinas especiales» conducirán a «las mejores» soluciones.

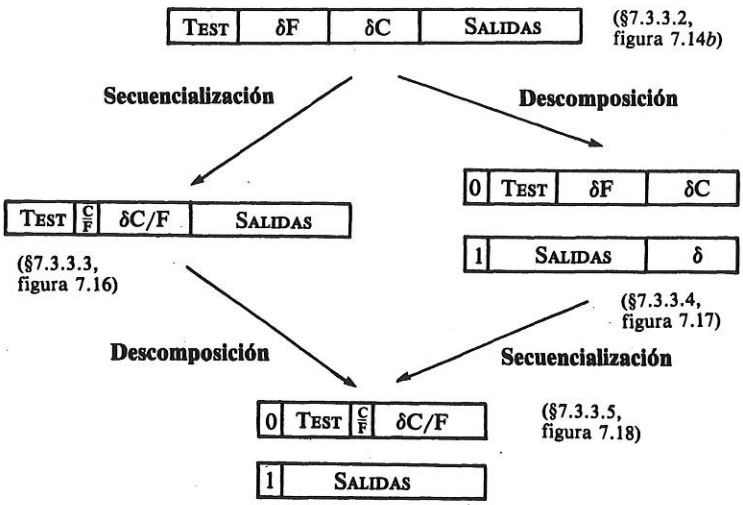


Figura 7.21. Relaciones entre los formatos de las microinstrucciones de las máquinas básicas de decisiones binarias.

Por último, es importante señalar el carácter básico del material presentado. Actualmente existen comercializados secuenciadores de potencia superior a los considerados. Así, por ejemplo, es frecuente encontrar en éstos facilidades como: (1) *llamadas y retornos de submicroprogramas* (el secuenciador suele contener una pequeña pila para las direcciones de las microinstrucciones); (2) *direccionamientos relativos* con lo que, entre otras propiedades, se puede reducir la longitud de la palabra (el secuenciador suele contener una unidad aritmética elemental); (3) *contadores de iteraciones*, para simplificar la microprogramación de bucles;...

7.4 REALIZACIÓN DE REDES DE PETRI

Una vez estudiada la realización de sistemas lógicos combinacionales y secuenciales, en este apartado se aborda la realización de sistemas concurrentes descritos con RdP. En lo sucesivo se considerarán únicamente RdP binarias. Redes con un reduci-

do número de lugares no binarios podrían ser transformadas en binarias, realizando la cuenta de marcas de estos lugares en una parte operativa (figura 2.19).

Para la realización directa de RdP binarias, a menudo se adoptará una simulación de su comportamiento basada en la inteconexión de diversos GR. Una discusión sobre técnicas especiales para la descomposición de RdP binarias en GR se presenta en §7.4.3.

7.4.1 Métodos lógicos de transición directa

La realización de RdP, igual que la de GR (§7.3), consiste en materializar:

1) el *marcado*, que representa el estado del sistema,
2) la función de *transición entre marcados* (estados),
3) la función de *salida*.

La realización de la función de salida no plantea problema específico alguno y podrá ser diseñada utilizando conceptos y técnicas presentadas anteriormente. Ahora bien, el *marcado de una RdP es en sí una codificación especial del estado interno del sistema*, por lo que su realización y la de su función de transición presentarán ciertos rasgos específicos.

La forma más directa de realizar una RdP binaria consiste en: (1) asociar un biestable a cada lugar y (2) materializar con una memoria o matriz lógica programable la función de transición; es decir, la activación y desactivación de dichos biestables. Este primer método posee la ventaja de permitir un diseño fácil y general, conceptualmente similar al presentado en el capítulo 6 (realización cableada modular), por lo que la realización de RdP no binarias es también inmediata†. Su principal inconveniente radica en el bajo rendimiento que se alcanza, normalmente, en la utilización de ROMs y PLAs. En efecto, baste constatar simplemente que la realización directa de la RdP de la figura 7.22 necesitará 12 entradas (5 entradas al sistema y 7 variables de marcado), 7 biestables [asociados a los lugares y cuyas salidas con las variables que representan el marcado, $M(p_j)$] y 11 salidas (dado que $b \equiv c$ sólo hay 4 salidas del sistema; si los biestables fueran D o T, se necesitarán 7 funciones de excitación, pero si los biestables fuesen R-S o J-K, se necesitarían dos funciones de excitación por biestable, con lo que el número total de salidas a generar sería 18 en vez de 11).

Si n es el número de lugares de la RdP, $n = |P|$, el anterior esquema de codificación representa el marcado con n variables, $M(p_j)$. Para reducir de forma importante este número, se puede optar por una codificación *total* (§7.2.3.1) o *independiente por campos* (§7.2.3.2).

7.4.1.1 *Codificación total del marcado*

Toda realización basada en una codificación total del marcado se ha de desarrollar en dos fases:

1) Obtención de los diversos marcados (éstos son las configuraciones, según la terminología de §7.2.3) y las evoluciones. Dicho de otro modo, la obtención del GR equivalente a la RdP (§1.5.4).

---

† Basta con utilizar *contadores* en vez de *biestables*. La activación y la desactivación de un biestable se convierten en la incrementación y la decrementación de un contador.

2) Realización del GR obtenido, para lo que se pueden utilizar las técnicas presentadas en §7.3.

La realización basada en la codificación total del marcado circunscribe a las RdP al modelado, puesto que la realización contemplará al sistema como secuencial (GR). Desde un punto de vista práctico, la codificación total puede ser interesante con sistemas en los que no exista un paralelismo tal que conduzca a un número excesivamente grande de marcados.

Los mayores inconvenientes que presenta esta forma de proceder residen en la complejidad operatoria y en la escasa flexibilidad de las realizaciones (una pequeña modificación en la RdP puede provocar importantes cambios). En el caso de la RdP de la figura 7.22, el GR equivalente tiene 10 estados, de donde son necesarias 4 variables de estado, como mínimo.

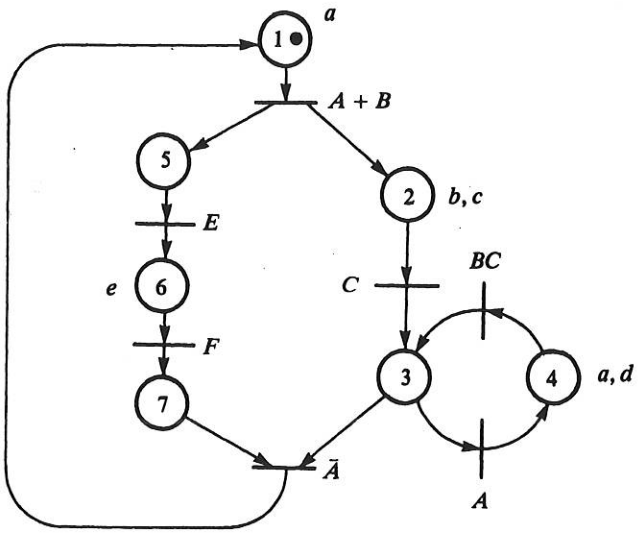


Figura 7.22. Red de Petri que se desea realizar.

7.4.1.2 *Codificación independiente por campos del marcado*

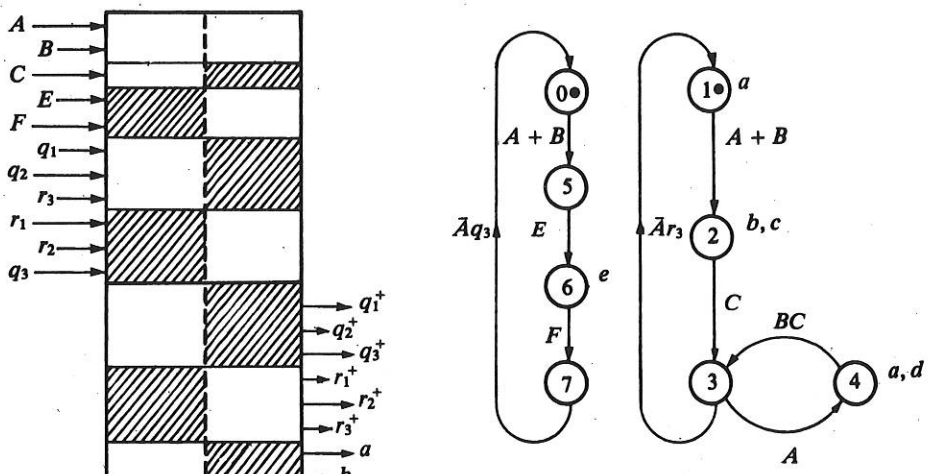
Después de obtener los marcados alcanzables en la RdP (es decir, las configuraciones del caso que nos ocupa), la codificación independiente por campos se puede llevar a término en dos fases (presentadas en §7.2.3.2):

1) Obtención de una *partición de las funciones* $M(p_j)$ de modo que en todo elemento de la misma, *campo*, se excluyan mutuamente los $M(p_j)$.
2) *Codificación independiente de cada uno de los campos* obtenidos.

A diferencia de lo presentado en §7.2.3.2, que se situaba en un contexto combinacional, aquí el próximo marcado depende del actual. Por consiguiente, la secuencia de valores que toma un campo es función del valor de los otros: existe una interdependencia en la evolución.

Volviendo a considerar la RdP de la figura 7.22, se puede observar que, por ejemplo, $\mathcal{P}_1 = \{M(p_1), M(p_2), M(p_3), M(p_4)\}$ y $\mathcal{P}_2 = \{M(p_5), M(p_6), M(p_7\}$ forman una partición de las funciones $M(p_j)$ tal que cada grupo está definido por lugares cuyos marcados están en exclusión mutua. En el caso que nos ocupa es fácil constatar que $\mathcal{P}_1$ y $\mathcal{P}_2$ son clases principales disjuntas y, además, $\mathcal{P}_1$ es una clase máxima de compatibilidad.

Codificando $\mathcal{P}_1$ y $\mathcal{P}_2$ con las variables intermedias q_2-q_1 y r_2-r_1, respectivamente, se obtendrán realizaciones con 9 entradas, 4 biestables y 8 salidas. (EJERCICIO. Obténgase una utilizando biestables D.)



(*a*) **PLA (las partes sombreadas están totalmente desaprovechadas).**

(*b*) **Grafos que simulan la RdP de la figura 7.22.**

Figura 7.23. Esquema de una posible realización de la RdP de la figura 7.22. (*Nota.* El marcado p_0 significa sólo que $\{p_5, p_6, p_7\}$ están desmarcados.)

La figura 7.23*a*, presenta «macroscópicamente» la PLA de otra realización en la que se generan adicionalmente dos funciones lógicas de *sincronización* (éstas son redundantes):

1) $q_3 = q_2\, q_1$ ($q_3 = 1$ sii p_3 está marcado: $q_3 = M(p_3)$),
2) $r_3 = r_2 r_1$ ($r_3 = 1$ sii p_7 está marcado: $r_3 = M(p_7)$).

Como se puede comprobar fácilmente, gracias a la adición de q_3 y de r_3, la PLA (figura 7.23*a*) más los correspondientes biestables realizan dos sistemas secuenciales que son prácticamente *disjuntos* (las salidas son disjuntas y las entradas comparten sólo A y B), pero *mutuamente sincronizados* (a través de las variables q_3 y r_3). La figura 7.23*b* representa los dos GR que, en este caso, simulan el comporta-

miento de la RdP. (El segundo de ellos, cambiando r_3 por D, es el utilizado en el §7.3, figura 7.10*a*.)

Una rápida consideración de la PLA de la figura 7.23*a* permite constatar que en gran parte su superficie está totalmente desaprovechada. Con el objeto de mejorar el rendimiento de utilización de los macrocomponentes, surge la idea de realizar la RdP *interconectando máquinas secuenciales*; cada una de ellas realizará un GR. De este modo, se puede obtener una realización con dos PLA, una de 6 entradas y 6 salidas y otra de 7 entradas y 4 salidas (verifíquese).

En resumen, al utilizar métodos lógicos de transición directa se observa que la codificación total del marcado conduce a realizar el GR equivalente a la RdP. La utilización de una codificación independiente por campos «equivale» a simular el comportamiento de la RdP mediante varios GR interconectados. (Se obtiene una descomposición de la RdP binaria en GR.) Más que realizar la RdP con una única PLA, normalmente será más económica la interconexión de máquinas secuenciales (basadas en PLAS más pequeñas) que realicen los diferentes GR. Si en vez de PLAS se utilizan ROMS, habida cuenta del crecimiento exponencial de su capacidad con el número de entradas, prácticamente siempre será más económico el realizar las RdP por interconexión de máquinas secuenciales.

A modo de observación final, interesa constatar que la realización de varios GR con una única ROM se lleva a cabo calculando el GR *producto*† de los que se desean realizar. En efecto, dado que para seleccionar una palabra, el decodificador de la ROM genera términos mínimos (disjuntos), la lectura de palabras es secuencial; por tanto, la realización de varios GR con una ROM se llevará a cabo como si se tratase de un único GR, el GR producto. Ésto constituye una diferencia fundamental con las realizaciones basadas en una PLA, puesto que en este último caso, pueden haber varios términos producto simultáneamente activos («lectura simultánea de varias palabras»). De lo anterior se desprende que si se desea realizar una RdP que presente evoluciones simultáneas con una única ROM, será más económica la utilización del esquema de codificación total (el GR equivalente a la RdP nunca tendrá más estados que el GR producto de los GR en que se descomponga la red).

EJERCICIO. Realícese con una única PLA de 8 entradas y 10 salidas la RdP de la figura 7.22 (*Sugerencia*: utilícense dos multiplexores aplicando el planteamiento básico sobre la reducción del número de variables de entrada, §7.3.2.1.)

EJERCICIO. (1) Descompóngase en GRs la RdP de la figura 7.25*a*. (2) Impóngase como restricción suplementaria a la descomposición el que cada salida deba ser generada por un único GR (partición de las salidas en GR).

7.4.2 Realización con máquinas de decisiones binarias

El funcionamiento de las máquinas de decisiones binarias (§7.3.3) es puramente secuencial, por lo que la realización directa de RdP con evoluciones paralelas no es posible, salvo si se microprograma el GR equivalente.

---

† Si GR_i tiene n_i estados, el grafo $GR = GR_1 \times GR_2$ poseerá, como máximo, $n_1 n_2$ estados (producto cartesiano de los estados de GR_1 y de GR_2).

Una segunda posibilidad para realizar una RdP consiste en interconectar tantas máquinas de decisiones binarias como GR se hayan obtenido al descomponerla. Para que este esquema funcione correctamente, deben *sincronizarse* las diferentes máquinas. La figura 7.24 muestra uno de los métodos más simples, utilizable con las máquinas de dos instrucciones presentadas en §7.3.3.4 y 5. La señal *comienzo de ciclo* establece la adquisición de las entradas al cargar un registro (se hace un muestreo de éstas) a la vez que se activan los biestables que autorizan la evolución de las máquinas (si un biestable está desactivado, la máquina correspondiente no evoluciona puesto que no le llegará su reloj, CK_i). Cuando la i-ésima máquina ejecuta una instrucción de salida (código de operación a 1, $C\emptyset_i = 1$), se desactiva el biestable de autorización de funcionamiento y queda bloqueada hasta que se genere el próximo comienzo de ciclo; es decir, hasta que todas las máquinas hayan ejecutado una microinstrucción de salida. Procediendo de este modo, si todas las máquinas son microprogramadas de forma que de no existir evolución en el GR que realiza, se ejecute la microinstrucción de salida asociada al estado, el funcionamiento global será correcto y se basará en una interpretación síncrona de la evolución de la RdP.

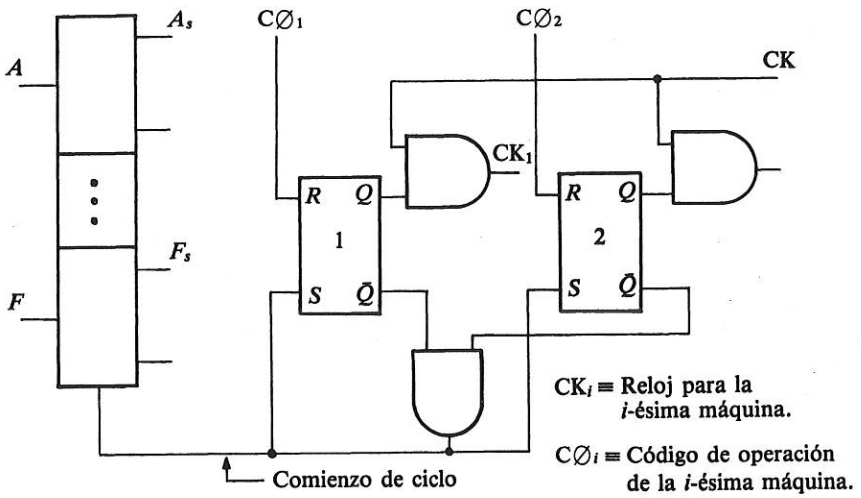


Figura 7.24. Dispositivo para la sincronización de máquinas de decisiones binarias con dos instrucciones. (*Nota*. Está particularizado para la realización de la RdP de la figura 7.22.)

Una tercera posibilidad para realizar una RdP consiste en ejecutar todos los GR con la misma máquina de decisiones binarias, multiplexando en el tiempo la actividad de ésta entre los diferentes GR. Es decir, se secuencializa el tratamiento de los GR con lo que el sistema se hace más lento. En la explotación de computadores, este tipo de técnica se denomina *multiprogramación*. Las máquinas presentadas en las figuras 7.14, 7.16, 7.17 y 7.18 no pueden ser multiprogramadas puesto que, sin considerar por el momento la realización de las salidas, sólo poseen un registro o

contador de estado o de programa y se necesitaría uno por GR. La relativa especificidad y complejidad material de las nuevas máquinas así como de sus técnicas de programación, harán que no exploremos más esta vía. (En el capítulo 9, utilizando microcomputadores estándar y técnicas de programación más simples, se obtendrán realizaciones *multiprogramadas*.)

En resumen, con una máquina de decisiones binarias (§7.3.3) se puede realizar el GR equivalente a una RdP dada. Mediante la interconexión material de varias máquinas, se pueden realizar los GR en que se haya descompuesto la RdP. En este último caso no debe olvidarse la sincronización entre las máquinas.

EJERCICIO. Prográmense dos máquinas de dos instrucciones y dirección implícita (§7.3.3.5) para realizar la RdP de la figura 7.22.

7.4.3 Descomposición de una RdP binaria en GR

En este apartado se presentan técnicas para la descomposición de una RdP binaria. El lector que abordara el último ejercicio del §7.4.1, observaría que el método directo de descomposición sugerido en ese párrafo resulta muy laborioso pues se han de seguir los siguientes pasos:

a) Obtener los marcados alcanzables (estados), M_k.
b) Construir los compatibles máximos con los $M(p_j)$.
c) Determinar una partición de los $M(p_j)$ que los cubra a todos.
d) Codificar los diferentes campos.
e) Determinar los eventos y salidas de interconexión entre GR.

En §7.4.3.1 se presenta una técnica que sustituye a las tres primeras fases, con lo que la descomposición suele ser extremadamente fácil. En §7.4.3.2 se presenta otra técnica que permite descomponer la RdP en menos GRs que elementos tiene el incompatible máximo de los $M(p_j)$. De este último modo, la realización final podrá hacerse con un menor número de máquinas funcionando en paralelo.

Para simplificar los tratamientos posteriores, en lo sucesivo se utilizará el *macrografo* (MG) asociado a la RdP que se desea realizar. Es decir, la RdP que se obtiene al sustituir por macrolugares (ML) las *subRdP reducibles* (§4.5.1). Evidentemente, este tipo de reducción es válido puesto que los marcados de los lugares de un ML de una RdP binaria estarán en exclusión mutua. Por otro lado, resultará «natural» que todo grupo de lugares reducibles a un ML se realice en un mismo GR.

7.4.3.1 *Descomposición en componentes conservativas y cobertura de los macrolugares* (ML)

a) *Descomposición*:

Sea Y una componente conservativa *monomarcada* del MG asociado a la RdP que se pretende realizar. Es decir, sea Y una componente conservativa tal que $Y^T \cdot M_0 = Y^T \cdot M = 1$.

Si todos los lugares del MG (o RdP) pueden estar marcados, entonces $Y \in \{0, 1\}^n$. De ello se desprende que $Y^T \cdot M_0 = 1$ puede ser reescrito como $\forall p_j \in ||Y|| \sum_{p_i} M(p_i) = 1$. Por consiguiente, los marcados $M(p_i)$ tales que los p_i pertenecen al soporte de una

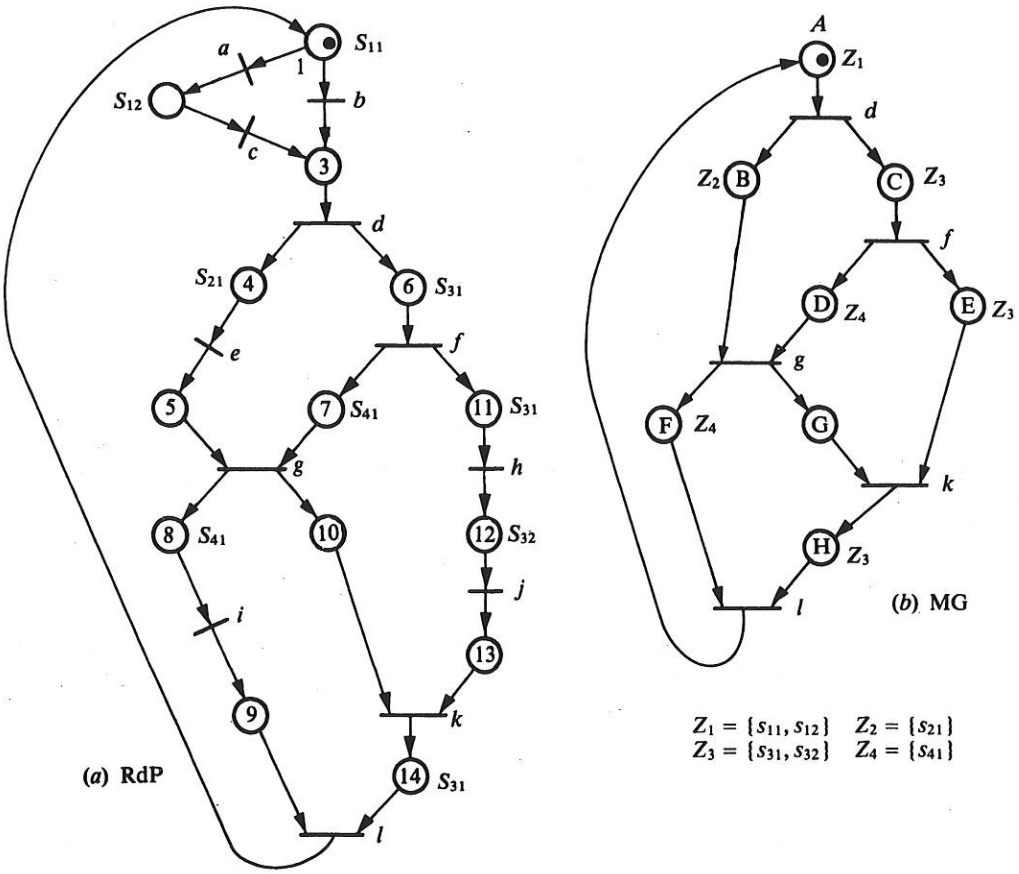


Figura 7.25. RdP y MG asociado

componente conservativa monomarcada, definen directamente un compatible máximo [los $M(p_i)$ están en exclusión mutua y no se puede añadir otro $M(p_j)$]. Además, la subRdP que definen los lugares, p_i, con sus transiciones de entrada y de salida es un GE binario.

Si el MG asociado a la RdP que se desea realizar es vivo (cosa que es de esperar), toda componente conservativa monomarcada será *elemental*, por lo que para su determinación bastará con aplicar el algoritmo del §4.7.2.2. En efecto, la vivacidad exige que sean disparables todas las transiciones del MG y, por consiguiente, exige que para toda componente conservativa, Y, se verifique: $Y^T \cdot M_0 \geqslant 1$ (proposición 4.17). Si Y no fuese elemental, cabría la posibilidad de descomponerla: $Y = \bar{Y} + \bar{\bar{Y}}$; ello implicaría: $Y^T \cdot M_0 = (\bar{Y} + \bar{\bar{Y}})^T \cdot M_0 = \bar{Y}^T \cdot M_0 + \bar{\bar{Y}} \cdot M_0 \geqslant 2$, de donde la no elementaridad de Y implica el que no pueda ser monomarcada.

La aplicación del algoritmo que permite calcular las componentes conservativas elementales (§4.7.2.2.) al MG (RdP) de la figura 7.25*b* arroja el siguiente resultado (todas son monomarcadas):

$Y_1^T = (1\,1\,0\,0\,0\,1\,0\,0)$ $\quad ||Y_1|| = \{ML_A, ML_B, ML_F\}$
$Y_2^T = (1\,1\,0\,0\,0\,0\,1\,1)$ $\quad ||Y_2|| = \{ML_A, ML_B, ML_G, ML_H\}$
$Y_3^T = (1\,0\,1\,1\,0\,1\,0\,0)$ $\quad ||Y_3|| = \{ML_A, ML_C, ML_D, ML_F\}$
$Y_4^T = (1\,0\,1\,1\,0\,0\,1\,1)$ $\quad ||Y_4|| = \{ML_A, ML_C, ML_D, ML_G, ML_H\}$
$Y_5^T = (1\,0\,1\,0\,1\,0\,0\,1)$ $\quad ||Y_5|| = \{ML_A, ML_C, ML_E, ML_H\}$.

b) *Cobertura*:

La realización de la RdP (figura 7.25*a*) puede hacerse de modo que cada ML_j se materialice una única vez, en uno de los GR que se obtengan. Esto plantea el problema de la *cobertura* de los macrolugares por las componentes conservativas monomarcadas.

Entre las múltiples coberturas posibles, interesa seleccionar una que simplifique la realización final. Así, por ejemplo, puede ser «razonable» la determinación de una (la) cobertura que minimice el número de GR necesarios. (*Nota.* Este problema es análogo al que se presenta en la segunda fase del método de Quine-McCluskey para simplificar la realización de funciones lógicas combinacionales con puertas: la determinación de un subconjunto mínimo de términos producto que cubra la función lógica [HILL 78].)

La tabla 7.5 presenta la *tabla de cobertura* para el ejemplo considerado. La componente Y_5 es *esencial* en toda cobertura puesto que es la única que permite cubrir ML_E. Al retener Y_5, se pueden eliminar las columnas asociadas a los ML_j que cubre: ML_A, ML_C, ML_H.

Macrolugares (elementos a cubrir)

Componentes conservativas monomarcadas (elementos cobertores)	ML_A	ML_B	ML_C	ML_D	ML_E	ML_F	ML_G	ML_H
Y_1	X	X				X		
Y_2	X	X					X	X
Y_3	X		X	X		X		
Y_4	X		X	X			X	X
Y_5	X		X		X			X

↓
Y_5 es esencial en toda cobertura

Tabla 7.5 Tabla de cobertura (una X en la casilla *i-j* significa que Y_i cubre a ML_j).

Se denomina *irredundante* toda aquella cobertura en la que la eliminación de un elemento cobertor implique que alguno(s) de los elementos a cubrir no sea(n) efectivamente cubierto(s); es decir, en una cobertura irredundante todos los cobertores son necesarios. Considerando la tabla 7.5, tras eliminar las columnas ML_A, ML_C y ML_H, se constata que para cubrir:

—ML_B se tiene que tomar Y_1 o Y_2
—ML_D se tiene que tomar Y_3 o Y_4
—ML_F se tiene que tomar Y_1 o Y_3
—ML_G se tiene que tomar Y_2 o Y_4

Sea $\mathcal{Y}_j$ la variable lógica que expresa la pertenencia de Y_j a una cobertura. El conjunto de todas las coberturas irredundantes resultará del producto de las funciones de cobertura de cada elemento a cubrir. En el del caso que nos ocupa viene dado por:

$$\mathcal{Y}_5(\mathcal{Y}_1 + \mathcal{Y}_2)(\mathcal{Y}_3 + \mathcal{Y}_4)(\mathcal{Y}_1 + \mathcal{Y}_3)(\mathcal{Y}_2 + \mathcal{Y}_4) = \mathcal{Y}_2\mathcal{Y}_3\mathcal{Y}_5 + \mathcal{Y}_1\mathcal{Y}_4\mathcal{Y}_5.$$

donde $\mathcal{Y}_5$ representa al cobertor esencial, $\mathcal{Y}_1 + \mathcal{Y}_2$ es la función de cobertura de ML_B, $\mathcal{Y}_3 + \mathcal{Y}_4$ es la función de cobertura de ML_D, etc.

La RdP de la figura 7.25*a* puede cubrirse a partir de las componentes $\{Y_2, Y_3, Y_5\}$ o bien $\{Y_1, Y_4, Y_5\}$. En ambos casos se definen GE monomarcados.

c) Lugar de reposo:

Consideremos en lo sucesivo, por ejemplo, la cobertura obtenida a partir de $\{Y_1, Y_4, Y_5\}$. Cada una de las componentes de esta cobertura permitirá la definición de un GR. La interconexión de los tres GR permitirá la simulación de la RdP.

Antes de continuar, conviene observar que el macrolugar ML_A aparece tres veces (una en cada componente) y los macrolugares ML_C y ML_H aparecen dos veces (en Y_4 e Y_5). Como para realizar la RdP basta con que cada ML_j se materialice en un único GR, se podrán llevar a cabo ciertas simplificaciones. Estas consistirán en sustituir *una* vez cada uno de los ML_j por la subRdP correspondiente; en las demás apariciones de cada ML_j se procederá a mantenerlo como un único lugar, l_j. Habida cuenta de que los lugares l_j que aparecen en un grafo no conservan información alguna sobre la coordinación evento-acción definida por la RdP inicial, podrán ser fusionados en uno único, al cual denominaremos *lugar de reposo*, LR (todo GR puede tener un único LR). La denominación «lugar de reposo» se ha adoptado por ser éste el único lugar marcado cuando los macrolugares sustituidos en el GR están desmarcados. (*Nota*: éste es el papel de p_0 en la figura 7.23*b*.)

Volviendo sobre el ejemplo considerado (figura 7.25), en la figura 7.26 se representan tres GR. Éstos han sido obtenidos al realizar determinadas sustituciones de macrolugares por las subRdP que representan. Las sustituciones realizadas son las siguientes:

1) Todos los macrolugares de Y_4; el grafo que se obtiene es GR_1. GR_1 no tiene lugar de reposo.
2) ML_B y ML_F de Y_1; se obtiene el grafo GR_2. LR_2 se obtiene puesto que ML_A, se sustituyó en GR_1
3) ML_E de Y_5; se obtiene GR_3. LR_3 proviene de la fusión de l_A, l_C, l_H, todos sustituidos en GR_1.

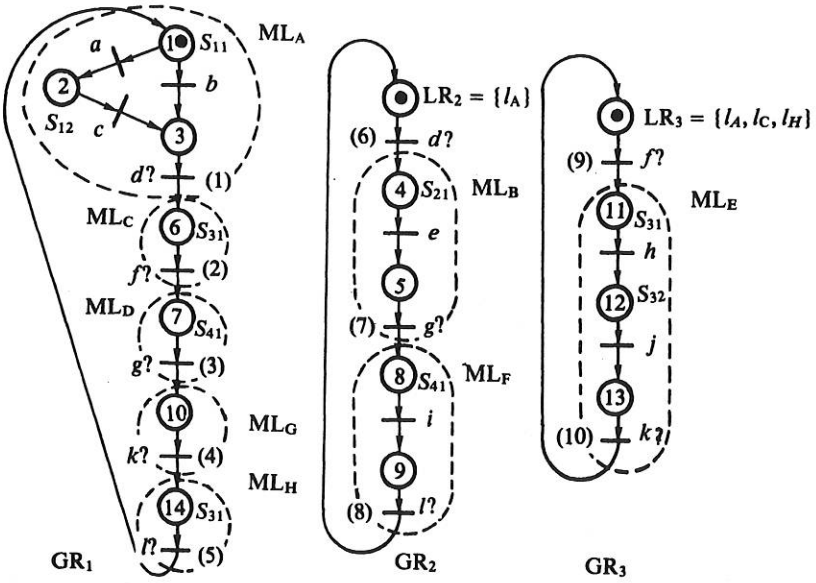


Figura 7.26. Grafos que simulan el comportamiento de la RdP de la figura 7.24.

d) *Eventos*:

Para terminar la definición de los GR basta con definir los eventos asociados a las transiciones de salida de los diferentes macrolugares. Ello ha de hacerse de forma que la coordinación entre los GR simule a la RdP original. Así, a los eventos que etiqueten transiciones de salida de los ML_j deberá añadírsele, eventualmente, la condición que complete la sensibilización de su transición. En la figura 7.26, los eventos etiquetados con «?» puede que tengan que ser completados.

La evolución $p_3 \rightarrow p_6$ se produce (figura 7.25*a*) si $M(p_3)d = 1$. Por consiguiente, $d? \equiv d$ para la transición 1 (figura 7.25). Del mismo modo, la evolución $p_7 \rightarrow p_{10}$ se llevará a cabo si $gM(p_5)M(p_7) = 1$. El evento asociado a la transición 3 debe completarse hasta $gM(p_5)$. El marcado de p_4 se produce cuando $M(p_3)d = 1$, luego la transición 6 debe etiquetarse $M(p_3)d$. El cálculo de los nuevos eventos arroja el siguiente resultado:

1) $d? \equiv d$
2) $f? \equiv f$
3) $g? \equiv g\,M(p_5)$
4) $k? \equiv k\,M(p_{13})$
5) $l? \equiv l\,M(p_9)$
6) $d? \equiv d\,M(p_3)$
7) $g? \equiv g\,M(p_7)$
8) $l? \equiv l\,M(p_{14})$
9) $f? \equiv f\,M(p_6)$
10) $k? \equiv k\,M(p_{10})$.

El conjunto de GR de la figura 7.26, completados los eventos según la lista anterior, permite simular el comportamiento de la RdP de la figura 7.25*a*.

EJERCICIO. Obténganse tres GR que simulen la RdP de la figura 7.25*a* a partir de la cobertura $\{Y_2, Y_3, Y_5\}$. Obsérvese que, en este caso, es posible obtener una mejor realización que la de la figura 7.26, puesto que se puede obtener una partición de las salidas en GR (es decir, cada salida estará generada por un único GR).

e) *Limitación*:

A pesar del atractivo que su simplicidad operativa le proporciona, el método de descomposición en GR presentando en este apartado no es siempre utilizable puesto que existen RdP binarias y vivas que no se descomponen en componentes conservativas monomarcadas. Así, a modo de ejemplo, la figura 7.27 presenta dos de tales redes. (Aunque la primera de ellas es binaria y viva gracias al interpretación, la segunda lo es considerada como red autónoma.) Ambas redes están constituidas por una única componente conservativa elemental, $Y = (2\,1\,1\,1\,1)$, no monomarcada: $Y^T \cdot M_0 = 2$.

Desde un punto de vista práctico, hay que señalar que, afortunadamente, rara vez una RdP binaria y viva que modela un determinado sistema no se descompone en componentes conservativas monomarcadas.

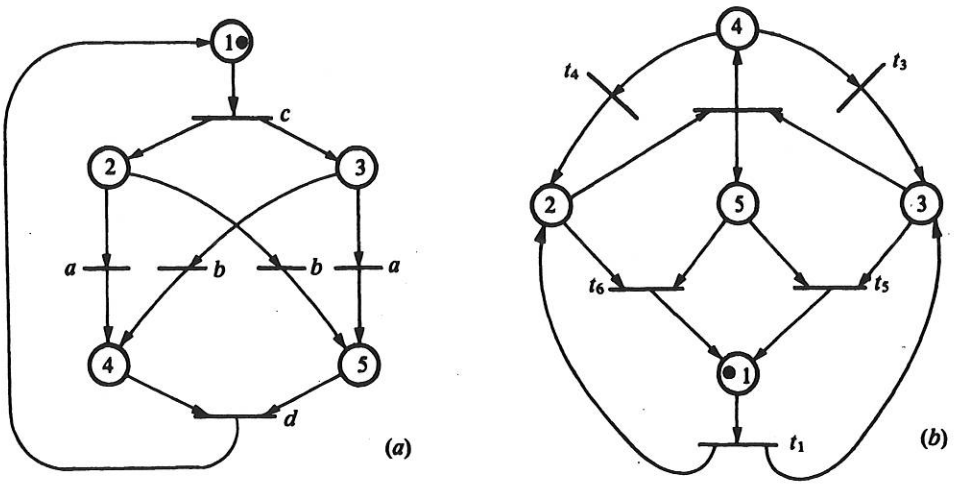


Figura 7.27. RdP vivas y binarias que no pueden descomponerse en componentes conservativas elementales monomarcadas.

EJERCICIO. Compruébese la afirmación anterior utilizando las RdP que modelan sistemas en los capítulos 1, 2 y 3.

EJERCICIO. ¿Merece la pena simplificar las RdP utilizando el método de los eventos fuente (§3.5) cuando se va a proceder a una realización con PLA o ROM?

7.4.3.2 *Descomposición en RdP binarias y vivas*

Hasta ahora, las RdP binarias y vivas han sido descompuestas directamente en GR. Para una RdP dada, son necesarios, como mínimo, tantos GR como lugares simultáneamente marcados existan (éstos forman un incompatible). De este modo, la descomposición directa de la RdP de la figura 7.25*a* conduce a 3 GR. Por otro lado, la RdP de la figura 7.27*b* puede llegar a tener sólo 2 lugares simultáneamente marcados, pero su descomposición directa conduce a 3 GR. (Esto puede comprenderse al observar que los marcados alcanzables son: $\{p_1, p_2\text{-}p_3, p_4\text{-}p_5, p_2\text{-}p_2\text{-}p_5, p_3\text{-}p_5\}$ y, evidentemente, $\{M(p_2), M(p_3), M(p_5)\}$ forma un incompatible (máximo), a pesar de que $p_2\text{-}p_3\text{-}p_5$ no pueden estar simultáneamente marcados.)

La reducción del número de GR necesarios en la descomposición y cobertura de una RdP puede ser interesante para obtener una realización con un menor número de subsistemas físicos. A continuación se presenta un método que procede recubriendo la RdP original por tantas otras redes como se desee. Posteriormente, cada red empleada en la cobertura podrá ser realizada directamente o previa transformación en GR (§1.5.4). En este último caso, se dirá que la descomposición en GR es *indirecta*. Como es de esperar las descomposiciones indirectas en GR suelen ser menos flexibles que las directas, aunque pueden permitir realizaciones más económicas.

ALGORITMO DE COBERTURA CON RDP Y REDUCCIÓN:

(1) Obténgase el MG asociado a la RdP.
(2) Particiónense los macrolugares del MG de acuerdo con el criterio del diseñador. [Cada elemento de la partición, $\mathcal{P}_j$, generará una de las RdP que constituirán la cobertura.]
(3) **Para cada** $\mathcal{P}_j$ {elemento de la partición anterior}
hacer (3.1) Aplíquense las reglas de reducción (§4.5) a los macrolugares que no pertenezcan a $\mathcal{P}_j$, sabiendo que ningún macrolugar podrá ser sustituido si ello afecta a transiciones de entrada y/o de salida de los macrolugares pertenecientes a $\mathcal{P}_j$.
(3.2) Obténganse los eventos asociados a las transiciones de tal forma que se mantenga la coordinación evento-acción†.
(3.3) Sustitúyanse los macrolugares que pertenezcan al elemento de la partición que se trata, por las correspondientes subRdP que los definen en la RdP inicial.

EJEMPLO. Sea de la RdP de la figura 7.25*a*.

Paso 1: Se obtiene el MG (figura 7.25*b*).

Paso 2: Se elige, por ejemplo la partición de los macrolugares: $\mathcal{P}_1 = \{A, B, C, D\}$ y $\mathcal{P}_2 = \{E, F, G, H\}$. Por consiguiente, la cobertura que se obtenga tendrá sólo dos elementos.

---

† En este paso puede ser interesante el aplicar el método de simplificación denominado *fusión de lugares* (§3.3) a los macrolugares de la RdP resultante que no pertenecen a $\mathcal{P}_j$.

Paso 3:

1.ª *iteración*: *Sea* $\mathcal{P}_1$ (*macrolugares sombreados en la figura* 7.28*a*).

(3.1) *ML_F es implícito (lo implican $\overline{ML_G}$ y ML_H). Se elimina.
*ML_E es implícito (lo implican $\overline{ML_D}$ y ML_G). Se elimina.
*ML_G y ML_H pueden reducirse a $\overline{ML_{GH}}$.
*ML_{GH} no puede ser sustituido puesto que sus transiciones de entrada y de salida son compartidas con macrolugares de $\mathcal{P}_1$, $\{ML_B, ML_D, ML_A\}$.

(3.2) *La única transición que es preciso considerar es la transición de salida de $\overline{ML_{GH}}$. Su disparo es posible sólo si p_9 y p_{14} (figura 7.26*a*) están marcados y se produce *l*.

(3.3) La sustitución de $\{ML_A, ML_B, ML_D\}$ conduce a la RdP de la figura 7.28*b*.

2.ª *iteración*: (*Realícese como ejercicio.*)

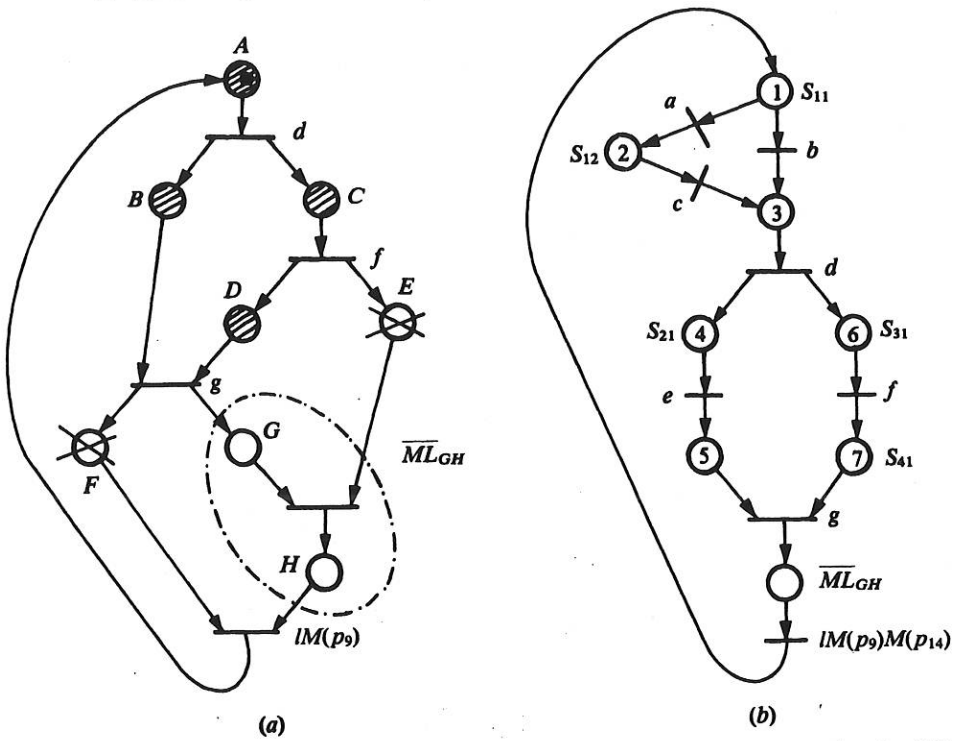


Figura 7.28. Obtención del elemento de la cobertura de la RdP que corresponde a los ML: *A*, *B*, *C*, *D*. (Obsérvese que una transformación local puede eliminar el paralelismo final.)

7.5 CONCLUSIÓN

En el presente capítulo hemos abordado la realización RdP con memorias (ROM,...) y matrices lógicas programables (PLA,...), macrocomponentes que han sido definidos funcionalmente.

La relativa ineficiencia de la realización directa de RdP con PLA o/y ROM ha conducido a su simulación mediante GR (realizables con máquinas secuenciales) interconectados. Es decir, la realización de sistemas concurrentes está basada principalmente en la realización de sistemas secuenciales. Dado que estos últimos se llevan a cabo como combinacionales realimentados a través de un registro, se comprenderá la estructura con que se ha dotado al capítulo.

Desde un punto de vista conceptual, puede decirse que las realizaciones con ROM y PLA son esencialmente tabulares, lo que las diferencia de las presentadas en el capítulo 6.

Dentro del marco de realizaciones tabulares, éstas evolucionan desde las que definen directamente *tablas de verdad* hasta las que representan *microprogramas*, entendidos éstos últimos en el sentido más algorítmico.

La multitud de variaciones imaginables a partir de los métodos de realización expuestos, así como de otros posibles, hace que el material presentado constituya sólo una primera aproximación a la realización microprogramada. Sin duda alguna, la aplicación más importante de este tipo de técnicas ha sido la realización de secuenciadores para computadores digitales. Los textos [HUSS 70] (auténtico pionero) y [KATZ 77], permiten completar la perspectiva en este área de aplicaciones secuenciales.

A modo de comentario general sobre el diseño, hay que señalar que la relativa «distancia» existente a veces entre la descripción de un sistema (RdP) y la realización con un determinado tipo de máquina, sugiere el interés del empleo de *paquetes de ayuda al diseño* (CAD, *Computer Aided Design*).

Por último, es interesante resaltar cómo el concepto de estado de un GR se aproxima al de dirección de un microprograma (§7.3.3). De este modo, resulta natural llegar a establecer organigramas «funcionalmente equivalentes» a un GR o RdP. En el anexo 1 se establece una relación similar desde un punto de partida distinto.

EJERCICIOS

7.1 Dado el GR de la figura 3.7, obténganse microprogramas que lo realicen sobre las máquinas de decisiones binarias con dos instrucciones. Especifíquese el contenido real de las memorias.

7.2 Obténgase una cobertura directa con GR de la RdP de la figura 2.3. ¿Cómo podría procederse para que cada salida sea generada por un único GR? (*Sugerencia*. Mejórese la solución trivial que consistiría en expansionar cada ML tantas veces como sea necesario para particionar las salidas. La simplificación por fusión de lugares puede ser de gran utilidad.)

7.3 Diséñese un sincronizador para máquinas de decisiones binarias con una única instrucción. (*Sugerencia*. La definición de una función de salida extra para la sincronización facilita la tarea.)

7.4 Diséñese un sincronizador para máquinas asíncronas-autosincronizadas (§7.3.3.5).

7.5 Desde el punto de vista de la realización de RdP vivas y binarias, ¿se podría haber definido el concepto de macrolugar como todo componente conexo obtenido al eliminar de la

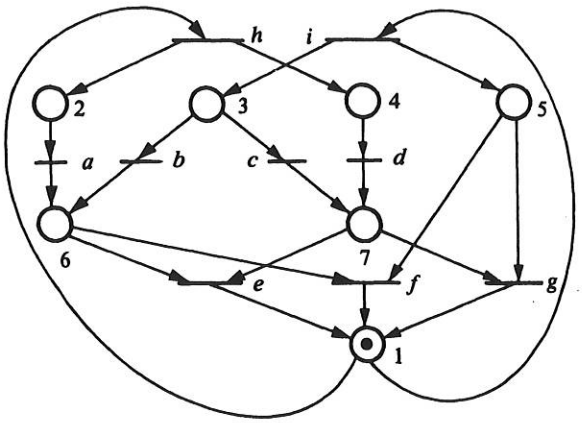


Figura E.7.1 RdP binaria y viva en la que la subRdP formada a partir de $\{p_2, p_3, p_4, p_6, p_7\}$ puede contener más de una marca.

RdP inicial las transiciones de tipo Y? (*Sugerencia*. La RdP de la figura E.7.1 puede serle útil para apoyar sus razonamientos.)

7.6 ¿Qué modificaciones habría que hacer en las técnicas presentadas en §7.4 para poder realizar RdP con arcos inhibidores? Realícese la RdPAI de la figura 2.20.

7.7 Para la RdP de la figura 7.22, obténgase una realización cableada (no modular) funcionalmente equivalente a la basada en el método lógico de transición directa. (*Sugerencia*. Después de obtener la tabla de programación de la PLA, calcúlense las ecuaciones lógicas que realiza y propóngase una implementación con puertas lógicas.)

8

Realización programada (I): autómatas programables generales

8.1 INTRODUCCIÓN

Desde principios de la década de los setenta, se observa que las técnicas de realización de automatismos lógicos evolucionan hacia métodos programados.

Las principales razones de esta evolución se pueden resumir en los puntos que siguen:

1) La *flexibilidad* de los sistemas programables, que facilita de forma importante la automatización de procesos complejos o/y existentes en un pequeño número de ejemplares. Por otro lado, la programabilidad simplifica extraordinariamente la adaptación de los automatismos ante los posibles cambios del proceso (cadenas de fabricación de «familias de piezas», procesos químicos con diagrama de flujo modificable, etc.).
2) La integración en el automatismo, a bajo coste, de *funciones complejas* muy diversas. De este modo, al lado de las funciones lógicas, pueden integrarse funciones de servicio (archivado, elaboración de balances, edición de mensajes, etc.) e incluso funciones de regulación (reguladores PID, etc.).
3) La *disminución del coste* relativo al desarrollo y a la producción del equipo, gracias a la utilización de macrocomponentes (disminución del número de tarjetas, conectores, conexiones, etc.).
4) El *aumento de la seguridad de funcionamiento*, obtenido en base a:
   - una mayor *fiabilidad*, debida a la disminución en el número de componentes del sistema.
   - una mayor *reparabilidad*, facilitada por la existencia de programas y/o dispositivos de diagnóstico de averías.
5) La posible *descentralización del control*, facilitada por la capacidad de comunicación de los microprocesadores, así como de otros macrocomponentes.

Esta evolución trajo consigo la aparición de computadores especializados en el tratamiento de problemas lógicos en un ámbito industrial que se denominan *Autómatas Programables*, AP (*Programmable Logic Controllers*, PLC). En realidad se trata de un fenómeno de transposición técnica similar al que se produce, de forma paralela, al reemplazar las cadenas analógicas de regulación por computadores numéricos.

A pesar de su reciente aparición ([HOLZ 69] es considerada cronológicamente como la primera referencia), los AP son, hoy en día, sistemas ampliamente utilizados. Entre otros aspectos claves que justifican su aceptación, pueden citarse:

a) los *lenguajes de programación*, fáciles de entender por personal no cualificado en informática.
b) la *modularidad*, especialmente a nivel de entradas y salidas.
c) la *adaptación a condiciones ambientales adversas* como puedan ser temperaturas entre 0° y 60° C, humedades relativas elevadas (hasta 95%), polvo, abundancia de parásitos eléctricos, etc.

En resumen, los constructores de AP han buscado la definición de equipos que, introduciendo innovaciones técnicas, no trastornen «las costumbres» de los usuarios directos de los mismos (personal de mantenimiento, operadores del proceso, etc.).

El objetivo que persigue cubrir este capítulo es una presentación general de la programación, la estructura y el funcionamiento de los AP. Para su lectura es recomendable un conocimiento básico sobre la estructura y el funcionamiento de los computadores digitales, nivel que se cubre con holgura en textos de introducción (por ejemplo [MEIN 72]).

Desde un punto de vista estructural, en este capítulo se diferencian claramente dos partes. En la primera de ellas se presentan las características generales de la estructura y el funcionamiento de los AP (§8.2), así como los rasgos básicos de los principales tipos de lenguajes de programación (§8.3). En este último apartado, se dedica especial atención a la presentación de técnicas de programación de RdP; en particular, se consideran problemas que surgen debido a la simulación puramente secuencial de la evolución del marcado de las RdP. Ello llevará a distinguir entre simulaciones *síncronas* y *no síncronas*.

En la segunda parte del capítulo (§8.4) se consideran, con cierto nivel de detalle, diversos AP-tipos. Las peculiaridades de su estructura y funcionamiento buscan la máxima simplificación de los lenguajes de programación y la eficiencia en su interpretación, de forma que la respuesta del AP a solicitaciones externas sea lo más rápida posible (no debe olvidarse que los AP trabajan siempre en *tiempo real*, mandando un proceso).

El estudio de sistemas específicamente diseñados para simular eficientemente las descripciones basadas en RdP se llevará a cabo en el próximo capítulo. La mayor simplicidad de la estructura y funcionamiento de los AP generales nos ha inclinado a adelantar su presentación con respecto a la de sistemas especialmente adaptados a la simulación de RdP.

8.2 ELEMENTOS SOBRE LA ESTRUCTURA Y EL FUNCIONAMIENTO DE LOS AUTÓMATAS PROGRAMABLES

Los autómatas programables utilizan normalmente estructuras clásicas, empleadas con frecuencia en los *mini* y *microcomputadores*. La figura 8.1 ilustra una estructura básica de AP. En este párrafo van a ser presentados, a un nivel general, cada uno de los bloques que aparecen, así como nociones sobre el funcionamiento de esta cla-

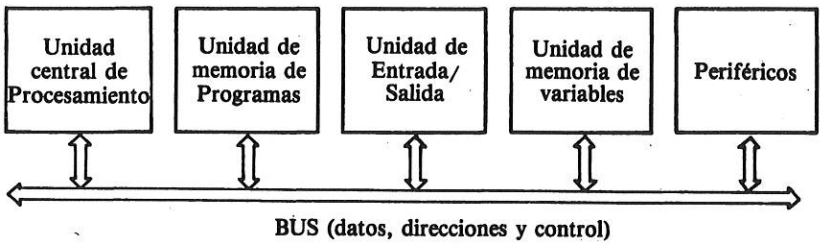


Figura 8.1 Estructura básica de un AP.

se de sistemas. Como cualidad funcional importante, se puede observar que un AP ejecuta cíclicamente una única tarea. Se denominará *ciclo de tratamiento* a una ejecución completa del programa introducido por el usuario.

8.2.1 Estructura de un AP. Generalidades

Las características principales de la estructura de un AP son:

a) Separación de la memoria en memoria de *programas* (organizada en palabras) y memoria de *variables* (organizada en bits).
b) Especialización de la Unidad de Tratamiento para el procesamiento de problemas de tipo lógico.
c) Existencia de periféricos especiales.
d) Interconexión entre los diferentes bloques mediante uno o varios buses.

La estructura representada en la figura 8.1 se denomina de *bus-único* (*unibus*). En ella el bus permite el tráfico de todas las informaciones: datos, direcciones y control. Ciertos AP utilizan estructuras multibus y, en particular, es frecuente la separación en bus-memoria y bus de entrada/salida (figura 8.2). Esta separación permite el procesamiento paralelo de información sobre ambos buses, por lo que la ejecución de los programas del usuario puede ser más rápida.

Antes de abordar el funcionamiento de un AP, se presentarán rápidamente las peculiaridades de los distintos bloques que lo componen.

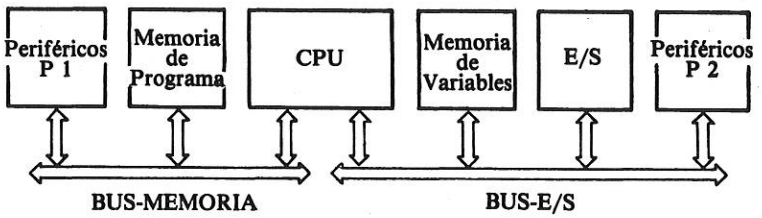


Figura 8.2. Estructura con bus-memoria y bus-E/s (*Nota.* CPU, *Central Processing Unit*).

8.2.2 Unidad de memoria

Como se ha dicho, se distinguen:

a) la *memoria de programa*, MP, que está organizada en palabras (8, 12 o 16 bits) y suele ser de sólo lectura (ROM, PROM o EPROM), con lo cual se incrementa la seguridad de su buen funcionamiento frente a parásitos eléctricos, caídas de tensión en la alimentación, etc.

b) la *memoria de variables*, MV, que está fundamentalmente organizada en bits, y es de lectura y escritura (RAM).

8.2.3 Unidad de entrada/salida

Habida cuenta que la principal aplicación de los AP es el tratamiento de problemas lógicos, las entradas y salidas (E/S) son fundamentalmente de tipo binario aunque en ciertos equipos aparezcan E/S numéricas (8, 16 o 24 bits) e incluso, a veces, analógicas.

En su forma más simple, las entradas y salidas se pueden adquirir o emitir *directamente*, en el momento en que se necesite la variable de entrada o se haya calculado el valor de la variable de salida. Ahora bien, si se desean evitar de forma sistemática los funcionamientos anómalos que puedan provenir de la alteración de los valores de las entradas durante un ciclo de tratamiento, éstos han de ser adquiridos simultáneamente y memorizados durante el ciclo. El anterior proceso se denomina *muestreo*. La memorización mencionada suele llevarse a cabo en una *memoria imagen de las entradas*, MIE, que, normalmente, será actualizada al comienzo de cada ciclo. La actualización puede ser llevada a cabo por la unidad central de procesamiento o mediante un procesador especial, denominado *procesador de entrada/salida*. Otra forma de proceder, algo más costosa pero de amplia utilización, consiste en cargar directamente los valores que se adquieren (correspondientes a las variables que se muestrean) en registros dispuestos en las interfases de entrada. En este último caso, el conjunto de registros de entrada forma la MIE y su actualización se puede realizar auténticamente de forma simultánea, en un único ciclo de reloj.

La emisión de las salidas se suele hacer directamente, a medida que se elaboran en cada ciclo de tratamiento. Si se desea realizar una emisión «cuasi-simultánea», se deberá utilizar una *memoria imagen de las salidas*, MIS, donde se vayan almacenando éstas a medida que son elaboradas. Al finalizar un ciclo, la unidad central de procesamiento o, si existiese, el procesador de entrada/salida, copiará la imagen sobre los registros de las interfases de salida. Procediendo de este modo, la utilización de una memoria imagen permite lograr, además de la emisión cuasi-simultánea, una mayor seguridad de buen funcionamiento ante posibles parásitos que puedan alterar puntualmente el contenido de registros de las interfases de salida. En efecto, si una perturbación ha alterado una variable de salida en la interfase, al recopiar su imagen, se restaurará el valor correcto.

La figura 8.3 presenta la arquitectura de un AP que posee un procesador de entrada/salida, memoria imagen de las entradas y memoria imagen de las salidas. El procesador de entrada/salida se suele encontrar en los grandes sistemas y se ocupa de

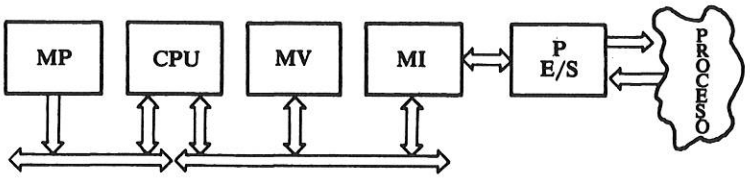


Figura 8.3. Memoria imagen (MI) y procesador de entrada/salida (P-E/S) en un AP.

realizar (cíclicamente o bajo demanda de la CPU) las adquisiciones y las emisiones de las entradas y las salidas (respectivamente), a partir de una MIE y de una MIS con las cuales trabaja la CPU.

Desde un punto de vista tecnológico, las E/S se presentan agrupadas en *tarjetas enchufables* (una tarjeta suele poseer un total de 8 a 16 E/S); es decir, se presentan de forma modular. Básicamente las *interfases de entrada* transforman los 24 o 48 voltios continuos o los 110 o 220 voltios alternos, que provengan del proceso, en señales de «bajo nivel» (energético), utilizables por la CPU. Las entradas suelen estar fuertemente protegidas. Así, es frecuente observar la presencia de aislamientos optoelectrónicos y mecanismos de protección contra parásitos (por ejemplo, filtrado analógico con $\tau \cong$ 8-20 mseg.)

Las *interfases de salida*, además de los aislamientos optoelectrónicos, generan señales de «alto nivel» (energético), capaces de mandar los accionadores (normalmente 24-48 voltios continuos o 110-220 voltios alternos, pudiendo llegar las corrientes hasta varios amperios).

Las consideraciones anteriores sobre las E/S hacen que, para sistemas AP de una dimensión media ($\cong$ 200 E/S), su coste sea muy importante, oscilando entre el 60 y 80% del coste total del equipo. El número de E/S de los AP varía típicamente entre 16 y 10240.

8.2.4 Unidad central de procesamiento

La CPU (*Central Processing Unit*) aparece como una unidad especializada, fundamentalmente, en el tratamiento de problemas lógicos, que, a veces, ofrece la posibilidad de su extensión para realizar cálculos numéricos.

La realización de la CPU puede ser *específica* (circuitería especialmente diseñada) o bien basarse en un *procesador estándar* (mini o microprocesador). En este último caso, un conjunto de programas-sistema permite que el AP sea capaz de interpretar y ejecutar los programas del usuario. En lo sucesivo permaneceremos a un nivel conceptual, por lo que se hablará de los AP como *máquinas virtuales*; es decir, independientemente de su realización física.

En la estructura de la CPU de un AP así como en la de cualquier computador digital, se distinguen dos partes claramente definidas:

— La *parte operativa*, PO.
— La *parte de control*, PC.

La PC o *unidad de control*, UC, tiene por misiones:

1) extraer de la memoria de programa las instrucciones,
2) interpretarlas (determinar su significado) y
3) ordenar su ejecución.

Para extraer las instrucciones de la memoria de programa, la UC dispone de un registro que señala la próxima instrucción que se debe ejecutar (contiene su dirección): el *contador de programa*, CP.

Para interpretar las instrucciones, el contenido de la palabra de la memoria cuya dirección está siendo designada por el CP es enviado a un registro especial, denominado de *instrucción*, RI. A partir de este registro se alimenta un *decodificador*, que parametriza la evolución de un sistema secuencial, denominado *secuenciador*. Este último tiene por misión generar las órdenes elementales (*microórdenes*) que permiten el funcionamiento del AP (transferencias entre registros, lectura/escritura de la memoria de variables, etc.).

La PO o *unidad de tratamiento*, UT, comprende dos grandes categorías de componentes:

1) Los *registros*.
2) El dispositivo de cálculo, normalmente una *unidad lógica*.

La unidad lógica, UL, constituye, casi siempre, el núcleo de la unidad de tratamiento. Se trata de un dispositivo, no siempre combinacional, capaz de realizar operaciones lógicas. La ineficiencia relativa de las *unidades aritméticas y lógicas*, UAL, de los computadores de propósito general para el cómputo de funciones lógicas (representadas mediante expresiones booleanas, diagramas lógicos, esquemas con relés, etc.) ha conducido al desarrollo de UL basadas en principios muy diferentes. En §8.4 se presentan, a nivel funcional, algunas de éstas.

Los AP son máquinas de una dirección, por lo que, a veces, aparece(n) un (varios) registro(s) *acumulador(es)* cuya función es la de servir como origen y destino de las operaciones de cálculo. En los AP, el acumulador, RA, es de un solo bit (figura 8.4).

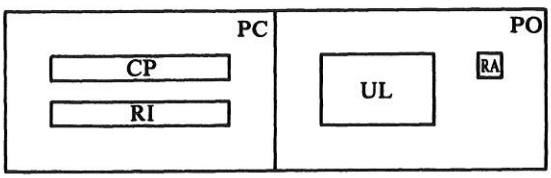


Figura 8.4. Registros básicos y unidad lógica de una CPU de AP. (*Nota*. Posteriormente, en el §8.4 se verá como esta estructura básica puede presentar notables variaciones en ciertos AP.)

Para aumentar la seguridad de funcionamiento, las CPU de los AP presentan, a veces, mecanismos *autodetectores de error*. Así, por ejemplo, es frecuente que se computen una función y su complemento, de forma que la comparación final permita detectar errores en el cálculo.

Desde el punto de vista de su realización, las CPU suelen ser unidades microprogramadas especialmente, o bien unidades basadas en un microprocesador estándar. Evidentemente, en este último caso, las prestaciones temporales (velocidad de cálculo) son inferiores a las que presentan las primeras.

8.2.5 Periféricos

8.2.5.1 *Batería de temporizadores y de contadores*

Considerada, a veces, como parte integrante de la unidad de tratamiento, proporciona las referencias de tiempo, por un lado, y asegura la función de cuenta, por otro.

8.2.5.2 *Consola de programación*

Se trata de un periférico muy complejo, frecuentemente diseñado alrededor de un microprocesador estándar monolítico, que realiza las funciones de:

—Traductor del lenguaje de programación.
—Cargador.
—Monitor de ayuda a la puesta a punto de los programas.
—Programador de memorias (PROM, EPROM).

Cuando el AP esté en servicio, normalmente la consola de programación estará desconectada. En ciertos AP, sobre todo en aquellos organizados alrededor de un microprocesador monolítico estándar (M6802, I8085, etc.), las funciones de la consola de programación son realizadas por la unidad central del mismo AP, aprovechándose de este modo el carácter universal del microprocesador del sistema.

8.2.6 Funcionamiento de un AP

Después de haber presentado los elementos básicos de un AP, estudiaremos su funcionamiento. Para ello, procederemos en dos tiempos, estudiando por un lado el funcionamiento a nivel de desarrollo de una instrucción y, por otro, el funcionamiento a nivel de desarrollo de un programa.

8.2.6.1 *Desarrollo de una instrucción*

Del mismo modo que en los computadores de propósito general, en el desarrollo de una instrucción se distingue la siguiente serie de fases o tratamientos elementales:

- la búsqueda (extracción de la instrucción designada por el CP)
- la decodificación (determinación de la significación de la instrucción)
- la ejecución
- la preparación para el desarrollo de la siguiente instrucción que se deba ejecutar (incremento del CP; es decir, CP : = CP + 1).

Desde el punto de vista temporal, la duración de la fase de ejecución es función de la instrucción en curso. En particular, si la instrucción hace referencia a un operando, la fase de ejecución comprenderá la búsqueda o almacenamiento de éste.

Al terminar la ejecución de la fase de búsqueda de una instrucción, normalmente se ejecuta, en paralelo con otras fases de su desarrollo, la preparación para el desarrollo de la instrucción siguiente.

8.2.6.2 *Desarrollo de un programa: ciclo de tratamiento*

Un programa se ejecuta gracias al encadenamiento en la ejecución de instrucciones. Como se comentó anteriormente, una característica esencial de los AP es su funcionamiento cíclico: al terminar completamente una ejecución del programa de control (tarea básica en el sistema), se puede iniciar otra nueva. En lo sucesivo, CT significará *ciclo de tratamiento*.

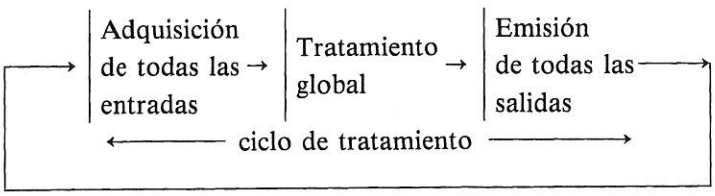
El encadenamiento en la ejecución del programa de control suele ser *directo*; es decir, al terminar un CT comienza el siguiente. A veces, un reloj externo lanza los CT periódicamente, cada θ ms†, lo cual da origen a un encadenamiento *síncrono* de ciclos. Por último, en algunos sistemas se lanza un CT sólo cuando, habiéndose terminado el CT anterior, se detecta por hardware externo un cambio de estado en alguna de las variables de entrada. Este método de encadenamiento de CT se denomina *autosincronizado*.

El encadenamiento directo es el de más fácil realización, pero el AP es monopolizado por la ejecución del programa de control. El encadenamiento síncrono es de fácil realización y permite que el AP ejecute otros trabajos distintos al programa de control. Éstos se denominan genéricamente *trabajos de fondo* (*background*) y pueden consistir en la comunicación con otros computadores, en la edición de mensajes, en la elaboración de balances, etc.

El encadenamiento autosincronizado conducirá, normalmente, a un menor número de lanzamientos de CT. De esta forma cabe esperar que el AP dispondrá de más tiempo para realizar los trabajos de fondo, sin que por ello se degraden las prestaciones temporales de la ejecución del programa de control. El inconveniente principal del lanzamiento autosincronizado es que su realización es más costosa.

Desde el punto de vista de la *adquisión* (A) y de la *emisión* (E) de información, los CT pueden clasificarse fundamentalmente en tres grupos:

(1) *Adquisición en bloque* (AB) y *emisión en bloque* (EB):

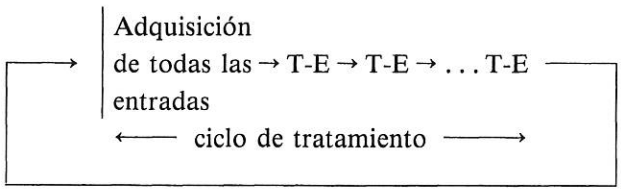


En este caso, se dice que el sistema funciona en modo *síncrono con respecto a las entradas y salidas*, aunque la duración del ciclo no será forzosamente constante. Este modo de funcionamiento necesita una memoria imagen de entradas y salidas.

---

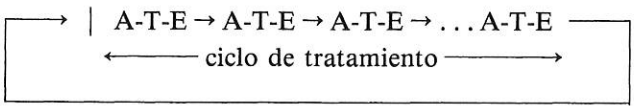
† Se supone que θ es superior al tiempo que tarda el más duradero CT de la aplicación.

(2) *Adquisición en bloque* (AB) y *emisión directa* (ED)†:



La adquisición en bloque implica la necesidad de una memoria imagen de las entradas. Las salidas se emiten a medida que se realiza su cómputo. La emisión directa de las salidas puede presentar problemas de continuidad en las acciones; de este modo, debe evitarse que, aunque transitoriamente, se ordene a un motor girar en ambos sentidos a la vez.

(3) *Adquisición directa* (AD) y *emisión directa* (ED)†:



La adquisición directa implica que una variable de entrada pueda ser considerada en un mismo ciclo con diversos valores, lo cual puede conducir a importantes problemas. La única ventaja que presenta es poder considerar varias veces en un ciclo a alguna variable especialmente crítica. No obstante, su empleo es peligroso.

Además de estos tres esquemas básicos de ciclo de tratamiento, la presencia de un procesador de entrada/salida puede introducir otros. Así, por ejemplo, se encuentran sistemas en los que las entradas son adquiridas y las salidas emitidas cada θ milisegundos, independientemente de la duración del ciclo de tratamiento.

Con frecuencia los AP incorporan dispositivos que supervisan parcialmente el buen desarrollo de un ciclo de tratamiento. En su forma más simple, éstos supervisan la duración de cada ciclo. Si un determinado ciclo tarda en ejecutarse más que la máxima duración prevista, se generará una alarma que advertirá a los operadores y, eventualmente, bloqueará la evolución del AP. El mencionado mecanismo, en esencia un monoestable rearmable, se denomina *perro guardián (watchdog)*.

8.3 LENGUAJES DE PROGRAMACIÓN

La programación de los autómatas programables es extremadamente simple, debido a que sus lenguajes están adaptados a la forma de expresión del usuario de automatismos lógicos. Son *lenguajes orientados al problema.*

A partir de esta premisa, se observan en los lenguajes dos concepciones bastante diferentes según que la lógica programada sea considerada:

- como transposición tecnológica inmediata de los *esquemas de realización cableada* de los automatismos
- o como transcripción, en forma programada, de una *descripción funcional* de los automatismos.

† A, adquisición; E, emisión; T, tratamiento.

La principal ventaja que ofrecen los sistemas del primer grupo, es la de no modificar en absoluto el modo de razonamiento de los técnicos de mantenimiento (esquema cableado). Por consiguiente, sus lenguajes de programación poseen instrucciones adaptadas a la *descripción de componentes* (relés, módulos Y-O-MEMORIA, etc.). Desgraciadamente, estos tipos de lenguajes poseen diversas limitaciones entre las que, con frecuencia, cabe destacar una restringida capacidad expresiva, e incluso una relativa ambigüedad en la descripción de ciertos esquemas.

Las herramientas de descripción funcional de automatismos lógicos en las que se basan los lenguajes del segundo grupo son los *organigramas*, los *grafos reducidos*, las *redes de Petri*, etc. La utilización de cualquiera de estas herramientas de descripción se encuentra normalmente completada con la incorporación en el lenguaje de la notación *booleana*, lo cual facilita la descripción de funciones combinacionales.

Desde un punto de vista histórico, los lenguajes del primer grupo han sido utilizados desde el nacimiento de los AP hacia 1969, mientras que los lenguajes del segundo grupo aparecieron hacia 1975.

Con el objeto de simplificar y de adaptar los lenguajes de programación a la clase de problemas para la que son concebidos, normalmente éstos tienen como características básicas las siguientes:

1) La existencia de *muy pocas instrucciones* (típicamente de 10 a 35).
2) La utilización casi exclusiva del *direccionamiento directo* de operandos. Si existen instrucciones de salto (incondicional y/o condicional), suelen utilizarse los direccionamientos *directo* y/o *relativo* al contador de programa.
3) La noción de *subprograma* aparece raramente y, en caso de estar incluida, se suele definir un único nivel de anidamiento.
4) La ausencia de *interrupciones* manipulables por el programador.

A continuación, presentaremos diversos tipos de lenguajes de programación que evidencian diferentes herramientas para la descripción de automatismos lógicos. Los dos primeros (§8.3.1 y 2) se basan en los esquemas de relés y en los diagramas lógicos, respectivamente. Son lenguajes que pertenecen claramente al grupo que preconiza la transposición tecnológica inmediata de los esquemas de realización cableada.

El tercer lenguaje que se presenta (§8.3.3) utiliza la notación algebraica de BOOLE para describir las funciones combinatorias. La presencia o ausencia de saltos condicionales tiene un enorme impacto sobre la programación de funciones secuenciales y concurrentes.

Por último, en el capítulo 9 (§9.2) se presentará un tipo de lenguaje de especificación de descripciones basadas en RdP. Al margen de su adaptación a la descripción de RdP, estos lenguajes exhiben características muy diferentes a las de los anteriores.

Para facilitar la comprensión, se presentarán lenguajes simplificados. En éstos se considerarán tan sólo las instrucciones fundamentales que posibilitan la programación de las funciones lógicas; es decir, lo que podríamos denominar el *lenguaje básico*. Entre las instrucciones que típicamente suelen añadirse al lenguaje básico, se encuentran aquellas que conciernen a las temporizaciones y a los contadores.

8.3.1 Lenguaje basado en los esquemas de contactos (ladder diagrams)

Adaptados a las descripciones realizadas con elementos de tipo electrotécnico (*relés*) comprenden, por ejemplo, las siguientes instrucciones:

1) —| |— X ; contacto X abierto
2) —|/|— X ; contacto X cerrado
3) —( )— Y ; salida Y
4) ⌐ ; apertura de rama
5) ↙ ; retorno a la última rama abierta
6) ⌐ ; cierre de una rama

De este modo, si se considera el esquema de la figura 8.5*a*, se obtendrá como programa el de la figura 8.5*b*. Se recomienda el estudio del programa, fijándose, en particular, en la utilización de la instrucción «↙—». Debe observarse que, aunque, a primera vista pudiera pensarse en la asociación:

⌐ = apertura de paréntesis
⌐ = cierre de paréntesis
↙ = unión lógica

} en la notación booleana

ésta no es válida (compruébese a partir de la figura 8.5).

A pesar de que la idea motriz que conduce al diseño de los lenguajes de este grupo es la de transcribir directamente los esquemas de relés, surgen dos tipos de dificultades en la programación:

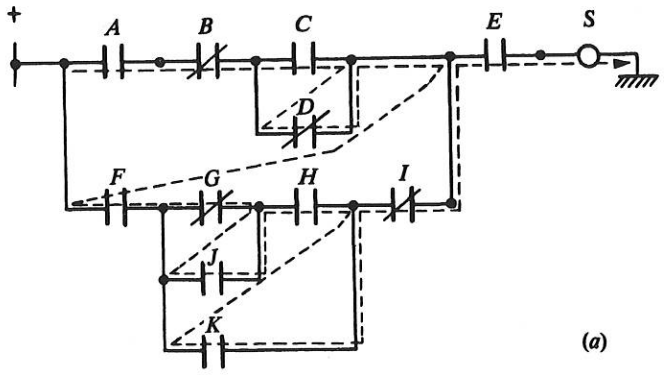
1) Dificultad de tipo *estructural*: sólo los esquemas *serie-paralelo* pueden programarse directamente. En efecto, los lenguajes existentes no permiten la programación directa (sin transformación) de los esquemas *en puente* (figura 8.6)
2) Dificultad de tipo *dinámico*: puesto que la ejecución del programa es secuencial, el orden en que se alinean las instrucciones es fundamental. Esta importante diferencia entre tratamiento secuencial del programa y evolución paralela en el modelo construido con relés, puede originar falsas acciones o secuencias de acciones.

Las aleatoriedades producidas por una interpretación secuencial de un modelo que evoluciona en paralelo (esquema de contactos, diagrama lógico, RdP, etc.) se denominarán genéricamente *aleatoriedades de programación*.

Las peculiaridades de los lenguajes basados en los esquemas de contactos hacen que en los sucesivo no se les vuelva a considerar. No obstante, es obligado señalar que su utilización es masiva en equipos procedentes de EEUU, donde argumentan que el verdadero usuario de un AP es el técnico de mantenimiento, hoy en día, normalmente, de formación electrotécnica.

8.3.2 Lenguaje basado en los diagramas lógicos

En este parágrafo vamos a distinguir dos puntos. Después de presentar el lenguaje (§8.3.2.1), consideraremos un modo de programar redes de Petri con éste (§8.3.2.2).



(*a*)

1		13	
2	A	14	
3	B	15	J
4		16	
5	C	17	H
6		18	
7	D	19	K
8		20	
9		21	I
10	F	22	
11		23	E
12	G	24	S

(*b*)

Figura 8.5. *a*) Esquema de contactos (serie-paralelo).
b) Programa que permite simular el esquema de contactos.

8.3.2.1 *El lenguaje*

Este tipo de lenguajes permite la descripción de un diagrama lógico, *módulo* por *módulo*. Comprenden, por ejemplo, las instrucciones siguientes:

1) ENT VAR ; Entrada a un módulo de la variable VAR
2) ENTC VAR ; Entrada a un módulo de la variable VAR complementada

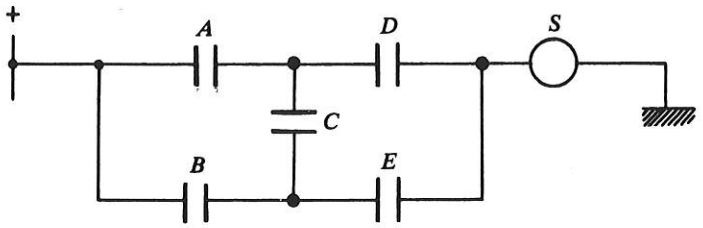
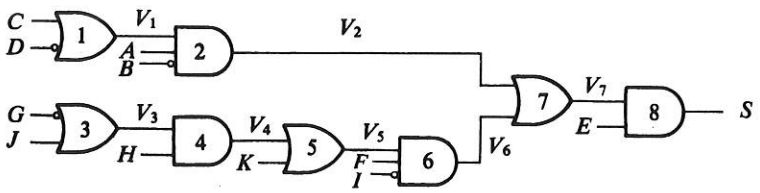


Figura 8.6 Esquema de contactos en puente.

```
3) Y VAR     ; El valor de la intersección lógica de las variables definidas co-
             ; mo entradas se asigna a la variable VAR (OPERACIÓN Y.)
4) O VAR     ; Asignación a la variable VAR de la unión lógica de las
             ; entradas al módulo (OPERACIÓN O.)
5) ACT VAR   ; Activación de la variable VAR si el resultado de la intersec-
             ; ción (FUNCIÓN Y) de las variables de entrada al módulo es «1»
6) DACT VAR; Desactivación de la variable VAR, condicionada por la intersec-
             ción considerada en la instrucción anterior
7) SAL DIR   ; Saltar a la dirección DIR
```

Si consideramos el diagrama lógico de la figura 8.7*a*, se podrá escribir un programa como el de la figura 8.7*b*. Debe observarse, sin embargo, que no es válida la



(*a*)

```
(b)   ENT  C        ENTC G        ENT  V4        ENT  V2
      ENTC D  } 1    ENT  J  } 3   ENT  K  } 5   ENT  V6  } 7
      O    V1       O    V3       O    V5        O    V7

      ENT  V1       ENT  V3       ENT  V5        ENT  V7
      ENT  A        ENT  H        ENT  F         ENT  E
      ENTC B  } 2   Y    V4  } 4   ENTC I  } 6    Y    S   } 8
      Y    V2                     Y    V6
```

Figura 8.7 Diagrama lógico (realiza la función *S* de la figura 8.5*a*) y programa que permite simularlo.

escritura de los módulos en un orden cualquiera. Así pues, siempre deberá escribirse en último lugar el módulo 8 y delante de éste el 7. Para construir un programa correcto (que no genere aleatoriedades de programación), *es necesario calcular la salida de un módulo después de haber calculado todas sus entradas.* Por consiguiente, la programación de un sistema secuencial fuertemente realimentado puede conducir a problemas de difícil solución.

Para encadenar el siguiente ciclo de tratamiento, se insertará al final de cada programa una instrucción «SAL dirección inicial».

En algunos lenguajes pertenecientes a este grupo, se encuentran instrucciones que permiten programar directamente otros módulos como son los NO-Y (NAND), NO-O (NOR), O-EXCLUSIVO (EXOR), PASO-A-PASO, etc. No obstante, las instrucciones presentadas permiten programar con facilidad algunos módulos que no aparecen explícitamente. En efecto, aplicando las leyes de MORGAN del algebra de BOOLE puede escribirse:

$$VNY = \text{NO-Y}\,(A, B) = \text{O}(\bar{A}, \bar{B}) \left\{ \begin{matrix} \text{ENTC } A \\ \text{ENTC } B \\ \text{O } VNY \end{matrix} \right\} \text{ módulo NO-Y (NAND)}$$

$$VNO = \text{NO-O}\,(A, B) = \text{Y}(\bar{A}, \bar{B}) \left\{ \begin{matrix} \text{ENTC } A \\ \text{ENTC } B \\ \text{Y } VNO \end{matrix} \right\} \text{ módulo NO-O (NOR)}$$

Por otro lado, es frecuente que, después de definir un conjunto de variables de entrada, se pueda definir una lista de salidas (asignaciones), correspondientes a módulos con idénticas entradas. Así por ejemplo, la función O-exclusivo, OEX, (EXOR), puede programarse de la forma siguiente:

$$VOX = \text{OEX}\,(A, B) = \underset{V_1}{A\bar{B}} + \underset{V_2}{\bar{A}B} \left\{ \begin{matrix} 1)\ \text{ENT} & A & 5)\ \text{ENT} & V_1 \\ 2)\ \text{ENTC} & B & 6)\ \text{ENTC} & V_2 \\ 3)\ Y & V_1 & 7)\ \text{O} & VOX \\ 4)\ \text{O} & V_2 & & \end{matrix} \right\} \text{módulo OEX}$$

Por último, las instrucciones ACT y DACT permiten una programación directa de los *biestables*. De este modo, resultan evidentes los programas siguientes:

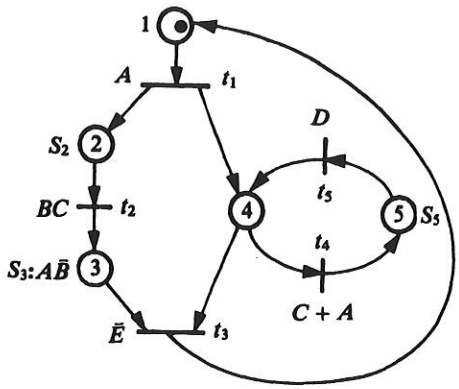
Biestable R-S con activación prioritaria:	**Biestable R-S con desactivación prioritaria:**
ENT R	ENT S
DACT Q	ACT Q
ENT S	ENT R
ACT Q	DACT Q

8.3.2.2 *Programación síncrona y no-síncrona de RdP*

Las RdP, así como los circuitos lógicos, evolucionan en *paralelo*. Dado que los AP son máquinas secuenciales, éstos procederán *simulando en serie* las evoluciones de las RdP.

Utilizando el lenguaje presentado en el apartado anterior, para codificar RdP se puede proponer un primer esquema basado en la simple transcripción del diagrama lógico que se obtiene al realizarla por transferencia impulsional (§6.2). Para proceder de forma sistemática, lo más natural parece ser la codificación en dos fases (figura 8.8b):

1) A cada *transición* se le asocia el código que permita evaluar la condición de disparo (intersección de la condición de sensibilización y del evento asociado). Además, en el caso en que haya que disparar la transición, se dispondrá el código que permita desactivar sus lugares de entrada, activar sus lugares de salida y, eventualmente, activar las acciones impulsionales asociadas.
2) A cada *lugar* se le asocia el código que posibilite la elaboración de las acciones asociadas.



(*a*)

FASE 1			FASE 2		
	...			...	
δt_3:	ENTC	E	δp_3:	ENT	A
	ENT	p_3		ENTC	B
	ENT	p_4		ENT	p_3
	ACT	p_1		Y	S_3
	DACT	p_3		...	
	DACT	p_4			
	...				

(*b*)

Figura 8.8. RdP y programación no-síncrona. (*Nota.* δt_i y δp_j representan las direcciones donde comienzan los segmentos de código asociados a t_i y p_j, respectivamente.)

Observación. La técnica sugerida para programar RdP es muy sistemática, pero puede ser simplificada en ciertos casos. Así, por ejemplo, en la codificación de la red de la figura 8.8*a* se puede tomar $p_2 \equiv s_2$ y $p_5 \equiv s_5$. Con ello, además de no utilizar las variables p_2 y p_5, se reducirá la longitud del programa y la duración de su ejecución.

Por desgracia, la técnica de programación presentada anteriormente puede conducir a ciertos funcionamientos anómalos de los designados con el nombre genérico de *aleatoriedades de programación*. Esencialmente, éstas derivan de la simulación *en serie* (secuencial) de una clase de modelos (RdP) que evolucionan *en paralelo*. Para ilustrar esta afirmación, se van a considerar dos casos, basados en las redes de las figuras 6.3*a* y 8.8*a*:

CASO 1. Sea la RdP de la figura 6.3*a*. Si dado que la red es binaria, el marcado se representa mediante un vector de booleanos, p_j, la simulación será incorrecta siempre que se programe t_1 antes que t_2. En efecto si se dispara t_1 antes que t_2, para la red considerada se alcanzaría transitoriamente un mar-

cado no-binario, valor cuya representación no es posible con un vector booleano de marcados.

CASO 2. Sea la RdP de la figura 8.8*a*. Supóngase que las transiciones se programan en el orden $t_1t_2t_3t_4t_5$. Si al iniciarse un CT se tiene $M(p_1)ABC = 1$, el marcado de la red evolucionará directamente desde p_1 a p_3-p_4. Por consiguiente, si las salidas se emitiesen *en bloque* (al final del CT), S_2 no sería emitida, con lo que la simulación sería incorrecta.

Como puede observarse fácilmente, para resolver los dos problemas planteados, basta con programar t_2 antes que t_1, lo que revela que *el resultado de un* CT depende del orden en que se programen las transiciones. Aunque factible, la adopción del principio de jugar con el orden en que se programen las transiciones puede complicar sensiblemente la programación en un caso general.

Un método simple y sistemático para resolver aleatoriedades de programación consiste en hacer que la RdP *evolucione de forma síncrona*. Es decir, la RdP se codificará de forma que su marcado evolucione en un ciclo de tratamiento como si estuviese realizada con biestables síncronos. Obsérvese que la definición de simulación síncrona adoptada no exige la presencia de un reloj tiempo real que lance los ciclos de tratamiento a una cadencia fija, ni que las salidas sean emitidas cuasi-simultáneamente.

La simulación síncrona de una RdP se puede dividir en tres fases:

1) Cálculo de las *condiciones de evolución del marcado*, en la que se pueden elaborar las condiciones de disparo de las transiciones o, directamente, las condiciones de marcado-desmarcado de los lugares.
2) *Actualización del marcado*; es decir, cálculo del nuevo *marcado*.
3) *Elaboración de las acciones* asociadas a los lugares.

FASE 1		FASE 2		FASE 3	
...		...		...	
δt_3: ENTC	E	$\delta\theta_3$: ENT	θ_3	δp_3: ENT	A
ENT	p_3	ACT	p_1	ENTC	B
ENT	p_4	DACT	p_3	ENT	p_3
Y	θ_3	DACT	p_4	Y	S_3
...		...		...	

Figura 8.9 Una programación síncrona de la RdP de la figura 8.8*a* (θ_3 representa la condición de disparo de t_3).

La figura 8.9 presenta una programación síncrona de parte de la RdP de la figura 8.8*a*. En ella la actualización del marcado se hace transición por transición, a partir de sus condiciones de disparo. Ahora bien, la programación de la RdP de la figura 6.3*a* no es correcta si en la actualización del marcado se considera t_1 antes que t_2; es decir, sigue vigente parte del problema de aleatoriedades de programación, habida cuenta que la activación de un lugar debe ser prioritaria sobre la desactivación.

La forma más inmediata y sistemática de resolver la dificultad apuntada consiste en considerar dos subfases en la fase de actualización del marcado:

1) Subfase de *desactivación de lugares*, en la que a partir de las condiciones de disparo de las transiciones se desactivarán sus lugares de entrada:

```
...
ENT    θ3
DACT   p3
DACT   p4
...
```

2) Subfase de *activación de lugares*, en la que a partir de las condiciones de disparo de las transciones se activarán sus lugares de salida:

```
...
ENT    θ3
ACT    p1
...
```

En resumen, se ha de resaltar que la simulación *serie* de un modelo que posee *evoluciones simultáneas* presenta dificultades para su programación. La programación síncrona resuelve de forma sistemática el problema de la simulación de RdP, pero necesita mayor cantidad de código, así como un mayor tiempo para la ejecución de los programas que la programación no-síncrona.

8.3.3 Lenguajes basados en la notación booleana

Estos lenguajes están normalmente desprovistos de primitivas que faciliten la programación de automatismos con memoria; no obstante, a veces, se encuentran complementados con instrucciones de salto condicional, lo cual permite la programación de descripciones realizadas con organigramas.

Los esquemas de las figuras 8.5 y 8.7 se programan, por ejemplo, mediante la asignación lógica siguiente:

$$S := (A\bar{B}(C + \bar{D}) + F((\bar{G} + J)H + K)\bar{I})E.$$

La interpretación y ejecución de las expresiones lógicas puede realizarse de diversas maneras. Si éstas se realizan de forma *directa*, la expresión se encontrará, símbolo por símbolo, en la memoria del AP. En este caso, para facilitar la ejecución, la variable a la que debe asignársele el valor de la función, se suele escribir en último lugar:

$$(A\bar{B}(C + \bar{D}) + F((\bar{G} + J)H + K)\bar{I})E =: S.$$

Habida cuenta que la ejecución directa de expresiones lógicas es complicada, a veces, se imponen ciertas restricciones a la escritura algebraica; de este modo, se observa que con frecuencia sólo puede haber *un único nivel de paréntesis*. En otros sistemas, se admite la apertura de varios niveles de paréntesis, pero *un paréntesis de cierre* produce la cerrazón de todos los paréntesis abiertos con anterioridad. En

estas circunstancias, la expresión algebraica anterior debe transformarse, por ejemplo, en la siguiente:

$$A\bar{B}E(C + \bar{D}) + F\bar{I}E(K + H(\bar{G} + J) =: S.$$

La mayor parte de las veces, la expresión algebraica es traducida a otro lenguaje más fácilmente ejecutable; es decir, se *compila* el programa del usuario.

A continuación, se considera la programación de RdP utilizando exclusivamente ecuaciones algebraicas de BOOLE. En §8.3.3.2 se presentará el impacto que, sobre la programación de RdP, supone la introducción de instrucciones de salto condicional en el lenguaje.

8.3.3.1 *Lenguaje exclusivamente basado en la notación algebraica y programación de RdP*

En el capítulo 6 se vió que un lugar se puede materializar con una memoria de activación prioritaria, cuya ecuación es la siguiente (§6.2.1):

$$Y_j := C_{Aj} + Y_j \bar{C}_{Dj}.$$

Por lo tanto, si se realiza una programación síncrona, ésta podrá responder al esquema que se presenta a continuación (en todas las fases se procede lugar por lugar):

1) Cálculo de las C_{Aj} y C_{Dj} (funciones de las condiciones de disparo de las transiciones).
2) Cálculo del nuevo marcado: $Y_j := C_{Aj} + Y_j\bar{C}_{Dj}$.
3) Cálculo de las acciones asociadas a los lugares.

Como puede comprobarse, en esencia la programación síncrona de la figura 8.9 y la aquí sugerida sólo difieren en la fase de *actualización del marcado* que en este caso se realiza lugar por lugar. Precisamente, dado que la actualización del marcado se realiza lugar por lugar, la programación de RdP binarias que podrían conducir a marcados transitoriamente no-binarios (figura 6.3*a*) no planteará problemas.

La ecuación que representa un lugar puede incorporar de forma sistemática la entrada de *inicialización general*, I. En este caso conviene distinguir los lugares inicialmente marcados de los otros. Las ecuaciones serán las siguientes:

1) Si p_j debe estar inicialmente marcado, se puede escribir $C^*_{Aj} = C_{Aj} + I$, lo cual implica $Y_j := C_{Aj} + Y_j\bar{C}_{Dj} + I$
2) Si p_j debe estar inicialmente desmarcado, se puede escribir $C^*_{Aj} = C_{Aj}\bar{I}$ y $C^*_{Dj} = C_{Dj} + I$, lo cual implica $Y_j :: = (C_{Aj} + Y_j\bar{C}_{Dj})\bar{I}$; es decir, I es una desactivación prioritaria.

8.3.3.2 *Lenguaje provisto de saltos condicionales y programación de RdP*

La programación de RdP puede realizarse siguiendo los esquemas utilizados anteriormente (§8.3.2.2 y §8.3.3.1). Al utilizarse los saltos condicionales, no se ejecutarán exhaustivamente todas las instrucciones de los programas. La ejecución será más rápida.

Considerando la actualización del marcado se podrá escribir:

1) *Tratamiento transición por transición* (§8.3.2.2):

   δt_i: **Si** $\bar{\Theta}_j$ **ir a** δt_{j+1};

   «activar los lugares de salida de t_j»;

   «desactivar los lugares de entrada de t_j»;

2) *Tratamiento lugar por lugar* (§8.3.3.1):

   δp_j: **si** $\bar{C}_{Dj}$ **ir a** γp_j;

   «desactivar Y_j»;

   γp_j: **si** $\bar{C}_{Aj}$ **ir a** δp_{j+1};

   «activar Y_j»;

EJERCICIO. ¿En qué medida son extensibles a las RdP no-binarias técnicas de programación de RdP binarias?

EJERCICIO. Si se utilizaran contadores para representar el marcado de los lugares, no sería necesario distinguir dos subfases en la fase de actualización del marcado. ¿Por qué?

8.4 PRESENTACIÓN SIMPLIFICADA DE ALGUNOS AUTÓMATAS PROGRAMABLES

La diversidad de principios que se emplean en la interpretación de diagramas y expresiones lógicas nos han impulsado a presentar, desde un punto de vista funcional, diversos tipos de máquinas†. Para facilitar la lectura global del apartado, se mostrará una rápida panorámica en §8.4.1; posteriormente, en §8.4.6 se resumirá de forma tabular (tabla 8.4) los principales rasgos considerados.

La presentación de los diferentes tipos de máquinas adoptará siempre el siguiente esquema (en algún caso, dos apartados se fusionarán en uno solo):

a) Presentación intuitiva del lenguaje de la máquina en forma simbólica (*ensamblador*).

b) Definición precisa de su semántica, basándose en la presentación de una estructura de unidad de tratamiento y de un *lenguaje de fases* asociado.

c) Comentarios, en los que se resaltarán algunas de sus peculiaridades.

Para precisar la semántica de las instrucciones de los lenguajes, punto *b*, se utilizará un formalismo que expresa las *transferencias de información* entre registros y/o posiciones de memoria.

Sus dos reglas básicas son:

1) La colocación del nombre de un registro entre paréntesis designa su *contenido*. Así, (A) designa al contenido de A. Por otro lado, ((A)) designa al contenido de la posición de memoria designada por el contenido de A, (A).

† Todos ellos están inspirados en sistemas reales.

2) «←» es el operador de *transferencia de información*. De este modo, A ← (B) expresa que el contenido de B se transfiere al registro A; es decir, A : = B. Por otro lado, A ← (B) *op* (C) expresa que el resultado de la operación *op* se tranfiere al registro A; es decir, A : = B *op* C.

Si RI y CP designan el *registro de instrucción* y el *contador del programa* de una máquina, de acuerdo con la notación anterior, la búsqueda de una instrucción se expresará globalmente por la transferencia RI ← ((CP)). Del mismo modo, la preparación para la ejecución de la siguiente instrucción se expresará: CP ← (CP) + 1.

En lo sucesivo, además de las anteriores denominaciones, se utilizarán las siguientes:

- D[RI], subregistro de RI que almacena la parte de dirección contenida en la instrucción.
- T, registro de trabajo, transparente al programador, denominado *tampón*.
- ϕ_i, *i*-ésima subfase de la ejecución específica de una instrucción. ϕ_0 será la búsqueda (fetch) de la instrucción. (*Nota*. Siempre se supondrá que al comenzar ϕ_1 ya se ha incrementado el CP; es decir, se ha realizado la preparación para la ejecución de la instrucción siguiente.)

8.4.1 Presentación general

Los cuatro sistemas AP que presentaremos en sucesivos apartados responden a concepciones diferentes. Éstos son los siguientes:

1) *Interpretador de diagramas lógicos*. Sistema diseñado para ejecutar directamente los programas escritos en el lenguaje basado en *diagramas lógicos* (§8.3.2). Necesita una unidad lógica con *dos acumuladores*: uno acumula las *intersecciones*, mientras que el otro acumula, simultáneamente, las *uniones*.
2) *Interpretador de expresiones lógicas postfijas*. Sistema que ejecuta, de forma muy eficiente, las *cadenas postfijas* (éstas se pueden obtener a partir de la notación algebraica infija).

   La eficiencia de este tipo de sistemas se incrementa notoriamente cuando se codifican en instrucciones secuencias de símbolos (operadores y variables) que aparecen con frecuencia en las cadenas postfijas. Se emplea una *pila* (especial) para la evaluación de las expresiones.
3) *Máquina basada en el concepto de operación lógica*. Sistema que corresponde a la estructura clásica del computador de próposito general. Se trata de una máquina con instrucciones de una dirección, lo cual hace necesario la presencia de *un registro acumulador*. Los cálculos se describen exclusivamente en términos de transferencias entre registros.
4) *Interpretador mixto de operaciones lógicas y saltos condicionales*. Sistema similar al anterior, desde el punto de vista estructural. Las prestaciones de la evaluación de expresiones lógicas se mejora al introducir el *salto condicional*. En efecto, de esta forma se evita la evaluación de subexpresiones que no pueden alterar el resultado global de una expresión.

Además de estos tipos de sistemas, se deben mencionar:

1) *Las máquinas de decisiones binarias* (véase §7.3.3), que, como es fácil comprobar, no poseen ninguna unidad lógica de cálculo.

2) *Los analizadores booleanos de estados finitos*, que ejecutan directamente todas aquellas expresiones booleanas en las que un único paréntesis de cierre provoca el cierre de todos los paréntesis previamente abiertos. Esta clase de sistemas utiliza *un autómata síncrono de estados finitos* ([SILV 78]) para calcular el valor de la expresión. Su relativa especificidad y complejidad conceptual han hecho que no los consideremos en este texto.

8.4.2 Interpretador de diagramas lógicos (M_1)

a) *Lenguaje ensamblador de la máquina* (L_1)

El lenguaje del sistema que se considera en este apartado es el presentado en §8.3.2.

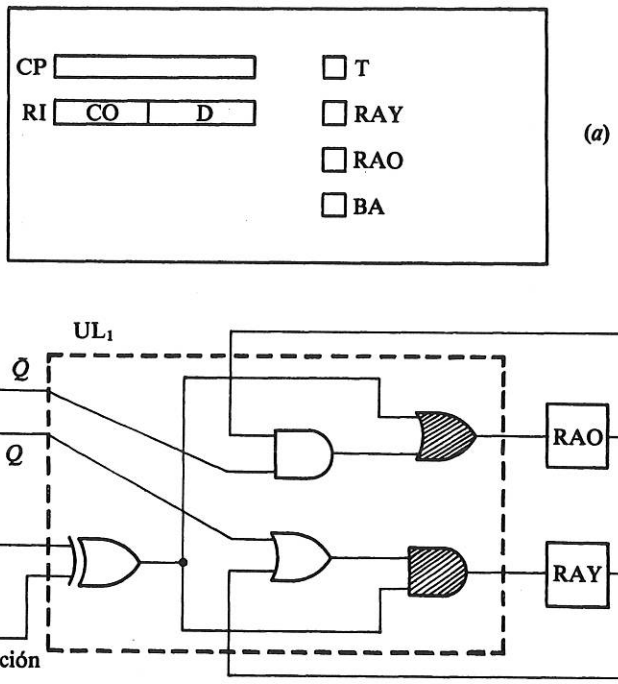


Figura 8.10. Arquitectura de la CPU_1 y posible esquema lógico de la UL_1. (*Nota.* T, RAO y RAY son biestables D, mientras que BA es R-S.)

b) *Estructura de la unidad de tratamiento y lenguaje de fases* (figura 8.10 y tabla 8.1)

Puesto que en todo programa codificado en L_1 se define el tipo de módulo lógico (Y, O, . . .) después de especificar sus entradas, esta UT utiliza dos registros acumuladores, uno destinado al cálculo de la *intersección* (operación Y, RAY) y otro destinado al cálculo de la *unión* (operación O, RAO).

	ϕ_1	ϕ_2
ENT V	T ← ((D[RI]));	**si** (BA) = 1 **entonces** RAY ← (T), RAO ← (T), BA ← 0; **si no** RAY ← (T) ∧ (RAY), RAO ← (T) ∨ (RAO);
ENTC V	T ← ((D[RI]));	**si** (BA) = 1 **entonces** RAY ← ($\bar{T}$), RAO ← ($\bar{T}$), BA ← 0; **si no** RAY ← ($\bar{T}$) ∧ (RAY); RAO ← ($\bar{T}$) ∨ (RAO);
Y V	(D[RI]) ← (RAY), BA ← 1;	—
O V	(D[RI]) ← (RAO), BA ← 1;	—
ACT V	**si** (RAY) = 1 **entonces** (D[RI]) ← 1, BA ← 1; **si no** BA ← 1;	—
DACT V	**si** (RAY) = 1 **entonces** (D[RI]) ← 1, BA ← 1; **si no** BA ← 1;	—
SAL d	CP ← (D[RI]), BA ← 1;	—

Tabla 8.1. Descripción de L_1 mediante lenguaje de fases.

El registro BA es un registro de estado de un bit, denominado *bit de asignación*. Su misión es la de diferenciar si la ejecución de las instrucciones ENT y ENTC se ha realizado después de:

(1) Una instrucción de asignación, {O, Y, ACT, DACT}, o de salto, {SAL}, en cuyo caso (BA) = 1.
(2) Una instrucción ENT o ENTC, en cuyo caso (BA) = 0.

Cuando (BA) = 1, la ejecución de las instrucciones ENT y ENTC produce la *carga* de los registros acumuladores RAO y RAY. Cuando (BA) = 0 se realizan las correspon-

dientes *acumulaciones* sobre RAO y RAY. Es decir, la presencia del registro BA hace que las instrucciones ENT y ENTC puedan servir como instrucciones de carga o de acumulación. Dicho de otro modo, si no existiese el registro BA, tendrían que sustituirse las instrucciones ENT y ENTC por las correspondientes de *carga*, CARENT y CARENTC y *acumulación*, ACENT y ACENTC.

La figura 8.10*b* muestra la simplicidad de la *unidad lógica*. Ésta se obtiene directamente a partir de las diferentes asignaciones a RAO y RAY, expresadas en la tabla 8.1 ($\varnothing_2$ de las instrucciones ENT y ENTC). Las puertas sombreadas en la figura 8.10*b* son las que, en esencia, multiplexadas en el tiempo, permiten el cómputo de cualquier función lógica.

c) Comentarios

C.1 Dado que la única instrucción de salto, SAL, es incondicional, todas las instrucciones del programa serán ejecutadas en cada ciclo de tratamiento; por ello, diremos que la máquina M_1 efectúa un *barrido exhaustivo* del programa.

C.2 La ausencia de una instrucción que indique el comienzo de un nuevo ciclo de tratamiento permite pensar que éste se desarrollará utilizando una admisión y emisión de información directas. Para ello se supone que no existe convención por la cual la ejecución de la instrucción de una determinada dirección, Δ, representa el comienzo de un ciclo (Δ podría ser, por ejemplo, la primera dirección del espacio memoria de programa).

C.3 Las primitivas ACT y DACT, poco usuales, permiten la programación simplificada de las RdP (§8.3.2).

C.4 El traductor que genere el código máquina será un simple ensamblador, encargado de sustituir el nombre simbólico de: (1) cada variable por su dirección y (2) cada código de operación por su código máquina.

8.4.3 Interpretador de cadenas postfijas (máquina de pila) (M_2)

La notación polaca postfija es una convención de escritura por la que los paréntesis de una expresión son suprimidos. En ella los operadores figuran a continuación de los operandos. Por ejemplo, $S_1 := F \cdot ((\bar{G} + J) \cdot H + K) \cdot \bar{I}$ se escribirá: $S_1 F \bar{G} J + H \cdot K + \cdot \bar{I} \cdot :=$, por lo que resulta fácil observar que:

1) El orden de los operandos es el mismo en ambos casos.
2) Una cadena postfija se analiza de izquierda a derecha. Las operaciones deberán realizarse a medida que se encuentren los operadores. Éstos se aplicarán sobre los dos operandos situados inmediatamente a su izquierda (eventualmente después de haber operado).

En el ejemplo desarrollado anteriormente, el operador «·», situado a la izquierda de I, tiene efecto sobre F y OP_3:

$$
\begin{cases}
F\bar{G}J + H \cdot K + \cdot \bar{I} \cdot \\
\quad \overset{*}{\smile} \\
\quad OP_1 \ldots\ldots\ldots\ldots OP_1 = \bar{G} + J \\
\quad\quad \overset{* \ *}{\smile} \\
\quad\quad OP_2 \ldots\ldots\ldots OP_2 = OP_1 \cdot H \\
\quad\quad\quad \overset{* \ *}{\smile} \\
\quad\quad\quad OP_3 \ldots\ldots\ldots OP_3 = OP_2 + K
\end{cases}
$$

En nuestro contexto el interés de esta notación es doble puesto que:

- permite una *compilación* elemental de las expresiones algebraicas, sin tener que considerar restricción alguna sobre ellas.
- la ejecución de un programa puede ser muy rápida; además, ésta puede ser acelerada siempre que se utilicen ciertos artificios. Algunos de éstos los estudiaremos más adelante.

Para fijar ideas, podemos definir un *lenguaje ensamblador elemental* adaptado a esta notación, que comprenda las instrucciones siguientes:

(1) LL V ; Llamada de la variable V
(2) LLC V ; Llamada de la variable V Complementada
(3) + ; Unión lógica de los dos últimos operandos
(4) • ; Intersección lógica de los dos últimos operandos
(5) := V ; Asignación a la variable V

Con este lenguaje, podemos codificar la cadena anterior mediante el siguiente programa (12 instrucciones):

1) LL F	5) LL H	9) •
2) LLC G	6) •	10) LLC I
3) LL J	7) LL K	11) •
4) +	8) +	12) := S_1

Como fué anunciado anteriormente (§8.4.1), se puede incrementar la eficiencia de este tipo de sistemas mediante la *codificación*, en instrucciones independientes, de *subsecuencias* de símbolos que aparezcan con frecuencia en las cadenas postfijas. En las observaciones que siguen se desecha la codificación de ciertas subsecuencias y se comprueba que siempre será necesario disponer de los operadores aislados (sin operandos). Posteriormente, se define un lenguaje «optimizado» que posibilita la escritura de programas más cortos y rápidos.

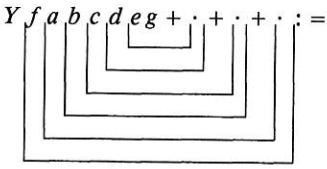
Observación 1. Debido a la propiedad de asociatividad de los operadores « + » y «·», toda cadena postfija puede ser transformada de tal forma que nunca dos operadores idénticos estén consecutivos. Por ejemplo: $s_2 = a \cdot b \cdot c \cdot d$, se puede escribir:

$$s_2\, a\, b\, c\, d \cdots := \quad \text{de donde la cadena} \quad s_2\, c\, d \cdot b \cdot a \cdot :=$$

Evidentemente, $s_2\, a\, b \cdot c \cdot d \cdot :=$ es también una cadena correcta. •

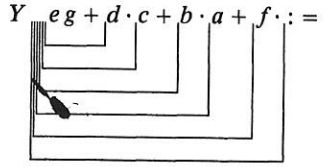
Observación 2. Normalmente (no siempre) las sucesiones de tres operadores « + · + » y « · + ·» pueden ser eliminadas.

De forma intuitiva, se expone a continuación una regla para transformar la cadena «postfija directa», obtenida a partir de la notación infija. Sea la siguiente asignación:
$Y = f \cdot (a + b \cdot (c + d \cdot (e + g)))$. Escrita de forma «postfija directa» resulta la cadena siguiente:



REGLA DE FORMACIÓN DE LA NUEVA CADENA:
(1) Se coloca el nombre simbólico de la función, Y;
(2) Cada vez que aparezcan consecutivamente los dos operandos y el operador, se les sitúa al final de la cadena en construcción. □

Procediendo de esta forma, la cadena anterior se transforma en la siguiente:



Observación 3. No siempre es posible eliminar toda secuencia de tres operadores. Para convencerse de ello, basta con analizar la expresión siguiente:

$$Z{:=}\,(a \cdot b + c \cdot d) \cdot (e \cdot f + g \cdot h) \Rightarrow Z\,ab \cdot cd \cdot + ef \cdot gh \cdot + \cdot :=$$

Las observaciones anteriores vienen a señalar que:

1) se pueden evitar las secuencias de operadores «··» y «+ +».
2) si no se codifican las ternas de operadores «+ · +» y «· + ·» en macrooperaciones, es necesario disponer de operadores aislados (sin parámetros): «+» y «·».

El lenguaje que se presenta a continuación dispone de los operadores aislados, de éstos con una variable y, por último, codifica secuencias de operadores «· +» y «+ ·» con operando. Este lenguaje especial permite una reducción ostensible de la longitud de los programas a la vez que acelera el cómputo de funciones lógicas.

a) Lenguaje ensamblador (L_2)

1) LL V
2) LLC V ; Llamada de la variable V (eventualmente complementada)
3) IN V
4) INC V ; Intersección lógica de la variable V (ev. complementada) con el resultado previo
5) UN V
6) UNC V ; Unión lógica de la variable V (ev. complementada) con el resultado previo
7) UI V
8) UIC V ; Unión lógica de la variable V (ev. compl.) con el resultado previo, seguida de intersección lógica con el resultado anterior (+ ·)

9) IU V
10) IUC V ; Intersección seguida de Unión ($\cdot +$)
11) := V ; Asignación del resultado a la variable V
12) SAC d ; Salto condicional si el resultado previo es cierto (se manipula como una asignación)

La representación aislada de operadores se realiza haciendo V = 0. De este modo, las instrucciones {IN 0, INC 0, UN 0, UNC 0. UI 0, UIC 0, IU 0, IUC 0} representan *sólo operadores*. Por ejemplo, la instrucción IN 0 representa el operador intersección, el cual se aplica sobre los dos resultados previos. Por otro lado, la instrucción LL 0 es una instrucción de no-operación, NOP. La instrucción LLC 0 es una instrucción que complementa el valor del resultado previo (fruto de una operación lógica o llamada de variable).

EJEMPLO. Sea la programación de $S_1 := F \cdot ((\bar{G} + J) \cdot H + K) \cdot \bar{I}$.
La cadena postfija es $S_1 F \bar{G}\, J + H \cdot K + \cdot \bar{I} \cdot :=$ de donde resulta el programa siguiente:

1) LL F
2) LLC G
3) UN J
4) IN H
(5) UI K
(6) INC I
(7) := S_1

Es decir, sólo se emplean 7 instrucciones (6 con variables + 1 asignación).

El hecho de que, normalmente, un programa tenga tantas instrucciones como apariciones de variables más una (para la asignación), unido a las observaciones presentadas, permite esperar que la minimización del número de instrucciones de un programa, y por tanto de la duración de su ejecución, se obtenga al *minimizar el número de apariciones de variables en la expresión lógica*.

EJERCICIO. Codifíquense las asignaciones siguientes:

1) $S := X\,Y + Y\,Z + Z\,X$ (necesita sólo 6 instrucciones).
2) $Z := (a\,b + c\,d) \cdot (e\,f + g\,h)$ (necesita 10 instrucciones).

b) *Estructura de la* UT *y lenguaje de fases asociado*

La estructura de la UT se define alrededor de una *pila* (figura 8.11). Una pila es una estructura de datos en la que se accede a las informaciones introducidas en orden inverso al utilizado para almacenarlas; es decir, utiliza una política de tipo «*último en entrar-primero en salir*» (*last-in-first-out*, LIFO). La *cima* de la pila contiene el último elemento entrado (tras llamada u operación). La cima de la pila es, en el caso presente, el *registro acumulador*, RA.

La tabla 8.2 presenta una definición formalizada de L_2. Como puede observarse, la pila es especial puesto que, además de a RA, puede accederse a X_1 y X_2.

Nota. La pila (figura 8.11) es en esencia un simple registro de desplazamiento.

	ϕ_1	ϕ_2
LL VAR	**si** VAR $<>0$ **entonces** T $\leftarrow$ ((D[RI]));	RA $\leftarrow$ (T), $X_1 \leftarrow$ (RA), $X_{i+1} \leftarrow (X_i)$; $\forall i = 1, \ldots, n-1$
	si no T $\leftarrow$ (RA);	RA $\leftarrow$ (T);
IN VAR	**si** VAR $<>0$ **entonces** T $\leftarrow$ ((D[RI]));	RA $\leftarrow$ (T) $\wedge$ (RA);
	si no —	RA $\leftarrow (X_1) \wedge$ (RA), $X_i \leftarrow (X_{i+1})$; $\forall i = 1, \ldots, n-1$
UN VAR	**si** VAR $<>0$ **entonces** T $\leftarrow$ ((D[RI]));	RA $\leftarrow$ (T) $\vee$ (RA);
	si no —	RA $\leftarrow (X_1) \vee$ (RA), $X_i \leftarrow (X_{i+1})$; $\forall i = 1, \ldots, n-1$
UI VAR	**si** VAR $<>0$ **entonces** T $\leftarrow$ ((D[RI]));	RA $\leftarrow$ [(T) $\vee$ (RA)] $\wedge (X_i)$,
	si no —	RA $\leftarrow$ [$(X_1) \vee$ (RA)] $\wedge (X_2)$, $X_i \leftarrow (X_{i+2})$; $\forall i = 1, \ldots, n-2$
IU VAR	**si** VAR $<>0$ **entonces** T $\leftarrow$ ((D[RI]));	RA $\leftarrow$ [(T) $\wedge$ (RA)] $\vee (X_i)$, $X_i \leftarrow (X_{i+1})$; $\forall i = 1, \ldots, n-1$
	si no —	RA $\leftarrow$ [$(X_1) \wedge$ (RA)] $\vee (X_2)$, $X_i \leftarrow (X_{i+2})$; $\forall i = 1, \ldots, n-2$
:= VAR	(D[RI]) $\leftarrow$ (RA);	RA $\leftarrow (X_1)$, $X_i \leftarrow (X_{i+1})$; $\forall i = 1, \ldots, n-1$
SAC d	**si** (RA) = 1 **entonces** CP $\leftarrow$ (D[RI]);	—

Tabla 8.2. Descripción de L_2 mediante lenguaje de fases.
(*Nota.* Las instrucciones LLC, INC, UNC, UIC e IUC son idénticas a sus correspondientes, sólo que en ϕ_2 debe considerarse ($\bar{T}$) en vez de (T). De este modo «LLC 0» provoca la complementación del contenido del registro RA.)

c) Comentarios

C.1 La evaluación de funciones lógicas se hace ejecutando todas las instrucciones; es decir, procediendo a un barrido exhaustivo del programa. Ahora bien, es fá-

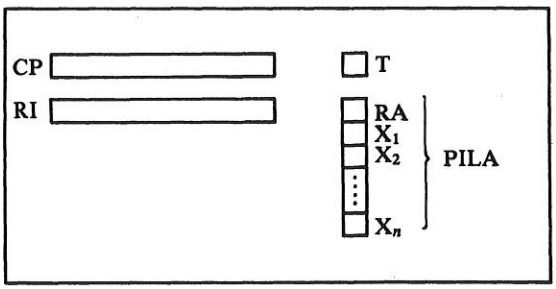


Figura 8.11. Arquitectura de la CPU_2: registros y pila.

cil comprobar que un programa en L_2 es más corto (posee menos instrucciones) y más rápido de ejecutar que un programa en L_1, si se supone que las fases duran el mismo tiempo.

C.2 L_2 permite la evaluación de cualquier expresión lógica mediante una compilación elemental.

C.3 La presencia del salto condicional permite la programación a partir de organigramas. La programación de RdP puede realizarse de forma similar a la ya expuesta en §8.3.3.

EJERCICIO. La consideración de la operación «O-exclusivo» y «O-exclusivo complementado», ¿le podría restar validez a las observaciones que han permitido diseñar L_2?

EJERCICIO. Diséñese una UL para la máquina de pila definida.

8.4.4 Máquina basada en el concepto de operación lógica (M_3)

Ésta es una máquina de registros clásica de una dirección. Sus instrucciones facilitan la evaluación de expresiones lógicas.

a) *Lenguaje* (L_3)

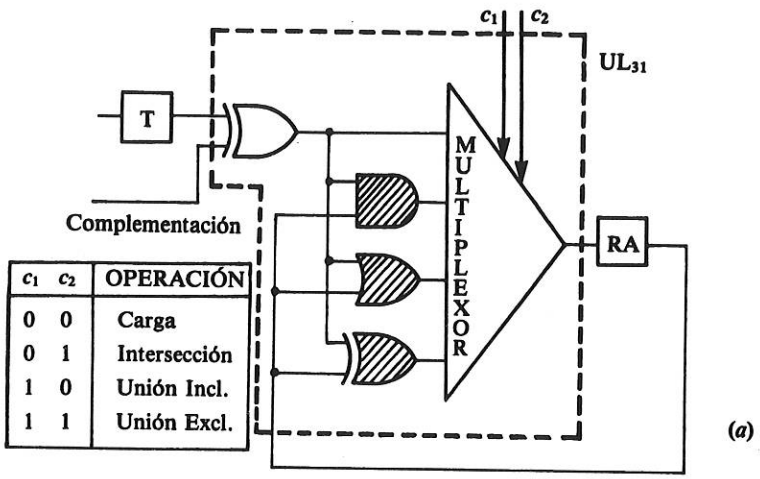
La evaluación de una expresión se programa definiendo una serie de *transferencias* (algunas con cálculo) entre registros y memoria. La estructura más elemental que podemos considerar es aquella que contiene los registros CP, RI, RA y un registro T (transparente al utilizador), además de una unidad lógica (figura 8.4).

El lenguaje puede ser del tipo:

1) CAR V ; Cargar el RA con V: RA $\leftarrow$ (V)
2) CARC V ; Cargar complemento: RA $\leftarrow$ ($\bar{V}$)
3) INT V ; Intersección: RA $\leftarrow$ (RA) $\wedge$ (V)
4) INTC V ; Int. complemento: RA $\leftarrow$ (RA) $\wedge$ ($\bar{V}$)
5) UNI V ; Unión: RA $\leftarrow$ (RA) $\vee$ (V)
6) UNIC V ; Unión complemento: RA $\leftarrow$ (RA) $\vee$ ($\bar{V}$)

7) ALM V ; Almacenar: V ← (RA)
8) ALMC V ; Almacenar complemento: V ← ($\overline{RA}$)
9) OEX V ; O-exclusivo: RA ← (RA) ⊕ (V)
10) OEXC V ; O-exclusivo complemento: RA ← (RA) ⊕ ($\overline{V}$)
11) SAL d ; Salto incondicional: CP ← d

EJERCICIO. Asóciense sendos lenguajes de fases al lenguaje L_3 si se dispone físicamente de UL_{31} y UL_{32} (figura 8.12).



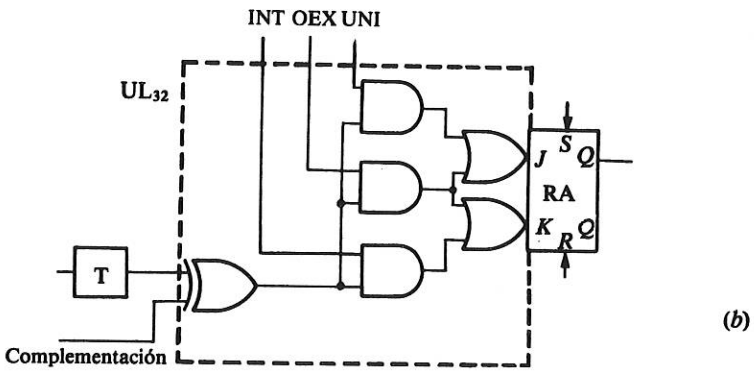


Figura 8.12. Dos posibles esquemas lógicos para UL_3. (*Nota*. Todos los biestables son del tipo D excepto RA, en la figura 8.12*b* que es de tipo J-K.)

b) Comentarios

B.1 A partir de una expresión algebraica, la generación de código, necesita el concurso de un *compilador*, aunque éste sea elemental (véanse obras especializadas en compilación. [GRIE 75], por ejemplo).
La programación de funciones lógicas puede exigir el recurso de *variables intermedias* que tienen que designarse explícitamente. En $\mathbf{M}_2$, las variables intermedias se encuentran en la pila, pero no son designadas explícitamente. La programación de $Z = A\bar{B} + \bar{C}B$ necesita, al menos, una variable intermedia (sea V_1):

1) CAR	A	5) INT	B
2) INTC	B	6) UNI	V_1
3) ALM	V_1	7) ALM	Z
4) CARC	C		

B.2 La definición de las salidas se lleva a cabo utilizando dos instrucciones de asignación (complementada o no).

B.3 El cálculo de una expresión lógica se realiza ejecutando todas las instrucciones que codifican la expresión. En el próximo párrafo veremos como este tipo de programación puede que no resulte interesante. La presencia de saltos condicionales permite el uso de otras técnicas de programación de expresiones lógicas más eficientes.

B.4 Un programa en L_3 nunca será más corto que en L_2, tanto si ambos poseen instrucciones OEX-OEXC, como si carecen de ellas.

8.4.5 Interpretador mixto de operaciones lógicas y saltos condicionales (M_4)

La definición del lenguaje L_4 se basa en la siguiente observación: puesto que una función lógica no posee más que dos resultados posibles «0» y «1», su evaluación puede realizarse comprobando si su valor:

a) No puede ser «0». Por ejemplo, si $F_1 = \underbrace{a\bar{b}c}_{m_1} + \underbrace{\bar{c}db}_{m_2} + \underbrace{ad}_{m_3}$, la condición $m_1 = 1$ implica que $F_1 = 1$, independientemente del valor de m_2 y m_3.

b) No puede ser «1». Por ejemplo, si $F_2 = \underbrace{(x + y + z)}_{M_1}\underbrace{(z + y + t)}_{M_2}\underbrace{(x + t)}_{M_3}$, la condición $M_1 = 0$, implica que $F_2 = 0$, independientemente del valor de M_2 y M_3.

Esta observación elemental permite una evaluación rápida de las expresiones lógicas siempre que se disponga de instrucciones de salto condicional.

a) Lenguaje máquina, estructura y lenguaje de fases de $\mathbf{M}_4$

Sea por ejemplo, el subconjunto de L_3:

1) CAR	V	5) ALM	V
2) CARC	V	6) ALMC	V
3) INT	V	7) SAL	d
4) INTC	V		

al cual le añadiremos las dos instrucciones especiales de salto condicional siguientes:

8) SCU d; Salto Condicional tipo Unión (salto si cierto) a la dirección *d*.

9) SCI d; Salto Condicional tipo Intersección (salto si falso) a la dirección *d*.

De forma intuitiva, diremos que el primero servirá para sustituir al operador « + », mientras que, el segundo lo utilizaremos, normalmente, para sustituir al operador « · », siempre que vaya seguido de «(», « · (».

La estructura de la CPU de esta máquina es idéntica, a nivel de registros, a la de M_3: {CP, RI, RA, T}. La definición de las instrucciones SCU y SCI se expone en la tabla 8.3

	ϕ_1
SCU d	**si** (RA) = 1 **entonces** CP ← (D[RI]) **si no** RA ← 1
SCI d	**si** (RA) = 0 **entonces** CP ← (D[RI])

Tabla 8.3. Descripción (parcial) de L_4 mediante lenguaje de fases.

Para facilitar la comprensión de los comentarios que seguirán, presentamos tres ejemplos de programación con L_4:

EJEMPLO 1. Sea $a \cdot \bar{b} + \bar{c} \cdot b = Z$.

1) CAR *a*
2) INTC *b*
3) SCU 6
4) INTC *c*
5) INT *b*
6) ALM *Z*

EJEMPLO 2. Sea $a \cdot (b + \bar{e}) + \bar{c} \cdot d = Y$.

1) CAR *a*
2) SCI 7
3) INT *b*
4) SCU 9
5) INTC *e*
6) SCU 9
7) INTC *c*
8) INT *d*
9) ALM *Y*

EJEMPLO 3. Sea $a + \bar{b} + \bar{c} + e = F \Leftrightarrow \bar{a}bc\bar{e} = \bar{F}$.

1) CAR	a			1) CARC	a
2) SCU	8			2) INT	b
3) INTC	b	⇒ ambos miembros ⇒		3) INT	c
4) SCU	8	de la expresión		4) INTC	e
5) INTC	c			5) ALMC	F
6) SCU	8				
7) INT	e				
8) ALM	F				

b) Comentarios

B.1 Al igual que con L_2 y L_3, la programación directa de expresiones lógicas en forma algebraica no es posible. Se requiere un proceso de compilación.

B.2 La asignación RA ← 1 (tabla 8.3) realizada al ejecutarse la instrucción «SCU d» cuando (RA) = 0, permite la utilización de las instrucciones INT e INTC inmediatamente después del salto. De este modo, en el ejemplo 1 se programó «INTC *c*» como cuarta instrucción. Si la mencionada asignación no se llevara a cabo, la instrucción anterior habría de ser sustituida por «CARC *c*». De acuerdo con esta observación, interesa resaltar que las instrucciones «CAR» y «CARC» desempeñan, en los programas presentados, un papel más restringido que el de la carga del acumulador; se utilizan como *delimitadores de expresiones*. A estas instrucciones se les asocia, con frecuencia, una definición nemotécnica distinta; por ejemplo, *principio de expresión*, «PEX *v*» y PEXC *v*».

Por último, cabe señalar que las mencionadas instrucciones «PEX *v*» o «CAR *v*» podrían ser eliminadas si añadiéramos a M_4 un *bit de asignación*, BA. De esta forma, la programación de expresiones se simplificaría al máximo, puesto que habría un único tipo de instrucción de llamada de variables. Esta opción ha sido adoptada en el sistema COLERES [DACL 76] en el que las instrucciones se denominan *test de variable*, «TV *v*» y «TVC *v*».

B.3 Las instrucciones SCU y SCI permiten saltar todas aquellas instrucciones cuya ejecución no puede cambiar el resultado de la expresión lógica en proceso de evaluación. Hablaremos de *salto explícito* en la evaluación de expresiones lógicas. Merced a esos saltos la evaluación puede ser muy rápida.

B.4 La programación de expresiones lógicas mediante L_4 *no necesita la definición de variables intermedias*. La comparación de los programas que codifican la expresión $Z = ab + cb$ con L_3 y L_4 es suficientemente ilustrativa. La ausencia de necesidad de variables intermedias en los programas codificados en L_4 puede comprenderse fácilmente. En efecto, las variables intermedias se utilizan para memorizar el resultado de una subexpresión; esta memorización no es necesaria si utilizamos L_4, en cuyo caso todo programa se reduce a un encadenamiento de intersecciones de variables y saltos condicionales.

Nota. En [MART 80] se presentan, informalmente, algoritmos para la generación de código en un lenguaje próximo a L_4 a partir de una expresión algebraica.

8.4.6 Comentarios generales

Hemos considerado en este apartado un grupo de cuatro tipos de AP especialmente diseñados para la interpretación de diagramas y expresiones lógicas. En el próximo capítulo analizaremos la simulación de RdP en máquinas de propósito general.

La tabla 8.4 resume gran parte de los comentarios realizados al analizar los diferentes sistemas. En ella se reflejan algunos de los aspectos más característicos encontrados. Mención aparte merece la complejidad de los traductores del lenguaje de programación a lenguaje máquina (ejecutable).

	M_1(§8.4.2, Diagramas Lógicos)	M_2(§8.4.3, Máquina de Pila)	M_3(§8.4.4, Máquina de Registros)	M_4(§8.4.5, Operaciones y Saltos)	M_5(§7.4.3, Decisiones Binarias)
Dispositivo de Cálculo	Unidad Lógica (UL)	UL	UL	UL (sólo intersecciones)	No posee
Número de Acumuladores	2 { RAY, RAO	1 (cima de la pila)	1	1	No posee
Necesidad de variables intermedias	SI	SI (implícitas en la pila)	SI	NO	NO
Utilización de saltos en la evaluación	NO	NO	NO	SI	SI
Bit de Asignación	SI	NO	NO	NO	NO
Instrucciones de asignación	* Almacenar resultado (Y, 0) * Activación/ /desactivación condicional	: =	ALM ALMC	ALM ALMC	Activación/ /desactivación incondicional

Tabla 8.4 Resumen de peculiaridades de los AP considerados.

El lenguaje L_1 (Diagramas Lógicos) es ejecutable casi directamente siempre que las variables se designen por sus direcciones y no por sus nombres simbólicos. Es decir, sólo necesita un *ensamblador* elemental. Los programas en L_2, L_3 y L_4 se obtendrán, a partir de una expresión algebraica, mediante *compilación*, cuya complejidad crece progresivamente.

En lo que concierne al tratamiento de expresiones lógicas, hemos presentado los principales métodos que existen. No obstante, cabe mencionar aún la evaluación de expresiones lógicas por aplicación de *máscaras a vectores de variables*. Así, por ejemplo, si se desea calcular $\varphi = A\bar{B}$ y en una palabra se posee la siguiente disposición de variables:

$$V = \boxed{A}\boxed{B}\boxed{C}\boxed{\ldots}\boxed{H}$$

es fácil observar que su intersección con la *máscara de selección*:

$$m_s = \boxed{1}\boxed{1}\boxed{0}\boxed{\ldots}\boxed{0}$$

seguida de un O-exclusivo con la *máscara de cálculo*:

$$m_c = \boxed{1}\boxed{0}\boxed{0}\boxed{\ldots}\boxed{0}$$

es nula si y sólo si $A = 1$ y $B = 0$ (de donde se desprende que $\varphi = 1$); es decir, $(V \cdot m_s) \oplus m_c = 0 \Leftrightarrow \varphi = 1$.

EJERCICIO. Determínense las operaciones que se debe realizar, así como las máscaras necesarias, para calcular las funciones:

1) $A + B$
2) $A + BC$

Finalmente, dentro de los comentarios generales sobre los AP presentados, interesa resaltar que si se les supone una realización material específica, las operaciones correspondientes a la fase $\varnothing_2$ de una instrucción podrán siempre realizarse en paralelo con la fase $\varnothing_0$ de la instrucción siguiente. En efecto, baste observar que durante $\varnothing_2$ las operaciones son siempre internas a la unidad de tratamiento.

8.5 CONCLUSIÓN

Los autómatas programables constituyen un concepto de computador para el control lógico de procesos. Una filosofía de equipo. Los microprocesadores son recursos tecnológicos cada vez más empleados en la realización de autómatas programables.

Básicamente los AP aparecen como máquinas que simulan en secuencia el comportamiento de un conjunto de dispositivos lógicos cableados. Es decir, se sustituye una realización *espacial* (cálculo paralelo con operadores aislados) por una *secuencial* (un procesador único con un conjunto restringido de operadores trabaja simulando, uno tras otro, el comportamiento de los diferentes dispositivos lógicos). Este modo de proceder tiene diversas ventajas (flexibilidad, coste, etc...), pero plantea dos problemas :

1) La aparición de funcionamientos no previstos, debidos a la simulación secuencial (en realidad, el conjunto de circuitos que se desea simular opera en paralelo). Las aleatoriedades de programación.
2) El tiempo de respuesta del AP será, normalmente, muy superior al del sistema cableado (si ambos se realizan con la misma tecnología), lo que, para determinados procesos muy rápidos, puede ser una limitación importante.

La tendencia actual es hacia la definición de AP que manipulen directamente descripciones funcionales (basadas en grafos de estado, redes de Petri, etc...) con lo que además de ganar en seguridad de la descripción, se pueden evitar sistemáticamente las aleatoriedades de programación. El capítulo 9, se ocupa de ello.

Funcionalmente los AP se caracterizan por ofrecer un lenguaje de programación simple y adaptado a la clase de problemas, con pocas instrucciones, direccionamiento directo (casi exclusivamente) y la no existencia de subprogramas ni de interrupciones manipulables por el programador. *Operacionalmente* los AP son sistemas muy evolutivos físicamente (modularidad en E/S, modularidad en memoria, procesadores etc...), disponen de periféricos que pueden ser muy sofisticados para la edición y puesta a punto de programas (maletas de programación) y ofrecen una elevada seguridad de funcionamiento. En efecto, la mantenibilidad se ve mejorada gracias a la existencia de programas de diagnóstico y localización de averías junto con la modularidad física; la credibilidad se aumenta gracias a la existencia de «perros guardianes», dispositivos de detección de errores en la UT, etc.,... *Tecnológicamente* en los AP se cuidan aspectos tan importantes como la inmunidad ante parásitos o perturbaciones en la alimentación y se ofrecen equipos adaptados a ambientes industriales de trabajo (admiten humedades relativas del 90 a 95 por 100, la presencia de polvo, etc...).

En lo que respecta a la evolución tecnológica, la aparición de los μ-procesadores (4-8 bits) ha provocado la aparición de los dos fenómenos fundamentales en los AP:

1) La evolución hacia los sistemas de control y regulación general, comprendiendo cada vez más funciones complementarias de regulación, archivado, etc.
2) La evolución hacia estructuras de control descentralizadas, gracias a la «capacidad de diálogo» de los μ-procesadores.

Por último, es importante resaltar que la aparición de μ-procesadores de un bit (μ-AP) evidencia el interés industrial de esta clase de sistemas, si bien parece ser que su éxito comercial no se ha consolidado.

EJERCICIOS

8.1 Prográmese en L_2 con seis instrucciones, la expresión lógica: $(A + B) \cdot (CD + EF)$. ¿Se puede programar con menos instrucciones? ¿Por qué?

8.2 Concíbase un AP elemental, con las instrucciones siguientes:

1) LL V
2) LLC V
3) +
4) ·
5) :=

8.3 Prográmese la expresión $S: = A\bar{B}(C + \bar{D}) + F\bar{I}(K + H(\bar{G} + \bar{J})$, en los lenguajes L_2, L_3 y L_4. Compárense los diferentes programas.

9

Realización programada (II): microcomputadores y autómatas programables especiales

9.1 INTRODUCCIÓN

Los computadores presentados en el capítulo anterior, autómatas programables de uso general, están especializados en el tratamiento de expresiones lógicas, con frecuencia definidas indirectamente merced a esquemas de relés o diagramas lógicos. La simulación de RdP se puede llevar a cabo con aquellos AP ejecutando las expresiones lógicas que definen:

1) el disparo de cada transición y las acciones que éstos generan.
2) las acciones asociadas a cada lugar.

En este capítulo se proponen esquemas mejor adaptados para la simulación de RdP binarias. Esencialmente se basan en la utilización de microcomputadores de propósito general y persiguen una simulación más *eficiente* (más rápida y, eventualmente, con menor ocupación de memoria) y *sin aleatoriedades*. Estos requisitos se consiguen gracias a una cierta sofisticación en los algoritmos de simulación, a pesar de lo cual seguirán siendo bastante simples.

Para presentar los métodos principales de realización, puede optarse, entre otros, por los dos enfoques siguientes:

1) Partir de la representación matricial de una RdP, analizar sus inconvenientes e introducir representaciones alternativas que, normalmente, permitan mejores prestaciones.
2) Partir de las técnicas clásicas de realización de grafos de estados reducidos, GR, y generalizarlas hasta obtener esquemas válidos para simular RdP.

La diversidad de técnicas que se pueden emplear para la simulación y el elevado número de posibilidades existentes para codificar informaciones relativas a las RdP (estructura, lugares marcados, transiciones sensibilizadas, etc.) nos han inclinado a utilizar el primero de los dos enfoques anteriormente enunciados. Al proceder de este modo, la unidad conceptual entre los diversos métodos de realización se hace más patente. Por otro lado, la extensión directa de los métodos clásicos de realización de GR no permite obtener de forma «natural» las diferentes clases de métodos de simulación posibles. La consideración de un «panorama» relativamente general

de métodos de simulación tiene su interés dado que, según la clase de RdP, se pueden definir métodos específicos con los que obtener las mejores prestaciones. Dicho de otro modo, *no* existe un método único, óptimo para simular cualquier RdP.

Este último capítulo comienza considerando la especificación de RdP mediante lenguajes textuales (§9.2) y exponiendo una serie de cuestiones previas a la realización (§9.3). Posteriormente (§9.4, §9.5, §9.6 y §9.7), de acuerdo con el enfoque adoptado, se aborda la simulación cuando la estructura y el marcado de la RdP se representan mediante unas *tablas* (estructuras de datos) que son accedidas por un algoritmo *interpretador* o *de simulación*. Así, en §9.4 se considera una eficiente representación *matricial*, mientras que en §9.5 se introduce una representación derivada de la anterior pero que utiliza *listas*, con lo que, normalmente, se obtienen codificaciones más compactas. En §9.6 y §9.7 se modifica el esquema de representación de §9.5 para obtener simulaciones más rápidas. Esencialmente, la idea será la de restringir el cálculo de las condiciones de sensibilización a un subconjunto de las transiciones de la red. En §9.8 se define un AP cuya programación se realizará gracias a la existencia de un *lenguaje algorítmico de bajo nivel* (tipo ensamblador) especial.

Los diferentes algoritmos se presentan con una notación relativamente informal que posibilita una rápida traducción a lenguajes ensambladores de microcomputadores de propósito general. A pesar de su apariencia «estructurada», los algoritmos no suelen estar estructurados porque se ha buscado la máxima *eficiencia* en los programas, cortos pero de una elevadísima frecuencia de ejecución. Las tablas 9.4, 9.5 y 9.6 presentan codificaciones de algunos de los esquemas básicos de simulación en el lenguaje ensamblador del microprocesador MOTOROLA 6801.

9.2 LENGUAJES DE ESPECIFICACION DE RdP

9.2.1 Lenguajes

Los lenguajes que permiten la definición de RdP pueden clasificarse, en una primera aproximación, en dos grandes grupos. Estos son:

1) lenguajes *algorítmicos*, también denominados *procedimentales* o *dinámicos*;
2) lenguajes *no-algorítmicos*, también denominados *no-procedimentales* o *estáticos*.

Los lenguajes del segundo grupo se dice que son de *especificación*, no de programación. Desde un punto de vista conceptual, los del primer grupo provienen de modificaciones, más o menos importantes, de lenguajes similares a los presentados en el capítulo anterior. Su innovación suele residir en la introducción de instrucciones primitivas capaces de manipular con mayor o menor comodidad o/y eficiencia, la descripción y la simulación de las RdP. En §9.8 se presenta un ejemplo de este tipo de lenguajes.

En el presente contexto, los lenguajes del segundo grupo son bastante más interesantes. Se dice que un lenguaje es no-algorítmico si *no contiene instrucciones*, entendidas éstas como órdenes que controlen explícitamente el comportamiento dinámico de un computador real o virtual (simulado). Un lenguaje no-algorítmico no asocia significación alguna al orden en el que se presentan las frases que describen el com-

portamiento que se desea realizar†. Cada *frase* es un conjunto de instrucciones que lleva asociado una etiqueta mediante la que se define(n) la(s) condición(es) que debe(n) cumplirse para que se lleve a cabo su ejecución.

Los lenguajes de especificación que se pueden definir varían entre los del tipo de *cumplimentación de estadillos* (*fill-in-the-blank*) y otros de «mayor nivel». En todos ellos se puede adoptar una definición conjunta de la estructura e interpretación asociada a la RdP, o proceder a su definición por separado.

A continuación, se considerará un ejemplo de lenguaje que define separadamente la estructura e interpretación asociada a la RdP. En éste, la estructura y el marcado inicial se especifican con un sublenguaje del tipo cumplimentación de estadillos, mientras que los eventos se expresan en un sublenguaje que acepta la notación algebraica.

La tabla 9.1*a* especifica la RdP de la figura 9.1. Como puede observarse fácilmente, se trata de una simple transcripción textual de la mencionada figura. La estructura de la RdP se especifica transición por transición. Cada *transición* se define mediante:

1) sus lugares de entrada y de salida y
2) el evento asociado (si existe).

La *interpretación* se especifica definiendo:

1) los eventos y
2) las acciones asociadas a cada lugar.

El *marcado inicial* se define después de la especificación de la estructura de la red. En este punto caben dos observaciones de interés:

1) Si por razones de seguridad, se juzgara importante la redundancia en la especificación de la estructura de la RdP, ésta se puede redefinir completamente especificando para cada lugar sus transiciones de entrada (*función de incidencia posterior*) y las de salida (*función de incidencia previa*). La tabla 9.1*b* reespecifica la estructura de la red de la figura 9.1.
2) Si existiesen acciones asociadas al disparo de las transiciones, éstas podrían especificarse con los eventos o en un apartado independiente (acciones impulsionales).

Algunas otras facilidades para especificar RdP pueden consistir en la capacidad de definición y utilización de *macrolugares*, *macrotransiciones*, *subRdP*, ... De este modo, la especificación de grandes sistemas podría simplificarse, utilizando directamente algunos de los principios metodológicos de descripción esbozados en el capítulo 5 (§5.5).

Las principales ventajas de la clase de lenguajes sugerida en este apartado frente a los lenguajes algorítmicos son:

1) La especificación directa de la RdP, lo que facilita la tarea de programación hasta el punto de no requerirse conocimientos informáticos para su desarrollo.

---

† Recuérdese que en los lenguajes algorítmicos (la inmensa mayoría), la ordenación de las instrucciones es fundamental: permite definir las secuencias de operación.

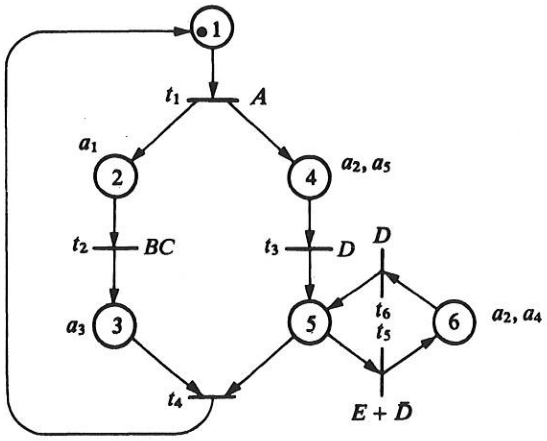


Figura 9.1. Red de Petri binaria.

ESTRUCTURA
$t_1 : p_1 | p_2, p_4 \$ e_1;$
$t_2 : p_2 | p_3 \$ e_2;$
$t_3 : p_4 | p_5 \$ e_3;$
$t_4 : p_3, p_5 | p_1;$
$t_5 : p_5 | p_6 \$ e_5;$
$t_6 : p_6 | p_5 \$ e_3$ #
MARCADO INICIAL p_1 #
EVENTOS
$e_1 := A; e_2 := BC;$
$e_3 := D; e_5 := E + \bar{D};$
ACCIONES
$p_2 : a_1;$
$p_3 : a_3;$
$p_4 : a_2, a_5;$
$p_6 : a_2, a_4$ #

(*a*)

ESTRUCTURA
$p_1 : t_4 | t_1;$
$p_2 : t_1 | t_2;$
$p_3 : t_2 | t_4;$
$p_4 : t_1 | t_3;$
$p_5 : t_3, t_6 | t_4, t_5;$
$p_6 : t_5 | t_6$ #

(*b*)

Notas:
1) «#» es un separador entre las diferentes partes de una especificación.
2) «$», «|» y «;» son otros tantos separadores.

Tabla 9.1 Especificación de la RdP de la figura 9.1

2) La eliminación de aleatoriedades de programación, como las aparecidas en el capítulo anterior, no tiene que ser resuelta por el programador.
3) La estructura y el marcado de la RdP pueden ser validados directamente.
4) La especificación redundante de la estructura de la RdP permite reducir considerablemente los errores en el proceso de entrada de datos.

9.2.2 Representación interna de la RdP y traducción

En el apartado anterior se ha considerado la especificación de RdP desde el punto de vista del operador/programador. Ahora bien, no resulta difícil comprender que por diversas razones (entre las que se puede encontrar la mejora de prestaciones) puede suceder que la anterior forma de especificación no sea adecuada para que la máquina lleve a cabo directamente la simulación. Dicho de otro modo, si la forma en que el programador especifica el sistema que se desea realizar difiere de la forma en que el computador debe considerarla, tiene que insertarse un *proceso de traducción*. Obsérvese que los procesos de traducción a que se hace referencia son absolutamente clásicos en informática. De este modo, un programa en lenguaje PASCAL (o FORTRAN, o COBOL, etc.) no se suele ejecutar directamente, sino después de una cierta transformación. Por tratarse de transformaciones entre dos lenguajes (formas de representación de ideas), éstas se denominan genéricamente traducciones.

En nuestro proceso de traducción, el *programa fuente* es el que define la RdP. El producto de la traducción puede variar entre dos formas radicalmente distintas, según el modo de ejecución elegido para la simulación. Así, ésta puede realizarse gracias a un programa *interpretador* que toma de unas *tablas* (estructuras de datos) la información que codifica la RdP; es decir, no existe de forma explícita el concepto de instrucción con su código de operación. Este tipo de técnica de simulación se denomina *conducida por tabla* (*table driven*) y permite codificar las RdP con una ocupación de memoria mínima, pero a costa de una simulación que puede ser relativamente lenta.

La segunda forma básica que puede tomar el producto de la traducción es una (varias) secuencia(s) de instrucciones. Es decir, se produce *un programa*. La simulación se dirá *conducida por programa*. Si éste es ejecutable directamente†, el proceso de traducción se denomina *compilación*. Cuando el proceso de traducción genere un programa cuyas instrucciones no pertenezcan al computador (sean instrucciones de un computador virtual), deberá existir otro programa, también denominado *interpretador*, que analice y ordene la ejecución del programa intermedio obtenido. Tanto en este último caso como en el de la simulación conducida por tabla, se dice que la realización es *interpretada por programa.*

En §9.4, §9.5, §9.6 y §9.7 se considerarán realizaciones *mixtas* en las que la simulación global estará dirigida por *tablas*, pero la evaluación de eventos se realizará gracias a *programas* similares a los presentados en §8.3. En §9.8 se considerará una realización en la que el proceso de traducción debe generar un programa ejecutable sobre una máquina (virtual si el soporte es un microcomputador estándar).

Para concluir estos comentarios, baste con señalar que el proceso de traducción desde los lenguajes de especificación como el considerado en §9.2.1 hacia la representación interna utilizada por el computador de simulación, suele ser razonablemente simple en términos generales. Dicho de otro modo, los traductores considerados no suelen ser programas ni muy voluminosos ni muy complejos.

---

† Es decir, si las instrucciones generadas pertenecen al repertorio del computador sobre el que se llevará a cabo la simulación.

9.3 CUESTIONES PREVIAS A LA REALIZACIÓN DE SISTEMAS ESPECIALIZADOS EN LA SIMULACIÓN DE RdP BINARIAS

A la hora de proceder a la definición de sistemas especializados en la simulación de RdP binarias, cabe plantear un cierto número de cuestiones previas de tipo general. En este apartado se pretende dar una breve perspectiva de algunas de éstas, a la vez que se establecen determinadas *decisiones* sobre los sistemas que se presentarán posteriormente.

9.3.1 Computador soporte

Entre los diferentes AP especializados en la simulación de RdP binarias que se han propuesto en la literatura técnica, existen dos grandes grupos:

1) Los sistemas basados en un *hardware específicamente diseñado*. Entre éstos los hay que emplean secuenciadores microprogramables (por ejemplo, COLERES [DACL 76] está diseñado alrededor de la serie 3000 de INTEL) o microprocesadores especiales más un cierto hardware adicional (por ejemplo, en [DEFE 79] se utiliza el microautómata MOTOROLA 14500B). Normalmente, estos sistemas son muy rápidos en la simulación.
2) Los sistemas basados en *microcomputadores de propósito general* (MOTOROLA 6802, INTEL 8085,...), cuya especialización se realiza por *software*. Evidentemente los sistemas de este grupo son más lentos que los del anterior.

En los apartados que siguen se considerará esencialmente la utilizacion de esquemas basados en microcomputadores de propósito general. La transposición de funciones software al hardware permitirá la definición de sistemas especializados más rápidos. Así, por ejemplo, en [SILV 80*a*] se presenta un sistema que, además del microcomputador estándar, utiliza un *explorador de lugares marcados*, procesador elemental especializado.

9.3.2 Definición de las entradas y salidas†

En su forma más simple, las entradas y salidas se pueden adquirir o emitir directamente, en el momento en que se necesite la variable de entrada o se haya elaborado la variable de salida. Ahora bien, si se desean evitar las aleatoriedades que puedan provenir de una alteración de los valores de las variables de entrada durante un ciclo de tratamiento, éstas se han de adquirir simultáneamente y memorizarlas en un proceso que se denomina *muestreo*.

La anterior memorización puede llevarse a cabo en una *memoria imagen de las entradas*, que será actualizada por el microcomputador al comenzar cada ciclo de tratamiento. Otra forma de proceder, algo más costosa y eficiente, pero de amplia utilización, consiste en cargar directamente los valores correspondientes en registros dispuestos en las interfases de entrada. En este último caso, el conjunto de registros

† Los conceptos que aquí se consideran fueron presentados en el capítulo 8. La breve mención que se establece en este apartado persigue facilitar la lectura independiente de este capítulo.

de entrada forma la memoria imagen de las entradas, y su actualización se realiza simultáneamente en un único ciclo de reloj.

La emisión de las salidas se suele hacer directamente, a medida que se calculan. Si se desea una emisión «cuasi-simultánea» de todas ellas, se utilizará una *memoria imagen de las salidas*, en la que éstas se van almacenando a medida que son elaboradas. Al finalizar el ciclo de tratamiento, el microprocesador copiará esta imagen sobre los registros de las interfases de salida. La utilización de una memoria imagen de las salidas permite, además de la emisión cuasi-simultánea, una mayor seguridad ante posibles parásitos que hayan alterado puntualmente el contenido de registros de la interfase de salida. (En efecto, si ha habido alteración, al recopiar la imagen de la variable de salida, se restaurará el valor correcto.)

En algunos casos, para simplificar la programación, la memoria imagen de las salidas es *borrada* (puesta a cero) al comienzo de cada ciclo de tratamiento. MIB significará *Memoria Imagen Borrada* al comenzar un ciclo de tratamiento. En este caso, en cada ciclo de tratamiento debe calcularse completamente el nuevo vector de salidas.

Normalmente se supondrá que las salidas se emiten directamente. Para presentar un esquema radicalmente distinto, en §9.8 se utilizará una MIB. Dado que la utilización de una memoria imagen de las entradas permite la eliminación de aleatoriedades debidas a los cambios de valor de las entradas durante un ciclo de tratamiento, en lo sucesivo se supondrá que siempre existe ésta.

9.3.3 Simulación síncrona/no síncrona

En lo sucesivo, se dirá que la simulación de una RdP es *síncrona* si el marcado evoluciona globalmente en un momento determinado. (Normalmente al finalizar el cálculo del nuevo marcado.)

Obsérvese que la definición de simulación síncrona adoptada no exige la presencia de un reloj tiempo real que lance los ciclos de tratamiento con una cadencia fija, ni que las salidas sean emitidas cuasi-simultáneamente.

La definición de evolución síncrona sólo pretende evitar las aleatoriedades que puedan surgir de una modificación *en serie* de los componentes del marcado, cuya evolución real debería ser *paralela*. De este modo, una simulación no-síncrona de la RdP de la figura 6.3*a* dará un marcado intermedio no binario siempre que la transición superior se trate antes que la otra de idéntica etiqueta. De lo anterior se desprende que si, por ejemplo, el marcado se representa con un vector booleano, la simulación será incorrecta puesto que transitoriamente este llegará a tener dos marcas en p_1. Por otro lado, es interesante observar que con evoluciones no-síncronas, el marcado final que se obtendría al terminar un ciclo de tratamiento depende del orden en que se consideren las transiciones.

La interpretación síncrona es siempre más costosa en memoria puesto que se ha de disponer del marcado anterior (el existente al comienzo del ciclo de tratamiento) y de información necesaria para construir el nuevo marcado. Evidentemente, la simulación síncrona será también algo más lenta. En evitación de aleatoriedades, en lo sucesivo se adoptará siempre una simulación síncrona.

9.3.4 Sobre la interpretación asociada a las RdP que se simulan

La definición de *eventos* puede variar desde la consideración de una única variable (complementada o no) hasta la definición de expresiones algebraicas, incluyendo flancos. En lo sucesivo se supondrán definidos los eventos mediante expresiones lógicas de las variables de entrada.

La definición de las *acciones* asociadas a los lugares, puede ser incondicional o condicionada por una determinada función lógica. La consideración exclusiva de acciones incondicionales asociadas a los lugares, permite simplificar la simulación puesto que, en este caso, se pueden representar todas las acciones como asociadas al disparo de transiciones (figura 9.2).

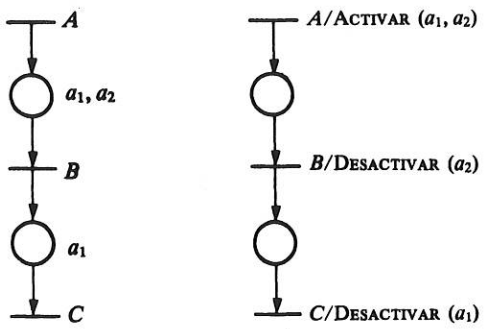


Figura 9.2. Las acciones incondicionales asociadas a los lugares pueden representarse como asociadas al disparo de transiciones.

En §9.4, §9.5, §9.6 y §9.7 se considerarán acciones asociadas a las transiciones y acciones incondicionales asociadas a los lugares. En §9.8 se supondrá que existen acciones condicionales asociadas a los lugares. En este último caso, la simulación realizará, normalmente, dos tratamientos bien diferenciados†:

1) evolución del marcado y generación de acciones asociadas a las transiciones;
2) generación de las acciones asociadas a los lugares.

9.4 MÉTODO MATRICIAL BÁSICO PARA LA SIMULACIÓN DE RdP BINARIAS

En este apartado se presenta un algoritmo de simulación basado en una representación matricial de la estructura de una RdP. Considerando el problema en toda su

† En el capítulo 8 estos dos tratamientos corresponden a dos fases bien diferenciadas de la simulación. El método de simulación que se presentará en §9.8 procede en una única fase, utilizando un esquema conceptualmente algo más complicado, pero eficiente.

generalidad, la evolución del marcado puede obtenerse fácilmente, mediante calculos *aritméticos* que utilicen las matrices de incidencia previa y posterior (§2.2.1), aunque si la RdP es, pura bastará con la ecuación de estado (§2.2.2):

$$M_k = M_{k-1} + \tilde{C} \cdot U.$$

Restringiendo la simulación a RdP binarias, a continuación se transformarán los anteriores cálculos aritméticos en cálculos *lógicos*, mucho más rápidos de ejecutar y conducentes a una menor ocupación de memoria. En el presente caso se ha optado por una simulación *síncrona*, *dirigida por tabla* (matrices), en la que *no existen acciones condicionales* asociadas a los lugares, pero en la que si se dispone de una *memoria imagen de las entradas.*

9.4.1 Fundamentos

Sean:

1) $E_{n\times m}$ $(= C^-_{n\times m})$ la matriz de incidencia previa de una RdP ordinaria. (Dado que la red es ordinaria, E será una matriz booleana.)
2) $\tilde{C}_{n\times m}$ la matriz de flujo de marcas asociada a la RdP binaria y $F_{n\times m}$ la matriz booleana, en la que $F_{ij} = 1$ si y sólo si p_i es lugar de entrada o de salida de t_j pero no simultáneamente lugar de entrada y de salida. Es decir, $F_{ij} = 1$ sii $\tilde{C}_{ij} \neq 0$. (Expresado de otra forma: $F = C^- \oplus C^+$.)
3) $M_{n\times 1}$ el vector booleano que representa el marcado de una RdP binaria.
4) A^j la j-ésima columna de la matriz A.

De acuerdo con la notación anterior, es fácil observar que:

a) t_j está sensibilizada por M sii $E^j \leqslant M$. Es decir: $\forall_i$, $E_{ij} \leqslant M(p_i)$.
b) el disparo de t_j permite obtener el marcado M^+ definido por la suma aritmética de M y la j-ésima columna de $\tilde{C}$: $M^+ = M + \tilde{C}^j$.

Ahora bien: $[E^j \leqslant M] \Leftrightarrow [E^j \wedge \bar{M} \leqslant M \wedge \bar{M}] \Leftrightarrow [E^j \wedge \bar{M} = 0]$. Por consiguiente, t_j estará sensibilizada por M si y sólo si es nula la *intersección lógica* de los vectores booleanos E^j y $\bar{M}$, $E^j \wedge \bar{M} = 0$, condición vectorial fácil de comprobar. Por otro la-

$$E = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array} \qquad F = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 & t_4 & t_5 & t_6 \end{matrix} \\ \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array} \qquad M = \begin{array}{c} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \\ \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{matrix} \end{array}$$

Tabla 9.2 Matrices E, F y M correspondientes a la RdP de la figura 9.1 (y tabla 9.1).

do, el nuevo marcado, M^+, se obtiene, en general, mediante una suma aritmética de los vectores M y $\tilde{C}^j$: $M^+ := M + \tilde{C}^j$. Habida cuenta que M y M^+ son vectores booleanos, la anterior suma puede considerarse módulo 2 si se reemplaza $\tilde{C}$ por F (matriz booleana que representa a $\tilde{C}$ módulo 2). En resumen, $M^+ := M + \tilde{C}^j$ puede reescribirse como $M^+ := M \oplus F^j$, donde $\oplus$ simboliza la *suma módulo* 2 u o-exclusivo (operación lógica). Puesto que la sensibilización de t_j se ha realizado utilizando $\bar{M}$, convendrá complementar la ecuación vectorial-booleana que permite calcular el nuevo marcado, $\bar{M}^+ := \bar{M} \oplus F^j$, de donde bastará operar con $\bar{M}$.

El algoritmo de simulación resultante es elemental (tabla 9.3). Este trabaja sobre las matrices E y F (que definen la estructura de la RdP) y el vector complemento del marcado, $\bar{M}$. Los eventos y acciones asociados a cada transición se codifican en procedimientos independientes (procedimientos evento-acción, *pevac*).

SIMULADOR (INTERPRETADOR)	Eventos y Acciones asociados a la RdP de la figura 9.1
(1) Adquisición de las entradas (muestreo de éstas); (2) **para** $i := 1$ **hasta** m **hacer** **si** $E^i \wedge \bar{M} = 0$ **entonces** **caso** i **de** 1: *pevac* 1; ⋮ ⋮ i: *pevac* i; ⋮ ⋮ m: *pevac* m; **fcaso** **si** $\mathcal{E}$ **entonces** $\bar{M}^+ := \bar{M}^+ \oplus F^i$; **fsi** **fsi** **fpara**; (3) $\bar{M} := \bar{M}^+$; (4) **ir a** 1;	**procedimiento** *pevac* 1 $\mathcal{E} := A$; $\{\mathcal{E} := e_1\}$ **si** ϵ **entonces** $a_1 := 1, a_2 := 1, a_3 := 1$ **fsi** **fproc** **procedimiento** *pevac* 2 $\mathcal{E} := BC$; $\{\mathcal{E} := e_2\}$ **si** $\mathcal{E}$ **entonces** $a_1 := 0, a_3 := 1$ **fsi** **fproc** **procedimiento** *pevac* 3 $\mathcal{E} := D$; $\{\mathcal{E} := e_3\}$ **si** $\mathcal{E}$ **entonces** $a_2 := 0; a_5 := 0$ **fsi** **fproc** **procedimiento** *pevac* 4 $a_3 := 0$; $\mathcal{E} := 1$; **fproc** **procedimiento** *pevac* 5 $\mathcal{E} := E + \bar{D}$; $\{\mathcal{E} := e_5\}$ **si** $\mathcal{E}$ **entonces** $a_2 := 1, a_4 := 1$ **fsi** **fproc** **procedimiento** *pevac* 6 $\mathcal{E} := D$; $\{\mathcal{E} := e_3\}$ **si** $\mathcal{E}$ **entonces** $a_2 := 0; a_4 := 0$ **fsi** **fproc**

Tabla 9.3 Algoritmo básico de simulación matricial.

Observaciones:

1) El algoritmo de simulación presentado en la tabla 9.3, pretende, esencialmente, una fácil comprensión de los fundamentos. Para obtener una realización más eficiente con microcomputador pueden introducirse diversas modificaciones. En §9.4.3 se consideran

algunas de éstas y se presenta un simulador escrito en el lenguaje ensamblador del microprocesador MOTOROLA 6801.

2) Si existiesen acciones condicionales asociadas a los lugares, bastaría con insertar las correspondientes ecuaciones entre los puntos (3) y (4) del algoritmo de simulación (tabla 9.3).
3) Si no se exigiese una simulación síncrona, bastaría con hacer $M = M^+$ y, por consiguiente, eliminar el punto (3).
4) Aunque sea evidente, interesa resaltar que el algoritmo presentado procede calculando la condición de disparo de todas las transiciones de la red y que la simulación es síncrona. Por lo tanto, si la red posee un conflicto efectivo,se «dispararán» todas las transiciones del mismo, de donde el marcado final será *incorrecto*. Expresando la idea de otro modo, lo que se pretende resaltar es que *la simulación síncrona será correcta sólo si la RdP no presenta conflictos efectivos.* Si la simulación no fuese síncrona ($M = M^+$), al disparar una transición, sus lugares de entrada se desmarcarían, de donde se desensibilizaría el resto de las transiciones en conflicto. Es decir, la simulación *asíncrona* resuelve todo conflicto dándole prioridad al disparo de la transición que el simulador considere en primer lugar.

9.4.2 Prestaciones

El método matricial expuesto en el apartado anterior basa su eficiencia en la compacidad de la representación booleana de la estructura (matrices $\boldsymbol{E}$ y $\boldsymbol{F}$) y el marcado (vectores M y M^+) de la RdP binaria. En efecto, si suponemos, por ejemplo, que el simulador se construye utilizando un microcomputador de 8 bits, cada matriz ocupará $\lceil n/8 \rceil \cdot m$ octectos (es decir, $\lceil n/8 \rceil$ octetos por columna). Por otro lado, puesto que a cada transición se le asocia un par evento-acción, se necesitará un puntero que señale el emplazamiento del correspondiente procedimiento (cada puntero ocupará 2 octetos). Asimismo, los vectores M y M^+ ocuparán $\lceil n/8 \rceil$ octectos de memoria viva cada uno. De lo anterior se deduce que dada una RdP con $n = 16$ y $m = 16$ (normalmente $n \cong m$), la representación de la estructura y el marcado necesitará $(2 + 2) \cdot 16 + 2 \cdot 16 + 2 \cdot 2 = 100$ octetos. Para obtener la ocupación de memoria total, basta con añadir a los valores anteriores la ocupación del algoritmo interpretador o simulador (del orden de 80 octetos) y la ocupación memoria de los procedimientos donde se evaluan los eventos y se calculan las acciones.

De acuerdo con lo expresado y admitiendo que normalmente $n \cong m$, el crecimiento de la ocupación de memoria es cuadrático en n (o m), lo que quiere decir que si la RdP es muy grande, la ocupación de memoria puede ser considerable. No obstante, el bajo coste de las memorias y su tendencia a bajar indican que esta dificultad en el crecimiento de la ocupación memoria no será, en principio, un factor limitativo muy importante.

Desde un punto de vista temporal, interesa resaltar que el algoritmo anterior ejecuta m ($\cong n$) tests del tipo $\boldsymbol{E}^j \wedge \bar{M} = 0$, y cada uno puede necesitar $\lceil n/8 \rceil$ operaciones (octetos de $\boldsymbol{E}^j$ y M). Es decir, la duración de un ciclo de simulación es también cuadrática en n (admitido que $n \cong m$).

Para mejorar las prestaciones de los métodos matriciales, en §9.5 se procederá a:

1) Representar mediante *listas* la estructura de la RdP, con lo que se pretende reducir la ocupación memoria. En efecto, la esperada reducción se basará en que, normalmente, $\boldsymbol{E}$ y $\boldsymbol{F}$ son matrices *cuasi-vacías.*

2) Iterar en la simulación sobre el *subconjunto de transiciones sensibilizadas* (u otro subconjunto que lo comprenda, pero de cardinal similar) en vez de sobre el conjunto de *todas* las transiciones de la RdP. Con esta nueva política de simulación se pretende reducir el tiempo de ejecución de un paso de simulación.

9.4.3 Una realización del simulador matricial

Para obtener una realización más eficiente del simulador considerado en §9.4.1, pueden tenerse en cuenta las siguientes observaciones:

1) Después de ejecutarse cualquier procedimiento evento-acción (*pevac*), siempre se vuelve al mismo punto del simulador, por lo cual la dirección de retorno puede estar en ellos. Es decir, los «*pevac*» no tienen porque ser *subprogramas cerrados.*
2) Después del retorno de cada «*pevac*» existe un nuevo test sobre $\mathcal{E}$, por lo que en lo sucesivo, los «*pevac*» se estructurarán como sigue:

```
pevac-i: 1) Calcular E;
         2) si E
              entonces
                Ejecutar las acciones asociadas al disparo de t_i;
                ir a ACTMAR;      {actualizar el marcado}
              si no
                ir a SIGTRA;      {siguiente transición}
            fsi;
```

De acuerdo con lo anterior, el algoritmo de simulación puede adoptar la estructura siguiente:

```
1) CCICLO:  i := 1; {comienzo de ciclo}
            Muestrear las entradas;
2) SENSIB:  Calcular E^i ∧ M̄
            si E^i ∧ M̄ = 0 {t_i está sensibilizada}
              entonces ir a pevac-i fsi;
3) SIGTRA:  i := i + 1;
            si i = m + 1 {se trataron todas las transiciones}
              entonces M̄ := M̄+; {fin de ciclo de tratamiento}
                       ir a CCICLO;
              si no    ir a SENSIB
            fsi;
4) ACTMAR: M̄+ := M̄+ ⊕ F^i;
            ir a SIGTRA;
```

Si se adopta la estructura de datos ilustrada por la figura 9.3, la tabla 9.4 presenta una codificación «optimizada» del simulador en lenguaje ensamblador del microcomputador MOTOROLA 6801. En ésta se han empleado las siguientes matrices y vectores:

1) MIPREV, que representa, transición por transición, la información sobre la *matriz de incidencia previa* ($\boldsymbol{E}$) y el puntero hacia los correspondientes procedimientos evento-acción, «*pevac*».

Tabla 9.4. Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador matricial de RdP binarias y ejemplo de RdP (figura 9.1).

```
* Programa de simulación del interpretador matricial síncrono de RdP
*
*
* Programa interpretador. Su ejecución comienza en CCICLO.
*
*ACTualización del MARcado
*
ACTMAR  LDAA    PTMIPR + 1      Apuntar sobre MBFLUJ a la transición dis-
        STAA    PTMFLJ + 1      parada, k
*
        LDS     PTMFLJ          El registro SP apunta a MBFLUJ
        LDX     #MAUXM          El registro X apunta a MAUXM
        LDAB    #NUMLUG         Cargar el Acumulador B con el número de
*                               bytes por transición
*
* BUcle de actualización del MARcado: MAUXM := MAUXM ⊕ MBFLUJ[k]
*
BUMAR   PULA                    Actualización de un byte
        EORA    0,X
        STAA    0,X
        INX                     Preparar el siguiente byte
        DECB                    Decrementar el contador de bytes
        BNE     BUMAR           Hasta que estén todos actualizados
*
* Cálculo de la SIGuiente TRAnsición a testear
*
SIGTRA  LDAB    PTMIPR + 1      Actualizar el puntero sobre MIPREV
        ADDB    #NUMLUG + 2
TSTRAN  CMPB    #LONGC          Si no se han testeado todas las
*                               transiciones,
        BNE     SENSIB          Calcular sensibilización de la transición
*
* Fin de un CICLO de tratamiento
*
```

(Continúa)

Tabla 9.4. (*Cont.*) Codificación en lenguaje ensamblador de MOTOROLA 6801 de un simulador de RdP binarias y ejemplo de RdP (figura 9.1).

```
FCICLO   LDAB    #NUMLUG
         LDX     #CMLMAR
         LDS     #MAUXM-1
COPIA    PULA
         STAA    0,X          } CMLMAR := MAUXM
         INX
         DECB
         BNE     COPIA
*
* Comienzo de CICLO de tratamiento
*
CCICLO   LDAA    VMIS         Muestreo de las entradas y Volcado de la
*                             Memoria Imagen de las Salidas. (La instruc-
*                             ción genera estas señales por hardware
*                             que detecta un acceso a la posición de me-
*                             moria VMIS.)
         CLRB                 Iniciar el puntero sobre MIPREV
*
* Cálculo de la SENSIBilización de una transición
*
SENSIB   STAB    PTMIPR + 1
         LDS     PTMIPR       El registro SP apunta a MIPREV
         LDX     #CMLMAR      El registro X apunta a CMLMAR
         LDAB    #NUMLUG      Cargar Acc. B con el n. bytes/transición
*
* La transición está sensibilizada si y sólo si MIPREV · CMLMAR = 0
*
BUCLE1   PULA                 Verificar la condición sobre un byte
         ANDA    0,X
         BNE     SIGTRA       Si MIPREV · CMLMAR ≠ 0, probar con otra
*                             transición
         INX                  Si no, preparar siguiente byte
         DECB
         BNE     BUCLE1       Hasta verificar todos.
*
* La transición está sensibilizada, por lo que se saltará al «pevac» correspondiente. Si la
* transición se ha disparado, la vuelta se realiza sobre ACTMAR, si no sobre SIGTRA.
*
```

(*Continúa*)

Tabla 9.4. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador matricial de RdP binarias y ejemplo de RdP (figura 9.1).

```
*
IRPEVA   PULX                    Leer dirección del «pevac»
         JMP     0,X
*
* DEFINICIÓN DE LA ESTRUCTURA DE DATOS UTILIZADA PARA DESCRIBIR LA
*RdP DE LA FIGURA 9.1
*
NUMLUG   EQU     1               NUMero de bytes necesarios para represen-
                                 tar los LUGares de una transición
NUMTRA   EQU     6               NUMero de TRAnsiciones de la RdP
*
* Definición de la Matriz de Incidencia PREVia. Como puede observarse, ésta está repre-
* sentada por filas (transiciones). Cada una de ellas ocupa (NUMLUG + 2) bytes (los 2
* bytes corresponden a la dirección del «pevac» asociado a la transición
*
         ORG     (* + $FF)!.$FF00 Debe comenzar página
MIPREV   FCB     0               Reservado para apuntar con el SP
         FCB     %10000000       {t1}
         FDB     PEVAC1
         FCB     %01000000       {t2}
         FDB     PEVAC2
         FCB     %00010000       {t3}
         FDB     PEVAC3
         FCB     %00101000       {t4}
         FDB     PEVAC4
         FCB     %00001000       {t5}
         FDB     PEVAC5
         FCB     %00000100       {t6}
         FDB     PEVAC6
*
* Definición de la Matriz Booleana de FLUjo. Como puede observarse ésta está represen-
* tada por filas (transiciones). Cada una de ellas ocupa (NUMLUG + 2) bytes (los 2 bytes
* nulos corresponden a la dirección del «pevac» en MIPREV, y se reservan por razones
* de eficiencia)
*
         ORG     (* + $FF)!.$FF00 Debe comenzar página
MBFLUJ   FCB     0               Reservado para apuntar con el SP
```

(*Continúa*)

Tabla 9.4. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador matricial de RdP binarias y ejemplo de RdP (figura 9.1).

```
            FCB  %11010000              {t1}
            FDB  0
            FCB  %01100000              {t2}
            FDB  0
            FCB  %00011000              {t3}
            FDB  0
            FCB  %10101000              {t4}
            FDB  0
            FCB  %00001100              {t5}
            FDB  0
            FCB  %00001100              {t6}
            FDB  0
*
LONGC       EQU  (NUMLUG+2)*NUMTRA      Numero de bytes de MIPREV (o de
*                                       MBFLUJ)
*   Definición del marcado inicial
*
CMLMAR      FCB  %01111111              Complemento de la Memoria de Lugares
*                                       MARcados (p1 marcado)
MAUXM       FCB  %01111111              Memoria AUXiliar del Marcado de lugares
*                                       (p1 marcado)
*
*   Definición de los punteros utilizados
*
PTMIPR      FDB  MIPREV                 PunTero sobre la Matriz de Incidencia
*                                       PRevia
PTMFLJ      FDB  MBFLUJ                 PunTero sobre la Matriz booleana de
*                                       FLujo
*
            END  CCICLO
```

2) MBFLUJ, que representa la información booleana sobre el *flujo de marcas* asociado al disparo de cada transición (F).
3) CMLMAR, memoria que representa el *complemento lógico del vector de marcado de los lugares* (Complemento Memoria Lugares Marcados)
4) MAUXM, memoria (vector) auxiliar del marcado.

Los punteros PTMIPR y PTMFLJ señalan la transición cuya condición de sensibilización se está calculando y la última transición disparada o en curso de disparo,

respectivamente. En la figura 9.3 PTMIPR apunta a t_3 mientras que PTMFLJ apunta a t_1. Como la simulación es síncrona, CMLMAR representa el complemento del marcado ilustrado en la figura 9.1 y MAUXML representa el complemento del marcado obtenido a partir del anterior, tras disparar t_1.

Observación. En la codificación de la tabla 9.4 se han adoptado las dos siguientes restricciones:

1) La matriz MIPREV (y por tanto MBFLUJ) puede ocupar, como máximo, 256 octetos. Es decir, la dimensión de la RdP simulable ha de ser tal que: $m(\lceil n/8 \rceil + 2) < 256$ lo que significa, por ejemplo, que se pueden simular redes con 32 lugares y 42 transiciones, o 40 lugares y 36 transiciones.
2) Las direcciones de comienzo de MIPREV y MBFLUJ son tales que el segundo octeto es uno (comienzan página, desperdiciando el primer octeto), lo que unido a la restricción del punto anterior, implica que el primer octeto de los punteros PTMIPR y PTMFLJ serán constantes.

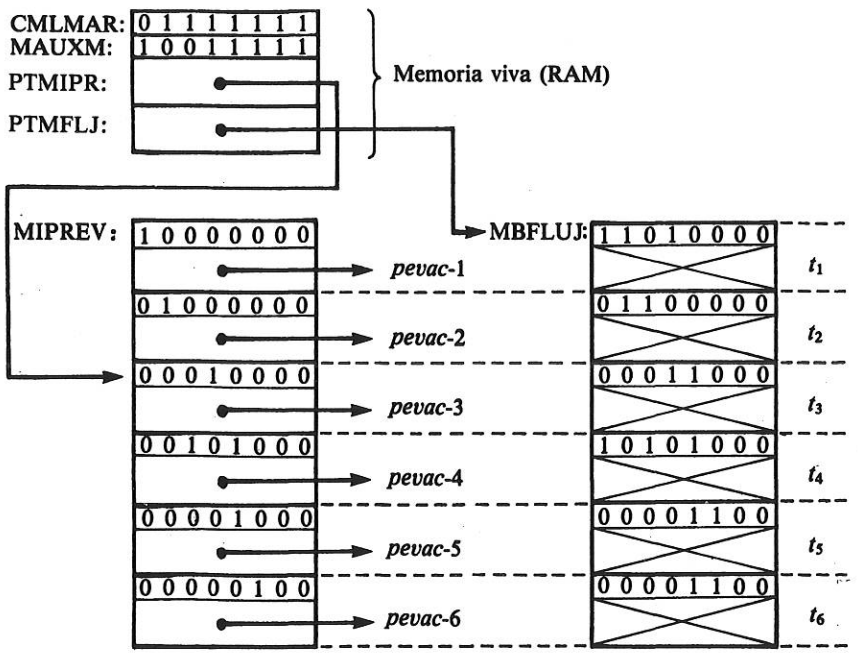


Figura 9.3. Una estructura de datos adoptada para representar matricialmente RdP. (En concreto se representa la red de la figura 9.1.)

9.5 REPRESENTACIONES BASADAS EN LISTAS (I): ESQUEMA BÁSICO

La ocupación de memoria de la representación matricial de una RdP ordinaria es básicamente proporcional a *nm*. En éste y en los siguientes apartados se introducen representaciones con listas, cuyas ocupaciones de memoria son normalmente linea-

les en n y m. (La reducción en la ocupación de memoria proviene de la representación exclusiva de los valores no nulos de las matrices, dado que éstas son, normalmente, cuasi-vacías.)

Después de considerar informalmente los conceptos de *lista* y *puntero* (§9.5.1) se introduce un esquema básico para la representación de RdP con listas (§9.5.2). En los apartados que siguen se plantearán conceptos y técnicas que permitan reducir aún más la ocupación de memoria de la codificación de RdP y, sobre todo, reducir la duración de la simulación (duración del ciclo tratamiento). Este último objetivo conducirá a esquemas de simulación *dirigidos por el marcado* (§9.6) o *las transiciones sensibilizadas* (§9.7).

9.5.1 Nociones previas†

Una *lista lineal* es una estructura de datos en la que un conjunto de elementos se presentan totalmente ordenados de una cierta forma. La relación de orden se expresa conceptualmente mediante la función *siguiente*. El elemento «*siguiente* (*e*)» es el sucesor del elemento *e* en la lista. Si *u* es el último elemento de la lista, *siguiente* (*u*) tomará el valor *nulo* que indicará la no existencia de elemento sucesor de *u*.

Para representar la relación de orden se pueden utilizar dos tipos de técnicas distintas:

1) Representaciones *contiguas*, en las que el $(k + 1)$-ésimo elemento se representa en la memoria a continuación del k-ésimo.
2) Representaciones *discontiguas* (*encadenadas*) o con *punteros*, en las que la ordenación de los elementos en la lista no tiene que coincidir con la ordenación de su representación en memoria. En este caso, la definición de *siguiente* (*e*) exige la presencia de una información asociada al elemento *e* que permita calcular la dirección donde se encuentra la información relativa al elemento sucesor. Esta información se denomina *puntero*. Para el último elemento de la lista, el puntero toma el valor *nulo*, que indica la no existencia de elemento sucesor.

Nota. En las representaciones contiguas de listas puede decirse que el puntero está implícito.

La figura 9.4 esquematiza dos representaciones de una RdP mediante listas en las que cada elemento debe codificar una *transición*. La primera lista utiliza explícitamente punteros para encadenar los elementos. Desde el punto de vista de la programación, cada uno de esos punteros codificará la dirección que corresponde a la posición de memoria donde se encuentra la información relativa al siguiente elemento. Puesto que t_6 es la última transición de la red, el puntero asociado no señala a ningún otro elemento, y toma el valor *nulo*. (Si no se va a ubicar ningún elemento a partir de la posición $\emptyset$, es frecuente tomar este valor de puntero como *nulo*. Si el puntero vale $\emptyset$, ello no quiere decir que el siguiente elemento se encuentre a partir de la dirección $\emptyset$, sino que no existe elemento sucesor.)

---

† La información que sobre listas se presenta en este apartado es absolutamente introductoria. Para profundizar acerca de estas estructuras de datos, se recomienda la lectura de textos sobre programación de computadores tales como [WIRT 76], por ejemplo.

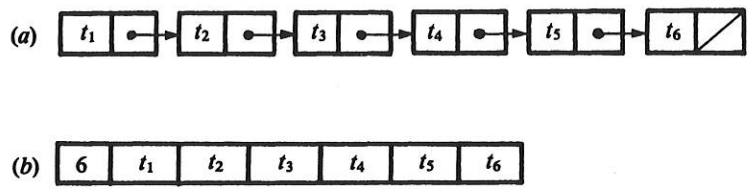
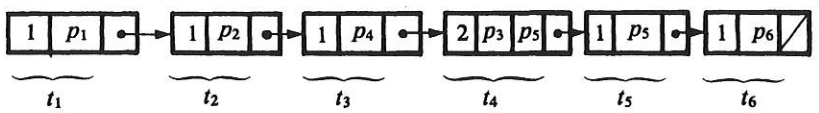


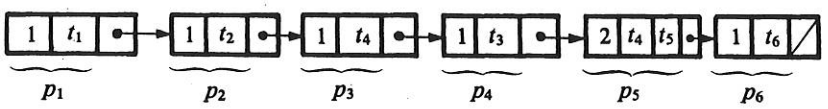
Figura 9.4. Dos representaciones físicas de una RdP con seis transiciones mediante una lista. (Cada elemento debe disponer de toda la información relativa a la transición correspondiente.)

La figura 9.4*b* esquematiza una segunda representación, en la que se precodifica el número de elementos, 6, y, posteriormente, se les considera uno a continuación de otro (los punteros no aparecen explícitamente, ya que el siguiente elemento en memoria es el sucesor en la lista). En lo sucesivo sólo se utilizarán estas dos representaciones de una lista. La primera de ellas se empleará para representar una RdP como *una lista de transiciones*. La segunda forma se utilizará para representar, por ejemplo, la lista de lugares de entrada asociada a una transición.

La representación de una matriz mediante una lista puede hacerse columna por columna o fila por fila. Si para economizar en la ocupación de memoria sólo se representan los valores no-nulos, cada elemento de la lista representará una fila o columna de la matriz (respectivamente) y será, a su vez, una lista. La figura 9.5 codifica de dos formas distintas la matriz de incidencia previa de la RdP de la figura 9.1. Como puede observarse se ha utilizado la propiedad de que E es una matriz booleana, por lo que un elemento está representado si y sólo si vale «1». De todo lo anterior resulta fácil deducir que este tipo de representación de matrices es especialmente interesante si las matrices son cuasi-vacías.



(*a*) Representación por columnas (transiciones)



(*b*) Representación por filas (lugares)

Figura 9.5. Representación mediante listas de la matriz E (tabla 9.2).

9.5.2 Esquema básico de representación de RdP con listas

La regla de evolución del marcado de una RdP está definida a partir de las transicio-

nes, por lo cual toda codificación de una red debe presentar de forma fácilmente accesible la información asociada a cada transición. Por ello, para codificar con listas una RdP, conviene considerar listas de transiciones. Si se representa la RdP con una única lista de transiciones (figura 9.4*a*), cada elemento de la lista deberá poseer toda la información relativa a una transición.

La transformación directa del esquema de representación matricial (§9.4) indica que cada elemento debe contener:

1) La *lista de lugares de entrada de la transición* (con ello se representa la matriz de incidencia previa, E).
2) La *lista de lugares cuyo marcado evoluciona al disparar la transición* (con ello se representa la matriz booleana de flujo de marcas, F).
3) El *puntero* hacia el procedimiento evento-acción asociado a la transición (*pevac-i*).
4) El puntero hacia la *siguiente transición*.

De acuerdo con el esquema anterior, la figura 9.6*a* codifica la información relativa a la transición t_1. Dado que, normalmente, los lugares de entrada a una transición deben cambiar su marcado al dispararse ésta, el esquema anterior no será muy eficiente (los lugares de entrada se representarán, normalmente, dos veces). Para remediar esta fuente de ineficiencia se puede proceder a representar la RdP mediante las matrices de incidencia previa e incidencia posterior. La figura 9.6*b* representa t_1 utilizando las listas de lugares de entrada y de salida.

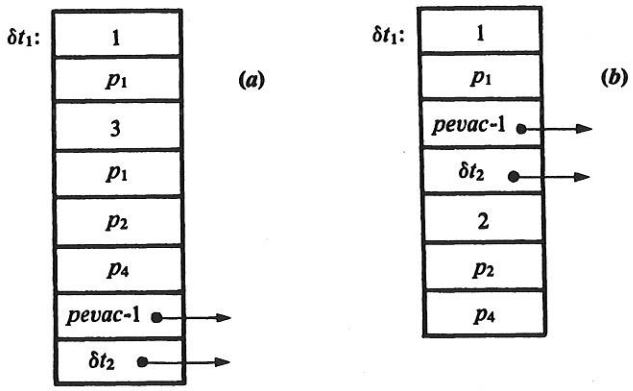


Figura 9.6. Dos codificaciones de la información relativa a una transición. (*Nota.* δt_i es la dirección donde comienza la información asociada a t_i).

Nota. En ambas listas la colocación de los punteros hacia *pevac*-1 y t_2 sólo responde a criterios de eficiencia en la simulación, pues de este modo podrá realizarse más rápidamente.

Adoptando el esquema de la figura 9.6*b*, la codificación de la estructura de una RdP se obtiene considerando el conjunto de todas las transiciones de la red.

La representación del *marcado* de una red binaria puede llevarse a cabo de múltiples formas. Por ejemplo, mediante:

a) Un vector booleano de marcado, M, y otro vector auxiliar, necesario para realizar las evoluciones síncronas, M^+. Al finalizar un paso de simulación se hará $M := M^+$.
b) Un vector booleano de marcado, M, y una lista auxiliar que se construirá en cada paso de simulación (ciclo de tratamiento) conteniendo los lugares cuyo marcado, debido a los disparos realizados, ha de cambiar. Al finalizar el paso de simulación se *complementará* el marcado de todos los lugares que aparezcan en la lista.
c) Una lista con los lugares marcados y otra lista auxiliar que se construirá en cada paso de simulación con lugares que deberán estar marcados al comenzar el siguiente paso. Al finalizar el paso de simulación, la lista recién construida pasará a ser la lista de los lugares marcados mientras que la nueva lista auxiliar se inicializará como vacía.

Como puede observarse, los tres casos enunciados utilizan diversas técnicas para la representación y construcción del nuevo marcado. Así, desde el punto de vista de las técnicas de representación del marcado, se utilizan las siguientes estructuras de datos:

a) 2 vectores: M y M^+
b) Vector M y lista de variaciones en el marcado.
c) 2 listas: M y construcción del nuevo marcado, M^+

La utilización de listas será tanto más interesante cuanto mayor sea el porcentaje de los lugares no marcados (más vacío esté el vector del marcado).

En lo que concierne a la construcción del nuevo marcado, en los tres casos anteriores se utilizan los siguientes principios de operación:

a) Operar sobre el marcado M para obtener el nuevo marcado, lo que representa la aplicación directa de la ecuación de estado. (Es el principio utilizado en §9.4.)
b) Construir una representación de las variaciones sobre el marcado, que finalmente aplicadas a éste permita obtener el nuevo marcado.
c) Construir el nuevo marcado sin utilizar «directamente» el marcado anterior. En efecto, si la RdP es binaria, el disparo de una transición hará que se marquen sus lugares de salida mientras que sus lugares de entrada serán desmarcados. Por otro lado, si no se dispara ninguna de las transiciones de salida de un lugar marcado, éste permanecerá marcado.

La codificación de la estructura de la RdP como una lista de transiciones (codificación similar a la de la figura 9.5*b*) en la que cada transición adopta el esquema de la figura 9.6*b* permite una fácil utilización de las codificaciones del marcado denominadas «*a*» y «*b*». La simulación eficiente de RdP utilizando la codificación del marcado denominada «*c*» exige una representación de la estructura de la RdP diferente de la considerada. Sobre este punto se volverá posteriormente, en §9.6.

Adoptando las estructuras de datos antes mencionadas para codificar la RdP, el algoritmo de simulación puede tener la siguiente forma:

```
CCICLO:   Muestrear las entradas; {comienzo de ciclo}
          t_i := t_1; {i: = 1}
          mientras que δt_i ≠ nulo hacer
SENSIB:     Calcular la condición de sensibilización de t_i, Σ_i ;
            si Σ_i = cierto
              entonces ir a pevac-i ;
            fsi;
SIGTRA:   t_i := siguiente (t_i); {i := i + 1}
          fmq;
          Obtener el nuevo marcado;
          ir a CCICLO;
ACTMAR:   Actualizar el marcado;
          ir a SIGTRA
```

De acuerdo con lo expresado en §9.4.3 sobre la estructura que en lo sucesivo se supondrá a los procedimientos evento-acción, el retorno de *pevac-i* al algoritmo de simulación se producirá mediante saltos a ACTMAR o SIGTRA, según que la transición se haya disparado o no. Si el marcado se representa mediante los vectores M y $M,^{+}$ actualizar el nuevo marcado consiste en modificar el valor de M^{+}, mientras que obtener el nuevo marcado consiste en copiar M^{+} sobre $M(M: = M^{+})$.

Si el marcado se representa mediante el vector M y la lista de lugares cuyo marcado debe cambiar, la actualización del marcado sólo añade a la lista los lugares de entrada y de salida de la transición disparada. Por último, la obtención del nuevo marcado se producirá en este caso complementando el valor lógico de las posiciones de M correspondientes a los lugares que aparecen en la lista considerada (recuérdese que sólo se tratan RdP binarias).

Observación. Para una programación más eficiente en lenguaje ensamblador del anterior algoritmo, se recomienda reestructurarlo, presentándolo de la forma siguiente (su ejecución comienza en CCICLO):

```
ACTMAR:  Actualizar el marcado;
SIGTRA:  t_i := siguiente (t_i); {i := i + 1}
         si t_i = nulo {se han terminado las transiciones a tratar, i = m + 1}
           entonces Obtener el nuevo marcado;
CCICLO:             Muestrear las entradas;
                    t_i := t_1; {i:= 1}
         fsi;
SENSIB:  Calcular la sensibilización de t_i, Σ_i;
         si Σ_i = cierto entonces ir a pevac-i;
                       si no     ir a SIGTRA;
         fsi;
```

9.5.3 Prestaciones

La ocupación de memoria de la representación anterior puede calcularse fácilmente. Si se utiliza un microcomputador de 8 bits, representando los punteros con 2 octetos y los enteros con un único octeto, la transición t_i necesitará $6 + |^{\cdot} t_i| + |t_i^{\cdot}|$ octectos. Definiendo el *factor de paralelismo*, f_p, como el número medio de lugares de salida de una transición y el *factor de sincronización*, f_s, como el número medio de lugares

de entrada a una transición, la representación es memoria de la estructura de la RdP ocupará $m \cdot (6 + f_p + f_s)$ octetos†. Por otro lado, la representación del marcado, M y M_i^+ será lineal en n ($2 \lceil n/8 \rceil$ octetos si se utilizan dos vectores).

Observación. Aunque la ocupación de memoria es ahora lineal en n y m, la ocupación de la representación matricial de RdP ordinarias (§9.3.1) puede ser inferior con redes de dimensión relativamente pequeñas. Así, si $n = m = 16$ y $f_p = f_s = 1.25$, la ocupación con listas será de 140 octectos mientras que la ocupación de la representación matricial podrá ser de 100 octetos.

La duración de un paso de simulación es algo más complicada de establecer, pero en cualquier caso es fácil observar que, aunque empleándose técnicas distintas, tanto con la representación matricial como con la basada en listas, se calcula la condición de sensibilización de *todas* las transiciones. La reducción de cálculo de condiciones de sensibilización a subconjuntos de transiciones permitirá reducir de forma notoria la mencionada duración.

Las anteriores reflexiones sobre las prestaciones indican que los métodos de simulación de RdP considerados en este apartado (§9.5) no tienen un gran interés en sí mismos. Básicamente han sido presentados para introducir de forma progresiva las realizaciones en las que la simulación es *dirigida por el marcado*. Éstas se abordan en el próximo apartado.

9.6 REPRESENTACIONES BASADAS EN LISTAS (II): SIMULACIÓN DIRIGIDA POR EL MARCADO

Las diferentes transformaciones a que se someterá en este apartado a la codificación de RdP presentada en §9.5.2 persiguen una más eficiente representación (menor ocupación de memoria) y una más rápida simulación.

Para proceder de forma progresiva en la exposición, se parte de la simulación de una subclase de RdP bien conocida, los *grafos reducidos* (GR). La generalización de las ideas básicas sobre la simulación de GR conducirá a métodos de simulación de RdP dirigidos por el marcado. En este punto conviene resaltar que la previa presentación de técnicas de simulación de GR está motivada por razones de tipo pedagógico ya que los esquemas de simulación de §9.6.2 y §9.6.4 pueden obtenerse directamente por transformación del presentado en §9.5.2.

9.6.1 Simulación de grafos reducidos

La figura 9.7*a* presenta un GR. Como en un GR todas las transiciones poseen un único lugar de entrada y otro de salida, la codificación de las transiciones pueden simplificarse (véase la figura 9.7*b* para t_1).

---

† Para un GE $f_p = f_s = 1$. Normalmente, pero no siempre, en las RdP se tiene $f_p = f_s$. Sus valores suelen estar comprendidos entre 1y 1,5.

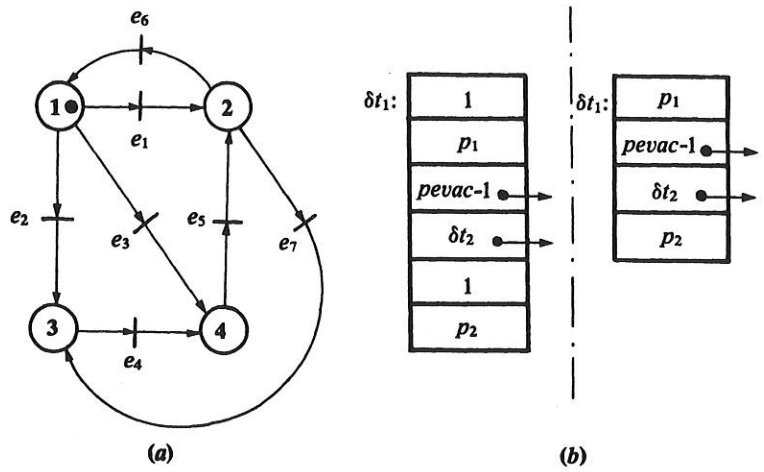


Figura 9.7. Grafo reducido y simplificación de la codificación de t_1.

A partir de la simplificación introducida, la figura 9.8*a* presenta, parcialmente, una codificación del GR anterior. Para mejorarla se van a introducir conjuntamente dos modificaciones:

1) *Factorizar las transiciones de salida de un lugar*, lo que conducirá a una reducción de los datos necesarios para codificar la estructura de la RdP.
2) Dirigir el proceso de simulación del GR de forma que sólo se considere el código asociado a las transiciones de salida del *único lugar marcado* que haya en cada paso de simulación.

El lugar p_1 (figura 9.7*a*) tiene tres transiciones de salida, por lo que en cada una de ellas (figura 9.8*a*) aparece p_1 como lugar de entrada. La anunciada factorización permite escribir como código asociado a p_1 el que aparece en la figura 9.8*b*. En la mencionada figura se puede apreciar que a partir de p_1 se apunta *la lista* de sus transiciones de salida. Habida cuenta que después de t_3 no existe ninguna otra transición de salida de p_1, el puntero de «*siguiente transición*» toma el valor *nulo*. Si f_d (factor de descendencia) indica el número medio de transiciones de salida de un lugar, la economía de memoria que se obtiene será de $n \cdot (f_d - 1) \cdot 2$ octetos, dado que:

1) cada dirección ocupará 2 octetos;
2) sólo se tiene una dirección asociada al lugar de entrada (el puntero que está al lado del vector de marcado en la figura 9.8*b*).

Para el GR de la figura 9.7*a*, la reducción en memoria es de $4 \cdot (7/4 - 1) \cdot 2 = 6$ octetos.

En otro orden de ideas, puesto que un GR tiene siempre marcado un único lugar, sólo estarán sensibilizadas sus transiciones de salida y, por lo tanto, sólo podrá dispararse una de éstas. Para acelerar la simulación, en vez de considerar en cada paso todas las transiciones del GR, bastará con tener en cuenta *las de salida del lugar marcado*. La primera forma de realizar esta idea se expone globalmente en la figura 9.8*b* en la que el acceso a las transiciones de salida del lugar marcado se realiza *escrutando* el vector del marcado.

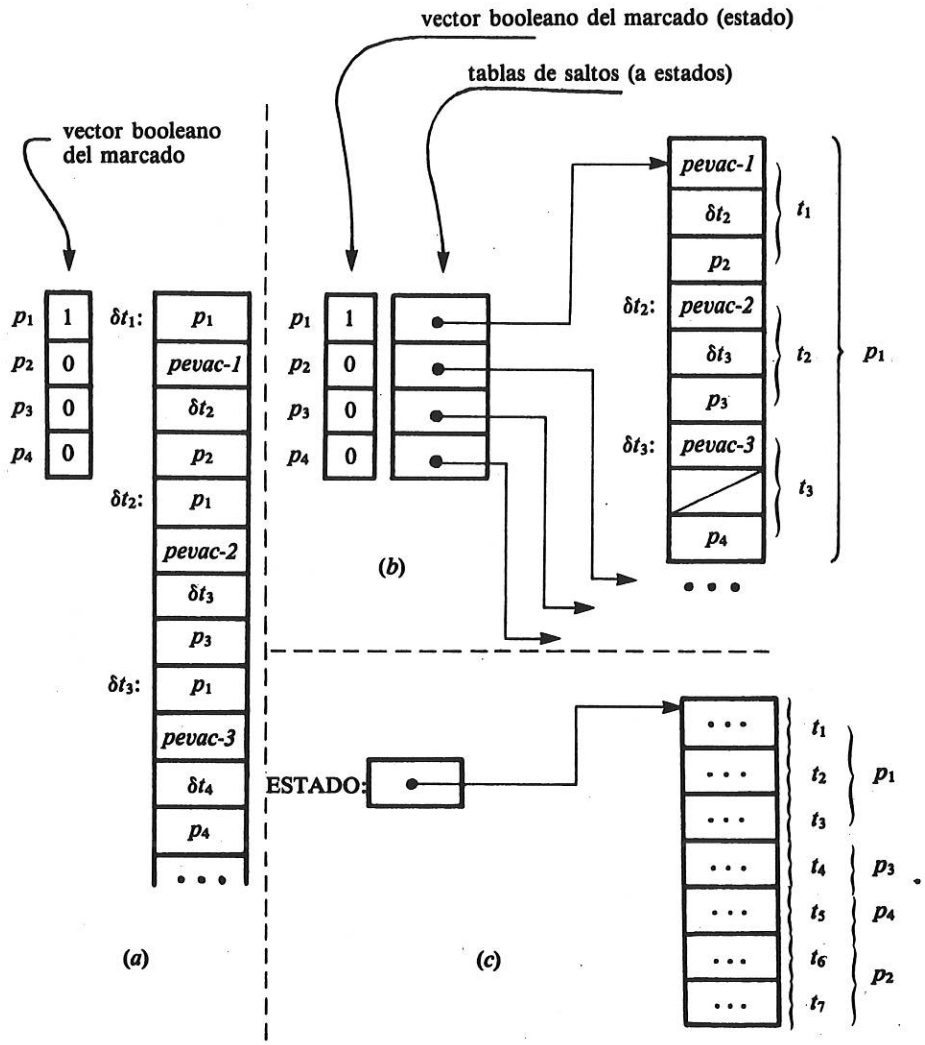


Figura 9.8. Diversas formas de codificar la estructura y el marcado de un GR (el de la figura 9.7a). (*Nota.* En la figura «*b*», la siguiente transición de t_3 es un puntero nulo, puesto que p_1 no posee ninguna otra transición de salida.)

En esencia, el algoritmo de simulación adopta la estructura siguiente:

```
CCICLO:  Muestrear las entradas;
         i := 1;
SENSIB:  si M[pi] = 1
           entonces tj := primera transición apuntada por pi;
                    ir a pevac-j;
         fsi;
```

```
SIGLUG:  i:= i + 1;
         si i < = n entonces ir a SENSIB;
                     si no    ir a CCICLO;
         fsi;
SIGTRA:  tj : = siguiente (tj);
         si tj = nulo entonces ir a CCICLO;
                       si no    ir a pevac-j;
         fsi;
ACTMAR: Actualizar el marcado;
         ir a CCICLO;
```

Observación. El algoritmo anterior hace terminar la actualización del marcado con «**ir a** CCICLO» puesto que se supone una simulación síncrona y, por lo tanto, en un GR sólo se puede disparar una transición. Otra forma menos eficiente, pero más próxima a la que se adoptará para la simulación de RdP (en un paso pueden dispararse varias transiciones) consiste en poner «**ir a** SIGLUG» y utilizar otra estructura de datos auxiliar para construir el siguiente marcado.

La mejora temporal obtenida al dirigir la simulación por el marcado es tanto más acusada cuanto mayor sea la relación entre el número total de transiciones y el número medio de transiciones de salida de los lugares (es decir, el número medio de transiciones sensibilizadas).

Para concluir esta introducción a la simulación de GR basta observar que el *vector del marcado* es cuasi-vacío y, por consiguiente, puede ser interesante codificarlo como una *lista*. Ahora bien, dado que en un GR siempre habrá un único lugar marcado, la lista tendrá en permanencia un único elemento (véase la figura 9.8*c*).

Procediendo del modo apuntado, se pueden obtener dos ventajas:

1) Reducir la ocupación de memoria (esta reducción será tanto más importante cuanto mayor sea el número de lugares).
2) Eliminar la escrutación del marcado de los lugares, puesto que se obtendrá directamente el puntero que señala las transiciones sensibilizadas (transiciones de salida del lugar marcado).

Los algoritmos de simulación que codifican la información sobre el GR de la forma indicada en la figura 9.8*c* adoptan, básicamente, el esquema siguiente:

```
CCICLO:  Muestrear las entradas;
         tj : = transición apuntada por ESTADO;
         ir a pevac-j;
SIGTRA:  tj : = siguiente (tj);
         si tj = nulo entonces ir a CCICLO;
                       si no    ir a pevac-j
         fsi;
ACTMAR: Actualizar el marcado; {despositar en ESTADO la dirección
                                del lugar de salida de tj}
         ir a CCICLO;
```

Observación. La actualización del marcado termina con un «**ir a** CCICLO» puesto que en un GR sólo hay un lugar marcado; si se considerasen RdP podrían haber varios lugares marcados y, por lo tanto, habrá que buscar el siguiente elemento de la lista de lugares marcados, hasta escrutar completamente la lista.

EJERCICIO. Modifíquense los algoritmos anteriores para el caso en el que se considere la simulación de q ($q > 1$) GR.

9.6.2 Simulación de RdP dirigida por escrutación del vector de marcado

En este apartado se generaliza el primer esquema presentado para la simulación dirigida por el marcado de grafos reducidos (GR) (§9.6.1). Las diferencias más importantes que se observarán en la simulación de RdP binarias con respecto a la de los GR provienen de la posibilidad de que existan:

1) varios lugares marcados simultáneamente.
2) transiciones con varios lugares de entrada y/o de salida.

De las diferencias apuntadas anteriormente, la que implicará mayores consideraciones será el que una transición pueda tener varios lugares de entrada. En efecto, ello hace que existan varios lugares que poseen la transición como una transición de salida. Dado que, evidentemente, una transición sólo debe considerarse una vez en cada paso de simulación, habrá que designar un único lugar de entrada para realizar los cálculos que, eventualmente, puedan llevar a su disparo.

Si al lugar p_i se le asocia la información correspondiente al disparo de la transición t_j (t_j será una transición de salida de p_i), se dirá que p_i *representa a* t_j. Si p_i es el único lugar de entrada de t_j, sólo él la puede representar y se dirá que p_i es el *lugar representante esencial de* t_j.

De acuerdo con lo expresado, al considerar la RdP de la figura 9.1 se observa que p_1, p_2, p_4, p_5 y p_6 son lugares representantes esenciales (representan a t_1, t_2, t_3, t_5 y t_6 respectivamente). Por otro lado, t_4 puede ser representada por p_3 o por p_5. Si se decide representar a t_4 por p_5, p_3 no representará a t_4. Se dirá que p_3 es un *lugar de sincronización* con respecto a t_4.

De una forma general se dirá que p_k es un *lugar de sincronización con respecto a la transición* t_q si, siendo lugar de entrada, no la representa. Si un lugar como p_3 (figura 9.1) sólo es de sincronización, cumple un mero papel de *indicador* (*flag*).

En un grafo de estados todos los lugares son representantes esenciales y, por lo tanto, no existen lugares de sincronización.

A modo de resumen, la codificación de la estructura de una RdP a partir de sus lugares puede realizarse, en principio, considerando cualquier subconjunto de lugares cuyas transiciones de salida *cubran* al conjunto de transiciones de la red. El conjunto de lugares utilizados en una cobertura dada estará formado por lugares representantes. Dado que pueden existir diversas coberturas para una RdP, se plantea de forma natural el problema de elegir la «*mejor cobertura*» con respecto a un determinado criterio. Para no interrumpir la introducción de ideas básicas sobre la simulación de RdP por escrutación del vector de marcados, se continuará con la codificación y simulación de una red. En §9.6.3 se volverá sobre la obtención de coberturas de transiciones por sus lugares de entrada.

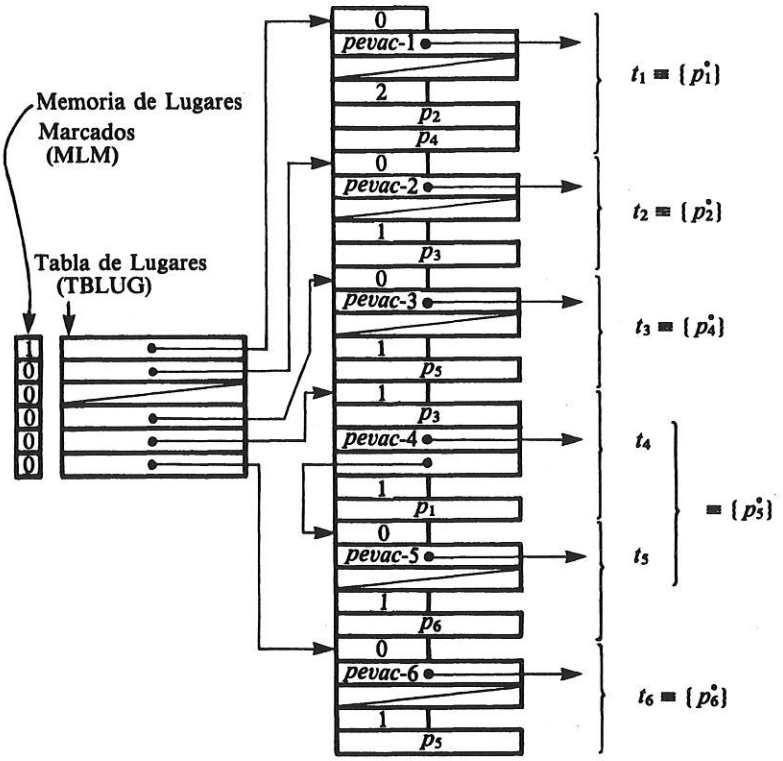


Figura 9.9. Codificación de la RdP de la figura 9.1. (La simulación no podrá ser síncrona debido a que existe un único vector para el marcado, MLM.) (*Nota.* El puntero de TBLUG asociado a p_3 es nulo puesto que p_3 no representa a ninguna transición.)

La figura 9.9 presenta una codificación de la estructura y marcado de la RdP de la figura 9.1, si se decide que p_5 represente a t_4. A partir de cada lugar existe una lista de transiciones representadas (para p_3 la lista es vacía puesto que este lugar no representa a t_4). La información relativa a cada transición adopta la estructura siguiente:

1) Lista de lugares de sincronización para t_j, que comprende:
   1.1 Su número ($|{}^\bullet t_j| - 1$).
   1.2 La definición de sus elementos (direcciones de los lugares).
2) Puntero al procedimiento evento-acción asociado a la transición, *pevac-j*.
3) Dirección de la siguiente transición representada por el mismo lugar. Si no existe otra más, el puntero tomará el valor nulo.
4) Lista de los lugares de salida de t_j, que comprende:
   4.1 Su número ($|t_j^\bullet|$).
   4.2 La definición de sus elementos (dirección de los lugares).

Al utilizar el esquema anterior, una RdP se *codifica* como un *conjunto de listas.* Cada lista codifica *el conjunto de transiciones de salida representado por un lugar.*

Sea una representación del *marcado* compuesta por:

1) Un vector booleano del marcado, MLM y
2) Un vector auxiliar, MAUXM, o bien una lista de los lugares a marcar y/o desmarcar cuando finalice el paso de simulación.

Un primer esquema de algoritmo de simulación síncrona eficiente puede tener la estructura que sigue (la ejecución de un ciclo comienza en CCICLO):

```
ACTMAR:  Desmarcar lugares de entrada (lugar representante y lugares
         de sincronización) de t_j; {sobre MAUXM}
         Marcar lugares de salida de t_j; {sobre MAUXM}
SIGLUG:  repetir i := i + 1 hasta que MLM [p_i] = 1; {buscar siguiente
                                                      lugar marcado
                                                      en MLM}
         si i = n + 1 {se consideraron todos los lugares de la red}
FCICLO:    entonces Obtener el nuevo marcado; {MLM: = MAUXM}
CCICLO:             i := 0; {inicializar búsqueda}
                    Muestrear las entradas;
                    ir a SIGLUG
           si no    t_j := primera transición descendiente de p_i;
         fsi;
TRATRA:  si t_j es representada por p_i
           entonces
SENSIB:       si p_i es representante esencial de t_j {|•t_j| − 1 = 0}
                 entonces ir a pevac-j; {p_i es el único lugar de entr. de t_j}
                 si no Calcular la sensibilización de t_j, Σ_j;
                       si Σ_j = 1 entonces ir a pevac-j fsi
              fsi;
         fsi;
         t_j := siguiente transición de t_j representada por p_i;
         si t_j = nulo
           entonces ir a SIGLUG;
           si no    ir a TRATRA
         fsi;
```

La comparación de este algoritmo con el presentado en la observación final de §9.5.2 permite resaltar que en la simulación dirigida por el marcado no se consideran todas las transiciones de la RdP, sino el *subconjunto* de las representadas por los lugares marcados. Cabe esperar, por lo tanto, que la simulación sea más rápida con el método recién expuesto. Haciendo referencia una vez más a la figura 9.1, la simulación dirigida por el marcado hará que, como máximo, deban considerarse en cada paso *dos* de las *seis* transiciones existentes.

La tabla 9.5 presenta una codificación del anterior algoritmo en lenguaje ensamblador del microcomputador MOTOROLA 6801.

Una ulterior mejora de prestaciones del programa (algoritmo + estructura de datos) anterior puede obtenerse al considerar que el vector del marcado, MLM, está formado por dos subvectores distintos (no forzosamente disjuntos):

1) el subvector del marcado de *lugares representantes* con respecto a alguna transición (algunos de ellos podrán ser, además, de sincronización).
2) el subvector del marcado de *lugares que no representan ninguna transición* (todos serán exclusivamente lugares de sincronización).

En estas condiciones, para alcanzar la anunciada mejora basta con que se realice la escrutación sobre el subvector del marcado de lugares representantes. Procediendo de esta forma, se podrá reducir el número de lugares a escrutar.

Observación. Si, además, los subconjuntos de lugares representantes y de sincronización son disjuntos, se podrá eliminar el test dispuesto en la etiqueta TRATRA (dado que todos los lugares del subvector que se escrute serán representantes de *todas* sus transiciones de salida). Sobre ello se volverá en §9.6.3.

Tabla 9.5. Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando el vector del marcado.

```
*
* Programa de simulación del interpretador síncrono de RdP utilizando vectores de luga-
* res, con escrutación del marcado. El disparo de cada transición (efectuado en el «pe-
* vac» correspondiente) provoca la actualización del marcado sobre un vector auxiliar
* MAUXM. Al comenzar un nuevo ciclo de tratamiento, este vector es copiado sobre el
* vector de lugares marcados: MLM.
*
* Para reducir el tiempo de actualización del marcado, se ha hecho fijo el desplazamiento
* entre los dos vectores de marcado (constante TOTLUG), con lo que se puede utilizar
* un acceso indexado desde MLM. Sin embargo, esta decisión limita a 255 el número má-
* ximo de lugares de la red.
*
* Programa interpretador. Su ejecución comienza en CCICLO
*
TOTLUG    EQU    7              n = 6 lugares de la RdP (figura 9.1) + 1 (Comb.
*                                General)
*
* ACTualización del MARcado al ser disparada una transición
*
ACTMAR    LDX    PTMLM          Apuntar el lugar representante en MLM
          COM    TOTLUG,X       Modificar el marcado del lugar sobre MAUXM
*
```

(*Continúa*)

Tabla 9.5. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando el vector del marcado.

```
            LDS     DIRCOM
            PULB                    Lectura del número de lugares de sincronización
            TSTB
            BEQ     FACTPR          Si es = 0, actualizar marcado de lugares de salida
*
* ACTualización de los lugares de SINcronización de la transición disparada
*
ACTSIN      PULX                    Apuntar al lugar correspondiente en MLM
            COM     TOTLUG,X        Modificar el marcado de dicho lugar sobre
*                                   MAUXM
            DECB
            BNE     ACTSIN
*
FACTPR      PULX                    Saltarse la dirección del «pevac» asociado
            PULX                    Saltarse la dirección de la siguiente transición
            PULB                    Lectura del número de lugares de salida
*
* ACTualización de los lugares de salida (POSteriores)
*
ACTPOS      PULX                    Apuntar al lugar correspondiente en MLM
            COM     TOTLUG,X        Modificar el marcado del lugar sobre MAUXM
            DECB
            BNE     ACTPOS
*
* Búsqueda del SIGuiente LUGar marcado
*
SIGLUG      LDX     PTMLM           Recuperar la dirección del último lugar marcado
BUSQDA      INX                     Apuntar al siguiente lugar
            LDAA    0,X
            BPL     BUSQDA          Hasta que un lugar esté marcado
            STX     PTMLM           Salvaguardar la dirección del lugar marcado
*
            LDX     #TBDLUG         Apuntar a la tabla de direcciones de lugares
            LDAB    PTMLM + 1
            ABX
            ABX
TTRA        LDS     0,X             Lectura de la dirección asociada al lugar marcado
            BNE     TRATRA          Si no es la última (COMbinatorio), tratar el lugar
*
```

(*Continúa*)

Tabla 9.5. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando el vector del marcado.

```
*
* Fin de un CICLO de tratamiento
*
FCICLO    LDX    #MLM          El registro X apunta a MLM
          LDAA   #TOTLUG-1     Cargar el acumulador A con n - 1
*
* Bucle de COPia del MARcado: MLM:= MAUXM
*
COPMAR    LDAB   TOTLUG,X      Lectura del marcado de un lugar de MAUXM
          STAB   0,X           Copiar el marcado del lugar sobre MLM
          INX                  Apuntar al siguiente lugar
          DECA
          BNE    COPMAR        Hasta que se haya copiado todo el vector
*
* Comienzo de un nuevo CICLO de tratamiento
*
CCICLO    LDAA   VMIS          Muestreo de las entradas y Volcado de la Memo-
*                              ria. Imagen de las salidas. (La instrucción genera
*                              estas señales por hardware: detección de un ac-
*                              ceso a la posición de memoria VMIS.
*
          LDX    #MLM          Iniciar el puntero sobre MLM
          BRA    BUSQDA + 1    Búsqueda del primer lugar marcado
*
* TRAtamiento asociado a una TRAnsición descendiente del lugar marcado
*
TRATRA    STS    DIRCOM        Guarda dirección de tratamiento de la transición
          PULB                 Lectura del número de lugares de sincronización
          TSTB
          BMI    SIGLUG        Si es negativo, el lugar es de sincronización
SENSIB    BEQ    IRPEVA        Si es = 0, la transición está sensibilizada
*
* TeST de SENsibilización de las transiciones descendientes del lugar marcado
*
TSTSEN    LDAA   #1000000B     Cargar el acumulador A con valor lógico «1»
BUCLE     PULX                 Apuntar con el reg. X a un lugar de sinc. pi
          ANDA   0,X           A ← (A) · M [pi]
          DECB
```

(*Continúa*)

Tabla 9.5. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando el vector del marcado.

```
         BNE     BUCLE           Hasta haber recorrido todos lugares de sincroni-
*                                zación
         TSTA
         BPL     NODISP          Si el producto lógico es = 0, no está sensibilizada
*
* Saltar a ejecutar el «pevac» asociado a la transición
*
IRPEVA   PULX                    Lectura de la dirección de salto
         JMP     0,X
*
* Cálculo de la SIGuiente TRAnsición a tratar
*
NODISP   PULX                    Saltarse la dirección del «pevac» asociado
SIGTRA   TSX
         LDS     0,X             Lectura de la direcc. de la sig. transición descend.
         BNE     TRATRA
         JMP     SIGLUG          Si es = 0, buscar un nuevo lugar marcado
*
* ESTRUCTURA DE DATOS ASOCIADA A LA RdP DE LA FIGURA 9.1
*
* Vector Memoria de los Lugares Marcados. El marcado de un lugar está representado
* por el bit más significativo del byte asociado. El máximo número de lugares marcados
* es de 255 (incluido el Comb.)
*
         ORG     (*+$FF)!.$FF00  Debe comenzar página
*
MLM      FCB     $80             {p1 marcado inicialmente}
         FCB     0               {p2}
         FCB     0               {p3}
         FCB     0               {p4}
         FCB     0               {p5}
         FCB     0               {p6}
         FCB     $80             {Combinatorio, marcado siempre}
*
* Vector Memoria AUXiliar del Marcado
*
MAUXM    FCB     $80             {p1 marcado inicialmente}
         FCB     0               {p2}
         FCB     0               {p3}
```

(*Continúa*)

Tabla 9.5. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando el vector del marcado.

```
        FCB     0               {p4}
        FCB     0               {p5}
        FCB     0               {p6}
        FCB     $80             {Combinatorio, marcado siempre}
*
*                               TaBLa de las Direcciones de la E.D. asociadas a
*                               cada lugar
*
TBDLUG  FDB     DLUG01-1        {p1}
        FDB     DLUG02-1        {p2}
        FDB     DLUG03-1        {p3}
        FDB     DLUG04-1        {p4}
        FDB     DLUG05-1        {p5}
        FDB     DLUG06-1        {p6}
        FDB     0               {Combinatorio}
*
PTMLM   FDB     MLM             Puntero sobre la MLM
DIRCOM  FDB     0               Dirección de comienzo de la transición a tratar
*
DIRLM0  EQU     MLM-1           Dirección del marcado del lugar p1
*
DLUG01  FCB     0               Número de lugares de sincronización de t1
        FDB     PEVAC1          Dirección del «pevac»
        FDB     0               Dirección de la siguiente transición
        FCB     2               Número de lugares de salida
        FDB     DIRLM0 + 2      {p2}
        FDB     DIRLM0 + 4      {p4}
*
DLUG02  FCB     0               Número de lugares de sincronización de t2
        FDB     PEVAC2          Dirección del «pevac»
        FDB     0               Dirección de la siguiente transición
        FCB     1               Número de lugares de salida
        FDB     DIRLM0 + 3      {p3}
*
DLUG03  FCB     $FF             p3 es un lugar de sincronización
*
DLUG04  FCB     0               Número de lugares de sincronización de t3
        FDB     PEVAC3          Dirección del «pevac»
        FCB     1               Número de lugares de salida
        FDB     DIRLM0 + 5      {p5}
```

(*Continúa*)

Tabla 9.5. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando el vector del marcado.

```
*
DLUG05   FCB   1              Número de lugares de sincronización de t4
         FDB   DIRLM0 + 3     {p3}
         FDB   PEVAC4         Dirección del «pevac»
         FDB   DTRAN5 - 1     Dirección de la siguiente transición, t5
         FCB   1              Número de lugares de salida
         FDB   DIRLM0 + 1     {p1}
*
DTRAN5   FCB   0              Número de lugares de sincronización de t5
         FDB   PEVAC5         Dirección del «pevac»
         FDB   0              Dirección de la siguiente transición
         FCB   1              Número de lugares de salida
         FDB   DIRLM0 + 6     {p6}
*
DLUG06   FCB   0              Número de lugares de sincronización de t6
         FDB   PEVAC6         Dirección del «pevac»
         FDB   0              Dirección de la siguiente transición
         FCB   1              Número de lugares de salida
         FDB   DIRLM0 + 5     {p5}
*
         END   CCICLO
```

9.6.3 Codificación de la estructura de una RdP: Lugares representantes y lugares de sincronización

Tal y como se anunció en §9.6.2, la codificación de la estructura de una RdP a partir de sus lugares puede realizarse, en principio, considerando cualquier subconjunto de lugares cuyas transiciones de salida *cubran* al conjunto de transiciones de la red.

Según las denominaciones presentadas en la tabla 7.5, en este problema de cobertura, los *elementos a cubrir* son las transiciones, los *elementos cobertores* son sus lugares de entrada y la *tabla de cobertura* no es ni más ni menos que la matriz de incidencia previa de la RdP. La única cobertura *no redundante* (§7.4.3*c*) de las transiciones de la RdP de la figura 9.1 por sus lugares de entrada es $\{p_1, p_2, p_4, p_5, p_6\}$.

Aunque para la red de la figura 9.1 sólo se ha obtenido una cobertura no redundante, en general se pueden obtener varias, lo cual plantea de forma natural el problema de la obtención de la mejor cobertura. Ahora bien, hablar de «la mejor» cobertura implica la previa definición de objetivos que permiten calificar las diferentes soluciones.

Si se pretende mandar un proceso en tiempo real, el objetivo básico será el elegir, para una RdP dada, la codificación que *minimice la duración* de un paso de simulación. En segundo lugar, se pretenderá también minimizar la ocupación de memoria de la codificación de la RdP.

Sean:

1) $\mathcal{P}_R$ un conjunto de lugares representantes que determine una cobertura no redundante de las transiciones de la red y
2) $\mathcal{P}_S$ el mínimo conjunto de lugares de sincronización tal que $\mathcal{P}_R \cup \mathcal{P}_S = P$ (conjunto de lugares de la RdP).

De acuerdo con las definiciones anteriores, la determinación de un $\mathcal{P}_R$ y del $\mathcal{P}_S$ correspondiente no depende más que de la estructura de la RdP; es decir, no es función del marcado inicial ni de la interpretación que se asocie a la red.

Desde el punto de vista de las prestaciones, la simulación de una RdP según el algoritmo presentado en §9.6.2 (o su versión mejorada de escrutación sobre el subvector de los lugares representantes) no es muy sensible a los diferentes $\mathcal{P}_R$ posibles. No obstante, es fácil intuir que, normalmente, las mejores prestaciones se obtendrán con aquellas coberturas que tengan un menor número de lugares representantes. En estas condiciones, la obtención de un (el) $\mathcal{P}_R$ óptimo es un problema clásico. A continuación se presenta un algoritmo que procede heurísticamente y permite obtener con gran rapidez una solución *subóptima* (con frecuencia la solución obtenida es óptima).

ALGORITMO PARA OBTENER UNA «BUENA» COBERTURA NO REDUNDANTE

Paso 1: *a*) $\mathcal{P}_R$: = Lugares representantes esenciales;
b) $\mathcal{P}_S$: = Lugares de sincronización inducidos por los lugares representantes esenciales;
c) Eliminar de la RdP las transiciones de salida de los lugares representantes esenciales;

Paso 2: **mientras que** no se hayan eliminado todas las transiciones
hacer
a) Elegir un lugar p de tal manera que sea lugar de entrada del mayor número de transiciones;
b) $\mathcal{P}_R := \mathcal{P}_R \cup p$; {añadir p a $\mathcal{P}_R$}
c) $\mathcal{P}_S := \mathcal{P}_S \cup \{{}^{\cdot}\{p^{\cdot}\} - p\}$; {añadir a $\mathcal{P}_S$ los lugares de sincronización inducidos por p};
d) Eliminar las transiciones de salida de p;
fmq;

EJEMPLO 1. Aplicando el algoritmo a la RdP de la figura 9.1 se tiene lo siguiente:

Paso 1: *a*) $\mathcal{P}_R := \{p_1, p_2, p_4, p_5, p_6\}$
b) $\mathcal{P}_S := \{p_3\}$
c) Se eliminan todas las transiciones

Paso 2: Como se han eliminado todas las transiciones, $\mathcal{P}_R$ y $\mathcal{P}_S$ definen una solución (ésta es óptima).

EJEMPLO 2. Si se considera la red de la figura 2.9, se puede obtener lo siguiente:

Paso 1: *a)* $\mathcal{P}_R := \{RL_1, AL_1, RL_2, AL_2, RR_1, AR_1, RR_2, AR_2\}$;
b) $\mathcal{P}_S := \emptyset$;
c) Se eliminan las transiciones:
$\{DL_1, FL_1, DL_2, FL_2, DR_1, FR_1, DR_2, FR_2\}$
Paso 2: primera aplicación:
a) $\mathcal{P}_R := \mathcal{P}_R \cup X_1$;
b) $\mathcal{P}_S := \{EL_1, ER_1, ER_2, X_2\}$;
c) Se eliminan las transiciones $\{CL_1, CR_1, CR_2\}$:
Paso 2: segunda aplicación:
a) $\mathcal{P}_R := \mathcal{P}_R \cup EL_2$;
b) $\mathcal{P}_S := \mathcal{P}_S$;
c) Se elimina la transición CL_2. Se han eliminado todas las transiciones.

La solución obtenida (es óptima) está definida por:

$$\mathcal{P}_R = \{RL_1, AL_1, RL_2, AL_2, RR_1, AR_1, RR_2, AR_2, X_1, EL_2\}$$
$$\mathcal{P}_S = \{EL_1, ER_1, ER_2, X_2\}$$

De las potencialmente múltiples coberturas que minimizan el número de lugares cobertores de una RdP, interesa destacar muy especialmente aquéllas en las que $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$. Es decir, aquellas coberturas en las que los conjuntos de lugares representantes y los de sincronización son *disjuntos*. Una de las principales razones que abogan por el mencionado interés es que los *conflictos efectivos* existentes en la descripción son *resueltos* por el algoritmo de simulación de §9.6.2, dándole prioridad al disparo de la primera transición de salida del lugar que aparezca y sea disparable.

En efecto, a modo de consideraciones previas puede establecerse que:

a) Si $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$, cada lugar representante ($p \in \mathcal{P}_R$) representará a *todas* sus transiciones de salida.
b) Una vez disparada una transición, el algoritmo de simulación no busca la siguiente transición representada por el mismo lugar (etiqueta SIGTRA) sino que busca la lista de transiciones representada por el siguiente lugar representante marcado (etiqueta SIGLUG).

De las dos consideraciones anteriores se deduce que nunca se podrán disparar dos transiciones de salida de un mismo lugar en un único paso de simulación. La primera de las transiciones de salida del lugar representante que sea disparable es la única que se disparará.

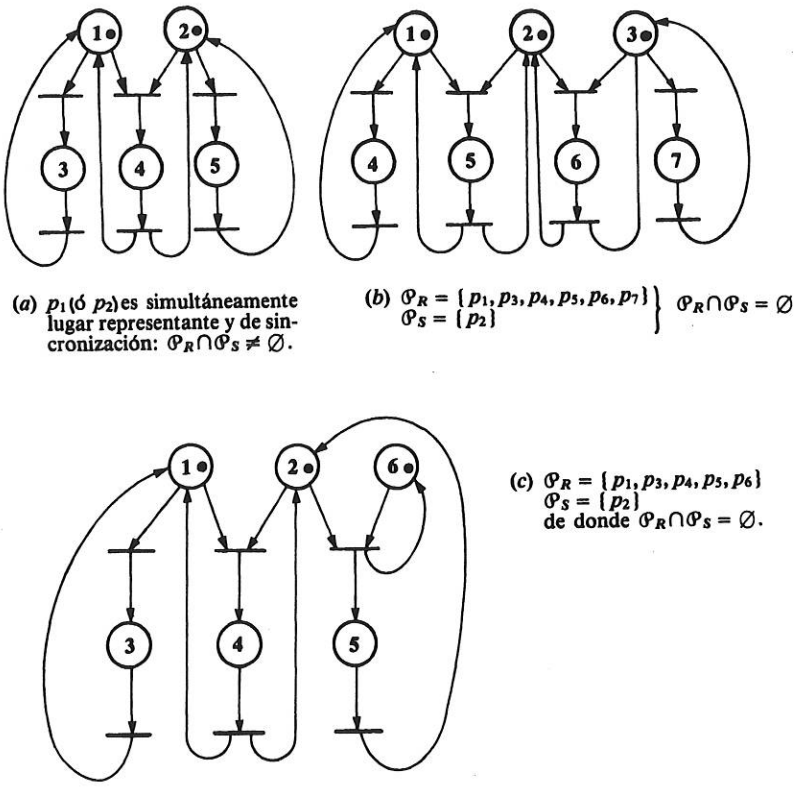
Nota. La prioridad en el disparo de transiciones permite, a veces, simplificar los eventos. De este modo, si dos transiciones de salida de un lugar están etiquetadas A y $\bar{A}B$, bastará con programar el evento A para la primera transición y sólo B para la segunda. Ello redundará, naturalmente, en codificaciones más cortas y simulaciones más rápidas.

En otro orden de ideas, en §9.6.4 se presentarán otras dos razones que testimonian el interés de las coberturas en las que son disjuntos los subconjuntos de lugares representantes y de sincronización. Ambas razones permiten la obtención de esquemas altamente eficientes para la simulación de RdP binarias.

Por último, después de lo reseñado en torno a cualidades de las coberturas tales que $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$, cabe señalar que:

1) En toda RdP *simple*, siempre puede encontrarse una partición. En efecto, si toda transición tiene como máximo un lugar de entrada, p, compartido con otras transiciones, la adición del mencionado lugar a $\mathcal{P}_R$ permite tener $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$. Además, la elección considerada posibilita la obtención de una cobertura con el mínimo número de lugares representantes.
2) La estructura de la RdP puede conducir, en ciertos casos, a una situación tal que $\mathcal{P}_R \cap \mathcal{P}_S \neq \emptyset$ (véase la figura 9.10*a*).

El análisis de las RdP que modelan sistemas permite afirmar que, normalmente, $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$. Ahora bien, si para una red dada, la condición anterior no se verifica, siempre existen *redes equivalentes* para las que ésta se cumple. Las nuevas redes se obtienen simplemente añadiendo *lugares implícitos* a la red de partida. Así, la red de la figura 9.10*c* es equivalente a la de la figura 9.10*a* (se le ha añadido el lugar implícito p_6 para representar su transición de entrada y de salida) y sin embargo $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$.



(*a*) p_1 (ó p_2) es simultáneamente lugar representante y de sincronización: $\mathcal{P}_R \cap \mathcal{P}_S \neq \emptyset$.

(*b*) $\mathcal{P}_R = \{p_1, p_3, p_4, p_5, p_6, p_7\}$, $\mathcal{P}_S = \{p_2\}$ } $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$

(*c*) $\mathcal{P}_R = \{p_1, p_3, p_4, p_5, p_6\}$, $\mathcal{P}_S = \{p_2\}$, de donde $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$.

Figura 9.10. Sobre la disjuntividad de $\mathcal{P}_R$ y $\mathcal{P}_S$ en redes no simples.

EJERCICIO. Demuéstrese que si $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$, la cobertura no puede ser redundante.

EJERCICIO. Propóngase un algoritmo para obtener $\mathcal{P}_R$ y $\mathcal{P}_S$ de modo que las coberturas que se obtengan sean tales que $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$ (partición). (*Indicación*. Téngase en cuenta que en determinados casos habrá que añadir lugares implícitos.)

9.6.4 Simulación de RdP por escrutación de la lista de lugares representantes marcados

Hasta ahora se ha venido adoptando una representación vectorial del marcado de una RdP binaria y se ha sugerido que cuando se desee la máxima rapidez en la simulación, conviene limitar la escrutación del marcado al subvector de los lugares representantes (a partir de los que se codifica la estructura de la red). Por otro lado, se ha establecido que si el porcentaje medio de lugares marcados de una red es pequeño, el proceso de escrutación, aun limitado al subvector de los lugares representantes, puede ser relativamente ineficiente.

Para mejorar sensiblemente las prestaciones, es posible adoptar una nueva *representación del marcado* compuesta por:

a) Una lista de los lugares representantes marcados.
b) Un vector del marcado de los lugares de sincronización.

El interés de realizar mediante una lista el subvector del marcado de los lugares representantes es notorio puesto que, al proceder de esta forma, se pueden obtener directamente las δp_i, direcciones donde se encuentran las informaciones relativas a las transiciones representadas por p_i. Es decir, desaparece la operación de escrutación del vector booleano del marcado que es tanto más ineficiente cuanto menor sea la proporción de lugares representantes marcados. El mantenimiento de la representación vectorial del marcado de los lugares de sincronización pretende que se facilite al máximo el cálculo de las condiciones de sensibilización de las transiciones, $\sum_j$.

Las prestaciones del nuevo esquema de simulación que se diseña podrán ser óptimas (cuasi-óptimas) si los lugares representantes y los de sincronización determinan subconjuntos *disjuntos*; es decir, $\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$. En efecto, si p_k fuese simultáneamente representante y de sincronización ($p_k \in \mathcal{P}_R \cap \mathcal{P}_S$), poseería una doble representación en memoria, lo que supondría:

a) Una mayor ocupación de memoria.
b) Una simulación más lenta, al tenerse que gestionar su doble representación en memoria.

En lo sucesivo, se supondrán *disjuntos* los subconjuntos de lugares representantes y de sincronización elegidos para representar una RdP.

A la hora de realizar una simulación síncrona, se pueden adoptar diversas codificaciones de la información auxiliar sobre la evolución del marcado. Dado que insertar y eliminar elementos de una lista en posiciones arbitrarias son operaciones relativamente costosas en tiempo, se utilizará la siguiente estrategia durante un paso de simulación:

1) Construir íntegra y directamente la lista de lugares representantes que estarán marcados al comenzar el paso siguiente.
2) Modificar el vector del marcado de los lugares de sincronización hasta obtener el nuevo vector.

De acuerdo con lo expresado y utilizando *pilas* (tipo especial de lista directamente soportado en la mayoría de los microcomputadores comercializados, gracias a la presencia de *punteros de pila*) la figura 9.11 representa globalmente el marcado de la RdP de la figura 9.1 cuando se ha disparado la transición t_1. La pila que contiene los lugares representantes marcados se denomina *de tratamiento* (PTRAT, su puntero es PPTRAT), mientras que la pila que se va llenando con los lugares representantes que estarán marcados al comenzar el ciclo siguiente se denomina *en formación* (PFORM, su puntero es PPFORM). La actualización final del marcado se reducirá a:

1) Copiar sobre el vector del marcado de los lugares de sincronización el nuevo vector; es decir, $M[p_3] := M^+[p_3]$.
2) Tomar como nueva pila de tratamiento la pila en formación (PPTRAT : = : = PPFORM) y reinicializar (vaciar) la pila en formación. La reinicialización de la pila en formación se realizará a partir de la dirección DCI (dirección para ciclo impar) o DCP (dirección para ciclo par) según la paridad del ciclo de simulación. De este modo, lo que se pretende es simplemente reconstruir una pila en formación sin alterar el contenido de la construida anteriormente (en este momento pila de tratamiento).

En resumen:

```
PPTRAT : = PPFORM;
C : = C̄; {cambio de la paridad del ciclo}
si C = 0 entonces PPFORM: = DCI sino PPFORM : = DCP fsi;
```

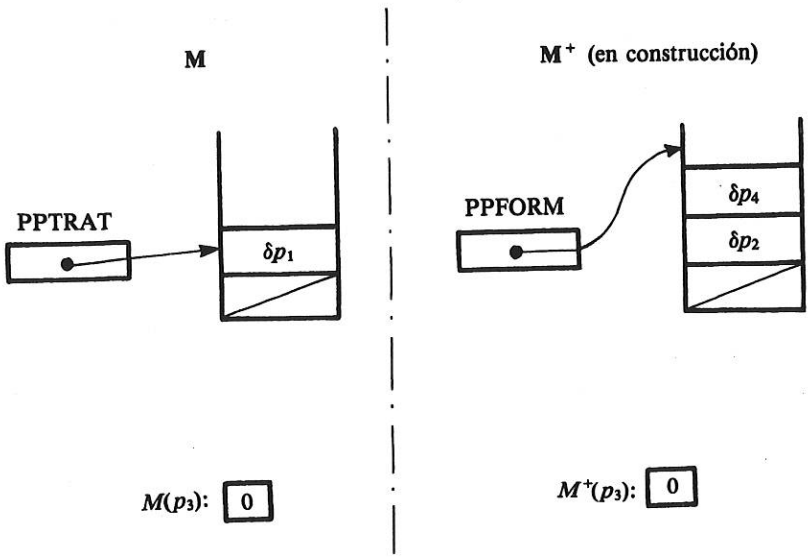


Figura 9.11. Representación del marcado de una RdP (figura 9.1) para una simulación síncrona utilizando pilas y vectores. En M, estaba marcado el lugar representante p_1; en M^+ están marcados los lugares representantes p_2 y p_4 (M^+ se alcanza a partir de M al disparar t_1). (*Nota.* El puntero de pila está listo para escribir, no para leer.)

Observación. Puesto que la pila de tratamiento ha sido vaciada al terminar un ciclo, la operación que se describe en el punto 2 es equivalente a la de intercambiar los punteros de ambas pilas, seguido de apilar en PFORM el valor nulo (fin de pila en formación):

```
AUXPUN : = PPTRAT;
PPTRAT : = PPFORM:
PPFORM : = AUXPUN;
Apilar en PFORM nulo; {realizable, en este caso, mediante:
                       PPFORM : = PPFORM – 2}
```

Antes de continuar, interesa resaltar una nueva e importante razón por la que conviene que se codifiquen las RdP a partir de conjuntos representantes y de sincronización que sean *disjuntos*. En efecto, sea de nuevo p_k un lugar representante y de sincronización ($p_k \in \mathcal{P}_R \cap \mathcal{P}_S$) marcado. Si no se dispara ninguna de las transiciones representadas por p_k, no se podrá concluir de forma inmediata si p_k estará marcado o no en el ciclo siguiente (ya que puede ser disparada alguna de sus transiciones de salida de las que no es representante). Por lo tanto, no se puede apilar directamente en la pila en formación el puntero asociado a p_k, debiéndose realizar un tratamiento más complejo, que reduce notoriamente las prestaciones temporales.

Ahora bien, si a partir de un lugar representante se codifican todas sus transiciones de salida, y si no se dispara ninguna de las transiciones que representa, éste podrá ser marcado inmediatamente (añadido a la pila en formación, PFORM).

La figura 9.12 ilustra la estructura de la RdP de la figura 9.1 adoptando una codificación diferente para las informaciones relativas a los lugares de sincronización y representantes. A partir de cada lugar representante se codifican todas sus transiciones de salida. La información relativa a cada transición adopta la estructura siguiente:

1) Lista de lugares de sincronización para t_j.
2) Puntero al procedimiento evento-acción asociado a t_j.
3) Dirección de la siguiente transición representada por el mismo lugar. Si no existe otra, el puntero tomará el valor nulo.
4) Lista de los lugares de salida de t_j que son representantes.
5) Lista de los lugares de salida de t_j que son de sincronización.

La comparación de las figuras 9.9 y 9.12 permite comprobar que la segunda codificación es algo más costosa en memoria puesto que los lugares de salida de cada transición se representan mediante dos listas:

a) lista de los lugares representantes.
b) lista de los lugares de sincronización.

El incremento de ocupación de memoria indicado será (podrá ser), para la codificación de la estructura de una RdP, de un octeto por transición. La menor ocupación de memoria de la nueva representación del *marcado* compensará, normalmente, en parte el incremento de ocupación de la codificación de la *estructura* de la RdP.

Codificada una RdP mediante una estructura de datos como la enunciada, un esquema de algoritmo de simulación de RdP puede adoptar la forma siguiente:

```
ACTMAR: Desmarcar los lugares de sincronización de entrada a tj;
        Marcar los lugares representantes de salida de tj;
        Marcar los lugares de sincronización de salida de tj;
```

```
SIGLUG: Desapilar la dirección asociada al siguiente lugar representante
        marcado de PTRAT, δpi;
        si fin de pila {se trataron todos los lugares representantes mar-
                        cados}
FCICLO:   entonces PPTRAT : = PPFORM;
                   C : = C̄;
                   si C = 0
                     entonces PPFORM : = DCI
                     si no    PPFORM : = DCP
                   fsi;
                   Msinc : = M+sinc; {actualizar el subvector
                                      de sincronización}
CCICLO:            Muestrear las entradas;
                   ir a SIGLUG
          si no    tj : = primera transición representada por pi;
        fsi;
TRATRA:
SENSIB: si pi es lugar representante esencial de tj
          entonces ir a pevac-j;
          si no
            Calcular la sensibilización de tj, ∑j;
            si ∑j = 1 entonces ir a pevac-j fsi;
        fsi;
SIGTRA: tj : = siguiente transición representada por pi;
        si tj = nulo entonces apilar pi en PFORM;
                              ir a SIGLUG;
                   si no      ir a TRATRA
        fsi;
```

La comparación de este algoritmo con el presentado en §9.6.2 permite observar que:

1) Todo lugar representante marcado ha de reapilarse en PFORM si no se dispara ninguna de sus transiciones de salida.
2) Al actualizar un marcado sólo se han de desmarcar los lugares de sincronización. El lugar representante se desmarcará debido a que no se apilará su dirección en PFORM.

En resumen, el algoritmo propuesto (véase en la tabla 9.6 una codificación en lenguaje ensamblador) no resulta ser más que una generalización del segundo considerado en §9.6.1 para los grafos reducidos. Las diferencias principales entre ambos provienen de los hechos siguientes:

1) En un GR existe un único lugar representante marcado y no existen lugares de sincronización. En una RdP existen lugares de sincronización y un número variable de lugares representantes marcados.
2) En un GR todos los lugares son representantes esenciales, por lo que es innecesario el cálculo de las condiciones de sensibilización de las transiciones.

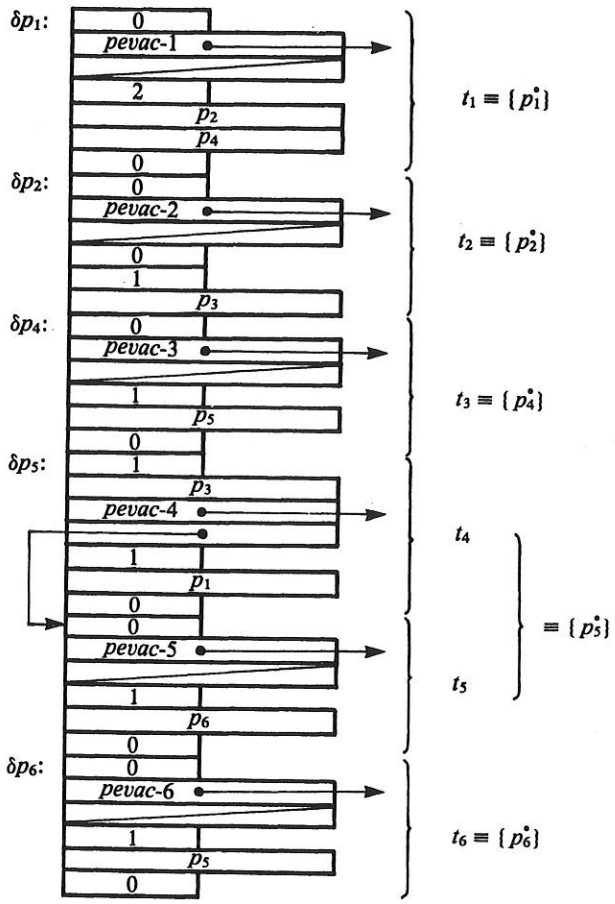


Figura 9.12. Codificación de la RdP de la figura 9.1 en la que se diferencian los lugares representantes y los de sincronización.

9.6.5 Comentarios generales

La simulación dirigida por el marcado ha sido abordada a lo largo de este apartado insistiendo en dos tipos de esquemas básicos. Éstos son los basados en la simulación a partir de la escrutación del vector de marcado o de la lista de lugares representantes marcados. Previamente, en §9.6.1, han sido introducidos algoritmos para el caso particular en que se simulen grafos reducidos.

Desde el punto de vista de la ocupación de memoria es fácil evaluar para cada RdP el consumo que genera cada esquema de simulación. En cualquier caso, las diferencias totales (codificación de la estructura de la red más la de su marcado) nunca serán muy significativas.

En lo que concierne a la duración de un paso de simulación, las diferencias dependerán de la RdP que se considere, pero su valor puede ser relativamente importante.

De este modo, aún sin pretender establecer conclusiones generales, puede decirse que, suponiendo nula la duración de los procedimientos evento-acción, para una red con $n \cong m \cong 32$, y unos 3 o 4 lugares marcados, la simulación dirigida a partir de pilas (§9.6.4) será, aproximadamente, el *doble de rápida* que la simulación dirigida a partir de vectores (§9.6.2) (véase [SILV 82*b*]). Para completar este primer cuadro comparativo, es importante señalar que la simulación matricial (§9.5) será unas *cinco veces más lenta* que la dirigida a partir de pilas. Evidentemente, la consideración real de la duración de ejecución de los procedimientos evento-acción conducirá a resultados globales más uniformes.

Tabla 9.6. Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando la lista de lugares representantes marcados.

```
*
* Programa de simulación del interpretador síncrono de RdP utilizando dos pilas (PTRAT
* y PFORM) para el tratamiento de los lugares representantes marcados, y dos vectores
* booleanos (MLS y AUXMLS) para los lugares de sincronización. PTRAT contiene los
* lugares representantes marcados en el ciclo de tratamiento considerado y PFORM los
* que estarán marcados en el ciclo siguiente. Al finalizar cada ciclo de tratamiento, se in-
* tercambian las dos pilas y AUXMLS es copiado sobre MLS.
*
* Programa interpretador. Su ejecución comienza en CCICLO
*
TOTLUG   EQU   7            n( = 6 lugares de la RdP de la figura 9.1) + 1
*                            (Combinatorio General)
LUGSIN   EQU   1            número de lugares de sincronización
*
* ACTualización del MARcado al ser disparada una transición
*
ACTMAR   LDS   DIRCOM
         PULB               Lectura del número de lugares de sincronización
         TSTB
         BEQ   ACTREP       Si es = 0, actualizar marcado de lugares de salida
*
* DesMarCar los lugares de SINcronización de la transición disparada
*
         CLRA               Cargar el acumulador A con valor lógico «0»
DMCSIN   PULX               Apuntar al lugar correspondiente en MLM
         STAA  LUGSIN,X     Desmarcar dicho lugar
         DECB
         BNE   DMCSIN       hasta haber desmarcado todos
*
* ACTualización de los lugares REPresentantes de salida
*
```

(*Continúa*)

Tabla 9.6. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando la lista de lugares representantes marcados.

```
*
ACTREP   PULX                  Saltarse la dirección del «pevac» asociado
         PULX                  Saltarse la dirección de la siguiente transición
         PULB                  Lectura del número de lugares representantes de
*                              salida
         TSTB
         BEQ     ACTLSS        Si es = 0, marcar los lugares de sincronización
*
* MARcado de lugares REPresentantes de salida de la transición disparada
*
         LDX     PPFORM        Inicializar puntero sobre PFORM
MARREP   PULA                  Lectura de la dirección de un lugar representante
         DEX
         STAA    0,X           Apilarla en PFORM
         PULA
         STAA    1,X
         DEX                   Actualizar puntero sobre PFORM
         DECB
         BNE     MARREP
         STX     PPFORM
*
* ACTualizar los Lugares de Salida de Sincronización
*
ACTLSS   PULB                  Lectura del número de lugares de salida de sincro-
*                              nización
         TSTB
         BEQ     SIGLUG        Si es = 0, buscar siguiente lugar marcado
         LDAA    #1000000B     Cargar el acumulador A con valor lógico «1»
*
* MARcado lugares de SINcronización de salida de la transición disparada
*
MARSIN   PULX                  Lectura de la dirección del lugar a marcar
         STAA    LUGSIN,X      Marcar dicho lugar
         DECB
         BNE     MARSIN        hasta haber marcado todos
*
* Búsqueda del SIGuiente LUGar marcado
*
```

(*Continúa*)

Tabla 9.6. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando la lista de lugares representantes marcados.

```
SIGLUG    LDS    PPTRAT        Lectura del puntero sobre PTRAT
          PULX                 Desapilar un lugar marcado
          STS    PPTRAT        Salvaguardar el puntero sobre PTRAT
          TXS
          BNE    TRATRA        Si no es el último (Combinatorio), tratar el lugar
*
* Fin de un CICLO de tratamiento: Intercambio de las pilas PTRAT y PFORM
*
FCICLO    LDX    PPFORM
          STX    PPTRAT
          LDAA   FPPAR!.$FF
          COM    CICLO
          BMI    GUARDA
          LDAA   #FPIMP!.$FF
GUARDA    STAA   PPFORM + 1
*
* MLS:= AUXMLS
*
          LDS #MLS + LUGSIN - 1      El reg. SP apunta a AUXMLS
          LDX    #MLS          El reg. X apunta a MLS
          LDAB   #LUGSIN       Inicializar contador de lugares de sincronización
COPLS     PULA                 Lectura del marcado de un lugar
          STAA   0,X           Salvaguarda del marcado
          DEX
          DECB
          BNE    COPLS         Hasta copiar todos los lugares de sincronización
*
* Comienzo de un nuevo CICLO de tratamiento
*
CCICLO    LDAA   VMIS          Muestreo de las entradas y Volcado de la Memo-
*                              ria. Imagen de las salidas. (La instrucción genera
*                              estas señales por hardware: detección de un ac-
*                              ceso a la posición de memoria VMIS.)
*
          BRA    SIGLUG        Búsqueda del primer lugar marcado
*
*
* TRAtamiento asociado a una TRAnsición descendiente del lugar marcado
```

(*Continúa*)

Tabla 9.6. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando la lista de lugares representantes marcados.

```
*
TRATRA   STS    DIRCOM      Guardar la direc. de comienzo
         PULB                Lectura del número de lugares de entrada – 1
         TSTB
SENSIB   BEQ    IRPEVA      Si es = 0, la transición está sensibilizada
*
* TeST de SENsibilización de las transiciones descendientes del lugar marcado
*
TSTSEN   LDAA   #10000000B  Iniciar el acumulador A con valor lógico «1»
BUCLE    PULX               Apuntar con reg. X a un lugar de sincr., pi
         ANDA   0,X         A ← (A) · M [pi]
         DECB
         BNE    BUCLE       Hasta haber recorrido todos lugares de sincroni-
*                           zación
         TSTA
         BPL    NODISP      Si el producto lógico es = 0, no está sensibilizada
*
* Saltar a ejecutar el «pevac» asociado a la transición
*
IRPEVA   PULX               Lectura de la dirección de salto
         JMP    0,X
*
* Cálculo de la SIGuiente TRAnsición a tratar
*
NODISP   PULX               Saltarse la dirección del «pevac» asociado
SIGTRA   TSX
         LDS    0,X         Lectura de la dir. de la sig. transición descen.
         BNE    TRATRA      Si queda alguna transición descendiente, tratarla
         LDX    PPTRAT      Si no, el lugar no se desmarca
         DEX
         LDX    0,X         Lectura del lugar representante marcado
         LDS    PPFORM
         PSHX               Apilarlo en PFORM
         STS    PPFORM
         JMP    SIGLUG      Buscar un nuevo lugar representante marcado
*
* ESTRUCTURA DE DATOS ASOCIADA A LA RdP DE LA FIGURA 9.1
*
```

(*Continúa*)

Tabla 9.6. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando la lista de lugares representantes marcados.

```
        ORG     (*+$FF)!.$FF00  Se comenzará página
*
MLS     FCB     0               Memoria de Lugares de Sincronización {p3}
AUXMLS  FCB     0               AUXiliar de la Memoria de Lugares de Sincroniza-
*                               ción (debe seguir a MLS)
*
PPTRAT  FDB     FPIMP-2         Puntero Pila de TRATamiento (p1 marcado inicial)
*
PPFORM  FDB     FPPAR           Puntero Pila de FORMación
DIRCOM  FDB     0
*
DLUG01  FCB     0               Número de lugares de sincronización de t1
        FDB     PEVAC1          Dirección del «pevac»
        FDB     0               Dirección de la siguiente transición
        FCB     2               Número de lugares representantes de salida
        FDB     DLUG02          {p2}
        FDB     DLUG04          {p4}
        FCB     0               Número de lugares de sincr. de salida de t1
*
DLUG02  FCB     0               Número de lugares de sincronización de t2
        FDB     PEVAC2          Dirección del «pevac»
        FDB     0               Dirección de la siguiente transición
        FCB     0               Número de lugares representantes de salida
        FCB     1               Número de lugares de sincr. de salida de t2
        FDB     MLS             {p3}
*
DLUG04  FCB     0               Número de lugares de sincronización de t3
        FDB     PEVAC3          Dirección del «pevac»
        FDB     0               Dirección de la siguiente transición
        FCB     1               Número de lugares representantes de salida
        FDB     DLUG05          {p5}
        FCB     0               Número de lugares de sincr. de salida de t3
*
DLUG05  FCB     1               Número de lugares de sincronizacón de t4
        FDB     MLS             {p3}
        FDB     PEVAC4          Dirección del «pevac»
        FDB     DTRAN5-1        Dirección de la siguiente transición – 1
        FCB     1               Número de lugares representantes de salida
```

(*Continúa*)

Tabla 9.6. (*Cont.*) Codificación en lenguaje ensamblador MOTOROLA 6801 de un simulador de RdP binarias que procede escrutando la lista de lugares representantes marcados.

```
            FDB     DLUG01      {p1}
            FCB     0           Número de lugares de sincr. de salida de t4
*
DTRAN5      FCB     0           Número de lugares de sincronización de t5
            FDB     PEVAC5      Dirección del «pevac»
            FDB     0           Dirección de la siguiente transición
            FCB     1           Número de lugares representantes de salida
*
            FDB     DLUG06-1    {p6}
            FCB     0           Número de lugares de sincr. de salida de t5
DLUG06      FCB     0           Número de lugares de sincronización de t6
            FDB     PEVAC6      Dirección del «pevac»
            FDB     0           Dirección de la siguiente transición
            FCB     1           Número de lugares representantes de salida
            FDB     DLUG05-1    {p5}
            FCB     0           Número de lugares de sincr. de salida t6
*
* Definición de las pilas de tratamiento y formación (PTRAT y PFORM)
* Deben estar en la misma página
*
            RMB     10          Reservar memoria para la pila par
FPPAR       EQU     * - 1
            FDB     0           Base de pila par: Combinatorio General
*
            RMB     8           Reservar memoria para la pila impar
            FDB     DLUG01      Está marcado el lugar p1
FPIMP       EQU     * - 1
            FDB     0           Base de pila impar: Combinatorio General
*
CICLO       FCB     $FF         Por defecto se comienza con ciclo impar
*
            END     CCICLO
```

A modo de un último comentario general, es importante señalar que las mejores prestaciones temporales del método basado en pilas (§9.6.4) se pagan con unas codificaciones de la estructura y el marcado de la red menos uniformes (existen dos tipos de lugares: representantes y de sincronización), lo que supone un mayor coste de traducción desde la especificación (§9.2). En este punto conviene recordar que la partición de los lugares de una RdP en representantes y de sincronización ($\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$) permite no sólo utilizar eficientemente el esquema de simulación

por escrutación de la lista de los lugares representantes marcados (§9.6.4), sino también resolver los conflictos durante la simulación, de forma sistemática (§9.6.3).

9.7 REPRESENTACIONES BASADAS EN LISTAS (III): SIMULACIÓN DIRIGIDA POR LAS TRANSICIONES SENSIBILIZADAS

La aproximación seguida a lo largo de §9.6 constituye una generalización de los métodos usuales en la simulación de grafos reducidos (§9.6.1). Ahora bien, al menos desde un punto de vista meramente conceptual, cabe esperar que pueda realizarse una más rápida simulación de las RdP a partir del *subconjunto de transiciones sensibilizadas* (es decir, en la simulación se considerará una transición si y sólo si está sensibilizada). A continuación se introduce un método de simulación válido para RdP *binarias* y se comentan sus limitaciones.

Sea VECSEN un vector de enteros denominado de *sensibilización* en el que cada componente memoriza el número de lugares de entrada no marcados de una transición. La i-ésima transición estará sensibilizada si y sólo si VECSEN $[i] = 0$. Obsérvese que VECSEN es una representación *indirecta*, y posiblemente *parcial*, del marcado de la red de Petri. En efecto, VECSEN $= (C^-)^T \cdot \bar{M}$, y puede que no exista una matriz C_g, inversa generalizada, tal que $C_g \cdot (C^-)^T = I_m$.

La escrutación de VECSEN permite acceder al conjunto de los procedimientos evento-acción asociados a las diferentes transiciones. Prescindiendo del cáracter síncrono que se desea para la simulación, el disparo de una transición deberá *decrementar* los elementos de VECSEN asociados a las transiciones de salida de los lugares de salida de la transición disparada. De forma análoga, el disparo de una transición debe *incrementar* los elementos de VECSEN asociados a las transiciones de salida de sus lugares de entrada. Con el fin de facilitar la simulación de una RdP se adoptará, para cada transición, la codificación que sigue:

1) Puntero al procedimiento evento-acción asociado, *pevac-i*.
2) Número de lugares de entrada de t_j, $|{}^\bullet t_j|$.
3) Lista de las transiciones de salida de los lugares de entrada de t_j, menos t_j: L_e
4) Lista de las transiciones de salida de los lugares de salida de t_j: L_s

La figura 9.13 representa, de acuerdo con el esquema anterior, la RdP de la figura 9.1. Un algoritmo de simulación que proceda escrutando VECSEN puede adoptar la estructura siguiente (la ejecución de un nuevo ciclo comienza en CCICLO):

```
ACTMAR: VECSEN[i] : = |•tj|; {desensibilización de la transición
                              disparada}
        Incrementar los elementos de VECSEN asociados a las
          transiciones de la lista Le;
        Decrementar los elementos de VECSEN asociados a las
          transiciones de la lista Ls;
SIGTRA: repetir i : = i + 1 hasta que VECSEN [i] = 0;
        si TBLTRA [i] = nulo
CCICLO:   entonces i : = 0;
                    Muestrear las entradas;
                    ir a SIGTRA
          si no     ir a pevac-i
        fsi;
```

Admitiendo una duración nula para la ejecución de los diferentes procedimientos evento-acción, el algoritmo de simulación considerado, para una red con $n = m = 32$ y 3 o 4 lugares marcados simultáneamente, es un 30-40% más lento que el algoritmo expuesto en §9.6.5 (escrutación de pilas). La transformación de este esquema de simulación en otro en el que no se escrute un vector de sensibilización sino una lista (pila) de transiciones sensibilizadas mejora las prestaciones temporales en un 25-35%, aproximadamente. No obstante, el método más rápido suele seguir siendo el expuesto en §9.6.5.

Nota. La relativa ineficiencia del método de simulación basado en una *lista de transiciones* sensibilizadas en vez de un vector se debe a la complejidad en la manipulación de la lista que se obtenga de VECSEN (no debe olvidarse que aún para RdP binarias VECSEN es un vector de enteros, no de booleanos). En [CHOC 80*a*] se presenta una aproximación mixta con vector y lista.

Otros *inconvenientes* de los esquemas de simulación dirigidos por las transiciones sensibilizadas son:

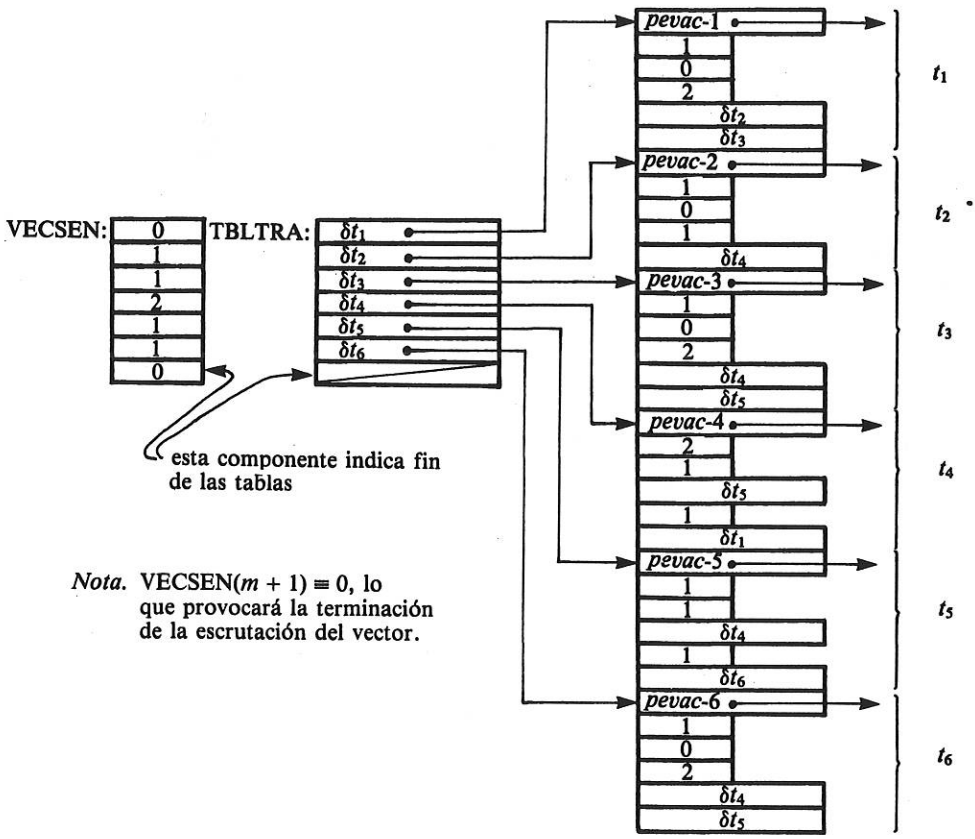


Figura 9.13. Simulación dirigida por transiciones sensibilizadas. Una representación de la estructura y el marcado de la RdP de la figura 9.1. (La simulación no podrá ser síncrona debido a que existe un único vector que indirectamente representa el marcado.)

1) Su difícil o ineficiente extensión al caso de RdP no binarias.
2) La imposibilidad directa de realizar acciones condicionales, pues no se conoce el marcado (sólo a través de VECSEN se dispone de información acerca del mismo, pero su caracterización puede ser incompleta).

Estos inconvenientes, unido a que las prestaciones que se obtienen no son normalmente superiores a las que se alcanzan en los esquemas de simulación dirigidos por el marcado, hace que no se continúe el estudio de algoritmos en los que la simulación se hace a partir de las transiciones sensibilizadas. No obstante, justo es de destacar que la simulación de *grafos marcados* puede ser más eficiente con métodos de simulación basados en listas de transiciones sensibilizadas, si no existen acciones condicionales.

EJERCICIO. Propóngase un algoritmo y una estructura de datos adecuados a la simulación de RdP binarias a partir de una lista de transiciones sensibilizadas.

9.8 UN AUTÓMATA PROGRAMABLE CON LENGUAJE ESPECIAL

9.8.1 Generalidades

La exposición del sistema simulador de RdP binarias que se considera en este apartado pretende presentar una realización que parta de unos supuestos diametralmente opuestos a los adoptados en las anteriores realizaciones (§9.4, §9.5, §9.6 y §9.7). De este modo, el sistema se caracteriza por:

1) Disponer de un *lenguaje de bajo nivel específico*. (La simulación será *conducida por programa*, no por tabla, véase §9.2.2).
2) Permitir la asociación de *acciones condicionales* a los lugares. (Hasta ahora sólo se admitían acciones incondicionales.)
3) Codificar *conjuntamente* la estructura y la interpretación (eventos y acciones) de la RdP. (Hasta ahora estos aspectos se separaban en una estructura de datos y los procedimientos evento-acción, respectivamente.)
4) Emplear una *memoria imagen de las salidas que sea borrada* sistemáticamente al comenzar un paso de simulación, MIB (§9.3.2). (Hasta ahora se consideraba una emisión directa de las salidas.)

Además de las cuatro características anteriores, el sistema:

1) Realiza una interpretación *síncrona* de la RdP.
2) Emplea de forma eficiente un *microcomputador de propósito general* (tipo MOTOROLA 6801/6802/6809, ZILOG 80, INTEL 8088/8071, etc).

Dadas sus excelentes prestaciones, el sistema que se considera se basa en el esquema de *simulación dirigida por la escrutación de una lista de lugares representantes marcados* (§9.6.4), adoptando una *partición* entre lugares representantes y de sincronización ($\mathcal{P}_R \cap \mathcal{P}_S = \emptyset$).

9.8.2 Estructura de un programa

Un programa se compone de tantas *frases* o *segmentos de código* como lugares representantes tenga la RdP, más una que se le asocia al conjunto de *funciones combinacio-*

nales que se deseen realizar. Dentro de esta frase especial, que se denomina *combinatorio general*, se incluye la definición de acciones asociadas a los lugares de sincronización.

Cada frase asociada a un lugar representante terminará con una instrucción «SP p_i», la cual no es más que un *separador* o *delimitador* de código parametrizado por el lugar al que corresponde, p_i. El combinatorio general estará prologado por la instrucción PCOMB (*principio del combinatorio*) y terminará con FCICLO (instrucción que indica el *final de un ciclo* de simulación).

Las frases asociadas a lugares representantes comprenderán dos partes netamente diferenciadas en las que se codificarán:

1) Las *transiciones de salida* del lugar representante (para cada una se incluirá el cálculo de la condición de sensibilización así como las acciones que hayan de emprenderse debido al posible disparo).
2) Las *acciones* asociadas al lugar. (Esta subfrase terminará con «SP p_i».)

La figura 9.14 ilustra la representación del marcado, estructura e interpretación de la RdP de la figura 9.1, suponiendo condicionadas las acciones a_5(por $\bar{B}$) y a_4 (por $E\bar{D}$). El programa expuesto está redactado en el lenguaje definido en el siguiente apartado. El marcado que se ilustra corresponde al caso en que la transición t_1 acaba de ser disparada (p_1 es el único lugar marcado en la pila de tratamiento, y el par $p_2 - p_4$ son los lugares que estarán marcados al comenzar el siguiente ciclo; se encuentran en la pila en formación).

9.8.3 Presentación del lenguaje de programación y funcionamiento global del sistema

9.8.3.1 *Funciones combinacionales*

Para programar funciones combinacionales se ha adoptado la técnica de emplear *saltos condicionales*, puesto que de este modo se pueden obtener ejecuciones muy rápidas. En efecto:

a) Si la función es de n variables, siempre se puede construir un programa que conduzca a una ejecución con n saltos condicionales como máximo.
b) El salto condicional es una instrucción que existe en todos los computadores de propósito general.

Las instrucciones necesarias para programar funciones lógicas combinacionales son (véase tabla 9.7):

1) SCV v, δ; {saltar si verdad: **si** v **entonces ir a** δ}
2) SCF v, δ; {saltar si falso: **si** $\bar{v}$ **entonces ir a** δ}
3) SLI δ; {saltar incondicionalmente: **ir a** δ}
4) ACV v; {activar durante el ciclo siguiente la variable v: $v := 1$}

Debido al sistemático borrado de la memoria imagen de las salidas, al comenzar un ciclo de simulación, no es necesario disponer de una instrucción de desactivación. De este modo, la expresión $R := A + B\bar{C}$ se programará simplemente:

```
    SCV A, δ1
    SCF B, δ2
    SCV C, δ2
δ1: ACV R      {la activación de R sólo durante el próximo ciclo}
δ2: ---------
```

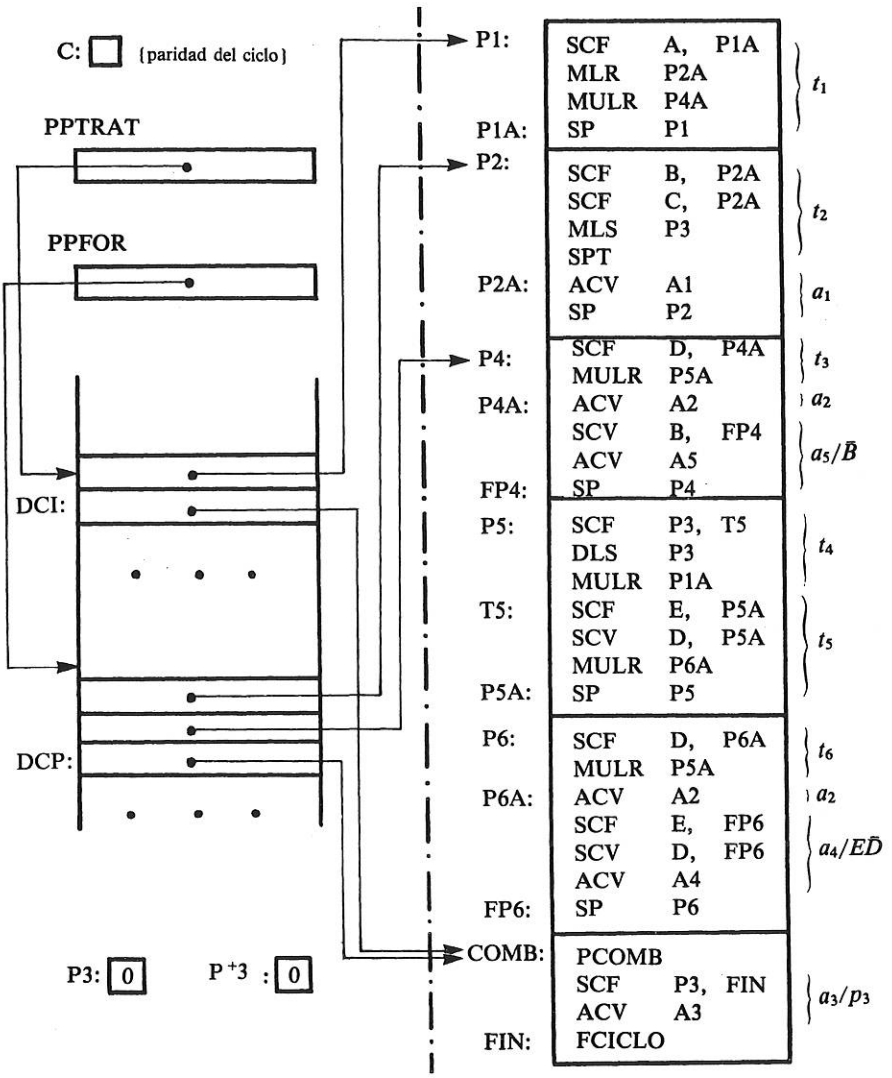


Figura 9.14. Codificación de la RdP de la figura 9.1, suponiendo condicionadas a_4 y a_5.

9.8.3.2 *Instrucciones especiales para la programación de RdP*

Dentro de este grupo de instrucciones se encuentran los *separadores*, así como las instrucciones para el *marcado* o *desmarcado* de lugares.

Las instrucciones del primer grupo que ya fueron introducidas anteriormente son:

5) SP p_i {separador de la frase asociada a p_i}
6) PCOMB {principio del combinatorio}
7) FCICLO {fin de un ciclo de interpretación}

SCV v, δ	**si** $v = 1$ **entonces ir a** δ **fsi;**
SCF v, δ	**si** $v = 0$ **entonces ir a** δ **fsi;**
SLI δ	**ir a** δ;
ACV v	v: = 1; {la activación es efectiva sólo durante el próximo ciclo}
SP p_i	**apilar** p_i en PFORM; **retornar**; {considerar siguiente lugar representante marcado}
PCOMB	M_{SINC}: = M^{+}_{SINC};
FCICLO	PPTRAT: = PPFORM; C: = $\bar{C}$; {complementar el bit C} **si** $C = 0$ **entonces** PPFORM: = DCI **si no** PPFORM: = DCP **fsi;** muestrear entradas y emitir salidas; borrar la memoria imagen de las salidas; **retornar**; {considerar el primer lugar representante marcado}
SPT	**retornar**; {considerar siguiente lugar representante marcado}
MLS k	$M^{+}_{SINC}[k]$: = 1; {k representa al k-ésimo lugar de sincronización}
DLS k	M^{+}_{SINC} [k]: = 0; {k representa al k-ésimo lugar de sincronización}
MLR p_{ia}	**ir a** subprograma p_{ia}; {apilar la dirección en PTRAT; **ir a** p_{ia}}
MULR p_{ia}	**ir a** p_{ia};

Tabla 9.7. Instrucciones de un lenguaje bajo nivel adaptado a la programación de RdP.

La tabla 9.7 define funcionalmente la instrucción «SP p_i», suponiendo que la *pila de tratamiento*, PTRAT, es la pila hardware del microprocesador. Dicho de otro modo, el puntero de la pila PTRAT, PPTRAT, es el puntero de pila del sistema, SP (*Stack Pointer* en M6801, I8085, Z80, M6809, etc.). Al proceder de este modo, se accede al código asociado al siguiente lugar representante marcado mediante un simple *retorno de subprograma* (instrucción RETURN o RTS, presente en casi todos los microprocesadores).

Una nueva instrucción de *separación* es:

8) SPT {separador de transición}

Esta instrucción se utiliza para delimitar el código asociado a una transición. La ejecución de «SPT» sólo tiene lugar al terminar el disparo de una transición que no marque ningún lugar representante. Su ejecución posibilita el encadenar la ejecución de la frase asociada al siguiente lugar representante marcado (dicho de otro modo, la ejecución de SPT provoca el acceso incondicional a la siguiente frase ejecutable). Dado que la pila de tratamiento será la pila del sistema, ello se obtiene directamente mediante una instrucción de retorno de subprograma (RETURN, RTS, ...). (*Nota.* Recuérdese que en la pila de tratamiento se almacenan las direcciones donde comienzan las frases asociadas a los lugares representantes marcados.)

Dentro del repertorio de instrucciones, para el *desmarcado* de lugares sólo se utiliza una:

9) DLS k {desmarcar k-ésimo lugar de sincronización}

La ausencia de instrucciones para desmarcar lugares representantes reside en el propio método de simulación adoptado (§9.6.4) para el sistema, el cual procede (re)marcando los lugares representantes marcados en cada ciclo (por lo tanto, la ausencia de tal acción de marcado supone el desmarcado). (*Nota.* Recuérdese que el (re)marcado del lugar representante p_i se realiza al ejecutarse SP p_i.)

Para *marcar lugares de sincronización* es necesaria una instrucción:

10) MLS k {marcar el k-ésimo lugar de sincronización}

El *marcado de lugares representantes* se definirá gracias a una instrucción, cuya comprensión es clave para entender el funcionamiento global del sistema. Esta instrucción es:

11) MLR p_{ia} {marcar el lugar representante p_i}

Desde un punto de vista funcional, la ejecución de MLR p_{ia} debe posibilitar:

a) La generación de las acciones asociadas a p_i, puesto que este lugar estará marcado al finalizar el ciclo.
b) El apilamiento de δp_i en la pila en formación, puesto que p_i estará marcado al comenzar el próximo ciclo.
c) La ejecución de la instrucción siguiente a «MLR p_{ia}».

Si «MLR p_{ia}» es una *llamada al subprograma* emplazado en la dirección δp_{ia} (CALL δp_{ia}), el funcionamiento obtenido será el deseado. En efecto, la secuencia de acciones que se producirá será la siguiente (recuérdese que la pila de tratamiento es la pila del sistema):

a) Apilar en la pila de tratamiento la dirección de la instrucción que sigue a «MLR p_{ia}».
b) Ejecutar el código que comience en δp_{ia}, es decir, el código que genera las acciones asociadas a p_i.
c) Ejecutar «SP p_i», es decir, apilar δp_i en la pila en formación y, al ejecutarse el retorno, comenzar la ejecución de la instrucción que sigue a «MLR p_{ia}»

Dado que es muy frecuente encontrar en la codificación de RdP la secuencia de instrucciones:

MLR p_{ia};
SPT;

se pueden mejorar las prestaciones definiendo una nueva instrucción que la sustituya (*Nota.* Véase el programa de la figura 9.14):

12) MULR p_{ia}; {marcar último lugar representante} .

La ejecución de «MULR p_{ia}» sólo debe producir la generación de las acciones asociadas a p_i y el apilamiento de δp_i en la pila en formación. Posteriormente, la ejecución debe continuar con el siguiente lugar representante marcado. Para realizar estas funciones debe ejecutarse un salto a δp_{ia} pero sin que se apile previamente ninguna dirección de retorno. Es decir, «MULR p_{ia}» será un simple *salto a* δp_{ia} (JMP δp_{ia}).

A modo de comentario final, sólo queda por señalar que el bit *C* (figura 9.14) indica la *paridad del ciclo* que se ejecuta con lo que se puede efectuar la correcta asignación al puntero de la pila en formación, PPFORM, al comenzar un nuevo ciclo de tratamiento.

Observación. Intercambiando los punteros y apilando COMB en PFORM, se obtiene idéntico comportamiento; es decir:

```
AUXPUN : = PPTRAT;
PPTRAT : = PPFORM;
PPFORM : = AUXPUN;
Apilar COMB en PFORM; {realizable, en este caso, mediante:
                              PPFORM : = PPFORM - 2}
```

Después de considerar las explicaciones anteriores, para comprender en detalle el funcionamiento del sistema se recomienda la «ejecución manual» de un paso de simulación, siguiendo con especial cuidado la traza del estado de las pilas.

Observación. La codificación de las diferentes instrucciones mediante el lenguaje ensamblador de un microcomputador dado, es elemental. Quizás la instrucción aparentemente más compleja de codificar sea «SP p_{ia}». Si se utiliza el microcomputador MOTOROLA 6809, esta instrucción se puede codificar simplemente mediante la secuencia (SP ≡ PPTRA; UP ≡ PPFOR):

```
LDX#p_ia  {carga inmediata de X}
STX ,--U  {apilamiento de X en la pila de usuario}
RTS       {retorno de subprograma}
```

9.9 CONCLUSIONES

La simulación de RdP binarias con microcomputadores de propósito general puede seguir multitud de esquemas. En este capítulo se ha pretendido presentar, desde un punto de vista conceptual, un panorama relativamente completo de métodos de realización. Todos ellos proceden simulando en serie (dentro de un ciclo) la evolución de una (las) RdP.

Para una aplicación dada, definidos unos criterios de calidad y un computador soporte, el lector podrá seleccionar el método que más le interese. Aunque en general no existirá un método óptimo único, los métodos basados en la simulación dirigida por el marcado ofrecerán generalmente, unas prestaciones muy aceptables. No obstante, si la RdP tiene pocos lugares y transiciones (digamos menos de 16), la utilización de esquemas basados en la representación matricial booleana de RdP puede

ser bastante eficiente, sobre todo cuando la palabra del microcomputador es de 16 bits. Ahora bien, así como una palabra larga favorece las prestaciones de los métodos matriciales, la presencia de métodos evolucionados de direccionamiento debe facilitar la realización de los esquemas de simulación basados en la escrutación de listas de lugares representantes marcados. En este sentido resulta significativa la adecuación del MOTOROLA 6809 a la realización del autómata programable presentado en §9.8.

La concepción de arquitecturas hardware especializadas en la simulación de RdP no ha sido tratada en este capítulo por su especificidad, lo que los hace, hoy en día, ser sistemas relativamente costosos. Evidentemente sus prestaciones pueden ser notoriamente superiores a la de los sistemas que se basan en microprocesadores de propósito general. El previsible desarrollo de la tecnología permitirá, en un plazo de tiempo relativamente corto, la realización de circuitos VLSI bajo demanda, a bajo coste, con lo que se podrán realizar económicamente microautómatas programables especializados en la simulación de RdP. Los algoritmos que básicamente utilizarán estas máquinas serán modificaciones, más o menos importantes, de los algoritmos presentados a lo largo de este capítulo.

Por último, cabe señalar que los diferentes algoritmos de simulación de RdP presentados no son, en esencia, más que elementales *despachadores de tarea* (*dispatchers*) de sistemas multiprogramados. Las *tareas* en el sistema están asociadas a los *lugares representantes*, mientras que de todas éstas sólo se encuentran *activas* las tareas asociadas a los lugares representantes *marcados*.

EJERCICIOS

E.9.1 Codifíquese de acuerdo con los esquemas presentados en §9.4, §9.5, §9.6 y §9.7 la RdP de la figura 2.9. Compárense los resultados.

E.9.2 Prográmese la RdP anterior con el lenguaje presentado en §9.8.

E.9.3 Redáctense los diferentes simuladores básicos en el lenguaje del microprocesador disponible. Evalúense sus prestaciones temporales.

Anexo 1

Redes de Petri y programación de computadores

Objetivo del anexo. En este anexo se abordará la descripción del flujo de control de un programa de computador digital. Para ello, se asocia una nueva interpretación a las redes de Petri.

Dado un programa, la *ejecución de una instrucción* (o bloque secuencial de instrucciones) se representará por el disparo de una *transición*. La función de incidencia previa de cada transición determina la condición que ha de cumplirse para que la instrucción se ejecute.

Un esquema de control de un *programa secuencial* puede obtenerse de forma inmediata adoptando el convenio de representación que sigue a continuación:

1) Las instrucciones sin selección se representan por una transición y un lugar de entrada a ésta.
2) Las instrucciones con selección (**si/entonces/si no; caso de**) se representan por un lugar con tantas transiciones de salida como alternativas tenga la selección.
3) Existe un arco que une cada transición al lugar que representa la instrucción que debe ejecutarse a continuación.
4) El marcado inicial comporta una única marca. Ésta realiza las veces de puntero, mediante el que se indica la primera instrucción que debe ejecutarse.

   Se distingue normalmente un lugar inicial (representa la primera instrucción ejecutable) y, al menos, una transición final. Si el programa fuera cíclico, se une, mediante un (varios) arco(s), la(s) transición(es) final(es) al lugar inicial.

Un esquema de control de un *programa concurrente* se representa considerando las tres primeras reglas enunciadas y, además, las dos siguientes:

4') El marcado inicial representa el conjunto de instrucciones inicialmente ejecutables, así como el estado de los diferentes recursos que pueda utilizar el programa.
5) Las instrucciones **par. comienzo** (respec. **par. fin**) se representan mediante una transición con tantos lugares de salida (respec. de entrada) como instrucciones, o bloques indivisibles de instrucciones, tengan que ser ejecutadas.

EJEMPLO. En la figura A.1.1 se representa un programa concurrente con una síntaxis de tipo PASCAL y una RdP que describe su *flujo de control*. A cada transición se le asocia una etiqueta constituída por el par *Predicado/Instrucción*. Si una transición no está condicionada por ningún predicado, se omite dicho campo en la etiqueta. Si una transición no representa a ninguna instrucción, se omite el campo correspondiente.

Principio: I_0
mientras que Condición 1 **hacer** (*p_2*)
si Condición 2 **entonces** I_1 (*p_3*)
si no I_2
fsi (*p_4*)
par. comienzo I_3, I_4 (*p_4*)
par. fin (*p_6, p_8*)
fmq (*p_2*)
ir a Principio

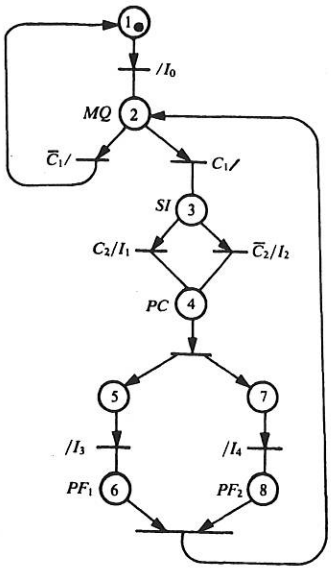


Figura A.1.1. Representación del flujo de control del programa.

El marcado inicial asociado a la red (figura A.1.1) indica que en el programa comenzará un ciclo de ejecución.

Una marca en p_2 indica que comienza la ejecución de la instrucción compuesta **mientras que/hacer**. Se puede establecer de idéntica forma la siguiente correspondencia entre marcados e instrucciones que hay que ejecutar:

$$\text{Marcado de} \begin{cases} p_3 \rightarrow \textbf{si/entonces/si no.} \\ p_4 \rightarrow \textbf{par. comienzo.} \\ p_6 \text{ y } p_8 \rightarrow \textbf{par. fin.} \end{cases}$$

Supongamos ahora que se desea obtener el esquema de control de una aplicación en la que varios procesos compiten por recursos del sistema (datos, programas, periféricos, etc.). Si las sincronizaciones están expresadas con *semáforos*†, el flujo de control podrá obtenerse de forma inmediata. En efecto, al nivel de detalle que interesa aquí, un semáforo se representará mediante un lugar marcado inicialmente

† En §4.7.3 se modelan los semáforos con mayor detalle y se deduce el *teorema del semáforo*.

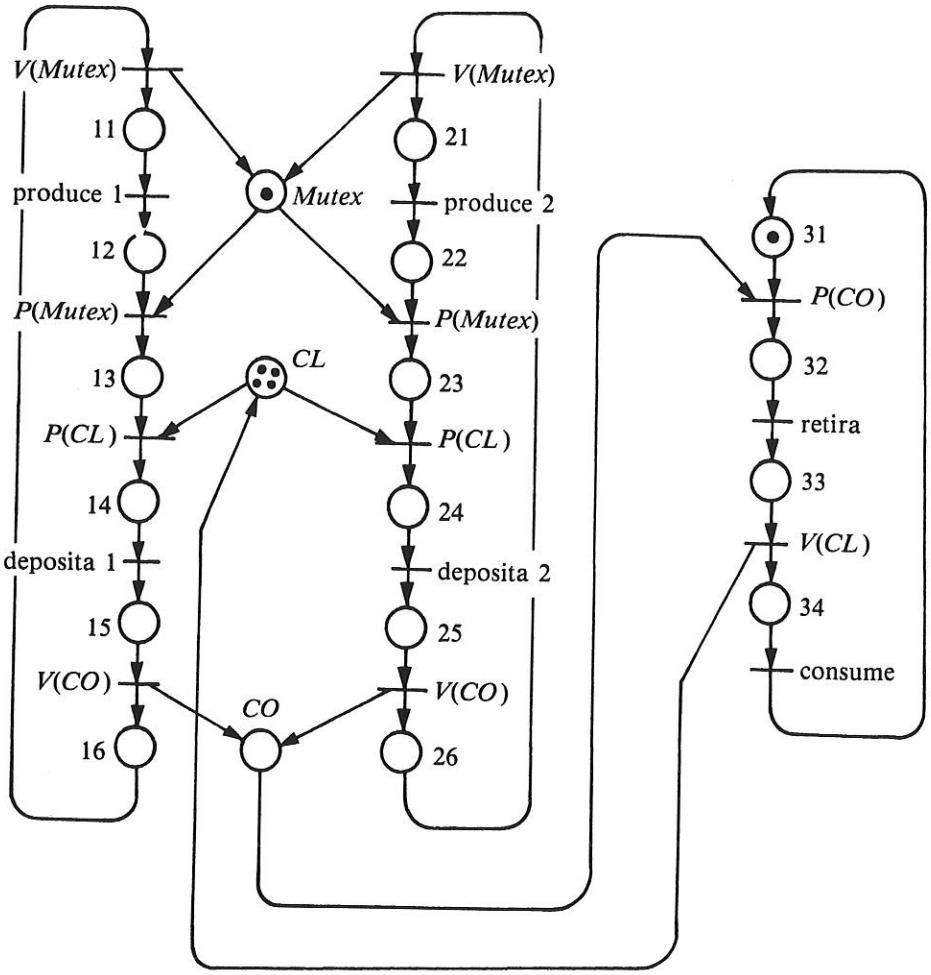


Figura A.1.2. Modelo de sistema con los dos productores en exclusión mutua y un consumidor (capacidad del almacén 4).

con un número de marcas igual al valor inicial del *contador del semáforo*. Las transiciones de entrada a S, lugar asociado al semáforo, representan la ejecución de las primitivas *señalar* (V). Las transiciones de salida de S representan la ejecución y el franqueo de las primitivas *esperar* (P).

EJEMPLO. Sea un sistema de producción-consumo (véase también el §2.4.1) que comprende dos *procesos productores*, un *proceso consumidor* y un *tampón circular* de cuatro compartimentos a través del cual se comunican los procesos. El acceso simultáneo de los procesos productores al tampón está prohibido.

Sean los semáforos *Mutex*, *CL* y *CO*, cuyas funciones son las siguientes:

1) *Mutex*: garantiza la exclusión mútua de los procesos productores en su acceso al tampón.
2) *CO*: representa el número de compartimentos ocupados. Garantiza el que no haya superproducción.
3) *CL*: representa el número de compartimentos libres. Garantiza el que no haya superconsumo (consumo ficticio).

Inicializando los semáforos *Mutex*, *CO y CL*, con 1, 0 y 4 podremos describir la aplicación mediante el programa siguiente:

Productor 1	*Productor 2*	*Consumidor*
produce 1	produce 2	P (*CO*)
P (*Mutex*)	P (*Mutex*)	retira
P (*CL*)	P (*CL*)	V (*CL*)
deposita 1	deposita 2	consume
V (*CO*)	V (*CO*)	
V (*Mutex*)	V (*Mutex*)	

La RdP de la figura A.1.2 representa el esquema de control de la aplicación.

Anexo 2

Elementos sobre grafos

Objetivo del anexo. La presentación de la terminología básica sobre grafos a la que se hace referencia en el texto (véase, por ejemplo, [BERG 73] [KAUF 76]).

A.2.1 CONCEPTO DE GRAFO. REPRESENTACIÓN Y ALGUNAS INTERPRETACIONES

Un *grafo*, G, es un par $\langle Q, \Gamma \rangle$ en el que:

1) Q es un conjunto finito, no vacío, de objetos denominados *nudos*.
2) $\Gamma \subset Q \times Q$ es un conjunto de objetos (γ_k) denominados *arcos* (orientados). Existe un arco que une q_i a q_j si y sólo si $(q_i, q_j) \in \Gamma$.

Un nudo se representa gráficamente mediante un *punto* (•) ó una *circunferencia* (◯). Un arco se representa por una *flecha*.

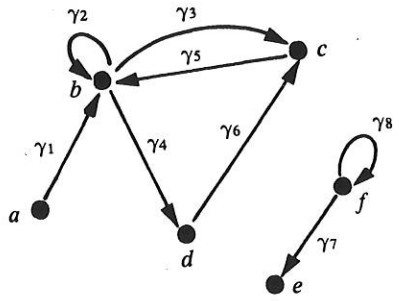
Existen dos representaciones matriciales clásicas de un grafo:

1) *La matriz «asociada» al grafo*, que consiste en una matriz cuadrada con tantas filas y columnas como nudos tiene el grafo. Sea $A = [a_{ij}]$ la matriz asociada a G. Esta se define de la forma siguiente:

$$a_{ij} = \begin{cases} 1 & \text{si} \quad (q_i, q_j) \in \Gamma \\ 0 & \text{si} \quad (q_i, q_j) \notin \Gamma \end{cases}$$

El grafo de la figura A.2.1 se representa por la matriz asociada siguiente:

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} a & b & c & d & e & f \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$



$G = \langle Q, \Gamma \rangle$
$Q = \{a, b, c, d, e, f\}$
$\Gamma = \{(a, b), (b, b), (b, c), (b, d), (c, b), (d, c), (f, e), (f, f)\}$

Figura A.2.1. Representación gráfica de G.

2) *La matriz de incidencia nudo-arco* (válida sólo si los arcos no tienen ambos extremos en un mismo nudo; es decir, si no existe q_i tal que $(q_i, q_i) \in \Gamma$). Se trata de una matriz rectangular con tantas filas y columnas como nudos y arcos tenga, respectivamente. Se define la matriz $M = [m_{ij}]$ de acuerdo con la expresión siguiente:

$$m_{ij} = \begin{cases} -1 \text{ si } (q_i, q_j) \in \Gamma \\ +1 \text{ si } (q_j, q_i) \in \Gamma \\ 0 \text{ en cualquier otro caso} \end{cases}$$

Como puede observarse en la matriz de incidencia nudo-arco, un elemento vale -1 si existe un arco saliente y $+1$ si existe un arco entrante.

El grafo obtenido al eliminar los arcos 2 y 8 (figura A.2.1) se representa por la matriz de incidencia siguiente:

$$M = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} -1 & 0 & 0 & 0 & 0 & 0 \\ +1 & -1 & -1 & +1 & 0 & 0 \\ 0 & +1 & 0 & -1 & +1 & 0 \\ 0 & 0 & +1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & 0 & 0 & -1 \end{pmatrix} \end{array}$$

Una generalización natural del concepto de grafo se obtiene al asociar un peso (un valor) a cada uno de los arcos. Se hablará de *grafo ponderado*. En este caso, la relación $\Gamma \subset Q \times Q$ se transforma en una función que, por ejemplo, puede tomar una de las formas siguientes:

$$\Gamma: Q \times Q \to \begin{cases} \{0, 1\} & \text{(caso anteriormente considerado)} \\ \mathbb{N} & \text{(enteros naturales)} \\ \mathbb{R} & \text{(reales)} \end{cases}$$

Esta función se define de acuerdo con la aplicación que se desee realizar con el

grafo. Las matrices que representan el grafo tendrán por elementos los valores que toma la función Γ.

Los grafos se utilizan para muy diversas aplicaciones (optimización de problemas de transporte, evolución funcional de un sistema discreto, evaluación de prestaciones, ...). A cada tipo de aplicación se le asocia una interpretación diferente. Veamos algunas de ellas:

1) *Conexión entre distintos puntos.* Los nudos representan los puntos (ciudades, emplazamiento de circuitos en una placa,...). A los arcos se les puede asociar un valor que represente la distancia entre los dos puntos que une o, simplemente, la existencia de una conexión,...

   Los problemas típicos que se definen sobre estos grafos son la optimización del trazado de las conexiones (se pretende evitar cruces entre conexiones, etc.), la elección de una ruta que pase por todos los emplazamientos (ciudades) de tal manera que la longitud (costo) sea mínima,...
2) *Modelación funcional de un sistema secuencial.* Los nudos representan los estados del sistema. A los arcos se les asocian los eventos que provocan las transiciones entre estados. Este tipo de aplicaciones se considera en el capítulo 1.
3) *Procesos markovianos.* Los nudos representan estados del proceso en estudio. A los arcos se les asocia la probabilidad de transición entre los estados.

   Problemas típicos que se definen sobre estos grafos son la evaluación de prestaciones (rendimientos, tasas de ocupación, etc.), de la seguridad de funcionamiento,...

A.2.2 TERMINOLOGÍA BÁSICA

Se dice que q_i es *predecesor* de q_j y q_j *sucesor de* q_i si $(q_i, q_j) \in \Gamma$. El *conjunto de los nudos predecesores* (respec. *sucesores*) de q_i se representan por $\Gamma^-(q_i)$ (respec. $\Gamma^+(q_i)$). Se dice, de este modo, que un nudo q_j es *vecino* de q_i si q_j es predecesor o sucesor de $q_i [q_j \in \Gamma^-(q_i) \cup \Gamma^+(q_i)]$.

Un *camino* o *ruta* es una secuencia $\{\gamma_1, \gamma_2, \dots\}$ de arcos tal que el extremo final de γ_i coincida con el extremo inicial de γ_{i+1}. Un *circuito* es un camino que no pasa dos veces por ningún nudo y tal que el nudo inicial y el nudo final es uno mismo. El número de arcos del camino (respec. circuito) es su *longitud*.

En el grafo de la figura A.2.1 se tienen, por ejemplo, el camino $\{\gamma_1, \gamma_2\}$ (de longitud 2) y el circuito $\{\gamma_4, \gamma_6, \gamma_5\}$ (de longitud 3).

Un grafo es *fuertemente conexo* si existe, al menos, un camino entre dos nudos cualesquiera. El grafo de la figura A.2.1 no es fuertemente conexo puesto que, por ejemplo, no existe ningún camino que llegue al nudo a. Un grafo es *conexo* si existe, al menos, una secuencia de arcos no orientados (*aristas*) que una dos nudos cualesquiera. El grafo que se considera tampoco es conexo pues $\{e, f\}$ no está conectado con $\{a, b, c, d\}$.

El conjunto de nudos, $\{q_j\}$, que pueden ser alcanzados por un camino a partir de un nudo q_i, y tales que a partir de q_j se alcance q_i, es la *componente fuertemente conexa* definida por q_i (o por uno cualquiera de los q_j, pues la fuerte conexión es

una relación de equivalencia). Si no se considera la orientación de los arcos, se define la *componente conexa* con q_i. $\{a, b, c, d\}$ y $\{b, c, d\}$ (figura A.2.1) son respectivamente una componente conexa y otra fuertemente conexa.

Si en un grafo se reduce cada componente fuertemente conexa a un nudo, se obtiene un grafo *sin circuitos*.

Un grafo es *bipartido* si el conjunto de sus nudos puede ser particionado en dos clases Q_1 y Q_2 de forma que dos nudos de la misma clase no están nunca unidos a través de un arco. Un grafo bipartido se representa por $G = \langle Q_1, Q_2, \Gamma \rangle$, o bien por $G = \langle Q_1, Q_2, \Gamma_1, \Gamma_2 \rangle$, donde $\Gamma_1 \subseteq Q_1 \times Q_2$ y $\Gamma_2 \subseteq Q_2 \times Q_1$. Si se trata de un grafo bipartido ponderado, $\Gamma_1 : Q_1 \times Q_2 \to V$, $\Gamma_2 : Q_2 \times Q_1 \to V$.

Una red de Petri ordinaria es un grafo bipartido. Una red de Petri generalizada es un grafo bipartido ponderado.

Un *árbol* es un grafo sin circuitos en el que:

1) Existe un único nudo, denominado *nudo raiz*, que no posee ningún nudo predecesor. Si q_0 es el nudo raiz, entonces $\Gamma^-(q_0) = \varnothing [|\Gamma^-(q_0)| = 0]$.
2) Cualquier nudo del grafo que no sea el nudo raiz posee un único nudo predecesor $[\forall q_i \in Q, q_i \neq q_0, |\Gamma^-(q_i)| = 1]$.
3) Existe un subconjunto de nudos, $Q_n \subset Q$, que no tienen nudo sucesor $(\forall q_j \in Q_n, \Gamma^+(q_j) = \varnothing)$. Son los *nudos terminales* u *hojas*.

Una aplicación clásica de los árboles es la representación de la genealogía de una persona o de una familia.

A.2.3 OBTENCIÓN DE LAS COMPONENTES CONEXAS Y LAS FUERTEMENTE CONEXAS DE UN GRAFO

En este apartado se presenta un algoritmo que permite obtener *la* componente conexa y *la* componente fuertemente conexa con un nudo dado. La aplicación reiterada de este algoritmo permite llevar a cabo la partición del grafo en componentes conexas y también en componentes fuertemente conexas (ambas propiedades definen relaciones de equivalencia).

Sea el nudo q_i.

ALGORITMO: CÁLCULO DE COMPONENTES CONEXA Y FUERTEMENTE CONEXA

1) Marcar « + − » y « − » el nudo q_i.
2) Marcar « + » todo nudo sucesor, aún no marcado con « + », de un nudo previamente marcado « + ».
3) Marcar « − » todo nudo predecesor, aún no marcado con « − », de un nudo previamente marcado con « − ».
4) Cuando no pueda marcarse ningún nudo más, tendremos que:
   4.1 la componente fuertemente conexa con q_i viene dada por el conjunto de nudos marcado « + » y « − ».
   4.2 la componente conexa con q_i viene dada por el conjunto de nudos marcado (« + » o/y « − »).

Volviendo sobre el grafo de la figura A.2.1, se puede deducir que si b es el nudo marcado inicialmente « + » y « − », se obtienen los marcados siguientes:

$$a = \{ - \}, b = c = d = \{ + , - \}, e = f = \emptyset$$

por lo tanto:

$$\begin{cases} b \text{ es conexo con } \{a, c, d\} \\ b \text{ es fuertemente conexo con } \{c, d\} \\ b \text{ no está conectado con } \{e, f\} \end{cases}$$

Notas

1) El conjunto de nudos marcados « + » contiene a los nudos que, mediante un camino de longitud cualquiera, se pueden alcanzar a partir de q_i. Se dice que dicho conjunto de nudos es el *cierre* o *espectro* de q_i. Se le representa por $\hat{\Gamma}^+(q_i)$.

2) El conjunto de nudos marcados « − » contiene a los nudos a partir de los cuales, mediante un camino de longitud cualquiera, se puede alcanzar q_i. Se dice que dicho conjunto de nudos es el *cierre* o *espectro inverso* de q_i. Se le representa por $\hat{\Gamma}^-(q_i)$.

3) Los espectros de un conjunto de nudos, C_j, se definen como la unión de los espectros de sus componentes:

$$C_j = \{q_{j1}, q_{j2}, \ldots, q_{jn}\} \Rightarrow \begin{cases} \hat{\Gamma}^+(C_j) = \hat{\Gamma}^+(q_{j1}) \cup \ldots \cup \hat{\Gamma}^+(q_{jn}) \\ \hat{\Gamma}^-(C_j) = \hat{\Gamma}^-(q_{j1}) \cup \ldots \cup \hat{\Gamma}^-(q_{jn}) \end{cases}$$

Anexo 3

Sistemas de ecuaciones lineales: obtención de bases de anuladores de una matriz

Objetivo del anexo. La presentación de un método eficaz para obtener bases de anuladores (derechos e izquierdos) de una matriz. A modo de introducción se recuerdan algunos conceptos y resultados que son utilizados con posterioridad. No obstante, se supone al lector familiarizado con los conceptos básicos de espacios vectoriales y de matrices (véase, por ejemplo, en castellano [HOFF 73] [BURG 77] [REZA 77] [ROJO 82]).

A.3.1 INTRODUCCIÓN: SISTEMAS DE ECUACIONES LINEALES Y HOMOGÉNEAS

Sea el sistema de n ecuaciones lineales con m incógnitas:

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1m}x_m &= 0 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2m}x_m &= 0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nm}x_m &= 0 \end{aligned}\right\} \qquad [1]$$

(n puede ser mayor, igual o menor, que m).

Si A es la matriz definida por $[a_{ij}]_{n\times m}$ y X el vector definido por $[X_i]_m$, el sistema de ecuaciones [1] se puede reescribir matricialmente:

$$A \cdot X = 0. \qquad [1\text{bis}]$$

Se dice que X es *solución* del sistema de ecuaciones [1/1bis] si $A \cdot X = 0$. Todo sistema de ecuaciones lineales homogénas admite al menos una solución que es $X = 0$ (solución trivial). El sistema de ecuaciones se denomina *determinado* o *indeterminado* según que admita una única solución (en este caso la trivial) o varias soluciones.

Sea $\mathcal{E}$ el espacio engendrado por un conjunto de vectores. Se denomina dimensión del espacio, *dim*($\mathcal{E}$), al máximo número de vectores del mismo que son linealmente independientes. Todo conjunto de *dim*($\mathcal{E}$) vectores linealmente independientes es una *base* del espacio y permite generar cualquier vector del mismo.

Sea $\mathcal{F}$ y $\mathcal{C}$ los espacios engendrados por las filas y las columnas de la matriz A, respectivamente. Sea *rango*(A) = r.

Proposición 1. La dimensión de $\mathfrak{F}$ y la de $\mathfrak{C}$ coinciden. Su valor es el rango de la matriz A: $dim(\mathfrak{F}) = dim(\mathfrak{C}) = r$. □

Sea $\mathfrak{Y}$ el espacio engendrado por las soluciones de la ecuación $Y^T \cdot A = 0$ (Y es un *anulador izquierdo* de A). Sea $\mathfrak{D}$ el espacio engendrado por las soluciones de la ecuación $A \cdot X = 0$ (X es un *anulador derecho* de A).

Proposición 2. 2.1 $\dim(\mathfrak{Y}) = n - r$.
2.2 $\dim(\mathfrak{D}) = m - r$. □

Una justificación de 2.1 se obtiene al considerar que cualquier vector C del espacio $\mathfrak{C}$ y cualquier vector Y del espacio $\mathfrak{Y}$ son *ortogonales* (producto escalar nulo). Por consiguiente, no se puede obtener ningún vector Y como combinación lineal de vectores C, y viceversa. Al considerar el espacio suma de $\mathfrak{C}$ e $\mathfrak{Y}$, y puesto que se ha visto que son ortogonales:

$$dim(\mathfrak{C} + \mathfrak{Y}) = n = dim(\mathfrak{C}) + dim(\mathfrak{Y}) = r + dim(\mathfrak{Y}).$$

Una justificación de 2.2 se puede obtener con el mismo tipo de razonamiento al considerar los espacios $\mathfrak{D}$ y $\mathfrak{F}$.

Corolario. Si $r = n$ [$r = m$], entonces $dim(\mathfrak{Y}) = 0$ [$dim(\mathfrak{C}) = 0$]; es decir, no existe anulador izquierdo [derecho] distinto de la solución trivial. El sistema homogéneo de ecuaciones correspondientes es determinado.

A.3.2 TRANSFORMACIONES ELEMENTALES

Una gran cantidad de métodos de resolución de sistemas de ecuaciones lineales se basa en la idea de *equivalencia*. Si A y $\bar{A}$ son matrices con el mismo número de columnas, los sistemas $A \cdot X = 0$ y $\bar{A} \cdot \bar{X} = 0$ son *equivalentes* si toda solución de uno de ellos es solución del otro, y viceversa.

Puesto que se consideran sistemas homogéneos, se trabajará sólo con la matriz del mismo, A.

Se denomina *transformación elemental* sobre A, a cualquiera de las operaciones siguientes:

1) Permutar dos filas (o columnas) entre sí.
2) Sumar a una fila (o columna) una combinación lineal de las restantes filas (o columnas).
3) Multiplicar una fila (o columna) por un escalar no nulo.

Proposición 3. Si la matriz $\bar{A}$ se obtiene por transformaciones elementales a partir de A, entonces A y $\bar{A}$ son equivalentes (en particular, $rango(A) = rango(\bar{A}) = r$).

A.3.3 OBTENCIÓN DE BASES DE ANULADORES

Sea I_n la matriz unitaria de rango n. Utilizando únicamente transformaciones ele-

mentales de filas y permutaciones de columnas, se puede transformar el par $[I_n \vdots A]$ en $[\bar{I}_n \vdots \bar{A}]$, donde:

$$\bar{A} = \begin{pmatrix} E \vdots F \\ \cdots\cdots \\ 0 \end{pmatrix} \qquad \begin{cases} E_{r\times r} \text{ matriz triangular superior} \\ F_{r\times(m-r)} \end{cases}$$

Cada una de las filas de $\bar{I}_n$ que se corresponda con una fila nula de A es un anulador izquierdo de $\bar{A}$ (y por tanto de A). Puesto que, de acuerdo con la proposición 3, la matriz $\bar{I}_n$ es de rango n, las $n - r$ filas que se correspondan con filas nulas de $\bar{A}$ son linealmente independientes. Por último, como la dimensión del espacio $\mathcal{Y}$ es $n - r$, resulta que las $n - r$ filas de $\bar{I}_n$ que se han considerado, forman una *base de anuladores izquierdos* de A, B_Y.

A partir de esta situación también es fácil obtener una *base de anuladores derechos* de la matriz A, B_D. En efecto,

$$\bar{A} \cdot B_D = \begin{pmatrix} E \vdots F \\ \cdots\cdots \\ 0 \end{pmatrix} \cdot \begin{pmatrix} B_{D1} \\ \cdots \\ B_{D2} \end{pmatrix} = 0 \Rightarrow E \cdot B_{D1} + F \cdot B_{D2} = 0$$

donde B_D posee m filas y, según la proposición 2, $m - r$ columnas. Las matrices B_{D1} y B_{D2} poseen r y $m - r$ filas, respectivamente. Eligiendo $B_{D2} = I_{m-r}$, se garantiza que el rango de B_{D2} sea $m - r$ y se simplifica el cálculo. En este caso se tiene $B_{D1} = -E^{-1} \cdot F$ (E es inversible puesto que su rango es r).

En resumen, una base de anuladores derechos viene dada por

$$B_D = \begin{pmatrix} -E^{-1} \cdot F \\ \cdots\cdots\cdots\cdots \\ I_{m-r} \end{pmatrix}$$

Observación 1. La matriz $\bar{A}$ puede transformarse en la matriz equivalente $\bar{\bar{A}} = \begin{pmatrix} I_r \vdots G \\ \cdots\cdots \\ 0 \end{pmatrix}$ utilizando sólo transformaciones sobre las filas. En este caso, se obtiene una base B_D de la forma siguiente:

$$B_D = \begin{pmatrix} -G \\ \cdots\cdots \\ I_{m-r} \end{pmatrix}$$

Observación 2. Todo el razonamiento desarrollado para obtener dos bases de anuladores puede hacerse a partir de la matriz transpuesta de A, A^T. En este caso, se obtiene primeramente una base de B_D^T y, a continuación, una base B_Y^T. Si se desea calcular una B_Y y una B_D siendo $n < m$ (típicamente es el caso de las *máquinas de estado*), el trabajo con la matriz A suele ser más eficaz. Si $n > m$, conviene trabajar con A^T (típicamente es el caso de los *grafos de sincronización* o *grafos marcados*).

Anexo 4

Cómputo eficiente de todas las componentes elementales de una RdP generalizada

Objetivo del anexo. La presentación de las bases teóricas de un algoritmo que permite calcular eficientemente todas las componentes elementales de una RdP [MART 81*b*]. Se incluye un listado de su codificación en PASCAL.

A.4.1 INTRODUCCIÓN

En §4.7.2.2 se presentó un algoritmo para la *obtención de todas las componentes conservativas elementales*. La aplicación de éste a la RdP de la figura 3.5 demostró que no sólo se obtienen las componentes conservativas elementales sino que también se obtienen componentes no elementales. Habida cuenta que, normalmente, sólo interesa la determinación de las componentes elementales (§4.7.2.3), debe resolverse el *problema de la eliminación de las componentes no elementales*. Para ello, se pueden utilizar dos vías distintas que son:

1) Aplicar el mencionado algoritmo y, posteriormente, eliminar las componentes no elementales por comparación entre sus soportes.
2) Eliminar las componentes no elementales a medida que sean generadas.

La primera de las dos vías sugeridas presenta algunos inconvenientes que podemos resumir en los dos puntos siguientes:

1) El algoritmo de generación de componentes será, probablemente, lento en la ejecución, dado que puede generarse un gran número de componentes no elementales (esto suele suceder si la mayoría de las transiciones poseen varios lugares de entrada y de salida). Además, las componentes no elementales incrementan la cantidad de memoria necesaria en la ejecución.
2) El proceso final de selección de componentes elementales será tanto más largo, cuanto mayor sea el número de componentes (elementales o no) generadas.

Por lo expuesto, se desarrollará la segunda alternativa. Una ventaja adicional de ésta, es que nos permitirá presentar una *caracterización algebraico-lineal* del concepto de componentes elemental.

A.4.2 CARACTERIZACIÓN DE LAS COMPONENTES CONSERVATIVAS ELEMENTALES

Como se ha anunciado previamente, se pretende la eliminación de las componentes no elementales durante el propio proceso de generación.

De acuerdo con la observación realizada en la introducción al §4.6, desarrollamos la teoría utilizando la matriz de incidencia, pero si la RdP no es pura, se trabajará con la matriz de flujos de marcas.

Sea l_i la i-ésima fila de la matriz de incidencia, C, y sea $Y^T = (\lambda_1 \ldots \lambda_q, 0 \ldots 0)$ con $\lambda_i \in \mathbb{Z}^+$, una componente de la RdP (el orden de los términos de la componente no le resta generalidad).

Condición necesaria y suficiente para que una componente conservativa sea elemental (proposición 1).

La componente Y es elemental si y sólo si $q = rango\begin{pmatrix} l_1 \\ \cdots \\ l_q \end{pmatrix} + 1$. □

DEMOSTRACIÓN. Dado que Y es una componente conservativa, se puede escribir

$$\sum_{i=1}^{q} \lambda_i l_i = 0, \qquad \lambda_i \in \mathbb{Z}^+ \qquad [1]$$

Si $r = rango\begin{pmatrix} l_1 \\ \cdots \\ l_q \end{pmatrix}$, se pueden determinar r entre las q filas, $\{l_1, \ldots, l_q\}$, que formarán *una base*. Por lo tanto, si $q > r + 1$, existirá una relación distinta de [1] de la forma:

$$\sum_{j=1}^{r+1} \mu_j l'_j = 0 \text{ donde} \begin{cases} l'_j \in \{l_1, \ldots, l_q\} \\ \mu_j \in \mathbb{Z} \end{cases} \qquad [2]$$

Entre [1] y [2] puede eliminarse al menos uno de los términos en l_i, quedando una relación [3] similar a [1] con todos los coeficientes positivos y con menor número de términos (sean s términos):

$$\sum_{k=1}^{s} \lambda''_k l''_k = 0, \text{ donde} \begin{cases} l''_k \in \{l_1, \ldots, l_q\} \\ \lambda''_k \in \mathbb{Z}^+ \\ s < q \end{cases} \qquad [3]$$

Si $s > r + 1$, puede reaplicarse el razonamiento presentado hasta llegar a otra relación [3] con $s'' = r + 1$ términos. En este caso, la relación será única por serlo la representación de un vector en función de una base. Por consiguiente, [3] define una componente elemental cuyas coordenadas no nulas serán los valores λ''_k. □

La inserción del resultado anterior en el bucle del algoritmo del §4.7.2.2 permite generar *todas* las componentes conservativas elementales y *sólo* éstas. No obstante, su utilización es poco eficiente desde un punto de vista computacional debido a la necesidad de calcular el rango de las matrices $\begin{pmatrix} l_1 \\ \cdots \\ l_q \end{pmatrix}$. Para mejorar las prestaciones

del algoritmo, se va a considerar un *mayorante del rango*, muy rápido de calcular: el número de columnas no nulas de la mencionada matriz, $r' \geqslant r$. Por desgracia, procediendo de este modo no se podrá garantizar la no generación de componentes no elementales, aunque la experiencia ha revelado su enorme interés.

Condición necesaria para que una componente conservativa sea elemental (proposición 2). Para que la componente $Y^T = (\lambda_1, \ldots, \lambda_q, 0, \ldots, 0)$, donde $\lambda_i \in \mathbb{Z}^+$, sea elemental es necesario que $q \leqslant r' + 1$. □

Su justificación es inmediata, dado que $r' \geqslant r$ y la componente Y es elemental si y sólo si $q = r + 1$ (proposición 1).

La inserción de esta condición como paso 2.3 en el algoritmo del §4.7.2.2 permite eliminar, durante el proceso de generación, aquellas componentes conservativas que no cumplan la condición necesaria de elementaridad. El algoritmo resultante se ha mostrado, desde un punto de vista práctico, excepcionalmente eficiente, permitiendo, normalmente, la eliminación de todas las componentes no elementales. Esta excepcional eficiencia puede comprenderse, en parte, dado que las matrices de incidencia son normalmente casi-vacías y, con frecuencia, coinciden el rango, r, y su mayorante, r'.

A.4.3 CODIFICACIÓN DEL ALGORITMO

De acuerdo con lo expuesto en el apartado anterior, la inserción del resultado enunciado por la proposición 2 en el algoritmo presentado en el §4.7.2.2, conduce a otro algoritmo en el que se elimina la mayoría (normalmente todas) de las componentes no elementales. El siguiente listado es una codificación en PASCAL del algoritmo «mejorado». Para calcular el mayorante del rango, r', se asocia a C la matriz booleana A. En A se asigna inicialmente a cada elemento nulo de la matriz de incidencia, C, un booleano de valor FALSE; si el elemento de C es no nulo se le asigna el valor TRUE. Posteriormente a cada elemento $a_{ij} \in A$ se le asignará el booleano resultado de las uniones lógicas de aquellos asociados a los elementos que, por combinación lineal, lo generen. El entero r' asociado a una componente conservativa es el número de booleanos que adoptan el valor TRUE en la fila correspondiente de la matriz booleana A.

```
PROGRAM componentes(input,output);

{-----------------------------------------------------------------------------------------------------------

Calcula las componentes conservativas elementales de una Red de Petri a partir de su matriz
de incidencia.

-----------------------------------------------------------------------------------------------------------}

CONST numcolumnas = 25; numfilas = 90;
        {Dimensionar adecuadamente según las necesidades}
TYPE matriz = ARRAY[1..numfilas,1..numcolumnas]OF integer;
        {Significado de las variables mas importantes:
         A :Matriz de Trabajo, inicialmente igual a la de incidencia de la Red de Petri.
         D :Matriz de componentes;
   filavacía:Valor booleano asociado a cada fila de «A» que indica si esta vacía, o bien
             si debe ser eliminada (valor «true»).
valornonulo:Valor booleano asociado a cada elemento de «A»
    puntero:indica el número de las filas de «A» y «D» utilizadas en un momento dado.
   puntero1:apunta a la última fila añadida a «A» (y a «D»)                    }
VAR filavacía:ARRAY[1..numfilas]OF boolean;
     A,D:matriz; numtransiciones,numlugares:integer;
     valornonulo:ARRAY[1..numfilas,1..numcolumnas]OF boolean;
     i,j,k:integer; puntero,puntero1:integer;

PROCEDURE leer;
    {Lee la matriz de incidencia de la red de Petri, la escribe en el fichero de salida e
     inicializa «D» igual a la matriz identidad.
     El fichero de entrada debe contener la siguiente información:
     primera línea: número de filas y de columnas de la matriz de incidencia de la red.
        sig. líneas: cada una contiene una fila de la matriz de incidencia, separándose
                     mediante blancos los diferentes elementos                  }
VAR  i,j : integer;
 BEGIN
  reset(input,'ENTRAD,CCE'); rewrite(output,'SALIDA.CCE');
  writeln(output, 'MATRIZ DE INCIDENCIA');
  read(input,numlugares,numtransiciones);
  puntero : = numlugares;
  FOR i : = 1 TO numlugares DO
   BEGIN
    FOR j : = 1 TO numtransiciones DO
     BEGIN
      read(input,A[i,j]); write(output,A[i,j] : 7);
      IF A[i,j] = 0
           THEN valornonulo[i,j] : = false
           ELSE valornonulo[i,j] : = true
     END;
    writeln(output); filavacía[i] : = false
   END;
```

```
    FOR i : = numlugares + 1 TO numfilas DO filavacia[i] : = true;
    FOR i: = 1 to numlugares DO
     FOR j : = 1 TO numlugares DO
      IF i = j
         THEN D[i,j]: = 1
         ELSE D[i,j]: = 0
END; {Fin de leer}

{-------------------------------------------------------------------------------------------------------------------------}
PROCEDURE escribir (var M:matriz;nfilas,ncolumnas:integer);
      {Escribe en el fichero de salida la matriz «M», omitiendo las filas marcadas como
       vacias.}
VAR i,j : integer;
 BEGIN
  IF nfilas  < = 0
  THEN writeln(output, 'NO EXISTEN COMPONENTES ELEMENTALES')
  ELSE
   BEGIN
    writeln(output, 'COMPONENTES CONSERVATIVAS ELEMENTALES')
    FOR i : = 1 TO nfilas DO
     BEGIN
      IF NOT filavacia[i]
      THEN
       BEGIN
        FOR j : = 1 TO ncolumnas DO write(output,M[i,j] : 7);
        writeln(output)
       END
     END
   END
 END; {Fin de escribir}

PROCEDURE combinacionlineal(fila1,fila2,columna : integer);
VAR i1,j1,factor1,factor2 : integer;

{-------------------------------------------------------------------------------------------------------------------------}
 FUNCTION mcd(uno,otro : integer) : integer;
            {Calcula el máximo común divisor de dos números}
  VAR dividendo,divisor,resto : integer;
   BEGIN
    IF uno > = otro
       THEN BEGIN dividendo : = uno; divisor : =otro END
       ELSE BEGIN dividendo : =otro; divisor : = uno END;
    WHILE divisor < >0 DO
      BEGIN
       resto : = dividendo MOD divisor;
       dividendo : =divisor; divisor : = resto
      END;
    mcd : = dividendo
   END; {Fin de mcd}

   {-------------------------------------------------------------------------------------------------------------------------}
```

```
FUNCTION buscafilavacía : integer;
          {Devuelve como resultado un puntero de la primera fila vacía de «A»}
    VAR i,vacía : integer;
     BEGIN
      i : = 1; vacía : = 0;
      WHILE i < = numfilas AND NOT filavacía [i] DO i: = i + 1;
      buscafilavacía : = i
     END; {Fin de buscafilavacía}

{-------------------------------------------------------------------------------------------------------------------}

BEGIN {Procedure combinacionlineal
        Realiza la combinación lineal de dos filas (fila1 y fila2) de «A» de forma que se
        anule su elemento «columna». Realiza la misma combinación con identicas filas
        de «D». Los booleanos «valornonulo» asociados a los elementos de la fila resul-
        tante se calculan como la unión lógica de los correspondientes a las filas antes
        citadas.}
 puntero1 : = buscafilavacía;
 j1 : = mcd(abs(A[fila1,columna],abs(A[fila2,columna]));
 factor1 : = abs(A[fila2,k]) DIV j1;
 factor2 : = abs(A[fila1,k]) DIV j1;
 FOR i1 : = 1 TO numtransiciones DO
  BEGIN
   A[puntero1,i1] : = A[fila1,i1] * factor1 + A[fila2,i1] * factor2;
   valornonulo[puntero1,i1] : = valornonulo[fila1,i1] OR valornonulo[fila2,i1]
  END;
 FOR i1 : = 1 TO numlugares DO
     D[puntero1,¡1] : = D[fila1,i1] * factor1 + D[fila2,i1] * factor2
END; {Fin de combinacionlineal}

PROCEDURE anulacolumnas;

{-------------------------------------------------------------------------------------------------------------------}

  FUNCTION cardinal (fila : integer) : integer;
            {Calcula el cardinal del soporte de la componente «fila» de «D»}.
   VAR      i1, card: integer;
    BEGIN
     card : = 0;
     FOR i1 : = 1 TO numlugares DO
         IF D[fila,i1] < >0 THEN card : = card + 1;
     cardinal : = card
    END; {Fin de cardinal}

{-------------------------------------------------------------------------------------------------------------------}
```

```
FUNCTION mayorantedelrango (fila1, fila2, columna: integer): integer;
    {Calcula un mayorante del rango del conjunto de filas que generan la componente
     resultante de combinar «fila1» y «fila2»                                        }
 VAR i1,r : integer;
  BEGIN
   r: = columna;
   FOR i1: = 1 TO columna - 1 DO
       IF NOT (valornonulo[fila1,i1] OR valornonulo[fila2,i1]) THEN r : = r - 1;
       mayorantedelrango : = r
  END; {Fin de mayorantedelrango}
{-------------------------------------------------------------------------------------------------------------------}
FUNCTION buscaultimaocupada :integer;
          {Devuelve como resultado un puntero de la ultima fila ocupada}
 VAR i,ocupada : integer;
  BEGIN
   i : = numfilas;
   WHILE (i > = 1) AND filavacía [i] DO i : = i - 1;
   buscaultimaocupada : = i
  END; {Fin de buscaultimaocupada}
{-------------------------------------------------------------------------------------------------------------------}
PROCEDURE eliminanonulas(columna : integer);
    {Marca como vacías las filas de «A» cuyo elemento «columna» sea no nulo}
 VAR i : integer;
  BEGIN
   FOR i : = 1 TO puntero DO
       IF A[i,columna] < > 0 THEN filavacía[i] : = true
  END; {Fin de eliminanonulas}

BEGIN {Procedure anulacolumnas
       Se encarga de controlar el algoritmo de generación de componentes}
 k : = 1; {«k» es la columna de «A» que se está anulando}
 WHILE k < = numtransiciones DO
   BEGIN
    i : = 1; puntero1 : = 0;
    WHILE i < = puntero DO {Se toma la fila i-ésima}
      BEGIN
       IF (A[i,k] < > 0) AND NOT filavacía[i]
       THEN
       FOR j : = i + 1 TO puntero DO {Se toma la fila j-esima}
        BEGIN
         {Compara elemento k-ésimo de las filas «i» y «j»}
         IF (A[i,k] * A[j,k] < 0) AND NOT filavacía[j]
         THEN {Al ser de distinto signo pueden combinarse linealmente}
          BEGIN
           combinacionlineal (i,j,k);
           {Se comparan el rango y el cardinal del soporte de la componente
            resultante}
           IF cardinal(puntero1) > mayorantedelrango (i,j,k) + 1
              {Si la componente no puede ser elemental se elimina}
              THEN filavacía [puntero1] : = true
          END
        END;
```

```
          i := i + 1
        END;
    eliminanonulas(k); puntero := buscaultimaocupada; k := k + 1
  END
END; {Fin de anulacolumnas}

{-----------------------------------------------------------------------------------------------------------------}

BEGIN {Programa principal}
  leer;
  anulacolumnas;
  escribir(d, puntero, numlugares)
END. {Fin del programa principal}
```

Anexo 5

Soluciones enteras no negativas de un sistema de ecuaciones e inecuaciones lineales

Objetivo del anexo: Abordar la determinación de las soluciones enteras no negativas de un sistema de ecuaciones e inecuaciones lineales. Se presentan reglas de *transformación* que reducen el problema mencionado a la obtención de las soluciones no negativas de un sistema de ecuaciones lineales homogéneas, problema resuelto en §4.7.2.2 (en el anexo 4 se mejora el algoritmo presentado en §4.7.2.2). Los métodos de resolución que se proponen *no pretenden ser óptimos* desde el punto de vista de las prestaciones.

A.5.1 INTRODUCCIÓN

La obtención del conjunto fundamental o generador de las soluciones no negativas de un sistema de ecuaciones e inecuaciones lineales es un problema que aparece en diferentes cálculos relacionados con la determinación de propiedades de RdP. De este modo, el *cálculo de cerrojos* (§4.8.2.1), el de *trampas* (§4.8.2.2) y el de *cerrojos y trampas* (§4.8.2.3) puede ser resuelto resolviendo sistemas de inecuaciones de la forma $A^T \cdot E \geqslant 0$, donde $A_{r \times 1}$ es un vector y $E_{r \times s}$ es una matriz de enteros. Por otro lado, en §4.7.1.1 se presenta una condición suficiente para la *verificación de restricciones sobre el marcado* de una RdP (proposición 4.12) tal que si la violación de restricciones se expresa linealmente, el problema se reduce a investigar la existencia de soluciones no negativas de un sistema de inecuaciones del tipo $A^T \cdot E \geqslant F^T$.

En este anexo se reducirá la resolución de un sistema *homogéno de inecuaciones* a la resolución de sistema *homogéneo de ecuaciones*. Posteriormente se reducirá la resolución de *un sistema no homogéneo de inecuaciones* a la resolución de un sistema *homogéneo de inecuaciones*.

A.5.2 SISTEMA HOMOGÉNEO DE ECUACIONES E INECUACIONES: VARIABLES DE HOLGURA

Para reducir el sistema de inecuaciones a un sistema de ecuaciones, basta con intro-

ducir una variable no negativa de *holgura* por inecuación. Cada variable de holgura permite transformar una inecuación en ecuación.

Sean las dos transformaciones siguientes:

a) Al añadir h_i a la inecuación ($h_i \in \mathbb{N}$):

$$a_1 e_{1i} + a_2 e_{2i} + \ldots + a_r e_{ri} \geqslant 0$$

se obtiene la ecuación

$$a_1 e_{1i} + a_2 e_{2i} + \ldots + a_r e_{ri} - h_i = 0$$

b) Toda inecuación del tipo:

$$a_1 e_{1j} + a_2 e_{2j} + \ldots + a_r e_{rj} \leqslant 0$$

puede ser transformada en una inecuación como la considerada anteriormente:

$$a_1(-e_{1j}) + a_2(-e_{2j}) + \ldots + a_r(-e_{ij}) \geqslant 0$$

NOTA: También se puede escribir directamente:

$$a_1 e_{1j} + a_2 e_{2j} + \ldots + a_r e_{rj} + h_j = 0$$

En resumen, la transformación *b*) permite reducir todo sistema homogéneo de inecuaciones lineales a la forma $A^T \cdot E \geqslant 0$. La adición de variables de holgura (transformación *a*) permite la reducción de las inecuaciones a ecuaciones:

$$A^T \cdot E \geqslant 0 \Leftrightarrow A^T \cdot E - H^T = 0 \Leftrightarrow (A^T | H^T)\begin{pmatrix} E \\ \hline I \end{pmatrix} = 0$$

Este último problema fue resuelto algorítmicamente en §4.7.2.2 y en el anexo 4. Ahora bien, observando el tipo de sistema de ecuaciones que se obtiene, es fácil deducir una variante del algoritmo presentado en §4.7.2.2, válido para sistemas de la forma $A^T \cdot E \geqslant 0$ y que no necesita incrementar la dimensión del sistema con variables de holgura.

Este puede adoptar el esquema siguiente:

(1) $B := E$, $D := I_n$ {matriz unitaria de dimensión n}
(2) **Para** $i := 1$ **hasta** s {número de columnas de E}
  2.1 *Añadir* a la matriz $[D \vdots B]$ todas las filas que resulten como combinación lineal positiva de pares de filas de $[D \vdots B]$ y que anulen la i-ésima columna de B.
  2.2. *Eliminar* de $[D \vdots B]$ las filas en las que la i-ésima columna de B sea negativa.
(3) Las filas de la matriz D final representan soluciones no negativas del sistema de inecuaciones $A^T \cdot E \geqslant 0$. Entre éstas se encontrarán, tras una eventual normalización, todas las soluciones elementales.

A.5.3 SISTEMA NO-HOMOGÉNEO DE ECUACIONES E INECUACIONES

Sea el sistema no homogéneo $A^T \cdot E \geqslant F^T$. Introduciendo la variable entera y positiva $v(v \in \mathbb{N}^+)$, se puede escribir:

$$A^T \cdot E \geqslant v \cdot F^T \Leftrightarrow A^T \cdot E - v \cdot F^T \geqslant 0 \Leftrightarrow (A^T \vdots v)\begin{pmatrix} E \\ \hline -F^T \end{pmatrix} \geqslant 0$$

Una vez resuelto el sistema homogéneo de inecuaciones (según lo postulado en §A.5.2.), aquellas soluciones A/v, en las que $v \geqslant 1$ y A/v sea un vector de enteros, comprenderán al conjunto de las soluciones elementales del sistema original, $A^T \cdot E \geqslant F^T$.

Anexo 6

Léxico sobre redes de Petri

La introducción de este léxico se ha realizado habida cuenta la confusión terminológica existente en la literatura sobre la teoría y aplicaciones de las redes de Petri. Las principales razones de la mencionada confusión se pueden agrupar en las tres siguientes:

(1) La *novedad* de la teoría y de sus aplicaciones, puesto que, a pesar de datar de 1962 el primer trabajo, el desarrollo de la disciplina se ha realizado prácticamente a partir de la década de los setenta y muy especialmente de su segundo lustro.

(2) El *carácter multidisciplinar* de la teoría, así como la diversidad de sus aplicaciones. De este modo, el desarrollo de la teoría ha ido surgiendo en base a necesidades sentidas por especialistas del hardware y del software de los computadores, de automatismos lógicos industriales, de organización industrial,... Además, en la teoría de redes de Petri, han trabajado matemáticos de formaciones muy diversas.

(3) La ausencia de obras que unifiquen/definan la (una) terminología.

La mencionada confusión terminológica alcanza hasta la definición del mismo concepto de red de Petri. Así, encontraremos que esta definición varía dentro de las ocho posibilidades siguientes:

$$\left\{\begin{array}{c}\text{red de Petri ordinaria}\\ \text{ó}\\ \text{red de Petri generalizada}\end{array}\right\} \quad \left\{\begin{array}{c}\text{no marcada}\\ \text{ó}\\ \text{marcada}\end{array}\right\} \quad \left\{\begin{array}{c}\text{con capacidad infinita}\\ \text{ó}\\ \text{con capacidad finita}\end{array}\right\}$$

De esta forma, en [PETE 77] o [AGER 79] el concepto de red de Petri es el de RdP ordinaria/no marcada/con capacidad infinita (no existe el concepto de capacidad). En el otro extremo, en [BRAU 80], se define con el mismo vocablo el concepto de red de Petri generalizada/marcada/con capacidad finita. En general no existe una contradicción entre los conceptos representados por un mismo vocablo, pero su nivel de generalidad varía. Así, en [PETE 77] se define el concepto de red de Petri conservativa como aquella en la que el número de marcas es constante a lo largo de cualquier evolución. Este concepto ha sido definido en el texto (def. 4.22) de for-

ma más general, puesto que se considera la constancia de una suma ponderada de las marcas.

En la traducción/adaptación de términos, hemos querido evitar, en la medida de lo posible, los anglicismos y galicismos. Así, por ejemplo, se han introducido las denominaciones de lugar y marcado.

Por último, queremos resaltar que la diferencia conceptual entre el término Grafo de Estados en un contexto clásico y en el de las redes de Petri, nos impulsó a introducir la denominación de Grafo (de Estados) Reducido. Este término sustituye al clásico concepto de Grafo de Estados o Grafo de Transición de Estados, evitándose de este modo la ambigüedad. La justificación de la acuñación del nuevo término, Grafo Reducido, está implícita en el capítulo 1, §1.4.

El léxico que se presenta a continuación no es exhaustivo. No obstante, además de los vocablos básicos de la teoría de redes de Petri, se han incluido aquellos específicos a la consideración de las redes como modelos de descripción de *sistemas lógicos*. Todos los vocablos se encuentran definidos en el texto, pudiéndose proceder a su localización en base a la información, contenida entre paréntesis, que se le haya asociado (*Nota*: **d** ≡ definición, **f** ≡ figura, § ≡ párrafo). Los vocablos se encuentran acompañados de sus sinónimos más usuales.

A

acontecimiento (d.2.14) [evento]	event	événement
alcanzable (marcado) (d.2.12)	reachable (marking)	atteignable (marquage)
arco (d.2.2.1.1)	arc	arc
arco inhibidor (d.2.5.2.3)	inhibitor arc	arco inhibiteur [arc complementé]
atribución (f.1.18)	meet	attribution
avance sincrónico (d.4.12)	synchronic lead	avance synchronique
avance sincrónico ponderado (d.4.13)	weighted synchronic lead	avance synchronique ponderée

B

binaria (RdP)(§4.2.3) [sana, segura]	safe (PN)	sauf (RdP) [sain, binaire]

C

cerrojo (d.4.27)	deadlock	verrou
cerrojo mínimo (§4.8.2)	minimal deadlock	verrou minimal
cíclica (RdP)(d.4.5)	----------	réinitialisable [propre]
componente canónica (d.4.25) [invariante canónico]	minimal (vector ordering) invariant	composante canonique
componente conservativa (d.4.23) [invariante de lugares]	p-invariant component [conservative component]	composante conservative [p-flot]

componente elemental (d.4.26) [invariante elemental]	minimal support invariant	composante élémentaire
componente repetitiva (§4.7.2.1) [invariante de transiciones]	t-invariant component [consistent component]	composante consistente [comp. repetitive, t-flot]
condición envolvente de un lugar (d.2.16)	----------	monôme enveloppe d'une place
condición envolvente de un marcado (d.3.4)	----------	monôme enveloppe d'un marquage
condición externa (d.2.13)	----------	condition externe
conflicto efectivo (d.4.10)	effective conflict	conflit effectif
conflicto estructural (d.4.9)	structural conflict [topologycal conflict, conflict]	conflict structurel [conflit topologyque]
conforme (§1.5.3.2)	----------	conforme
conjunción (f.1.18)	wait	jonction
conjunto de lugares de entrada de *t*(d.2.4)	pre-set of *t*	ensemble de transitions d'entrée de *t*
conjunto de lugares implicantes (d.3.1)	implicating places set	ensemble de places impliquantes
conjunto fundamental (de componentes conservativas) (d.4.26) [generadores]	set of generators (for the p-invariants)	ensemble fondamental (de composant conservatives) [generateurs]
conj. de lugares de salida de *t* (d.2.4)	post-set of *t*	ensemble de places de sortie de *t*
conj. de marcados alcanzables (d.2.12)	forward marking class	ensemble de marquages atteignables
conj. de transiciones de entrada de *p* (d.2.4)	pre-set of *p*	ensemble de transitions d'entrée de p
conj. de transiciones de salida de *p* (d.2.4)	post-set of *p*	ensemble de transitions de sortie de *p*
conj. de secuencias de disparo (d.2.10)	set of firable sequences	ensemble de séquences de franchissement
conservativa (RdP) (d.4.22)	conservative net [p-invariant net]	conservatif (RdP)

D

disparar (una transición sensibilizada) (d.2.9)	firing (of a enabled transition)	franchir ou mettre à feu (une transition validée)
distribución (f.1.18)	split	distribution
dual (RdP) (d.4.23)	dual (PN)	dual (RdP)

E

ecuación de estado de una RdP (§2.2.2)	state equation of a PN	equation d'état d'un RdP

estructuralmente limitada (RdP) (d.4.8)	structurally bounded (PN)	structurellement borné (RdP)
estructuralmente viva (RdP) (d.4.4)	structurally live (PN)	structurellement vivant (RdP)
evento (d.2.14) [acontecimiento]	event	événement
exclusión mutua (d.4.11) (lugares en)	mutual exclusion	exclusion mutuelle

G

grafo de cobertura (§4.3) [grafo de alcanzabilidad]	coverability graph [reachability graph]	graphe de couverture [graphe d'atteignabilité]
grafo de estados (d.2.17) [máquina de estados]	state graph [state machine graph]	graphe d'état [machine d'état]
grafo de marcados (d.4.14)	marking graph	graphe des marquages
grafo de sincronización (d.2.18)	synchronization graph	graphe de synchronisation
grafo marcado (d.2.18)	marked graph	graphe marqué

I

incidencia previa (función) (d.2.1)	input function [pre-function, backward function]	fonction d'incidence avant
incidencia posterior (función) (d.2.1)	output function [post-function, forward-function]	fonction d'incidence arrière

L

libre de monopolio (RdP) (d.5.1)	monopoly free	libre de monopole
limitada (RdP) (d.4.7) [acotada]	bounded (PN)	borné (RdP)
lugar (d.2.1)	place	place
lugar ascendiente de una subRdP (d.2.16)	----------	place ascendante d'un sous RdP
lugar complementario (d.2.22)	complementary place	place complementaire
lugar de sincronización (§.9.6.2/3)	synchronization place	place de synchronisation
lugar descendiente de una subRdP (d.4.16)	----------	place descendante d'un sousRdP
lugar fuente (d.3.6/7)	source place	place source
lugar identidad (§3.3.2)	identity place	place identité
lugar implicante (d.2.1)	implicating place	place impliquante
lugar implícito (d.3.1) [lugar redundante]	redundant place [implicit place]	place implicite [place redondante]
lugar *k*-limitado (d.4.6)	*k*-bounded place	place *k*-bornée
lugar representante de una transición (§9.6.2/3)	----------	place pivot ou place clé d'une transition

lugar sumidero (d.3.7)	sink place	place puits
lugar sustituible (d.4.18)	substituable place	place subtituible
lugares compatibles (d.3.3)	----------	places compatibles [places fusionnables]
lugares directamente fusionables o equivalentes (d.3.2)	----------	----------

M

macrografo asociado a una RdP (§7.4.3)	----------	macrographe associé à un RdP
macrolugar (d.4.17)	macroplace	macroplace
máquina de estado (d.2.13)	state-machine	machine d'état
marca (§2.2.1.2)	token (marker)	marque
marcado (d.2.6)	marking	marquage
marcado inicial (d.2.7)	initial marking	marquage initial
marcado superior (§4.4.1)	higher marking	marquage supérieur
matriz de incidencia previa (§2.2.1.1)	pre-incidence matrix	matrice d'incidence avant
matriz de incidencia posterior (§2.2.1.1)	post-incidence matrix	matrice d'incidence arriére

N

nudo 0 (f.1.18)	OR-node	noeud OU
nudo Y (f.1.18)	AND-node	noeud ET

P

parcialmente viva (RdP) (d.4.3) [seudo-viva]	pseudo-live	partiellement bloquée [pseudo-vivant]
peso activo de una red marcada (§6.5.3.1)	----------	poids actif d'un réseau marqué
peso del arco (§2.2.1.1)	arc multiplicity [arc token width]	poids de l'arc
peso total de una red (§6.5.3.1)	----------	poids total d'un réseau
propiedad cerrojo-trampa (d.4.29)	deadlock-trap property	propriété verrou-trappe

R

RdP con arcos inhibidores (d.2.23)	PN with zero-test	RdP avec test de zéro [avec arcs inhibiteurs]
RdP con capacidad (d.2.21)	capacity PN	RdP à capacité

RdP coloreada (§2.5.2.5)	coloured PN	RdP coloré
RdP generalizada (d.2.1)	generalised PN	RdP généralisé
RdP interpretada (d.2.15)	interpreted PN	RdP interpreté
RdP *k*-limitada (d.4.7)	*k*-bounded PN	RdP *k*-borné
RdP libre elección (d.2.19)	free-choice PN	RdP à choix libre
RdP marcada (d.2.7)	marked PN	RdP marqué
RdP ordinaria (d.2.2) [red de Petri]	Petri net	RdP ordinaire
RdP pura (d.2.3)	pure PN [self-loop-free PN]	RdP pure
RdP simple (d.2.20)	simple PN	RdP simple
RdP temporizada (§2.5.2.6) (d.2.20/20 bis)	timed PN	RdP temporisé
repetitiva (RdP) (d.4.21)	consistent PN [*t*-invariant net]	RdP répetitif [RdP consistant]

S

secuencia de disparos (d.2.10) [sec. de franqueos]	firing sequence	séquénce de franchissement [séquénce de mise à feu]
selección (f.1.18)	branch	sélection
soporte de una componente conservativa (d.4.24)	support of a p-invariant	support d'une composante conservative
subred de Petri (d.2.5)	Petri subnet	sous-réseau de Petri
subRdP potencialmente reducible a un lugar (d.4.15)	----------	sous-RdP pontentiellement reductible à une place
sustitución de un lugar (§5.5.1)	substitution of a place	substitution d'une place
sustitución de una transición (f. 5.12)	substitution of a transition	substitution d'une transition

T

trampa (d.4.28)	trap	trappe [piège]
trampa mínima (§4.8.2.2)	minimal trap	trappe minimale
transición (d.2.1)	transition	transition
transición conmutador (§2.5.2.4)	switch transition	----------
transición fuente (d.3.5)	source transition	transition source
transiciones idénticas (d.4.19)	similar transitions [identical transitions]	transitions identiques
transición identidad (d.4.20)	identity transition	transition identité
transición O-EXCLUSIVO (f.2.22)	EXOR-transition	transition OU-EXCLUSIF
transición receptiva (§2.4.1)	----------	transition receptive

transición sensibilizada (d.2.8) [validada]	enabled transition	transition sensibilisée [validée]

V

vector característico de una secuencia de disparos (d.2.11)	characteristic vector of a firing sequence	vecteur caractérisque d'une séquence de franchissements
vector de pesos (§6.5.3.1)	weights vector	vecteur des poids
viva (transición) (d.4.1)	live (transition)	vive (transition)
viva (RdP) (d.4.2)	live (PN)	vive (RdP)
viva (RdP marcada) (d.4.2)	live (marked PN)	vivant (RdP marqué)

Bibliografía

La relación bibliográfica que sigue cubre la temática abordada pero no es exhaustiva. La selección realizada se basa, entre otros criterios, en el fácil acceso a la referencia y/o su interés histórico-conceptual. Referencias adicionales sobre la teoría y aplicaciones de las redes de Petri pueden encontrarse a partir de:

(1) Los trabajos relacionados. En particular [AGER 79], [BRAU 80], [MOAL 80], [PETE 81].

(2) La «*Bibliography of Net Theory*» compilada por E. PLESS & H. PLUNNECKE y editada por la Gesellschaft für Mathematik und Datenverarbeitung. (GMD-ISF Postfach 1240, D-5205, SANKT AUGUSTIN L. ALEMANIA.)

(3) La «*Newsletter on Petri Nets and related System Models*». (LEHRSTUHL FUR INFORMATIK II. Rheinlschnestfalische. Technische Hochschule Aachen. Büchel 29-31. 5100 AACHEN. ALEMANIA.)

REFERENCIAS CAPÍTULO 1

El análisis y diseño de sistemas secuenciales es un tema clásico ampliamente tratado en la literatura. Entre otros textos se encuentran:

a) en español: [HILL 78], [MAND 79], [ALDA 80].
b) en inglés: [MCCL 65], [UNGE 69], [TORN 72], [FLET 80].
c) en francés: [PERR 67], [MANG 78].

Los trabajos de introducción-motivación hacia las redes de Petri no son muy numerosos. Entre otros cabe citar [DACL 76], [PETE 77], [AGER 79], [SILV 82a].

REFERENCIAS CAPITULO 2

El trabajo pionero sobre las redes que nos ocupan es la tesis doctoral de C. Petri [PETR 62] aunque es obligado constatar que la definición original dista bastante de las aceptadas hoy en día. La terminología básica sobre redes de Petri se recoge, con diferentes variaciones, en [BRAU 80], [MOAL 80] y [PETE 81a].

Dos trabajos claves sobre subclases de redes de Petri son [COMM 71] y [HACK 72]. En el segundo de ellos se introducen las redes de libre elección así como las redes simples. Entre las principales extensiones de las redes de Petri se pueden citar los trabajos [AGER 73] (redes con arcos inhibidores), [GENR 79] (redes predicado-transición) y [JENS 81] (redes coloreadas).

Otras referencias: Insertas en el texto del capítulo.

REFERENCIAS CAPÍTULO 3

En [DACL 76] se aborda la simplificación de RdP por eliminación de lugares implícitos. Desde un punto de vista algorítmico, la técnica empleada es poco eficiente en general, puesto que se basa en la obtención del GR equivalente a la RdP. La presentación realizada en este capítulo deriva de [SILV 80c] (resultados básicos) y [MART 81b] (fundamentos teóricos del algoritmo).

La fusión de lugares se aborda en [GIRA 73] (para los GR) y [DACL 76]. La presentación realizada en este texto deriva de las anteriores referencias. La simplificación de lugares fuente se introduce en [SILV 77] y [SILV 78]

Otras referencias: [BERT 78] caracteriza, para redes no limitadas, el concepto de lugar implícito. El estudio de compatibles puede encontrarse en multitud de textos; por ejemplo: [HILL 78], [PERR 67], [TORN 72]. En [DADD 76b] se considera la simplificación de redes bajo una óptica distinta a la adoptada en el texto.

REFERENCIAS CAPÍTULO 4

Al presentarse en este capítulo el núcleo de la teoría de redes de Petri, las referencias que sirven de fuente son muchas y, desgraciadamente, muy dispersas.

Los textos [BRAU 80] y [GIRA 82], permiten cubrir la mayor parte del material, pero su presentación es, en general, muy abstracta. En [PETE 81a] se consideran sólo las técnicas de análisis basadas en el árbol de alcanzabilidad (no presentadas en este texto) y algebra lineal, aunque el tratamiento del segundo grupo es muy superficial. En ningún caso se presentan de forma sistemática algoritmos que hagan operativos los resultados que permiten el análisis.

Las referencias principales, clasificadas por técnicas de análisis, son:

(1) *Enumeración*: [KARP 69], [BERT 78].
(2) *Reducción*: [BERT 78, 79a, 79b], [SZLA 77], [BOUS 78], [SILV 81]
(3) *Algebra lineal y RdP*: [LAUT 74], [LIEN 76a, 76b], [MURA 77a] [MEMM 78b, 79a, 79b], [SIFA 78, 79b], [RAMC 73], [MART 81b].
(4) *Cerrojos y trampas*: [COMM 71], [HACK 72], [MEMM 78a, 78b],[JANT 79], [SIFA 79b], [TOUD 81].

REFERENCIAS CAPÍTULO 5

El impacto del tiempo y de la interpretación sobre el análisis de redes autónomas fue planteado en [GHOS 77] y [MOAL 78], respectivamente.

Los aspectos metodológicos en la descripción de sistemas se basan en los conceptos de estructuración-modularización, que aparecen en diversas ramas de la ingeniería. En particular, dentro del ámbito de la programación de computadores pueden considerarse las referencias [DAHL 72], [DIJK 76], [LEDG 75].

La aplicación de los conceptos de estructuración y de modularidad a las redes de Petri ha sido tratada, por ejemplo, en [VALE 76, 78a, 79], [SUZU 80] [ANDR 80, 81b].

Otras referencias: [BRAU 80], [GIRA 82], [MOAL 81].

REFERENCIAS CAPÍTULO 6

La realización asíncrona ha llevado a la definición de una célula de memoria, funcionalmente idéntica a la CUSA [DAVI 77, 80]. Los métodos, de cableado propuestos difieren de los anteriores; en particular, el método de transferencia impulsional conduce a un cableado análogo al propuesto en [DAVI 80] pero en el que pueden eliminarse (bajo ciertas hipótesis) los retrasos en la célula.

Metodológicamente, el análisis de aleatoriedades presentado se basa en [DAVI 80].

La síntesis de células síncronas es un ejercicio clásico de transformación de biestables. (Véase, por ejemplo [HILL 78], [MANG 78].)

Otras referencias: [DACL 76], [MARI 76], [SIFA 79a], [COUR 80].

REFERENCIAS CAPÍTULO 7

Una interesante introducción a la lógica matricial y su aplicación al diseño de sistemas combinacionales y secuenciales se presenta en [LIPP 76]. La optimización de las relaciones basadas en PLA se aborda, entre otras referencias, en [FLEI 75], [AUGI 78], [ROTH 78] y [PERE 80]. La última de las anteriores considera la optimización desde el punto de vista del constructor de PLAS, no como usuario de PLAS estándar. La minimización de la longitud de una palabra por codificación independiente de campos ha sido enfocada siguiendo a [JAYA 76]; una más precisa caracterización del concepto de clase principal ha permitido obtener mejores soluciones que en el trabajo mencionado.

Uno de los primeros trabajos sistemáticos sobre la realización de sistemas secuenciales con ROM es [CLAR 73]. La referencia [ANDR 81a] es una interesante monografía sobre microprogramación.

La idea de descomponer una RdP binaria en GR para una posterior realización microprogramada se propone en [ANDR 76] de forma heurística. En [MART 81a] y [AUGI 79] se introducen las técnicas de descomposición presentadas en §7.4.3.1. y 2, respectivamente.

Otras referencias: [AYAC 77], [DADD 76a], [LEUN 77], [HOZJ 79], [MANG 82].

REFERENCIAS CAPÍTULO 8

La información concerniente a los Autómatas Programables (estructura, funcionamiento y lenguajes de programación) ha sido recogida de un estudio sistemático de

los equipos ofrecidos por los constructores, así como de diversos artículos de síntesis aparecidos, entre otras, en las revistas *Control Engineering* y *Automatique e Informatique Industrielle*. Una primera versión de gran parte de este capítulo se encuentra en [SILV 78].

Referencias adicionales son [GSLA 74], [MICH 79], [SILV 77], [TOUL 78] Y [THÉL 80]. Las tres últimas se ocupan de la programación de RdP con autómatas programables.

Otras referencias de interés: [HOLZ 69] (referencia histórica), [HEUM 75], [MORA 75], [CATI 79].

REFERENCIAS CAPITULO 9

La definición de sistemas especiales para la simulación de RdP binarias ha motivado numerosos trabajos. Las referencias [DACL 76], [DEFR 79] y [SILV 80a] presentan equipos en los que se ha diseñado especialmente el hardware o se ha añadido hardware especial a un microcomputador de propósito general. Utilizando exclusivamente técnicas software sobre computadores de propósito general, se encuentran, entre otras, las referencias [SILV 78], [SILV 79c], [TOUR 76], [CHOC 80], [COUR 80] y [SILV 82c].

Desde un punto de vista conceptual, la presentación del capítulo sigue, en gran parte, el esquema adoptado en [SILV 82c].

La codificación y simulación de grafos reducidos (de estado) es un tema clásico, abordado, entre otras referencias modernas, en [PRAT 75], [ALAB 75] y [LAND 79].

La referencia [TAFA 79] presenta un interpretador de RdP con capacidad, extensión no considerada en este texto.

AGER 73 AGERWALA T., FLYNN M.: Comments on capabilities, limitations and correctness of Petri nets. *First Annual Symposium on Computer Architecture*. Florida, pp. 81-86.

AGER 79 AGERWALA T.: Putting Petri nets to work. *Computer*, December, pp. 85-94

ALAB 75 ALABAU A., FIGUERAS J.: Some aspects of the implementations on a microcomputer of secuential systems. *Symp.* MIMI-75. Zürich, pp. 162-166.

ALDA 80 ALDANA F., ESPARZA R., MARTÍNEZ P. M.: *Electrónica industrial: Técnicas digitales.* Marcombo. Barcelona.

ANDR 76 ANDRÉ C., BOERI F., MARÍN J.: Synthèse et réalisation des systèmes logiques à évolutions simultanées. *Rairo-Automatique*, vol. 10, n. 4, Avril, pp. 67-86.

ANDR 79 ANDRE C., ARMAND P., BOERI F.: Synchronic relations and applications in parallel computation. *Digital Processes*, vol. 5, n. 2, pp. 99-113.

ANDR 80 ANDRE C.: Behaviour of a place-transition net on a subset of transitions. *First European Workshop on Applications and Theory of Petri Nets.* Strasbourg, September. (Publicado en [GIRA 82], pp. 131-135.)

ANDR 81a ANDREWS M.: *Principles of firmware engineering in microprograma control.* Pitman Publishing Limited. London.

ANDR 81b ANDRE C.: Use of behaviour equivalence in Place-Transition net analysis. *Second European Workshop on Application and Theory of Petri Nets.* Bad Honnef, September. (Publicado en [GIRA 82], pp. 241-250.)

AUGI 78 AUGIN M., BOERI F., ANDRE C.: An algorithm for designing multiple boolean functions: Application to PLA's. *Digital Processes*, vol. 3, n. 4, pp. 215-230.

AUGI 79 AUGIN M., BOERI F., ANDRE C.: New design using PLA's and Petri nets. *Measurement and Control Internations Symp.* MECO-79. Athens, June.

AYAC 77 AYACHE J. M., LE DANOIS P.: Synthesis of logic systems with PLA's. *Journées d'étude: Logique cablée ou logique programmée.* Lausanne, Mars, pp. 89-95.

BAER 73 BAER J. L.: A survey of some theoretical aspects of multiprocessing. *Computing Surveys*, March, pp. 31-80.

BERG 73 BERGE C.: *Graphes et Hypergraphes.* Dunod. Paris.

BERT 78 BERTHELOT G.: Vérfication des réseaux de Petri. *Thèse Doc, 3éme Cycle*, Université P. et M. Curie. Paris, Janvier.

BERT 79a BERTHELOT G.: Preuve de non blocage des programmes parallèles par réduction des rèseaux de Petri. *First European Conference on Parallel and Distributed Processing.* Toulouse, February, pp. 251-259.

BERT 79b BERTHELOT G., ROUCAIROL G., VALK R.: Reductions of nets and parallel programs. *Advanced Course on General Net Theory of Processes and Systems.* Hamburg, October. (Publicado en [BRAU 80], pp. 277-290.)

BERT 79c BERTHOMIEU B.: Analyse structurelle des réseaux de Petri: Méthodes et outils. *Thèse Doc. Ingénieur*, Univ. Paul Sabatier. Toulouse, Septembre.

BLAN 79a BLANCHARD M.: Automatismes logiques: Grafcet ou réseaux de Petri?. *Le Nouvel Automatisme*, Mai, pp. 45-52.

BLAN 79b BLANCHARD M.: *Le Grafcet.* Ed. Cepadues. Toulouse.

BOSS 79 BOSSY J. C., BRARD P., FAUGERE P., MERLAND C.: *Le Grafcet.* Educalivre, Paris.

BOUS 78 Boussin J.: *Synthesis and analysis of logic automation systems.* IFAC 7th Triennal World Congress, Helsinki, June. (Pergamon Press.)

BOUT 76 — BOUTE R. T.: The binary decision marchine as programmable controller. *Euromicro Newsletter*, vol. 2, N. 1, January, pp. 16-22.

BOWM 70 — BOWMAN M., MCVEY E. S.: A method for the fast approximate solution of large prime implicant charts. IEEE *Trans. on Computers*, vol. C-19, n. 2, February, pp. 169-173.

BOYE 79 — BOYER H., NORBERT M., PHILIPPE R.: *Automatisation moderne: Grafcet et Automates Programables*. Ed. de la Capitelle, Uzes.

BRAM 83 — BRAMS G. W.: *Réseaux de Petri. Theorie et pratique (2 tomes)*. Masson, Paris.

BRAU 80 — BRAUER W. (EDITOR): Net theory and applications. *Lecture Notes in Computer Science 84*, Springer-Verlag. Berlin.

BURG 77 — DE BURGOS J.: *Curso de álgebra y geometría*. Ed. Alhambra. Madrid.

CAFE 80 — CAFERRA R., SIMONET M.: Détermination des configurations bloquantes dans les réseaux de Petri. *Congrés* AFCET-Informatique. Nancy, Novembre.

CATI 79 — CATIER E.: Fiables, modulaires et bon marché, les automates programmables industriels prennent le relais. *Automatique et Informatique Industrielle*, n. 73, Janvier, pp. 16-25.

CLAR 73 — CLARE C. R.: *Designing logic systems using state machines*. Mc Graw-Hill. New York.

COMM 71 — COMMONER F., HOLT A. W., EVEN S., PNUELLI A. Marked directed graphs. *Journal of Computer and Systems Science 5*, October, pp. 511-523.

COUR 77 — COURTOIS P. J., GEORGES J.: On starvation prevention. *Rairo-Informatique*, vol. 11, n. 2, pp. 127-141.

COUR 80 — COURVOISIER M., VALETTE R.: *Systèmes de commande en temps réel*. Editions SCM. Paris.

CRES 77 — CRESPI-REGHIZZI S., MANDRIOLI D.: Petri nets and Szilard languages. *Information & Control*, vol. 33, n. 2, February, pp. 177-192.

CROC 75 — CROCUS: *Systèmes d'explotation des ordinateurs*. Dunod, Paris.

CHEZ 79 — CHEZALVIEL-PRADIN B.: Un outil graphique interactif pour la vérification des systèmes à évolution parallèle décrits par réseaux de Petri. *Thèse Doc. Ingénieur*, Univ. Paul Sabatier. Toulouse, Decembre.

CHOC 80a — CHOCRON D.: Un système de programmation par RdP de Contrôleurs Industriels. *Master on Computer Science*, Montreal.

CHOC 80b — CHOCRON D., CERNY E.: A Petri-net based industrial sequencer. IEEE *Symp. on Industrial Control and Instrumentation: Application of mini and microcomputers*, Philadelphia. IEEE Press, CH 1551-1/80, pp. 18-22.

DACL 72 — DACLIN E.: Synthèse des circuits séquentiels: De la théore à la pratique. Rairo-Automatique, n. 2, Mai, pp. 3-58.

DACL 76 — DACLIN E., BLANCHARD M.: *Synthèse des systémes logiques*. Ed. Cepadues, Collection Sup-Aero. Toulouse.

DADD 76a — DADDA L.: On the simulation of Petri nets as a control tool. *Euromicro Newsletter*, vol. 2, n. 1, January, pp. 38-45.

DADD 76b — DADDA L.: The synthesis of Petri nets for controlling purposes and the reduction of their complexity. *Euromicro-76*. Amsterdam, October, pp. 251-259.

DAHL 72 — DAHL O. J., DIJKSTRA E. W., HOARE C. A. R.: *Structured programming*. Academic Press. London.

DANT 63 DANTZIG G. B.: *Linear programing and extensions*. Princenton University Press.

DAVI 77 DAVID R.: Modular design of asynchronous circuits defined by graphs. IEEE *Trans. on Computers*, vol. C-26, n. 8, August, pp. 727-737.

DAVI 80 DAVID R., TELLEZ R., MITRANI E.: Emploi de celules universelles pour la synthèse des systèmes asynchrones décrits par réseaux de Petri. *Digital Processes*, vol. 6, pp. 185-198.

DEFR 79 DEFRENNE J., MANESSE G., TOULOTTE J. M.: Gestion rapide des feseaux de Petri par microprocesseur. *Symposium* MIMI-79. Zurich, May, pp. 62-66.

DELL 77 DELLANDE B. W.: A programmable logic controller on chip. The latest tool for control engineers. *Control Engineering*, August, pp. 38-39.

DENN 73 DENNIS J. B.: Concurrency in software systems. *Advanced Course in Software Engineering* (Ed. Baer J. L.). Springer-Verlag. Berlin. pp. 111-127.

DIJK 71 DIJKSTRA E. W.: Hierarchical ordering of sequential processes. *Acta Informatica*, vol. 1, pp. 115-138.

DIJK 76 DIJKSTRA E. W.: *A discipline of programming*. Prentice-Hall Series in Automatic Computation. Englewood Cliffs, New Jersey.

FLEI 75 FLEISHER H., MAISSEL L. I.: An introduction to array logic. IBM *J. Res. Develop.*, March, pp. 98-109.

FLET 80 FLETCHER W.: *An engineering approach to digital design*. Prentice-Hall. Englewood Cliffs, New Jersey.

FLOR 81 FLORIN G., NATKIN S.: Evaluation based upon stochastc Petri nets of the maximum throughput of a full duplex protocol. *Second European Workshop on Application and Theory of Petri Nets*. Bad Honnef, September. (Publicado en [GIRA 82], pp. 280-288).

GENR 79 GENRICH H. J., LAUTENBACH K.: The analysis of distributed systems by means of predicate/transition nets. *Semantics of Concurrent Computation*. Evian. Lecture Notes in Computer Sciences, vol. 70, Springer-Verlag, pp. 123-146.

GHOS 77 GHOSH S.: Some comments on timed Petri nets. *Journées d'Etude* AFCET: Réseaux de Petri. Paris, Mars, pp. 151-163.

GIRA 73 GIRARD P., NASLIN P.: *Construction des machines séquentielles industrielles*. Dunod, Paris.

GIRA 77 GIRAULT C.: Réseaux de Petri et synchronisation des processus. *Journées d'Etude* AFCET: Programmation Globale des Synchronisations dans les Applications en Temps Réel. Paris, Novembre. (Publicado en Temps Réel, n. 2, 1977, pp. 153-171.)

GIRA 82 GIRAULT C., REISING W. (ED.): *Application and theory of Petri nets*. Informatik Fachberichte, n. 52, Springer-Verlag. Berlin.

GRAS 66 GRASSELLI A.: A note on the derivation of maximal compatibility classes. *Calcolo*, vol. 3, n. 2, pp. 165-176.

GRAS 70 GRASSELLI A., MONTANARI U.: On the minimization of Read-Only-Memories in microprogrammed digital computers. IEEE *Trans. on Computers*, vol. C-19, November, pp. 1111-1114.

GRIE 75 GRIES D.: *Construcción de compiladores*. Paraninfo. Madrid.

GSLA 74 GROUPE SYSTEMES LOGIQUES DE L'AFCET: Les automates programmables. *Journée d'Etude* AFCET. Toulouse, Novembre.

GSLA 75 GROUPE SYSTEMES LOGIQUES DE L'AFCET: Evolution dans la conception des systèmes logiques. *Journée d'Etude* AFCET. Nive, Juin.

GSLA 77 — GROUPE SYSTEMES LOGIQUES DE L'AFECT, COMMISSION DE NORMALISATION: *Normalisation de la représentation du cachier des charges d'un automatisme logique.* Informe publicado integramente en «Automatique et Informatique Industrielle» (Novembre et Décémbre 1977) y en «Automatisme» (Mars-Avril, 1978).

HACK 72 — HACK M.: Analysis of production schemata by Petri nets. *Master of Science*, MAC TR. 94, MIT, Cambridge Mass,. Dept. Electrical Engineering, February.

HACK 75 — HACK M. H. Y., PETERSON J. L.: Petri nets and languages. *Conference on Petri Nets and Related Methods,* MIT, MAC TR. 94, Cambridge, Mass., July.

HAYE 81 — HAYES A. B.: Stored state asynchronous sequential circuits. IEEE *Trans. on Computers*, vol. C-30, n. 8, August, pp. 596-600.

HEUM 75 — HEUMANN G. W., SINKER R.: European programmable logic controllers. *Control Engineering*, September, pp. 66-67.

HILL 78 — HILL F. J., PETERSON G. R.: *Teoría de conmutación y diseño lógico.* Limusa. México.

HOFF 73 — HOFFMAN K., KUNZE R.: *Algebra lineal.* Prentice-Hall International. Madrid.

HOLZ 69 — HOLZER J. M., CHACEM D. E., RICKETTS A. W.: The black box: Programmable logic for repetitive control. *Control Engineering*, May, pp. 61-65.

HOZJ 79 — HOZJAJENCK J.: The use of Petri net models and FPLA's in programmable hardwired system design. *Euromicro-79.* Stockholm, North-Holland Publ. Company, pp. 259-269.

HUSS 70 — HUSSON S.: *Microprogramming: principles and practices.* Prentice-Hall. Englewood Cliffs, New Jersey.

JANT 79 — JANTZEN M., VALK R.: Formal properties of Place-Transition nets. *Advanced Course on General Net Theory of Processes and Systems.* Hamburg, October. (Publicado en [BRAU 80], pp. 165-212.)

JAYA 76 — JAYASRI T., BASU D.: An approach to organizing microinstructions which minimices the width of control store words. IEEE *Trans. on Computers,* vol. C-25, n. 5, May, pp. 514-521.

JENS 81*a* — JENSEN K.: Coloured Petri nets and the invariant method. *Theoret ical Computer Science* 14, pp. 317-336.

JENS 81*b* — JENSEN K.: How to find invariants for colored petri nets. *Mathematical Foundations of Computer Science.*

KARP 66 — KARP R. M., MILLER R. E.: Properties of a model for parallel computation: Determinacy, termination, queuing. SIAM 5. *Appl. Math*, vol. 14, n. 6, November, pp. 1390-1411.

KARP 69 — KARP R., MILLER R.: Parallel program schemata. *Journal of Computer and System Science,* vol. 3, n. 4, May, pp. 167-195.

KATZ 77 — KATZAN, H.: *Microprogramming primer.* McGraw-Hill. New York.

KAUF 76 — KAUFFMAN A.: *Puntos y flechas. Teoría de los Grafos.* Marcombo. Barcelona.

KOSA 73 — KOSARAJU S. R.: Limitation of Dijkstra's semaphore primitives and Petri nets. *Operating Systems Review*, vol. 7, n. 4, October, pp. 122-126.

KUNT 68 — KUNTZMAN J.: *Algèbre de Boole.* Dunod, Paris.

LAND 79 — LANDAU J. V.: State description techniques applied to industrial machine control. *Computer*, February, pp. 32-40.

LAUT 74 — LAUTENBACH K., SCHMID H. A.: Use of Petri nets for proving correctness of concurrent process systems. IFIP-74, North Holland Pub. Comp., pp. 187-191.

LAUT 79 — LAUTENBACH K., THIAGARAJAN P. S.: Analysis of a ressource allocation problem using Petri nets. *First European Conference on Parallel and Distributed Processing.* Toulouse, February, pp. 260-266.

LEDG 75 — LEDGARD H. F., MARCOTTY M.: A genealogy of control structures. *Comm. of the ACM*, vol. 8, n. 11, November, pp. 629-639.

LEUN 77 — LEUNG K. C., MICHEL C., LE BEUX P.: Logical systems using PLA's and Petri nets. Programmable hardwired systems. *Information Processing.* Montreal, August, North-Holland Publishing Company, pp. 607-611.

LIEN 76a — LIEN Y. G.: Termination properties of generalized Petri nets. *Siam J. Computing*, vol. 5, n. 2, June, pp. 251-256.

LIEN 76b — LIEN Y. G.: A note on transition systems. *J. Information Science*, vol. 10, n. 4, June, pp. 251-65.

LIPP 76 — LIPP H. M.: ARRAY LOGIC. *Euromicro-76*, North-Holland. Venice, October, pp. 57-64.

LIPT 75 — LIPTON R.: The reachability problem and the boundness problem for Petri nets are exponential-space hard. *Conf. on Petri Nets and Related Methods.* MIT Cambridge, Mass., July.

MAND 79 — MANDADO E.: Sistemas electrónicos digitales. Marcombo. Barcelona.

MANG 78 — MANGE D.: *Analyse et synthèse des systèmes logiques.* Ed. Giorgi. St. Saphorin.

MANG 82 — MANGE D., SÁNCHEZ E., STAUFFER A.: *Systèmes logiques programmés.* Presses Polytechniques Romandes. Lausanne.

MARI 76 — MARIN J., ANDRE C., BOERI F.: Conception de systèmes séquentiels totalement autotestables à partir des réseaux de Petri. *Rairo-Automatique*, vol. 10, n. 11, pp. 23-40.

MART 80 — MARTÍNEZ J., SILVA M., VELILLA S.: Traducción de expresiones lógicas: Aplicación a autómatas programables y a la enseñanza. XII *Reunión de la Sociedad de Estadística, Investigación Operativa e Informática.* Jaca, Septiembre.

MART 81a — MARTÍNEZ J., SILVA M., VELILLA S.: Componentes conservativas elementales y realización microprogramada de una red de Petri conforme. IV *Jornadas de Automática.* Jaca, Septiembre, pp. 317-327.

MART 81b — MARTÍNEZ J., SILVA M.: A simple and fast algorithm to obtain all invariants of a generalized Petri net. *Second European Workshop on Application and Theory of Petri Nets.* Bad Honnef, September. (Publicado en [GIRA 82], pp. 301-310.)

MART 82 — MARTÍNEZ J., SILVA M.: A Package for Computer Design of Concurrent Logic Control Systems. *Symp. IFAC on Software for Computer Control, SOCOCO.* Madrid, September, pp. 221-226.

MASC 79 — MASC 79: *Masc 16, modules for alarm and sequence control.* SEMS n. 1. 164. 705. 000/3601. Grenoble, Février.

MCCL 65 — MCCLUSKEY E. J.: *Introduction to the theory of switching systems circuits.* McGraw-Hill. New York.

MEIN 72 — MEINADIER J. P.: *Estructura y funcionamiento de los computadores digitales.* Ed. AC. Madrid.

MEKL 80 MEKLY L. J. AND YAU S. S.: Software design representation using abstract process networks. *IEEE Trans. on Software Engineering*, vol. SE-6, n. 5, September, pp. 420-434.

MELD 71 MELDMAN J. A., HOLT A. W.: Petri nets and legal systems. *Jurimetrics*, J 12, 2, December, pp. 65-65.

MEMM 78a MEMMI G.: Fuites dans les réseaux de Petri. *Rairo-Informatique Théorique*, vol. 12, n. 2, pp. 125-144.

MEMM 78b MEMMI G.: Fuites et semi-flots dans les réseaux de Petri. *Thèse Doct. Ingénieur*, Université P. et M. Curie. Paris, Décembre.

MEMM 79a MEMMI G.: Notion de dualité dans les réseaux de Petri. *International Symposium on Semantics of Concurrent Computation*. Evian, July. Lecture Notes in Computer Sciencer, n. 70. Springer-Verlag, Berlin, pp. 91-108.

MEMM 79b MEMMI G. ROUCAIROL G.: Linear algebra in net theory. *Advanced Course on General Net Theory of Processes and Systems*. Hamburg, October. (Publicado en [BRAU 80], pp. 213-223].

MICH 79 MICHEL G., LAURGEAU C., ESPIAU B.: *Les automates programmables industriels*. Dunod. Paris.

MISU 73 MISUNAS D.: Petri nets and speed independent design. *Comm. ACM*, vol. 16, n. 8, pp. 474-481.

MOAL 76 MOALLA M., SIFAKIS J., ZACHARIADES M.: MAS, un outil d'aide à la description et à la conception des automatismes logiques. *Colloque* ADEPA-AFCET: *Automatismes Logiques. Recherches et Applications Industrielles*. Paris, Décembre, pp. 21-38.

MOAL 78 MOALLA M., PILOU J., SIFAKIS J.: Réseaux de Petri synchronisés. *Rairo-Automatique*, vol. 12, n. 2, pp. 103-130.

MOAL 80 MOALLA M., SIFAKIS J, SILVA M.: A la recherche d'une methodologie de conception sûre des automatismes logiques basée sur l'utilisation des rèseaux de Petri. *Monographie* AFCET: *Sûreté de Fonctionnement des Systèmes Informatiques* (5éme chapitre). Ed. Hommes et téchniques. Suresnes.

MOAL 81 MOALLA M.: Spécification et conception sore d'automatismes discrets complexes, basées sur l'utilisation du GRAFCET et des réseaux de Petri. *Thèse Doc. d'Etat*, U. S. M. - I. N. P. Grenoble, Juillet.

MORA 75 MORA V.: Conception et réalisation d'un système informatique de commande binaire de processus industriels. *Thèse Doc. 3éme Cycle*, Univ. Paul Sabatier. Toulouse, Juin.

MURA 77a MURATA T.: State equation, controllability and maximal matchings of Petri nets. IEEE *Trans. on Automatic Control*, vol. AC-22, n. 3, June, pp. 412-416.

MURA 77b MURATA T.: Circuit theoretic analysis and sysnthesis of marked graphs. IEEE *Trans, on Circuits and Systems*, vol. CAS-24, n. 7, July, pp. 400-405.

MURA 80a MURATA T., KOH J. Y.: Reduction and Expansion of live and safe masked graphs. IEEE *Trans. on Circuits and Systems*, vol. CAS-27, n.° 1, Jawary, pp. 68-70.

MURA 80b MURATA T.: Synthesis of decision-free concurrent systems for prescribed ressonsces and performance. IEEE *Trans. on Software Engineering*, vol. se-b, n.° 6, November, pp. 525-530.

PATI 73 PATIL S. S., DENNIS J. B.: The description and realization of digital systems. *Rairo-Automatique*, n. 1, Février, pp. 53-59.

PATI 75 PATIL S. S.: Micro-control for parallel asynchronous computers. *Euromicro Workshop*, North-Holland. Nice, June, pp. 17-24.

PERE 80 PEREZ T.: Optimisation en surface des PLAs. *Rapport de Recherche n. 216*, Ecole Nationale d'Informatique et Mathématiques Appliquées. Grenoble.

PERR 67 PERRIN J. P., DENOUETTE M., DACLIN E.: *Systèmes logiques* (2 tomes). Dunod, Paris.

PETE 77 PETERSON J. L.: Petri nets. *Computing Surveys*, vol. 9, n. 3, September, pp. 235-251.

PETE 81a PETERSON J. L.: *Petri net theory and the modeling of systems.* Prentice-Hall. Englewood Cliffs, New Jersey.

PETE 81b PETERSON J. L.: A note on coloured Petri nets. *Inf. Process Letters*, vol. 11, n. 1, August, pp. 40-43.

PETR 62 PETRI C. A.: *Comunication with automata.* Supplement 1 to Technical Report RADC-TR-65-377, vol, 1. New York 1966. (Traducido de «Kommunication mit Automaten», Universidad de Bonn, 1962.)

PETR 73 PETRI C. A.: Concepts of net theory. *Proc. Symposium and Summer School on Mathematical Foundations of Computer Science.* High Tatras, September.

PLEY 77 PLEYBER J., SILVA M.: Software specification for sequential processes. IFAC-IFIP *Workshop on Real Time Programming.* Eindhoven, June, pp. 67-73. (Ed. Pergamon Press, Oxford.)

PLUM 73 PLUMMER W. W.: Asynchronous arbiter. IEEE *Trans. on Computer*, vol. C-21, n. 1, January, pp. 37-42.

PRAT 75 PRATT W. C., BROWN F. M.: Automated Design of Microprocessor Based Controllers. IEEE *Trans. on* IECI, vol. 22, n. 3, August, pp. 273-279.

RAMA 80 RAMAMOORTHY C. V., GARY S. H.: Perfomance evaluation of asynchronous concurrent systems using Petri nets. IEEE *Trans. on Software Engineering*, vol. SE-6, n. 5, September, pp. 440-449.

RAMC 73 RAMCHANDANI C.: Analysis of asynchronous concurrent systems of timed Petri nets. *Ph. D. Thesis*, MIT, September.

REIS 82 REISING W.: *Petrinetze.* Springer-Verlag. Berlin.

REZA 77 REZA F.: *Los espacios lineales en la ingeniería.* Ed. Reverté. Barcelona.

ROTH 78 ROTH J. P.: Programmed logic array optimization. IEEE *Trans. on Computers*, vol. C-27, n. 2, Febrary, pp. 174-176.

ROJO 82 ROJO J.: *Álgebra lineal.* Editoria AC, Madrid.

SCHR 74 SCHROTT G.: Petri nets as a tool for comparing strategies of a operation system. *Symp.* IFAC-IFIP Real Time Programming. Budapest, pp. 3-15.

SESC 75 SESCOSEM (CATÁLOGO): *Logic TTL integrated circuits.* Ed. Radio. Paris, pp. 573-580.

SHOL 74 SHOLL H. A.: Direct transition memory and its application in computer design. IEEE *Trans. on Computers*, vol. C-23, n. 10, October, pp. 1048-1061.

SHOL 75 SHOLL H. A., SHOU-CHUNG Y.: Design of asynchronous sequential network using ROM. IEEE *Trans. on Computers*, vol. C-24, n. 2, February, pp. 195-206.

SIFA 77 SIFAKIS J.: Use of petri nets for performance evaluation. *Measuring, Modelling and Evaluating Computer Systems.* North-Holland Publ. Company, pp. 75-93.

SIFA 78 — SIFAKIS J.: Structural properties of Petri nets. *Mathematical Foundations of Computers Science*, J. Winkowski Ed., Springer-Verlag. Berlin, pp. 474-483.

SIFA 79a — SIFAKIS J.: Realization of fault-tolerant systems by coding Petri nets. *Journal of Design Automation and Fault-Tolerant Computing*, vol. 3, n.º 2.

SIFA 79b — SIFAKIS J.: Le contrôle des systèmes asynchrones: Concepts, proprietés, analyse statique. *Thèse Docteur ès Sciences*, U.S.M. - I.N.P. Grenoble, Juin.

SILV 77 — SILVA M., DAVID R.: On the programmation of asynchronous sequential systems by logic equations. *Symp. IFAC on Discrete Systems.* Dresde, March, pp. 52-62.

SILV 78 — SILVA M.: Contributions à la synthèse programmée des automatismes logiques. *Thèse Docteur Ingénieur*, I.N.P. Grenoble, Juin.

SILV 79a — SILVA M.: Evalution des perfomances des applications temporelles de type logique. *Symposium* MIMI-79. Zürich, May, pp. 152-157.

SILV 79b — SILVA M.: Evaluación de funciones lógicas: Generación óptima de código a partir de un grafo de decisiones binarias. *IV Congreso Nacional de Automática e Informática*, Madrid, Octubre, pp. 628-636.

SILV 79c — SILVA M., DAVID R.: Synthèse programmée des automatismes logiques décrits par réseaux de Petri: Une méthode de mise en oeuvre sur microcalculateurs. *Rairo-Automatique*, vol. 13, n. 4, Novembre, pp. 369-393.

SILV 80a — SILVA M., VELILLA S.: Sistema especializado en la simulación de redes de Petri sanas. *I Simposium Nacional* IFAC *sobre Modelado y Simulación.* Sevilla, Mayo, pp. 81-88.

SILV 80b — SILVA M.: Redes de Petri y validación de sistemas con actividades concurrentes. *XII Reunión Nacional de la Sociedad de Estadística, Investigación Operativa e Informática.* (Ponencia base.) Jaca, Septiembre.

SILVA 80c — SILVA M.: Simplification des réseaux de Petri par élimination des places implicites. *Digital Processes*, vol. 6, pp. 245-256.

SILV 81 — SILVA M.: Sur le concept de macroplace et son utilisation pour l'analyse des réseaux de Petri. *Rairo-Automatique*, vol. 15, n. 4, pp. 57-67.

SILV 82a — SILVA M.: Hacia una nueva concepción del análisis y diseño de los sistemas lógicos secuenciales. *Regulación y Mando Automático*, n. 118, pp. 51-57.

SILV 82*b* — SILVA M., DAVID R.: Binary decision graphs. Aceptado para su publicación por *Digital Processes.*

SILV 82c — SILVA M., VELILLA S.: Programmable logic controllers and Petri nets. *Symp. IFAC on Software for Computer Control.* Madrid, September, pp. 29-34.

SOLO 80 — SOLODÓVNIKOV A. S.: *Sistemas de desigualdades lineales. Editorial MIR. Moscú.*

SURF 76 — UPS/UER-INFORMATIQUE, ONERA/CERT, CNRS/LAAS: Mémoire de définition du projet pilote «*Sûreté de fonctionnement des systèmes informatiques*». Toulouse, Janvier.

SUZU 80 — SUZUKI I., MURATA T.: Stepwise refinement of transitions and places. *I European Workshop on the Application and Theory of Petri Nets.* Strasbourg, September. (Publicado en [GIRA 82], pp. 136-141.)

SZLA 77 SZLANKO J.: Petri nets for proving some correctness properties of parallel programs. IFAC/IFIP *Workshop on Real-Time Programming*. Eindhoven, Juin, pp. 75-83. (Ed. Pergamon Press, Oxford.)

TAFA 79 TAFAZZOLI M. E.: Réalisation d'un interpréteur material de réseaux de Petri à capacité. Application á la réalisation d'un multiprocesseur. *Thèse Doct. 3éme Cycle*, Université de Nice, Novembre.

TELE 77 TELEMECANIQUE ELECTRIQUE: *Automatismes séquentiels*. Documentación serie TST2.

THAY 81 THAYSE A.: P-Functions: A New Tool for the Analysis and Sysnthesis of Binary Programs. IEEE *Trans. on Computers*, vol. C-30, n. 2, February, pp. 126-134.

THEL 78 THELLIEZ S.: *Pratique séquentielle et réseaux de Petri*. Eyrolles. Paris.

THEL 80 THELLIEZ S., TOULOTTE J. M.: *Grafcet et logique industrielle programmée*. Eyrolles. Paris.

TISO 67 TISON P.: Generalisation of consensus theory and applications to the minimization of Boolean functions. IEEE *Trans. on Electronic Computers*. Vol. EC-16, pp. 446-456.

TORN 72 TORNG H.: *Switching circuits. Theory and logic design*. Addison-Wesley, Reading, Mass.

TOUD 81 TOUDIC J. M.: Algorithmes d'analyse structurelle des réseaux de Petri. *Thèse 3éme Cycle*, Univ. P. et M. Curie, Paris VI, Octobre.

TOUR 76 TOURRES L.: Une méthode nouvelle d'étude des systèmes logiques et son application à la réalisation d'automatismes programmés. *Revre Générale de l'Electricité*, t. 85, n. 3, Mars. pp. 215-219.

TOUL 78 TOULOTTE J. M.: Réseaux de Petri et automates programmables. *Automatisme*, tome 23, n. 6-7. Juillet-Août, pp. 200-211.

UNGE 69 UNGER S. H.: *Asynchronous sequential switching circuits*. Wiley-Interscience, New York.

VALE 76 VALETTE R.: Sur la déscription, l'analyse et la validation des systèmes de commande parallèle. *Thése Doct. ès Science*, Univ. Paul Sabatier, Toulouse, Novembre.

VALE 77 VALETTE R., COURVOISIER M.: Recherche d'un modele adapté aux systèmes de commande de processus à évolutions parallèles. *Rairo-Automatique*, vol. 11, n. 1, pp. 51-85.

VALE 78a VALETTE R., DIAZ M.: Top-down formal specification and vérification of parallel control systems. *Digital Processes*, vol. 4, n. 3, pp. 181-189.

VALE 78b VALETTE R.: Etude comparative de deux cutils de représentation: Grafcet et réseau de Petri. *Le Nouvel Automatisme*, Décembre, pp. 337-382.

VALE 79 VALETTE R.: Analysis of Petri nets by stepwise refinements. *Journal of Computer and System Sciences*, vol. 18, n. 1, pp. 35-46.

WEND 77 WENDT S.: Using Petri nets in the design process for interacting asynchronous sequential circuits. *Symp.* IFAC *on Discrete Systems*. Dresden, Mai, pp. 130-138.

WIRT 76 WIRTH N.: *Algorithms + Data Structures = Programs*. Prentice-Hall. (Editado en español por Ed. Castillo. Madrid, 1980.)

YOEL 77 YOELI M., BARZILAI Z.: Behavioural descriptions of communication switching systems using extended Petri nets. *Digital Processes*, vol. 3, pp. 307-320.

Índice

A

Acción (célula de) 222
Acontecimiento 36 (*véase* Evento)
Activación
  condición de 199-202, 213
  condición de 199, 200
Actualización del marcado 294, 295, 326, 336, 343, 355, 364
Alcanzable (marcado) 34, 179
Alcanzabilidad 138
Aleatoriedad
  de programación 289, 293
  esencial 211
Aleatorio (fenómeno) 199, 206, 212
Álgebra
  de Boole 163, 167, 262
  lineal 101, 163
Algoritmo de cómputo de
  cobertura con RdP y reducción 275
  cobertura no redundante 350
  compatibles máximos 78
  componentes elementales 146, 389
  componentes conexa y fuertemente conexa 380
  lugares potencialmente implicantes 70
Algoritmo de simulación de grafos reducidos 339, 340
Algoritmos de simulación de RdP
  básico con listas 336
  básico matricial 324
  dirigido por el marcado y
    escrutación de lista 355, 356
    escrutación de vector 343
  dirigidos por transiciones sensibilizadas 364
Algoritmos para el análisis de
  reductibilidad de subRdP 116
  vivacidad en grafo de marcados 108
Análisis
  de ciclicidad 109
  de conflictividad 109
  dinámico 100
  estático 100
  estructural 100, 101, 162
    global 128, 137
    local 137
  por enumeración 100, 110
  por reducción 110
  por simulación 100
  por transformación 100, 101
Analizadores booleanos de estados finitos 299
Anulador de una matriz
  derecho 284, 285
  izquierdo 384, 385
  no negativo 142
Anuladores (base de) 138, 148, 149, 383, 385
Árbitro 212
Árbol 380
Arco 16, 30, 377
  inhibidor 55, 56
  peso de un 30, 51
Arista 379
Ascendiente (lugar) 112
Aserción 139, 140, 171
Asignación
  de las acciones 72

Atribución 20
Autómata
de MEALY 5, 6, 243, 244
de MOORE 5, 6, 243, 244
finito 1, 4
no finito 2
programable 279, 297
Autónoma (RdP) 91, 169
Autoverificable (realización) 225
Avance sincrónico 99, 157-162
ponderado 100, 160

B

Base de anuladores 138, 148, 149, 383-385
Biestable 196
D 218, 220
J-K 218, 220
lógica de transformación de 217
programación de 292
R-S 198, 199
T 218, 221
Binario
lugar 94
red de Petri 22, 94, 195
Bit de asignación 300, 310
Bloque 92
BOOLE 163, 167, 262

C

CAD 125, 181, 182
Camino 379
Campo
de salidas 251
de test 250
Célula de memoria 198
activación de 202, 207, 209
asíncrona 201, 209
conexión de 202-204, 214
desactivación de 203, 207, 209
esquemas lógicos de 201
síncrona 217-219
Cerrojo 162, 163, 166
mínimo 167
Cíclica (RdP) 94-96
Ciclo de tratamiento 281, 286
encadenamientos 286
tipos 286, 287
Cierre de un nudo 381
Cobertor (elemento) 271
Cobertura 271
algoritmos 275, 350
componente esencial 271
irredundante 272, 349
tabla de 271, 349
Codificación
de la estructura de RdP 349
del estado 195
por campos 241, 265
total 239, 264
Código
de HAMMING 226
Coloreada (RdP) 60
COMMONER (teorema de) 165
Compatibilidad (relación) 78, 238, 242
algoritmo 78, 79
clase de 78, 238, 242
clase máxima de (o compatible máximo) 78, 238
clase principal de 242
tabla de 78, 81, 110
Compatible máximo (o clase máxima de compatibilidad) 78, 238
Compilación de expresiones 302, 308, 311, 319
Componente
canónica 143, 144
conexa 379, 380
conservativa monomarcada 216, 269, 271
elemental 143, 144, 387
fuertemente conexa 106, 107, 379, 380
repetitiva 143, 157, 162
soporte de una 143
Complementario (lugar) 53, 57
Concurrente 2, 16
Condición
de activación de un biestable 199-203, 213
de desactivación de un biestable 199-203, 213-216
de evolución del marcado 294
de mantenimiento 246-247
envolvente de un lugar 39, 77
envolvente de un marcado 83
externa 36, 78

Conexionado
por llamada-respuesta 209, 212-216
por transferencia impulsional 198, 200-205
Conflicto 22
efectivo 22, 97, 198, 211, 325, 351
estructural 97
Conforme (RdP) 22, 122, 165
Conjunción 21
Conjunto
de lugares de entrada 31
de lugares de salida 31
de lugares implicantes 66
de secuencias de disparo 33
de transiciones de entrada 31
de transiciones de salida 32
de vectores característicos 34
fundamental 143-145, 148
Conservativa (RdP) 128, 130, 133-134
Consola de programación 285
Contactos (esquema de) 289
Contador
módulo $k+1$ 196
conmutador (transición) 60
CUSA 201, 209

D

D (biestable) 218, 220, 245
Decisiones binarias (máquina) 250-252, 267, 298
Decodificador 244, 284
programble 237
Desactivación
condición de 199-203, 213-216
simplificada 213-214
Desarrollo de una instrucción 285
Descendiente (lugar) 112
Descomposición de RdP
en GR 269
en RdP 275
Descripción
descendente o por refinamientos sucesivos 3, 16, 23, 92, 182-190
modular 92, 190-193
Diagrama
de fases 6, 7
de KARNAUGH 221, 233
lógico (lenguaje) 289
primitivo de fases 7
Discriminación condicional 187
Disparo de una transición sensibilizada 33
Distancia
de HAMMING 226
sincrónica (*véase* Avance)
Distribución 21
Dual (RdP) 131
Duración mínima de eventos 210

E

Ecuación
de estado (RdP) 34, 128, 157, 323
de excitación 248, 262
Ecuaciones
lineales (sistema) 383
Enumeración (análisis por) 100-110
Error
mecanismos autodetectores 284
Espectro (de un nudo) 381
Estado
código del 197, 245, 248
del autómata 4
ecuación de 34, 128, 157, 323
estable 6
inicial 4
interno 35
sucesor 245, 251
transitorio 6
Esquemas de contactos 289
Estructural (análisis) 100-101, 128-169
Estructuralmente
limitada (RdP) 95, 133-136
viva (RdP) 93, 134-136
Evento 36, 78, 210
transformación 37-40, 80-82, 273
Evolución
autónoma 6
controlada 6
del marcado 16, 33
Excitación (ecuación de), 248, 262
Exclusión mutua 30, 92, 98, 151
Explorador de lugares marcados 320
Expresiones lógicas
interpretación y ejecución 295-296
compilación 302, 308, 311, 319

F

Fases
  diagrama de 6, 7
  tabla de 1, 6-7
Fenómenos aleatorios 199, 206-212
Filósofos y «spaghetti» 174
Fuente
  lugar 65, 82-85
  transición 82
Función
  de lectura 4
  de salida 4
  de transición 4
Funcionamiento
  autónomo 29
  por llamada-respuesta 209, 212-217
  por transferencia impulsional 198, 212
  no autónomo 30
Funciones
  idénticas 238, 246
  intermedias 239
  lógicas de sincronización 266
  partición de 265
  pseudoidénticas 238
Fusión de lugares 65, 75

G

Generalizada (RdP) 30, 51
Grafo
  acíclico 377
  bipartido 30, 380
  conexo 379
  de alcanzabilidad 101
  de cobertura 101
  de estados 10, 48
  de marcados 102, 110
  de sincronización 48, 385
  fuertemente conexo 379
  marcado 48, 132, 165, 385
  ponderado 378
  reducido 1, 10, 15, 19, 23, 196, 243, 161

H

HAMMING 226
Holgura (variable) 141, 396
Hoja de un árbol 380

I

Idénticas (transiciones) 123
Identidad
  lugar 74
  transición 123
Implicantes (lugares) 66, 154
Implícito (lugar) 64-67, 69, 111, 123, 153, 160, 352
Incidencia
  matriz 31, 32, 128, 378
  posterior (función de) 30
  posterior (matriz) 31
  previa (función de) 30
  previa (matriz) 31, 326
Incompatible máximo 242
Inecuaciones lineales (sistema) 168, 395-396
Inhibidor (arco) 55-56
Inicial (marcado) 18, 33
Instrucción
  desarrollo 285
  única 254, 257, 261
Instrucciones
  con dirección explícita 258
  con dirección implícita 259
Interpretación (asociada a RdP) 29, 35, 373
Interpretada (RdP) 36, 177
Interpretador
  algoritmo 316
  de diagramas lógicos 298, 299
  de expresiones lógicas 298, 301
  mixto operaciones y saltos 298, 308
Invariante lineal
  de disparo 137, 157-158
  de marcado 137-138
  elemental 150

J

J-K (biestable) 218, 220

K

k-limitada (RdP) 94
KARNAUGH 221, 233
KIRCHOFF 129, 131

L

Lectores-redactores 45, 153, 155, 162
Lectura del marcado de un lugar 204, 214
Lenguaje (reconocido por una RdP) 33
Lenguajes de programación 287
  de esquemas de contactos 289
  de diagramas lógicos 289
  de notación booleana 295
  de RdP 316
Libre elección (RdP) 50,165
Libre de monopolio (RdP) 177
Limitada (RdP) 96, 178
Limitado (lugar) 94
Límite de un lugar 150
Lineal
  álgebra 101, 163
  sistema 35, 139, 383, 395
Listas 331, 332
  de lugares 334, 342
  de transiciones 333
  lineal 332
  representaciones de RdP basadas en 331, 333-336
Lugar 16, 30
  ascendiente de subRdP 112
  binario 94
  complemetario 53, 57
  de entrada 31
  de reposo 272
  de salida 31
  de sincronización 342, 349
  descendiente de subRdP 112
  fuente 65, 82-83
  identidad 74
  implícito 64-67, 69, 111, 123, 153, 160, 352
  k-limitado 94
  lectura del marcado de un 204, 214
  potencialmente implícito 66, 154
  potencialmente reducible 111
  representante de transición 341, 349
  representante esencial 341, 349
  sumidero 83
  sustituible 118-119, 124
Lugares
  compatibles 75-78
  equivalentes 75, 78
  fuente (método de) 78, 84
  fusionables 75, 78
  implicantes 66, 154

LL

Llamada-respuesta
  conexionado/funcionamiento 209, 212-217

M

Macrofago 269-270
Macrolugar 112-116, 269
Máquina
  de decisiones 2^{ν}-arias 250, 252
  de decisiones binarias 252, 267, 298
  de estado 48, 132, 385
  de MEALY 243-244
  de MOORE 80-81, 243-244
  de TURING 57
  M_1 (diagramas lógicos) 299
  M_2 (cadenas postfijas) 301
  M_3 (operación lógica) 306
  M_4 (mixta) 308
  secuencial 267
Marca 16, 32
Marcada (RdP) 33
Marcado 32
  actualización 294—295, 326, 336, 343, 355, 364
  alcanzable 34, 179
  evolución del 16, 33
  inferior 103
  inicial 18, 33
  potencialmente alcanzable 139, 153
  superior 103
Matriz
  asociada al grafo 377
  de flujo de marcas 67, 330
  de incidencia 31-32, 128, 378
  de incidencia nudo-arco 378
  de incidencia posterior 31
  de incidencia previa 31, 326
  de receptividad 260-262
  lógica programable (*véase* PLA)
  O 234-237, 239
  Y 234-237, 239
MEALY 5-6, 243-244
Memoria
  asociativa 237, 245
  de programa 282
  de sólo lectura (ROM) 230, 238

Memoria (*Cont.*)
de variables 282
imagen borrada (MIB) 321, 366
imagen de entradas (MIE) 282, 321
imagen de salidas (MIS) 282, 321
Método
cerrojos y trampas 323-325
de enumeración 101-110
de fusión de lugares 75
de lugares fuente 78, 84
de ponderación 225-226
de reducción 155 (*veáse* Análisis)
dinámicos 100
estructural 137, 162
matricial de simulación 323-325
Métodos
de análisis (*veáse* Análisis)
de descripción 92, 181
dinámicos 100
estáticos 100
Microcomputador 315, 320, 326, 344, 371
Microinstrucción 255
de asignación 259
de salto condicional 258
de salto incondicional 259
única 255
Microprocesador 285, 313
Microprograma 255, 263
Minimización 63-64
Modelación (*véase* Descripción)
Modelo
estructural 1
funcional 1
Monomios lógicos 237
Monopolio (RdP libre de) 177
MOORE 5-6, 80-81, 243-244
Muestreo 282, 321
Multiplexores 248, 249, 251
Multiprogramación 268, 372

N

Nudo 377
espectro de un 381
predecesor 379
raíz 380
sucesor 379
terminal 106, 380
Nudo-O 21, 48, 203
Nudo-Y 21, 48, 115, 203

O

O-exclusivo (transición) 60
Ordinaria (RdP) 31
Organigrama 288-295

P

Parcialmente viva (RdP) 93, 152-153, 164-165
Parte
de control 2-3, 284
operativa 2-3, 284
Perro guardián 287
PERT, 49
Peso
activo de RdP 225
del arco 30, 51
total de la RdP 226
PETRI (*véase* de Petri)
Pila (LIFO) 304, 354
de tratamiento 354-6
en formación 354-6
PLA 229
definición 234
estructuras multi- 235-245
secuenciales 236
tabla de programación de 246
Polaca postfija (notación) 301
Ponderación (método de) 225-226
Potencialmente
alcanzable (marcado) 153
implicantes (lugares) 66, 153
implícito (lugar) 66-67, 153-154
reducible (subred) 111, 117
Procedimiento evento-acción 324, 342
abierto 326
cerradop 324
Productor-Consumidor 40, 192, 375
Programable
autómata 279, 297
decodificador 237
matriz lógica (*véase* PLA)
Programación (*véase* Lenguajes)
concurrente 373
consola de 285
de computadores 373
estructurada 170
no síncrona de RdP 292

Programación (*Cont.*) (*véase* Lenguajes)
secuencial 373
síncrona de RdP 292
tabla de 246, 249
Propiedad cerrojo-trampa 164-165
Propiedades de buen funcionamiento 92
Puntero 332, 334
nulo 332
representaciones con 332
siguiente transición 334
Pura (RdP) 31, 157

R

Realización cableada de RdP 195, 222-226
asíncrona 198-217
autoverificable 225
asíncrona-autosincronizada 260
de una temporización 205
modular 196
síncrona 198, 217-222
Realización con macrocomponentes
métodos lógicos de transición directa
GR 243, 248
RdP 264
métodos con secuenciador
GR 252, 262
RdP 267
Realización programada de RdP binarias (*vease* Simulación programada de RdP)
con ecuaciones y saltos 296-297
con ecuaciones lógicas 296-297
con diagramas lógicos 292-295
Receptiva (transición) 36
Receptividad 11, 36
matriz de 260-262
Red de Petri
autónoma 91, 169
binaria 22, 94, 195
cíclica 94-96
coloreada 60
con arcos inhibidores 55-56
con capacidad 52
conforme 22, 122, 165
conservativa 128, 130, 133-134
dual 131
ecuación de estado de 34, 128, 323
estructuralmente limitada 95, 133-134
estructuralmente viva 93, 134-136
generalizada 30, 51
interpretada 36, 177
k-limitada 94
libre elección 50, 165
libre de monopolio 177
limitada 96, 178
marcada 33
no-autónoma 30, 173, 179
no-limitada 22
ordinaria 31
parcialmente viva 93, 152-153, 164-165
pura 31, 157
repetitiva 128-129, 133-135
simple 50, 352
temporizada 61-62, 173-177
viva 21, 93-96
Reducido (grafo) 1, 10, 15, 19, 23, 196, 243, 261
Reducción (análisis por) 110-27
Reducción de
longitud de palabra 238-243
RdP 110-127
subRdP 111
Redundancias
estructurales 225
funcionales 225
no separables 225
separables 225
Registro
acumulador 284, 304, 311
contador de programa 253, 284
de estado 243-247, 255
de instrucción 284
de salidas 245, 251
Regla de evolución del marcado 17, 33, 36
Relaciones
invariantes 179
sincróicas 97
Repetición (estructura de) 187
Repetitiva (RdP) 128-129, 133-135
Representante
de transición (lugar) 341, 349
esencial 341, 349
ROM 229-230, 238

S

Salida(s)
continuidad de 207

Salida(s) (*Cont.*)
función de 4
memoria imagen de 282, 321, 366
memoria imagen borrada de 321, 366
realización de 204-205
registro de 245-251
Secuencia
de disparos 33, 128, 159, 179
estructura 187
Secuenciador
dispositivo 244, 252, 260, 284
método con 252-262
Selección 20
Semáforo 158, 191, 374
de exclusión mutua 192
teorema del 158-159
Sensibilidad de una salida 11, 36
Sensibilizada (transición) 33, 68
Simple (RdP) 50, 352
Simplificación de la descripción 63-65, 222-225
Simulación conducida por
programa 319
tabla 319
Simulación programada de GR 337
Simulación programada de RdP binarias
(*véase* Realización programada de RdP)
con diagramas lógicos 294
con ecuaciones lógicas 296
con ecuaciones y saltos 296-297
dirigida por el marcado
escrutación lista 353, 366
escrutación vector 341
dirigida por transiciones
escrutación total 331
sensibilizadas 363
lenguajes especiales 366
método matricial 323
síncrona 320-321
Sincronización de máquinas secuenciales 268
Síncrono(a)
biestable 217, 221
célula 217-219
realización cableada 217
sistema 36
Sincronización
de máquinas secuenciales 268
entre subsistemas 190-191
Sistema
asíncrono 6
concurrente 2, 16
lineal 35, 139, 383, 395
secuencial 1, 3, 10
síncrono 6, 36
Soporte de una componente 143
STIEMKE 134
Subconjunto final de estados 94
Subprograma 45, 326
Subred de Petri 32
potencialmente reducible 111
reducible 112, 116
Sumidero
lugar 83
transición 83
Sustitución
de un lugar 118-122, 183
de una transición 188-189
Sustituible (lugar) 118-119, 124

T

T (biestable) 218, 221, 247
Tabla de
cobertura 271, 349
compatibilidad o de pares compatibles 78, 81, 110
estados 5
fases 1, 6
fases primitiva 7
KARNAUGH 221, 233
programación 246, 248
verdad 233, 248, 262
verdad compactada 246
Temporización (realización de) 205
Temporizada 61-62, 173-177
Teorema
de COMMONER 165
de STIEMKE 134
del semáforo 158-159
Término producto 235, 247
Tiempo de disparo 61
Trampa 162-164
mínima 169
Transformación
de descomposición 259
de eventos 37-40, 80-82, 204
de GR 254
de secuencialización 259
elemental 384

Transición 16, 30
conmutador 60
fuente 82
identidad 123
no estándar 59
no viva 22
O-exclusivo 60
receptiva 36
sensibilizada 33, 326
sumidero 82
viva 21, 92
Transiciones idénticas 123
TURING 57

V

Validación 23, 91
Variable
de holgura 141, 396
intermedia 239-240, 308, 310
interna de memorización 197
nula 136
Vector(es)
característico de una secuencia de disparos 34, 128
de pesos 225
del marcado 32, 323
incomparables 144-145
Verificación 91
Viva
transición 21, 92
Red de Petri marcada 21, 93-96
estructuralmente (RdP) 93, 134-136
Vivacidad 106, 152, 162-163, 174, 179

THOMSON

www.paraninfo.es
www.thomsonlearning.com

Editorial AC

ISBN 84-7288-045-1