

# INTRODUCCION A LA TEORIA DE AUTOMATAS, LENGUAJES y COMPUTACION

JOHN E. HOPCROFT  
JEFFREY D. ULLMAN

CECSA

# INTRODUCCION A LA TEORIA DE AUTOMATAS, LENGUAJES Y COMPUTACION

JOHN E. HOPCROFT

*Cornell University*

JEFFREY D. ULLMAN

*Princeton University*

Traducción

Homero Flores Samaniego

Físico

Facultad de Ciencias de la  
Universidad Nacional Autónoma de México

Revisión Técnica

Raymundo Hugo Rangel

Físico

Profesor de la División de Ingeniería Eléctrica,  
Electrónica y en Computación de la Facultad de  
Ingeniería de la UNAM

UNIVERSIDAD DE LA REPUBLICA  
FACULTAD DE INGENIERIA  
Dpto. DE DOCUMENTACION Y BIBLIOTECA  
BIBLIOTECA CENTRAL  
Ing. Edg. García de Zúñiga  
MONTEVIDEO - URUGUAY

No. de Entrada 053641  
30-01-01

COMPAÑIA EDITORIAL CONTINENTAL, S.A. DE C.V.  
MEXICO

SIBUR

## PROLOGO

UNIVERSIDAD DE LA REPUBLICA  
FACULTAD DE INGENIERIA  
DEPARTAMENTO DE  
DOCUMENTACION Y BIBLIOTECA  
MONTEVIDEO - URUGUAY

*Título original de la obra:*

INTRODUCTION TO AUTOMATA THEORY LANGUAGES, AND COMPUTATION  
By J.E. Hopcroft and J.D. Ullman.  
ISBN 0-201-02988-X

Traducción autorizada por:

Copyright © by Addison-Wesley Publishing Company, Inc.

Derechos reservados respecto a la primera edición en español  
© 1993 por COMPAÑIA EDITORIAL CONTINENTAL, S.A. DE C.V.  
Renacimiento Núm. 180, Col. San Juan Tlilhuaca, Delegación Azcapotzalco.  
Código Postal 02400, México, D.F.

Miembro de la Cámara Nacional de la Industria Editorial  
Registro Núm. 43

ISBN 968-26-1222-5

Prohibida la reproducción total o parcial de esta obra bajo cualquiera de sus formas, electrónica o mecánica, incluyendo el fotocopiado o por cualquier sistema de almacenamiento de información, sin el consentimiento previo y por escrito del editor.

Primera edición, 1993

Impreso en México  
Printed in Mexico

004  
H 791 i Ef  
e-2-

Hace diez años nos comprometimos a producir un libro que abarcara el material conocido sobre lenguajes formales, teoría de autómatas y complejidad computacional. En retrospectiva, sólo unos pocos resultados significativos fueron ignorados en todas sus páginas. Al escribir un nuevo libro sobre la materia, encontramos que el campo se ha extendido en tantas direcciones nuevas que tratar el tema de manera total y uniforme se hace imposible. Más que intentar ser enciclopédicos, hemos sido drásticos al editar el material, seleccionando sólo los temas centrales del desarrollo teórico del campo o sólo los que tienen importancia en las aplicaciones de ingeniería.

Durante los pasados diez años dos nuevas direcciones de investigación han adquirido extrema importancia. La primera lo constituye el uso de conceptos de la teoría de lenguaje, como el no determinismo y las jerarquías de complejidad, para probar los límites inferiores de la complejidad inherente de ciertos problemas prácticos. La otra ha sido la aplicación de las ideas de la teoría de lenguaje, como las expresiones regulares y las gramáticas libres de contexto, en el diseño de *software*, como compiladores y procesadores de textos. Ambos desarrollos han ayudado a darle forma a la organización de este libro.

### USO DEL LIBRO

Para un curso de nivel superior, deben consultarse los ocho primeros capítulos, excepto el material que trata sobre ambigüedad inherente, en el capítulo 4 y algunas partes del capítulo 8. Los capítulos 7, 8, 12 y 13 forman el núcleo de un curso sobre complejidad computacional. Un curso avanzado en teoría del lenguaje puede elaborarse a partir de los capítulos 2 al 7, del 9 al 11 y con el 14.

## EJERCICIOS

Los problemas más difíciles están señalados con dos asteriscos y los de dificultad intermedia se identifican con uno solo. Los ejercicios marcados con una S tienen su solución al final del capítulo. No hemos intentado proporcionar un manual de respuestas, sino una breve selección de ejercicios cuya solución sea particularmente instructiva.

## RECONOCIMIENTOS

Agradecemos a las siguientes personas sus comentarios y consejos oportunos: Al Aho, Nissim Francez, Jon Goldstine, Juris Hartmanis, Dave Maier, Fred Springsteel y Jacobo Valdés. El manuscrito fue magníficamente mecanografiado por Marie Olton y April Roberts, en Cornell y Gerree Pecht en Princeton.

*Ithaca, New York  
Princeton, New Jersey  
marzo 1979*

J.E.H.  
J.D.U.

## CONTENIDO

### Capítulo 1 Preliminares

1.1	Cadenas, alfabetos y lenguajes .....	1
1.2	Grafos y árboles .....	2
1.3	Pruebas inductivas .....	4
1.4	Notación de conjuntos .....	5
1.5	Relaciones .....	7
1.6	Sinopsis del libro .....	9

### Capítulo 2 Autómatas finitos y expresiones regulares

2.1	Sistemas de estados finitos .....	15
2.2	Definiciones básicas .....	18
2.3	Autómatas finitos no determinísticos .....	21
2.4	Autómatas finitos con movimientos- $\epsilon$ .....	26
2.5	Expresiones regulares .....	30
2.6	Autómatas finitos de dos direcciones .....	39
2.7	Autómatas finitos con salida .....	45
2.8	Aplicaciones de los autómatas finitos .....	48

### Capítulo 3 Propiedades de los conjuntos regulares

3.1	El lema de sondeo para conjuntos regulares .....	59
3.2	Propiedades de cerradura para conjuntos regulares .....	62
3.3	Algoritmos de decisión para conjuntos regulares .....	67
3.4	Teorema de Myhill-Nerode y minimización de autómatas finitos .....	69

<b>Capítulo 4 Gramáticas libres de contexto</b>	
4.1 Motivación e introducción .....	83
4.2 Gramáticas libres de contexto .....	85
4.3 Árboles de derivación .....	88
4.4 Simplificación de gramáticas libres de contexto .....	94
4.5 Forma normal de Chomsky .....	100
4.6 Forma normal de Greibach .....	101
4.7 Existencia de lenguajes libres de contexto inherentemente ambiguos	106
<b>Capítulo 5 Autómatas de apilamiento</b>	
5.1 Descripción informal .....	117
5.2 Definiciones .....	119
5.3 Autómatas de apilamiento y lenguajes libres de contexto .....	124
<b>Capítulo 6 Propiedades de los lenguajes libres de contexto</b>	
6.1 Lema de sondeo para CFLs .....	135
6.2 Propiedades de cerradura para CFLs .....	141
6.3 Algoritmos de decisión para CFLs .....	147
<b>Capítulo 7 Máquinas de Turing</b>	
7.1 Introducción .....	157
7.2 Modelo de la máquina de Turing .....	158
7.3 Lenguajes y funciones computables .....	162
7.4 Técnicas para la construcción de máquinas de Turing .....	164
7.5 Modificaciones de las máquinas de Turing .....	171
7.6 Hipótesis de Church .....	178
7.7 Máquinas de Turing como enumeradores .....	180
7.8 Máquinas de Turing restringidas equivalentes al modelo básico .....	183
<b>Capítulo 8 Irresolubilidad</b>	
8.1 Problemas .....	191
8.2 Propiedades de los lenguajes recursivos y numerables de manera recursiva .....	193
8.3 Máquinas de Turing universales y un problema irresoluble .....	195
8.4 Teorema de Rice y algunos otros problemas irresolubles .....	199
8.5 Irresolubilidad del problema de correspondencia de Post .....	208
8.6 Cálculos válidos y no válidos de TMs: una herramienta para probar problemas CFL irresolubles .....	215
8.7 Teorema de Greibach .....	219
8.8 Introducción a la teoría de las funciones recursivas .....	221
8.9 Cálculos de oráculo .....	224
<b>Capítulo 9 La jerarquía de Chomsky</b>	
9.1 Gramáticas regulares .....	233
9.2 Gramáticas no restringidas .....	236
9.3 Lenguajes sensibles al contexto .....	240
9.4 Relaciones entre tipos de lenguajes .....	243
<b>Capítulo 10 Lenguajes determinísticos libres de contexto</b>	
10.1 Formas normales para DPDA .....	250
10.2 Cerradura de los DCFLs con respecto a la complementación .....	251
10.3 Máquinas de predicción .....	256
10.4 Propiedades de cerradura adicionales de los DCFLs .....	259
10.5 Propiedades de decisión de los DCFLs .....	262
10.6 Gramáticas $LR(0)$ .....	264
10.7 Gramáticas $LR(0)$ y los DPDA .....	269
10.8 Gramáticas $LR(k)$ .....	277
<b>Capítulo 11 Propiedades de cerradura de familias de lenguajes</b>	
11.1 Tríos y tríos completos .....	287
11.2 Transformaciones de máquinas secuenciales generalizadas .....	289
11.3 Otras propiedades de cerradura para tríos .....	294
11.4 Familias abstractas de lenguajes .....	295
11.5 Independencia de las operaciones AFL .....	296
11.6 Resumen .....	299
<b>Capítulo 12 Teoría de complejidad computacional</b>	
12.1 Definiciones .....	305
12.2 Aceleración lineal, compresión de cinta y reducciones en el número de cintas .....	308
12.3 Teoremas de jerarquías .....	316
12.4 Relaciones entre medidas de complejidad .....	321
12.5 Lemas de traslación y las jerarquías no determinísticas .....	323
12.6 Propiedades de las medidas de complejidad general: teoremas de espacio, aceleración y unión .....	327
12.7 Teoría de complejidad axiomática .....	334
<b>Capítulo 13 Problemas no tratables</b>	
13.1 Tiempo y espacio polinomiales .....	343
13.2 Algunos problemas $NP$ -completos .....	348
13.3 La clase $co-NP$ .....	365
13.4 Problemas ESPACIOP completos .....	368
13.5 Problemas completos para $\mathcal{P}$ y $ESPACION(\log n)$ .....	372
13.6 Algunos problemas demostrablemente no tratables .....	374
13.7 La cuestión $\mathcal{P} = NP$ para las máquinas de Turing con oráculos: límites en nuestra capacidad para determinar si $\mathcal{P} = NP$ .....	387
<b>Capítulo 14 Características principales de otras clases de lenguaje</b>	
14.1 Autómatas de presión auxiliares .....	403

14.2 Autómatas de pila .....	408
14.3 Lenguajes indexados .....	416
14.4 Sistemas de desarrollo.....	418
 Bibliografía .....	
 Índice .....	425
 .....	441

## CAPITULO

## 1

## PRELIMINARES

En este capítulo revisamos las principales ideas matemáticas necesarias para la comprensión del material que se presenta en el libro. Estos conceptos incluyen grafos, árboles, conjuntos, relaciones, cadenas, lenguajes abstractos e inducción matemática. Presentamos también una breve introducción al libro y una motivación para su lectura. El lector que posee las bases en los temas de matemáticas mencionados puede pasar directamente a la sección 1.6 en donde encontrará observaciones y comentarios de motivación.

## 1.1 CADENAS, ALFABETOS Y LENGUAJES

Un “símbolo” es una entidad abstracta que no definiremos, de la misma manera que los conceptos de “punto” y “línea”, no se definen en geometría. Las letras y los dígitos son ejemplos de símbolos usados con frecuencia. Una *cadena* (o *palabra*) es una secuencia finita de símbolos yuxtapuestos. Por ejemplo, *a*, *b* y *c* son símbolos, y *abcb* es una cadena. La *longitud* de una cadena *w*, que se denota como  $|w|$ , es el número de símbolos que componen la cadena. Por ejemplo, *abcb* tiene longitud 4. La cadena vacía, denotada por  $\epsilon$ , es la cadena que consiste en cero símbolos. Por tanto,  $| \epsilon | = 0$ .

Los *prefijos* de una cadena están formados por los primeros símbolos de ésta; y los *sufijos*, por los últimos. Por ejemplo, la cadena *abc* tiene como prefijos:  $\epsilon$ , *a*, *ab* y *abc*; sus sufijos son  $\epsilon$ , *c*, *bc* y *abc*. Un prefijo o sufijo de una cadena que no sea la misma cadena es un prefijo o sufijo *propios*.

La *concatenación* de dos cadenas es la cadena que se forma al escribir la primera

seguida de la segunda, sin que haya espacio entre ellas. Por ejemplo, la concatenación de *perro* y *casa* es *perrocasa*. La yuxtaposición se utiliza como el operador de concatenación. Esto es, si  $w$  y  $x$  son cadenas, entonces  $wx$  es la concatenación de estas dos cadenas. La cadena vacía es la identidad para el operador de concatenación. Es decir,  $\epsilon w = w\epsilon = w$  para cada cadena  $w$ .

Un *alfabeto* es un conjunto finito de símbolos. Un *lenguaje (formal)* es un conjunto de cadenas de símbolos tomados de algún alfabeto. El conjunto vacío,  $\emptyset$ , y el conjunto formado por la cadena vacía  $\{\epsilon\}$  son lenguajes. Nótese que son diferentes: el último tiene un elemento, mientras que el primero no. El conjunto de *palíndromos* (cadenas que se leen igual de izquierda a derecha y viceversa) sobre el alfabeto  $\{0,1\}$  es un lenguaje infinito. Algunos elementos de este lenguaje son  $\epsilon, 0, 1, 00, 11, 010$  y  $1101011$ . Adviértase que el conjunto de todos los palíndromos sobre una colección finita de símbolos no es, técnicamente hablando, un lenguaje, porque sus cadenas no se construyen colectivamente a partir de un alfabeto.

Otro lenguaje es el conjunto de cadenas sobre un alfabeto fijo  $\Sigma$ . Denotamos a este lenguaje como  $\Sigma^*$ . Por ejemplo, si  $\Sigma = \{a\}$ , entonces  $\Sigma^* = \{\epsilon, a, aa, aaa, \dots\}$ . Si  $\Sigma = \{0, 1\}$ , entonces  $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$ .

## 1.2 GRAFOS Y ARBOLES

Un grafo, denotado por  $G = (V, E)$ , consiste en un conjunto finito de vértices (o nodos)  $V$  y un conjunto de pares de vértices  $E$  llamados aristas. En la Fig. 1.1 se muestra un grafo. En éste,  $V = \{1, 2, 3, 4, 5\}$  y  $E = \{(n, m) \mid n + m = 4 \text{ o } n + m = 7\}$ .

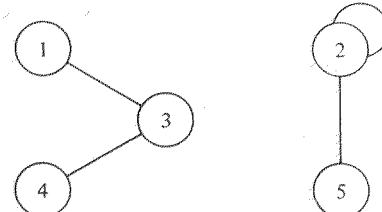


Fig. 1.1 Ejemplo de un grafo.

Una *trayectoria* en un grafo es una secuencia de vértices  $v_1, v_2, \dots, v_k$ ,  $k \geq 1$ , tal que existe un arista  $(v_i, v_{i+1})$  para cada  $i$ ,  $1 \leq i \leq k$ . La *longitud* de la trayectoria es  $k - 1$ . Por ejemplo, 1, 3, 4 es una trayectoria en el grafo de la Fig. 1.1; como también lo es 5 por sí mismo. Si  $v_i = v_k$ , la trayectoria es un *ciclo*.

### Grafos dirigidos

Un grafo *dirigido* (o *digráfo*), que también se denota como  $G = (V, E)$ , consiste en un conjunto finito de vértices  $V$  y un conjunto de pares ordenados de vértices  $E$ , llamados

*arcos*. Denotamos un arco de  $v$  a  $w$  como  $v \rightarrow w$ . En la Fig. 1.2 aparece un ejemplo de un digrafo.

Una *trayectoria* en un digrafo es una secuencia de vértices  $v_1, v_2, \dots, v_k$ ,  $k \geq 1$ , tal que  $v_i \rightarrow v_{i+1}$  es un arco para cada  $i$ ,  $1 \leq i < k$ . Decimos que la trayectoria es desde  $v_1$  a  $v_k$ . Así pues,  $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$  es una trayectoria de 1 a 4 en el digrafo de la Fig. 1.2. Si  $v \rightarrow w$  es un arco, decimos que  $v$  es un *predecesor* de  $w$  y que  $w$  es un *sucesor* de  $v$ .

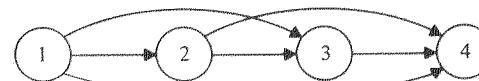


Fig. 1.2 Digráfo  $((1, 2, 3, 4), \{i \rightarrow j \mid i < j\})$ .

### Arboles

Un *árbol* (estrictamente hablando, un *árbol dirigido y ordenado*) es un digrafo que posee las siguientes propiedades.

1. Existe un vértice, llamado *raíz*, que no tiene predecesores y del cual parte una trayectoria hacia cada vértice.
2. Cada vértice, diferente de la raíz, tiene exactamente un predecesor.
3. Los sucesores de cada vértice están ordenados “a partir de la izquierda”.

Se deberán dibujar los árboles con la raíz en la cima y todos los arcos apuntando hacia abajo. Las flechas de los arcos, por tanto, no necesitan indicar la dirección y no se indicará. Los sucesores de cada vértice se dibujarán de izquierda a derecha. La Fig. 1.3 muestra el ejemplo de un árbol que es un diagrama de la oración “la rápida zorra

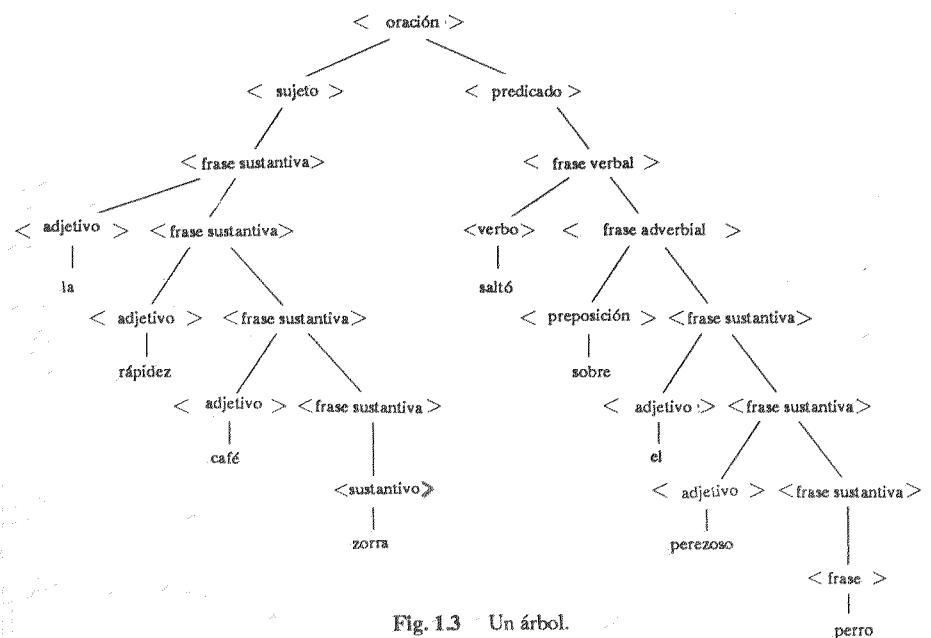


Fig. 1.3 Un árbol.

café saltó sobre el perro perezoso". En este ejemplo los vértices no se nombran, sino que se les da una "etiqueta", que puede ser una palabra o partes del discurso.

Existe una terminología especial para los árboles, que difiere de la terminología que se utiliza en un grafo cualquiera. A un sucesor de un vértice se le llama *hijo* y al predecesor *padre*. Si existe una trayectoria del vértice  $v_1$  al vértice  $v_2$ , entonces se dice que  $v_1$  es el *ancestro* de  $v_2$  y que  $v_2$  es el descendiente de  $v_1$ . Nótese que no se tiene una regla para el caso  $v_1 = v_2$ ; cualquier vértice es un ancestro y descendiente de sí mismo. Un vértice que no posee hijos se conoce como *hoja* y a los otros vértices se les llama *vértices interiores*. Por ejemplo, en la Fig. 1.3, el vértice con la etiqueta <verbo> es un hijo del vértice que tiene la etiqueta <frase verbal>, y este último es el padre del primero. El vértice con la etiqueta "perro" es un descendiente de sí mismo, el vértice con la etiqueta <frase verbal>, el vértice que posee la etiqueta <coración> y otros seis vértices más. Los vértices etiquetados con palabras de la oración son las hojas, y los que están señalados con partes del discurso entre paréntesis son los vértices interiores.

### 1.3 PRUEBAS INDUCTIVAS

La prueba de muchos de los teoremas de este libro se hace mediante inducción matemática. Supóngase que tenemos una proposición  $P(n)$  sobre un entero negativo  $n$ . Un ejemplo que se utiliza con frecuencia es tomar  $P(n)$  como

$$\sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6} \quad (1.1)$$

El principio de la inducción matemática consiste en que  $P(n)$  se obtiene de

- a)  $P(0)$ , y
- b)  $P(n-1)$  implica que  $P(n)$  vale para  $n \geq 1$ .

A la condición (a) en una prueba inductiva se le conoce como *base*, y a la condición (b) como *paso inductivo*. La parte izquierda de (b), es decir  $P(n-1)$ , se conoce como *hipótesis inductiva*.†

**Ejemplo 1.1** Probemos (1.1) mediante inducción matemática. Establecemos (a) haciendo  $n$  igual a cero en (1.1) y observamos que en ambos lados de la ecuación se tiene cero. Para probar (b), sustituimos  $n-1$  por  $n$  en (1.1) y tratamos de probar (1.1) del resultado obtenido. Es decir, debemos mostrar que para  $n \geq 1$

$$\sum_{i=0}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6} \text{ implica que } \sum_{i=0}^n i^2 = \frac{n(n+1)(2n+1)}{6}.$$

† La inducción matemática consiste en probar la proposición a demostrar para  $n = 0$ ; una vez hecho esto se supone válida para  $n-1$  y se prueba para  $n$  (N. del T.).

Puesto que

$$\sum_{i=0}^n i^2 = \sum_{i=0}^{n-1} i^2 + n^2,$$

y como nos dieron

$$\sum_{i=0}^{n-1} i^2 = \frac{(n-1)n(2n-1)}{6},$$

sólo necesitamos mostrar que

$$\frac{(n-1)n(2n-1)}{6} + n^2 = \frac{n(n+1)(2n+1)}{6}.$$

La última igualdad se sigue de una simple manipulación algebraica, con lo que queda probada (1.1).

### 1.4 NOTACION DE CONJUNTOS

Suponemos que el lector conoce el concepto de *conjunto*, una colección de objetos (*elementos* de ese conjunto) sin repetirse. Los conjuntos finitos pueden especificarse haciendo una lista de sus elementos. Por ejemplo, usamos  $\{0, 1\}$  para denotar el alfabeto de símbolos 0 y 1. También especificamos a los conjuntos con un *formador de conjuntos*:

$$\{x \mid P(x)\}, \quad (1.2)$$

$$\text{o} \quad \{x \text{ en } A \mid P(x)\}. \quad (1.3)$$

La proposición (1.2) se lee "el conjunto de objetos  $x$  de modo que  $P(x)$  es verdadera", en la que  $P(x)$  es alguna proposición sobre los objetos  $x$ . La proposición (1.3) es "el conjunto de  $x$  objetos del conjunto  $A$  de modo que  $P(x)$  es verdadera", y es equivalente a  $\{x \mid P(x) \text{ y } x \text{ está en } A\}$ . Por ejemplo:

$$\{i \mid i \text{ es un entero y existe un entero } j \text{ de modo que } i = 2j\}$$

es una manera de especificar el conjunto de los enteros pares.

Si cada elemento de  $A$  es elemento de  $B$ , entonces escribimos  $A \subseteq B$ , y diremos que  $A$  está *contenido* en  $B$ .  $A \supseteq B$  es lo mismo que  $B \subseteq A$ . Si  $A \subseteq B$  pero  $A \neq B$ , es decir, cada elemento de  $A$  está en  $B$  y existe algún elemento de  $B$  que no está en  $A$ , entonces escribiremos  $A \subsetneq B$ . Los conjuntos  $A$  y  $B$  son *iguales* si tienen los mismos elementos. Esto es,  $A = B$  si y sólo si  $A \subseteq B$  y  $B \subseteq A$ .

### Operaciones sobre los conjuntos

Las operaciones habituales que se definen sobre los conjuntos son:

1.  $A \cup B$ , la unión de  $A$  con  $B$ , es

$$\{x \mid x \text{ está en } A \text{ o } x \text{ está en } B\}.$$

2.  $A \cap B$ , la *intersección* de  $A$  y  $B$ , es

$$\{x \mid x \text{ está en } A \text{ y } x \text{ está en } B\}.$$

3.  $A - B$ , la *diferencia* de  $A$  y  $B$ , es

$$\{x \mid x \text{ está en } A \text{ y } x \text{ no está en } B\}.$$

4.  $A \times B$ , el *producto cartesiano* de  $A$  y  $B$ , es el conjunto de pares ordenados  $(a, b)$  tales que  $a$  está en  $A$  y  $b$  está en  $B$ .

5.  $2^A$ , el *conjunto potencia* de  $A$ , es el conjunto formado por todos los subconjuntos de  $A$ .

**Ejemplo 1.2** Sea  $A = \{1, 2\}$  y  $B = \{2, 3\}$ . Entonces

$$\begin{aligned} A \cup B &= \{1, 2, 3\}, \quad A \cap B = \{2\}, \quad A - B = \{1\}, \\ A \times B &= \{(1, 2), (1, 3), (2, 2), (2, 3)\}, \end{aligned}$$

y

$$2^A = \{\emptyset, \{1\}, \{2\}, \{1, 2\}\}.$$

Nótese que si  $A$  y  $B$  tienen  $n$  y  $m$  elementos respectivamente, entonces  $A \times B$  tiene  $nm$  elementos y  $2^A$  tiene  $2^n$  elementos.

## Conjuntos infinitos

Nuestra intuición, cuando nos extendemos a conjuntos infinitos, puede ser engañosa. Dos conjuntos  $S_1$  y  $S_2$  tienen la misma *cardinalidad* (número de miembros) si existe una transformación uno a uno de elementos de  $S_1$  a  $S_2$ . Para los conjuntos finitos, si  $S_1$  es un conjunto propio† de  $S_2$ , entonces  $S_1$  y  $S_2$  tienen diferente cardinalidad. Sin embargo, si  $S_1$  y  $S_2$  son infinitos, la última proposición puede ser falsa. Sea  $S_1$  el conjunto de los enteros pares y  $S_2$  el conjunto de todos los enteros. Es claro que  $S_1$  es un subconjunto propio de  $S_2$ . No obstante,  $S_1$  y  $S_2$  tienen la misma cardinalidad, puesto que la función  $f$  definida como  $f(2i) = i$  es una transformación uno a uno de los enteros pares a los enteros.

No todos los conjuntos finitos tienen la misma cardinalidad. Considérese el conjunto de todos los enteros y el de todos los reales. Supóngase que el conjunto de los reales puede ponerse en una correspondencia uno a uno con los enteros. Entonces considere el número real cuyo  $i$ -ésimo dígito después del punto decimal es el  $i$ -ésimo dígito del  $i$ -ésimo real más  $5 \bmod 10$ . Este número real no puede estar en correspondencia con un entero puesto que difiere de cada real que ha sido transformado a un entero. De lo anterior, concluimos que los reales no pueden situarse en una correspondencia de

† Se dice que un subconjunto  $A$  de  $B$  es *propio*, si  $A \neq B$  (N. del T.).

uno a uno con los enteros. De manera intuitiva notamos que existen demasiados reales para hacerlo. La construcción hecha más arriba se conoce como *diagonalización* y constituye una herramienta importante en la ciencia de la computación.

Los conjuntos que pueden ponerse en correspondencia uno a uno con los enteros se dice que son *infinitos contables*, o simplemente *contables*. Los números racionales y el conjunto  $\Sigma^*$  de las cadenas de longitud finita, formadas con el alfabeto  $\Sigma$ , son infinitos contables. El conjunto de todos los subconjuntos de  $\Sigma^*$  y el conjunto de todas las funciones que transforman a los enteros a  $\{0, 1\}$  son de la misma cardinalidad que los reales, y no son contables.

## 1.5 RELACIONES

Una *relación* (binaria) es un conjunto de pares. El primer componente de cada par se toma de un conjunto llamado *dominio*, y el segundo componente de cada par pertenece a un conjunto (tal vez distinto del primero) llamado *contradominio*.† Al principio sólo utilizaremos relaciones en las que el dominio y contradominio son el mismo conjunto  $S$ . En este caso diremos que la relación es *sobre*  $S$ . Si  $R$  es una relación y  $(a, b)$  es un par en  $R$ , entonces, con frecuencia, escribiremos  $aRb$ .

### Propiedades de las relaciones

Decimos que una relación  $R$  sobre  $S$  es

1. *reflexiva* si  $aRa$  para toda  $a$  en  $S$ ;
2. *irreflexiva* si  $aRa$  es falsa para toda  $a$  en  $S$
3. *transitiva* si  $aRb$  y  $bRc$  implican  $aRc$ ;
4. *simétrica* si  $aRb$  implica  $bRa$ ;
5. *asimétrica* si  $aRb$  implica que  $bRa$  es falsa.

Nótese que cualquier relación asimétrica deberá ser irreflexiva.

**Ejemplo 1.3** La relación  $<$  sobre el conjunto de los enteros es transitiva porque  $a < b$  y  $b < c$  implica que  $a < c$ . Es asimétrica y por tanto irreflexiva porque  $a < b$  implica que  $b < a$  es falsa.

† O *codominio*. Muchos autores suelen llamar a este conjunto *rango*, del inglés *range*, que es una mala traducción, pues *rango* en español tiene calidad de *ategoría* o *jerarquía* y no de *intervalo* o *gama* como en inglés (N. del T.).

## Relaciones de equivalencia

Se dice que una relación  $R$  que es reflexiva, simétrica y transitiva es una relación de *equivalencia*. Una propiedad importante de una relación de equivalencia  $R$  sobre un conjunto  $S$  es que  $R$  divide a  $S$  en clases de equivalencia no vacías disjuntas (véase Ejercicio 1.8 y su solución). Lo anterior significa que  $S = S_1 \cup S_2 \cup \dots$ , en donde para cada  $i$  y  $j$ ,  $i \neq j$ :

1.  $S_i \cap S_j = \emptyset$ ;
2. para cada  $a$  y  $b$  en  $S_i$ ,  $aRb$  es verdadera;
3. para cada  $a$  y  $b$  en  $S_j$ ,  $aRb$  es falsa.

Los conjuntos  $S_i$  se conocen como *clases de equivalencia*. Nótese que el número de clases puede ser infinito.

**Ejemplo 1.4** Un ejemplo común de una relación de equivalencia lo constituye la congruencia módulo un entero  $m$ . Escribimos  $i \equiv_m j$  o  $i \equiv j \pmod{m}$  si  $i$  y  $j$  son enteros tales que  $i - j$  es divisible entre  $m$ . El lector puede comprobar fácilmente que  $\equiv_m$  es reflexiva, transitiva y simétrica. Las clases de equivalencia de  $\equiv_m$  son en número  $m$ :

$$\begin{aligned} & \{\dots, -m, 0, m, 2m, \dots\}, \\ & \{\dots, -(m-1), 1, m+1, 2m+1, \dots\}, \\ & \vdots \\ & \{\dots, -1, m-1, 2m-1, 3m-1, \dots\}. \end{aligned}$$

## Cerraduras de las relaciones

Supóngase que  $\mathcal{P}$  es un conjunto de propiedades de las relaciones. La *cerradura*  $\mathcal{P}^*$  de una relación  $R$  es la relación más pequeña  $R'$  que incluye a todos los pares de  $R$  y posee las propiedades de  $\mathcal{P}$ . Por ejemplo, la *cerradura transitiva* de  $R$ , denotada como  $R^+$ , está definida por:

1. Si  $(a, b)$  está en  $R$ , entonces  $(a, b)$  está en  $R^+$ .
2. Si  $(a, b)$  está en  $R^+$  y  $(b, c)$  está en  $R$ , entonces  $(a, c)$  está en  $R^+$ .
3. No hay nada en  $R^+$  a menos que se haya obtenido de los incisos (1) y (2).

Deberá resultar evidente que cualquier par colocado en  $R^+$  mediante las reglas (1) y (2) le pertenece, de otra manera  $R^+$  podría no incluir a  $R$  o no ser transitiva. Una sencilla prueba inductiva muestra, también, que  $R^+$  es de hecho transitiva. Así,  $R^+$  incluye a  $R$ , es transitiva y contiene el mismo número de pares que cualquier relación que incluya a  $R$  y sea transitiva.

La *cerraduras reflexiva y transitiva* de  $R$ , denotada  $R^*$ , es  $R^+ \cup \{(a, a) \mid a \text{ está en } S\}$ .

**Ejemplo 1.5** Sea  $R = \{(1, 2), (2, 2), (2, 3)\}$  una relación sobre el conjunto  $\{1, 2, 3\}$ . Entonces

$$R^+ = \{(1, 2), (2, 2), (2, 3), (1, 3)\},$$

y

$$R^* = \{(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)\}.$$

## 1.6 SINOPSIS DEL LIBRO

La ciencia de la computación es el cuerpo sistematizado del conocimiento concerniente al cálculo. Sus principios pueden remontarse al diseño de algoritmos por parte de Euclides y del uso de la complejidad asintótica y la reducibilidad por los babilonios (Hogben, 1955). En la actualidad, sin embargo, el interés está modelado por dos importantes eventos: el advenimiento de las computadoras digitales modernas, capaces de realizar muchos millones de operaciones por segundo, y la formalización del concepto de procedimiento efectivo, que trajo como consecuencia la existencia de funciones probablemente no calculables.

La ciencia de la computación consta de dos componentes principales: primero, las ideas y modelos fundamentales subyacentes en el cálculo; y, segundo, las técnicas de ingeniería para el diseño de sistemas de computación, tanto *hardware* como *software*, en especial la aplicación de la teoría al diseño. El presente libro está pensado como una introducción a la primera área, las ideas fundamentales que son la base del cálculo, aunque haremos breves anotaciones sobre las más importantes aplicaciones.

La ciencia de la computación teórica tuvo sus inicios en varios campos distintos: el de los biólogos que estudiaban modelos para las redes neuronales; el de los ingenieros eléctricos que trabajan en el desarrollo de la teoría de la interrupción como una herramienta para el diseño de *hardware*; el trabajo de los matemáticos sobre los fundamentos de la lógica; y el de los lingüistas en la investigación de las gramáticas de los lenguajes naturales. A partir de todos estos estudios nacieron los modelos centrales de la ciencia de la computación teórica.

Los conceptos de autómatas finitos y expresiones regulares (capítulos 2 y 3) fueron desarrollados originalmente pensando en las redes neuronales y los circuitos de interrupción. Recientemente, han servido como útiles herramientas en el diseño de analizadores de léxico, que es la parte de un compilador que agrupa caracteres en tokens: unidades indivisibles como los nombres de variable y las palabras clave. Certo número de sistemas compiladores de escritura transforman de manera automática expresiones regulares en autómatas finitos que serán utilizados como analizadores de léxico. Se han encontrado varios otros usos para las expresiones regulares y los autómatas finitos en editores de textos, concordancia de patrones, diferentes procesadores de textos y programas para buscar archivos, y como conceptos matemáticos con aplicación a otras áreas, como la lógica. Al final del capítulo 2 delinearemos algunas de las aplicaciones de esta teoría.

La noción de gramática libre de contexto y la correspondiente de autómata de apilamiento (capítulos 4 al 6) han contribuido enormemente a la especificación de

lenguajes de programación y al diseño de *parsers*:<sup>†</sup> otra parte clave de un compilador. Las especificaciones formales de los programas de computación han sustituido a las descripciones de lenguajes extensas y, con frecuencia, incompletas o ambiguas. El entendimiento de las capacidades del autómata de apilamiento ha simplificado mucho el análisis gramatical. Es interesante observar que el diseño de *parsers* fue un difícil problema para los primeros compiladores, y que muchos de los primeros *parsers* fueron bastante inefficientes e innecesariamente restrictivos. Ahora, gracias a la expansión del conocimiento sobre una variedad de técnicas basadas en la gramática libre de contexto, el diseño de analizadores de gramática ya no es un problema, y el análisis gramatical ocupa sólo un porcentaje bajo del tiempo utilizado en una compilación normal. En el capítulo 10 haremos un esbozo de las principales formas en que se pueden construir *parsers* eficientes que se comportan como autómatas de apilamiento a partir de ciertos tipos de gramáticas libres de contexto.

En el capítulo 7 nos encontramos con las máquinas de Turing y nos enfrentamos a uno de los problemas fundamentales de la ciencia de la computación; esto es, existen más funciones que nombres para definirlas o que algoritmos para calcularlas. Por tanto, nos enfrentamos a la existencia de funciones que simplemente no son calculables; es decir, no existe un programa de computación que pueda ser escrito, el cual, dado un argumento para la función, produzca el valor de la función para dicho argumento y funcione para todos los argumentos posibles.

Supóngase que para cada función calculable existe un programa de computación o un algoritmo que la calcule, y también que cualquier programa o algoritmo pueda ser especificado de manera finita. De esta manera, los programas de computación no son más que cadenas finitas de símbolos sobre algún alfabeto finito. De aquí que el conjunto de todos los programas de computación sea contablemente infinito. Considere ahora funciones que transformen a los enteros a 0 y 1. Suponga que el conjunto de todas las funciones de ese tipo es contablemente infinito y que tales funciones se han puesto en correspondencia con los enteros. Sea  $f_i$  la función correspondiente al  $i$ -ésimo entero. Entonces la función

$$f(n) = \begin{cases} 0 & \text{si } f_n(n) = 1 \\ 1 & \text{de otra manera} \end{cases}$$

no puede corresponder a ningún entero, lo que constituye una contradicción. [Si  $f(n) = f_i(n)$ , entonces tenemos la contradicción  $f(f) = f_i(f)$  y  $f(f) \neq f_i(f)$ .] Este argumento se formaliza en los capítulos 7 y 8, donde veremos que ciertos problemas fáciles de establecer no pueden resolverse con la computadora, aun pensando que, a primera vista, parecen ser sensibles al cálculo.

Sin embargo, podemos hacer algo más que el decir si un problema puede resolverse o no con una computadora. El solo hecho de que un problema se pueda resolver no significa que exista un algoritmo práctico para resolverlo. En el capítulo 12 veremos que hay problemas abstractos que se pueden resolver con una computadora, pero que requieren cantidades extraordinarias de tiempo y/o espacio para su solución.

Más adelante, en el capítulo 13, descubriremos que existen problemas reales e importantes que también caen en esta categoría. La naciente teoría de los “problemas intratables” está destinada a influir profundamente en la forma de enfocar nuestros problemas.

## EJERCICIOS

### 1.1 En el árbol de la Fig. 1.4,

- ¿qué vértices son hojas, y cuáles son vértices interiores?
- ¿qué vértices son los hijos de 5?
- ¿qué vértice es el padre de 5?
- ¿cuál es la longitud de la trayectoria de 1 a 9?
- ¿cuál vértice es la raíz?

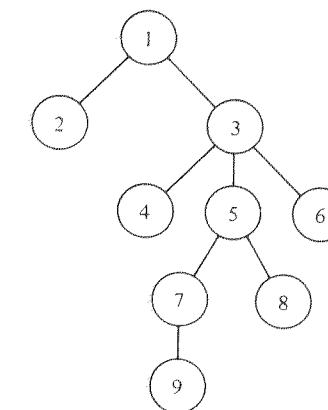


Fig. 1.4 Un árbol.

### 1.2 Pruebe por inducción sobre $n$ que

$$\text{a)} \sum_{i=0}^n i = \frac{n(n+1)}{2} \quad \text{b)} \sum_{i=0}^n i^3 = \left( \sum_{i=0}^n i \right)^2$$

\* 1.3 Un palíndromo puede definirse como una cadena que se lee igual de izquierda a derecha y viceversa, o mediante la siguiente definición.

- $\epsilon$  es un palíndromo.
- Si  $a$  es cualquier símbolo, entonces la cadena  $a$  es un palíndromo.
- Si  $a$  es cualquier símbolo y  $x$  es un palíndromo, entonces  $axa$  es un palíndromo.
- Ninguna cadena será palíndromo, a menos que se forme a través de (1) a (3).

Pruebe por inducción que las dos definiciones son equivalentes.

<sup>†</sup> Del inglés *parse*, que significa analizar gramaticalmente (N. del T.).

\*1.4 Las cadenas de paréntesis balanceados pueden definirse, al menos, de dos formas.

1. Una cadena  $w$  sobre el alfabeto  $\{(, )\}$  está balanceada si, y únicamente si:
  - a)  $w$  tiene igual número de  $($  y  $)$ .
  - b) cualquier prefijo de  $w$  tiene al menos tantos  $($  como  $)$ .
2. a)  $\epsilon$  está balanceada.
- b) Si  $w$  es una cadena balanceada, entonces  $(w)$  está balanceada.
- c) Si  $w$  y  $x$  son cadenas balanceadas, entonces también lo será  $wx$ .
- d) Cualquier otra cadena no está balanceada.

Pruebe por inducción sobre la longitud de una cadena que las definiciones (1) y (2) definen a la misma clase de cadenas.

\*1.5 ¿Qué está mal en la siguiente “prueba” inductiva sobre el hecho de que todos los elementos de un conjunto cualquiera deben ser idénticos? Para los conjuntos con un elemento la proposición es trivialmente verdadera. Suponga que la proposición es válida para conjuntos con  $n-1$  elementos, y considere un conjunto  $S$  con  $n$  elementos. Sea  $a$  un elemento de  $S$ . Escriba  $S = S_1 \cup S_2$ , en donde  $S_1$  y  $S_2$  tienen cada uno  $n-1$  elementos, y ambos contienen a  $a$ . Por la hipótesis inductiva todos los elementos de  $S_1$  son idénticos a  $a$  y de manera similar todos los elementos de  $S_2$  son idénticos a  $a$ . Por tanto, todos los elementos de  $S$  son idénticos a  $a$ .

1.6 Muestre que las siguientes son relaciones de equivalencia y dé sus clases de equivalencia.

- a) La relación  $R_1$  sobre los enteros, definida por  $iR_1 j$  si y sólo si  $i = j$ .
- b) La relación  $R_2$  sobre la gente, definida por  $pR_2 q$  si y sólo si  $p$  y  $q$  nacieron en la misma hora, el mismo día de cualquier año.
- c) Igual que (b), sólo que en el mismo año, en lugar de cualquier año.

1.7 Encuentre las cerraduras transitiva, reflexiva y transitiva, y la simétrica de la relación.

$$\{(1, 2), (2, 3), (3, 4), (4, 5)\}.$$

\*S1.8 Pruebe que cualquier relación de equivalencia  $R$  sobre un conjunto  $S$ , parte a  $S$  en clases de equivalencia disjuntas.

\*1.9 Dé un ejemplo de una relación que sea simétrica y transitiva, pero que no sea reflexiva. [Fíjese cuándo se necesita la reflexividad para mostrar que una relación de equivalencia define a las clases de equivalencia; véase la solución al Ejercicio 1.8.]

\*1.10 Pruebe que cualquier subconjunto de un conjunto infinito contable es finito o infinito contable.

\*1.11 Pruebe que el conjunto de todos los pares ordenados de enteros es infinito contable.

1.12 ¿Es la unión de una colección infinita contable de conjuntos infinitos contables, infinita contable? ¿Lo es el producto cartesiano?

#### Soluciones a los ejercicios seleccionados

1.3 Es claro que cada cadena que satisface la segunda condición se lee igual de izquierda a derecha y de derecha a izquierda. Supóngase que  $x$  se lee de la misma manera. Probamos por

inducción sobre la longitud de  $x$  que las xs que son palíndromos se concluyen de las reglas (1) a (3). Si  $|x| \leq 1$ , entonces  $x$  es o bien  $\epsilon$  o un solo símbolo  $a$ , y, por tanto, vale la regla (1) o (2). Si  $|x| > 1$ , entonces  $x$  comienza y termina con el mismo símbolo  $a$ . Por tanto,  $x = awa$ , en donde  $w$  se lee igual de izquierda a derecha y viceversa y tiene una longitud menor a la de  $x$ . Por la hipótesis de inducción, las reglas (1) a (3) implican que  $w$  es un palíndromo. Así pues, por la regla (3),  $x = awa$  es un palíndromo.

1.8 Sea  $R$  una relación de equivalencia sobre  $S$  y supóngase que  $a$  y  $b$  son elementos de  $S$ . Sean  $C_a$  y  $C_b$  las clases de equivalencia que contienen a  $a$  y  $b$  respectivamente; es decir,  $C_a = \{c \mid aRc\}$  y  $C_b = \{c \mid bRc\}$ . Demostraremos que  $C_a = C_b$  o que  $C_a \cap C_b = \emptyset$ . Supóngase que  $C_a \cap C_b \neq \emptyset$ ; sea  $d$  elemento de  $C_a \cap C_b$  y  $e$  elemento cualquiera de  $C_a$ . Por tanto  $aRe$ . Como  $d$  está en  $C_a \cap C_b$  tenemos  $aRd$  y  $bRd$ . Por simetría  $dRa$ . Por transitividad (dos veces),  $bRa$  y  $bRe$ . Así pues,  $e$  está en  $C_b$  y de aquí que  $C_a \subseteq C_b$ . Un procedimiento similar prueba que  $C_b \subseteq C_a$ , así que  $C_a = C_b$ . Por tanto, las clases de equivalencia distintas son disjuntas. Para mostrar que las clases forman una partición sólo tenemos que observar que, por reflexividad, cada  $a$  está en la clase de equivalencia  $C_a$ , de tal modo que la unión de las clases de equivalencia es  $S$ .

## CAPITULO

# 2

## AUTOMATAS FINITOS Y EXPRESIONES REGULARES

UNIVERSIDAD DE LA REPUBLICA  
FACULTAD DE INGENIERIA  
DEPARTAMENTO DE  
DOCUMENTACION Y BIBLIOTECA  
MONTEVIDEO - URUGUAY

### 2.1 SISTEMAS DE ESTADOS FINITOS

El autómata finito es un modelo matemático de un sistema, con entradas y salidas discretas. El sistema puede estar en cualquiera de un número finito de configuraciones o "estados". El estado del sistema resume la información concerniente a entradas anteriores y que es necesaria para determinar el comportamiento del sistema para entradas posteriores. El mecanismo de control de un elevador es un buen ejemplo de un sistema de estados finitos. El mecanismo no recuerda todas las demandas previas de servicio, sino sólo el piso en el que se encuentra, la dirección de movimiento (hacia arriba o hacia abajo) y el conjunto de demandas de servicio aún no satisfechas.

En la ciencia de la computación encontramos muchos ejemplos de sistemas de estados finitos, y la teoría de autómatas finitos es una útil herramienta para el diseño de tales sistemas. Un primer ejemplo es un circuito de interrupción, como la unidad de control de una computadora. Un circuito de interrupción está compuesto por un número finito de compuertas, cada una de las cuales puede estar en una de dos condiciones, por lo general denotadas por 0 y 1. Estas condiciones podrían, en términos electrónicos, ser dos niveles de voltaje distintos en la compuerta de salida. El estado de una red de interrupción con  $n$  compuertas es, por lo tanto, una de las  $2^n$  asignaciones de 0 y 1 para las diferentes compuertas. Aunque el voltaje en cada compuerta pueda tomar cualquier de un conjunto infinito de valores, el circuito electrónico está diseñado de tal manera que sólo los dos voltajes correspondientes a 0 y 1 son estables, y otros voltajes se ajustarán, casi inmediatamente, a uno de esos voltajes. Los circuitos de interrupción se diseñan intencionalmente de esta manera, de modo que puedan ser vistos como

sistemas de estado finito, separando, por ello, el diseño lógico de una computadora de la implementación electrónica.

Ciertos programas, utilizados con frecuencia, como los editores de texto y los analizadores de léxico que se encuentran en la mayoría de los compiladores, son diseñados como sistemas de estado finito. Por ejemplo, un analizador de léxico recorre los símbolos de un programa de computación para localizar las cadenas de caracteres correspondientes a identificadores, constantes numéricas, palabras reservadas, etcétera. En este proceso el analizador de léxico necesita recordar solamente una cantidad finita de información, como qué longitud de un prefijo de una palabra reservada ha visto desde el inicio. La teoría de los autómatas finitos se utiliza mucho en el diseño de procesadores eficientes de cadenas de este tipo y de otros. En la Sección 2.8 mencionamos algunas de estas aplicaciones.

La computadora misma puede ser vista como un sistema de estado finito, aunque hacer esto no resulta tan útil como uno quisiera. Teóricamente, el estado del procesador central, memoria principal y almacenamiento auxiliar en cualquier instante de tiempo es uno de un gran número finito de estados. Estamos suponiendo, por supuesto, que existe un número fijo de discos, tambores, cintas, etcétera, disponibles, y que uno no puede extender la memoria indefinidamente. Tornar una computadora como un sistema de estado finito, sin embargo, no es realista o satisfactorio matemáticamente. Establece un límite artificial a la capacidad de memoria. Y, por tanto, no es capaz de capturar la esencia real de la computación. Para captar de manera apropiada el concepto de computación se necesita una memoria potencialmente infinita, aun cuando cada instalación sea finita. Los modelos infinitos de computadora se discutirán en los Capítulos 7 y 8.

Resulta tentador, también, considerar al cerebro humano como un sistema de estado finito. El número de células cerebrales o *neuronas* es limitado, cuando mucho  $2^{35}$ . Podemos considerar, aunque existe evidencia de lo contrario, que el estado de cada neurona puede describirse mediante un pequeño número de *bits*. Si esto es así, la teoría del estado finito se aplica al cerebro. Sin embargo, el número de estados es tan grande que es poco probable que este planteamiento tenga como resultado observaciones útiles sobre el cerebro humano, no más que las suposiciones de estado finito ayudan a entender sistemas de computación grandes pero finitos.

Tal vez la razón más importante para estudiar los sistemas de estado finito es la naturalidad del concepto, como lo indica el hecho de que surge en diferentes y múltiples lugares. Esto es una indicación de que hemos captado el concepto de un tipo fundamental de sistemas, un tipo que es rico en estructura y aplicaciones potenciales.

### Un ejemplo

Antes de definir de manera formal los sistemas de estado finito, consideremos un ejemplo. En la orilla izquierda de un río se encuentra un hombre, junto con un lobo, una cabra y una col. Hay un bote con la capacidad suficiente para llevar al hombre y a uno de los otros tres. El hombre con la col y demás compañeros deben cruzar el río, y el hombre puede llevar a uno solo a la vez. Sin embargo, si el hombre deja solos al lobo

y a la cabra en cualquier lado del río, con toda seguridad que el lobo se comerá a la cabra. Del mismo modo, si la cabra y la col se quedan juntas, la cabra se comerá a la col. ¿Es posible que se pueda cruzar el río sin que nada sea comido por nadie?

El problema se puede modelar observando que la información pertinente está en los ocupantes de cada orilla después de cruzar. Existen 16 subconjuntos del hombre (*H*), el lobo (*L*), la cabra (*G*) y la col (*C*). Un estado corresponde al subconjunto que se encuentra en la ribera izquierda. Los estados están etiquetados por parejas separadas por un guión, por ejemplo *HG-LC*, en donde los símbolos que están a la izquierda del guión denotan el subconjunto que está en la orilla izquierda del río; los símbolos que están a la derecha del guión representan el subconjunto que se encuentra en la orilla opuesta. Algunos de los 16 estados, como *GC-HL*, son fatales y nunca deben accesarse al sistema.

Las "entradas" al sistema son las acciones del hombre. Puede cruzar solo (entrada *h*), con el lobo (entrada *l*), la cabra (entrada *g*) o con la col (entrada *c*). El estado inicial es *HLGC-Ø* y el estado final es *Ø-HLGC*. En la Fig. 2.1 se muestra el diagrama de transición.

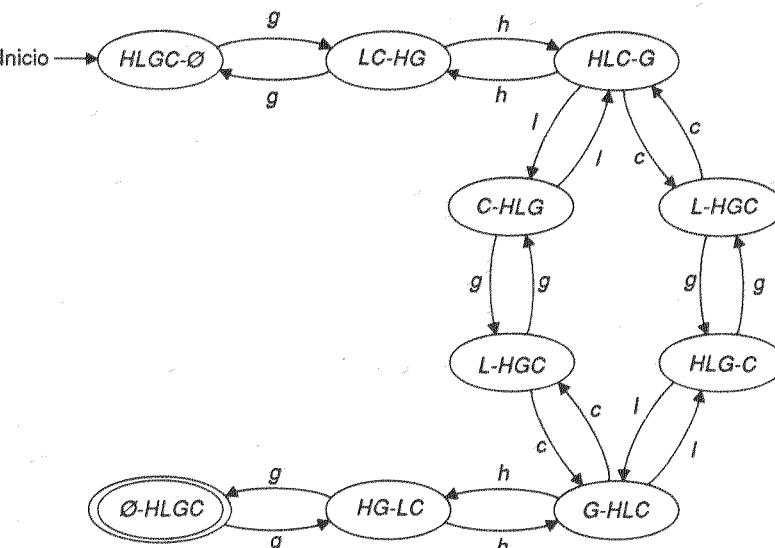


Fig. 2.1 Diagrama de transiciones para el problema del hombre, el lobo, la cabra y la col.

Hay dos soluciones del problema igualmente cortas, como puede verse si se buscan las trayectorias que van del estado inicial al final (que está dentro de un círculo doble). Existe un número infinitamente grande de soluciones diferentes del problema, pero todas, con excepción de dos, involucran ciclos inútiles. El sistema de estado finito puede ser considerado como el que define a un lenguaje infinito, el conjunto de todas las cadenas que etiquetan las trayectorias que van del estado inicial al final.

Antes de seguir adelante, es necesario anotar que hay, al menos, dos formas

importantes en las que el ejemplo anterior es característico de los sistemas de estado finito. Primero, sólo existe un estado final; en general puede haber muchos. Segundo sucede que para cada transición existe una transición inversa sobre el mismo símbolo lo que no necesariamente sucede en el caso general. Nótese también que el término "estado final", aunque es tradicional, no significa que el cálculo debe terminar cuando se alcanza. Podríamos continuar haciendo transiciones, por ejemplo, al estado *HG-LC* del ejemplo anterior.

## 2.2 DEFINICIONES BASICAS

Un *autómata finito* (FA)<sup>†</sup> consiste en un conjunto finito de estados y un conjunto de transiciones de estado a estado, que se dan sobre símbolos de entrada tomados de un alfabeto  $\Sigma$ . Para cada símbolo de entrada existe exactamente una transición a partir de cada estado (posiblemente de regreso al mismo estado). Un estado, por lo general denotado como  $q_0$ , es el estado inicial, en el que el autómata comienza. Algunos estados están designados como final o de aceptación.

A un FA se le asocia un grafo dirigido, conocido como *diagrama de transiciones* ( $Q, \Sigma, \delta, q_0, F$ ), en el que  $Q$  es un conjunto finito de *estados*,  $\Sigma$  un *alfabeto de entrada* finito,  $q_0$ , elemento de  $Q$ , el *estado inicial*,  $F \subseteq Q$  el conjunto de *estados finales* y  $\delta$  la *función de transición* que transforma  $Q \times \Sigma$  en  $Q$ . Esto es,  $\delta(q, a)$  es un estado para cada estado  $q$  y símbolo de entrada  $a$ .

**Ejemplo 2.1** En la Fig. 2.2 se ilustra el diagrama de transiciones de un FA. El estado inicial,  $q_0$ , está indicado por la flecha con la etiqueta "inicio". Existe sólo un estado final, también  $q_0$  en este caso, indicado por el doble círculo. El FA acepta todas las cadenas de ceros y unos, en las que el número de 0s y el número de 1s son pares. Para comprender esto, considere el "control" según se traslada de estado en estado en el diagrama. El control comienza en  $q_0$  y debe terminar en  $q_0$ , si se acepta la secuencia de entrada. Cada entrada 0 ocasiona que el control cruce la línea horizontal a–b, mientras que una entrada 1 no. Por tanto, el control permanece en un estado por encima de la recta a–b si y sólo si la entrada vista hasta aquí contiene un número par de ceros. De manera similar, el control se encuentra en un estado a la izquierda de la recta vertical c–d si y sólo si la entrada contiene un número par de unos. Así pues, el control está en  $q_0$  si y sólo si en la entrada existe un número par de 0s y uno par de 1s. Nótese que el FA utiliza su estado para registrar solamente la paridad del número de 0s y del de 1s, no de los números en sí, que requeriría de un número infinito de estados.

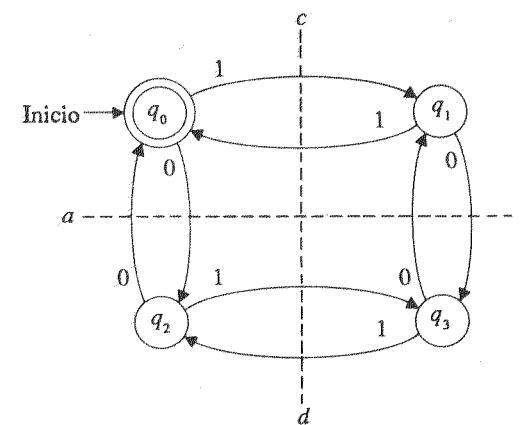


Fig. 2.2 Diagrama de transiciones de un autómata finito.

Denotamos formalmente a un *autómata finito* por el conjunto de cinco cantidades  $(Q, \Sigma, \delta, q_0, F)$ , en el que  $Q$  es un conjunto finito de *estados*,  $\Sigma$  un *alfabeto de entrada* finito,  $q_0$ , elemento de  $Q$ , el *estado inicial*,  $F \subseteq Q$  el conjunto de *estados finales* y  $\delta$  la *función de transición* que transforma  $Q \times \Sigma$  en  $Q$ . Esto es,  $\delta(q, a)$  es un estado para cada estado  $q$  y símbolo de entrada  $a$ .

Visualizamos a un FA como un *control finito*, que se encuentra en algún estado de  $Q$ , y que lee una secuencia de símbolos de  $\Sigma$  escritos en una cinta, según se muestra en la Fig. 2.3. En un movimiento, el FA que se encuentra en el estado  $q$  y está leyendo el símbolo  $a$ , entra al estado  $\delta(q, a)$  y mueve su cabeza un símbolo hacia la derecha. Si  $\delta(q, a)$  es un estado de aceptación, entonces se considera que el FA ha aceptado la cadena escrita en su cinta de entrada hasta la posición a la que se ha movido la cabeza, si la cabeza se ha movido fuera del extremo derecho de la cinta, entonces el FA acepta a la cinta completa. Nótese que conforme un FA barre una cinta, puede aceptar muchos prefijos diferentes.

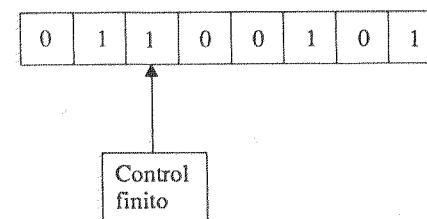


Fig. 2.3 Un autómata finito.

<sup>†</sup> Como el lector podrá observar, las iniciales que utilizamos a lo largo del texto corresponden a las del inglés, por ser éstas de uso generalizado. En este caso, FA corresponde al concepto, *finite automaton* (N. del T.).

Para describir formalmente el comportamiento de un FA sobre una cadena, necesitamos extender la función de transición  $\delta$  para que se pueda aplicar a un estado

y una cadena, más que a un estado y un símbolo. Definimos una función  $\hat{\delta}$  de  $Q \times \Sigma^*$  a  $Q$ . La intención es que  $\hat{\delta}(q, w)$  represente al estado en el que estará el FA después de leer la cadena  $w$  del estado  $q$ . Dicho de otra forma,  $\hat{\delta}(q, w)$  es el único estado  $p$  tal que existe una trayectoria en el diagrama de transiciones de  $q$  a  $p$ , con la etiqueta  $w$ . De manera formal, definimos

1.  $\hat{\delta}(q, \epsilon) = q$ , y
2. para todas las cadenas  $w$  y símbolos de entrada  $a$ ,

$$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a).$$

De esta forma (1) establece que sin leer un símbolo de entrada el FA no puede cambiar de estado, (2) nos dice cómo encontrar el estado después de leer una cadena entrada no vacía  $wa$ . Esto es, hallar el estado  $p = \hat{\delta}(q, w)$  después de leer  $w$ . Y entonces calcular el estado  $\delta(p, a)$ .

Puesto que  $\delta(q, a) = \delta(\hat{\delta}(q, \epsilon), a) = \delta(q, a)$  [haciendo  $w = \epsilon$  en la regla (2) arriba], no puede haber desacuerdo entre  $\delta$  y  $\hat{\delta}$  sobre los argumentos para los cuales se definieron. En adelante, por razones de conveniencia, utilizaremos  $\delta$  en lugar de  $\hat{\delta}$ .

*Convención* Nos esforzaremos en usar los mismos símbolos para significar la misma cosa a lo largo de todo el material sobre autómatas finitos. En particular, a menos que se establezca de otra manera, el lector podrá suponer que:

1.  $Q$  es un conjunto de estados. Los símbolos  $q$  y  $p$ , con o sin subíndices, son estados.  $q_0$  es el estado inicial.
2.  $\Sigma$  es el alfabeto de entrada. Los símbolos  $a$  y  $b$ , con o sin subíndices, y los dígitos son símbolos de entrada.
3.  $\delta$  es una función de transición.
4.  $F$  es un conjunto de estados finales.
5.  $w, x, y$  y  $z$ , con o sin subíndices, son cadenas de símbolos de entrada.

Se dice que una cadena  $x$  es *aceptada* por un autómata finito  $M = (Q, \Sigma, \delta, q_0, F)$  si  $\delta(q_0, x) = p$  para algún  $p$  en  $F$ . El *lenguaje aceptado por  $M$* , denominado  $L(M)$ , es el conjunto  $\{x \mid \delta(q_0, x) \text{ está en } F\}$ . Un lenguaje es un *conjunto regular* (o simplemente *regular*) si es el conjunto aceptado por algún autómata finito.<sup>†</sup> El lector notará que cuando hablamos de un lenguaje aceptado por un autómata finito  $M$ , nos referimos al conjunto específico  $L(M)$ , no solamente cualquier conjunto de cadenas que son aceptadas por  $M$ .

**Ejemplo 2.2** Considérese de nuevo el diagrama de transición de la Fig. 2.2 en nuestra notación formal, este FA se denota como  $M = (Q, \Sigma, \delta, q_0, F)$ , en el que  $Q = \{q_0, q_1, q_2, q_3\}$ ,  $\Sigma = \{0, 1\}$ ,  $F = \{q_0\}$  y  $\delta$  como se muestra en la Fig. 2.4.

<sup>†</sup> El término "regular" viene de "expresiones regulares", formalismo que introduciremos en la Sec. 2.5 y que define la misma clase de lenguaje que los FAs.

Estados	Entradas	
	0	1
$q_0$	$q_2$	$q_1$
$q_1$	$q_3$	$q_0$
$q_2$	$q_0$	$q_3$
$q_3$	$q_1$	$q_2$

Fig. 2.4  $\delta(q, a)$  para el FA de la Fig. 2.2.

Supóngase que el número 110101 es una entrada para  $M$ . Notamos que  $\delta(q_0, 1) = q_1$  y  $\delta(q_1, 1) = q_0$ . Así pues,

$$\delta(q_0, 11) = \delta(\delta(q_0, 1), 1) = \delta(q_1, 1) = q_0.$$

Podríamos remarcar que, por tanto, 11 está en  $L(M)$ , pero estamos interesados en 110101. Seguimos adelante y notamos que  $\delta(q_0, 0) = q_2$ . Entonces

$$\delta(q_0, 110) = \delta(\delta(q_0, 11), 0) = \delta(q_0, 0) = q_2.$$

Continuando en este tenor, encontramos que

$$\delta(q_0, 1101) = q_3, \quad \delta(q_0, 11010) = q_1$$

y, finalmente

$$\delta(q_0, 110101) = q_0.$$

La secuencia completa es

$$q_0^1 q_1^1 q_0^0 q_2^1 q_3^0 q_1^1 q_0.$$

Así pues, 110101 está en  $L(M)$ . Como lo mencionamos,  $L(M)$  es el conjunto de cadenas con un número par de ceros y un número par de unos.

## 2.3 AUTOMATAS FINITOS NO DETERMINISTICOS

Introducimos ahora el concepto de autómata finito no determinístico. Se llegará a la conclusión de que cualquier conjunto aceptado por un autómata finito no determinístico puede, también, ser aceptado por un autómata finito determinístico. Sin embargo, el autómata finito no determinístico es un concepto útil cuando se trata de demostrar teoremas. También el concepto de no determinismo juega un papel central en las teorías de lenguajes y computación, y es útil para comprender, en un principio y de manera completa, este concepto en un contexto muy simple. Más adelante, encontraremos autómatas cuyas versiones determinísticas y no determinísticas se sabe que no son equivalentes, y otros para los cuales la equivalencia es una profunda e importante interrogante.

Considérese una modificación al modelo del autómata finito para permitirle ninguna, una o más transiciones de un estado sobre el mismo símbolo de entrada. Este

nuevo modelo se conoce como *autómata finito no determinístico (NFA)*. En la Fig. 2.5 se muestra un diagrama de transiciones de un autómata finito no determinístico. Observe que existen dos aristas con la etiqueta 0 que salen del estado  $q_0$ , uno que regresa a  $q_0$  y otro que va al estado  $q_3$ .

Una secuencia de entrada  $a_1a_2\dots, a_n$  es aceptada por un autómata finito no determinístico si existe una secuencia de transiciones, correspondiente a la secuencia de entrada, que conduzca del estado inicial a algún estado final. Por ejemplo, 01001 es aceptado por el NFA de la Fig. 2.5 debido a que existe una secuencia de transiciones que pasa por los estados  $q_0, q_0, q_0, q_3, q_4, q_4$ , etiquetados con 0, 1, 0, 0, 1. Este NFA en particular acepta todas las cadenas que tengan dos ceros consecutivos o dos unos. Nótese que el FA de la sección anterior (FA *determinístico* o DFA) es un caso especial del NFA en el que, para cada estado, existe una transición única sobre cada símbolo. Por tanto, en un DFA, para una cadena de entrada  $w$  y un estado  $q$  dados, habrá exactamente una trayectoria con la etiqueta  $w$  que comience en  $q$ . Para determinar si una cadena es aceptada por un DFA es suficiente verificar esta trayectoria. Para un NFA podría haber muchas trayectorias con la etiqueta  $w$ , y se deben verificar todas para determinar si una o más acaban en un estado final.

En función del diagrama que se presenta en la Fig. 2.3, con un control finito que lee una cinta de entrada, podemos considerar que el NFA también lee una cinta de entrada. Sin embargo, en cualquier instante, el control finito puede encontrarse en un cierto número de estados. Cuando se puede hacer una selección del siguiente estado, como el caso del estado  $q_0$  sobre la entrada 0 en la Fig. 2.5, podemos imaginar que se

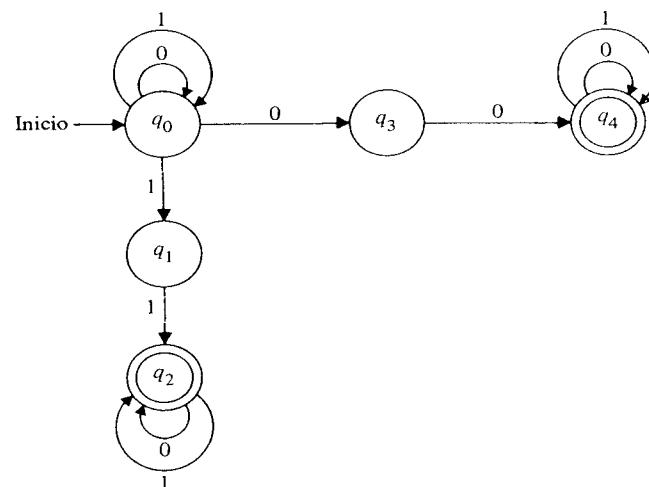


Fig. 2.5 Diagrama de transiciones para un autómata finito no determinístico.

hacen duplicados del autómata. Para cada estado siguiente posible existe una copia del autómata cuyo control finito se encuentra en ese estado. Esta proliferación se muestra en la Fig. 2.6 para el NFA de la Fig. 2.5, con entrada 01001.

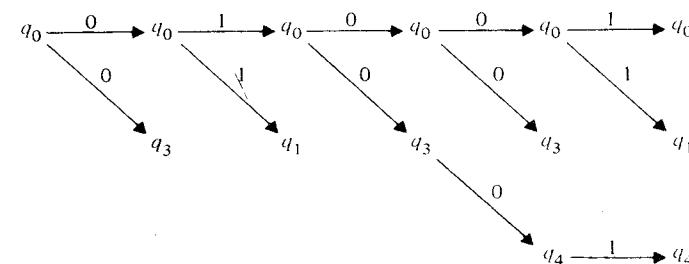


Fig. 2.6 Proliferación de estados de un NFA.

De manera formal denotamos a un *autómata finito no determinístico* mediante  $(Q, \Sigma, \delta, q_0, F)$ , en donde  $Q$ ,  $\Sigma$ ,  $q_0$  y  $F$  (estados, entradas, estado inicial y estados finales) poseen el mismo significado que para un DFA, pero en este caso  $\delta$  es una transformación de  $Q \times \Sigma$  a  $2^Q$ . (Recuérdese que  $2^Q$  es el conjunto de potencias de  $Q$ , el conjunto de todos los subconjuntos de  $Q$ ). La intención es que  $\delta(q, a)$  sea el conjunto de todos los estados  $p$  tales que exista una transición con etiqueta  $a$  de  $q$  a  $p$ .

**Ejemplo 2.3** En la Fig. 2.7 se da la función  $\delta$  para el NFA de la Fig. 2.5.

Estados	Entradas	
	0	1
$q_0$	$\{q_0, q_3\}$	$\{q_0, q_1\}$
$q_1$	$\emptyset$	$\{q_2\}$
$q_2$	$\{q_2\}$	$\{q_2\}$
$q_3$	$\{q_4\}$	$\emptyset$
$q_4$	$\{q_4\}$	$\{q_4\}$

Fig. 2.7 La transformación  $\delta$  para el NFA de la Fig. 2.5.

La función  $\delta$  puede extenderse a una función  $\hat{\delta}$  que transforma  $Q \times \Sigma^*$  a  $2^Q$  y que refleje sucesiones de entradas de la manera siguiente:

1.  $\hat{\delta}(q, \epsilon) = \{q\}$ ,
2.  $\hat{\delta}(q, wa) = \{p \mid \text{para algún estado } r \text{ de } \hat{\delta}(q, w), p \text{ esté en } \delta(r, a)\}$ .

La condición (1) no permite un cambio en un estado sin que esté presente una entrada. La condición (2) indica que al iniciar en el estado  $q$  y leer la cadena  $w$  seguida por el símbolo de entrada  $a$ , podemos llegar al estado  $p$  si y sólo si uno de los estados posibles en los que podíamos estar después de leer  $w$  es  $r$ , y de  $r$  podemos ir a  $p$  si leemos  $a$ .

Nótese que  $\hat{\delta}(q, a) = \delta(q, a)$  en donde  $a$  es un símbolo de entrada. Por tanto, podemos utilizar de nuevo  $\delta$  en lugar de  $\hat{\delta}$ . Es útil también extender  $\delta$  a argumentos que estén en  $2^Q \times \Sigma^*$  mediante

$$3. \delta(P, w) = \bigcup_{q \in P} \delta(q, w)$$

para cada conjunto de estados  $P \subseteq Q$ .  $L(M)$ , en el que  $M$  está en el NFA  $(Q, \Sigma, \delta, q_0, F)$ , es  $\{w \mid \delta(q_0, w) \text{ contiene un estado en } F\}$ .

**Ejemplo 2.4** Considérese de nuevo el NFA de la Fig. 2.5, cuya función de transición  $\delta$  se presentó en la Fig. 2.7. Sea la entrada 01001.

$$\delta(q_0, 0) = \{q_0, q_3\}.$$

$$\delta(q_0, 01) = \delta(\delta(q_0, 0), 1) = \delta(\{q_0, q_3\}, 1) = \delta(q_0, 1) \cup \delta(q_3, 1) = \{q_0, q_1\}.$$

De manera similar, calculamos

$$y \quad \delta(q_0, 010) = \{q_0, q_3\}, \quad \delta(q_0, 0100) = \{q_0, q_3, q_4\}$$

$$\delta(q_0, 01001) = \{q_0, q_1, q_4\}.$$

### Equivalencias de los DFAs y los NFAs

Puesto que cada DFA es un NFA, queda claro que la clase de lenguaje aceptado por los NFAs incluye a los conjuntos regulares (los lenguajes aceptados por los DFAs). Sin embargo, resulta que éstos son los únicos conjuntos aceptados por los NFAs. La demostración consiste en mostrar que los DFAs pueden simular a los NFAs; esto es, para cada NFA podemos construir un DFA *equivalente* (uno que acepta el mismo lenguaje). La forma en que un DFA simula a un NFA consiste en permitir que los estados del DFA correspondan a conjuntos de estados del NFA. El DFA que se ha construido mantiene, en su control finito, el rastro de todos los estados en los que el NFA puede estar después de leer la misma entrada que el DFA ha leído. La construcción formal se encuadra en nuestro primer teorema.

**Teorema 2.1** Sea  $L$  un conjunto aceptado por un autómata finito no determinístico. Entonces existe un autómata finito determinístico que acepta a  $L$ .

**Demostración** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un NFA que acepta a  $L$ . Defínase un DFA,  $M' = (Q', \Sigma, \delta', q'_0, F')$ , de la manera siguiente. Los estados de  $M'$  son todos los subconjuntos del conjunto de estados de  $M$ . Es decir,  $Q' = 2^Q$ .  $M'$  conservará, en su estado, el rastro de todos los estados en los que puede estar  $M$  en cualquier tiempo dado.  $F'$  es el conjunto de todos los estados de  $Q'$  que contienen un estado final de  $M$ . Un elemento de  $Q'$  será denotado por  $[q_1, q_2, \dots, q_i]$ , en donde  $q_1, q_2, \dots, q_i$  se encuentran en  $Q$ . Observe que  $[q_1, q_2, \dots, q_i]$  es un solo estado del DFA que corresponde a un conjunto de estados del NFA. Nótese también que  $q'_0 = [q_0]$ .

Definimos

$$\delta'([q_1, q_2, \dots, q_i], a) = [p_1, p_2, \dots, p_j]$$

si y sólo si

$$\delta(\{q_1, q_2, \dots, q_i\}, a) = \{p_1, p_2, \dots, p_j\}.$$

Esto es,  $\delta'$  aplicada a un elemento  $[q_1, q_2, \dots, q_i]$  de  $Q'$  se calcula aplicando  $\delta$  a cada estado de  $Q$  que esté representado por  $[q_1, q_2, \dots, q_i]$ . Al aplicar  $\delta$  a cada  $q_1, q_2, \dots, q_i$  y tomar la unión, obtenemos algún conjunto nuevo de estados,  $p_1, p_2, \dots, p_j$ . Este nuevo conjunto de estados tiene un representante,  $[p_1, p_2, \dots, p_j]$  en  $Q'$ , y este elemento es el valor de  $\delta'([q_1, q_2, \dots, q_i], a)$ .

Es fácil mostrar, por inducción sobre la longitud de la cadena de entrada  $x$  que

$$\delta'(q_0, x) = [q_1, q_2, \dots, q_i]$$

si y sólo si

$$\delta(q_0, x) = \{q_1, q_2, \dots, q_i\}.$$

**Base** El resultado es trivial para  $|x| = 0$ , ya que  $q'_0 = [q_0]$  y  $x$  debe ser  $\epsilon$ .

**Inducción** Supóngase que la hipótesis es verdadera para entradas de longitud  $m$  o menores. Sea  $xa$  una cadena de longitud  $m + 1$  con  $a$  en  $\Sigma$ . Entonces

$$\delta'(q'_0, xa) = \delta'(\delta'(q'_0, x), a).$$

Tomando en cuenta la hipótesis inductiva,

$$\delta'(q'_0, x) = [p_1, p_2, \dots, p_j]$$

si y sólo si

$$\delta(q_0, x) = \{p_1, p_2, \dots, p_j\}.$$

Pero, por la definición de  $\delta'$

$$\delta'([p_1, p_2, \dots, p_j], a) = [r_1, r_2, \dots, r_k]$$

si y sólo si

$$\delta(\{p_1, p_2, \dots, p_j\}, a) = \{r_1, r_2, \dots, r_k\}.$$

Por tanto,

$$\delta'(q'_0, xa) = [r_1, r_2, \dots, r_k]$$

si y sólo si

$$\delta(q_0, xa) = \{r_1, r_2, \dots, r_k\},$$

que establece la hipótesis inductiva.

Para completar la prueba, sólo tenemos que añadir que  $\delta'(q'_0, x)$  está en  $F'$  exactamente cuando  $\delta(q_0, x)$  contiene un estado de  $Q$  que esté en  $F$ . Así pues,  $L(M) = L(M')$ .  $\square$

Como los autómatas finitos determinísticos y no determinísticos aceptan los mismos conjuntos, no haremos distinción alguna entre ellos, a menos que sea necesario, sino que simplemente nos referiremos a ambos como autómatas finitos.

**Ejemplo 2.5** Sea  $M = (\{q_0, q_1\}, \{0, 1\}, \delta, q_0, \{q_1\})$  un NFA en el que  $\delta(q_0, 0) = \{q_0, q_1\}$ ,  $\delta(q_0, 1) = \{q_1\}$ ,  $\delta(q_1, 0) = \emptyset$ ,  $\delta(q_1, 1) = \{q_0, q_1\}$ .

Podemos construir un DFA,  $M' = (\{q_0\}, \{0, 1\}, \delta', [q_0], F)$ , que acepte a  $L(M)$ , de la manera siguiente.  $Q$  consiste en todos los subconjuntos de  $\{q_0, q_1\}$ . Denotamos a los elementos de  $Q$  por  $[q_0]$ ,  $[q_1]$ ,  $[q_0, q_1]$  y  $\emptyset$ . Como  $\delta(q_0, 0) = \{q_0, q_1\}$ , tenemos

$$\delta'(\{q_0\}, 0) = [q_0, q_1].$$

De manera parecida,

$$\delta'(\{q_0\}, 1) = [q_1], \quad \delta'(\{q_1\}, 0) = \emptyset, \quad \text{y} \quad \delta'(\{q_1\}, 1) = [q_0, q_1].$$

Naturalmente,  $\delta'(\emptyset, 0) = \delta'(\emptyset, 1) = \emptyset$ . Por último,

$$\delta'(\{q_0, q_1\}, 0) = [q_0, q_1],$$

porque

$$\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\} \cup \emptyset = \{q_0, q_1\},$$

y

$$\delta'(\{q_0, q_1\}, 1) = [q_0, q_1].$$

porque

$$\delta(\{q_0, q_1\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) = \{q_1\} \cup \{q_0, q_1\} = \{q_0, q_1\}.$$

El conjunto  $F$  de estados finales es  $\{[q_1], [q_0, q_1]\}$ .

En la práctica, resulta con frecuencia que muchos estados del NFA no son accesibles a partir del estado inicial  $[q_0]$ . Es, por tanto, una buena idea empezar con el estado  $[q_0]$  y añadir estados al DFA sólo si éstos son el resultado de una transición de un estado añadido previamente.

## 2.4 AUTOMATAS FINITOS CON MOVIMIENTOS $\epsilon$

Podemos extender nuestro modelo de autómata finito no determinístico para incluir transiciones sobre la entrada vacía  $\epsilon$ . El diagrama de transiciones de dicho NFA que acepta al lenguaje consistente en cualquier número (incluyendo el cero) de 0s seguidos por cualquier número de 1s seguidos, a su vez, por cualquier número 2s, se muestra en la Fig. 2.8. Como siempre, decimos que un NFA acepta una cadena  $w$  si existe alguna trayectoria con la etiqueta  $w$  que va del estado inicial a un estado final. Por supuesto, deben incluirse las aristas con etiqueta  $\epsilon$  en la trayectoria, aunque las  $\epsilon$ s no aparezcan explícitamente en  $w$ . Por ejemplo, la palabra 002 es aceptada por el NFA de la Fig. 2.8 mediante la trayectoria  $q_0, q_0, q_0, q_1, q_2, q_2$  con arcos cuyas etiquetas son  $0, 0, \epsilon, \epsilon, 2$ .

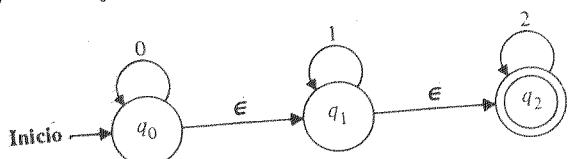


Fig. 2.8 Autómata finito con movimientos  $\epsilon$ .

De manera formal definimos un *autómata finito no determinístico con movimientos  $\epsilon$*  como la quíntupla  $(Q, \Sigma, \delta, q_0, F)$  con todos sus componentes igual que antes, con excepción de  $\delta$ , la función de transición, que ahora transforma  $Q \times (\Sigma \cup \{\epsilon\})$  a  $2^Q$ . La intención es que  $\delta(q, a)$  consista en todos los estados  $p$  tales que exista una transición con etiqueta  $a$  que vaya de  $q$  a  $p$ , siendo  $a$  el símbolo  $\epsilon$  o un símbolo de  $\Sigma$ .

**Ejemplo 2.6** En la Fig. 2.9 se muestra la función de transición para el NFA de la Fig. 2.8.

Ahora extenderemos la función de transición  $\delta$  a una función  $\hat{\delta}$  que transforme  $Q \times \Sigma^*$  a  $2^Q$ . Esperamos que  $\hat{\delta}(q, w)$  sean todos los estados  $p$  tales que uno pueda ir de

Estados	Entradas			
	0	1	2	$\epsilon$
$q_0$	{ $q_0$ }	$\emptyset$	$\emptyset$	{ $q_1$ }
$q_1$	$\emptyset$	{ $q_1$ }	$\emptyset$	{ $q_2$ }
$q_2$	$\emptyset$	$\emptyset$	{ $q_2$ }	$\emptyset$

Fig. 2.9  $\delta(q, a)$  para el NFA de la Fig. 2.8.

que  $q$  a  $p$  siguiendo una trayectoria con etiqueta  $w$ , incluyendo, tal vez, las aristas con etiqueta  $\epsilon$ . Cuando se construya  $\hat{\delta}$  será importante calcular el conjunto de estados que se pueden alcanzar desde un estado dado  $q$  utilizando solamente transiciones  $\epsilon$ . Esta cuestión es equivalente a preguntarse cuáles vértices pueden alcanzarse desde un vértice (*o fuente*) dado en una gráfica dirigida. El vértice fuente es el vértice del estado  $q$  en el diagrama de transición, y la correspondiente gráfica dirigida consiste en todos los arcos con etiqueta  $\epsilon$ . Utilizamos el término CERRADURA  $\epsilon(q)$  para denotar al conjunto de todos los vértices  $p$  tales que existe una trayectoria de  $q$  a  $p$  con etiqueta  $\epsilon$ .

**Ejemplo 2.7** En la Fig. 2.8 se tiene que CERRADURA  $\epsilon(q_0) = \{q_0, q_1, q_2\}$ . Es decir, la trayectoria que consiste sólo en  $q_0$  (no existen arcos en la trayectoria) es una trayectoria que va de  $q_0$  a  $q_0$  con todos los arcos con etiqueta  $\epsilon$ .† La trayectoria  $q_0, q_1, q_2$  muestra que  $q_1$  está en la CERRADURA  $\epsilon(q_0)$  y la trayectoria  $q_0, q_1, q_2$  muestra que  $q_2$  está en la CERRADURA  $\epsilon(q_0)$ .

Naturalmente que podemos hacer la CERRADURA  $\epsilon(P)$ , en la que  $P$  es un conjunto de estados, sea  $\bigcup_{q \in P} \text{CERRADURA } \epsilon(q)$ . Definimos ahora  $\hat{\delta}$  de la manera siguiente

$$1. \hat{\delta}(q, \epsilon) = \text{CERRADURA } \epsilon(q).$$

† Recuerde que una trayectoria de longitud cero no tiene arcos, por tanto todos sus arcos tienen etiqueta  $\epsilon$ .

2. Para  $w$  en  $\Sigma^*$  y  $a$  en  $\Sigma$ ,  $\hat{\delta}(q, wa) = \text{CERRADURA} \in (P)$ , en donde  $P = \{p\}$  para algún  $r$  en  $\hat{\delta}(q, w)$ ,  $p$  está en  $\delta(r, a)$ .

Es conveniente extender  $\delta$  y  $\hat{\delta}$  a conjuntos de estados mediante

3.  $\delta(R, a) = \bigcup_{q \in R} \delta(q, a)$  y  
4.  $\hat{\delta}(R, w) = \bigcup_{q \in R} \hat{\delta}(q, w)$

para conjuntos de estados  $R$ . Nótese que en este caso,  $\hat{\delta}(q, a)$  no necesariamente es igual a  $\delta(q, a)$ , ya que  $\hat{\delta}(q, a)$  incluye a todos los estados que se pueden alcanzar desde  $q$  mediante trayectorias con etiqueta  $a$  (incluyendo trayectorias con arcos con etiqueta  $\epsilon$ ), mientras que  $\delta(q, a)$  sólo incluye aquellos estados que se pueden alcanzar desde  $q$  mediante arcos cuya etiqueta sea  $a$ . De manera similar,  $\hat{\delta}(q, \epsilon)$  no es necesariamente igual a  $\delta(q, \epsilon)$ . Por tanto, es necesario hacer una distinción entre  $\delta$  y  $\hat{\delta}$  cuando hablamos de un NFA con transiciones  $\epsilon$ .

Definimos  $L(M)$ , el lenguaje aceptado por  $M = (Q, \Sigma, \delta, q_0, F)$ , como  $\{w \mid \hat{\delta}(q_0, w) \text{ contiene un estado en } F\}$ .

**Ejemplo 2.8** Considérese de nuevo el NFA de la Fig. 2.8,

$$\hat{\delta}(q_0, \epsilon) = \text{CERRADURA} \in (q_0) = \{q_0, q_1, q_2\}.$$

Por tanto

$$\begin{aligned}\hat{\delta}(q_0, 0) &= \text{CERRADURA} \in (\hat{\delta}(q_0, \epsilon), 0)) \\ &= \text{CERRADURA} \in (\delta(\{q_0, q_1, q_2\}, 0)) \\ &= \text{CERRADURA} \in (\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0)) \\ &= \text{CERRADURA} \in (\{q_0\} \cup \emptyset \cup \emptyset) \\ &= \text{CERRADURA} \in (\{q_0\}) = \{q_0, q_1, q_2\}.\end{aligned}$$

Entonces

$$\begin{aligned}\hat{\delta}(q_0, 01) &= \text{CERRADUR} \in (\hat{\delta}(q_0, 0), 1)) \\ &= \text{CERRADUR} \in (\delta(\{q_0, q_1, q_2\}, 1)) \\ &= \text{CERRADUR} \in (\{q_1\}) = \{q_1, q_2\}.\end{aligned}$$

### Equivalencia de los NFAs con y sin movimientos $\epsilon$

Como en el caso del no determinismo, la capacidad de hacer transiciones sobre  $\epsilon$  no permite al NFA aceptar conjuntos no regulares. Mostraremos esto mediante la simulación de un NFA con transiciones  $\epsilon$  utilizando un NFA sin tales transiciones.

**Teorema 2.2** Si  $L$  es aceptado por un NFA con transiciones  $\epsilon$ , entonces  $L$  es aceptado por un NFA sin transiciones  $\epsilon$ .

**Demostración** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un NFA con transiciones  $\epsilon$ . Construyase  $M' = (Q, \Sigma, \delta, q_0, F')$  en el que

$$F' = \begin{cases} F \cup \{q_0\} & \text{si CERRADURA} \in (q_0) \text{ contiene un estado de } F, \\ F & \text{en cualquier otro caso} \end{cases}$$

y  $\delta'(q, a)$  es  $\hat{\delta}(q, a)$  para  $q$  en  $Q$  y  $a$  en  $\Sigma$ . Nótese que  $M'$  no tiene transiciones  $\epsilon$ . Así pues podemos utilizar  $\hat{\delta}'$  como  $\delta'$ , pero debemos seguir haciendo distinción entre  $\delta$  y  $\hat{\delta}$ .

Deseamos mostrar por inducción sobre  $|x|$  que  $\delta'(q_0, x) = \hat{\delta}(q_0, x)$ . Sin embargo, esta proposición puede no ser verdadera para  $x = \epsilon$ , ya que  $\delta'(q_0, \epsilon) = \{q_0\}$ , mientras que  $\hat{\delta}(q_0, \epsilon) = \text{CERRADURA} \in (q_0)$ . Por tanto empezamos nuestra inducción en 1.

*Base*  $|x| = 1$ . Entonces  $x$  es un símbolo  $a$ , y  $\delta'(q_0, a) = \hat{\delta}(q_0, a)$ , por definición de  $\delta'$ .

*Inducción*  $|x| > 1$ . Sea  $x = wa$  para un símbolo  $a$  en  $\Sigma$ . Entonces

$$\delta'(q_0, wa) = \delta'(\delta'(q_0, w), a).$$

Por la hipótesis inductiva,  $\delta'(q_0, w) = \hat{\delta}(q_0, w)$ . Sea  $\hat{\delta}(q_0, w) = P$ . Debemos mostrar que  $\delta'(P, a) = \delta(q_0, wa)$ . Pero

$$\delta'(P, a) = \bigcup_{q \in P} \delta'(q, a) = \bigcup_{q \in P} \hat{\delta}(q, a).$$

Entonces, como  $P = \hat{\delta}(q_0, w)$  tenemos

$$\bigcup_{q \in P} \hat{\delta}(q, a) = \hat{\delta}(q_0, wa)$$

por la regla (2) de la definición de  $\hat{\delta}$ . Así pues,

$$\delta'(q_0, wa) = \hat{\delta}(q_0, wa).$$

Para completar la prueba demostraremos que  $\delta'(q_0, x)$  contiene un estado  $F'$  si y sólo si  $\delta(q_0, x)$  contiene un estado de  $F$ . Si  $x = \epsilon$ , la verdad de la proposición se ve de manera inmediata de la definición de  $F'$ . Esto es,  $\delta'(q_0, \epsilon) = \{q_0\}$  y  $q_0$  está situado en  $F'$  cuando  $\delta(q_0, \epsilon)$ , que es la CERRADURA  $\in (q_0)$ , contenga un estado de  $F$  (posiblemente  $q_0$ ). Si  $x \neq \epsilon$ , entonces  $x = wa$  para algún símbolo  $a$ . Si  $\delta(q_0, x)$  contiene un estado de  $F$ , entonces, seguramente,  $\delta'(q_0, x)$  contiene al mismo estado en  $F'$ . En el otro sentido, si  $\delta'(q_0, x)$  contiene un estado de  $F'$  diferente a  $q_0$ , entonces  $\delta(q_0, x)$  contiene un estado en  $F$ . Si  $\delta'(q_0, x)$  contiene a  $q_0$  y  $q_0$  no está en  $F$ , entonces, como  $\delta(q_0, x) = \text{CERRADURA} \in (\delta(\delta(q_0, w), a))$ , el estado que esté en la CERRADURA  $\in (q_0)$  y en  $F$  debe estar  $\delta(q_0, x)$ .

**Ejemplo 2.9** Apliquemos la construcción del Teorema 2.2 al NFA de la Fig. 2.8. En la Fig. 2.10 hacemos un resumen de  $\hat{\delta}(q, a)$ . Podemos también considerar la Fig. 2.10 como la función de transición  $\delta'$  del NFA sin transiciones  $\epsilon$  construido mediante el Teorema 2.2. El conjunto de estados finales  $F'$  incluye a  $q_2$  porque éste se encuentra en  $F$ , y también incluye a  $q_0$  porque la CERRADURA  $\in (q_0)$  y  $F$  tienen un estado  $q_2$  en común. En la Fig. 2.11 se presenta el diagrama de transiciones para  $M'$ .

Estados	Entradas		
	0	1	2
$q_0$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_2\}$
$q_1$	$\emptyset$	$\{q_1, q_2\}$	$\{q_2\}$
$q_2$	$\emptyset$	$\emptyset$	$\{q_2\}$

Fig. 2.10  $\hat{\delta}(q, a)$  para la Fig. 2.8.

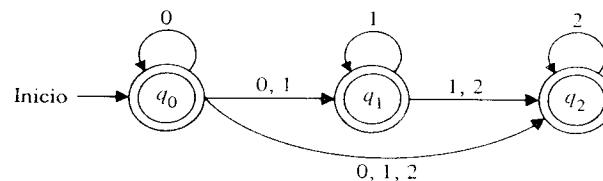


Fig. 2.11 NFA sin transiciones  $\epsilon$ .

## 2.5 EXPRESIONES REGULARES

Los lenguajes aceptados por un autómata finito se describen con facilidad mediante expresiones simples llamadas *expresiones regulares*. En esta sección introducimos las operaciones de concatenación y cerradura sobre conjunto de cadenas, definimos las expresiones regulares y probamos que el tipo de lenguajes aceptados por los autómatas finitos es precisamente el tipo de lenguajes que se pueden describir con expresiones regulares.

Sea  $\Sigma$  un conjunto finito de símbolos y  $L, L_1$  y  $L_2$  conjuntos de cadenas de  $\Sigma^*$ . La *concatenación* de  $L_1$  y  $L_2$ , denotada como  $L_1 L_2$ , es el conjunto  $\{ xy \mid x \text{ está en } L_1 \text{ y } y \text{ en } L_2\}$ . Es decir, las cadenas de  $L_1 L_2$  están formadas mediante la selección de una cadena  $L_1$  seguida de una cadena  $L_2$ , en todas las combinaciones posibles. Defínase  $L^0 = \{ \epsilon \}$  y  $L^i = LL^{i-1}$  para  $i \geq 1$ . La *cerradura de Kleene* (o solamente *cerradura*) de  $L$ , denotada como  $L^*$ , es el conjunto

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

y la *cerradura positiva* de  $L$ , denotada por  $L^+$ , es el conjunto

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Es decir,  $L^*$  denota palabras construidas mediante la concatenación de cualquier número de palabras de  $L$ .  $L^+$  es lo mismo que  $L^*$ , sólo que ahora se excluye el caso de cero palabras, cuya “concatenación” se define como  $\epsilon$ . Nótese que  $L^+$  contiene a  $\epsilon$  si y sólo si  $L$  lo hace.

**Ejemplo 2.10** Sea  $L_1 = \{10, 1\}$  y  $L_2 = \{011, 11\}$ . Entonces  $L_1 L_2 = \{10011, 1011, 111\}$ . También,

$$\{10, 11\}^* = \{\epsilon, 10, 11, 1010, 1011, 1110, 1111, \dots\}.$$

Si  $\Sigma$  es una alfabeto, entonces  $\Sigma^*$  denota a todas las cadenas de símbolos de  $\Sigma$ , como se estableció previamente. Nótese que no hacemos distinción entre  $\Sigma$  como un alfabeto y  $\Sigma$  como un lenguaje de cadenas de longitud 1.

Sea  $\Sigma$  un alfabeto. La expresión regular sobre  $\Sigma$  y los conjuntos que denotan se definen de manera recursiva como sigue.

1.  $\emptyset$  es una expresión regular y denota al conjunto vacío.
2.  $\epsilon$  es una expresión regular y denota al conjunto  $\{\epsilon\}$ .
3. Para cada  $a$  en  $\Sigma$ ,  $a^\dagger$  es una expresión regular y denota al conjunto  $\{a\}$ .
4. Si  $r$  y  $s$  son expresiones regulares que denotan a los lenguajes  $R$  y  $S$ , respectivamente, entonces  $(r + s)$ ,  $(rs)$  y  $(r^*)$  son expresiones regulares que denotan a los conjuntos  $R \cup S$ ,  $RS$  y  $R^*$ , respectivamente.

Al escribir expresiones regulares podemos omitir muchos paréntesis si suponemos que  $*$  tiene prioridad sobre la concatenación, sobre  $+$ , y que la concatenación tiene prioridad sobre  $+$ . Por ejemplo,  $((0(1^*)) + 0)$  puede escribirse como  $01^* + 0$ . También podemos abreviar la expresión  $rr^*$  escribiendo  $r^*$ . Cuando sea necesario distinguir entre una expresión regular  $r$  y el lenguaje denotado por  $r$ , utilizaremos  $L(r)$  para este último. Cuando no haya lugar a confusiones utilizaremos  $r$  para las expresiones regulares y para el lenguaje denotado por la expresión regular.

**Ejemplo 2.11**  $00$  es la expresión regular que representa a  $\{00\}$ . La expresión  $(0 + 1)^*$  denota a todas las cadenas de ceros y unos. Por tanto,  $(0 + 1)^*00(0 + 1)^*$  representa a todas las cadenas de ceros y unos con al menos dos ceros consecutivos. La expresión regular  $(1 + 10)^*$  denota a todas las cadenas de 0s y 1s que comienzan con 1 y no tienen dos ceros consecutivos. Como prueba de lo anterior es fácil hacer una demostración por inducción sobre  $i$  de que  $(1 + 10)^i$  no tiene dos ceros consecutivos.<sup>††</sup> Aún más, dada alguna cadena que comience con 1 y que no tenga ceros consecutivos, uno puede partir dicha cadena en unos, seguidos de un 0, si existe. Por ejemplo, 1101011 puede partirse en 1–10–10–1–1. Esta partición muestra que cualquiera de estas cadenas está en  $(1 + 10)^i$ ,

<sup>†</sup> Para recordarle al lector que un símbolo es parte de una expresión regular, lo escribiremos en negritas. Sin embargo, consideraremos  $a$  y  $a^\dagger$  como el mismo símbolo.

<sup>††</sup> Si  $r$  es una expresión regular,  $r^i$  significa  $rr \dots r$  ( $i$  veces).

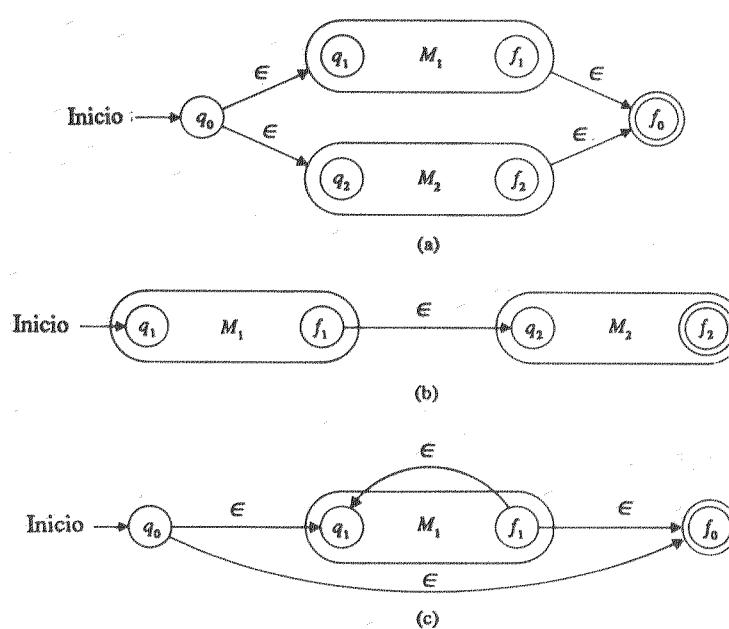


Fig. 2.14 Construcciones utilizadas en la inducción del Teorema 2.3. (a) Para la unión. (b) Para la concatenación. (c) Para la cerradura.

CASO 2  $r = r_1 r_2$ . Sean  $M_1$  y  $M_2$  como en el Caso 1 y constrúyase

$$M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, \{q_1\}, \{f_2\}),$$

en donde  $\delta$  está dada por

- i)  $\delta(q, a) = \delta_1(q, a)$  para  $q$  en  $Q_1 - \{f_1\}$  y  $a$  en  $\Sigma_1 \cup \{\epsilon\}$ ,
- ii)  $\delta(f_1, \epsilon) = \{q_2\}$ .
- iii)  $\delta(q, a) = \delta_2(q, a)$  para  $q$  en  $Q_2$  y  $a$  en  $\Sigma_2 \cup \{\epsilon\}$ .

La construcción de  $M$  está dada en la Fig. 2.14 (b). Cada trayectoria en  $M$  de  $q$ , a  $q_2$  es una trayectoria etiquetada con alguna cadena  $x$  y va de  $q_1$  a  $f_1$ , seguida por el borde de  $f_1$  a  $q_2$  etiquetado con  $\epsilon$ , seguido, a su vez, por una trayectoria etiquetada con alguna cadena  $y$  que va de  $q_2$  a  $f_2$ . Por tanto,  $L(M) = \{xy \mid x \text{ está en } L(M_1) \text{ y } y \text{ está en } L(M_2)\}$  y  $L(M) = L(M_1)L(M_2)$  como se quería demostrar.

CASO 3  $r = r_1^*$ . Sea  $M_1 = (Q_1, \Sigma_1, \delta_1, q_1, \{f_1\})$  y  $L(M_1) = r_1$ . Constrúyase

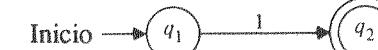
$$M = (Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\}),$$

en el que  $\delta$  está dada por

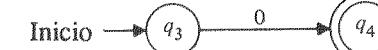
- i)  $\delta(q_0, \epsilon) = \delta(f_1, \epsilon) = \{q_1, f_1\}$ ,
- ii)  $\delta(q, a) = \delta_1(q, a)$  para  $q$  en  $Q_1 - \{f_1\}$  y  $a$  en  $\Sigma_1 \cup \{\epsilon\}$ .

La construcción de  $M$  se presenta en la Fig. 2.14 (c). Cualquier trayectoria de  $q_0$  a  $f_0$  consiste en una trayectoria de  $q_0$  a  $f_0$  sobre  $\epsilon$  o una trayectoria de  $q_0$  a  $q_1$  sobre  $\epsilon$ , seguida de cierto número (posiblemente cero) de trayectorias de  $q_1$  a  $f_1$ , después se retrocede a  $q_1$  sobre  $\epsilon$ , cada trayectoria etiquetada con una cadena de  $L(M_1)$ , seguida, a su vez, por una trayectoria de  $q_1$  a  $f_1$  sobre una cadena de  $L(M_1)$ , después a  $f_0$  sobre  $\epsilon$ . Así pues, existe una trayectoria en  $M$  de  $q_0$  a  $f_0$  con etiqueta  $x$  si y sólo si podemos escribir  $x = x_1 x_2 \cdots x_j$  para alguna  $j \geq 0$  (el caso  $j = 0$  significa  $x = \epsilon$ ) tal que cada  $x_i$  esté en  $L(M_1)$ . De aquí que  $L(M) = L(M_1)^*$  como se desea.  $\square$

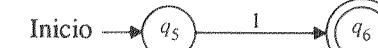
**Ejemplo 2.12** Construyamos un NFA para la expresión regular  $01^* + 1$ . Tomando en cuenta las reglas que acabamos de discutir, esta expresión en realidad es  $(0(1^*)) + 1$ , así que es de la forma  $r_1 + r_2$ , en donde  $r_1 = 01^*$  y  $r_2 = 1$ . El autómata para  $r_2$  es fácil; éste es



Podemos expresar  $r_1$  como  $r_3 r_4$ , en donde  $r_3 = 0$  y  $r_4 = 1^*$ . El autómata para  $r_3$  también es fácil:



A su vez,  $r_4$  es  $r_5^*$ , en donde  $r_5$  es 1. Un autómata para  $r_5$  es



Nótese que la necesidad de mantener los estados de autómatas diferentes disjuntos nos prohíbe utilizar el mismo NFA para  $r_2$  y  $r_5$ , aunque sean la misma expresión.

Para construir un NFA para  $r_4 = r_5^*$  utilice la construcción de la Fig. 2.14 (c). Construya los estados  $q_7$  y  $q_8$  que jueguen el mismo papel que  $q_0$  y  $f_0$ , respectivamente. El NFA que resulta para  $r_4$  se muestra en la Fig. 2.15 (a). Para  $r_1 = r_3 r_4$  utilice la construcción de la Fig. 2.14 (b). En la Fig. 2.15 (b) se muestra el resultado. Por último utilice la construcción de la Fig. 2.14 (a) para encontrar el NFA para  $r = r_1 + r_2$ . En esta construcción se crearon dos estados  $q_9$  y  $q_{10}$  que jugaron el papel de  $q_0$  y  $f_0$ , el resultado se muestra en la Fig. 2.15 (c).

La prueba del Teorema 2.3 es, en esencia, un algoritmo para convertir una expresión regular en un autómata finito. Sin embargo, el algoritmo supone implícitamente que la expresión regular se encuentra completamente entre paréntesis. Para las expresiones regulares que no tengan paréntesis redundantes, debemos determinar si la expresión es de la forma  $p + q$ ,  $pq$  o  $p^*$ . Lo anterior es equivalente a analizar gramaticalmente una cadena de un lenguaje libre de contexto; por tanto, un algoritmo como éste se verá hasta el Cap. 5, donde puede llevarse a cabo de una manera más elegante.

Ahora debemos demostrar que cada conjunto aceptado por un autómata finito es denotado por alguna expresión regular. Este resultado completará el círculo que se presenta en la Fig. 2.12.

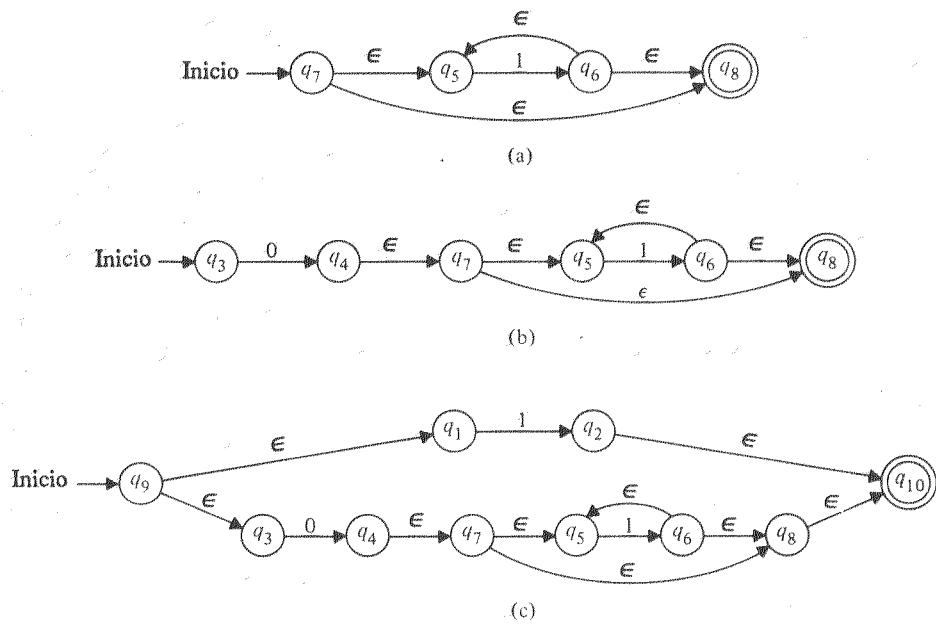


Fig.2.15 Construcción de un NFA a partir de una expresión regular. a) para  $r_i = 1^*$ . b) para  $r_i = 01^*$ . c) para  $r = 01^* + 1$ .

**Teorema 2.4** Si  $L$  es aceptado por un DFA, entonces  $L$  es denotado por una expresión regular.

**Demostración** Sea  $L$  el conjunto aceptado por el DFA

$$M = (\{q_1, \dots, q_n\}, \Sigma, \delta, q_1, F).$$

Sea  $R_{ij}^k$  el conjunto de todas las cadenas  $x$  tales que  $\delta(q_i, x) = q_j$ , y si  $\delta(q_i, y) = q_j$  para cualquier  $y$  que sea prefijo (segmento inicial) de  $x$ , diferente a  $x$  o  $\epsilon$ , entonces  $y \leq k$ . Es decir,  $R_{ij}^k$  es el conjunto de todas las cadenas que toma el autómata finito para ir del estado  $q_i$  al  $q_j$  sin pasar por ningún estado cuyo número sea mayor que  $k$ . Nótese que con la expresión «pasar por un estado» queremos decir entrar y después salir. Por tanto  $i$  o  $j$  pueden ser mayores que  $k$ : Como no existe un estado con número mayor que  $n$ ,  $R_{ij}^k$  denota a las cadenas que toman de  $q_i$  a  $q_j$ . Podemos definir  $R_{ij}^k$  de manera recursiva:

$$\begin{aligned} R_{ij}^k &= R_{ik}^{k-1}(R_{kk}^{k-1})^*R_{kj}^{k-1} \cup R_{ij}^{k-1}, \\ R_{ij}^0 &= \begin{cases} \{a \mid \delta(q_i, a) = q_j\} & \text{if } i \neq j, \\ \{a \mid \delta(q_i, a) = q_j\} \cup \{\epsilon\} & \text{if } i = j. \end{cases} \end{aligned} \quad (2.1)$$

Informalmente, la definición de  $R_{ij}^k$  dada arriba, significa que las entradas que hacen que  $M$  vaya de  $q_i$  a  $q_j$  sin pasar por un estado mayor que  $q_k$  son

1. las que se encuentran en  $R_{ij}^{k-1}$  (o sea, nunca pasan por un estado tan alto como  $q_k$ ); o
2. las compuestas por una cadena de  $R_{ij}^{k-1}$  (que lleva a  $M$  a  $q_k$  por primera vez) seguida por cero o más cadenas de  $R_{ij}^{k-1}$  (que lleva a  $M$  de  $q_k$  de regreso a  $q_j$  sin pasar por  $q_k$  o un estado con número mayor) seguida por una cadena de  $R_{ij}^{k-1}$  (que lleva a  $M$  del estado  $q_k$  al  $q_j$ ).

Debemos demostrar que para cada  $i, j$  y  $k$ , existe una expresión regular  $r_{ij}^k$  que denota al lenguaje  $R_{ij}^k$ . Procederemos mediante inducción en  $k$ .

**Base** ( $k = 0$ ).  $R_{ij}^0$  es un conjunto finito de cadenas, cada una de las cuales es  $\epsilon$  o un solo símbolo. Por tanto,  $r_{ij}^0$  puede escribirse como  $a_1 + a_2 + \dots + a_p$  (o  $a_1 + a_2 + \dots + a_p + \epsilon$  si  $i = j$ ), en donde  $\{a_1, a_2, \dots, a_p\}$  es el conjunto de todos los símbolos  $a$  tales que  $\delta(q_i, a) = q_j$ . Si no existen tales  $a$ , entonces  $\emptyset$  (o  $\epsilon$  en el caso  $i = j$ ) sirve como  $r_{ij}^0$ .

**Inducción** La fórmula recursiva para  $R_{ij}^k$  dada en (2.1) sólo implica a los operadores de expresiones regulares: unión, concatenación y cerradura. Por la hipótesis de inducción, para cada  $\ell$  y  $m$  existe una expresión regular  $r_{\ell m}^{k-1}$  tal que  $L(r_{\ell m}^{k-1}) = R_{\ell m}^{k-1}$ . Así pues, para  $r_{ij}^k$  podemos seleccionar la expresión regular

$$(r_{ik}^{k-1})(r_{kk}^{k-1})^*(r_{kj}^{k-1}) + r_{ij}^{k-1},$$

que completa la inducción.

Para terminar la prueba sólo tenemos que observar que

$$L(M) = \bigcup_{q_j \in F} R_{ij}^n$$

ya que  $R_{ij}^n$  representa las etiquetas de todas las trayectorias de  $q_i$  a  $q_j$ . De aquí que  $L(M)$  sea denotada por la expresión regular

$$r_{1j_1}^n + r_{1j_2}^n + \dots + r_{1j_p}^n,$$

en donde  $F = \{q_{j_1}, q_{j_2}, \dots, q_{j_p}\}$ . □

**Ejemplo 2.13** Sea  $M$  el FA que se muestra en la Fig. 2.16. Los valores de  $r_{ij}^k$  para todas las  $i$  y  $j$  y para  $k = 0, 1$  o  $2$  se tabulan en la Fig. 2.17. Se han utilizado ciertas equivalencias entre las expresiones regulares como  $(r + s)t = rt + st$  y  $(\epsilon + r)^* = r^*$  para simplificar las expresiones (véase el Ejercicio 2.16). Por ejemplo, estrictamente hablando, la expresión para  $r_{22}^1$  está dada por

$$r_{22}^1 = r_{21}^0(r_{11}^0)^*r_{12}^0 + r_{22}^0 = 0(\epsilon)^*0 + \epsilon.$$

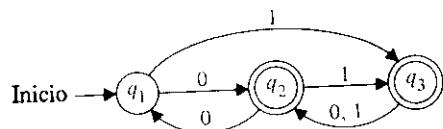


Fig. 2.16 FA del Ejemplo 2.13.

	$k = 0$	$k = 1$	$k = 2$
$r_{11}^k$	$\epsilon$	$\epsilon$	$(00)^*$
$r_{12}^k$	0	0	$0(00)^*$
$r_{13}^k$	1	1	$0^*1$
$r_{21}^k$	0	0	$0(00)^*$
$r_{22}^k$	$\epsilon$	$\epsilon + 00$	$(00)^*$
$r_{23}^k$	1	$1 + 01$	$0^*1$
$r_{31}^k$	$\emptyset$	$\emptyset$	$(0 + 1)(00)^*0$
$r_{32}^k$	$0 + 1$	$0 + 1$	$(0 + 1)(00)^*$
$r_{33}^k$	$\epsilon$	$\epsilon$	$\epsilon + (0 + 1)0^*1$

Fig. 2.17 Tabulación de  $r_k^k$  para el NFA de la Fig. 2.16.

De manera similar,

$$r_{13}^2 = r_{12}^1(r_{22}^1)^*r_{23}^1 + r_{13}^1 = 0(\epsilon + 00)^*(1 + 01) + 1.$$

Reconociendo que  $(\epsilon + 00)^*$  es equivalente a  $(00)^*$  y que  $1 + 01$  es equivalente a  $(\epsilon + 0)1$ , tenemos

$$r_{13}^2 = 0(00)^*(\epsilon + 0)1 + 1.$$

Observe que  $(00)^*(\epsilon + 0)$  es equivalente a  $0^*$ . Así pues,  $0(00)^*(\epsilon + 0)1 + 1$  es equivalente a  $00^*1 + 1$  y por tanto a  $0^*1$ .

Para completar la construcción de la expresión regular para  $M$ , que es  $r_{12}^3 + r_{13}^3$ , escribimos

$$\begin{aligned} r_{12}^3 &= r_{13}^2(r_{33}^2)^*r_{32}^2 + r_{12}^2 \\ &= 0^*1(\epsilon + (0 + 1)0^*1)^*(0 + 1)(00)^* + 0(00)^* \\ &= 0^*1((0 + 1)0^*1)^*(0 + 1)(00)^* + 0(00)^* \end{aligned}$$

y

$$\begin{aligned} r_{13}^3 &= r_{13}^2(r_{33}^2)^*r_{33}^2 + r_{13}^2 \\ &= 0^*1(\epsilon + (0 + 1)0^*1)^*(\epsilon + (0 + 1)0^*1) + 0^*1 \\ &= 0^*1((0 + 1)0^*1)^*. \end{aligned}$$

De aquí que

$$r_{12}^3 + r_{13}^3 = 0^*1((0 + 1)0^*1)^*(\epsilon + (0 + 1)(00)^*) + 0(00)^*.$$

## 2.6 AUTOMATAS FINITOS DE DOS DIRECCIONES

Hemos visto al autómata finito como una unidad de control que lee una cinta, moviéndose un cuadrado hacia la derecha en cada movimiento. Le agregamos el no determinismo al modelo, que permitió la existencia de muchas «copias» de la unidad de control y así poder barrer la cinta de manera simultánea. En seguida añadimos las transiciones  $\epsilon$ , que permitieron el cambio de estado sin leer el símbolo de entrada o mover la cabeza de la cinta. Otra extensión interesante es permitir que la cinta tenga la capacidad de moverse tanto a la izquierda como a la derecha. Un autómata finito de esta naturaleza se conoce como *autómata finito de dos direcciones*. Acepta una cadena de entrada si mueve la cabeza de la cinta fuera del extremo derecho de la cinta, y al mismo tiempo accesa un estado de aceptación. Veremos que aun esta generalización no es suficiente para aumentar el poder del autómata finito; un FA de dos direcciones sólo acepta conjuntos regulares. Solamente damos la prueba para el caso especial de un FA de dos direcciones que es determinístico, y cuya cabeza de cinta debe moverse hacia la izquierda o la derecha en cada movimiento (no permanece estacionaria). En los ejercicios se toma en cuenta un modelo más general.

Un *autómata finito determinístico de dos direcciones* (2DFA) es una quíntupla  $M = (Q, \Sigma, \delta, q_0, F)$ , en la que  $Q$ ,  $\Sigma$ ,  $q_0$  y  $F$  se definen igual que antes, y  $\delta$  es una transformación de  $Q \times \Sigma$  a  $Q \times [L, R]$ . Si  $\delta(q, a) = (p, L)$ , entonces en el estado  $q$ , barriendo el símbolo de entrada  $a$ , el 2DFA entra en el estado  $p$  y mueve su cabeza hacia la izquierda un cuadro. Si  $\delta(q, a) = (p, R)$ , el 2DFA entra en el estado  $p$  y mueve su cabeza hacia la derecha un cuadro.

Cuando se hizo la descripción del comportamiento de un FA de una dirección, extendimos  $\delta$  a  $Q \times \Sigma^*$ . Esto corresponde a considerar al FA como si recibiera un símbolo en un canal de entrada, procesaría tal símbolo y pidiera el siguiente. Este concepto es insuficiente para un FA de dos direcciones, ya que el 2DFA puede moverse a la izquierda. Por tanto, la noción de entrada que se está escribiendo en la cinta es de la mayor importancia. En lugar de intentar extender  $\delta$ , introduciremos el concepto de *descripción instantánea* (ID) de un 2DFA, que describe a la cadena de entrada, el estado actual y la posición actual de la cabeza de entrada. Despues introduciremos la relación  $\overline{\mid}_M$  sobre las ID, tal que  $I_1 \overline{\mid}_M I_2$  si y sólo si  $M$  puede desplazarse de la descripción instantánea  $I_1$  a  $I_2$  en un sólo movimiento.

Un ID de  $M$  es una cadena de  $\Sigma^*Q\Sigma^*$ . La ID  $wqx$ , en la que  $w$  y  $x$  están en  $\Sigma^*$  y  $q$  en  $Q$ , tiene la intención de representar los siguientes hechos:

1.  $wx$  es la cadena de entrada,
2.  $q$  es el estado actual y
3. la cabeza de entrada se encuentra leyendo el primer símbolo de  $x$ .

Si  $x = \epsilon$ , entonces la cabeza de entrada se ha movido fuera del extremo derecho de la entrada.

Definimos la relación  $\overline{\mid}_M$  o simplemente  $\overline{\mid}$ , si  $M$  se sobreentiende, mediante:

1.  $a_1 a_2 \cdots a_{i-1} q a_i \cdots a_n \vdash a_1 a_2 \cdots a_{i-1} a_i p a_{i+1} \cdots a_n$  siempre que  $\delta(q, a_i) = (p, R)$  y

2.  $a_1 a_2 \cdots a_{i-2} a_{i-1} q a_i \cdots a_n \cdots a_1 a_2 \vdash a_{i-2} p a_{i-1} a_i \cdots a_{i-1} a_n$  siempre que  $\delta(q, a_i) = (p, L)$  e  $i > 1$ .

La condición  $i > 1$  evita cualquier acción en caso de que la cabeza de la cinta se moviera saliéndose del extremo izquierdo de la cinta. Nótese que no es posible hacer un movimiento si  $i = n + 1$  (la cabeza de la cinta se ha salido del extremo derecho). Sea  $\vdash^*$  la cerradura reflexiva y transitiva de  $\vdash$ . Es decir,  $I \vdash^* I$  para toda ID  $I$ , e  $I_1 \vdash^* I_k$  siempre que  $I_1 \vdash I_2 \vdash \cdots \vdash I_k$  para algunas  $I_2, \dots, I_{k-1}$ .

Definimos

$$L(M) = \{w \mid q_0 w \vdash^* wp \text{ para alguna } p \text{ en } F\}.$$

Esto es,  $w$  es aceptada por  $M$  si, comenzando en el estado  $q_0$  con  $w$  sobre la cinta de entrada y la cabeza en el extremo izquierdo de  $w$ ,  $M$  finalmente entra en un estado final al mismo tiempo que sale del extremo derecho de la cinta de entrada.

**Ejemplo 2.14** Considere un 2DFA  $M$  que se comporta de la manera siguiente: se comienza en el estado  $q_0$ ,  $M$  repite un ciclo de movimientos en los que la cabeza de la cinta se mueve hacia la derecha hasta que encuentra dos unos, entonces se mueve hacia la izquierda hasta encontrar un cero, en el cual es reaccesado el estado puntual  $q_0$ , y el ciclo se repite. Más precisamente,  $M$  posee tres estados, todos ellos estados finales;  $\delta$  se da en la Fig. 2.18.

	0	1
$q_0$	$(q_0, R)$	$(q_1, R)$
$q_1$	$(q_1, R)$	$(q_2, L)$
$q_2$	$(q_0, R)$	$(q_2, L)$

Fig. 2.18 Función de transición para el 2DFA del Ejemplo 2.14.

Considérese la entrada 101001. Puesto que  $q_0$  es el estado inicial, la primera ID es  $q_0 101001$ . Para obtener la segunda ID, tome en cuenta que el símbolo que está inmediatamente a la derecha del estado  $q_0$  en la primera ID es un 1 y  $\delta(q_0, 1)$  es  $(q_1, R)$ . Así pues, la segunda ID es  $1q_1 01001$ . Continuando de esta forma llegamos al resultado que se presenta en la Tabla 2.1. De aquí que  $M$  finalmente se mueva fuera del extremo derecho de la cinta en un estado de aceptación. Por tanto, 101001 está en  $L(M)$ .

Tabla 2.1

$q_0 101001$	$1q_1 01001$
	$10q_1 001$
	$1q_2 01001$
	$10q_0 1001$
	$101q_1 001$
	$1010q_1 01$
	$10100q_1 1$
	$1010q_2 01$
	$10100q_0 1$
	$101001q_1$

### Secuencias cruzadas

Una representación útil del comportamiento de un 2DFA consiste en la entrada, la trayectoria que sigue la cabeza y el estado que se toque cada vez que se cruza la frontera entre dos cuadros de la cinta, con la suposición de que el control entra en su nuevo estado antes de mover la cabeza. Por ejemplo, en la Fig. 2.19 se presenta el comportamiento del 2DFA del Ejemplo 2.14 sobre 101001.

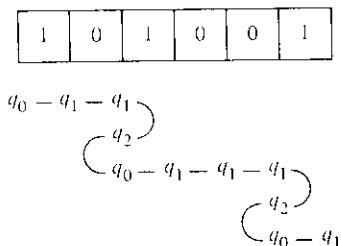


Fig. 2.19 Comportamiento del 2DFA del Ejemplo 2.14.

La lista de estados por debajo de cada frontera entre cuadros se conoce como *secuencia cruzada*. Adviértase que si un 2DFA acepta su entrada, ninguna secuencia cruzada puede tener un estado repetido si la cabeza se mueve en la misma dirección, de otra manera el 2DFA, siendo determinístico, estaría en un ciclo y por tanto no podría nunca caer fuera del extremo derecho.

Otra observación importante acerca de las secuencias cruzadas consiste en la primera vez que se cruza una frontera, la cabeza debe moverse hacia la derecha. Los cruces posteriores deben ser en direcciones alternas. Por tanto, los elementos con número impar de una secuencia cruzada representan movimientos a la derecha y los elementos con número par representan movimientos hacia la izquierda. Si la entrada es aceptada, se concluye que todas las secuencias cruzadas son de longitud impar.

Se dice que una secuencia cruzada  $q_1, q_2, \dots, q_k$  es válida si es de longitud impar,

y no existen dos elementos con número impar y par que sean idénticos. Un 2DFA con  $s$  estados puede tener secuencias cruzadas válidas con longitud de cuando mucho  $2s$ , de tal forma que el número de secuencias cruzadas sea finito.

Nuestra estrategia para demostrar que cualquier conjunto aceptado por un 2DFA<sup>M</sup> es regular consiste en construir un NFA equivalente cuyos estados son las secuencias cruzadas válidas de  $M$ . Para construir la función de transición del NFA primero examinamos la relación entre secuencias cruzadas adyacentes.

Supóngase que se tiene un cuadro de cinta aislado que contiene al símbolo  $a$  y se tiene también dos secuencias cruzadas válidas  $q_1, q_2, \dots, q_k$  y  $p_1, p_2, \dots, p_\ell$  en los límites izquierdo y derecho del cuadro, respectivamente. Nótese que puede no haber cadenas de entrada que puedan ser ligadas a los extremos izquierdo y derecho del símbolo  $a$ , que en realidad produzcan estas dos secuencias cruzadas. Sin embargo podemos probar que en realidad produzcan estas dos secuencias de la manera siguiente: Si la cabeza de la cinta se mueve hacia la izquierda del cuadro que contiene  $a$  en el estado  $q_i$ , reinicie al autómata en el cuadro que contiene  $a$  en el estado  $q_{i+1}$ . De forma similar, cuando la cabeza de la cinta se mueva hacia la derecha del cuadro en el estado  $p_i$ , reinicie al autómata en el cuadro  $p_{i+1}$ . Con este método podemos probar las dos secuencias cruzadas para asegurarnos que son localmente consistentes. A continuación, estas ideas se establecen con más precisión.

En los incisos (i) a (v) que se dan más adelante definimos, de manera recursiva, los conceptos de pares<sup>de concordancia derecha y de concordancia izquierda</sup>. La intención es que  $q_1, q_2, \dots, q_k$  concuerden con la derecha con  $p_1, p_2, \dots, p_\ell$  sobre  $a$ , si las dos secuencias son consistentes, suponiendo que inicialmente alcanzamos  $a$  en el estado  $q_1$  moviéndonos hacia la derecha; y para que las dos secuencias concuerden por la izquierda, si son consistentes, suponemos que inicialmente alcanzamos  $a$  en el estado  $p_1$ , si nos movemos hacia la izquierda. En cada caso, tomamos  $q_1, q_2, \dots, q_k$  en el límite izquierdo de  $a$  y  $p_1, p_2, \dots, p_\ell$ , en el límite derecho.

- i) La secuencia nula concuerda por la izquierda y por la derecha con la secuencia nula. Es decir, si nunca alcanzamos al cuadro que contiene  $a$ , entonces se desprenden de que las fronteras en ninguno de los dos lados sean cruzadas.
- ii) Si  $q_1, \dots, q_k$  concuerdan por la derecha a  $p_1, \dots, p_\ell$ , y  $\delta(q_1, a) = (q_2, L)$ , entonces  $q_1, \dots, q_k$  concuerda por la derecha con  $p_1, \dots, p_\ell$ . Esto es, si el primer cruce de la frontera izquierda es en el estado  $q_1$  y la cabeza se mueve inmediatamente hacia la izquierda al estado  $q_2$ , entonces, si seguimos estos dos cruces mediante un comportamiento consistente, comenzando en otro cruce del límite izquierdo, obtenemos un par consistente de secuencias con un primer cruce hacia la derecha, es decir, un par que concuerda por la derecha.
- iii) Si  $q_1, \dots, q_k$  concuerda por la izquierda con  $p_2, \dots, p_\ell$ , y  $\delta(q_1, a) = (p_1, R)$ , entonces  $q_1, \dots, q_k$  concuerda por la derecha  $p_1, \dots, p_\ell$ . Esto es, si el primer cruce del límite izquierdo se hace en el estado  $q_1$  y la cabeza se mueve inmediatamente hacia la derecha en el estado  $p_1$ , entonces si seguimos estos dos cruces mediante un comportamiento consistente, comenzando con un cruce del límite derecho, obtenemos un par consistente de secuencias en las que el primer cruce es hacia la derecha, es decir, un par que concuerda por la derecha. Nótese que este caso introduce la necesidad de secuencias concordantes por la izquierda, aun cuando en

realidad sólo estemos interesados en pares que concuerden por la derecha.

- iv) Si  $q_1, \dots, q_k$  concuerda por la izquierda con  $p_3, \dots, p_\ell$ , y  $\delta(p_1, a) = (p_2, R)$ , entonces  $q_1, \dots, q_k$  concuerda por la izquierda con  $p_1, \dots, p_1$ . La justificación es parecida a la de la regla (ii).
- v) Si  $q_2, \dots, q_k$  concuerda por la derecha con  $p_2, \dots, p_\ell$ , y  $\delta(p_1, a) = (q_1, L)$ , entonces  $q_1, \dots, q_k$  concuerda por la izquierda con  $p_1, \dots, p_\ell$ . La justificación es parecida a la de la regla (iii).

**Ejemplo 2.15** Considérese al 2DFA  $M$  del Ejemplo 2.14 y un cuadro de cinta que contiene al símbolo 1. La secuencia nula concuerda por la izquierda a la secuencia nula, y  $\delta(q_0, 1) = (q_1, R)$ . Por tanto,  $q_0$  concuerda por la derecha con  $q_1$  sobre 1 por la regla (iii). Puesto que  $\delta(q_1, 1) = (q_2, L)$ ,  $q_1, q_2, q_0$  concuerda por la derecha con  $q_1$  sobre 1 por la regla (ii). Este debe ser el caso ya que, de hecho, existe un cálculo de aceptación en el que este par de secuencias en realidad ocurre hacia la izquierda y la derecha de un cuadro que contiene un 1. Advierta, sin embargo, que un par de secuencias podrían concordar aunque no exista un cálculo en el cual aparezcan una al lado de otra, y podría ser imposible encontrar cadenas para ser situadas a la izquierda y derecha que «lleven al cálculo» a los estados correctos.

### Equivalencia de los autómatas finitos de una y dos direcciones

**Teorema 2.5** Si  $L$  es aceptado por un 2DFA, entonces  $L$  es un conjunto regular.

**Demostración** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un 2DFA. La demostración consiste en la construcción de un NFA  $M'$  que acepte a  $L(M)$ . Defínase  $M'$  como la quíntupla  $(Q', \Sigma', \delta', q'_0, F')$ , en la que

1.  $Q'$  consiste en todas las secuencias cruzadas válidas para  $M$ .
2.  $q'_0$  es la secuencia cruzada que consiste solamente en  $q_0$ .
3.  $F'$  es el conjunto de todas las secuencias cruzadas de longitud uno y que consisten en un estado de  $F$ .
4.  $\delta'(c, a) = \{d | d$  es una secuencia cruzada válida que concuerda por la derecha con  $c$  sobre la entrada  $a\}$ . Nótese que como  $d$  es válida debe ser de longitud impar.

De manera intuitiva se tiene la idea de que  $M'$  reúne las piezas del cálculo de  $M$  conforme recorre la cadena de entrada. Esto se hace adivinando secuencias cruzadas sucesivas. Si  $M'$  ha adivinado que  $c$  es la secuencia cruzada en una frontera y que  $a$  es el siguiente símbolo de entrada, entonces  $M'$  puede adivinar cualquier secuencia cruzada válida a la que  $c$  ajuste por la derecha sobre la entrada  $a$ . Si el cálculo adivinado tiene como resultado a  $M$  moviéndose fuera del extremo derecho de la entrada hacia un estado de aceptación, entonces  $M'$  acepta a  $L(M)$ .

Mostraremos, ahora, que  $L(M') = L(M)$ . Sea  $w$  una cadena de  $L(M)$ . Observe las secuencias cruzadas que son generadas por un cálculo de aceptación de  $M$  sobre  $w$ . Cada secuencia cruzada ajustada por la derecha con la que se encuentra en el siguiente límite, así que  $M'$  puede adivinar la secuencia cruzada apropiada (entre otras alternativas) y aceptarla.

De manera inversa, si  $w$  está en  $L(M')$ , considera la secuencia cruzada  $c_0, c_1, \dots, c_n$  de  $M$  correspondiente a los estados de  $M'$  conforme  $M'$  barre  $w = a_1 a_2 \dots a_n$ . Para cada  $i$ ,  $0 \leq i < n$ ,  $c_i$  ajusta por la derecha con  $c_{i+1}$  sobre  $a_i$ . Podemos construir un cálculo aceptable de  $M$  sobre la entrada  $w$  mediante la determinación del momento en que la cabeza invierte su dirección. En particular, probamos por inducción sobre  $i$  que  $M'$  al leer  $a_1 a_2 \dots a_i$  puede entrar al estado  $c_i = [q_1, \dots, q_k]$  sólo si

1.  $M$  iniciado en el estado  $q_0$  sobre  $a_1 a_2 \dots a_i$  primero se moverá hacia la derecha a partir de la posición  $i$  en el estado  $q_1$ , y
2. para  $j=2, 4, \dots$ , si  $M$  es iniciado en la posición  $i$  del estado  $q_j$ ,  $M$  se moverá finalmente hacia la derecha de la posición  $i$  del estado  $q_{j+1}$  (esto implica que  $k$  debe ser impar).

**Base** ( $i = 0$ ). Como  $c_0 = [q_0]$ , (1) se satisface porque  $M$  comienza su cálculo "moviéndose a la derecha" de la posición 0 en el estado  $q_0$ . La condición (2) se cumple de manera inmediata.

**Inducción** Supóngase que la hipótesis es verdad para  $i - 1$ . Supóngase también que  $M'$  al leer  $a_1 a_2 \dots a_i$  puede entrar al estado  $c_i = [p_1, \dots, p_\ell]$  desde el estado  $c_{i-1} = [q_1, \dots, q_k]$ . Como  $k$  y  $\ell$  son impares y  $c_{i-1}$  concuerda por la derecha con  $c_i$  sobre  $a_i$ , debe existir una  $j$  impar tal que en el estado  $q_j$  sobre la entrada  $a_i$ ,  $M$  se mueva hacia la derecha. Sea  $j_1$  la menor de tales  $j$ . Por definición de "concorda por la derecha" se sigue que  $\delta(q_{j-1}, a_i) = (p_1, R)$ . Esto demuestra a (1). También por la definición de "concorda por la derecha" (regla iii)  $[q_{j+1}, \dots, q_k]$  concuerda por la izquierda con  $[p_2, \dots, p_\ell]$ . Ahora, si  $\delta(p_j, a_i) = (p_{j+1}, R)$  para toda  $j$  par, entonces se sigue (2) de manera inmediata. En el caso de que para la  $j_2$  par más pequeña,  $\delta(p_{j_2}, a_i) = (q, L)$ , entonces por la definición de "concorda por la izquierda" (regla v)  $q$  debe ser  $q_{j_1+1}$  y  $[q_{j_1+2}, \dots, q_k]$  "concorda por la derecha" con  $[p_{j_2+1}, \dots, p_\ell]$ . El argumento se repite entonces con las últimas secuencias en lugar de  $c_{i-1}$  y  $c_i$ .

Con la hipótesis de inducción para toda  $i$  establecida, el hecho de que  $c_n = [p]$  para alguna  $p$  implica que  $M$  acepta a  $a_1 a_2 \dots a_n$ .  $\square$

**Ejemplo 2.16** Considera la construcción de un NFA  $M'$  equivalente al 2DFA  $M$  del Ejemplo 2.14. Puesto que sólo se entra a  $q_2$  mediante un movimiento hacia la izquierda y a  $q_1$  y  $q_3$  sólo se entra con movimientos a la derecha, todos los componentes con número par de secuencias cruzadas deben ser  $q_2$ . Como una secuencia cruzada válida debe ser de longitud impar y dos estados con número impar no pueden ser el mismo, debe ser de los cuatro posibles. Como en el caso de dos estados con número par, existen sólo cuatro secuencias cruzadas de interés; éstas se enumeran en la Fig. 2.20 junto con sus concordancias por la derecha.

Secuencias cruzadas válidas	Concordancias por la derecha sobre 0	Concordancias por la derecha sobre 1
$[q_0]$	$[q_0]$	$[q_1]$
$[q_1]$	$[q_1], [q_1, q_2, q_0]$	$[q_1]$
$[q_0, q_2, q_1]$	$-$	$-$
$[q_1, q_2, q_0]$	$-$	$[q_1]$

Fig. 2.20 Secuencias cruzadas válidas junto con sus concordancias por la derecha.

Notamos inmediatamente que el estado  $[q_0, q_2, q_1]$  puede ser eliminado del NFA  $M'$  construido, puesto que no tiene concordancia por la derecha. El  $M'$  resultante se muestra en la Fig. 2.21. Nótese que  $L(M') = (\epsilon + 1)(0 + 01)^*$ , es decir, todas las cadenas de 0s y 1s que no tienen dos 1s consecutivos.

Considérese la entrada 1001, que es aceptada por  $M'$  utilizando la secuencia de estados  $[q_0], [q_1], [q_1], [q_1, q_2, q_0], [q_1]$ . Podemos visualizar a las secuencias cruzadas como en la Fig. 2.22. Nótese que  $\delta(q_0, 1) = (q_1, R)$  justifica el primer movimiento y que  $\delta(q_1, 0) = (q_1, R)$  justifica al segundo y al tercero. Puesto que  $\delta(q_1, 1) = (q_2, L)$  vemos

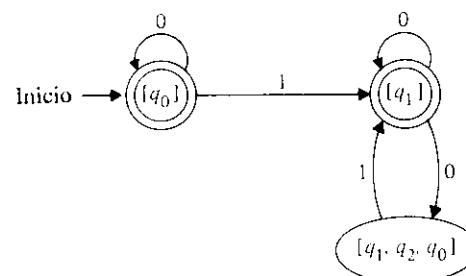


Fig. 2.21 NFA  $M'$  construido a partir del 2DFA  $M$ .

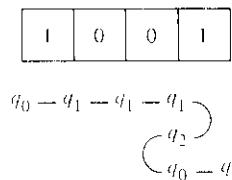


Fig. 2.22 Secuencias cruzadas del 2 DFA sobre la entrada 1001.

la justificación para el cuarto movimiento, que invierte la dirección del viaje. Entonces  $\delta(q_2, 0) = (q_0, R)$  otra vez invierte la dirección y, finalmente,  $\delta(q_0, 1) = (q_1, R)$  explica el último movimiento.

## 2.7 AUTOMATAS FINITOS CON SALIDA

Una limitación del autómata finito, como lo hemos definido, consiste en que su salida está limitada a una señal binaria: "aceptada"/"no aceptada". Se han considerado modelos en los que la salida se escoge de algún otro alfabeto. Existen dos planteamientos distintos; la salida puede estar asociada con el estado (conocida como *máquina de Moore*) o con la transición (conocida como *máquina de Mealy*). Definiremos cada una de éstas de manera formal y después mostraremos que los dos tipos de máquina producen las mismas transformaciones entrada-salida.

## Máquinas de Moore

Una máquina de Moore es un conjunto de seis elementos  $(Q, \Sigma, \Delta, \delta, \lambda, q_0)$  en el que  $Q, \Sigma, \Delta$  y  $q_0$  se definen como en el caso del DFA.  $\Delta$  es el alfabeto de salida y  $\lambda$  es una transformación de  $Q$  a  $\Delta$  que da la salida asociada con cada estado. La salida de  $M$  en respuesta a la entrada  $a_1 a_2 \dots a_n$ ,  $n \geq 0$ , es  $\lambda(q_0) \lambda(q_1) \dots \lambda(q_n)$ , en donde  $q_0, q_1, \dots, q_n$  es la secuencia de estados tales que  $\delta(q_{i-1}, a) = q_i$  para  $1 \leq i \leq n$ . Adviértase que cualquier máquina de Moore produce una salida  $\lambda(q_0)$  en respuesta a la salida  $\epsilon$ . El DFA puede ser considerado un caso especial de una máquina de Moore en la que el alfabeto de salida es  $\{0, 1\}$  y el estado  $q$  es de "aceptación" si y sólo si  $\lambda(q) = 1$ .

**Ejemplo 2.17** Supóngase que deseamos determinar el residuo mod 3 para cada cadena binaria tratada como un entero binario. Para empezar, obsérvese que si  $i$  escrito en binario está seguido por un 0, la cadena resultante tiene valor  $2^i$ , y si  $i$  en binario está seguido por un 1, la cadena resultante tiene valor  $2^i + 1$ . Si el residuo de  $i/3$  es  $p$ , entonces el residuo de  $2^i/3$  es  $2p \bmod 3$ . Si  $p = 0, 1$  o  $2$ , entonces  $2p \bmod 3$  es 0, 2 o 1, respectivamente. De manera similar, el residuo de  $(2i+1)/3$  es 1, 0 o 2, respectivamente.

Es suficiente, por tanto, diseñar una máquina de Moore con tres estados,  $q_0, q_1$  y  $q_2$ , en donde  $q_j$  es accesado si y sólo si la entrada vista hasta aquí tiene residuo  $j$ . Definimos

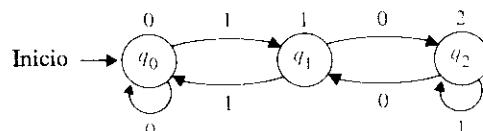


Fig. 2.23 Máquina de Moore que calcula residuos.

$\lambda(q_j) = j$  para  $j = 0, 1$  y  $2$ . En la Fig. 2.23 presentamos el diagrama de transiciones en donde las salidas etiquetan a los estados. La función de transición  $\delta$  se construye de modo que refleje las reglas que toman en cuenta el cálculo de residuos descrito arriba.

En la entrada 1010 la secuencia de estados accesados es  $q_0, q_1, q_2, q_2, q_1$ , y produce la secuencia de salida 01221. Es decir,  $\epsilon$  (que tiene "valor" 0) tiene un residuo de 0, 1 tiene residuo 1, 2 (en decimal) tiene residuo 2, 5 tiene residuo 2 y 10 (en decimal) tiene residuo 1.

## Máquinas de Mealy

Una máquina de Mealy es también un conjunto de seis parámetros  $M = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$ , en el que todos los parámetros se definen igual que en el caso de la máquina de Moore, con excepción de  $\lambda$ , que transforma de  $Q \times \Sigma$  a  $\Delta$ . Esto es,  $\lambda(q, a)$  da la salida asociada con la transición del estado  $q$  sobre la entrada  $a$ . La salida de  $M$  en respuesta a la entrada  $a_1 a_2 \dots a_n$  es  $\lambda(q_0, a_1) \lambda(q_1, a_2) \dots \lambda(q_{n-1}, a_n)$ , en donde  $q_0, q_1, \dots, q_n$  es la secuencia de estados tales que  $\delta(q_{i-1}, a_i) = q_i$  para  $1 \leq i \leq n$ . Nótese que esta secuencia

tiene longitud  $n$  más que  $n + 1$  como en el caso de la máquina de Moore, y sobre la entrada  $\epsilon$  una máquina de Mealy produce una salida  $\epsilon$ .

**Ejemplo 2.18** Aun cuando el alfabeto de salida tenga sólo dos símbolos, el modelo de la Máquina de Mealy puede salvar estados cuando se le compara con un autómata finito. Considere el lenguaje  $(0+1)^*(00+11)$  de todas las cadenas de 0s y 1s cuyos últimos dos símbolos son los mismos. En el capítulo siguiente desarrollaremos las herramientas necesarias para demostrar que este lenguaje no es aceptado por ningún DFA con menos de cinco estados. Sin embargo, podemos definir una máquina de Mealy de tres estados que utilice su estado para recordar el último símbolo leído, emita una salida y cuando la entrada actual alcance a la anterior y que emita  $n$  en cualquier otro caso. La secuencia de ys y ns emitida por la máquina de Mealy corresponde a la secuencia de aceptación y no aceptación de estados accesados por un DFA sobre la misma entrada; sin embargo, la máquina de Mealy no emite ninguna salida anterior a una entrada, mientras que el DFA rechaza la cadena  $\epsilon$  cuando su estado inicial no es un estado final.

La máquina de Mealy  $M = ((q_0, p_0, p_1), \{0, 1\}, \{y, n\}, \delta, \lambda, q_0)$  se presenta en la Fig. 2.24. Utilizamos la etiqueta  $a/b$  en el arco que va de  $p$  a  $q$  para indicar que  $\delta(p, a) = q$  y  $\lambda(p, a) = b$ . La respuesta de  $M$  a la entrada 01100 es  $nyny$ , y la sucesión de estados accesados es  $q_0, p_0, p_1, p_1, p_0, p_0$ . Nótese cómo  $p_0$  recuerda a un cero y  $p_1$  recuerda a un uno. El estado  $q_0$  es inicial y "recuerda" que aún no se ha recibido ninguna entrada.

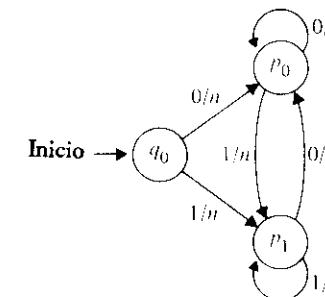


Fig. 2.24 Máquina de Mealy.

## Equivalencia de las máquinas de Moore y Mealy

Sea  $M$  una máquina de Mealy o de Moore. Defínase  $T_M(w)$ , para la cadena de entrada  $w$ , como la salida producida por  $M$  sobre la entrada  $w$ . Puede ser que nunca exista una identidad exacta entre las funciones  $T_M$  y  $T_{M'}$  si  $M$  es una máquina de Mealy y  $M'$  una de Moore, porque  $|T_M(w)|$  es uno menos que  $|T_{M'}(w)|$  para cada  $w$ . No obstante, podemos despreciar la respuesta de una máquina de Moore a la entrada  $\epsilon$  y decir que la máquina de Mealy  $M$  y la máquina de Moore  $M'$  son equivalentes si para todas las entradas  $w$ ,  $bT_M(w) = T_{M'}(w)$ , en donde  $b$  es la salida de  $M'$  para su estado inicial. Podemos entonces probar los teoremas siguientes, que igualan los modelos de Mealy y Moore.

**Teorema 2.6** Si  $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$  es una máquina de Moore, entonces existe una máquina de Mealy  $M_2$  que es equivalente a  $M_1$ .

**Demuestra** Sea  $M_2 = (Q, \Sigma, \Delta, \delta, \lambda', q_0)$  y defínase  $\lambda'(q, a)$  como  $\lambda(\delta(q, a))$  para todos los estados  $q$  y símbolos de entrada  $a$ . Entonces  $M_1$  y  $M_2$  acceden la misma secuencia de estados sobre la misma entrada, y con cada transición  $M_2$  emite la salida que  $M_1$  asocia con el estado accesado.

**Teorema 2.7** Sea  $M_1 = (Q, \Sigma, \Delta, \delta, \lambda, q_0)$  una máquina de Mealy. Entonces existe una máquina de Moore  $M_2$  equivalente a  $M_1$ .

**Demuestra** Sea  $M_2 = (Q \times \Delta, \Sigma, \Delta, \delta', \lambda', [q_0, b_0])$ , en donde  $b_0$  es un miembro de  $\Delta$  seleccionado de manera arbitraria. Esto es, los estados de  $M_2$  son pares  $[q, b]$  que consisten en un estado de  $M_1$  y un símbolo de salida. Defínase  $\delta'([q, b], a) = [\delta(q, a), \lambda(q, a)]$  y  $\lambda'([q, b]) = b$ . El segundo componente de un estado  $[q, b]$  de  $M_2$  es la salida hecha por  $M_1$  sobre alguna transición al estado  $q$ . Sólo los primeros componentes de los estados de  $M_2$  determinan los movimientos hechos por  $M_2$ . Una fácil inducción sobre  $n$  muestra que si  $M_1$  accesa a los estados  $q_0, q_1, \dots, q_n$  sobre la entrada  $a_1 a_2 \dots a_n$  y emite las salidas  $b_1, b_2, \dots, b_n$ , entonces  $M_2$  accesa a los estados  $[q_0, b_0], [q_1, b_1], \dots, [q_n, b_n]$  y emite las salidas  $b_0, b_1, b_2, \dots, b_n$ .  $\square$

**Ejemplo 2.19** Sea  $M_1$  la máquina de Mealy que se presenta en la Fig. 2.24. Los estados de  $M_2$  son  $\{q_0, y\}, [q_0, n], [p_0, y], [p_0, n], [p_1, y]$  y  $[p_1, n]$ . Escójase  $b_0 = n$ , haciendo

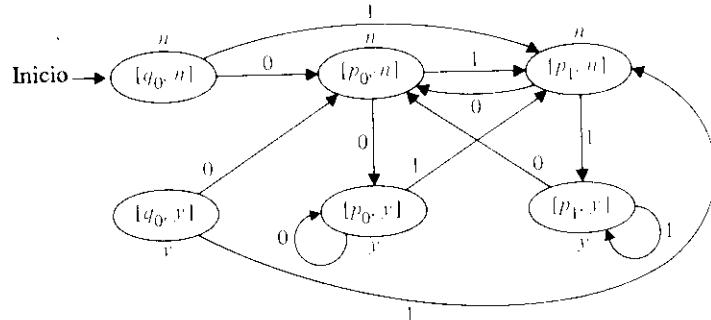


Fig. 2.25 Máquina de Moore construida a partir de la máquina de Mealy.

$[q_0, n]$  el estado inicial de  $M_2$ . En la Fig. 2.25 se muestran las transiciones y salidas de  $M_2$ . Nótese que el estado  $[q_0, y]$  nunca puede ser accesado y puede eliminarse.

## 2.8 APLICACIONES DE LOS AUTOMATAS FINITOS

Existe un gran número de problemas de diseño de software que se simplifican mediante la conversión automática de la notación de las expresiones regulares a una instrumentación eficiente de computadora del autómata finito correspondiente. En el presente trabajo mencionaremos dos de tales aplicaciones; las notas bibliográficas contienen referencias a algunas otras aplicaciones.

## Analizadores de léxico

Los tokens de un lenguaje de programación son, casi sin excepción, expresables en forma de conjuntos regulares. Por ejemplo, los identificadores de ALGOL, que son letras mayúsculas o minúsculas seguidas por cualquier sucesión de letras y dígitos, sin límite en la longitud, pueden expresarse como

$$(\text{letra}) (\text{letra} + \text{dígito})^*$$

en donde "letra" significa  $A + B + \dots + Z + a + b + \dots + z$  y "dígito"  $0 + 1 + \dots + 9$ . Los identificadores de FORTRAN, con un límite de longitud de seis y las letras restringidas a las mayúsculas y al símbolo \$, pueden expresarse como

$$(\text{letra}) (\epsilon + \text{letra} + \text{dígito})^5$$

en donde "letra" significa  $(\$ + A + B + \dots + Z)$ . Las constantes aritméticas de SNOBOL que no permiten la notación exponencial presente en muchos otros lenguajes) pueden expresarse como

$$(\epsilon + -) (\text{dígito}^* (\cdot \text{dígito}^* + \epsilon) + \cdot \text{dígito}^*)$$

Un cierto número de *generadores de analizadores de léxico* toman como entrada una secuencia de expresiones regulares que describen a los tokens y producen un solo autómata finito que reconoce a cualquier token. Por lo general, convierten la expresión regular a un NFA con transiciones  $\epsilon$  y después construyen subconjuntos de estados para producir un DFA directamente, en lugar de eliminar primero las transiciones  $\epsilon$ . Cada estado final indica el token particular que se ha encontrado, de modo que el autómata es en realidad una máquina de Moore. La función de transición del FA está codificada en una de varias formas para tomar menos espacio que el que tomaría la tabla de transiciones si se representara como un arreglo en dos dimensiones. El analizador de léxico producido por el generador es un programa fijo que interpreta tablas codificadas, junto con la tabla particular que representa al FA que reconoce a los tokens (especificados al generador en forma de expresiones regulares). Este analizador de léxico puede ser utilizado, entonces, como el módulo de un compilador. Ejemplos de generadores de analizadores de léxico que siguen el planteamiento anterior se encuentran en Johnson *et al.* [1968] y Lesk [1975].

## Editores de texto

Ciertos editores de texto y programas similares permiten la sustitución de una cadena por cualquier cadena que concuerde con una expresión regular. Por ejemplo, el editor de texto UNIX permite que un comando como

$$s/bbb^*/b/$$

sustituya a un solo espacio en blanco para la primera cadena de dos o más espacios en blanco que se encuentran en una línea dada. La expresión "cualquier" denota a  $a_1 + a_2 + \dots + a_n$ , en la que las  $a_i$  son todos los caracteres de una computadora excepto el carácter

"nueva línea". Podemos convertir una expresión regular  $r$  a un DFA que acepte a cualquier\* $r$ . Nótese que la presencia de la operación cualquier\* nos permite reconocer un miembro de  $L(r)$  que comience en cualquier lugar de la línea. Sin embargo, la conversión de una expresión regular a un DFA toma mucho más tiempo que el que toma barrer una sola línea corta utilizando el DFA, y el DFA podría tener un cierto número de estados que son una función exponencial de la longitud de la expresión regular.

Lo que realmente sucede en el editor de texto UNIX es que la expresión regular cualquier\* $r$  es convertida a un NFA con transiciones  $\epsilon$ , y el NFA entonces se simula directamente, como se sugiere en la Fig. 2.26. Sin embargo, una vez que se ha construido una columna, listando todos los estados en los que puede estar el NFA sobre un prefijo determinado de la salida, la columna inmediata anterior ya no es necesaria y se elimina con el fin de ahorrar espacio. Este planteamiento para el reconocimiento de conjuntos regulares fue expresado por vez primera en Thompson [1968].

## EJERCICIOS

- \*S 2.1 Encuentre un autómata finito cuyo comportamiento corresponda al circuito que se muestra en la Fig. 2.26, es decir que los estados finales correspondan a una salida 1. El círculo con un punto significa una compuerta Y, cuya salida es 1 sólo si ambas entradas tienen valor 1. El círculo con un + representa una compuerta 0, cuya salida es 1 cuando cualquiera de las entradas tenga valor 1. El círculo con un ~ representa un inversor, cuya salida es 1 para la entrada 0 y 0 para la entrada 1. Supóngase que existe el tiempo suficiente entre cada cambio de los valores de la entrada para que las señales se propaguen y para que la red alcance una configuración estable.

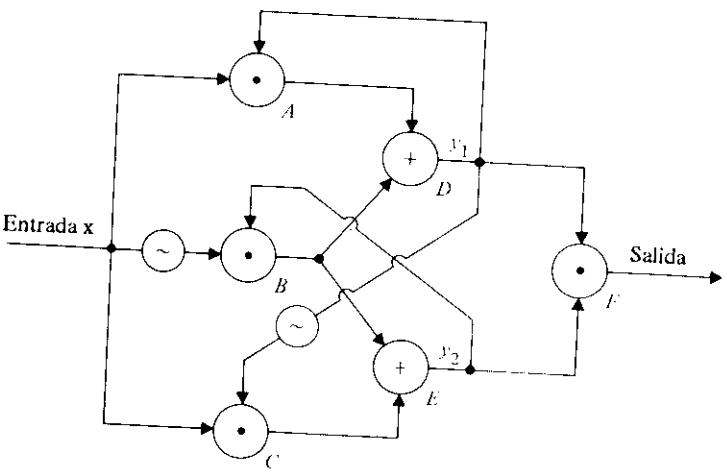


Fig. 2.26 Circuito lógico.

2.2 Históricamente, los autómatas finitos se utilizaron por primera vez para modelar redes de neuronas. Encuentre un autómata finito cuyo comportamiento sea equivalente a la red neuronal que se muestra en la Fig. 2.27. Los estados finales del autómata corresponden a una salida 1 de la red. Cada neurona tiene sinapsis excitantes (círculos) e inhibitorias (puntos). Una neurona produce una salida 1 si el número de sinapsis excitantes con entrada 1 excede al número de sinapsis inhibitorias con entrada 1, por al menos el umbral de la neurona (el número que se encuentra dentro del triángulo). Supóngase que existe tiempo suficiente entre cada cambio de valor de entrada para que las señales se propaguen y para que la red alcance una configuración estable. A continuación, suponga que inicialmente los valores de  $y_1$ ,  $y_2$  y  $y_3$  son todos 0.

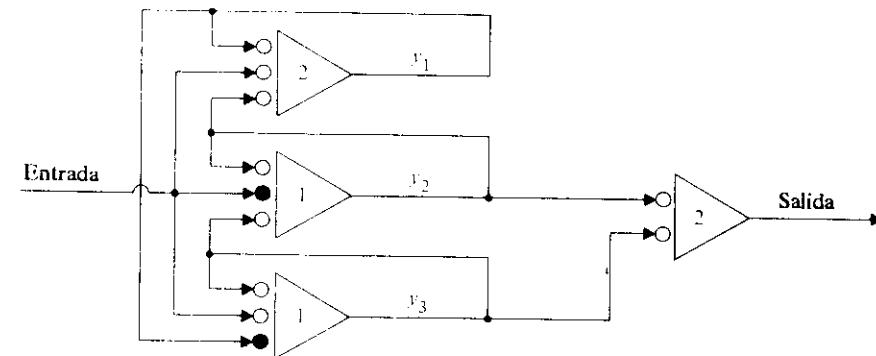


Fig. 2.27 Una red neuronal.

2.3 Considérese el juguete que se muestra en la Fig. 2.28. Se deja caer una canica en A o B. Los niveladores  $x_1$ ,  $x_2$  y  $x_3$  hacen que la canica caiga a la izquierda o a la derecha. Cuando una canica choca con un nivelador hace que éste cambie de estado, de tal modo que la siguiente canica que choque con él tomará la rama opuesta.

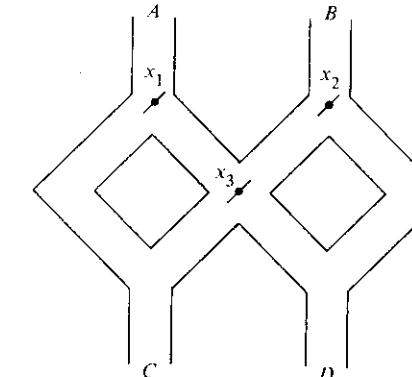


Fig. 2.28 Un juguete.

a) Modele este juguete mediante un autómata finito. Represente a una canica en  $A$  con una entrada 0 y una en  $B$  con una entrada 1. Una secuencia de entrada es aceptada si la última canica sale en el punto  $D$ .

b) Describa el conjunto aceptado por el autómata finito.

c) Modele el juguete como una máquina de Mealy cuya salida es la secuencia de  $Cs$  y  $Ds$  por las que caen canicas sucesivas.

**2.4** Supóngase que  $\delta$  es la función de transición de un DFA. Pruebe que para cualesquiera cadenas de entrada  $x$  y  $y$ ,  $\delta(q, xy) = \delta(\delta(q, x), y)$ . [Sugerencia: utilice inducción sobre  $|y|$ .]

**2.5** Proporcione autómatas finitos determinísticos que acepten los siguientes lenguajes sobre el alfabeto  $\{0, 1\}$ .

a) El conjunto de todas las cadenas que terminen en 00.

b) El conjunto de todas las cadenas que posean tres 0s consecutivos.

c) El conjunto de todas las cadenas tales que cada bloque de cinco símbolos consecutivos contenga al menos dos 0s.

d) El conjunto de todas las cadenas que comienzan con un 1 y que, interpretado como la representación binaria de un entero, sea congruente con cero mod 5.

e) El conjunto de todas las cadenas tales que el décimo símbolo a partir del extremo derecho sea 1.

\***2.6** Describa en español los conjuntos aceptados por los autómatas finitos cuyo diagrama de transiciones se dan en la Fig. 2.29 (a)-(c).

\***S 2.7** Pruebe que el FA cuyo diagrama de transiciones se da en la Fig. 2.30 acepta al conjunto de todas las cadenas sobre el alfabeto  $\{0, 1\}$  con un número igual de 0s y 1s, tales que cada prefijo tenga cuando mucho un 0 más que 1s y cuando mucho 1 más que 0s.

**2.8** Dé autómatas finitos no determinísticos que acepten a los siguientes lenguajes.

a) El conjunto de cadenas de  $(0+1)^*$  tal que algunos dos 0s están separados por una cadena cuya longitud es  $4i$ , para alguna  $i \geq 0$ .

b) El conjunto de todas las cadenas sobre el alfabeto  $\{a, b, c\}$  que tengan el mismo valor cuando se evalúan de izquierda a derecha y de derecha a izquierda mediante la multiplicación que se da en la tabla de la Fig. 2.31.

c) El conjunto de todas las cadenas de 0s y 1s tales que el décimo símbolo a partir del extremo derecho sea un 1. ¿Cómo compara su respuesta con el DFA del Problema 2.5 (e)?

**2.9** Constrúyase un DFA equivalente al NFA dado.

a)  $(\{p, q, r, s\}, \{0, 1\}, \delta_1, p, \{s\})$ . b)  $(\{p, q, r, s\}, \{0, 1\}, \delta_2, p, \{q, s\})$   
en donde  $\delta_1$  y  $\delta_2$  están dadas en la Fig. 2.32.

**2.10** Escriba expresiones regulares para cada uno de los lenguajes siguientes sobre el alfabeto  $\{0, 1\}$ . Porporcione una justificación de la corrección de su expresión.

\*a) El conjunto de todas las cadenas que contengan cuando mucho un par de 0s consecutivos y cuando mucho un par de 1's, consecutivos.

b) El conjunto de todas las cadenas en las que cada par de 0's adyacentes aparecen antes de cualquier par de las adyacentes.

c) El conjunto de todas las cadenas que no contengan a 101 como subcadena.

\*d) El conjunto de todas las cadenas con un número igual de 0s y 1s tales que ningún prefijo tenga dos 0s más que 1s ni dos 1s más que 0s.

**2.11** Describa en español los conjuntos representados por las siguientes expresiones regulares.

a)  $(11 + 0)^*(00 + 1)^*$

b)  $(1 + 01 + 001)^*(\epsilon + 0 + 00)$

c)  $[00 + 11 + (01 + 10)(00 + 11)^*(01 + 10)]^*$

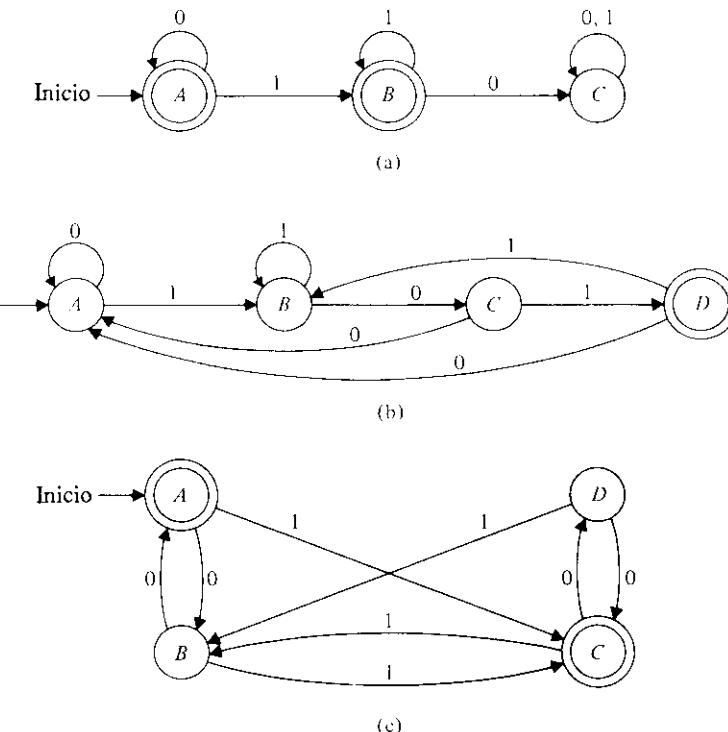


Fig. 2.29 Diagrama de transiciones para los autómatas finitos.

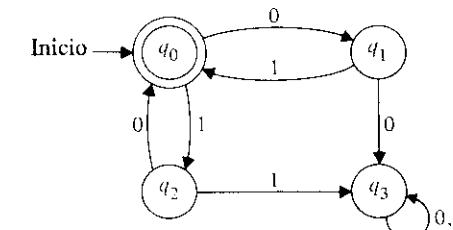


Fig. 2.30 Diagrama de transiciones.

	a	b	c
a	a	a	c
b	c	a	b
c	b	c	a

Fig. 2.31 Tabla de multiplicación no asociativa.

	0	1
p	p, q	p
q	r	r
r	s	—
s	s	s

$\delta_1$

	0	1
p	q, s	q
q	r	q, r
r	s	p
s	—	p

$\delta_2$

Fig. 2.32 Dos funciones de transición.

**2.12** Construya autómatas finitos equivalentes a las siguientes expresiones regulares.

- a)  $10 + (0+11)0^*$
- b)  $01 [(10)^* + 111]^* + 0]1^*$
- c)  $((0+1)(0+1))^* + ((0+1)(0+1)(0+1))^*$

**2.13** Construya expresiones regulares que correspondan a los diagramas de estado que se dan en la Fig. 2.33.

**2.14** Utilice las ideas que se dan en la demostración del Teorema 2.4 para construir algoritmos para los siguientes problemas.

- a) Encontrar la trayectoria de menor costo entre dos vértices de un grafo dirigido en el que cada borde está etiquetado con un costo no negativo.
- b) Determíñese el número de cadenas de longitud  $n$  aceptadas por un FA.
- c) Constrúyase un NFA equivalente al 2DFA  $\{(q_0, \dots, q_5), \{0, 1\}, \delta, q_0, \{q_2\}\}$ , en el que está  $\delta$  representada en la Fig. 2.34.

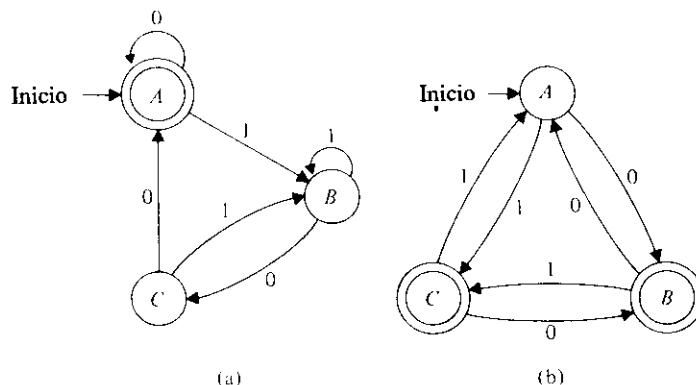


Fig. 2.33 Diagramas de transiciones.

	0	1
$q_0$	$(q_0, R)$	$(q_1, R)$
$q_1$	$(q_1, R)$	$(q_2, R)$
$q_2$	$(q_2, R)$	$(q_3, L)$
$q_3$	$(q_4, L)$	$(q_3, L)$
$q_4$	$(q_0, R)$	$(q_4, L)$

Fig. 2.34 Una función de transición para un 2DFA.

**2.16** Pruebe las siguientes identidades para las expresiones regulares  $r, s$  y  $t$ . En ese caso,  $r = s$  significa  $L(r) = L(s)$ .

- a)  $r + s = s + r$
- b)  $(r + s) + t = r + (s + t)$
- c)  $(rs)t = r(st)$
- d)  $(r(s+t)) = rs+rt$
- e)  $(r+s)t = rt+st$
- f)  $\emptyset^* = \emptyset$
- g)  $(r^*)^* = r^*$
- h)  $(\epsilon + r)^* = r^*$
- i)  $(r^*s^*)^* = (r+s)^*$

**2.17** Pruebe la validez o no validez de las siguientes expresiones regulares  $r, s$  y  $t$ .

- a)  $(rs+r)^*r = r(sr+r)^*$
- b)  $s(rs+s)^*r = rr^*s(rr^*)^*$
- c)  $(r+s)^* = r^* + s^*$

**2.18** Un autómata finito no determinístico de dos direcciones (2NFA) se define de la misma manera que el 2DFA, excepto que el 2NFA posee un conjunto de movimientos posibles para cada estado y símbolo de entrada. Pruebe que el conjunto aceptado por cualquier 2NFA es regular. [Sugerencia: La observación que se hizo en la demostración del Teorema 2.5 sobre el hecho de que ningún estado puede repetirse siguiendo la misma dirección en una secuencia cruzada ya no es válida. Sin embargo, para cada entrada aceptada podemos considerar el cálculo más corto que conduzca a la aceptación.]

**2.19** Muestre que al añadir la capacidad del 2NFA para mantener su cabeza estacionaria (y cambiar de estado) en un movimiento no incrementa el tipo de lenguajes aceptados por un 2NFA.

**\*2.20** Un 2NFA con *señaladores de extremo* es un 2NFA con símbolos especiales  $\epsilon$  y  $\$$  que marcan los extremos izquierdo y derecho de la entrada. Decimos que una entrada  $x$ , que no contiene los símbolos  $\epsilon$  o  $\$$ , es aceptada si el 2NFA iniciado con  $\epsilon x \$$  en su cinta y con la cabeza de la cinta que lee el símbolo  $\epsilon$  entra en un estado de aceptación en cualquier lugar de la entrada. Muestre que el 2NFA con señaladores de extremo acepta sólo conjuntos regulares.

**2.21** Considérese un 2DFA  $M = (Q, \Sigma, \delta, q_0, F)$ . Para cada cadena  $x$  constrúyase una transformación  $f$  de  $Q$  a  $Q \cup \{\epsilon\}$ , en donde  $f(q) = p$  si el 2DFA, iniciado en el símbolo que se encuentra más a la derecha de  $x$ , finalmente se mueve fuera de  $x$  hacia la derecha, en el estado  $p$ .  $f(q) = \epsilon$  significa que el 2DFA, cuando se inicia en el símbolo que está más a la derecha de  $x$ , nunca sale de ésta o la abandona por el extremo izquierdo. Constrúyase un DFA que simule a  $M$  mediante el almacenamiento en su control finito de una tabla  $f$  en lugar de una secuencia cruzada.

**\*\*2.22** Sean  $r$  y  $s$  expresiones regulares. Considérese la ecuación  $X = rX + s$ , en donde  $rX$  representa la concatenación de  $r$  y  $X$ , y  $+$  es la unión. Bajo la suposición de que el conjunto representado por  $r$  no contiene a  $\epsilon$ , encuentre la solución para  $X$  y pruebe que ésta es única. Cuál es la solución si  $L(r)$  contiene a  $\epsilon$ ?

**\*\*2.23** Se puede construir una expresión regular a partir de un autómata finito mediante la solución de un conjunto de ecuaciones lineales de la forma en donde  $a_{ij}$  y  $c_i$  son conjuntos

$$\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix},$$

de cadenas representadas por expresiones regulares,  $+$  representa la unión y la multiplicación significa concatenación. Dé un algoritmo para resolver tales ecuaciones.

#### 2.24 Proporciónense máquinas de Mealy y Moore para los procesos siguientes:

- Para una entrada de  $(0 + 1)^*$ , si la entrada termina en 101, salida A; si la entrada termina en 110, salida B; en cualquier otro caso salida C.
- Para una entrada de  $(0 + 1 + 2)^*$  imprima el residuo módulo 5 de la entrada tratada como un número ternario (base 3, con dígitos 0, 1, 2).

#### Soluciones a los ejercicios seleccionados

**2.1** Nótese que la salida en la compuerta  $y_1$  afecta a la salida en la compuerta  $y_2$  y viceversa. Supondremos valores para  $y_1$  y  $y_2$  y los utilizaremos para calcular nuevos valores. Entonces repetiremos el proceso con los valores nuevos hasta que alcancemos un estado estable del sistema. En la Fig. 2.35 hemos tabulado los valores estables de  $y_1$  y  $y_2$  para cada valor posible supuesto de  $y_1$  y  $y_2$  para valores de entrada 0 y 1.

	Entrada	
$y_1, y_2$	0	1
00	00	01
01	11	01
11	11	10
10	00	10

(a)

	0	1
$q_0$	$q_0$	$q_1$
$q_1$	$q_2$	$q_1$
$q_2$	$q_2$	$q_3$
$q_3$	$q_0$	$q_3$

(b)

Fig. 2.35 Transiciones en un circuito de interrupción.

Si  $y_1$  y  $y_2$  se suponen ambas con valor 0, entonces las compuertas A y B tienen salida 0 y la compuerta C tiene salida igual al valor de la entrada  $x$ . Como ambas entradas a la compuerta D son 0, la salida en la compuerta D es 0. La salida en la compuerta E tiene el valor de la entrada  $x$ . Así pues, el primer renglón de la Fig. 2.35 (a) tiene entradas 00 y M01. Las entradas restantes se calculan de forma similar.

Podemos modelar el circuito mediante la asignación de un estado a cada par de valores de  $y_1$  y  $y_2$ . Esto se hace en la Fig. 2.35(b). Puesto que  $y_1 = y_2 = 1$  produce una salida 1,  $q_2$  es un estado final. Puede verse que el circuito registra la paridad de pulsos (entradas 1) y produce un pulso de salida para cada pulso de entrada con número impar.

**2.7** Se nos pide que probemos que un conjunto, descrito de manera informal en español, es el conjunto aceptado por el FA. Es claro que no podemos dar una demostración formal completa. Podemos argumentar, de forma intuitiva, que alguna descripción formal del conjunto es equivalente a la descripción en español y entonces proceder formalmente, o, de otra manera, simplemente dar una prueba informal. Escogemos esta última.

La prueba consiste en deducir las propiedades de las cadenas, llevando al autómata a cada uno de los cuatro estados, y entonces probar por inducción sobre la longitud de una cadena que nuestra interpretación es correcta.

Decimos que una cadena  $x$  es *propia* si cada prefijo de  $x$  tiene cuando mucho un 0 más que 1s y cuando más un 1 más que 0s. Argumentamos por inducción sobre la longitud de una cadena  $x$  que

- $\delta(q_0, x) = q_0$  si y sólo si  $x$  es propia y contiene un número igual de 0s y 1s.
- $\delta(q_0, x) = q_1$  si y sólo si  $x$  es propia y contiene un 0 más que 1s.
- $\delta(q_0, x) = q_2$  si y sólo si  $x$  es propia y contiene un 1 más que 0s.
- $\delta(q_0, x) = q_3$  si y sólo si  $x$  no es propia.

Obsérvese que la hipótesis de inducción es más fuerte que el teorema deseado. Las condiciones (2), (3) y (4) se agregan para permitir que la inducción continúe.

Probaremos las proposiciones «si» de los incisos (1) a (4) primero. La base de la inducción,  $|x| = 0$ , se concluye porque la cadena vacía tiene un número igual de 0s y 1s y  $\delta(q_0, \epsilon) = q_0$ .

Supóngase que la hipótesis de inducción es verdadera para toda  $x$ ,  $|x| < n$ ,  $n \geq 1$ . Considérese una cadena  $y$  de longitud  $n$ , tal que  $y$  es propia y tiene el mismo número de 0s y 1s. Tómese en cuenta, primero, el caso en que  $y$  termine en 0. Entonces  $y = x0$ , en donde  $x$  es propia y tiene un 1 más que 0s. Por tanto,  $\delta(q_0, x) = q_2$ . En consecuencia,

$$\delta(q_0, y) = \delta(q_0, x0) = \delta(q_2, 0) = q_0.$$

El caso en que  $y$  termine en un 1 se trata de manera análoga.

En seguida considérese una cadena  $y$ ,  $|y| = n$  tal que  $y$  es propia y tiene un 0 más que 1s. Si  $y = x0$ , entonces  $x$  tiene dos 0s más que 1s, lo que contradice el hecho de que  $y$  es propia. Así pues  $y = x1$ , en donde  $x$  es propia y tiene un número igual de 0s y 1s. Por la hipótesis de inducción,  $\delta(q_0, x) = q_0$ ; por tanto,  $\delta(q_0, y) = q_1$ .

La situación en la que  $y$  es propia y tiene un 1 más que 0s, y la situación en la que  $y$  no es propia se tratan de manera similar.

Debemos, ahora, mostrar que las cadenas que alcanzan cada estado tienen la interpretación dada en (1) a (4). Supóngase que  $\delta(q_0, y) = q_1$  y  $|y| \geq 1$ . Si  $y = x0$ , entonces  $\delta(q_0, x) = q_2$ , puesto que  $q_2$  es el único estado con una transición 0 al estado  $q_0$ . Así pues, por la hipótesis de inducción  $x$  es propia y tiene un número igual de 0s y 1s. El caso en el que  $y$  termina en un 1 es similar, como también lo son los casos  $\delta(q_0, y) = q_1, q_2 \circ q_3$ .

#### NOTAS BIBLIOGRAFICAS

El estudio formal original de los sistemas de estado finito (redes neuronales parecidas a la que aparece en el Ejercicio 2.2) fue hecho por McCulloch y Pitts [1943]. Kleene [1956] tomó en cuenta las expresiones regulares y modeló las redes de neuronas de McCulloch y Pitts mediante autómatas finitos, probando así la equivalencia entre los dos conceptos. Aproximadamente en esa misma época Huffman [1954], Moore [1956] y Mealy [1955], estudiaron modelos similares, siendo los dos últimos autores quienes dieron nombre a los términos "máquina de Moore" y "Máquina de Mealy". Los autómatas finitos no determinísticos fueron introducidos por Rabin y Scott [1959], quienes probaron su equivalencia con los autómatas determinísticos. El concepto de autómata finito de dos direcciones y su equivalencia con la variedad en una sola dirección se debió al trabajo independiente de Rabin y Scott [1959] y Shepherdson [1959].

La demostración de la equivalencia de las expresiones regulares y los autómatas finitos, como se presentó en este libro (a través de los NFAs con transiciones  $\epsilon$ ) fue modelada a partir

de McNaughton y Yamada [1960]. Brzozowski [1962, 1964] desarrolló la teoría de las expresiones regulares. El hecho de que la solución única a  $X = rX + s$  (Ejercicio 2.22) es  $r^*s$  si  $L(R)$  no contiene  $\epsilon$  se conoce como lema de Arden [1960]. Floyd [1967] aplicó la idea del no determinismo a los programas. Salomaa [1966] dio axiomatizaciones de las expresiones regulares.

Las aplicaciones de los autómatas finitos a circuitos de interrupción pueden encontrarse en Kohavi [1970] y Friedman [1975]. El uso de la teoría para diseñar analizadores de léxico es tratado en Johnson *et al.* [1968] y Lesk [1975]. Otros usos de la teoría de los autómatas finitos en el diseño de editores de texto y otros programas de procesamiento de textos se discuten en Thompson [1968], Bullen y Millen [1972], Aho y Corasick [1975], Knuth, Morris y Pratt [1977] y Aho y Ullman [1977].

Algunos trabajos más que tratan sobre autómatas finitos son los de Arbib [1970], Conway [1971], Minsky [1967], Moore [1964] y Shannon y McCarthy [1956].

## CAPÍTULO

# 3

## PROPIEDADES DE LOS CONJUNTOS REGULARES

Existen varias preguntas que uno puede hacerse relativas a los conjuntos regulares. Una pregunta importante es: dado un lenguaje  $L$  especificado de alguna manera, ¿es  $L$  un conjunto regular? También queríamos saber cuándo los conjuntos regulares representados por diferentes expresiones regulares son los mismos o encontrar el autómata finito con el menor número de estados que denote al mismo lenguaje como un FA dado.

En este capítulo proporcionamos herramientas para tratar cuestiones como éstas, que conciernen a los conjuntos regulares. Demostramos un "lema de sondeo" para mostrar que ciertos lenguajes son *no* regulares. Damos "propiedades de cerradura" de los conjuntos regulares; al hecho de que los lenguajes construidos a partir de conjuntos regulares, de ciertas maneras específicas, también deben ser regulares y pueden utilizarse para probar o refutar algunos otros lenguajes como regulares. La cuestión de la regularidad o no regularidad puede resolverse, en ocasiones con la ayuda del Teorema de Myhill-Nerode, que se da en la Sec. 3.4. Además, damos algoritmos para contestar un cierto número de otras cuestiones sobre las expresiones regulares y autómatas finitos tales como si un FA dado acepta a un lenguaje infinito.

### 3.1 EL LEMA DE SONDEO PARA CONJUNTOS REGULARES

En esta sección demostramos un resultado básico, conocido como el *lema de sondeo*, que es una poderosa herramienta para demostrar que ciertos lenguajes son *no* regulares. También resulta útil en el desarrollo de algoritmos para responder a ciertas cuestiones

relacionadas con los autómatas finitos, como la que se refiere a que si el lenguaje aceptado por un FA es finito o infinito.

Si un lenguaje es regular, entonces es aceptado por un DFA  $M = (Q, \Sigma, \delta, q_0, F)$  que contiene un número determinado de estados, digamos  $n$ . Considerese una entrada de  $n$  o más símbolos  $a_1 a_2 \dots a_m$ ,  $m \leq n$ , y para  $i = 1, 2, \dots, m$  sea  $\delta(q_0, a_1 a_2 \dots a_i) = q_i$ . No es posible que cada uno de los  $n+1$  estados  $q_0, q_1, \dots, q_n$  sean distintos, puesto que sólo existen  $n$  estados diferentes. Por tanto existen dos enteros  $j$  y  $k$ ,  $0 \leq j < k \leq n$ , tales que  $q_j = q_k$ . La trayectoria con etiqueta  $a_1 a_2 \dots a_m$  en el diagrama de transiciones de  $M$  se ilustra en la Fig. 3.1. Como  $j < k$ , la cadena  $a_{j+1} \dots a_k$  tiene una longitud de por lo menos 1, y ya que  $k \leq n$ , su longitud no es mayor que  $n$ .

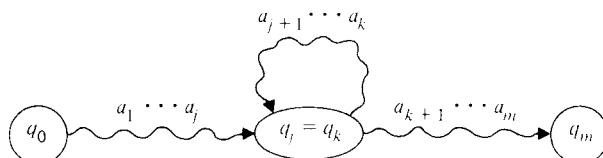


Fig. 3.1 Trayectoria en un diagrama de transiciones de un DFA  $M$ .

Si  $q_m$  está en  $F$ , es decir,  $a_1 a_2 \dots a_m$  está en  $L(M)$ , entonces  $a_1 a_2 \dots a_j a_{k+1} a_{k+2} \dots a_m$  está también en  $L(M)$ , ya que existe una trayectoria de  $q_0$  a  $q_m$  que pasa por  $q_j$  pero no alrededor de la trayectoria  $a_{j+1} \dots a_k$ . De manera formal, según el Ejercicio 2.4,

$$\begin{aligned}\delta(q_0, a_1 \dots a_j a_{k+1} \dots a_m) &= \delta(\delta(q_0, a_1 \dots a_j), a_{k+1} \dots a_m) \\ &= \delta(q_j, a_{k+1} \dots a_m) \\ &= \delta(q_k, a_{k+1} \dots a_m) \\ &= q_m.\end{aligned}$$

De manera similar, podemos ir alrededor del ciclo de la Fig. 3.1 más de una vez: de hecho, tantas veces como se desee. Así pues,  $a_1 \dots a_j (a_{j+1} \dots a_k)^i a_{k+1} \dots a_m$  está en  $L(M)$  para cualquier  $i \geq 0$ . Lo que hemos probado es que dada una cadena lo suficientemente larga, aceptada por un FA, podemos encontrar una subcadena cerca del principio de la cadena que puede ser "sondeada", es decir, repetida tantas veces como se quiera, y la cadena resultante será aceptada por el FA. El planteamiento formal del lema de sondeo lo exponemos a continuación.

**Lema 3.1** Sea  $L$  un conjunto regular. Entonces existe una constante  $n$  tal que si  $z$  es cualquier palabra de  $L$  y  $|z| \geq n$ , podemos escribir  $z = uvw$  de tal modo que  $|uv| \leq n$ ,  $|v| \geq 1$  y para toda  $i \geq 0$ ,  $uv^i w$  está en  $L$ . Aún más,  $n$  no es mayor que el número de estados del FA más pequeño que acepta a  $L$ .

**Demostración** Véase la discusión que precedió al planteamiento del lema. En ésta,  $z$  es  $a_1 a_2 \dots a_m$ ,  $u = a_1 a_2 \dots a_j$ ,  $v = a_{j+1} \dots a_k$  y  $w = a_{k+1} \dots a_m$ .  $\square$

Nótese que el lema de sondeo establece que si un conjunto regular contiene una cadena larga  $z$ , entonces contiene un conjunto infinito de cadenas de la forma  $uv^i w$ . El lema no establece que cada cadena, con la longitud suficiente, de un conjunto regular sea de la forma  $uv^i w$  para alguna  $i$  grande. De hecho,  $(0 + 1)^*$  contiene cadenas arbitrariamente largas en las que no aparecen subcadenas tres veces consecutivas. (La prueba se deja como ejercicio).

### Aplicaciones del lema de sondeo

El lema de sondeo es extremadamente útil en la demostración de que ciertos conjuntos no son regulares. La metodología general en esta aplicación consiste en un "argumento adversario" que se da en la forma siguiente.

1. Seleccione el lenguaje  $L$  que usted desea probar que no es regular.
2. El "adversario" toma  $n$ , la constante mencionada en el lema de sondeo. Se debe esperar, en lo que sigue, que el adversario escoja cualquier entero finito  $n$ , pero una vez que lo ha hecho,  $n$  no puede cambiarse.
3. Seleccione una cadena  $z$  de  $L$ . Su decisión puede depender implícitamente del valor de  $n$  escogida en (2).
4. El adversario descompone  $z$  en  $u$ ,  $v$  y  $w$ , sujeto a las siguientes limitaciones:  $|uv| \leq n$  y  $|v| \geq 1$ .
5. Se llega a una contradicción del lema de sondeo al demostrar que, para  $u$ ,  $v$  y  $w$  determinadas por el adversario, existe una  $i$  para la cual  $uv^i w$  no está en  $L$ . Se puede concluir entonces que  $L$  no es regular. La selección que haga de  $i$  puede depender de  $n$ ,  $u$ ,  $v$  y  $w$ .

Es interesante notar que la elección del lector en el "juego" anterior corresponde a los cuantificadores universales ( $\forall$  o "para todo") y las elecciones hechas por el ("adversario" corresponden a los cuantificadores existenciales ( $\exists$  o "existe") del planteamiento formal del lema de sondeo:

$$\forall L (\exists n) (\forall z) [z \text{ en } L \text{ y } |z| \geq n \text{ implica que}$$

$$(\exists u, v, w) (z = uvw, |uv| \leq n, |v| \geq 1 \text{ y } (\forall i) (uv^i w \text{ es en } L)).$$

**Ejemplo 3.1** El conjunto  $L = \{0^{i^2} \mid i \text{ es un entero}, i \geq 1\}$ , que consiste en todas las cadenas de 0s cuya longitud es un cuadrado perfecto; no es regular. Supóngase que  $L$  es regular y sea  $n$  el entero del lema de sondeo. Sea  $z = 0^{n^2}$ . Según el lema de sondeo,  $0^{n^2}$  puede escribirse como  $uvw$ , en donde  $1 \leq |v| \leq n$  y  $uv^i w$  está en  $L$  para toda  $i$ . En particular, sea  $i = 2$ . Sin embargo,  $n^2 < |uv^2 w| \leq n^2 + n < (n+1)^2$ . Es decir, la longitud de  $uv^2 w$  se encuentra apropiadamente entre  $n^2$  y  $(n+1)^2$ , y por tanto no es un cuadrado perfecto. Así pues,  $uv^2 w$  no se encuentra en  $L$ , lo que significa una contradicción. Concluimos que  $L$  es no regular.

**Ejemplo 3.2** Sea  $L$  un conjunto de cadenas de 0s y 1s, que comienzan con un 1, cuyo valor considerado como número binario es primo. Haremos uso del teorema de

sondeo para demostrar que  $L$  no es regular. Necesitamos dos resultados de la teoría de los números. El primero consiste en que el número de primos es infinito y que existen, en consecuencia, primos arbitrariamente grandes. El segundo, debido a Fermat, es que  $2^{p-1} - 1$  es divisible por  $p$  para cualquier número primo  $p > 2$ . Dicho de otra manera,  $2^{p-1} \equiv 1 \pmod p$  (véase Hardy y Wright [1938]).

Supóngase que  $L$  es regular, y sea  $n$  el entero del lema de sondeo. Sea  $z$  la representación binaria de un primo  $p$  tal que  $p > 2^n$ . Un primo con estas características existe puesto que existe un número infinito de primos. Por el lema de sondeo podemos escribir  $z = uvw$ , en donde  $|v| \geq 1$  y  $uvw$  es la representación binaria de un primo para toda  $i$ . Sea  $n_u$ ,  $n_v$  y  $n_w$  los valores de  $u$ ,  $v$  y  $w$  considerados como números binarios. Si  $u$  o  $w$  son  $\epsilon$ , entonces  $n_u$  o  $n_w$ , respectivamente, es 0. Escoja  $i = p$ . Entonces  $uv^p w$  es la representación binaria de un primo  $q$ . El valor numérico de  $q$  es

$$n_u 2^{|w|+p|v|} + n_v 2^{|w|} (1 + 2^{|v|} + \dots + 2^{(p-1)|v|}) + n_w.$$

Por el teorema de Fermat,  $2^{p-1} \equiv 1 \pmod p$ . Si elevamos ambos lados a la potencia  $|v|$ , tendremos  $2^{(p-1)|v|} \equiv 1 \pmod p$ . Por tanto

$$2^{p|v|} = 2^{(p-1)|v|} 2^{|v|} \equiv 2^{|v|} \pmod p.$$

Sea  $s = 1 + 2^{|v|} + \dots + 2^{(p-1)|v|}$ . Entonces

$$(2^{|v|} - 1)s = 2^{p|v|} - 1,$$

que es  $2^{|v|} - 1 \pmod p$ . En consecuencia,  $(2^{|v|} - 1)(s - 1)$  es divisible entre  $p$ . Pero  $1 \leq |v| \leq n$ , de modo que  $2 \leq 2^{|v|} \leq 2^n < p$ . Por tanto  $p$  no puede dividir a  $2^{|v|} - 1$ , así que divide a  $s - 1$ . Es decir,  $s \equiv 1 \pmod p$ . Pero,

$$q = n_u 2^{|w|+p|v|} + n_v 2^{|w|} s + n_w,$$

de modo que

$$q \equiv n_u 2^{|w|+p|v|} + n_v 2^{|w|} + n_w \pmod p. \quad (3.1)$$

Pero el lado derecho de (3.1) es el valor numérico de  $p$ . Así pues  $q \equiv p \pmod p$ , es decir,  $q$  es divisible entre  $p$ . Como  $q > p > 1$ ,  $q$  no puede ser primo. Pero según el lema de sondeo, la representación binaria de  $q$  está en  $L$ , lo que es una contradicción. Concluimos que  $L$  no es regular.

## 3.2 PROPIEDADES DE CERRADURA PARA CONJUNTOS REGULARES

Existen muchas operaciones sobre lenguajes que conservan a los conjuntos regulares, en el sentido de que las operaciones aplicadas a los conjuntos regulares dan como resultado conjuntos regulares. Por ejemplo, la unión de dos conjuntos regulares es un conjunto regular, ya que si  $r_1$  y  $r_2$  son expresiones regulares que representan a los conjuntos regulares  $L_1$  y  $L_2$ , entonces  $r_1 + r_2$  representan  $L_1 \cup L_2$ , de forma que  $L_1 \cup L_2$  también es regular. De forma similar, la concatenación de conjuntos regulares es un conjunto regular y la cerradura de Kleene de un conjunto regular es regular.

Si una clase de lenguajes es cerrada bajo una operación en particular, llamamos a ese hecho una *propiedad de cerradura* de la clase de lenguajes. Estamos particularmente interesados en las *propiedades de cerradura efectivas* en las que, dados los descriptores para los lenguajes de la clase, existe un algoritmo para construir una representación del lenguaje que sea producto de la aplicación de la operación a esos lenguajes. Por ejemplo, acabamos de dar un algoritmo para construir una expresión regular para la unión de dos lenguajes representados por expresiones regulares, de modo que la clase de conjuntos regulares efectivamente es cerrada con respecto a la unión. Las propiedades de cerradura que se dan en este libro son efectivas, a menos que se indique lo contrario.

Deberá observarse que las equivalencias, que se mostraron en el Capítulo 2 entre los diferentes modelos de autómatas finitos y expresiones regulares son equivalencias efectivas, en el sentido de que los algoritmos fueron dados para pasar de una representación a otra. Por tanto, al probar propiedades de cerradura efectivas podemos escoger la representación que más nos convenga, por lo general expresiones regulares o autómatas finitos determinísticos. Consideraremos ahora una secuencia de propiedades de cerradura de conjuntos regulares; en los ejercicios se dan otras propiedades de cerradura complementarias.

**Teorema 3.1** Los conjuntos regulares son cerrados con respecto a la unión, concatenación y cerradura de Kleene.

**Demostración** La demostración se concluye de forma inmediata a partir de la definición de expresión regular.  $\square$

### Operaciones booleanas

**Teorema 3.2** La clase de conjuntos regulares es cerrada bajo la complementación. Es decir, si  $L$  es un conjunto regular y  $L \subseteq \Sigma^*$ , entonces  $\Sigma^* - L$  es un conjunto regular.

**Demostración** Sea  $L \subseteq L(M)$  para el DFA  $M = (Q, \Sigma_1, \delta, q_0, F)$  y  $L \subseteq \Sigma^*$ . Primero, podemos suponer  $\Sigma_1 = \Sigma$ , puesto que si existen símbolos de  $\Sigma_1$  que no están en  $\Sigma$ , podemos eliminar las transiciones de  $M$  sobre los símbolos que no están en  $\Sigma$ . El hecho de que  $L \subseteq \Sigma$  nos asegura que no cambiemos el lenguaje de  $M$ . Si existen símbolos de  $\Sigma$  que no están en  $\Sigma_1$ , entonces ninguno de tales símbolos aparecen en las palabras de  $L$ . Podemos introducir, por tanto, un “estado muerto”  $d$  en  $M$  con  $\delta(d, a) = d$  para toda  $a$  en  $\Sigma$  y  $\delta(q, a) = d$  para toda  $q$  en  $Q$  y  $a$  en  $\Sigma - \Sigma_1$ .

Ahora, para aceptar a  $\Sigma^* - L$ , completemos los estados finales de  $M$ . Esto es, sea  $M' = (Q, \Sigma, \delta, q_0, Q - F)$ . Entonces  $M'$  acepta una palabra  $w$  si y sólo si  $\delta(q_0, w)$  está en  $Q - F$ , es decir,  $w$  está en  $\Sigma^* - L$ . Nótese que es muy importante para la prueba que  $M$  sea determinístico y que no tenga movimientos  $\epsilon$ .  $\square$

**Teorema 3.3** Los conjuntos regulares son cerrados con respecto a la intersección.

**Demostración**  $L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}$ , en donde la barra denota complemento con respecto a un alfabeto que incluya a los alfabetos de  $L_1$  y  $L_2$ . La cerradura con respecto a la intersección se concluye a partir de la cerradura con respecto a la unión y la complementación.  $\square$

Es valioso hacer notar que existe una construcción directa de un DFA para la intersección de dos conjuntos regulares. La construcción implica el tomar el producto cartesiano de estados y la delineamos de la forma siguiente:

Sean  $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$  y  $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$  dos autómatas finitos determinísticos. Sea

$$M = (Q_1 \times Q_2, \Sigma, \delta, [q_1, q_2], F_1 \times F_2),$$

en donde para toda  $p_1$  en  $Q_1$ ,  $p_2$  en  $Q_2$  y  $a$  en  $\Sigma$ ,

$$\delta([p_1, p_2], a) = [\delta_1(p_1, a), \delta_2(p_2, a)].$$

Se puede mostrar de manera sencilla que  $T(M) = T(M_1) \cap T(M_2)$ .

### Sustituciones y homomorfismos

La clase de conjuntos regulares posee la interesante propiedad de ser cerrada con respecto a la sustitución en el sentido siguiente. Para cada símbolo  $a$  del alfabeto de algún conjunto regular  $R$ , sea  $R_a$  un conjunto regular particular. Supóngase que reemplazamos en cada palabra  $w_1 w_2 \dots w_n$  de  $R$  por el conjunto de palabras de la forma  $w'_1 w'_2 \dots w'_n$ , en donde  $w'_i$  es una palabra cualquiera de  $R_{w_i}$ . El resultado, entonces, es siempre un conjunto regular. De manera más formal, una *sustitución*  $f$  es una transformación de un alfabeto  $\Sigma$  en subconjuntos de  $\Delta^*$ , para algún alfabeto  $\Delta$ . Por tanto  $f$  asocia un lenguaje con cada símbolo de  $\Sigma$ . La transformación  $f$  se extiende a cadenas de la manera siguiente:

1.  $f(\epsilon) = \epsilon$ ;
2.  $f(xa) = f(x)f(a)$ .

La transformación  $f$  se extiende a lenguajes mediante la definición

$$f(L) = \bigcup_{x \in L} f(x).$$

**Ejemplo 3.3** Sean  $f(0) = a$  y  $f(1) = b^*$ . Esto es,  $f(0)$  es el lenguaje  $\{a\}$  y  $f(1)$  es el lenguaje que consiste en todas las cadenas de  $b$ s. Entonces  $f(010)$  es el conjunto regular  $ab^*a$ . Si  $L$  es el lenguaje  $0^*(0+1)1^*$ , entonces  $f(L)$  es  $a^*(a+b^*), (b^*)^* = a^*b^*$ .

**Teorema 3.4** La clase de conjuntos regulares es cerrada bajo la sustitución.

**Demostración** Sea  $R \subseteq \Sigma^*$  un conjunto regular y, para cada  $a$  de  $\Sigma$ , sea  $R_a \subseteq \Delta^*$  también un conjunto regular;  $f: \Sigma \rightarrow \Delta^*$  es la sustitución definida por  $f(a) = R_a$ . Selecciónense expresiones regulares que representen a  $R$  y a cada  $R_a$ . Reemplácese cada vez que se dé el símbolo  $a$  en la expresión regular para  $R$  por la expresión regular para  $R_a$ . Para demostrar que la expresión regular que resulta representa a  $f(R)$ , observe que la sustitución de una unión, producto o cerradura es la unión, producto o cerradura

de la sustitución. [Por tanto, por ejemplo,  $f(L_1 \cup L_2) = f(L_1) \cup f(L_2)$ .] Una simple inducción sobre el número de operadores en la expresión regular completa la prueba.

Nótese que en el ejemplo 3.3 calculamos  $f(L)$  tomando la expresión regular de  $L 0^*(1+0)1^*$  y sustituyendo  $a$  por  $0$  y  $b$  por  $1$ . El hecho de que la expresión regular resultante sea equivalente a la expresión regular más simple  $a^*b^*$  es mera coincidencia.

Un tipo de sustitución que es de especial interés es el homomorfismo. Un *homomorfismo*  $h$  es una sustitución tal que  $h(a)$  contiene una sola cadena para cada  $a$ . Por lo general tomamos  $h(a)$  como la cadena misma, más que el conjunto que contiene a tal cadena. Es de utilidad definir la *imagen homomórfica inversa* de un lenguaje  $L$  como

$$h^{-1}(L) = \{x \mid h(x) \text{ está en } L\}.$$

También utilizamos, para la cadena  $w$ :

$$h^{-1}(w) = \{x \mid h(x) = w\}.$$

**Ejemplo 3.4** Sean  $h(0) = aa$  y  $h(1) = aba$ . Entonces  $h(010) = aaabaaa$ . Si  $L_1$  es  $(01)^*$ , entonces  $h(L_1)$  es  $(aaaba)^*$ . Sea  $L_2 = (ab + ba)^*a$ . Entonces  $h^{-1}(L_2)$  solamente consiste en la cadena 1. Para ver esto, observe que una cadena de  $L_2$  que comience con  $b$  no puede ser  $h(x)$  para cualquier cadena  $x$  de 0s y 1s, puesto que  $h(0)$  y  $h(1)$  comienzan cada uno con una  $a$ . Así pues, si  $h^{-1}(w)$  es no vacía y  $w$  se encuentra en  $L_2$ , entonces  $w$  comienza con  $a$ . Ahora,  $w = a$ , en cuyo caso  $h^{-1}(w)$  seguro está vacío, o  $w$  es  $abw'$  para alguna  $w'$  en  $(ab + ba)^*a$ . Concluimos que cada palabra en  $h^{-1}(w)$  comienza con un 1, y como  $h(1) = aba$ ,  $w'$  debe comenzar con  $a$ . Si  $w' = a$ , tenemos  $w = aba$  y  $h^{-1}(w) = \{1\}$ . Sin embargo, si  $w' \neq a$ , entonces  $w' = abw''$  y, en consecuencia,  $w = ababw''$ . Pero ninguna cadena  $x$  de  $(0+1)^*$  tiene  $h(x)$  que comience  $abab$ . Por consiguiente, concluimos que  $h^{-1}(w)$  está vacío en este caso. Así pues, la única cadena de  $L_2$  que tiene una imagen inversa bajo  $h$  es  $aba$ , y por tanto  $h^{-1}(L_2) = \{1\}$ .

Obsérvese que  $h(h^{-1}(L_2)) = \{aba\} \neq L_2$ . Por el contrario, es fácil mostrar que  $h(h^{-1}(L)) \subseteq L$  y  $h^{-1}(h(L)) \supseteq L$  para cualquier lenguaje  $L$ .

**Teorema 3.5** La clase de conjuntos regulares es cerrada con respecto a los homomorfismos y los homomorfismos inversos.

**Demostración** La cerradura con respecto a los homomorfismos se sigue inmediatamente a partir de la cerradura con respecto a la sustitución, ya que cada homomorfismo es una sustitución, en la que  $h(a)$  tiene sólo un miembro.

Para mostrar la cerradura con respecto a los homomorfismos inversos, sean  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA que acepte a  $L$ , y  $h$  el homomorfismo de  $\Delta$  a  $\Sigma^*$ . Construimos un DFA  $M'$  que acepte a  $h^{-1}(L)$  mediante la lectura del símbolo  $a$  de  $\Delta$  y que simule a  $M$  sobre  $h(a)$ . Formalmente, sea  $M' = (Q, \Delta, \delta', q_0, F)$  y defínase  $\delta'(q, a)$ , para  $q$  en  $Q$  y  $a$  en  $\Delta$ , como  $\delta(q, h(a))$ . Nótese que  $h(a)$  puede ser una cadena larga o  $\epsilon$ , pero  $\delta'$  está definida, por extensión, sobre todas las cadenas. Es fácil mostrar, por inducción sobre  $|x|$ , que  $\delta'(q_0, x) = \delta(q_0, h(x))$ . Por consiguiente,  $M'$  acepta a  $x$  si y sólo si  $M$  acepta a  $h(x)$ . Esto es,  $L(M') = h^{-1}(L(M))$ .  $\square$

**Ejemplo 3.5** La importancia de los homomorfismos y los homomorfismos inversos radica en su capacidad de simplificar demostraciones. Sabemos, por ejemplo, que  $\{0^n 1^n \mid n \geq 1\}$  es no regular. De manera intuitiva,  $\{a^n b a^n \mid n \geq 1\}$  es no regular por las mismas razones. Es decir, si tuviéramos un FA  $M$  que acepte  $\{a^n b a^n \mid n \geq 1\}$ , podríamos aceptar  $\{0^n 1^n \mid n \geq 1\}$  mediante la simulación de  $M$  sobre la entrada  $a$  para cada 0. Cuando es visto el primer 1, simule  $M$  sobre  $ba$  y de aquí en adelante simule  $M$  sobre  $a$  para cada 1 visto. No obstante, para ser rigurosos es necesario demostrar formalmente que  $\{a^n b a^n \mid n \geq 1\}$  es no regular. Esto se hace mostrando que  $\{a^n b a^n \mid n \geq 1\}$  puede ser convertido a  $\{0^n 1^n \mid n \geq 1\}$  mediante el uso de operaciones que conserven la regularidad. Por tanto,  $\{a^n b a^n \mid n \geq 1\}$  no puede ser regular.

Sean  $h_1$  y  $h_2$  los homomorfismos

$$\begin{aligned} h_1(a) &= a, & h_2(a) &= 0, \\ h_1(b) &= ba, & h_2(b) &= 1, \\ h_1(c) &= a, & h_2(c) &= 1. \end{aligned}$$

Entonces

$$h_2(h_1^{-1}(\{a^n b a^n \mid n \geq 1\}) \cap a^* b c^*) = \{0^n 1^n \mid n \geq 1\}. \quad (3.2)$$

Esto es,  $h_1^{-1}(\{a^n b a^n \mid n \geq 1\})$  consiste en todas las cadenas de  $(a+c)^* b (a+c)^*$  tales que el número de símbolos que anteceden a  $b$  es uno mayor que el número de símbolos que están después de  $b$ . Por consiguiente,

$$h_1^{-1}(\{a^n b a^n \mid n \geq 1\}) \cap a^* b c^* = \{a^n b c^{n-1} \mid n \geq 1\}.$$

La ecuación (3.2) entonces se concluye de manera inmediata mediante la aplicación del homomorfismo  $h_2$ .

Si  $\{a^n b a^n \mid n \geq 1\}$  fuera regular, entonces, ya que, los homomorfismos, homomorfismos inversos y la intersección con un conjunto regular conservan todos la propiedad de ser regulares, se concluiría que  $\{0^n 1^n \mid n \geq 1\}$  es regular, lo que implica una contradicción.

### Cocientes de lenguajes

Fijemos ahora nuestra atención a la última propiedad de cerradura de los conjuntos regulares, que se demostrará en esta sección. Un cierto número más de propiedades de cerradura se darán en los ejercicios. Defínase el *cociente* de los lenguajes  $L_1$  y  $L_2$ , que se escribe  $L_1 / L_2$ , como

$$\{x \mid \text{existe } y \text{ en } L_2 \text{ tal que } x y \text{ está en } L_1\}.$$

**Ejemplo 3.6** Sean  $L_1 = 0^* 10^*$  y  $L_2 = 10^* 1$ . Entonces  $L_1 / L_2$  es el vacío. Puesto que cada  $y$  de  $L_2$  tiene dos 1 y cada cadena  $x y$  que se encuentra en  $L_1$  puede tener sólo 1, no existe  $x$  tal que  $x y$  esté en  $L_1$  y  $y$  en  $L_2$ .

Sea  $L_3 = 0^* 1$ . Entonces  $L_1 / L_3$  es  $0^*$ , ya que para cualquier  $x$  de  $0^*$  podemos escoger  $y = 1$ . Es claro que  $x y$  se encuentra en  $L_1 = 0^* 10^*$  y  $y$  en  $L_3 = 0^* 1$ . Como las palabras de  $L_1$  y  $L_3$  tienen un 1, no es posible que las palabras que no estén  $0^*$  estén en  $L_1 / L_3$ .

Como otro ejemplo  $L_2 / L_3 = 10^*$ , ya que para cada  $x$  de  $10^*$  podemos escoger, de nuevo,  $y = 1$  de  $L_3$  y  $x y$  estará en  $L_2 = 10^* 1$ . Si  $x$  y está en  $L_2$  y  $y$  en  $L_3$ , entonces es evidente que  $x$  está en  $10^*$ .

**Teorema 3.6** La clase de conjuntos regulares es cerrada con respecto al cociente con conjuntos arbitrarios. †

**Demuestra** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un autómata finito que acepta a algún conjunto regular  $R$ , y sea  $L$  un lenguaje cualquiera. El cociente  $R/L$  es aceptado por un autómata finito  $M' = (Q, \Sigma, \delta, q_0, F')$ , que se comporta como  $M$  excepto que los estados finales de  $M'$  son todos estados  $q$  de  $M$  tales que existe  $y$  en  $L$  para el cual  $\delta(q, y)$  está en  $F$ . Entonces  $\delta(q_0, x)$  está en  $F'$  si y sólo si existe  $y$  tal que  $\delta(q_0, xy)$  está en  $F$ . Por tanto  $M'$  acepta a  $R/L$ . □

Se deberá observar que la construcción del Teorema 3.6 es diferente a todas las otras construcciones hechas en este capítulo, en el sentido de que no es efectiva. Puesto que  $L$  es un conjunto cualquiera, puede no existir ningún algoritmo que determine si existe  $y$  de  $L$  tal que  $\delta(q, y)$  esté en  $F$ . Aun si restringimos  $L$  a alguna clase representable de manera finita, podríamos aún no tener una construcción efectiva, a menos que exista un algoritmo para probar la existencia de dicha  $y$ . En efecto, estamos diciendo que para cualquier  $L$ , seguramente existe algún  $F'$  tal que  $M$ , con  $F'$  como el conjunto de estados finales, acepte a  $R/L$ . Sin embargo, podríamos no ser capaces de decir cuál subconjunto de  $Q$  deberá escogerse como  $F'$ . En la siguiente sección veremos que si  $L$  es un conjunto regular, podemos determinar  $F'$ , de modo que los conjuntos regulares estén efectivamente cerrados con respecto al cociente con un conjunto regular.

### 3.3 ALGORITMOS DE DECISION PARA CONJUNTOS REGULARES

Es importante tener algoritmos para responder a las diferentes cuestiones concernientes a los conjuntos regulares. Los tipos de preguntas que nos interesan ahora incluyen: ¿es un lenguaje dado vacío, finito o infinito? ¿Es un conjunto regular equivalente a otro?, etcétera. Antes de que podamos establecer la existencia de algoritmos para responder a tales cuestiones, debemos decidir sobre la representación. Para nuestros propósitos supondremos que los conjuntos regulares están representados por autómatas finitos. Podríamos también haber supuesto que los conjuntos regulares están representados por expresiones regulares o alguna otra notación, ya que existen traducciones mecánicas de estas notaciones a autómatas finitos. No obstante, uno puede imaginar representaciones para las cuales no existen tales algoritmos de traducción, y para dichas representaciones pueden no existir algoritmos para determinar si un lenguaje dado está vacío.

El lector, en esta etapa, puede sentir que es obvio que podamos determinar si un conjunto regular está vacío. Veremos en el Capítulo 8, sin embargo, que para muchas clases de lenguaje de interés la pregunta no puede contestarse.

† En este teorema la cerradura no es efectiva.

## Vacuidad, finitud e infinitud

Los algoritmos para determinar si un conjunto regular es el vacío, es finito o infinito puede estar basado en el siguiente teorema. Discutiremos los algoritmos eficientes después de presentar el teorema.

**Teorema 3.7** El conjunto de oraciones aceptadas por un autómata finito  $M$  que tiene  $n$  estados es:

1. no vacío si y sólo si el autómata finito acepta una oración de longitud menor a  $n$ .
2. infinito si y sólo si el autómata acepta alguna oración de longitud  $l$ , en donde  $n \leq l \leq 2^n$ .

Así pues existe un algoritmo para determinar si un autómata finito acepta un número finito o infinito de oraciones o no acepta ninguna.

### Demostración

1. La parte "si..." es obvia. Supóngase que  $M$  acepta a un conjunto no vacío. Sea  $w$  una palabra tan corta como cualquier palabra aceptada. Por el lema de sondeo,  $|w| < n$ , porque si  $w$  fuera la palabra más corta y  $|w| \geq n$ , entonces  $w = u v y$  y  $u y$  es más corta en el lenguaje.
2. Si  $w$  está en  $L(M)$  y  $n \leq |w| < 2n$ , entonces por el lema de sondeo,  $L(M)$  es infinito. Esto es,  $w = w_1 w_2 w_3$ , y para toda  $i$ ,  $w_i w_2 w_3$  está en  $L$ . De manera inversa, si  $L(M)$  es infinito, entonces existe  $w$  en  $L(M)$ , en donde  $|w| \geq n$ . Si  $|w| < 2n$ , hemos terminado. Si ninguna palabra tiene longitud entre  $n$  y  $2n - 1$ , sea  $w$  de longitud al menos de  $2n$ , pero tan corta como cualquier palabra de  $L(M)$  cuya longitud sea mayor o igual a  $2n$ . De nuevo por el lema de sondeo, podemos escribir  $w = w_1 w_2 w_3$  con  $1 \leq |w_2| \leq n$  y  $w_1 w_3$  en  $L(M)$ . Podemos estar en dos situaciones:  $w$  no es la palabra más corta de longitud  $2n$  o más; o  $|w_1 w_3|$  está entre  $n$  y  $2n - 1$ . En ambos casos se tienen una contradicción.

En la parte (1), el algoritmo para decidir si  $L(M)$  es un vacío es: "Véase si alguna palabra de longitud hasta  $n$  está en  $L(M)$ ." Está claro que existe un procedimiento que garantiza que se detendrá. En la parte (2), al algoritmo para decidir si  $L(M)$  es infinito es: "Véase si alguna palabra de longitud comprendida entre  $n$  y  $2n - 1$  está en  $L(M)$ ". De nuevo, es obvio que existe un procedimiento tal que garantiza que se detendrá.  $\square$

Debemos comprender que los algoritmos sugeridos en el Teorema 3.7 son altamente eficientes. Sin embargo, se puede probar con facilidad si un DFA acepta al conjunto vacío al tomar su diagrama de transiciones y eliminar todos los estados que no se pueden alcanzar sobre cualquier entrada desde el estado inicial. Si permanece uno o más estados finales, el lenguaje es no vacío. Entonces, sin cambiar el lenguaje aceptado, podemos eliminar todos los que no sean finales y de los cuales no se pueda alcanzar un estado final. El DFA acepta un lenguaje infinito si y sólo si el diagrama de transiciones resultante tiene un ciclo. El mismo método funciona para los NFAs, pero debemos verificar que existe un ciclo etiquetado con algo además de  $\epsilon$ .

## Equivalencia

Enseguida mostraremos que existe un algoritmo para determinar si dos autómatas finitos aceptan el mismo conjunto.

**Teorema 3.8** Existe un algoritmo para determinar si dos autómatas finitos son equivalentes (es decir, si aceptan el mismo lenguaje).

*Demostración* Sean  $M_1$  y  $M_2$  FA que aceptan a  $L_1$  y  $L_2$ , respectivamente. Según los teoremas 3.1, 3.2 y 3.3,  $(L_1 \cap L_2) \cup (\bar{L}_1 \cap \bar{L}_2)$  es aceptado por algún autómata finito,  $M_3$ . Es fácil ver que  $M_3$  acepta una palabra si y sólo si  $L_1 \neq L_2$ . De aquí que, por el teorema 3.7, exista un algoritmo para determinar si  $L_1 = L_2$ .  $\square$

## 3.4 TEOREMA DE MYHILL-NERODE Y MINIMIZACION DE AUTOMATAS FINITOS

Recuérdese de la Sección 1.5 nuestra discusión sobre relaciones de equivalencia y clases de equivalencia. Podemos asociar con un lenguaje cualquiera  $L$  una relación de equivalencia natural  $R_L$ ; es decir,  $x R_L y$  si y sólo si para cada  $z$ ,  $xz$  y  $yz$  están en  $L$  o no lo están. En el peor de los casos cada cadena está, por sí misma, en una clase de equivalencia, pero puede haber menos clases. En particular, el *índice* (número de clases de equivalencia) siempre es finito si  $L$  es un conjunto regular.

Existe también una relación de equivalencia natural sobre las cadenas asociadas con un autómata finito. Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA. Para  $x$  y  $y$  en  $\Sigma^*$  sea  $x R_M y$  si y sólo si  $\delta(q_0, x) = \delta(q_0, y)$ . La relación  $R_M$  es reflexiva, simétrica y transitiva, puesto que el símbolo "=" tiene tales propiedades, y por tanto  $R_M$  es una relación de equivalencia.  $R_M$  divide el conjunto  $\Sigma^*$  en clases de equivalencia, una por cada estado que se puede alcanzar desde  $q_0$ . Además, si  $x R_M y$ , entonces  $xz R_M yz$  para toda  $z$  en  $\Sigma^*$ , ya que, según el ejercicio 2.4,

$$\delta(q_0, xz) = \delta(\delta(q_0, x), z) = \delta(\delta(q_0, y), z) = \delta(q_0, yz).$$

Una relación de equivalencia  $R$  tal que  $x R y$  implique  $xz R yz$  se dice que es *invariante por la derecha (con respecto a la concatenación)*. Vemos que cada autómata finito induce una relación de equivalencia invariante por la derecha, definida como se definió  $R_M$  sobre las cadenas de entrada. Este resultado se formaliza en el siguiente teorema.

**Teorema 3.9** (*Teorema de Myhill-Nerode*) Las tres proposiciones siguientes son equivalentes:

1. El conjunto  $L \subseteq \Sigma^*$  es aceptado por algún autómata finito.
2.  $L$  es la unión de algunas clases de equivalencia de una relación de equivalencia invariante por la derecha de índice finito.
3. Sea  $R_L$  la relación de equivalencia definida por:  $x R_L y$  si y sólo si para toda  $z$  en  $\Sigma^*$ ,  $xz$  está en  $L$  exactamente cuando  $yz$  está en  $L$ . Entonces  $R_L$  es de índice finito.

**Demostración**

(1)  $\rightarrow$  (2) Supóngase que  $L$  es aceptado por algún DFA  $M = (Q, \Sigma, \delta, q_0, F)$ . Sea  $R_M$  la relación de equivalencia  $x R_M y$  si y sólo si  $\delta(q_0, x) = \delta(q_0, y)$ .  $R_M$  es invariante por la derecha ya que, para cualquier  $z$ , si  $\delta(q_0, x) = \delta(q_0, y)$ , entonces  $\delta(q_0, xz) = \delta(q_0, yz)$ . El índice de  $R_M$  es finito, puesto que el índice es cuando mucho el número de estados de  $Q$ . Aún más,  $L$  es la unión de aquellas clases de equivalencia que incluyen una cadena  $x$  tal que  $\delta(q_0, x)$  está en  $F$ , es decir, las clases de equivalencia correspondientes a los estados finales.

(2)  $\rightarrow$  (3) Mostramos que cualquier relación de equivalencia  $E$  que satisface (2) es un *refinamiento* de  $R_L$ ; esto es, cada clase de equivalencia de  $E$  está completamente contenida en alguna clase de equivalencia de  $R_L$ . Por consiguiente, el índice de  $R_L$  no puede ser mayor que el índice de  $E$  y, por tanto, es finito. Supóngase que  $x R_L y$  como  $E$  es invariante por la derecha, para cada  $z$  en  $\Sigma^*$ ,  $xz Eyz$ , y por tanto  $yz$  está en  $L$  si y sólo si  $xz$  está en  $L$ . Así pues,  $x R_L y$ , y en consecuencia, la clase de equivalencia de  $x$  en  $E$  está contenida en la clase de equivalencia de  $x$  en  $R_L$ . Concluimos que cada clase de equivalencia de  $E$  está contenida en alguna clase de equivalencia de  $R_L$ .

(3)  $\rightarrow$  (1) Primero debemos mostrar que  $R_L$  es invariante por la derecha. Supóngase que  $x R_L y$  y sea  $w$  una cadena en  $\Sigma^*$ . Debemos probar que  $xw R_L yw$ ; esto es, para cualquier  $z$ ,  $xwz$  está en  $L$  exactamente cuando  $ywz$  está en  $L$ . Pero como  $x R_L y$ , sabemos por la definición de  $R_L$  que para cualquier  $v$ ,  $xv$  está en  $L$  exactamente cuando  $yv$  está en  $L$ . Hacemos  $v = wz$  para probar que  $R_L$  es invariante por la derecha.

Ahora hagamos  $Q'$  el conjunto finito de clases de equivalencia de  $R_L$  y  $[x]$  el elemento de  $Q'$  que contiene a  $x$ . Definase  $\delta'([x], a) = [xa]$ . La definición es consistente, ya que  $R_L$  es invariante por la derecha. De haber escogido  $y$  en lugar de  $x$  de la clase de equivalencia  $[x]$ , hubiéramos obtenido  $\delta'([x], a) = [ya]$ . Pero  $x R_L y$ , así que  $xz$  está en  $L$  exactamente cuando  $yz$  está en  $L$ . En particular, si  $z = az'$ ,  $xaz'$  está en  $L$  exactamente cuando  $yaz'$  está en  $L$ , de modo que  $x R_L ya$  y  $[xa] = [ya]$ . Sea  $q'_0 = [\epsilon]$  y  $F' = \{[x] \mid x \text{ está en } L\}$ . El autómata finito  $M' = (Q', \Sigma, \delta', q'_0, F')$  acepta a  $L$ , puesto que  $\delta'(q'_0, x) = [x]$ , y por tanto  $x$  está en  $L$  ( $M'$ ) si y sólo si  $x$  está en  $F'$ .  $\square$

**Ejemplo 3.7** Sea  $L$  el lenguaje  $0 * 10 *$ .  $L$  es aceptado por el DFA  $M$  de la Fig. 3.2. Considérese la relación  $R_M$  definida por  $M$ . Como todos los estados son alcanzables desde el estado inicial,  $R_M$  tiene seis clases de equivalencia, que son

$$\begin{aligned} C_a &= (00)^*, & C_d &= (00)^*01, \\ C_b &= (00)^*0, & C_e &= 0^*100^*, \\ C_c &= (00)^*1, & C_f &= 0^*10^*1(0 + 1)^* \end{aligned}$$

$L$  es la unión de tres de estas clases,  $C_c$ ,  $C_d$  y  $C_e$ .

La relación  $R_L$  para  $L$  tiene  $x R_L y$  si y sólo si se cumple una de las siguientes i) ni  $x$  ni  $y$  poseen 1s,

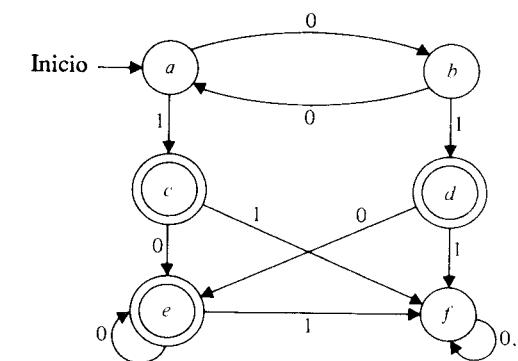


Fig. 3.2 DFA  $M$  que acepta a  $L$ .

- ii)  $x$  y  $y$  tienen un 1 cada una, o
- iii)  $x$  y  $y$  tienen cada una más de un 1.

Por ejemplo, si  $x = 010$  y  $y = 1000$ , entonces  $xz$  está en  $L$  si y sólo si  $z$  está en  $0^*$ . Pero  $yz$  está en  $L$  bajo exactamente las mismas condiciones. Como otro ejemplo, si  $x = 01$  y  $y = 00$ , entonces podemos escoger  $z = 0$  para mostrar que  $x R_L y$  es falso. Esto es,  $xz = 010$  está en  $L$ , pero  $yz = 000$  no lo está.

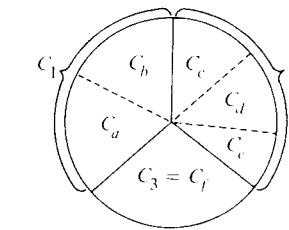
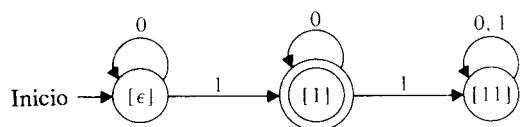


Fig. 3.3 Diagrama que presenta a  $R_M$  como un refinamiento de  $R_L$ .

Podemos representar las tres clases de equivalencia de  $R_L$  con  $C_1 = 0^*$ ,  $C_2 = 0^*10$  y  $C_3 = 0^*10^*1(0 + 1)^*$ .  $L$  es el lenguaje que consiste en sólo una de estas clases,  $C_2$ . En la Fig. 3.3 se ilustra la relación de  $C_a$ , ...,  $C_f$  con  $C_1$ ,  $C_2$  y  $C_3$ . Por ejemplo,  $C_a \cup C_b = (00)^* + (00)^*0 = 0^* = C_1$ .

A partir de  $R_L$  podemos construir un DFA de la manera siguiente. Tómense representantes de  $C_1$ ,  $C_2$  y  $C_3$ , digamos  $\epsilon$ , 1 y 11. Entonces sea  $M'$  el DFA que se muestra en la Fig. 3.4. Por ejemplo,  $\delta'([1], 0) = [1]$ , ya que si  $w$  es cualquier cadena de [1] (nótese que [1] es  $C_1$ ), digamos  $0^i 10^j$ , entonces  $w0$  es  $0^i 10^{j+1}$ , que también está en  $C_1 = 0^*10^*$ .

Fig. 3.4 El DFA  $M'$ .

### Minimización de autómatas finitos

El teorema de Myhill-Nerode tiene, entre otras consecuencias, la implicación de que existe un DFA de estado mínimo esencialmente único para cada conjunto regular.

**Teorema 3.10** El autómata de estado mínimo que acepta a un conjunto  $L$  es único hasta un isomorfismo (es decir, un renombramiento de estados) y está dado como  $M'$  en la demostración del Teorema 3.9.

*Demuestra* En la demostración del Teorema 3.9 vimos que cualquier DFA  $M = (Q, \Sigma, \delta, q_0, F)$  que acepta a  $L$  define una relación de equivalencia que es un refinamiento de  $R_L$ . Por consiguiente el número de estados de  $M$  es mayor o igual que el número de estados de  $M'$  del Teorema 3.9. Si la igualdad es válida, entonces cada uno de los estados de  $M$  puede ser identificado con uno de los estados de  $M'$ . Esto es, sea  $q$  un estado de  $M$ . Debe existir alguna  $x$  en  $\Sigma^*$ , tal que  $\delta(q_0, x) = q$ , de otra manera  $q$  puede ser eliminada de  $Q$ , y hallar, así, un autómata más pequeño. Identifíquese a  $q$  con el estado  $\delta(q_0, x)$  de  $M'$ . Esta identificación será consistente. Si  $\delta(q_0, x) = \delta(q_0, y) = q$ , entonces, según la demostración del teorema 3.9,  $x$  y  $y$  están en la misma clase de equivalencia de  $R_L$ . En consecuencia,  $\delta'(q_0, x) = \delta'(q_0, y)$ .  $\square$

### Un algoritmo de minimización

Existe un método simple para encontrar el DFA de estado mínimo  $M'$  de los Teoremas 3.9 y 3.10, equivalente a un DFA dado  $M = (Q, \Sigma, \delta, q_0, F)$ . Sea el símbolo  $\equiv$  la relación de equivalencia sobre los estados de  $M$ , tal que  $p \equiv q$  si y sólo si para cada cadena de entrada  $x$ ,  $\delta(p, x)$  es un estado de aceptación si y sólo si  $\delta(q, x)$  es un estado de aceptación. Obsérvese que existe un isomorfismo entre aquellas clases de equivalencia de  $\equiv$  que contienen un estado que se puede alcanzar desde  $q_0$  mediante alguna cadena de entrada y los estados del FA de estado mínimo  $M'$ . Así pues los estados de  $M'$  pueden identificarse con estas clases.

Más que dar un algoritmo formal para encontrar las clases de equivalencia  $\equiv$  de primero veremos un ejemplo. En primer lugar se necesita cierta terminología. Si  $p \equiv q$ , decimos que  $p$  es *equivalente* a  $q$ . Decimos que  $p$  es *distinguible* de  $q$  si existe una  $x$  tal que  $\delta(p, x)$  esté en  $F$  y  $\delta(q, x)$  no, o viceversa.

**Ejemplo 3.8** Sea  $M$  el autómata finito que se muestra en la Fig. 3.5. En la Fig. 3.6 hemos construido una tabla con una entrada por cada par de estados. Ponemos una  $X$  en la tabla cada vez que descubrimos un par de estados que no pueden ser equivalentes. Inicialmente se sitúa una  $X$  en cada entrada que corresponda a un estado final y uno no final. En nuestro ejemplo, ponemos una  $X$  en las entradas  $(a, c), (b, c), (c, d), (c, e), (c, f), (c, g)$  y  $(c, h)$ .

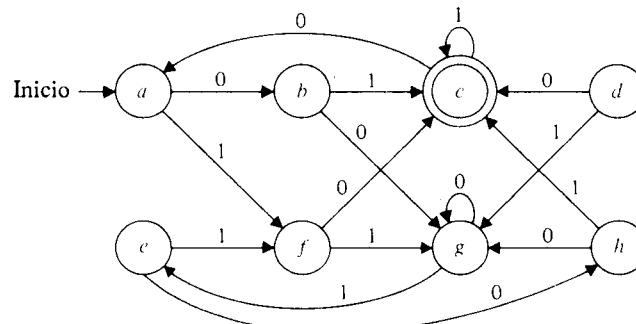


Fig. 3.5 Autómata finito.

b	X						
c	X	X					
d	X	X	X				
e		X	X	X			
f	X	X	X			X	
g	X	X	X	X	X	X	
h	X		X	X	X	X	X
a		b	c	d	e	f	g

Fig. 3.6 Cálculo de estados equivalentes.

En seguida, para cada par de estados  $p$  y  $q$  que aún no se sabe si son distinguibles, consideramos los pares de estados  $r = \delta(p, a)$  y  $s = \delta(q, a)$  para cada símbolo de entrada  $a$ . Si se ha mostrado que los estados  $r$  y  $s$  son distinguibles mediante alguna cadena  $x$ , entonces  $p$  y  $q$  son distinguibles mediante la cadena  $ax$ . Por consiguiente si la entrada  $(r, s)$  de la tabla tiene una  $X$ , también se debe poner una en la entrada  $(p, q)$ . Si la entrada  $(r, s)$  aún no tiene una  $X$ , entonces el par  $(p, q)$  se coloca en una lista que está asociada con la entrada  $(r, s)$ , si en lo futuro, la entrada  $(r, s)$  recibe una  $X$ , entonces cada par de la lista asociada con la entrada  $(r, s)$  también deberá tener una  $X$ .

Continuando con el ejemplo, colocamos una  $X$  en la entrada  $(a, b)$ , puesto que la entrada  $(\delta(b, 1), \delta(a, 1)) = (c, f)$  ya tiene una  $X$ . De forma similar, la entrada  $(a, d)$  recibe una  $X$  ya que la entrada  $(\delta(a, 0), \delta(d, 0)) = (b, c)$  tiene una. Tomando en consideración la entrada  $(a, e)$  sobre el símbolo de entrada 0 trae como resultado que el par  $(a, e)$  sea colocado en la lista de los pares asociados con  $(b, h)$ . Obsérvese que sobre la entrada 1, tanto  $a$  como  $e$  van al mismo estado  $f$  y en consecuencia ninguna cadena que comience con un 1 puede distinguir  $a$  de  $e$ . Debido a la entrada 0, el par  $(a, g)$  se coloca en la lista asociada con  $(b, g)$ . Cuando se considera la entrada  $(b, g)$ , ésta recibe una  $X$  debido a una entrada 1, y por tanto el par  $(a, g)$  también debe tener una  $X$ , pues está en la lista asociada con  $(b, g)$ . La cadena 01 distingue  $a$  de  $g$ .

Al completar la tabla de la Fig. 3.6, concluimos que los estados equivalentes son  $a \equiv e, b \equiv h$  y  $d \equiv f$ . En la Fig. 3.7 se da el autómata finito de estado mínimo.

En la Fig. 3.8 se ilustra el algoritmo para marcar pares de estados no equivalentes. El lema 3.2 demuestra que el método delineado efectivamente señala todos los pares de estados no equivalentes.

**Lema 3.2** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA. Entonces  $p$  es *distinguible* de  $q$  si y sólo si la entrada correspondiente al par  $(p, q)$  está señalada según el procedimiento anterior.

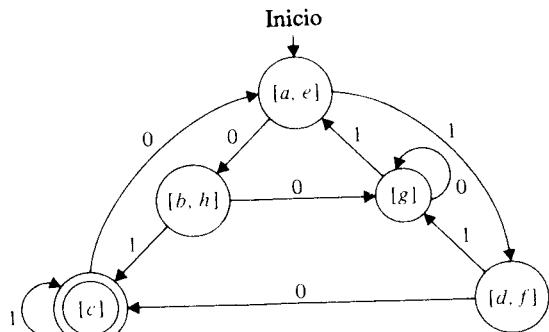


Fig. 3.7 Autómata finito de estado mínimo.

```

begin
1. for  $p$  en  $F$  y  $q$  en  $Q - F$  do marcado de  $(p, q)$ .
2. for cada par de estados distintos  $(p, q)$  en  $F \times F$  o en  $(Q - F) \times (Q - F)$  do
3. if para algún símbolo de entrada  $a$ ,  $(\delta(p, a), \delta(q, a))$  está marcado then
begin
4. señale  $(p, q)$ ;
5. marque recursivamente todos los pares no marcados de la lista
para  $(p, q)$  y de las listas de otros pares marcados en este paso.
end
else /* ningún par  $(\delta(p, a), \delta(q, a))$  está marcado */
6. for todos los símbolos de entrada  $a$  do
7. ponga  $(p, q)$  en la lista para  $(\delta(p, a), \delta(q, a))$  a menos que
end

```

Fig. 3.8 Algoritmo para marcar pares de estados no equivalentes.

**Demostración** Supóngase que  $p$  es distinguible de  $q$  y sea  $x$  la cadena más pequeña que distingue a  $p$  de  $q$ . Probamos por inducción sobre la longitud de  $x$  que la entrada correspondiente al par  $(p, q)$  está señalada. Si  $x = \epsilon$ , entonces exactamente uno de  $p$  y  $q$  es estado final y de aquí que la entrada se señale en la línea (1). Supóngase que la hipótesis es verdadera para  $|x| < i, i > 1$ , y sea  $|x| = i$ . Escríbase  $x = ay$ , y hágase  $t = \delta(p, a)$  y  $u = \delta(q, a)$ . Ahora  $y$  distingue a  $t$  de  $u$  y  $|y| = i - 1$ . Así pues, por la hipótesis de inducción la entrada correspondiente al par  $(t, u)$  tendrá que señalarse. Si este evento ocurre después, de que se ha considerado el par  $(p, q)$ , entonces la entrada  $(p, q)$  ya ha sido marcada cuando se consideró  $(t, u)$ , o el par  $(p, q)$  se encuentra en la lista asociada con  $(t, u)$ , en cuyo caso es marcada en la línea (5). Si  $(p, q)$  se considera después de  $(t, u)$  entonces  $(p, q)$  se señala al mismo tiempo que es considerada. En cualquier caso la entrada  $(p, q)$  se señala. Una inducción parecida sobre el número de pares señalados muestra que si la entrada  $(p, q)$  está señalada, entonces  $p$  y  $q$  son distinguibles.  $\square$

El algoritmo de la Fig. 3.8 es más eficiente que el evidente algoritmo para marcar, aunque no es el más eficiente posible. Sea  $\Sigma$  un alfabeto con  $k$  símbolos y  $Q$  un conjunto de  $n$  estados. La línea 1 toma  $0(n^2)$  pasos.<sup>†</sup> El ciclo de líneas que va de la 2 a la 7 se ejecuta  $0(n^2)$  veces, al menos una por cada par de estados. El tiempo total que se gasta en las líneas 2 a 4, 6 y 7 es  $0(kn^2)$ . El tiempo que se invierte en la línea 5 es la suma de la longitud de todas las listas. Pero cada par  $(r, s)$  se coloca en al menos  $k$  listas en la línea 7. Por tanto el tiempo que se gasta en la línea 5 es  $0(kn^2)$ , de modo que el tiempo total también es  $0(kn^2)$ .

**Teorema 3.11** El DFA construido por el algoritmo de la Fig. 3.8, en el que se han eliminado los estados inaccesibles, es el DFA de estado mínimo para su lenguaje.

**Demostración** Sea  $M = (Q, \Sigma, \delta, q_0, F)$  el DFA al cual se aplica el algoritmo y sea  $M' = (Q', \Sigma, \delta', [q_0], F')$  el DFA construido. Esto es,

$$\begin{aligned} Q' &= \{[q] \mid q \text{ es accesible desde } q_0\}, \\ F' &= \{[q] \mid q \text{ está en } F\} \end{aligned}$$

y

$$\delta'([q], a) = [\delta(q, a)].$$

Es fácil mostrar que  $\delta'$  está definida de manera consistente, puesto que si  $q \equiv p$ , entonces  $\delta(q, a) \equiv \delta(p, a)$ . Esto es, si  $\delta(q, a)$  se distingue de  $\delta(p, a)$  mediante  $x$ , entonces  $ax$  distingue a  $q$  de  $p$ . También es fácil mostrar que  $\delta'([q_0], w) = [\delta(q_0, w)]$  por inducción en  $|w|$ . Por consiguiente  $L(M') = L(M)$ .

Ahora debemos probar que  $M'$  tiene no más estados que  $R_L$  tiene clases de equivalencia, en donde  $L = L(M)$ . Supóngase que esto sucede; entonces existen dos estados accesibles  $q$  y  $p$  en  $Q$  tales que  $[q] \neq [p]$ , sin embargo existen  $x$  y  $y$  tales que  $\delta(q_0, x) = q, \delta(q_0, y) = p$  y  $x R_L y$ . Pedimos que  $p \equiv q$ , porque si no, entonces alguna  $w$  de  $\Sigma^*$  distingue a  $p$  de  $q$ . Pero entonces  $xw R_L yw$  es falso, porque podemos hacer  $z = \epsilon$  y observar que exactamente una de  $xwz$  y  $ywz$  está en  $L$ . Pero como  $R_L$  es inva-

<sup>†</sup> Decimos que  $g(n)$  es  $O(f(n))$  si existen constantes  $c$  y  $n_0$  tales que  $g(n) \leq cf(n)$  para toda  $n \geq n_0$ .

riante por la derecha,  $xwR_Lyw$  es verdadera. De aquí que  $q$  y  $p$  no existan y  $M'$  no tenga más estados que el índice de  $R_L$ . En consecuencia  $M'$  es el DFA de estado mínimo para  $L$ .  $\square$

## EJERCICIOS

**3.1** ¿Cuáles de los siguientes lenguajes son conjuntos regulares? Demuestre su respuesta.

- $\{0^{2n} \mid n \geq 1\}$
- $\{0^m 1^n 0^{m-n} \mid m \geq 1 \text{ y } n \geq 1\}$
- $\{0^n \mid n \text{ es primo}\}$
- el conjunto de todas las cadenas que no tienen tres 0s consecutivos.
- el conjunto de todas las cadenas que no tienen un número igual de 0s y 1s.
- $\{x \mid x \text{ está en } (0+1)^*\text{ y }x = x^k\} \text{ } x^k \text{ es } x \text{ escrito al revés; por ejemplo, } (011)^k = 110$
- $\{xwx^k \mid x, w \text{ están en } (0+1)^*\}$
- \* $\{xx^kw \mid x, w \text{ están en } (0+1)^*\}$

**3.2** Pruebe la siguiente extensión del lema de sondeo para conjuntos regulares. Sea  $L$  un conjunto regular. Entonces existe una constante  $n$  tal que para cada  $z_1, z_2, z_3$ , con  $z_1, z_2, z_3$  en  $L$  y  $|z_2| = n, z_2$  puede escribirse como  $z_2 = uvw$  tal que  $|v| \geq 1$  y para cada  $i \geq 0$ ,  $z_1uv^iwz_3$  está en  $L$ .

**3.3** Utilice el ejercicio 3.2 para probar que  $\{0^i 1^m 2^m \mid i \geq 1, m \geq 1\}$  es no regular.

**3.4** Sea  $L$  un conjunto regular. ¿Cuáles de los siguientes conjuntos son regulares? Justifique su respuesta.

- $\{a_1a_3a_5 \dots a_{2n-1} \mid a_1a_2a_3a_4 \dots a_{2n} \text{ está en } L\}$
- $\{a_2a_4a_6 \dots a_{2n} \mid a_1a_2 \dots a_{2n} \text{ está en } L\}$
- S c) CICLO ( $L$ ) =  $\{x_1x_2 \mid x_1x_2 \text{ está en } L \text{ para las cadenas } x_1 \text{ y } x_2\}$
- d) MAX( $L$ ) =  $\{x \in L \mid \text{para ninguna } y \text{ diferente de } x \text{ se tiene } xy \in L\}$
- e) MIN( $L$ ) =  $\{x \in L \mid \text{ningún prefijo propio de } x \text{ está en } L\}$
- f) INIT( $L$ ) =  $\{x \mid \text{para alguna } y, xy \in L\}$
- g)  $L^R = \{x \mid x^R \text{ está en } L\}$
- h)  $\{x \mid xx^R \text{ está en } L\}$

**3.5** Sea valor ( $x$ ) el resultado que se obtiene cuando los símbolos de  $x$  se multiplican de izquierda a derecha, de acuerdo con la tabla de la Fig. 2.31.

- ¿Es  $L = \{xy \mid |x| = |y| \text{ y valor}(x) = \text{valor}(y)\}$  regular?
- ¿Es  $L = \{xy \mid \text{valor}(x) = \text{valor}(y)\}$  regular?

Justifique su respuesta.

**\*3.6** Muestre que  $\{0^i 1^j \mid \gcd(i, j) = 1\}$  es no regular.

**\*\*3.7** Sea  $L$  cualquier subconjunto de  $0^*$ . Pruebe que  $L^*$  es regular.

**3.8** Un conjunto de enteros es lineal si es de la forma  $\{c + pi \mid i = 0, 1, 2, \dots\}$ . Un conjunto es semilineal si es la unión finita de conjuntos lineales. Sea  $R \subseteq 0^*$  regular. Pruebe que  $\{i \mid 0^i \text{ está en } R\}$  es semilineal.

**3.9** ¿Es la clase de conjuntos regulares cerrada con respecto a la unión infinita?

**3.10** ¿Cuál es la relación entre la clase de conjuntos regulares y la menor clase de lenguajes, cerrada con respecto a la unión, intersección y complemento, que contiene a todos los conjuntos finitos?

**\*3.11** Dé la construcción de un autómata finito para probar que la clase de conjuntos regulares es cerrada con respecto a la sustitución.

**\*\*3.12** ¿Es la clase de los conjuntos regulares cerrada con respecto a la sustitución inversa?

**3.13** Sea  $h$  el homomorfismo  $h(a) = 01, h(b) = 0$ .

- Encuentre  $h^{-1}(L_1)$ , en donde  $L_1 = (10+1)^*$
- Encuentre  $h(L_2)$ , en donde  $L_2 = (a+b)^*$
- Encuentre  $h^{-1}(L_3)$ , en donde  $L_3$  es el conjunto de todas las cadenas de 0s y 1s con un número igual de 0s y 1s.

**3.14** Muestre que 2DFA con señaladores de extremo (véase el Ejercicio 2.20) aceptan sólo conjuntos regulares mediante el uso de las propiedades de cerradura que se desarrollaron en este capítulo.

**\*\*3.15** El uso de  $\cap$  con expresiones regulares no permite la representación de nuevos conjuntos. Sin embargo permite expresiones más compactas. Muestre que  $\cap$  puede acortar una expresión regular mediante una cantidad exponencial. [Indicación: ¿Cuál es la expresión regular de longitud más corta que describe al conjunto que consiste en la oración (...(( $a_0^2 a_1^2)^2 a_2^2 \dots)^2?)?$

**\*\*3.16** Sea  $L$  un lenguaje. Defínase  $1/2(L)$  como

$$\{x \mid \text{para alguna } y \text{ tal que } x = |y|, xy \text{ está en } L\}$$

Es decir,  $1/2(L)$  es la primer mitad de las cadenas de  $L$ . Pruebe para cada  $L$  regular que  $1/2(L)$  es regular.

**\*\*3.17** Si  $L$  es regular, ¿es el conjunto del primer tercio de las cadenas de  $L$  regular? ¿Qué se puede decir del último tercio? ¿Del tercio de en medio? Determíñese si es regular el siguiente conjunto:

$$\{xz \mid \text{para alguna } y \text{ con } |x| = |y| = |z|, xyz \text{ está en } L\}$$

**\*\*3.18** Muestre que si  $L$  es regular, también lo son

- $\text{SQRT}(L) = \{x \mid \text{para alguna } y \text{ con } |y| = |x|^2, xy \text{ está en } L\}$
- $\text{LOG}(L) = \{x \mid \text{para alguna } y \text{ con } |y| = 2^{|x|}, xy \text{ está en } L\}$

**\*3.19** Un 2DFA de una cuenta es un 2DFA que posee la capacidad adicional de señalar el cuadro de una cinta colocando una cuenta en él. La siguiente función de estado depende del estado presente, el símbolo de la cinta barrido y la presencia o ausencia de una cuenta en el cuadro de la cinta que se está leyendo. Un movimiento consiste en un cambio de estado, en la dirección del movimiento de la cabeza y posiblemente en colocar o retirar la cuenta de la celda de la cinta barrida. El autómata se «bloquea» si intenta colocar una segunda cuenta de la celda de la cinta barrida. El autómata se «bloquea» si intenta colocar una segunda cuenta en la entrada. Pruebe que los 2DFA de una cuenta aceptan solamente conjuntos regulares. [Sugerencia: añada dos registros adicionales a la entrada que contengan tablas en donde se indique para cada estado  $p$ , el estado  $q$  en el que el 2DFA regresará si se mueve a la derecha o a la izquierda de la celda de la cinta en el estado  $p$ , bajo la suposición de que no se encuentra con la cuenta. Observe que el 2DFA de una cuenta que se encuentra operando sobre la cinta aumentada no necesita nunca abandonar su cuenta. Después haga uso de una transformación homomórfica para eliminar los registros adicionales.]

**\*3.20** Al convertir un NFA en un DFA el número de estados puede aumentar en forma considerable. Dé límites superior e inferior para el máximo incremento en el número de estados para un NFA de  $n$  estados. [Sugerencia: Tome en cuenta los Ejercicios 2.5(e) y 2.8(c).]

**3.21** Dé un procedimiento de decisión para determinar si el conjunto aceptado por un DFA es

- a) el conjunto de todas las cadenas sobre un alfabeto dado.  
 b) *cofinito* (un conjunto cuyo complemento es finito).

**\*\*3.22** Considérese un DFA  $M$ . Supóngase que  $M$  tiene cuando más  $n$  estados y que se desea determinar el diagrama de transiciones de  $M$ . Aún más, supóngase que la única forma de obtener información con respecto a  $M$  es aportando una secuencia de entrada  $x$  y observando los prefijos de  $x$  que son aceptados.

- a) ¿Qué suposiciones debe hacer con respecto al diagrama de transiciones de  $M$  con el fin de ser capaz de determinar dicho diagrama?  
 b) Dé un algoritmo para determinar el diagrama de transiciones de  $M$  (excepto para el estado inicial) incluyendo la construcción de  $x$  bajo las suposiciones de la parte (a).

**\*\*S3.23** Dé un procedimiento de decisión eficiente para determinar si  $x$  está en el lenguaje representado por una *expresión regular extendida* (una expresión regular con operadores  $\cup$ ; (concatenación),  $*$ ,  $\cap$  y  $\neg$ , el complemento).

**3.24** Dé un procedimiento de decisión eficiente para determinar si una *expresión regular semi-extendida*  $r$  (una expresión regular con operadores  $\cup$ , ;  $*$ ,  $\cap$ ) denota a un conjunto no vacío. [Indicación: El espacio  $O(|r|)$  y el tiempo  $O(2^{|r|})$  son suficientes.]

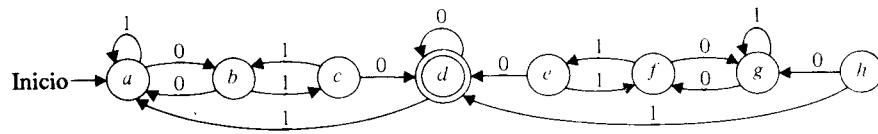


Fig. 3.9. Un autómata finito.

**3.25** Encuentre el autómata finito de estado mínimo equivalente al diagrama de transiciones de la Fig. 3.9.

**3.26**

- a) ¿Cuáles son las clases de equivalencia de RL del teorema de Myhill-Nerode (Teorema 3.9) para  $L = \{0^n 1^n \mid n \geq 1\}$ ?  
 b) Utilice la respuesta dada en (a) para mostrar que  $\{0^n 1^n \mid n \geq 1\}$  es no regular.  
 c) Repita (a) para  $\{x \mid x \text{ tiene un número igual de } 0s \text{ y } 1s\}$ .

**\*3.27**  $R$  es una relación de congruencia si  $xRy$  implica que  $wxzRwyz$  para todo  $w$  y  $z$ . Pruebe que un conjunto es regular si y sólo si es la unión de algunas de las clases de congruencia de una relación de congruencia de índice finito.

**\*3.28** Sea  $M$  un autómata finito con  $n$  estados. Sean  $p$  y  $q$  estados distinguibles de  $M$  y  $x$  la cadena más corta que distingue a  $p$  de  $q$ . ¿Qué tan larga puede ser una cadena  $x$  como función de  $n$ ?

**\*\*3.29** En un FA de dos cintas cada estado está designado como si leyera la cinta 1 o la 2. Un par de cadenas  $(x, y)$  es aceptado si el FA, cuando se presenta con cadenas  $x$  y  $y$  en sus respectivas cintas, alcanza un estado final con las cabezas de cinta situadas inmediatamente a la derecha de  $x$  y  $y$ . Sea  $L$  el conjunto de pares aceptados por un FA de dos cintas  $M$ . Dé algoritmos para responder a las siguientes cuestiones.

- a) ¿Está  $L$  vacío?    b) ¿Es  $L$  finito?  
 c) ¿Existen  $L_1$  y  $L_2$  tales que  $L = L_1 \times L_2$ ?

### 3.30

- a) Pruebe que existe una constante  $c > 0$  tal que el algoritmo de la Fig. 3.8 requiera un tiempo mayor que  $cn^2$  para un número infinito de DFAs, en donde  $n$  es el número de estados y el alfabeto de entrada tiene dos símbolos.  
**\*\* b)** Dé un algoritmo para minimizar los estados de un DFA cuyo tiempo de ejecución es  $O(|\Sigma| n \log n)$ . Aquí  $\Sigma$  es el alfabeto de entrada. [Sugerencia: En lugar de preguntar por cada par de estados  $(p, q)$  y cada entrada  $a$  si  $\delta(p, a)$  y  $\delta(q, a)$  son distinguibles, divida los estados en estados finales y no finales. Entonces refine la partición mediante la consideración de todos los estados cuyo estado siguiente, bajo algún símbolo de entrada, es un bloque particular de particiones. Cada vez que un bloque es dividido, refine la partición aún más mediante el uso del subbloque más pequeño. Utilice un proceso de listado para hacer el algoritmo tan eficiente como sea posible.]

### Soluciones a los ejercicios seleccionados

**3.4(b)**  $L' = \{a_2 a_1 a_4 a_3 \dots a_{2n} a_{2n-1} \mid a_1 a_2 \dots a_{2n} \text{ está en } L\}$  es regular. Sea  $M = (Q, \Sigma, \delta, q_0, F)$  un DFA que acepta a  $L$ . Construimos un DFA  $M'$  que acepte a  $L'$ .  $M'$  procesará los símbolos de la cinta por pares. Al observar el primer símbolo  $a$  de un par,  $M'$  almacena  $a$  en su control finito. Entonces, al observar el segundo símbolo  $b$ ,  $M'$  se comporta como  $M$  sobre la entrada  $ba$ . De manera más formal,

$$M' = (Q \cup Q \times \Sigma, \Sigma, \delta', q_0, F)$$

UNIVERSIDAD DE LA REPÚBLICA  
FACULTAD DE INGENIERÍA  
DEPARTAMENTO DE  
DOCUMENTACIÓN Y BIBLIOTECA  
MONTEVIDEO - URUGUAY

en donde

- i)  $\delta'(q, a) = [q, a]$ , y  
 ii)  $\delta(q, a, b) = \delta(q, ba)$ .

Para probar que  $M'$  acepta a  $L'$  mostramos por inducción en  $i$  que

$$\delta'(q, a_2 a_1 a_4 a_3 \dots a_i a_{i-1}) = \delta(q, a_1 a_2 \dots a_i)$$

Claramente, para  $i = 0$ ,  $\delta'(q, \epsilon) = q = \delta(q, \epsilon)$ . Supongamos que la hipótesis es verdadera para todo  $j < i$  par. Por la hipótesis de inducción,

$$\begin{aligned} \delta'(q, a_2 a_1 \dots a_{i-2} a_{i-3}) &= \delta(q, a_1 a_2 \dots a_{i-3}) \\ &= p \text{ para algún } p. \end{aligned}$$

Por consiguiente

$$\begin{aligned} \delta'(q, a_2 a_1 \dots a_i a_{i-1}) &= \delta'(p, a_i a_{i-1}) \\ &= \delta([p, a_i], a_{i-1}) \\ &= \delta(p, a_{i-1} a_i) \\ &= \delta(q, a_1 a_2 \dots a_i) \end{aligned}$$

Por tanto  $a_2 a_1 a_4 a_3 \dots a_i a_{i-1}$  está en  $L(M')$  si y sólo si  $a_1 a_2 \dots a_i$  está en  $L(M)$ , y en consecuencia  $L(M') = L'$ .  $\square$

**3.23** Claramente, uno puede construir un autómata finito equivalente a  $R$  mediante la combinación de los autómatas finitos correspondientes a las subexpresiones de  $R$  y después con

la simulación del autómata en  $x$ . Debemos examinar el proceso de combinación para ver cómo afecta al tamaño del autómata resultante. Si trabajamos con DFAs entonces el número de estados de una unión o intersección crece según el producto. Sin embargo, la concatenación y la cerradura pueden aumentar el número de estados de manera exponencial, según se tiene necesidad de convertir los DFAs a NFAs y entonces llevar a cabo la construcción del subconjunto. Si trabajamos con NFAs, entonces el número de estados es aditivo para la unión, la concatenación y la cerradura, y aumenta según el producto para la intersección. Sin embargo, los complementos requieren una conversión de un NFA a un DFA y por tanto se tiene un aumento exponencial en el número de estados. Como los operadores pueden estar anidados, el número de estados puede exponenciarse en el orden de  $n$  veces para una expresión con  $n$  operadores, y por consiguiente esta técnica no es en general factible.

Un método más eficiente, basado en una técnica de programación dinámica, (véase Aho, Hopcroft y Ullman [1974]) produce un algoritmo cuyo tiempo de ejecución es polinomial con respecto a la longitud de la entrada  $w$  y la longitud de la expresión regular  $s$ . Sea  $n = |w| + |s|$ . Constrúyase una tabla en la cual, para cada subexpresión  $r$  de  $s$  y cada subcadena  $x_{ij}$  de  $w$ , se dé la respuesta a la pregunta: ¿está  $x_{ij}$  en  $L(r)$ , en donde  $x_{ij}$  es la subcadena de  $w$  de longitud  $j$  que comienza en la posición  $i$ ? La tabla será, cuando más, de tamaño  $n^3$ , puesto que, cuando más, existan  $n$  subexpresiones de  $s$  y  $n(n+1)/2$  subcadenas de  $w$ . Llénese la tabla comenzando con las entradas para las subexpresiones pequeñas (aquellas que no tengan operadores, es decir,  $a$ ,  $\epsilon$ , o  $\emptyset$ ). Entonces llénense las entradas para  $x$  y  $r$ , en donde  $r$  tiene una de las formas  $r_1 \cap r_2$ ,  $r_1 + r_2$ ,  $r_1 r_2$ ,  $r_1^*$  o  $\neg r_1$ . Trataremos solamente el caso de  $r_1^*$ . Procederemos de acuerdo con el orden de la longitud de  $x$ . Para determinar si  $x$  está en  $r_1^*$ , dado que ya sabemos, para cada subcadena  $y$  de  $x$ , si  $y$  está en  $r_1$  o está en  $r_1^*$ , sólo necesitamos verificar para cada  $x_1$  y  $x_2$  tales que  $x = x_1 x_2$  y  $x_1 \neq \epsilon$ , si  $x_1$  está en  $r_1$  y  $x_2$  está en  $r_1^*$ . Por consiguiente, para calcular la entrada de la tabla para  $x$  y  $r$  se requiere un tiempo de  $O(|x| + |r|)$ . Por tanto, el tiempo que se necesita para llenar la tabla completa es  $O(n^4)$ . Para determinar si  $w$  está en  $s$  sólo necesitamos consultar la entrada para  $s$  y  $w$ , y nos damos cuenta que  $w = x_{ik}$  en donde  $k = |w|$ .

## NOTAS BIBLIOGRAFICAS

El lema de sondeo para conjuntos regulares está basado en la formulación de Bar-Hillel, Perles y Shamir [1961]. El Teorema 3.4, cerradura con respecto a la sustitución, se tomó también de la misma referencia. El Teorema 3.5, cerradura con respecto al homomorfismo inverso, se tomó de Ginsburg y Rose [1963b] y el Teorema 3.6, con respecto a los cocientes, es de Ginsburg y Spanier [1963].

Los Teoremas 3.7 y 3.8, sobre los algoritmos de decisión, fueron sacados de Moore [1956]. Ginsburg y Rose [1966] dan varias propiedades de cerradura para conjuntos regulares además de las de este libro.

El teorema 3.9, al que hemos nombrado Teorema de Myhill-Nerode, se debe en realidad a un trabajo de Nerode [1958]. El resultado similar del Ejercicio 3.27, sobre relaciones de congruencia se debe a Myhill [1957]. El algoritmo para minimizar autómatas finitos se debe a Huffman [1954] y Moore [1956]. Hopcroft [1971] proporciona un algoritmo más eficiente.

El ejemplo 3.2, sobre el no reconocimiento de los números primos en binario, fue demostrado por Minsky y Papert [1966] utilizando otro método. Las operaciones de remoción proporcional, como las que se ven en el Ejercicio 3.16, fueron estudiadas por primera vez en su generalidad por Stearns y Hartmanis [1963]. Las generalizaciones como las que aparecen en el Ejercicio 3.18 fueron consideradas por Kosaraju [1974] y Seiferas [1974] y la cuestión sobre qué funciones de la longitud de la cadena pueden ser eliminadas del frente para producir conjuntos

regulares fue resuelta completamente por Seiferas y McNaughton [1976]. Una solución al Ejercicio 3.22 fue considerada por primera vez por Hennie [1964]. Un algoritmo para determinar la equivalencia para FA determinísticos de dos cintas se encuentra en Bird [1973].

## 4.1 MOTIVACION E INTRODUCCION

En este capítulo introducimos las gramáticas libres de contexto y los lenguajes que describen: los lenguajes libres de contexto. Los lenguajes libres de contexto, como los conjuntos regulares, son de gran importancia práctica, sobre todo en la definición de lenguajes de programación, en la formalización del concepto de análisis de gramática, simplificación de la traducción de lenguajes de programación y en otras aplicaciones del procesamiento de cadenas. Como ejemplo, las gramáticas libres de contexto son útiles para describir expresiones aritmética que tengan una anidación arbitraria de paréntesis balanceados y estructuras de bloque en los lenguajes de programación (es decir, el par **begin** y **end**<sup>†</sup> como paréntesis balanceados). Ninguno de estos aspectos de los lenguajes de computación pueden ser representados por expresiones regulares.

Una *gramática libre de contexto* es un conjunto de variables (también conocidas como *categorías no terminales o sintácticas*) cada una de las cuales representa un lenguaje. Los lenguajes representados por las variables se describen de manera recursiva en términos de las mismas variables y de símbolos primitivos llamados *terminales*. Las reglas que relacionan a las variables se conocen como *producciones*. Una producción típica establece que el lenguaje asociado con una variable dada contiene cadenas que se forman mediante la concatenación de cadenas tomadas de los lenguajes representados por otras ciertas variables, posiblemente junto con algunas terminales.

<sup>†</sup> Las instrucciones propias de lenguajes de computación, como **begin**, **end**, **do**, **else**, etcétera, no las traducimos al español, pues se trata de términos utilizados universalmente en su idioma original. (N. del T.)

La motivación original para el desarrollo de las gramáticas libres de contexto fue la necesidad de describir los lenguajes naturales. Podemos escribir reglas como las siguientes:

$$\begin{aligned}
 <\text{oración}> &\rightarrow <\text{frase sustantiva}> <\text{frase verbal}> \\
 <\text{frase sustantiva}> &\rightarrow <\text{adjetivo}> <\text{frase sustantiva}> \\
 <\text{frase sustantiva}> &\rightarrow <\text{sustantivo}> \\
 <\text{sustantivo}> &\rightarrow \text{niño} \\
 <\text{adjetivo}> &\rightarrow \text{pequeño}
 \end{aligned} \tag{4.1}$$

en donde las categorías sintácticas† se escriben entre paréntesis en ángulo y las terminales se denotan con palabras sin paréntesis como “niño” y “pequeño”.

El significado de

$$<\text{oración}> \rightarrow <\text{frase sustantiva}> <\text{frase verbal}>$$

es que una manera de formar una oración (una cadena del lenguaje de las categorías sintácticas  $<\text{oración}>$ ) consiste en tomar una frase sustantiva seguida de una frase verbal. El significado de

$$<\text{sustantivo}> \rightarrow \text{niño}$$

es que la cadena que consiste en el símbolo terminal “niño” está en el lenguaje de la categoría sintáctica  $<\text{sustantivo}>$ . Nótese que “niño” es un solo símbolo terminal y no una cadena de tres símbolos.

Por ciertas razones, las gramáticas libres de contexto, en general, no se consideran como adecuadas para la descripción de los lenguajes naturales, como el español. Por ejemplo, si extendemos las producciones de (4.1) a todo el idioma, podríamos ser capaces de tomar “la roca” como una frase sustantiva y “corre” como una frase verbal. Por consiguiente, “la roca corre” sería una oración, que carece de significado. Es claro que se necesita alguna información semántica para reglamentar cadenas sin significado que son sintácticamente correctas. Problemas más sutiles se presentan cuando se intenta asociar el significado de la oración con su procedencia. Sin embargo, las gramáticas libres de contexto juegan un importante papel en lingüística computacional.

Mientras que los lingüistas estudiaban las gramáticas libres de contexto, los científicos de la computación comenzaron a describir lenguajes de programación mediante una notación conocida como *forma de Backus - Naur* (BNF), que es la notación para las gramáticas libres de contexto con cambios menores en su formato y algunas abreviaturas. Este uso de las gramáticas libres de contexto ha simplificado grandemente la definición de lenguajes de computación y la construcción de compiladores. La razón para que se diera este suceso,

† Recuérdese que el término «categoría sintáctica» es un sinónimo de «variable». Se prefiere el primero cuando se habla de lenguajes naturales.

indudablemente, se debe en parte a la forma natural en que la mayoría de las construcciones de programas de computación son descritas por gramáticas. Por ejemplo, considérese el conjunto de producciones

$$\begin{aligned}
 1. & <\text{expresión}> \rightarrow <\text{expresión}> + <\text{expresión}> \\
 2. & <\text{expresión}> \rightarrow <\text{expresión}> * <\text{expresión}> \\
 3. & <\text{expresión}> \rightarrow (<\text{expresión}>) \\
 4. & <\text{expresión}> \rightarrow \text{id}
 \end{aligned} \tag{4.2}$$

que define las expresiones aritméticas que contienen operadores + y \*, y con los operandos representados por el símbolo **id**. Aquí  $<\text{expresión}>$  es la única variable, y las terminales son +, \*, (,) e **id**. Las dos primeras producciones dicen que una expresión puede componerse de dos expresiones conectadas por un signo de adición o de multiplicación. La tercera producción dice que una expresión puede ser otra expresión rodeada por paréntesis. La última dice que un solo operando es una expresión.

Mediante la aplicación repetida de producciones podemos obtener expresiones más y más complicadas. Por ejemplo,

$$\begin{aligned}
 \text{expresión} \Rightarrow & <\text{expresión}> * <\text{expresión}> \\
 \Rightarrow & (<\text{expresión}>) * <\text{expresión}> \\
 \Rightarrow & (<\text{expresión}>) * \text{id} \\
 \Rightarrow & (<\text{expresión}> + <\text{expresión}>) * \text{id} \\
 \Rightarrow & (<\text{expresión}> + \text{id}) * \text{id} \\
 \Rightarrow & (\text{id} + \text{id}) * \text{id}
 \end{aligned} \tag{4.3}$$

El símbolo  $\Rightarrow$  denota el acto de derivación, es decir, la sustitución de una variable por el lado derecho de una producción para esa variable. La primera línea de (4.3) se obtiene de la segunda producción. La segunda línea se obtiene sustituyendo la primera  $<\text{expresión}>$  en la línea 1 por el lado derecho de la tercera producción. Las líneas restantes son el resultado de aplicar las producciones (4), (1), (4) y (4). La última línea,  $(\text{id} + \text{id}) * \text{id}$ , consiste solamente en símbolos terminales y por tanto es una palabra en el lenguaje de  $<\text{expresión}>$ .

## 4.2 GRAMATICAS LIBRES DE CONTEXTO

Ahora formalizaremos las nociones intuitivas introducidas en la sección anterior. Una *gramática libre de contexto* (CFG o simplemente *gramática*) se denota por  $G = (V, T, P, S)$ , en donde  $V$  y  $T$  son conjuntos finitos de *variables* y *terminales*, respectivamente. Suponemos que  $V$  y  $T$  son disjuntos.  $P$  es un conjunto finito de producciones; cada producción es de la forma  $A \rightarrow \alpha$ , en donde  $A$  es una variable y  $\alpha$  es una cadena de símbolos tomados de  $(V \cup T)^*$ . Por último,  $S$  es una variable especial conocida como el *símbolo inicial*.

**Ejemplo 4.1** Supóngase que utilizamos  $E$  en lugar de < expresión > para la variable de la gramática (4.2). Entonces podemos expresar esta gramática de manera formal como  $(\{E\}, \{+, *, (), \text{id}\}, P, E)$ , en donde  $P$  consiste en

$$\begin{aligned} E &\rightarrow E + E \\ E &\rightarrow E * E \\ E &\rightarrow (E) \\ E &\rightarrow \text{id} \end{aligned}$$

En éste y los dos siguientes capítulos utilizaremos las reglas siguientes con respecto a las gramáticas.

1. Las letras mayúsculas  $A, B, C, D, E$  y  $S$  representan variables;  $S$  es el símbolo inicial, a menos que se indique lo contrario.
2. Las letras minúsculas  $a, b, c, d, e$ , los dígitos y las cadenas en negritas son terminales.
3. Las letras mayúsculas  $X, Y$  y  $Z$  representan símbolos que pueden ser terminales o variables.
4. Las letras minúsculas  $u, v, w, x, y$  y  $z$  denotan cadenas de terminales.
5. Las letras griegas minúsculas  $\alpha, \beta$ , y  $\gamma$  denotan cadenas de variables y terminales.

Adhiriéndonos a estas reglas anteriores, podemos deducir las variables, terminales y símbolos iniciales de una gramática simplemente mediante el examen de las producciones. Por tanto presentaremos con frecuencia una gramática solamente listando sus producciones. Si  $A \rightarrow \alpha_1, A \rightarrow \alpha_2, \dots, A \rightarrow \alpha_k$  son las producciones para la variable  $A$  de alguna gramática, entonces podemos expresarlas mediante la notación

$$A \rightarrow \alpha_1 | \alpha_2 | \cdots | \alpha_k,$$

en donde la línea vertical se lee “o”. La gramática completa del Ejemplo 4.1 podría escribirse como

$$E \rightarrow E + E | E * E | (E) | \text{id}$$

### Derivaciones y lenguajes

Definiremos, ahora, de manera formal el lenguaje generado por una gramática  $G = (V, T, P, S)$ . Para hacerlo, desarrollaremos la notación para representar una derivación. Primero definimos dos relaciones  $\Rightarrow$  y  $\stackrel{*}{\Rightarrow}$  entre cadenas de  $(V \cup T)^*$ . Si  $A \rightarrow \beta$  es una producción de  $P$  y  $\alpha$  y  $\gamma$  son cualesquiera cadenas de  $(V \cup T)^*$ , entonces  $\alpha A \gamma \stackrel{*}{\Rightarrow} \alpha \beta \gamma$ . Decimos que la producción  $A \rightarrow \beta$  está aplicada a la cadena  $\alpha A \gamma$  para obtener  $\alpha \beta \gamma$  o que  $\alpha A \gamma$  deriva directamente a  $\alpha \beta \gamma$  en la gramática  $G$ . Dos cadenas están relacionadas por  $\Rightarrow$  exactamente cuando la segunda se obtiene a partir de la primera mediante una aplicación de una producción.

Supóngase que  $\alpha_1, \alpha_2, \dots, \alpha_m$  son cadenas de  $(V \cup T)^*$ ,  $m \geq 1$  y

$$\alpha_1 \stackrel{*}{\Rightarrow} \alpha_2, \alpha_2 \stackrel{*}{\Rightarrow} \alpha_3, \dots, \alpha_{m-1} \stackrel{*}{\Rightarrow} \alpha_m.$$

Entonces decimos que  $\alpha_1 \stackrel{*}{\Rightarrow} \alpha_m$  o  $\alpha_1$  deriva a  $\alpha_m$  en la gramática  $G$ . Es decir,  $\stackrel{*}{\Rightarrow}$  es la cerradura transitiva y reflexiva de  $\Rightarrow$  (véase la Sección 1.5 sobre la discusión de cerraduras de relaciones). Alternativamente,  $\alpha \stackrel{*}{\Rightarrow} \beta$  si  $\beta$  se concluye de  $\alpha$  mediante la aplicación de cero o más producciones de  $P$ . Nótese que  $\alpha \Rightarrow \alpha$  para cada cadena  $\alpha$ . Por lo general, si está claro que gramática  $G$  está involucrada, utilizamos  $\Rightarrow$  en lugar de  $\stackrel{*}{\Rightarrow}$ , y  $\stackrel{*}{\Rightarrow}$  en lugar de  $\stackrel{*}{\Rightarrow}$ . Si  $\alpha$  deriva a  $\beta$  exactamente en  $i$  pasos, decimos que  $\alpha \stackrel{i}{\Rightarrow} \beta$ .

El lenguaje generado por  $G$  denotado por  $L(G)$  es  $\{w \mid w \text{ está en } T^*\text{ y } S \stackrel{*}{\Rightarrow} w\}$ . Esto es, una cadena está en  $L(G)$  si:

1. La cadena consiste solamente en terminales.
2. La cadena puede derivarse a partir de  $S$ .

Llamamos a  $L$  un lenguaje libre de contexto (CFL) si éste es  $L(G)$  para alguna CFG  $G$ . Una cadena de terminales y variables  $\alpha$  se conoce como forma oracional si  $S \stackrel{*}{\Rightarrow} \alpha$ . Definimos las gramáticas  $G_1$  y  $G_2$  como equivalentes si  $L(G_1) = L(G_2)$ .

**Ejemplo 4.2** Considérese una gramática  $G = (V, T, P, S)$ , en donde  $V = \{S\}$ ,  $T = \{a, b\}$  y  $P = \{S \rightarrow aSb, S \rightarrow ab\}$ . Aquí,  $S$  es la única variable;  $a$  y  $b$  son terminales. Existen dos producciones,  $S \rightarrow aSb$  y  $S \rightarrow ab$ . Mediante la aplicación de la primera producción  $n - 1$  veces, seguida por una aplicación de la segunda producción, tenemos

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow a^3Sb^3 \Rightarrow \cdots \Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^n b^n.$$

Aún más, las únicas cadenas de  $L(G)$  son  $a^n b^n$  para  $n \geq 1$ . Cada vez que  $S \rightarrow aSb$  se utiliza, el número de  $S$ s permanece igual. Después de utilizar la producción  $S \rightarrow ab$  encontramos que el número de  $S$ s en la forma oracional disminuye en uno. Por consiguiente, después de utilizar  $S \rightarrow ab$  ninguna  $S$  permanece en la cadena resultante. Puesto que ambas producciones tienen una  $S$  a la izquierda, el único orden en el cual las producciones pueden aplicarse es  $S \rightarrow aSb$ , algún número de veces seguida por la aplicación de  $S \rightarrow ab$ . Así pues,  $L(G) = \{a^n b^n \mid n \geq 1\}$ .

El Ejemplo 4.2 es un ejemplo simple de una gramática. Fue relativamente fácil determinar qué palabras eran derivables y cuáles no. En general, puede ser excesivamente duro determinar qué es generado por la gramática. Presentamos aquí otro ejemplo, un poco más difícil.

**Ejemplo 4.3** Considérese  $G = (V, T, P, S)$ , en la que  $V = \{S, A, B\}$ ,  $T = \{a, b\}$  y  $P$  consiste en lo siguiente:

$$\begin{array}{ll} S \rightarrow aB & A \rightarrow bAA \\ S \rightarrow bA & B \rightarrow b \end{array}$$

$$\begin{array}{ll} A \rightarrow a & B \rightarrow bS \\ A \rightarrow aS & B \rightarrow aBB \end{array}$$

El lenguaje  $L(G)$  es el conjunto de todas las palabras que pertenecen a  $T^*$  y que consisten en un número igual de  $a$ s y  $b$ s. Probaremos esta proposición por inducción sobre la longitud de una palabra.

**Hipótesis inductiva** Para  $w$  en  $T^*$ ,

1.  $S \xrightarrow{*} w$  si y sólo si  $w$  consiste en un número igual de  $a$ s y  $b$ s.
2.  $A \xrightarrow{*} w$  si y sólo si  $w$  tiene una  $a$  más que  $bs$ .
3.  $B \xrightarrow{*} w$  si y sólo si  $w$  tiene una  $b$  más que  $as$ .

Ciertamente, la hipótesis inductiva es verdadera si  $|w| = 1$ , ya que  $A \xrightarrow{*} a$ ,  $B \xrightarrow{*} b$  y ninguna cadena terminal es derivable de  $S$ . También, porque todas las producciones, con excepción de  $A \rightarrow a$  y  $B \rightarrow b$ , aumentan la longitud de una cadena, ninguna cadena de longitud distinta de  $a$  y  $b$  es derivable de  $A$  y  $B$ , respectivamente. También se tiene que ninguna cadena de longitud uno es derivable de  $S$ .

Supóngase que la hipótesis inductiva es verdadera para toda  $w$  de longitud  $k - 1$  o menor. Mostraremos que es verdadera para  $|w| = k$ . Primero, si  $S \xrightarrow{*} w$ , entonces la derivación debe empezar con  $S \rightarrow aB$  o  $S \rightarrow bA$ . En el primer caso,  $w$  es de la forma  $aw_1$ , en la que  $|w_1| = k - 1$  y  $B \xrightarrow{*} w_1$ . Por la hipótesis de inducción, el número de  $b$ 's en  $w_1$  es uno más que el número de  $a$ s, de modo que  $w$  consiste en un número igual de  $a$ s y  $b$ s. Un argumento similar se aplica si la derivación comienza con  $S \rightarrow bA$ .

Debemos, ahora, probar la condición “sólo si” de la parte (1), es decir, si  $|w| = k$  y  $w$  consiste en un número igual de  $a$ s y  $b$ s, entonces  $S \Rightarrow w$ . El primer símbolo de  $w$  es  $a$  o es  $b$ . Supóngase que  $w = aw_1$ . Ahora  $|w_1| = k - 1$  y  $w_1$  tiene una  $b$  más que  $a$ s. Por la hipótesis de inducción,  $B \xrightarrow{*} w_1$ . Pero entonces  $S \Rightarrow aB \xrightarrow{*} aw_1 = w$ . Un argumento similar se aplica si el primer símbolo de  $w$  es  $b$ .

Nuestra tarea aún no termina. Para completar la prueba, debemos probar las partes (2) y (3) de la hipótesis inductiva para  $w$  de longitud  $k$ . Hacemos esto de una manera similar al método que utilizamos en la prueba de la parte (1); esta parte se deja al lector.

### 4.3 ARBOLES DE DERIVACION

Es útil presentar las derivaciones como árboles. Estos diagramas, conocidos como árboles de *derivación* o de *análisis gramatical*, superponen una estructura sobre las palabras de un lenguaje, que es de utilidad en las aplicaciones tales como la compilación de los lenguajes de programación. Los vértices de un árbol de derivación están etiquetados con símbolos terminales o variables de la gramática, o posiblemente con  $\epsilon$ . Si un vértice interior  $n$  tiene etiqueta  $A$  y los

hijos de  $n$  están etiquetados con  $X_1, X_2, \dots, X_k$  partiendo de la izquierda, entonces  $A \rightarrow X_1 X_2 \dots X_k$  debe ser una producción. La Fig. 4.1 muestra el árbol de análisis gramatical para la derivación (4.3). Nótese que si leemos las hojas, de izquierda a derecha, obtenemos la última línea de (4.3), ( $id + id$ ) \*  $id$ .

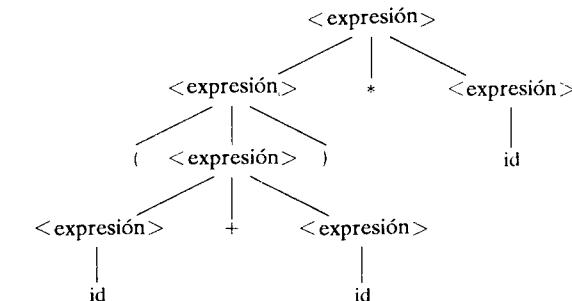


Fig. 4.1 Árbol de derivación.

De manera más formal, sea  $G = (V, T, P, S)$  una CFG. Un árbol es un *árbol de derivación* (o de *análisis gramatical*) para  $G$  si:

1. Cada vértice tiene una *etiqueta*, que es un símbolo de  $V \cup T \cup \epsilon$ .
2. La etiqueta de la raíz es  $S$ .
3. Si un vértice es interior y tiene etiqueta  $A$ , entonces  $A$  debe estar en  $V$ .
4. Si  $n$  tiene etiqueta  $A$  y vértices  $n_1, n_2, \dots, n_k$  son los hijos del vértice  $n$ , de izquierda a derecha, con etiquetas  $X_1, X_2, \dots, X_k$ , respectivamente, entonces

$$A \rightarrow X_1 X_2 \dots X_k$$

debe ser una producción en  $P$ .

5. Si el vértice  $n$  tiene etiqueta  $\epsilon$ , entonces  $n$  es una hoja y es el único hijo de su padre.

**Ejemplo 4.4** Considérese la gramática  $G = (S, A), \{a, b\}, P, S$ , en donde  $P$  consiste en

$$S \rightarrow aAS | a$$

$$A \rightarrow SbA | SS | ba$$

Sólo por esta vez dibujaremos un árbol con círculos en lugar de puntos, representando a los vértices. Los vértices se numerarán para su referencia. Las etiquetas se pondrán junto a los vértices. Véase la Fig. 4.2.

Los vértices interiores son 1, 3, 4, 5 y 7. El vértice 1 tiene etiqueta  $S$ , y sus hijos, a partir de la izquierda, tienen etiquetas  $a$ ,  $A$  y  $S$ . Nótese que  $S \rightarrow aAS$  es una producción. De manera parecida, el vértice 3 tiene etiqueta  $A$ , y las etiquetas de sus hijos son  $S$ ,  $b$  y  $A$ , a partir de la izquierda.  $A \rightarrow SbA$  es también una producción. Los vértices 4 y 5 tienen, cada uno, etiquetas  $S$ . El hijo único de cada

uno de éstos tiene etiqueta  $a$ , y  $S \rightarrow a$  es una producción. Por último, el vértice 7 tiene etiqueta  $A$  y sus hijos, de izquierda a derecha, tienen etiquetas  $b$  y  $a$ .  $A \rightarrow ba$  es también una producción. Por consiguiente, se han alcanzado las condiciones para que la Fig. 4.2 sea un árbol de derivación para  $G$ .

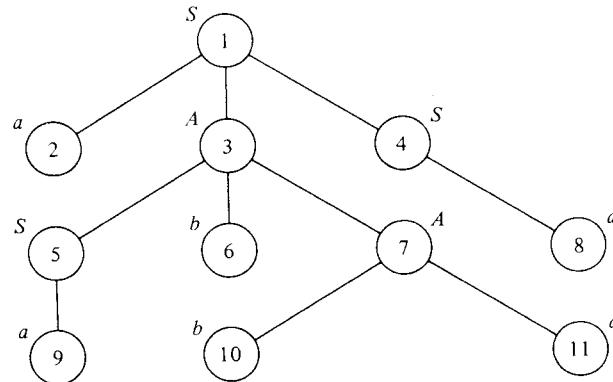


Fig. 4.2 Ejemplo de un árbol de derivación.

Podemos extender el ordenamiento “desde la izquierda” de los hijos para producir un ordenamiento de izquierda a derecha de todas las hojas. De hecho, para cualesquier dos vértices, de los cuales ninguno es ancestro del otro, uno se encuentra a la izquierda del otro. Dados los vértices  $v_1$  y  $v_2$ , sigáse las trayectorias a partir de tales vértices hacia la raíz hasta que se encuentren en algún vértice  $w$ .

Sean  $x_1$  y  $x_2$  los hijos de  $w$  que están en las trayectorias que parten de  $v_1$  y  $v_2$ , respectivamente. Si  $v_1$  no es ancestro de  $v_2$ , o viceversa, entonces  $x_1 \neq x_2$ . Supóngase que  $x_1$  está a la izquierda de  $x_2$  en el ordenamiento de los hijos de  $w$ . Entonces  $v_1$  está a la izquierda de  $v_2$ . En el caso opuesto,  $v_2$  está a la izquierda de  $v_1$ . Por ejemplo, si  $v_1$  y  $v_2$  son los vértices 9 y 10 de la Fig. 4.2, entonces  $w$  es 3,  $x_1 = 5$  y  $x_2 = 7$ . Como 5 está a la izquierda de 7, se concluye que 9 está a la izquierda de 11.

Veremos que un árbol de derivación es una descripción natural de una forma oracional particular de la gramática  $G$ . Si leemos las etiquetas de las hojas de izquierda a derecha, tendremos una forma oracional. Llamamos a esta cadena el *producto* del árbol de derivación. Más adelante, veremos que si  $\alpha$  es el producto de algún árbol de derivación para la gramática  $G = (V, T, P, S)$ , entonces  $S \xrightarrow{*} \alpha$ , y viceversa.

Necesitamos un concepto adicional, el de *subárbol*. Un subárbol de un árbol de derivación es un vértice particular del árbol junto con todos sus descendientes, las aristas que los conectan y sus etiquetas. Tiene la apariencia de un árbol de derivación, excepto que la etiqueta de la raíz puede no ser el símbolo

inicial de la gramática. Si la variable  $A$  es la etiqueta de la raíz, entonces llamamos al subárbol *árbol A*. Por consiguiente “árbol  $S$ ” es un sinónimo de “árbol de derivación”, si  $S$  es el símbolo inicial.

**Ejemplo 4.5** Considérense la gramática y el árbol de derivación del Ejemplo 4.4. El árbol de derivación de la Fig. 4.2 se reproduce sin vértices numerados en la Fig. 4.3 (a). El producto del árbol en la Fig. 4.3 (a) es  $aabbba$ . Refiriéndose de nuevo a la Fig. 4.2, vemos que las hojas son los vértices con número 2, 9, 6, 10, 11 y 8, en ese orden a partir de la izquierda. Estos vértices tienen etiquetas  $a$ ,  $a$   $b$ ,  $b$ ,  $a$ ,  $a$ , respectivamente. Nótese que en este caso todas las hojas tienen terminales como etiquetas, pero no existe una razón por la cual esto deba ser siempre así, algunas hojas podrían tener etiqueta  $\epsilon$  de alguna variable. Nótese que  $S \xrightarrow{*} aabbba$  por la derivación.

$$S \Rightarrow aAs \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbba.$$

La Fig. 4.3 (b) muestra un subárbol del árbol ilustrado en la parte (a). Corresponde al vértice 3 de la Fig. 4.2, junto con sus descendientes. El producto del subárbol es  $abba$ . La etiqueta de la raíz del subárbol es  $A$  y  $A \xrightarrow{*} abba$ . Una derivación, en este caso, es

$$A \Rightarrow SbA \Rightarrow abA \Rightarrow abba.$$

### La relación entre los árboles de derivación y las derivaciones

**Teorema 4.1** Sea  $G = (V, T, P, S)$  una gramática libre de contexto. Entonces  $S \xrightarrow{*} \alpha$  si y sólo si existe un árbol de derivación de la gramática  $G$  con producto  $\alpha$ .

**Demostración** Encontraremos más fácil probar algo superior al teorema. Lo que demostraremos es que para cualquier  $A$  en  $V$ ,  $A \xrightarrow{*} \alpha$  si y sólo si existe un árbol  $A$  con  $\alpha$  como el producto.

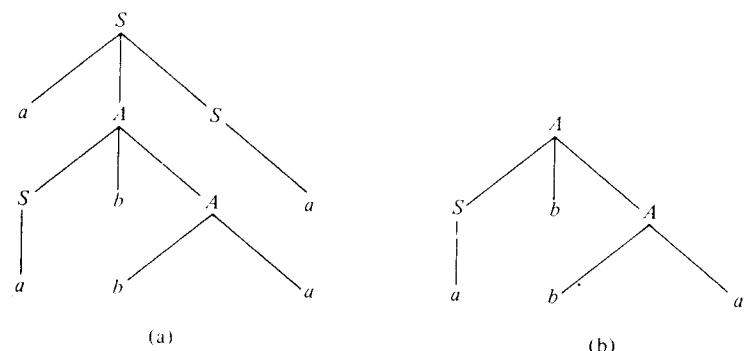


Fig. 4.3 Árbol y subárbol de derivación.

Primero, supóngase que  $\alpha$  es el producto de un árbol  $A$ . Probamos, por inducción sobre el número de vértices interiores del árbol, que  $A \xrightarrow{*} \alpha$ . Si existe sólo un vértice interior, el árbol deberá verse como el que se presenta en la Fig. 4.4. En

ese caso,  $X_1 X_2 \dots X_n$  debe ser  $\alpha$ , y  $A$  debe ser una producción de  $P$ , según la definición de árbol de derivación.

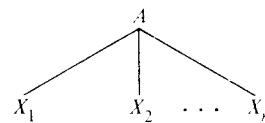


Fig. 4.4 Árbol con un vértice interior.

Ahora, supóngase que el resultado es verdadero para árboles que poseen hasta  $k - 1$  vértices interiores. También supóngase que  $\alpha$  es el producto de un árbol  $A$  con  $k$  vértices interiores para alguna  $k > 1$ . Considérense los hijos de la raíz. No todos pueden ser hojas. Sean las etiquetas de los hijos  $X_1, X_2, \dots, X_n$  de izquierda a derecha. Entonces, con toda seguridad,  $A \rightarrow X_1 X_2 \dots X_n$  es una producción en  $P$ . Nótese que  $n$  puede ser cualquier entero mayor o igual que uno en el siguiente argumento.

Si el  $i$ -ésimo hijo no es una hoja, es la raíz de un subárbol y  $X_i$  debe ser una variable. El subárbol debe ser algún árbol  $X_i$  y tiene algún producto  $\alpha_i$ . Si el vértice  $i$  es una hoja, sea  $\alpha_i = X_i$ . Es fácil ver que si  $j < i$ , el vértice  $j$  y todos sus descendientes se encuentran a la izquierda del vértice  $i$  y de todos sus descendientes. Por consiguiente,  $\alpha = \alpha_1 \alpha_2 \dots \alpha_n$ . Un subárbol debe tener menos vértices interiores que los que tiene su árbol, a menos que el subárbol sea el árbol completo. Por la hipótesis de inducción, para cada vértice  $i$  que no sea una hoja,  $X_i \xrightarrow{*} \alpha_i$ , ya que el subárbol con raíz  $X_i$  no es el árbol completo. Si  $X_i = \alpha_i$ , entonces, con toda seguridad,  $X_i \xrightarrow{*} \alpha_i$ . Podemos juntar todas estas derivaciones parciales para ver que

$$A \Rightarrow X_1 X_2 \dots X_n \xrightarrow{*} \alpha_1 X_2 \dots X_n \xrightarrow{*} \alpha_1 \alpha_2 X_3 \dots X_n \xrightarrow{*} \dots \xrightarrow{*} \alpha_1 \alpha_2 \dots \alpha_n = \alpha. \quad (4.4)$$

Por consiguiente,  $A \xrightarrow{*} \alpha$ . Adviértase que (4.4) es sólo una de muchas derivaciones posibles que se pueden producir a partir de un árbol de análisis gramatical dado.

Ahora supóngase que  $A \xrightarrow{*} \alpha$ . Debemos mostrar que existe un árbol  $A$  con producto  $\alpha$ . Si  $A \xrightarrow{*} \alpha$  mediante un solo paso, entonces  $A \rightarrow \alpha$  es una producción de  $P$ , y existe un árbol con producto  $\alpha$ , de la forma que se presenta en la Fig. 4.4.

Supongamos, ahora, que para cualquier variable,  $A$  si  $A \xrightarrow{*} \alpha$  mediante una derivación con menos de  $k$  pasos, entonces existe un árbol  $A$  con producto  $\alpha$ . Supóngase que  $A \xrightarrow{*} \alpha$  mediante una derivación de  $k$  pasos. Sea el primer paso  $A \rightarrow X_1 X_2 \dots X_n$ . Deberá estar, claro que cualquier símbolo de  $\alpha$  debe ser uno de  $X_1, X_2, \dots, X_n$  o ser derivado de uno de estos. También, la porción de  $\alpha$  derivada de  $X_i$ , debe estar a la izquierda de los símbolos derivados de  $X_j$ , si  $i < j$ . Por tanto, podemos escribir  $\alpha$  como  $\alpha_1 \alpha_2 \dots \alpha_n$ , en donde, para cada  $i$  entre 1 y  $n$ ,

1.  $\alpha_i = X_i$  si  $X_i$  es una terminal y
2.  $X_i \xrightarrow{*} \alpha_i$  si  $X_i$  es una variable.

Si  $X_i$  es una variable, entonces la derivación de  $\alpha_i$  de  $X_i$  debe tomar menos de  $k$  pasos, ya que la derivación completa  $A \xrightarrow{*} \alpha$  toma  $k$  pasos y definitivamente el primero no es parte de la derivación  $X_i \xrightarrow{*} \alpha_i$ . Por tanto, por la hipótesis inductiva, para cada  $X_i$  que sea una variable, existe un árbol  $X_i$  con producto  $\alpha_i$ . Sea este árbol  $T_i$ .

Comenzamos construyendo un árbol  $A$  con  $n$  hojas etiquetadas  $X_1, X_2, \dots, X_n$  y sin ningún otro vértice. Este árbol se muestra en la Fig. 4.5 (a). Cada vértice con etiqueta  $X_i$ , en donde  $X_i$  no es terminal, se reemplaza por el árbol  $T_i$ . Si  $X_i$  es terminal, no se hacen sustituciones. En la Fig. 4.5 (b) aparece un ejemplo. El producto de este árbol es  $\alpha$ .  $\square$

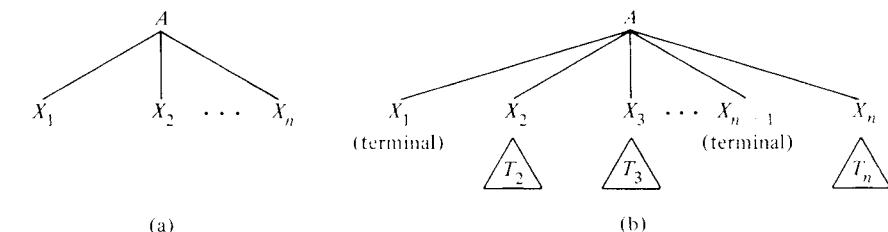


Fig. 4.5 Árboles de derivación.

**Ejemplo 4.6** Considérese la derivación  $S \xrightarrow{*} aabbaa$  del Ejemplo 4.5. El primer paso es  $S \rightarrow aAS$ . Si seguimos la derivación, vemos que  $A$  finalmente es sustituida por  $SbA$ , después por  $abA$  y finalmente por  $abba$ . La Fig. 4.3 (b) representa un árbol de análisis gramatical para esta derivación. El único símbolo derivado de  $S$  en  $aAS$  es  $a$ . (Esta sustitución representa el último paso.) La Fig. 4.6 (a) es un árbol para la última derivación.

La Fig. 4.6 muestra el árbol de derivación para  $S \rightarrow aAS$ . Si sustituimos los vértices con etiqueta  $A$ , de la Fig. 4.6 (b), por el árbol de la Fig. 4.3 (b) y el vértice con etiqueta  $S$ , de la Fig. 4.6 (b), con el árbol de la Fig. 4.6 (a), obtendremos el árbol de la Fig. 4.3 (a), cuyo producto es  $aabbaa$ .

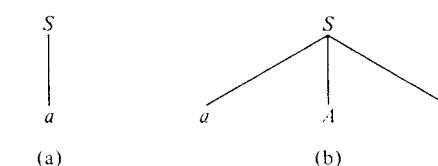


Fig. 4.6 Árboles de derivación.

## Derivación a la extrema izquierda y a la extrema derecha; ambigüedad

Si en cada paso de una derivación se aplica una producción a la variable que se encuentra más a la izquierda, entonces la derivación es extrema izquierda. De manera similar, se dice que una derivación, en la que la variable que está más a la derecha se sustituye en cada paso, es extrema derecha. Si  $w$  está en  $L(G)$  para la CFG  $G$ , entonces  $w$  tiene al menos un árbol de análisis gramatical, y correspondiendo a un árbol de análisis gramatical particular,  $w$  tiene una derivación izquierda y una derecha únicas. En la demostración del Teorema 4.1, la derivación de  $\alpha$  de  $A$  correspondiente al árbol en cuestión es extrema izquierda, siempre y cuando las derivaciones  $X_i \xrightarrow{*} \alpha_i$  sean extremas izquierdas. Si en lugar de la derivación (4.4) hacemos (de manera recursiva) la derivación  $X_i \xrightarrow{*} \alpha_i$  extrema derecha y sustituimos las  $X_i$  por  $\alpha_i$  por la derecha más que por la izquierda, podríamos obtener la derivación derecha correspondiente al árbol de análisis gramatical.

Por supuesto,  $w$  puede tener varias derivaciones derechas o izquierdas, ya que puede haber más de un árbol de derivación para  $w$ . Sin embargo, es fácil mostrar que de cada árbol de derivación, sólo se pueden obtener una derivación extrema izquierda y una extrema derecha. También, la construcción del Teorema 4.1 produce árboles de derivación a partir de diferentes derivaciones extremas izquierdas o extremas derechas.

**Ejemplo 4.7** La derivación extrema izquierda correspondiente al árbol de la Fig. 4.3 (a) es

$$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa.$$

La correspondiente derivación derecha es

$$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbaa.$$

Una gramática libre de contexto  $G$  de la que alguna palabra tenga dos árboles de análisis gramatical se dice que es ambigua. A partir de lo que hemos dicho más arriba, una definición equivalente de ambigüedad la constituye el hecho de que alguna palabra tenga más de una derivación extrema izquierda o más de una extrema derecha. Un CFL para el cual cada CFG es ambiguo se dice que es un CFL inherentemente ambiguo. En la Sección 4.7 mostraremos que los CFL inherentemente ambiguos existen.

## 4.4 SIMPLIFICACIÓN DE GRAMÁTICAS LIBRES DE CONTEXTO

Existen varias formas en las que se puede restringir el formato de las producciones sin reducir el poder generativo de las gramáticas libres de contexto. Si  $L$  es un lenguaje libre de contexto entonces puede ser generado por una gramática libre de contexto  $G$  que posee las propiedades siguientes.

1. Cada variable y cada terminal de  $G$  aparece en la derivación de alguna palabra de  $L$ .
2. No existen producciones de la forma  $A \rightarrow B$  en las que  $A$  y  $B$  sean variables.

Aún más, si  $\epsilon$  no está en  $L$ , es necesario que no haya producciones de la forma  $A \rightarrow \epsilon$ . De hecho, si  $\epsilon$  no está en  $L$ , podemos necesitar que cada producción de  $G$  sea de una de las formas  $A \rightarrow BC$  y  $A \rightarrow a$ , en donde  $A, B$  y  $C$  son variables cualesquiera y  $a$  es una terminal cualquiera. De manera alternativa, podemos hacer que cada producción de  $G$  sea de la forma  $A \rightarrow a\alpha$ , en donde  $\alpha$  es una cadena de variables (tal vez vacías). Estas dos formas especiales se conocen como forma normal de Chomsky y forma normal de Greibach, respectivamente.

### Símbolos inútiles

Ahora nos abocaremos a la tarea de eliminar los símbolos inútiles de una gramática. Sea  $G = (V, T, P, S)$  una gramática. Un símbolo  $X$  es *útil* si existe una derivación  $S \xrightarrow{*} \alpha$   $X \beta \xrightarrow{*} w$  para algunas  $\alpha, \beta$  y  $w$ , en las que  $w$  está en  $T^*$  (recuérdese la regla que ya hemos visto con respecto a los nombres de los símbolos y cadenas). De otra manera  $X$  es *inútil*. Existen dos aspectos de la utilidad. Primero, algunas cadenas terminales deben ser derivables de  $X$ , y segundo,  $X$  debe ocurrir en alguna cadena derivable de  $S$ . Estas dos condiciones no son, sin embargo, suficientes para garantizar que  $X$  es útil, ya que  $X$  puede ocurrir sólo en formas oracionales que contengan una variable de la cual no se pueda derivar ninguna cadena terminal.

**Lema 4.1** Dada una CFG  $G = (V, T, P, S)$ , con  $L(G) \neq \emptyset$ , podemos efectivamente encontrar una CFG equivalente  $G' = (V, T, P', S)$  tal que para cada  $A$  en  $V$  exista alguna  $w$  en  $T^*$  para la cual  $A \xrightarrow{*} w$ .

**Demostración** Cada variable  $A$  con producción  $A \rightarrow w$  en  $P$  claramente pertenece a  $V'$ . Si  $A \rightarrow X_1 X_2 \dots X_n$  es una producción, en la que cada  $X_i$  es una terminal o una variable que ya está situada en  $V'$ , entonces una cadena terminal puede derivarse de  $A$  mediante una derivación que comience  $A \Rightarrow X_1 X_2 \dots X_n$  y, por consiguiente,  $A$  pertenece a  $V'$ . El conjunto  $V'$  puede calcularse mediante un sencillo algoritmo iterativo.  $P'$  es el conjunto de todas las producciones cuyos símbolos están en  $V' \cup T$ .

El algoritmo de la Fig. 4.7 encuentra todas las variables  $A$  que pertenecen a  $V'$ . Seguramente, si  $A$  se agrega a NEWV en la línea (2) o en la (5), entonces  $A$  deriva una cadena terminal. Para mostrar que NEWV no es demasiado pequeña, deberemos mostrar que si  $A$  deriva una cadena terminal  $w$ , entonces  $A$  se añade finalmente a NEWV. Hacemos esto por inducción sobre la longitud de la derivación  $A \xrightarrow{*} w$ .

**Base** Si la longitud es uno, entonces  $A \rightarrow w$  es una producción y  $A$  se agrega a NEWV en el paso (2).

**Inducción** Sea  $A \xrightarrow{*} X_1 X_2 \dots X_n \xrightarrow{*} w$  mediante una derivación en  $k$  pasos. Entonces podemos escribir  $w = w_1 w_2 \dots w_n$ , en donde  $X_i \xrightarrow{*} w_i$  para  $1 \leq i \leq n$ , mediante una derivación con menos de  $k$  pasos. Por la hipótesis de inducción,

```

begin
1.   OLDV := ∅;
2.   NEWV := {A | A → w para alguna w en T*};
3.   while OLDV ≠ NEWV do
        begin
            OLDV := NEWV;
            NEWV := OLDV ∪ {A | A → α para alguna α en (T ∪ OLDV)*}
        end;
    V' := NEWV
end

```

Fig. 4.7 Cálculo de  $V'$ .

aquellas  $X_i$  que son variables finalmente se agregan a NEWV. En la prueba del ciclo *while* en la línea (3), inmediatamente después que la última  $X_i$  se ha añadido a NEWV, no podemos tener  $NEWV = OLDV$  porque la última de estas  $X_i$  no está en OLDV. Por tanto el ciclo while se repite al menos una vez más, y  $A$  será agregada a NEWV en la línea (5).

Tómese  $V'$  como el conjunto calculado en la línea (6) y  $P'$  como todas las producciones cuyos símbolos están en  $V' \cup T$ . Seguramente  $G' = (V', T, P', S)$  satisface la propiedad de que si  $A$  está en  $V'$ , entonces  $A \xrightarrow{*} w$  para alguna  $w$ . También, puesto que cada derivación de  $G'$  es una derivación de  $G$ , sabemos que  $L(G') \subseteq L(G)$ . Pero si existe alguna  $w$  en  $L(G)$  que no esté en  $L(G')$ , entonces cualquier derivación de  $w$  en  $G$  debe involucrar una variable que esté en  $V - V'$  o una producción de  $P - P'$  (lo que implica que existe una variable en  $V - V'$  que ha sido utilizada). Pero entonces existe una variable de  $V - V'$  que deriva una cadena terminal, lo que significa una contradicción.  $\square$

**Lema 4.2** Dada una CFG  $G = (V, T, P, S)$  podemos encontrar efectivamente una CFG  $G' = (V', T', P', S)$  equivalente tal que para cada  $X$  en  $V' \cup T'$  existan  $\alpha$  y  $\beta$  en  $(V \cup T')^*$  para las que  $S \xrightarrow{*} \alpha X \beta$ .

**Demostración** El conjunto  $V' \cup T'$  de símbolos que aparecen en las formas oracionales de  $G$  se construye mediante un algoritmo iterativo. Colóquese  $S$  en  $V'$ . Si  $A$  está colocado en  $V'$  y  $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$ , entonces agreguense todas las variables de  $\alpha_1, \alpha_2, \dots, \alpha_n$ , al conjunto  $V'$  y todas las terminales de  $\alpha_1, \alpha_2, \dots, \alpha_n$  a  $T'$ .  $P'$  es el conjunto de producciones de  $P$  que contienen solo símbolos de  $V' \cup T'$ .  $\square$

Aplicando, primero, el Lema 4.1 y, después, el Lema 4.2, podemos convertir una gramática a una equivalente que no contenga símbolos inútiles. Es interesante notar que aplicando el Lema 4.2, primero, y el Lema 4.1 después, tal vez no se eliminan los símbolos inútiles.

**Teorema 4.2** Cada CFL no vacío es generado por una CFG que no posee símbolos inútiles.

**Demostración** Sea  $L = L(G)$  un CFL no vacío. Sean  $G_1$  el resultado de aplicar la construcción del Lema 4.1 a  $G$ , y  $G_2$ , el resultado de aplicar la construcción del Lema 4.2 a  $G_1$ . Supóngase que  $G_2$  tiene un símbolo inútil  $X$ . Según el Lema 4.2, existe una derivación  $S \xrightarrow{G_2} \alpha X \beta$ . Puesto que todos los símbolos de  $G_2$  son símbolos de  $G_1$ , se concluye del Lema 4.1 que  $S \xrightarrow{G_1} \alpha X \beta \xrightarrow{G_1} w$  para alguna cadena terminal  $w$ . Por consiguiente, ningún símbolo de la derivación  $\alpha X \beta \xrightarrow{G_1} w$  se elimina según el Lema 4.2. En consecuencia,  $X$  deriva una cadena terminal de  $G_2$  y, de aquí que,  $X$  no es inútil como se había supuesto.  $\square$

**Ejemplo 4.8** Considerérese la gramática

$$\begin{aligned} S &\rightarrow AB|a \\ A &\rightarrow a \end{aligned} \quad (4.5)$$

Aplicando el Lema 4.1, encontramos que ninguna cadena terminal es derivable de  $B$ . Por tanto, eliminamos  $B$  y la producción  $S \rightarrow AB$ . Aplicando el Lema 4.2 a la gramática

$$\begin{aligned} S &\rightarrow a \\ A &\rightarrow a \end{aligned} \quad (4.6)$$

encontramos que sólo  $S$  y  $a$  aparecen en las formas oracionales. Por consiguiente  $(\{S\}, \{a\}, \{S \rightarrow a\}, S)$  es una gramática equivalente que no tiene símbolos inútiles.

Supóngase que primero aplicamos el lema 4.2 a (4.5). Podríamos encontrar que todos los símbolos aparecen en formas oracionales. Entonces aplicando el Lema 4.1 nos quedaríamos con (4.6) que tiene un símbolo inútil,  $A$ .

### Producciones $\in$

Pondremos ahora nuestra atención a la eliminación de producciones de la forma  $A \rightarrow \epsilon$ , a las que llamaremos *producciones  $\in$* . Seguramente  $\epsilon$  si está en  $L(G)$ , no podemos eliminar todas las producciones  $\in$  de  $G$ , pero si  $\epsilon$  no está en  $L(G)$ , se hace evidente que podemos eliminarlas. El método consiste en determinar para cada variable  $A$  si  $A \xrightarrow{*} \epsilon$ . Si esto sucede, decimos que  $A$  es *anulable*. Podemos sustituir cada producción  $B \rightarrow X_1 X_2 \dots X_n$  por todas las producciones formadas mediante la eliminación de algún subconjunto de aquellas  $X_i$  que son anulables, pero no incluimos  $B \rightarrow \epsilon$ , aun si todas las  $X_i$  son anulables.

**Teorema 4.3** Si  $L = L(G)$  para alguna CFG  $G = (V, T, P, S)$ , entonces  $L - \{\epsilon\}$  es  $L(G')$  para una CFG  $G'$  que no tiene símbolos inútiles o producciones  $\in$ .

**Demostración** Podemos determinar los símbolos anulables de  $G$  a través del siguiente algoritmo iterativo. Para empezar, si  $A \rightarrow \epsilon$  es una producción, enton-

ces  $A$  es anulable. En seguida, si  $B \rightarrow \alpha$  es una producción y todos los símbolos de  $\alpha$  se han encontrado anulables, entonces  $B$  es anulable. Repetimos este proceso hasta que no se encuentren más símbolos anulables.

El conjunto de producciones de  $P'$  se construye de la manera siguiente. Si  $A \rightarrow X_1 X_2 \dots X_n$  está en  $P$ , entonces agréguese todas las producciones  $A \rightarrow \alpha_1 \alpha_2 \dots \alpha_n P'$  en donde

1. si  $X_i$  no es anulable, entonces  $\alpha_i = X_i$ ;
2. si  $X_i$  es anulable, entonces  $\alpha_i$  es  $X_i$  o  $\epsilon$ ;
3. no todas las  $\alpha_i$  son  $\epsilon$ .

Sea  $G'' = (V, T, P', S)$ . Pedimos que para toda  $A$  en  $V$  y  $w$  en  $T^*$ ,  $A \xrightarrow{G''} w$  si y sólo si  $w \neq \epsilon$  y  $A \xrightarrow{G'} w$ .

Si sea  $A \xrightarrow{G''} w$  y  $w \neq \epsilon$ . Demostramos por inducción en  $i$  que  $A \xrightarrow{G''} w$ . La base,  $i = 1$ , es inmediata, pues  $A \rightarrow w$  debe ser una producción de  $P'$ . Como  $w \neq \epsilon$  también es una producción de  $P'$ . Según el paso inductivo, sea  $i > 1$ . Entonces  $A \xrightarrow{G''} X_1 X_2 \dots X_n \xrightarrow{G''} w$ . Escribábase  $w = w_1 w_2 \dots w_n$  tal que para cada  $j$ ,  $X_j \xrightarrow{G''} w_j$  en menos de  $i$  pasos. Si  $w_j \neq \epsilon$  y  $X_j$  es una variable, entonces por la hipótesis de inducción tenemos que  $X_j \xrightarrow{G''} w_j$ . Si  $w_j = \epsilon$ , entonces  $X_j$  es anulable. Por tanto,  $A \rightarrow \beta_1 \beta_2 \dots \beta_n$  es una producción de  $P'$ , en donde  $\beta_j = X_j$  si  $w_j \neq \epsilon$  y  $\beta_j = \epsilon$  si  $w_j = \epsilon$ . Como  $w \neq \epsilon$ , no todas las  $\beta_j$  son  $\epsilon$ . De aquí que tengamos una derivación en  $G''$ .

$$A \Rightarrow \beta_1 \beta_2 \dots \beta_n \xrightarrow{*} w_1 \beta_2 \dots \beta_n \xrightarrow{*} w_1 w_2 \beta_3 \dots \beta_n \xrightarrow{*} \dots \xrightarrow{*} w_1 w_2 \dots w_n = w$$

en  $G''$ .

Sólo si Supóngase que  $A \xrightarrow{G''} w$ . Seguramente  $w \neq \epsilon$ , ya que  $G''$  no tiene producciones  $\epsilon$ . Mostramos por inducción en  $i$  que  $A \xrightarrow{G''} w$ . Para la base,  $i = 1$ , obsérvese que  $A \rightarrow w$  es una producción de  $P'$ . Debe existir una producción  $A \rightarrow \alpha$  en  $P$  tal que eliminando ciertos símbolos anulables de  $\alpha$  nos quedamos con  $w$ . Entonces existe una derivación  $A \xrightarrow{G} \alpha \xrightarrow{G''} w$ , en la que la derivación  $\alpha \xrightarrow{G''} w$  implica la derivación de  $\epsilon$  a partir de los símbolos anulables de  $\alpha$  que fueron eliminados para conseguir  $w$ .

Para el paso inductivo, hagamos  $i > 1$ . Entonces  $A \xrightarrow{G''} X_1 X_2 \dots X_n \xrightarrow{i-1} w$ . Debe existir alguna  $A \rightarrow \beta$  en  $P$  tal que  $X_1 X_2 \dots X_n \xrightarrow{*} \beta$  se encuentra eliminando algunos símbolos anulables de  $\beta$ . Por consiguiente  $A \xrightarrow{*} X_1 X_2 \dots X_n$ . Escribábase  $w = w_1 w_2 \dots w_n$ , tal que para toda  $j$ ,  $X_j \xrightarrow{G''} w_j$  mediante menos de  $i$  pasos. Por la hipótesis de inducción,  $X_j \xrightarrow{*} w_j$  si  $X_j$  es una variable. Ciertamente, si  $X_j$  es una terminal entonces  $w_j = X_j$  y  $X_j \xrightarrow{*} w_j$  es trivialmente verdadera. En consecuencia  $A \xrightarrow{*} w$ .

El último paso de la demostración consiste en aplicar el Teorema 4.2 a  $G'$  para producir  $G'$  sin símbolos inútiles. Ya que la construcción de los Lemas 4.1 y 4.2 no introducen ninguna producción,  $G'$  no tiene ni símbolos anulables ni inútiles. Aún más,  $S \xrightarrow{*} w$  si y sólo si  $w \neq \epsilon$  y  $S \xrightarrow{*} w$ . Esto es,  $L(G') = L(G) - \{\epsilon\}$ .  $\square$

De aquí en adelante supondremos que ninguna gramática tiene símbolos inútiles. Ahora pondremos nuestra atención a las producciones de la forma  $A \rightarrow B$  cuyo lado derecho consiste en una sola variable. Llamamos a estas producciones *unitarias*. Todas las otras producciones, incluyendo las de la forma  $A \rightarrow a$  y las producciones  $\epsilon$ , son producciones *no unitarias*.

**Teorema 4.4** Cada CFL sin  $\epsilon$  está definido por una gramática que no tiene símbolos inútiles, producciones  $\epsilon$ , o producciones unitarias.

**Demostración** Sea  $L$  un CFL sin  $\epsilon$  y  $L = L(G)$  para alguna  $G = (V, T, P, S)$ . Según el Teorema 4.3, supóngase que  $G$  no tiene producciones  $\epsilon$ . Constrúyase un nuevo conjunto de producciones  $P'$  de  $P$ , mediante la inclusión, primero, de todas las producciones no unitarias de  $P$ . Entonces, supóngase que  $A \xrightarrow{*} B$ , para  $A$  y  $B$  de  $V$ . Agréguese a  $P'$  todas las producciones de la forma  $A \rightarrow \alpha$ , en donde  $B \rightarrow \alpha$  es una producción no unitaria de  $P$ .

Observe que podemos probar de manera sencilla si  $A \xrightarrow{*} B$ , ya que  $G$  no tiene producciones  $\epsilon$ , y si

$$A \xrightarrow{G} B_1 \xrightarrow{G} B_2 \xrightarrow{G} \dots \xrightarrow{G} B_m \xrightarrow{G} B,$$

y algunas variables aparecen dos veces en la secuencia, podemos encontrar una secuencia más corta de producciones unitarias que traen como resultado que  $A \xrightarrow{G} B$ . Por tanto es suficiente considerar aquellas secuencias de producciones unitarias que no repiten ninguna de las variables de  $G$ .

Ahora tenemos una gramática modificada,  $G' = (V, T, P', S)$ . Seguramente, si  $A \rightarrow \alpha$  es una producción de  $P'$ , entonces  $A \xrightarrow{*} \alpha$ . Por consiguiente, si existe una derivación de  $w$  en  $G'$ , entonces existe una derivación de  $w$  en  $G$ .

Supóngase que  $w$  está en  $L(G)$ , y considérese una derivación extrema izquierda de  $w$  en  $G$ , digamos

$$S = \alpha_0 \xrightarrow{G} \alpha_1 \xrightarrow{G} \dots \xrightarrow{G} \alpha_n = w.$$

Si, para  $0 \leq i < n$ ,  $\alpha_i \xrightarrow{G} \alpha_{i+1}$  a través de una producción no unitaria, entonces  $\alpha_i \xrightarrow{*} \alpha_{i+1}$ . Supóngase que  $\alpha_i \xrightarrow{*} \alpha_{i+1}$  a través de una producción unitaria, pero que  $\alpha_{i-1} \xrightarrow{*} \alpha_i$  mediante una producción no unitaria, o  $i = 0$ . También supóngase que  $\alpha_{i+1} \xrightarrow{*} \alpha_{i+2} \xrightarrow{*} \dots \xrightarrow{*} \alpha_j$ , todo mediante producciones unitarias, y  $\alpha_j \xrightarrow{*} \alpha_{j+1}$  mediante una producción no unitaria. Entonces  $\alpha_i, \alpha_{i+1}, \dots, \alpha_j$  son todas de la misma longitud, y como la derivación es extrema izquierda, el símbolo sustituido en cada una de éstas debe estar en la misma posición. Pero entonces  $\alpha_i \xrightarrow{*} \alpha_{j+1}$  mediante una de las producciones de  $P' - P$ . De aquí que  $L(G') = L(G)$ . Para completar la demostración, observamos que  $G'$  no tiene producciones unitarias o producciones  $\epsilon$ . Si utilizamos los Lemas 4.1 y 4.2 para eliminar los símbolos inútiles, no agregamos ninguna producción, de modo que el resultado de aplicar la construcción de tales lemas a  $G'$  es una gramática que satisface el teorema.  $\square$

## 4.5 FORMA NORMAL DE CHOMSKY

Demostramos ahora el primero de dos teoremas de forma normal. Cada uno de éstos establece que todas las gramáticas libres de contexto son equivalentes a gramáticas con restricciones sobre las formas de las producciones.

**Teorema 4.5 (forma normal de Chomsky o CNF)** Cualquier lenguaje libre de contexto sin  $\epsilon$ , es generado por una gramática en la que todas las producciones son de la forma  $A \rightarrow BC$  o  $A \rightarrow a$ . Aquí,  $A, B$  y  $C$  son variables y  $a$  es una terminal.

**Demostración** Sea  $G$  una gramática libre de contexto que genera un lenguaje que no contiene  $a \in \epsilon$ . Por el Teorema 4.4, podemos encontrar una gramática equivalente,  $G_1 = (V, T, P, S)$ , tal que  $P$  no contenga producciones unitarias o producciones  $\epsilon$ . Por tanto, si una producción tiene un símbolo simple a la derecha, ese símbolo es una terminal, y la producción se halla ya en una forma aceptable.

Ahora considérese una producción  $P$  de la forma  $A \rightarrow X_1 X_2 \dots X_m$ , en donde  $m \geq 2$ . Si  $X_i$  es una terminal,  $a$ , introduzcanse una nueva variable  $C_a$  y una producción  $C_a \rightarrow a$ , que es una forma permisible. Entonces sustituya  $X_i$  por  $C_a$ . Sea el nuevo conjunto de variables  $V'$  y sea el nuevo conjunto de producciones  $P'$ . Considerese la gramática  $G_2 = (V', T, P', S)$ . † Si  $\alpha \xrightarrow{G_1} \beta$ , entonces  $\alpha \xrightarrow{G_2} \beta$ . Por tanto  $L(G_1) \subseteq L(G_2)$ . Ahora mostraremos por inducción sobre el número de pasos de una derivación que si  $A \xrightarrow{G_2} w$ , para  $A$  en  $V$  y  $w$  en  $T^*$ , entonces  $A \xrightarrow{G_1} w$ . El resultado es trivial para derivaciones de un paso. Supóngase que es verdadero para derivaciones de hasta  $k$  pasos. Sea  $A \xrightarrow{G_2} w$  una derivación ( $k + 1$ ) pasos. El primer paso debe ser de la forma  $A \xrightarrow{G_2} B_1 B_2 \dots B_m$ ,  $m \geq 2$ . Podemos escribir  $w = w_1 w_2 \dots w_m$ , en donde  $B_i \xrightarrow{G_2} w_i$ ,  $1 \leq i \leq m$ .

Si  $B_i$  es  $C_{a_i}$  para alguna terminal  $a_i$ , entonces  $w_i$  debe ser  $a_i$ . Por la construcción de  $P'$ , existe una producción  $A \rightarrow X_1 X_2 \dots X_m$  de  $P$  en la que  $X_i = B_i$  si  $B_i$  está en  $V$  y  $X_i = a_i$  si  $B_i$  está en  $V - V$ . Para estas  $B_i$  en  $V$ , sabemos que la derivación  $B_i \xrightarrow{G_2} w_i$  no toma más de  $k$  pasos, de modo que por la hipótesis inductiva  $X_i \xrightarrow{G_1} w_i$ . En consecuencia,  $A \xrightarrow{G_1} w$ .

Hemos demostrado el resultado intermedio que consiste en que cualquier lenguaje libre de contexto puede ser generado por una gramática para la cual cada producción es de la forma  $A \rightarrow a$  o  $A \rightarrow B_1 B_2 \dots B_m$ , para  $m \geq 2$ . Aquí  $A$  y  $B_1, B_2, \dots, B_m$  son variables, y  $a$  es una terminal.

Considerese dicha gramática  $G_2 = (V', T, P', S)$ . Modificamos  $G_2$  mediante la adición de algunos símbolos más a  $V'$  y sustituyendo algunas producciones de  $P'$ . Para cada producción  $A \rightarrow B_1 B_2 \dots B_m$  de  $P'$ , en donde  $m \geq 3$ , creamos nuevas variables  $D_1, D_2, \dots, D_{m-2}$  y sustituimos  $A \rightarrow B_1 B_2 \dots B_m$  por el conjunto de producciones.

$$\{A \rightarrow B_1 D_1, D_1 \rightarrow B_2 D_2, \dots, D_{m-3} \rightarrow B_{m-2} D_{m-2}, D_{m-2} \rightarrow B_{m-1} B_m\}.$$

† Nótese que  $G_2$  aún no está en la forma normal de Chomsky.

Sean  $V''$  el nuevo vocabulario no terminal y  $P''$  el nuevo conjunto de producciones. Sea  $G_3 = (V'', T, P'', S)$ .  $G_3$  está en CNF. Es claro que si  $A \xrightarrow{G_2} \beta$ , entonces  $A \xrightarrow{G_3} \beta$  de manera que  $L(G_2) \subseteq L(G_3)$ . Pero también es verdad que  $L(G_3) \subseteq L(G_2)$ , como puede mostrarse, esencialmente, de la misma manera en que se mostró que  $L(G_2) \subseteq L(G_1)$ . La demostración se dejará al lector. □

**Ejemplo 4.9** Consideremos la gramática  $(\{S, A, B\}, \{a, b\}, P, S)$  que tiene las producciones:

$$\begin{aligned} S &\rightarrow bA \mid aB \\ A &\rightarrow bAA \mid aS \mid a \\ B &\rightarrow aBB \mid bS \mid b \end{aligned}$$

y encontraremos una gramática equivalente en CNF.

Primero, las únicas producciones que ya se encuentran en forma apropiada son  $A \rightarrow a$  y  $B \rightarrow b$ . No existen producciones unitarias, así que podemos empezar sustituyendo las terminales de la derecha por variables, excepto en el caso de las producciones  $A \rightarrow a$  y  $B \rightarrow b$ .  $S \rightarrow bA$  se sustituye por  $S \rightarrow C_b A$  y  $C_b \rightarrow b$ . De manera similar,  $A \rightarrow aS$  se sustituye por  $A \rightarrow C_a S$  y  $C_a \rightarrow a$ ;  $A \rightarrow bAA$  es sustituida por  $A \rightarrow C_b AA$ ;  $S \rightarrow bS$  se reemplaza por  $S \rightarrow C_b B$ ;  $B \rightarrow bS$  es reemplazado por  $B \rightarrow C_b S$ ; y  $B \rightarrow aBB$  es sustituida por  $B \rightarrow C_a BB$ .

En la siguiente etapa, la producción  $A \rightarrow C_b AA$  se sustituye por  $A \rightarrow C_b D_1$  y  $D_1 \rightarrow AA$  y la producción  $B \rightarrow C_a BB$  se reemplaza por  $B \rightarrow C_a D_2$  y  $D_2 \rightarrow BB$ . Las producciones para la gramática de CNF se muestran a continuación.

$$\begin{array}{ll} S \rightarrow C_b A \mid C_a B & D_1 \rightarrow AA \\ A \rightarrow C_a S \mid C_b D_1 \mid a & D_2 \rightarrow BB \\ B \rightarrow C_b S \mid C_a D_2 \mid b & C_a \rightarrow a \\ & C_b \rightarrow b \end{array}$$

## 4.6 FORMA NORMAL DE GREIBACH

Ahora desarrollaremos un teorema de forma normal que utiliza producciones cuyo lado derecho comienza con un símbolo terminal seguido, tal vez, por algunas variables. Primeramente demostraremos dos lemas que establecen que podemos modificar las producciones de una CFG de cierta forma sin afectar al lenguaje generado.

**Lema 4.3** Defínase una producción  $A$  como una producción con variable  $A$  en la izquierda. Sea  $G = (V, T, P, S)$  una CFG. Sea  $A \rightarrow \alpha_1 B \alpha_2$  una producción de  $P$  y sea  $B \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_r$  el conjunto de todas las producciones  $B$ . Hagamos  $G_1 = (V, T, P_1, S)$  la gramática obtenida de  $G$  mediante la eliminación de la

producción  $A \rightarrow \alpha_1 B \alpha_2$  de  $P$  y la adición de las producciones  $A \rightarrow \alpha_1 \beta_1 \alpha_2 \mid \alpha_1 \beta_1 \alpha_2 \mid \dots \mid \alpha_1 \beta_r \alpha_2$ . Entonces  $L(G) = L(G_1)$ .

**Demostración** Obviamente  $L(G_1) \subseteq L(G)$ , ya que si  $A \rightarrow \alpha_1 \beta_i \alpha_2$  se utiliza en una derivación de  $G_1$ , entonces  $A \xrightarrow{G_1} \alpha_1 B \alpha_2 \xrightarrow{G_1} \alpha_1 \beta_i \alpha_2$  puede utilizarse en  $G$ . Para mostrar que  $L(G) \subseteq L(G_1)$ , notamos simplemente que  $A \rightarrow \alpha_1 B \alpha_2$  es la única producción de  $G$  que no está en  $G_1$ . Cuando  $A \rightarrow \alpha_1 B \alpha_2$  se utiliza en una derivación mediante  $G$ , la variable  $B$  debe ser reescrita en algún paso posterior utilizando una producción de la forma  $B \rightarrow \beta_i$ . Estos dos pasos pueden sustituirse por el paso simple  $A \xrightarrow{G_1} \alpha_1 \beta_i \alpha_2$ .

**Lema 4.4** Sea  $G = (V, T, P, S)$  una CFG. Sea  $A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_s$  el conjunto de las producciones  $A$  para las cuales  $A$  es el símbolo que está más a la izquierda del lado derecho. Sean  $A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_r$  las restantes producciones  $A$ . Hagamos  $G_1 = (V \cup \{B\}, T, P_1, S)$  la CFG formada por la adición de la variable  $B$  a  $V$  y la sustitución de todas las producciones  $A$  por las producciones:

$$\begin{aligned} 1 \quad & A \rightarrow \beta_i \\ A \rightarrow \beta_i B \end{aligned} \left\{ \begin{array}{l} 1 \leq i \leq s, \\ 2 \quad B \rightarrow \alpha_i \\ B \rightarrow \alpha_i B \end{array} \right\} \left\{ \begin{array}{l} 1 \leq i \leq r. \end{array} \right.$$

Entonces  $L(G_1) = L(G)$ .

**Demostración** En una derivación izquierda una secuencia de producciones de la forma  $A \rightarrow A\alpha_i$  debe terminar, finalmente, con una producción  $A \rightarrow \beta_j$ . La secuencia de sustituciones

$$\begin{aligned} A \Rightarrow A\alpha_{i_1} \Rightarrow A\alpha_{i_2}\alpha_{i_1} \Rightarrow \dots \Rightarrow A\alpha_{i_p}\alpha_{i_{p-1}} \dots \alpha_{i_1} \\ \Rightarrow \beta_j \alpha_{i_p} \alpha_{i_{p-1}} \dots \alpha_{i_1} \end{aligned}$$

En  $G$  puede ser sustituida en  $G_1$  por

$$\begin{aligned} A \Rightarrow \beta_j B \Rightarrow \beta_j \alpha_{i_p} B \Rightarrow \beta_j \alpha_{i_p} \alpha_{i_{p-1}} B \\ \Rightarrow \dots \Rightarrow \beta_j \alpha_{i_p} \alpha_{i_{p-1}} \dots \alpha_{i_2} B \\ \Rightarrow \beta_j \alpha_{i_p} \alpha_{i_{p-1}} \dots \alpha_{i_1}. \end{aligned}$$

También se puede hacer la transformación inversa. Por tanto  $L(G) = L(G_1)$ . La Fig. 4.8 muestra esta transformación sobre árboles de derivación, en las que vemos que en  $G_1$  una cadena de  $A$ s que se extiende hacia la izquierda es sustituida por una cadena de  $B$ s que se extienden a la derecha  $\square$

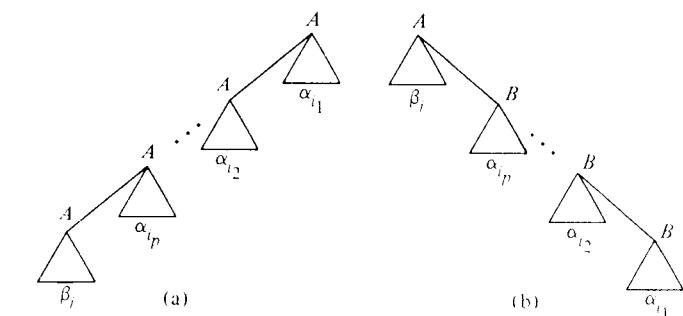


Fig. 4.8 Transformación del Lema 4.4 sobre una parte de un árbol de derivación.

**Teorema 4.6 (Forma normal de Greibach o GNF)** Cada lenguaje libre de contexto  $L$  que no contenga  $\epsilon$ , puede ser generado por una gramática para la cual cada producción es de la forma  $A \rightarrow a\alpha$ , en donde  $A$  es una variable,  $a$  una terminal y  $\alpha$  una cadena (posiblemente vacía) de variables.

**Demostración** Sea  $G = (V, T, P, S)$  una gramática forma normal de Chomsky que genera al CFL  $L$ . Tómese  $V = \{A_1, A_2, \dots, A_m\}$ . El primer paso de la construcción consiste en modificar las producciones de modo que si  $A_i \rightarrow A_j$ ,  $\gamma$  es una producción, entonces  $j > i$ . Comenzando con  $A_1$  y continuando hasta  $A_m$ , hacemos lo anterior como sigue. Suponemos que las producciones han sido modificadas de forma que para  $1 \leq i \leq k$ ,  $A_i \rightarrow A_j$ ,  $\gamma$  es una producción sólo si  $j > i$ . Ahora modificaremos las producciones  $A_k$ .

Si  $A_k \rightarrow A_\ell$ ,  $\gamma$  es una producción con  $j < k$ , generaremos un nuevo conjunto de producciones mediante la sustitución por  $A_j$  del lado derecho de cada producción  $A_\ell$  de acuerdo con el Lema 4.3. Repitiendo el proceso cuando mucho  $k - 1$  veces, obtenemos producciones de la forma  $A_k \rightarrow A_\ell \gamma$ ,  $1 \geq \ell \geq k$ . Las producciones con  $\ell = k$  se sustituyen entonces de acuerdo con el Lema 4.4, introduciendo una nueva variable  $B_k$ . En la Fig. 4.9 se da el algoritmo preciso.

```

begin
  for k := 1 to m do
    begin
      for j := 1 to k - 1 do
        for cada producción de la forma  $A_k \rightarrow A_j \alpha$  do
          begin
            for todas las producciones  $A_\ell \rightarrow \beta$  do
              agregue la producción  $A_k \rightarrow \beta \alpha$ ;
              elimine la producción  $A_k \rightarrow A_j \alpha$ 
          end;
        for cada producción de la forma  $A_k \rightarrow A_k \alpha$  do
          begin
            agregue las producciones  $B_k \rightarrow \alpha$  y  $B_k \rightarrow \alpha B_k$ ;
            elimine la producción  $A_k \rightarrow A_k \alpha$ 
          end;
        for cada producción  $A_k \rightarrow \beta$  en donde  $\beta$  no
          comienza con  $A_k$  do
          agregue la producción  $A_k \rightarrow \beta B_k$ 
      end;
    end;
  end;

```

Fig. 4.9 Paso 1 del algoritmo de la forma normal de Greibach.

Repetiendo el proceso anterior para cada variable original, tendremos sólo producciones de la forma:

- 1  $A_i \rightarrow A_j \gamma$ ,  $j > i$ ,
- 2  $A_i \rightarrow a\gamma$ ,  $a$  está en  $T$ ,
- 3  $B_i \rightarrow \gamma$ ,  $\gamma$  está en  $(V \cup \{B_1, B_2, \dots, B_{i-1}\})^*$ .

Nótese que el símbolo que está más a la izquierda del lado derecho de cualquier producción para  $A_m$  debe ser una terminal, ya que  $A_m$  es la variable con el mayor número. El símbolo que está más a la izquierda del lado derecho de cualquier producción para  $A_{m-1}$  debe ser  $A_m$  o un símbolo terminal. Cuando es  $A_m$ , podemos generar nuevas producciones sustituyendo  $A_m$  por el lado derecho de las producciones para  $A_m$  de acuerdo con el Lema 4.3. Estas producciones deben tener su lado derecho que comience con un símbolo terminal. Entonces procedemos con las producciones para  $A_{m-2}, \dots, A_2, A_1$  hasta que el lado derecho de cada producción para  $A_i$  comience con un símbolo terminal.

Como último paso examinaremos las producciones para las nuevas variables,  $B_1, B_2, \dots, B_m$ . Puesto que comenzamos con una gramática en la forma normal de Chomsky, es fácil probar por inducción, sobre el número de aplicaciones de los Lemas 4.3 y 4.4, que el lado derecho de cada producción  $A_i$ ,  $1 \leq i \leq n$ , comienza con una terminal o  $A_j A_k$  para algunas  $j$  y  $k$ . Por tanto, la cadena  $\alpha$  de la línea (7) de la Fig. 4.9 no puede nunca estar vacía o comenzar con alguna  $B_j$  así que ninguna producción  $B_i$  puede comenzar con otra  $B_j$ . Por consiguiente todas las producciones  $B_i$  tienen su lado derecho que comienza con terminales o  $A_i$ , y una aplicación más del Lema 4.3 para cada producción  $B_i$  completa la construcción.  $\square$

#### Ejemplo 4.10 Convirtamos a la forma normal de Greibach la gramática

$$G = (\{A_1, A_2, A_3\}, \{a, b\}, P, A_1),$$

en donde  $P$  consiste en lo siguiente:

$$\begin{aligned} A_1 &\rightarrow A_2 A_3 \\ A_2 &\rightarrow A_3 A_1 | b \\ A_3 &\rightarrow A_1 A_2 | a \end{aligned}$$

**Paso 1** Como el lado derecho de las producciones para  $A_1$  y  $A_2$  comienzan con terminales o variables con número mayor, empezamos con la producción  $A_3 \rightarrow A_1 A_2$  y sustituimos la cadena por  $A_2 A_3 A_1$ . Nótese que  $A_1 \rightarrow A_2 A_3$  es la única producción con  $A_1$  en la izquierda. El conjunto resultante de producciones es:

$$\begin{aligned} A_1 &\rightarrow A_2 A_3 \\ A_2 &\rightarrow A_3 A_1 | b \\ A_3 &\rightarrow A_2 A_3 A_2 | a \end{aligned}$$

Como el lado derecho de la producción  $A_3 \rightarrow A_2 A_3 A_2$  comienza con una variable de número menor, sustituimos la primera ocurrencia de  $A_2$  con  $A_3 A_1$  y  $b$ . Por tanto  $A_3 \rightarrow A_2 A_3 A_2$  se sustituye con  $A_3 \rightarrow A_3 A_1 A_3 A_2$  y  $A_3 A_2$ . El nuevo conjunto es

$$\begin{aligned} A_1 &\rightarrow A_2 A_3 \\ A_2 &\rightarrow A_3 A_1 | b \\ A_3 &\rightarrow A_3 A_1 A_3 A_2 | b A_3 A_2 | a \end{aligned}$$

Aplicamos ahora el Lema 4.4 a las producciones

$$A_3 \rightarrow A_3 A_1 A_3 A_2 | b A_3 A_2 | a.$$

El símbolo  $B_3$  se introduce y la producción  $A_3 \rightarrow A_3, A_1 A_3 A_2$  se sustituye por  $A_3 \rightarrow b A_3 A_2 B_3, A_3 \rightarrow a B_3, B_3 \rightarrow A_1 A_3 A_2$  y  $B_3 \rightarrow A_1 A_3 A_2 B_3$ . El conjunto resultante es

$$\begin{aligned} A_1 &\rightarrow A_2 A_3 \\ A_2 &\rightarrow A_3 A_1 | b \\ A_3 &\rightarrow b A_3 A_2 B_3 | a B_3 | b A_3 A_2 | a \\ B_3 &\rightarrow A_1 A_3 A_2 | A_1 A_3 A_2 B_3 \end{aligned}$$

**Paso 2** Ahora todas las producciones con  $A_3$  en la izquierda tienen lado derecho que comienzan con terminales. Estas se utilizan para sustituir a  $A_3$  en la producción  $A_2 \rightarrow A_3 A_1$  y entonces las producciones con  $A_2$  en el lado izquierdo se utilizan para sustituir a  $A_2$  en la producción  $A_1 \rightarrow A_2 A_3$ . El resultado es el siguiente.

$$\begin{array}{ll} A_3 \rightarrow b A_3 A_2 B_3 & A_3 \rightarrow b A_3 A_2 \\ A_3 \rightarrow a B_3 & A_3 \rightarrow a \\ A_2 \rightarrow b A_3 A_2 B_3 A_1 & A_2 \rightarrow b A_3 A_2 A_1 \\ A_2 \rightarrow a B_3 A_1 & A_2 \rightarrow a A_1 \\ A_2 \rightarrow b & \\ A_1 \rightarrow b A_3 A_2 B_3 A_1 A_3 & A_1 \rightarrow b A_3 A_2 A_1 A_3 \end{array}$$

$$A_1 \rightarrow aB_3 A_1 A_3$$

$$A_1 \rightarrow bA_3$$

$$B_3 \rightarrow A_1 A_3 A_2$$

$$A_1 \rightarrow aA_1 A_3$$

$$B_3 \rightarrow A_1 A_3 A_2 B_3$$

**Paso 3** Las dos producciones  $B_3$  se convierten a una forma apropiada, lo que trae como resultado otras 10 producciones más. Es decir, las producciones

$$B_3 \rightarrow A_1 A_3 A_2 \quad \text{y} \quad B_3 \rightarrow A_1 A_3 A_2 B_3$$

quedan alteradas mediante la colocación del lado derecho de las cinco producciones con  $A_1$  en la izquierda en lugar de la primera  $A_1$ . En consecuencia  $B_3 \rightarrow A_1 A_3 A_2$  se convierte en

$$B_3 \rightarrow bA_3 A_2 B_3 A_1 A_3 A_2, \quad B_3 \rightarrow aB_3 A_1 A_3 A_3 A_2.$$

$$B_3 \rightarrow bA_3 A_3 A_2, \quad B_3 \rightarrow bA_3 A_2 A_1 A_3 A_3 A_2, \quad B_3 \rightarrow aA_1 A_3 A_3 A_2.$$

Las otras producciones para  $B_3$  se sustituyen de manera similar. El conjunto final de producciones es

$$A_3 \rightarrow bA_3 A_2 B_3$$

$$A_3 \rightarrow aB_3$$

$$A_2 \rightarrow bA_3 A_2 B_3 A_1$$

$$A_2 \rightarrow aB_3 A_1$$

$$A_2 \rightarrow b$$

$$A_1 \rightarrow bA_3 A_2 B_3 A_1 A_3$$

$$A_1 \rightarrow aB_3 A_1 A_3$$

$$A_1 \rightarrow bA_3$$

$$B_3 \rightarrow bA_3 A_2 B_3 A_1 A_3 A_3 A_2 B_3$$

$$B_3 \rightarrow aB_3 A_1 A_3 A_3 A_2 B_3$$

$$B_3 \rightarrow bA_3 A_3 A_2 B_3$$

$$B_3 \rightarrow bA_3 A_2 A_1 A_3 A_3 A_2 B_3$$

$$B_3 \rightarrow aA_1 A_3 A_3 A_2 B_3$$

$$A_3 \rightarrow bA_3 A_2$$

$$A_3 \rightarrow a$$

$$A_2 \rightarrow bA_3 A_2 A_1 A_1$$

$$A_2 \rightarrow aA_1 A_1$$

$$A_1 \rightarrow bA_3 A_2 A_1 A_3$$

$$A_1 \rightarrow aA_1 A_3$$

$$A_1 \rightarrow bA_3$$

$$B_3 \rightarrow bA_3 A_2 B_3 A_1 A_3 A_3 A_2 A_2$$

$$B_3 \rightarrow aB_3 A_1 A_3 A_3 A_2 A_2$$

$$B_3 \rightarrow bA_3 A_3 A_2 A_2$$

$$B_3 \rightarrow bA_3 A_2 A_1 A_3 A_3 A_2 A_2$$

$$B_3 \rightarrow aA_1 A_3 A_3 A_2 A_2$$

#### 4.7 EXISTENCIA DE LENGUAJES LIBRES DE CONTEXTO INHERENTEMENTE AMBIGUOS

Es fácil presentar gramáticas libres de contexto ambiguas. Por ejemplo, considérese la gramática con producciones  $S \rightarrow A$ ,  $S \rightarrow B$ ,  $A \rightarrow a$  y  $B \rightarrow a$ . Lo que no es tarea fácil es presentar un lenguaje libre de contexto para el cual cada

CFG es ambigua. En esta sección mostramos que, efectivamente, existen CFLs inherentemente ambiguos. La prueba es un tanto tediosa, y el estudiante puede, si lo desea, saltarse esta sección sin pérdida de continuidad. La existencia de tales lenguajes se utiliza solamente en el Teorema 8.16.

Mostraremos que el lenguaje

$$L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$$

es inherentemente ambiguo haciendo ver que un número infinito de cadenas de la forma  $a^n b^n c^m d^m$ ,  $n \geq 1$ , deben tener dos derivaciones extremas izquierdas distintas. Procederemos estableciendo primero dos lemas de aspecto técnico.

**Lema 4.5** Sea  $(N_i, M_i)$ ,  $1 \leq i \leq r$ , pares de conjuntos de enteros. (Estos conjuntos pueden ser finitos o infinitos.) Sean

$$S_i = \{(n, m) \mid n \text{ in } N_i, m \text{ in } M_i\}$$

y

$$S = S_1 \cup S_2 \cup \dots \cup S_r.$$

Si cada par de enteros  $(n, m)$  está en  $S$  para toda  $n \neq m$ , en donde  $n \neq m$ , entonces  $(n, n)$  está en  $S$  para todo entero excepto algún conjunto finito de  $n$ .

**Demostración** Suponga que para todas  $n \neq m$ , con  $n \neq m$ , cada  $(n, m)$  está en  $S$ , y que existe un número infinito de  $n$  tales que  $(n, n)$  no está en  $S$ . Sea  $J$  el conjunto de todas las  $n$  tales que  $(n, n)$  no está en  $S$ . Consideraremos una secuencia de conjuntos  $J_r, J_{r-1}, \dots, J_1$  tales que

$$J \supseteq J_r \supseteq J_{r-1} \supseteq \dots \supseteq J_1.$$

Cada  $J_i$  será infinita, y para cada  $n \neq m$  en  $J_i$ ,  $(n, m)$  no está en

$$S_i \cup S_{i+1} \cup \dots \cup S_r.$$

Para  $n$  en  $J$ , sucede que  $n$  no está en  $N_r$ , o  $n$  no está en  $M_r$ ; de otra manera  $(n, n)$  estaría en  $S$ , y por tanto en  $S$ . En consecuencia existe un subconjunto infinito de  $J$ , llamémosle  $J_r$ , tal que para toda  $n$  en  $J_r$ ,  $n$  no está en  $N_r$ , o para toda  $n$  en  $J_r$ ,  $n$  no está en  $M_r$ . Ahora, para  $n \neq m$  en  $J_r$ ,  $(n, m)$  no está en  $S_r$

Supóngase que  $J_r, J_{r-1}, \dots, J_{i+1}$  han sido construidos, en las que  $i \leq r-1$ . Entonces  $J_i$  se construye de la forma siguiente. Para cada  $n$  en  $J_{i+1}$ , se tiene que  $n$  no está en  $N_i$  o no está en  $M_i$ ; de otra manera  $(n, n)$  estaría en  $S_i$  y por tanto en  $S$ , que es una contradicción ya que  $J_{i+1} \subseteq J$ . Por consiguiente, o un subconjunto infinito

de  $J_{i+1}$ , no está en  $N_i$  o un subconjunto infinito de  $J_{i+1}$  no está en  $M_i$ . En cualquier caso, hagamos  $J_i$  tal subconjunto infinito. Ahora para toda  $n$  y  $m$  en  $J_i$ ,  $(n, m)$  no está en  $S_i$  y en consecuencia no está en  $S_i \cup S_{i+1} \cup \dots \cup S_r$ .

Puesto que  $J_1$  contiene un número infinito de elementos, existen  $n$  y  $m$  en  $J_1$ ,  $n \neq m$ . Ahora  $(n, m)$  no está en  $S_1 \cup S_2 \cup \dots \cup S_r = S$ , contradiciendo la suposición de que todo  $(n, m)$  con  $n \neq m$ , está en  $S$ . Por tanto  $(n, n)$  está en  $S$  para todo entero, excepto para algún conjunto finito de  $n$ .  $\square$

**Lema 4.6** Sea  $G$  una CFG no ambigua. Entonces podemos construir efectivamente una CFG no ambigua  $G'$  equivalente a  $G$ , tal que  $G'$  no tenga símbolos o producciones inútiles, y para cada variable  $A$ , posiblemente diferente al símbolo inicial de  $G'$ , tenemos la derivación  $A \xrightarrow{G'} x_1 A x_2$ , en donde  $x_1$  y  $x_2$  no son los dos  $\epsilon$ .

*Demostración* La construcción de los Lemas 4.1 y 4.2, eliminando los símbolos y las producciones inútiles, no puede convertir una gramática no ambigua a una ambigua, ya que el conjunto de árboles de derivación para las palabras no cambia. La construcción del Teorema 4.4, eliminando las producciones unitarias, no puede introducir ambigüedades. Esto sucede porque si introducimos la producción  $A \rightarrow \alpha$ , debe existir una única  $B$  tal que  $A \xrightarrow{*} B$  y  $B \rightarrow \alpha$  sea una producción, de otra manera la gramática original no era no ambigua. De forma similar, la construcción del Teorema 4.3, eliminando las producciones  $\epsilon$ , no introduce la ambigüedad.

Por tanto vamos a suponer que  $G$  no posee símbolos o producciones inútiles, producciones  $\epsilon$  ni producciones unitarias. Supóngase que para ninguna  $x_1$  y  $x_2$  ni ambas  $\epsilon$ , se cumple  $A \xrightarrow{*} x_1 A x_2$ . Entonces sustitúyase cada  $A$  en el lado derecho de cualquier producción por todos los lados derechos de las producciones  $A$ . Como no existen producciones unitarias, producciones  $\epsilon$  ni símbolos inútiles, no puede haber una producción  $A \rightarrow \alpha_1 A \alpha_2$ , de otra forma existe una derivación  $A \xrightarrow{*} x_1 A x_2$ , en la que  $x_1$  y  $x_2$  no son ambas  $\epsilon$ . Según el Lema 4.3, el cambio que se hizo arriba no modifica al lenguaje generado. Cada nueva producción viene de una única secuencia de viejas producciones, de otra forma  $G$  sería ambigua. Por tanto, la gramática resultante es no ambigua. Vemos que ahora  $A$  es inútil y puede ser eliminada. Despues de eliminar de esta manera las variables que violan la condición del lema, la nueva gramática es equivalente a la vieja, aún es no ambigua y satisface al lema.  $\square$

**Teorema 4.7** El CFL,

$$L = \{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\},$$

es inherentemente ambiguo.

*Demostración* Supóngase que existe una gramática no ambigua que genera a  $L$ . Según el Lema 4.6 podemos construir una gramática no ambigua  $G = (V, T, P, S)$

que genere a  $L$  sin símbolos inútiles, y para cada  $A$  en  $V - \{S\}$ ,  $A \xrightarrow{*} x_1 A x_2$  para algunas  $x_1$  y  $x_2$  en  $T^*$ , no ambas  $\epsilon$ .

Notamos que la gramática  $G$  tiene las siguientes propiedades:

1. Si  $A \xrightarrow{*} x_1 A x_2$ , entonces  $x_1$  y  $x_2$  consisten, cada uno, en solamente un tipo de símbolo ( $a$ ,  $b$ ,  $c$  o  $d$ ); de otra forma

$$S \xrightarrow{*} w_1 A w_3 \xrightarrow{*} w_1 x_1 x_1 A x_2 x_2 w_3 \xrightarrow{*} w_1 x_1 x_1 w_2 x_2 x_2 w_3,$$

para algunas  $w_1$ ,  $w_2$  y  $w_3$ . Esta última cadena terminal no está en  $L$ .

2. Si  $A \xrightarrow{*} x_1 A x_2$  consiste en diferentes símbolos. Si no fuera así, en una derivación que involucra a  $A$ , podríamos incrementar el número de un tipo de símbolo en una oración de  $L$  sin aumentar el número de cualquier otro tipo de símbolo, por ello se genera una oración que no está en  $L$ .

3. Si  $A \xrightarrow{*} x_1 A x_2$  entonces  $|x_1| = |x_2|$ . De otra forma, podríamos encontrar palabras que estén en  $L$  y que tengan más de un símbolo que cualquier otro.

4. Si  $A \xrightarrow{*} x_1 A x_2$  y  $A \xrightarrow{*} x_3 A x_4$ , entonces  $x_1$  y  $x_3$  consisten en el mismo tipo de símbolos. De igual forma  $x_2$  y  $x_4$ . De no ser así, se violaría la Propiedad 1, dada más arriba.

5. Si  $A \xrightarrow{*} x_1 A x_2$ , entonces se cumple una de las siguientes

- $x_1$  consiste únicamente en  $as$  y  $x_2$  solamente en  $bs$  o  $ds$ ,
- $x_1$  consiste únicamente en  $bs$  y  $x_2$  solamente en  $cs$  o
- $x_1$  consiste únicamente en  $cs$  y  $x_2$  solamente en  $ds$ .

En cualquiera de los otros casos es fácil derivar una cadena que no esté en  $L$ . Así pues las variables que no sean  $S$  pueden dividirse en cuatro clases,  $C_{ab}$ ,  $C_{ad}$ ,  $C_{bc}$  y  $C_{cd}$ .  $C_{ab}$  es el conjunto de todas las  $A$  que están en  $V$  tales que  $A \xrightarrow{*} x_1 A x_2$ , con  $x_1$  en  $a^*$  y  $x_2$  en  $b^*$ ;  $C_{ad}$ ,  $C_{bc}$  y  $C_{cd}$  se definen de manera análoga.

6. Una derivación que contenga un símbolo que esté en  $C_{ab}$  o en  $C_{cd}$  no puede contener un símbolo que esté en  $C_{ad}$  o en  $C_{bc}$  o viceversa. De otra manera, podríamos incrementar el número de tres tipos de símbolos de una oración en  $L$  sin aumentar el número del cuarto tipo. En ese caso, existiría una oración en  $L$  para la que el número de ocurrencias de un tipo de símbolos sería menor que la de cualquier otro.

Ahora notamos que si una derivación contiene una variable en  $C_{ab}$  o en  $C_{cd}$ , entonces la cadena terminal generada debe estar en  $\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\}$ . Porque supóngase que  $A$ , que está en  $C_{ab}$ , aparece en una derivación de una oración  $x$  que no está en  $\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\}$ . Entonces  $x$  debe ser de la forma  $a^n b^m c^m d^n$ ,  $m \neq n$ . Como  $A$  está en  $C_{ab}$ , podría generarse una oración

$a^{n+p}b^{m+p}c^md^n, m \neq n$ , para alguna  $p > 0$ . Dicha oración no está en  $L$ . Un argumento similar se puede dar para  $A$  en  $C_{cd}$ . Un razonamiento parecido implica que si una derivación contiene a una variable que esté en  $C_{ad}$  o en  $C_{bc}$ , entonces la oración generada debe estar en  $\{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\}$ .

Dividimos  $G$  en dos gramáticas,

$$G_1 = (\{S\} \cup C_{ab} \cup C_{cd}, T, P_1, S)$$

y

$$G_2 = (\{S\} \cup C_{ad} \cup C_{bc}, T, P_2, S),$$

en donde  $P_1$  contiene a todas las producciones de  $P$  con una forma variable  $C_{ab}$  o  $C_{cd}$  en la izquierda o la derecha, y  $P_2$  contiene a todas las producciones de  $P$  con una variable de  $C_{ad}$  o de  $C_{bc}$  en la izquierda o la derecha. Además,  $P_1$  contiene a todas las producciones de  $P$  de la forma  $S \rightarrow a^n b^n c^m d^m, n \neq m$ , y  $P_2$  contiene todas las producciones de  $P$  de la forma  $S \rightarrow a^n b^m c^m d^n, n \neq m$ . Las producciones de  $P$  de la forma  $S \rightarrow a^n b^n c^n d^n$  no están en  $P_1$  ni en  $P_2$ .

Como  $G$  genera a

$$\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1\} \cup \{a^n b^m c^m d^n \mid n \geq 1, m \geq 1\},$$

$G_1$  debe generar a todas las oraciones que estén en

$$\{a^n b^n c^m d^m \mid n \geq 1, m \geq 1, n \neq m\}$$

más posiblemente algunas oraciones en  $\{a^n b^n c^n d^n \mid n \geq 1\}$ , y  $G_2$  debe generar a todas las oraciones que estén en

$$\{a^n b^m c^m d^n \mid n \geq 1, m \geq 1, n \neq m\}$$

más posiblemente algunas oraciones de  $\{a^n b^n c^n d^n \mid n \geq 1\}$ . Mostraremos que éste no puede ser el caso, a menos que tanto  $G_1$  como  $G_2$  generen a todas las oraciones de  $\{a^n b^n c^n d^n \mid n \geq 1\}$ , excepto un conjunto finito de ellas. Por consiguiente, todas las oraciones de  $\{a^n b^n c^n d^n \mid n \geq 1\}$ , excepto un número finito de éstas, son generadas por  $G_1$  y  $G_2$  y, en consecuencia, por dos derivaciones distintas de  $G$ . Esto contradice la hipótesis de que  $G$  sea no ambigua.

Para ver que  $G_1$  y  $G_2$  generan todas las oraciones de  $\{a^n b^n c^n d^n \mid n \geq 0\}$ , excepto un número finito de ellas, numérense las producciones de  $P_1$  de la forma  $S \rightarrow \alpha$  de 1 a  $r$ . Para  $1 \leq i \leq r$ , si  $S \rightarrow \alpha$  es la  $i$ -ésima producción, sea  $N_i$  el conjunto de todas las  $n$  tales que

$$S \xrightarrow[G_1]{\alpha} a^n b^n c^m d^m$$

para alguna  $m$ , y sea  $M_i$  el conjunto de todas las  $m$  tales que

$$S \xrightarrow[G_1]{\alpha} a^n b^n c^m d^m$$

para alguna  $n$ . Dejamos al lector probar que para cualquier  $n$  de  $N_i$  y cualquier  $m$  de  $M_i$ ,

$$S \xrightarrow[G_1]{\alpha} a^n b^n c^m d^m.$$

[*Sugerencia*: Recuerde que las variables de  $\alpha$  están en  $C_{ab}$  o en  $C_{cd}$ .] Del Lema 4.5 se concluye inmediatamente que  $G_1$  debe generar a todas las oraciones de  $\{a^n b^n c^n d^n \mid n \geq 1\}$ , excepto un número finito de ellas.

Un argumento similar se aplica a  $G_2$ . El lector puede mostrar fácilmente que  $G_2$  no puede tener en su lado derecho dos o más variables. Numeramos ciertas producciones y pares de producciones con un orden simple. Las producciones de la forma  $S \rightarrow \alpha_1 B \alpha_2$ , en donde  $B$  está en  $C_{bc}$ , tendrán un número, y si este número es  $i$ , hagamos  $N_i$  el conjunto de todas las  $n$  tal que, para alguna  $m$

$$S \Rightarrow \alpha_1 B \alpha_2 \xrightarrow{*} a^n b^m c^m d^n.$$

También hagamos que  $M_i$  sea el conjunto de  $m$  tal que, para alguna  $n$ ,

$$S \Rightarrow \alpha_1 B \alpha_2 \xrightarrow{*} a^n b^m c^m d^n.$$

El par de producciones  $S \rightarrow \alpha$  y  $A \rightarrow \alpha_1 B \alpha_2$  recibirán un número si  $\alpha$  contiene una variable que esté en  $C_{ad}$ ,  $A$  esté en  $C_{ad}$  y  $B$  esté en  $C_{bc}$ . Si a este par se le asigna el número  $i$ , entonces definase  $N_i$  como el conjunto de  $n$  tal que, para alguna  $m$ .

$$S \Rightarrow \alpha \xrightarrow{*} x_1 A x_2 \Rightarrow x_1 \alpha_1 B \alpha_2 x_2 \xrightarrow{*} a^n b^m c^m d^n.$$

Definase también  $M_i$  como el conjunto de las  $m$  tal que, para alguna  $n$ .

$$S \Rightarrow \alpha \xrightarrow{*} x_1 A x_2 \Rightarrow x_1 \alpha_1 B \alpha_2 x_2 \xrightarrow{*} a^n b^m c^m d^n.$$

Otra vez, para cada  $n$  en  $N_i$  y  $m$  en  $M_i$ ,

$$S \xrightarrow[G_2]{\alpha} a^n b^m c^m d^n,$$

y por tanto se concluye, según el Lema 4.5, que  $G_2$  genera todas las oraciones de  $\{a^n b^n c^n d^n \mid n \geq 1\}$ , excepto un número finito de éstas. Concluimos que para alguna  $n$ ,  $a^n b^n c^n d^n$  está tanto en  $L(G_1)$  como en  $L(G_2)$ . Esta oración tiene dos derivaciones izquierdas en  $G$ .  $\square$

## EJERCICIOS

**4.1** Dé gramáticas libres de contexto que generen a los conjuntos siguientes.

- S a)** El conjunto de palíndromos (cadenas que se leen igual de derecha a izquierda y viceversa) sobre el alfabeto  $\{a, b\}$ .
- b)** El conjunto de todas las cadenas de paréntesis balanceados, es decir, cada paréntesis izquierdo tiene su correspondiente paréntesis derecho y cada par de paréntesis se encuentran anidados de manera apropiada.
- \* c)** El conjunto de todas las cadenas sobre el alfabeto  $\{a, b\}$  con exactamente el doble de aes que de bs.
- d)** El conjunto de todas las cadenas sobre el alfabeto  $\{a, b, ., +, *, (,), \in, \emptyset\}$  que son expresiones regulares bien formadas sobre el alfabeto  $\{a, b\}$ . Nótese que debemos hacer diferencia entre  $\in$  como la cadena vacía y como un símbolo en una expresión regular. Utilizamos  $\in$  en el último caso.
- \* e)** El conjunto de todas las cadenas sobre el alfabeto  $\{a, b\}$  que no son de la forma  $vw$  para alguna cadena  $w$ .
- f)**  $\{a^i b^j c^k \mid i \neq j \text{ o } j \neq k\}$ .

**\*4.2** Sea  $G$  la gramática

$$S \rightarrow aS \mid aSbS \mid \epsilon.$$

Demuestre que

$$L(G) = \{x \mid \text{cada prefijo de } x \text{ tiene al menos tantas aes como bs}\}.$$

**\*4.3** Para  $i \geq 1$ , sea  $b_i$  la cadena de  $1(0 + 1)^*$  que es la representación binaria de  $i$ . Constrúyase una CFG que genere a

$$\{0, 1, \#\}^+ - \{b_1 \# b_2 \# \dots \# b_n \mid n \geq 1\}.$$

**\*4.4** Constrúyase una CFG que genere al conjunto

$$\{w \# w^R \# \mid w \text{ en } (0 + 1)^+\}^*.$$

**\*4.5** La gramática

$$E \rightarrow E + E * E \mid (E) \mid \text{id}$$

genera al conjunto de las expresiones aritméticas con  $+$ ,  $*$ , paréntesis e **id**. La gramática es ambigua ya que **id** + **id** \* **id** puede ser generada por dos derivaciones izquierdas distintas.

- a) Construya una gramática equivalente no ambigua.
- b) Construya una gramática no ambigua para todas las expresiones aritméticas que no tengan paréntesis redundantes. Un conjunto de paréntesis es redundante si su eliminación no cambia la expresión, esto es, los paréntesis son redundantes en **id** + (**id** \* **id**) pero no lo son en (**id** + **id**) \* **id**.
- \*4.6** Supóngase que  $G$  es una CFG con  $m$  variables y en la cual ningún lado derecho de una producción es mayor que  $\ell$ . Muestre que si  $A \xrightarrow[G]{*} \epsilon$ , entonces existe una derivación

de no más de  $\frac{\ell^m - 1}{\ell - 1}$  pasos mediante la cual  $A$  deriva a  $\epsilon$ . ¿Qué tan cerca de este límite se puede llegar en realidad?

**\*4.7** Muestre que para cada CFG  $G$  existe una constante  $c$  tal que si  $w$  está en  $L(G)$  y  $w \neq \epsilon$ , entonces  $w$  tiene una derivación de no más de  $c \mid w \mid$  pasos.

**4.8** Sea la gramática

$$\begin{aligned} S &\rightarrow aB \mid bA \\ A &\rightarrow a \mid aS \mid bAA \\ B &\rightarrow b \mid bS \mid aBB \end{aligned}$$

Para la cadena *aaabbabbba* encuentre

- a) una derivación extrema izquierda,
- b) una derivación extrema derecha
- c) un árbol de análisis gramatical.

**\*4.9** ¿Es la gramática del Ejercicio 4.8 no ambigua?

**4.10** Encuentre una CFG que no contenga símbolos inútiles equivalentes a

$$\begin{aligned} S &\rightarrow AB \mid CA & B &\rightarrow BC \mid AB \\ A &\rightarrow a & C &\rightarrow aB \mid b \end{aligned}$$

**4.11** Supóngase que  $G$  es una CFG y  $w$ , de longitud  $\ell$ , está en  $L(G)$ .

¿Qué longitud tiene una derivación de  $w$  en  $G$  si

- a)  $G$  está en la CNF?
- b)  $G$  está en la GNF?

**4.12** Sea  $G$  la CFG que genera fórmulas bien formadas del cálculo proposicional con predicados  $p$  y  $q$ :

$$S \rightarrow \sim S \mid [S \supseteq S] \mid p \mid q.$$

Las terminales son  $p, q, \sim, [,], \supseteq$ . Encuentre una gramática en la forma normal de Chomsky que genere a  $L(G)$ .

**4.13** Muestre que la conversión a la forma normal de Chomsky puede elevar al cuadrado el número de producciones de una gramática. [Sugerencia: Considérese la eliminación de producciones unitarias.]

**4.14** Encuentre una gramática en la forma normal de Greibach equivalente a la siguiente CFG:

$$\begin{aligned} S &\rightarrow AA \mid 0 \\ A &\rightarrow SS \mid 1 \end{aligned}$$

**4.15** Muestre que cada CFL que no contenga a  $\epsilon$ , puede ser generado por una CFG cuyas producciones son todas de la forma  $A \xrightarrow{} a$  o  $A \xrightarrow{} BC$ , en donde  $B \neq C$  y si  $A \xrightarrow{} \alpha_1 B \alpha_2$  y  $A \xrightarrow{} \gamma_1 B \gamma_2$  son producciones, entonces  $\alpha_1 = \gamma_1 = \epsilon$  o  $\alpha_2 = \gamma_2 = \epsilon$ .

**\*4.16** Muestre que cada CFL que no contenga a  $\epsilon$ , puede ser generado por una CFG cuyas producciones son todas de la forma  $A \xrightarrow{} a$ ,  $A \xrightarrow{} aB$  y  $A \xrightarrow{} aBC$ .

**4.17** Muestre que cada CFL que no contenga  $a \in$  es generado por una CFG cuyas producciones son todas de la forma  $A \rightarrow a$  y  $A \rightarrow aab$ .

**4.18** ¿Puede cada CFL, sin producciones  $\epsilon$ , ser generado por una CFG cuyas producciones son todas de la forma  $A \rightarrow BCD$  y  $A \rightarrow a$ ?

**\*4.19** Muestre que si todas las producciones de una CFG son de la forma  $A \rightarrow wB$  o  $A \rightarrow w$ , entonces  $L(G)$  es un conjunto regular.

**\*\*4.20** Se dice que una CFG es *lineal* si ningún lado derecho de una producción tiene más de una instancia para una variable. ¿Cuáles de los lenguajes del Ejercicio 4.1 tienen gramáticas lineales?

**\*\*S 4.21** Una *gramática de operador* es una CFG, que no posee producciones  $\epsilon$ , de forma tal que no existen símbolos consecutivos en el lado derecho de las producciones que sean variables. Muestre que cada CFL sin  $\epsilon$  tiene una gramática de operador.

**\*\*4.22** El algoritmo que se dio en la Fig. 4.7, para determinar qué variables derivan cadenas terminales no es el más eficiente posible. Dé un programa de computación que realice la tarea en  $O(n)$  pasos si  $n$  es la suma de la longitud de todas las producciones.

**\*\*4.23** ¿Es el conjunto  $\{a^ib^jc^k \mid i \neq j, j \neq k \text{ y } k \neq i\}$  un CFL?

[*Sugerencia:* Desarrolle una forma normal parecida a la del Teorema 4.7. En la Sección 6.1 se desarrolla un lema de sondeo que hace mucho más fáciles esta clase de ejercicios. El lector puede comparar su solución con la del Ejercicio 6.3.]

### Soluciones a los ejercicios seleccionados

**4.1** a) La definición de “palíndromo”, una cadena que se lee igual de izquierda a derecha y viceversa no es de utilidad alguna para encontrar una CFG. Lo que debemos hacer en este caso y en otros muchos es recablar la definición y convertirla a una forma recursiva. Podemos definir a los palíndromos sobre  $\{0, 1\}$  de manera recursiva como sigue:

1.  $\epsilon, 0$  y  $1$  son palíndromos;
2. Si  $w$  es un palíndromo, también lo son  $0w0$  y  $1w1$ ;
3. Ninguna otra cadena es un palíndromo.

Demostramos en el Ejercicio 1.3 que ésta es una definición válida para los palíndromos. Una CFG para palíndromos se desprende de manera inmediata de (1) y (2). Esta es:

$$S \rightarrow 0 \mid 1 \mid \epsilon \quad (\text{de 1});$$

$$S \rightarrow 0S0 \mid 1S1 \quad (\text{de 2}).$$

**4.16** Sea  $G = (V, T, P, S)$  una gramática GNF que genera a  $L$ . Supóngase que  $k$  es la longitud del lado derecho más largo de una producción de  $G$ . Sea  $V' = \{\alpha \mid \alpha \text{ está en } V^* \text{ y } |\alpha| < k\}$ . Para cada producción  $A \rightarrow \alpha$  de  $P$  y cada variable  $[A\beta]$  de  $V'$  coloque  $[A\beta] \rightarrow a[\alpha][\beta]$  en  $P'$ . En el caso en que  $\alpha$  o  $\beta$  sea  $\epsilon$ ,  $[\epsilon]$  se elimina del lado derecho de la producción.

**4.21** Sea  $G = (V, T, P, S)$  una gramática GNF que genera a  $L$ . Mediante el Ejercicio 4.16 podemos suponer que todas las producciones son de la forma  $A \rightarrow a$ ,  $A \rightarrow aB$  y  $A \rightarrow aBC$ . Primero sustituya cada producción  $A \rightarrow aBC$  por  $A \rightarrow a[BC]$ , en donde  $[BC]$  es una nueva

variable. Después de haber sustituido todas las producciones de la forma  $A \rightarrow aBC$ , entonces para cada variable  $[BC]$ , producción  $B$ ,  $B \rightarrow \alpha$ , y cada producción  $C$ ,  $C \rightarrow \beta$ , introducida por vez primera, añade la producción  $[BC] \rightarrow \alpha\beta$ . Nótese que  $\alpha$  y  $\beta$  son terminales simples o de la forma  $bE$ , en las que  $E$  puede ser una variable nueva o vieja. La gramática resultante es una gramática de operador equivalente a la original.

### NOTAS BIBLIOGRAFICAS

El origen del formalismo de las gramáticas libres de contexto se encuentra en Chomsky [1956]; escritos posteriores de importancia hechos por Chomsky sobre la materia aparecen en Chomsky [1959, 1963]. La notación Backus-Naur relacionada se utilizó para la descripción del ALGOL en Backus [1959] y Naur [1960]. La relación entre CFGs y BNF se estableció en Ginsburg y Rice [1962].

La forma normal de Chomsky se basa en Chomsky [1959]. En realidad, Chomsky demostró el resultado más firme que se presentó en el Ejercicio 4.15. La forma normal de Greibach fue demostrada por Greibach [1965]. El método para su demostración que se utilizó en este libro se debe a M. C. Paull. El lector deberá considerar también el algoritmo de Rosenkrantz [1967], que tiene la propiedad de que nunca aumenta a más del cuadrado el número de variables, mientras que al algoritmo del Teorema 4.6 puede exponenciar el número. En esta misma referencia se puede hallar la solución a los Ejercicios 4.16, 4.17 y 4.21.

La ambigüedad de las CFGs fue estudiada por primera vez de manera formal por Floyd [1962a], Cantor [1962], Chomsky y Schutzenberger [1963] y Greibach [1963]. La ambigüedad inherente fue estudiada por Gross [1964] y Ginsburg y Ullian [1966a, b].

Se han hecho importantes aplicaciones de la teoría de las gramáticas libres de contexto al diseño de compiladores. Véase Aho y Ullman [1972, 1973, 1977], Lewis, Rosenkrantz y Stearns [1976] y las notas bibliográficas del Capítulo 10 para la descripción de algo del trabajo que se ha hecho en esta área. Material adicional sobre lenguajes libres de contexto puede encontrarse en Ginsburg [1966] y Salomaa [1973].

UNIVERSIDAD DE LA REPÚBLICA  
FACULTAD DE CIENCIAS  
DEPARTAMENTO DE  
INVESTIGACIONES Y BIBLIOTECA  
MONTEVIDEU - URUGUAY

# 5

## AUTOMATAS DE APILAMIENTO

### 5.1 DESCRIPCION INFORMAL

Así como las expresiones tienen un autómata equivalente (el autómata finito) las gramáticas libres de contexto tienen su contraparte en forma de máquina, es decir, el autómata de apilamiento. En este caso la equivalencia es un poco menos satisfactoria, ya que el autómata de apilamiento es un dispositivo no determinístico y su versión determinística sólo acepta un subconjunto de todos los CFLs. Afortunadamente, este subconjunto incluye a la sintaxis de la mayoría de los lenguajes de programación. (Véase el Capítulo 10 para un estudio detallado de los lenguajes de los autómatas de apilamiento determinísticos.)

El autómata de apilamiento es esencialmente un autómata finito que posee control sobre una cinta de entrada y una lista en forma de pila o del tipo “el primero entra, el último sale”. Es decir, sólo pueden accesarse o retirarse los símbolos que se encuentran en la cima o al frente de la lista. Cuando un símbolo se coloca en la cima de la lista, el símbolo anterior se convierte en el segundo, el segundo en el tercero y, así, sucesivamente. De manera similar, cuando un símbolo se retira del frente de la lista, el símbolo inmediato anterior se convierte en el primer símbolo, el tercero en el segundo y, así, sucesivamente.

Un ejemplo común de apilamiento es la pila de platos tal y como podemos verla conformada en el dispositivo de la fuente de las cafeterías. Se tiene un resorte debajo de los platos con la fuerza justa para empujar la pila y que sólo aparezca un plato al nivel del mostrador. Cuando el plato de hasta arriba se saca de la pila, la carga en el resorte

se aligera, provocando que el plato inmediatamente debajo del primero aparezca a la altura del mostrador. Si un plato se coloca encima de la pila, ésta es presionada hacia abajo de modo que sólo aparece el plato de encima a la altura del mostrador. Para nuestros propósitos suponemos que el resorte es de longitud arbitraria, de manera que podemos apilar tantos platos como queramos.

Una pila de platos con estas características, acoplada a un control finito, puede utilizarse para reconocer un conjunto no regular. El conjunto  $L = \{wcw^R \mid w \text{ está en } (0+1)^*\}$  es un lenguaje libre de contexto, generado por la gramática  $S \rightarrow 0S0 \mid 1S1 \mid c$ . No es difícil mostrar que  $L$  no puede ser aceptado por cualquier autómata finito. Para aceptar a  $L$  debemos hacer uso de un control finito con dos estados,  $q_1$  y  $q_2$ , y una pila en la cual coloquemos platos azules, verdes y rojos. El dispositivo operará bajo las siguientes reglas:

1. La máquina comienza con un plato rojo sobre la pila y con el control finito en el estado  $q_1$ .
2. Si la entrada al dispositivo es 0 y éste está en el estado  $q_1$ , se coloca sobre la pila un plato azul. Si la entrada del dispositivo es 1 y éste se encuentra en el estado  $q_1$ , se coloca un plato verde sobre la pila. En ambos casos el control finito permanece en el estado  $q_1$ .
3. Si la entrada es  $c$  y el dispositivo está en el estado  $q_1$ , éste cambia al estado  $q_2$  y no se añaden ni se retiran platos.
4. Si la entrada es 0 y el dispositivo está en el estado  $q_2$  con un plato azul, que representa 0, en la cima, el plato se retira. Si la entrada es 1 y el dispositivo se encuentra en el estado  $q_2$  con un plato verde, que representa 1, en el tope de la pila, el plato se retira. En ambos casos el control finito permanece en el estado  $q_2$ .
5. Si el dispositivo está en el estado  $q_2$  y un plato rojo se encuentra en la cima de la pila, el plato se retira sin esperar la siguiente entrada.
6. Para todos los casos distintos a los mencionados arriba, el dispositivo no puede hacer ningún movimiento.

Las reglas anteriores se resumen en la Fig. 5.1.

Diremos que el dispositivo descrito arriba acepta a una cadena de entrada si, al procesar el último símbolo de la cadena, la pila se vacía completamente. Nótese que, una vez que se vacía la pila, no es posible hacer más movimientos.

Esencialmente, el dispositivo opera de la manera siguiente. En el estado  $q_1$ , la máquina hace una imagen de su entrada colocando un plato azul en la cima de la pila cada vez que aparece un 0 en la entrada, y colocando un plato verde cada vez que aparece un 1 en la entrada. Cuando la entrada es  $c$ , el dispositivo entra al estado  $q_2$ . En seguida, la entrada restante se compara con la pila mediante el retiro de un plato azul de encima de la pila cada vez que el símbolo de entrada es 0, y un plato verde cuando el símbolo de entrada es 1. Cuando el plato de encima es del color indebido, el dispositivo se detiene y no es posible seguir procesando la entrada. Si todos los platos ajustan con las entradas, finalmente quedará expuesto el plato rojo del fondo de la pila. El plato rojo es retirado de manera inmediata y se dice que el dispositivo ha aceptado a la cadena de entrada. Todos los platos pueden ser retirados sólo cuando la cadena que entra al dispositivo después de  $c$  es el inverso de lo que entró antes de  $c$ .

## 5.2 DEFINICIONES

Formalizaremos ahora el concepto de autómata de apilamiento (PDA). El PDA tendrá una *cinta de entrada*, un *control finito* y una *pila*. La pila es una cadena de símbolos de algún alfabeto. El símbolo que se encuentra más a la izquierda se considera como que está en la "cima" de la pila. El dispositivo será no determinístico y tendrá algún número finito de alternativas de movimiento en cada situación. Los movimientos serán de dos tipos. En el primer tipo de movimiento se utiliza un símbolo de entrada. Dependiendo del símbolo de entrada, del símbolo de la cima y el estado del control finito, es posible un cierto número de alternativas. Cada alternativa consiste en un estado posterior para el control finito y una cadena (posiblemente vacía) de símbolos para sustituir al símbolo que se encuentra en la cima de la pila. Después de seleccionar una alternativa, la cabeza de entrada avanza un símbolo.

El segundo tipo de movimiento (conocido como movimiento  $\epsilon$ ) es parecido al primero, excepto que el símbolo de entrada no se utiliza y la cabeza de la entrada no avanza después del movimiento. Este tipo de movimiento permite al PDA manipular la pila sin leer símbolos de entrada.

Primer plato	Estado	Entrada		
		0	1	c
Azul	$q_1$	Agregue un plato azul; permanezca en el estado $q_1$	Agregue un plato verde; permanezca en el estado $q_1$	Vaya al estado $q_2$
	$q_2$	Retire el plato del tope; permanezca en el estado $q_2$	—	—
Verde	$q_1$	Agregue un plato azul; permanezca en el estado $q_1$	Agregue un plato verde; permanezca en el estado $q_1$	Vaya al estado $q_2$
	$q_2$	—	Retire el plato del tope; permanezca en el estado $q_2$	—
Rojo	$q_1$	Agregue un plato azul; permanezca en el estado $q_1$	Agregue un plato verde; permanezca en el estado $q_1$	Vaya al estado $q_2$
	$q_2$	Sin esperar la nueva entrada, retire el plato del tope.		

Fig. 5.1 Control finito para el autómata de presión que acepta al lenguaje  $\{wcw^R \mid w \text{ está en } (0+1)^*\}$ .

Finalmente, debemos definir el lenguaje aceptado por un autómata de apilamiento. Existen dos modos naturales de hacerlo. El primero, que ya hemos visto, consiste en definir el lenguaje aceptado como el conjunto de todas las entradas para las cuales una sucesión de movimientos ocasiona que el autómata de apilamiento vacíe su pila. Este lenguaje se conoce como el lenguaje aceptado mediante el agotamiento de la pila.

La segunda manera de definir el lenguaje aceptado es parecida a la forma en que un autómata finito acepta sus entradas. Es decir, designamos algunos estados como estados finales y definimos el lenguaje aceptado como el conjunto de todas las entradas para las cuales alguna selección de movimientos ocasiona que el autómata de apilamiento accese un estado final.

Como veremos, las dos definiciones de aceptación son equivalentes en el sentido de que si un conjunto es aceptado, mediante el agotamiento de la pila, por algún PDA, puede ser aceptado mediante el acceso de un estado final por otro PDA y viceversa.

La aceptación mediante un estado final es el concepto más común, pero resulta más sencillo demostrar el teorema básico de los autómatas de apilamiento utilizando la aceptación por agotamiento de la pila. Este teorema establece que un lenguaje es aceptado por un autómata de apilamiento si y sólo si es un lenguaje libre de contexto. Un *autómata de apilamiento*  $M$  es un sistema  $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ , en donde

1.  $Q$  es un conjunto finito de *estados*;
2.  $\Sigma$  es un alfabeto llamado *alfabeto de entrada*;
3.  $\Gamma$  es un alfabeto, conocido como *alfabeto de la pila*;
4.  $q_0$ , que está en  $Q$ , es el *estado inicial*;
5.  $Z_0$ , que está en  $\Gamma$ , es un símbolo particular de la pila, llamado *símbolo inicial*;
6.  $F \subseteq Q$  es el conjunto de *estados finales*;
7.  $\delta$  es una transformación de  $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$  en los subconjuntos finitos  $Q \times \Gamma^*$ .

A menos que se establezca otra cosa, utilizaremos letras minúsculas cerca del frente del alfabeto para denotar símbolos de entrada y letras minúsculas cerca del final del alfabeto para denotar cadenas de símbolos de entrada. Las letras mayúsculas representan símbolos de pila y las letras griegas indican cadenas de símbolos de pila.

## Movimientos

La interpretación de

$$\delta(q, \epsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\}$$

en donde  $q$  y  $p_i$ ,  $1 \leq i \leq m$ , son estados,  $a$  está en  $\Sigma$ ,  $Z$  es un símbolo de pila y  $\gamma_i$  está en  $\Gamma^*$ ,  $1 \leq i \leq m$ , consiste en que el PDA en el estado  $q$ , con símbolo de entrada  $a$  y  $Z$  el símbolo de la cima de la pila, puede, para cualquier  $i$ , accesar el estado  $p_i$ , sustituir el símbolo  $Z$  por la cadena  $\gamma_i$  y avanzar la cabeza de entrada un símbolo. Adoptamos la convención de que el símbolo que está más a la izquierda de  $\gamma_i$  será colocado encima de la pila y el símbolo que está más a la derecha será colocado en el fondo. Nótese que no es permisible escoger el estado  $p_j$  y la cadena  $\gamma_j$ , para alguna  $j \neq i$  en un movimiento. La interpretación de

$$\delta(q, \epsilon, Z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\}$$

es que el PDA en el estado  $q$ , independientemente del símbolo de entrada que está leyendo, y con  $Z$  el símbolo de la cima de la pila, puede accesar el estado  $p_i$  y sustituir  $Z$  por  $\gamma_i$  para cualquiera  $i$ ,  $1 \leq i \leq m$ . En este caso, la cabeza de entrada no avanza.

**Ejemplo 5.1** La Fig. 5.2 presenta un autómata de apilamiento formal que acepta a  $\{wcw^R \mid w \text{ está en } (0 + 1)^*\}$  mediante agotamiento de pila. Nótese que para un movimiento en el que PDA escribe un símbolo en la cima de la pila,  $\delta$  tiene el valor  $(q, \gamma)$  en donde  $|\gamma| = 2$ . Por ejemplo,  $\delta(q_1, 0, R) = \{(q_1, BR)\}$ . Si  $\gamma$  fuera de longitud uno, el PDA simplemente sustituiría el símbolo de la cima de la pila por un nuevo símbolo y no aumentaría la longitud de la pila.

Adviértase que la regla  $\delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\}$  significa que el PDA, en el estado  $q_2$  con  $R$  el símbolo de la cima, puede borrar a  $R$  independientemente del símbolo de entrada. En este caso, la cabeza de entrada no avanza y, de hecho, no hay necesidad de que haya ninguna entrada restante.

$$\begin{aligned} M &= (\{q_1, q_2\}, \{0, 1, c\}, \{R, B, G\}, \delta, q_1, R, \emptyset) \\ \delta(q_1, 0, R) &= \{(q_1, BR)\} & \delta(q_1, 1, R) &= \{(q_1, GR)\} \\ \delta(q_1, 0, B) &= \{(q_1, BB)\} & \delta(q_1, 1, B) &= \{(q_1, GB)\} \\ \delta(q_1, 0, G) &= \{(q_1, BG)\} & \delta(q_1, 1, G) &= \{(q_1, GG)\} \\ \delta(q_1, c, R) &= \{(q_2, R)\} \\ \delta(q_1, c, B) &= \{(q_2, B)\} \\ \delta(q_1, c, G) &= \{(q_2, G)\} \\ \delta(q_2, 0, B) &= \{(q_2, \epsilon)\} & \delta(q_2, 1, G) &= \{(q_2, \epsilon)\} \\ \delta(q_2, \epsilon, R) &= \{(q_2, \epsilon)\} \end{aligned}$$

Fig. 5.2 Autómata de apilamiento formal que acepta a  $\{wcw^R \mid w \text{ está en } (0 + 1)^*\}$  mediante agotamiento de pila.

## Descripciones instantáneas

Para describir de manera formal la configuración de un PDA en un instante dado, definimos lo que se conoce como *descripción instantánea (ID)*. La ID debe, por supuesto, registrar el estado y el contenido de la pila. Sin embargo, encontramos útil incluir también el concepto de “entrada no utilizada”. Por consiguiente definimos una ID como el triplete  $(q, w, \gamma)$ , en donde  $q$  es un estado,  $w$  una cadena de símbolos de entrada y  $\gamma$  una cadena de símbolos de pila. Si  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  es un PDA, decimos que  $(q, aw, Z\alpha) \vdash_M (p, w, \beta\alpha)$  si  $\delta(q, a, Z)$  contiene a  $(p, \beta)$ . Nótese que  $a$  puede ser  $\epsilon$  o un símbolo de entrada. Por ejemplo, en el PDA de la Fig. 5.2, el hecho de que  $(q_1, BG)$  está en  $\delta(q_1, 0, G)$  nos dice que  $(q_1, 011, GRR) \vdash (q_1, 11, BGGR)$ .

Utilizamos  $\underline{|}_M^*$  para la cerradura reflexiva y transitiva de  $\underline{|}_M$ . Esto es,  $I \underline{|}_M^* I$  para cada ID  $I$  e  $I \underline{|}_M J$  y  $J \underline{|}_M^* K$  implica que  $I \underline{|}_M^* K$ . Escribimos  $I \stackrel{i}{\underline{|}} K$  si la ID  $I$  puede convertirse en  $K$  después de exactamente  $i$  movimientos. El subíndice se elimina de  $\underline{|}_M^*$ ,  $\underline{|}_M$  y  $\underline{|}_M^*$  cuando el PDA particular  $M$  se sobreentiende.

### Lenguajes aceptados

Para el PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  definimos a  $L(M)$ , el *lenguaje aceptado mediante estado final*, como

$$\{w \mid (q_0, w, Z_0) \underline{|}^* (p, \epsilon, \gamma) \text{ para alguna } p \text{ en } F \text{ y } \gamma \text{ en } \Gamma^*\}.$$

Definimos  $N(M)$ , el *lenguaje aceptado mediante agotamiento de pila* (o *pila nula*) como

$$\{w \mid (q_0, w, Z_0) \underline{|}^* (p, \epsilon, \epsilon) \text{ para alguna } p \text{ en } Q\}.$$

Cuando la aceptación es mediante agotamiento de pila, el conjunto de estados finales no es relevante y, en este caso, por lo general hacemos que el conjunto de estados finales sea el conjunto vacío.

**Ejemplo 5.2** La Fig. 5.3 da un PDA que acepta a  $\{ww^R \mid w \text{ está en } (0+1)^*\}$ . Las reglas (1) hasta (6) permiten que  $M$  almacene la entrada en la pila. En las reglas (3) y (6),  $M$  tiene la opción de dos movimientos.  $M$  puede decir que la mitad de la cadena de entrada ha sido alcanzada y elegir el segundo movimiento:  $M$  se pasa al estado  $q_2$  e intenta concordar los restantes símbolos de entrada con el contenido de la pila. Si  $M$  elige de manera correcta, y si la entrada es de la forma  $ww^R$ , entonces las entradas concordarán,  $M$  agotará su pila y en consecuencia aceptará a la cadena de entrada.

$$M = (\{q_1, q_2\}, \{0, 1\}, \{R, B, G\}, \delta, q_1, R, \emptyset)$$

- |   |   |
|---|---|
| 1) $\delta(q_1, 0, R) = \{(q_1, BR)\}$                  | 6) $\delta(q_1, 1, G) = \{(q_1, GG), (q_2, \epsilon)\}$ |
| 2) $\delta(q_1, 1, R) = \{(q_1, GR)\}$                  | 7) $\delta(q_2, 0, B) = \{(q_2, \epsilon)\}$            |
| 3) $\delta(q_1, 0, B) = \{(q_1, BB), (q_2, \epsilon)\}$ | 8) $\delta(q_2, 1, G) = \{(q_2, \epsilon)\}$            |
| 4) $\delta(q_1, 0, G) = \{(q_1, BG)\}$                  | 9) $\delta(q_1, \epsilon, R) = \{(q_2, \epsilon)\}$     |
| 5) $\delta(q_1, 1, B) = \{(q_1, GB)\}$                  | 10) $\delta(q_2, \epsilon, R) = \{(q_2, \epsilon)\}$    |

Fig. 5.3 PDA no determinístico que acepta a  $\{wcw^R \mid w \text{ está en } (0+1)^*\}$  mediante agotamiento de pila.

Como sucede con el autómata finito no determinístico, un PDA no determinístico  $M$  acepta una entrada si cualquier sucesión de alternativas ocasiona que  $M$  agote su pila. Así pues,  $M$  siempre “toma la decisión correcta”, debido a que una decisión equivocada, por sí misma, no ocasiona que una entrada sea rechazada. Una entrada es rechazada sólo si no existen “alternativas correctas”. La Fig. 5.4 muestra las IDs accesibles de  $M$  cuando  $M$  procesa a la cadena 001100.

### PDAs determinísticos

El PDA del Ejemplo 5.1 es determinístico en el sentido de que, cuando mucho, un movimiento es posible a partir de cualquier ID. Formalmente, decimos que un PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ , es

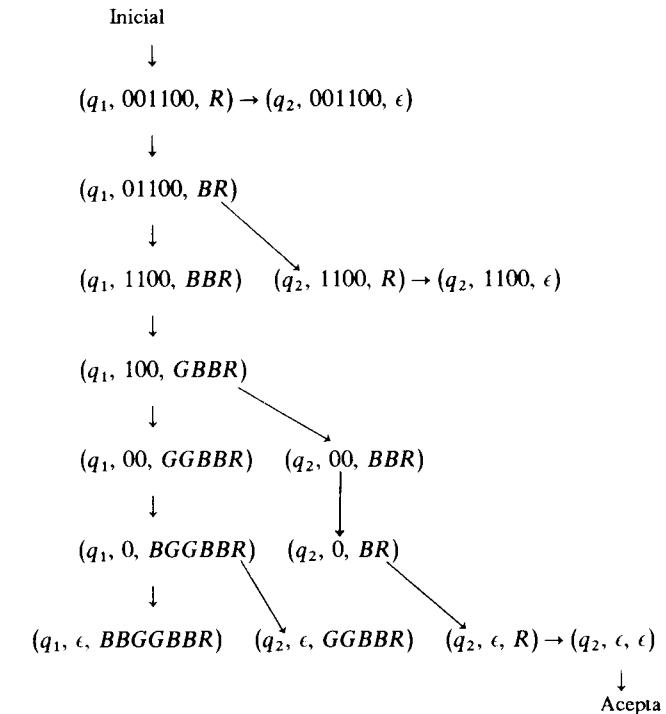


Fig. 5.4 IDs accesibles al PDA de la Fig. 5.3 con entrada 001100.

es determinístico si:

1. para cada  $q$  en  $Q$  y  $Z$  en  $\Gamma$ , cuando  $\delta(q, \epsilon, Z)$  no es el vacío, entonces  $\delta(q, a, Z)$  está vacía para toda  $a$  en  $\Sigma$ ;
2. para ninguna  $q$  en  $Q$ ,  $Z$  en  $\Gamma$  y  $a$  en  $\Sigma \cup \{\epsilon\}$   $\delta(q, a, Z)$  contiene más de un elemento.

La condición 1 evita la posibilidad de una elección entre un movimiento independiente del símbolo de entrada (movimiento  $\epsilon$ ) y un movimiento que involucra a un símbolo de entrada. La condición 2 evita la elección de un movimiento para cualquier  $(q, a, Z)$  a  $(q, \epsilon, Z)$ . Advierta que, a diferencia del autómata finito, se entiende que un PDA es no determinístico, a menos que se diga lo contrario.

Para los autómatas finitos, los modelos determinístico y no determinístico eran equivalentes con respecto a los lenguajes aceptados. Esto no sucede así con un PDA. De hecho,  $ww^R$  es aceptado por un PDA no determinístico, pero no por cualquier PDA determinístico.

### 5.3 AUTOMATAS DE APILAMIENTO Y LENGUAJES LIBRES DE CONTEXTO

Demostraremos ahora el resultado fundamental, que consiste en que la clase de lenguajes aceptados por los PDAs es precisamente la clase de lenguajes libres de contexto. Mostraremos primero que los lenguajes aceptados por los PDAs mediante el acceso de un estado final son exactamente los lenguajes aceptados por los PDAs mediante agotamiento de pila. Despues mostraremos que los lenguajes aceptados mediante agotamiento de pila son exactamente los lenguajes libres de contexto.

#### Equivalencia de la aceptación mediante acceso de estado final y por agotamiento de pila

**Teorema 5.1** Si  $L$  es  $L(M_2)$  para algún PDA  $M_2$ , entonces  $L$  es  $N(M_1)$  para algún PDA,  $M_1$ .

*Demuestra* En pocas palabras, nos gustaría que  $M_1$  simulara a  $M_2$ , con la opción de que  $M_1$  borrar su pila cuando  $M_2$  accesaría un estado final. Utilizamos  $q_e$  de  $M_1$  para borrar la pila, y usamos  $X_0$  como un señalador del fondo de la pila, para  $M_1$ , así que  $M_1$  no acepta de manera accidental cuando  $M_2$  agota su pila sin accesar un estado final. Sea  $M_2 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  un PDA tal que  $L = L(M_2)$ . Hagamos

$$M_1 = (Q \cup \{q_e, q_0'\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q_0', X_0, \emptyset),$$

en donde  $\delta'$  se define de la manera siguiente.

1.  $\delta'(q_0', \epsilon, X_0) = \{q_0, Z_0, X_0\}$ .
2.  $\delta'(q, a, Z)$  incluye a los elementos de  $\delta(q, a, Z)$  para toda  $q$  en  $Q$ ,  $a$  en  $\Sigma$  o  $a = \epsilon$  y  $Z$  en  $\Gamma$ .
3. Para toda  $q$  en  $F$  y  $Z$  en  $\Gamma \cup \{X_0\}$ ,  $\delta'(q, \epsilon, Z)$  contiene a  $(q_e, \epsilon)$ .
4. Para toda  $Z$  en  $\Gamma \cup \{X_0\}$ ,  $\delta'(q_e, \epsilon, Z)$  contiene a  $(q_e, \epsilon)$ .

La regla (1) ocasiona que  $M_1$  accese a la ID inicial de  $M_2$ , pero  $M_1$  tendrá su propio señalador de fondo de la pila,  $X_0$ , que se encuentra por debajo de los símbolos de la pila de  $M_2$ . La regla (2) permite que  $M_1$  simule a  $M_2$ . Siempre que  $M_2$  accese un estado final, las reglas (3) y (4) permiten al  $M_1$  tener la alternativa de accesar el estado que y borrar su pila, por consiguiente aceptando la entrada, o continuando con la simulación de  $M_2$ . Debe hacerse notar que  $M_2$  posiblemente puede borrar su pila completa para alguna entrada  $x$  que no esté en  $L(M_2)$ . Esta es la razón por la cual  $M_1$  tiene su propio señalador de fondo de pila. De otra manera  $M_1$ , al simular a  $M_2$ , podría borrar por completo su propia pila, por tanto aceptaría  $x$  cuando no debiera hacerlo.

Hagamos que  $x$  esté en  $L(M_2)$ . Entonces  $(q_0, x, Z_0) \xrightarrow{*_{M_2}} (q, \epsilon, \gamma)$  para alguna  $q$  en  $F$ . Considérese ahora a  $M_1$  con entrada  $x$ . Mediante la regla (1),

$$(q_0', x, X_0) \xrightarrow{*_{M_1}} (q_0, x, Z_0 X_0),$$

Mediante la regla (2), cada movimiento de  $M_2$  es un movimiento legal para  $M_1$ , por consiguiente

$$(q_0, x, Z_0) \xrightarrow{*_{M_1}} (q, \epsilon, \gamma).$$

Si un PDA puede realizar una sucesión de movimientos a partir de una ID dada, puede hacer la misma sucesión de movimientos a partir de cualquier ID obtenido del primero mediante la inserción de una cadena fija de símbolos de pila por debajo del contenido de la pila original. Por tanto

$$(q_0, x, X_0) \xrightarrow{*_{M_1}} (q_0, x, Z_0 X_0) \xrightarrow{*_{M_1}} (q, \epsilon, \gamma X_0).$$

Por las reglas (3) y (4),

$$(q, \epsilon, \gamma X_0) \xrightarrow{*_{M_1}} (q_e, \epsilon, \epsilon).$$

En consecuencia,

$$(q_0', x, X_0) \xrightarrow{*_{M_1}} (q_e, \epsilon, \epsilon),$$

y  $M_1$  acepta a  $x$  mediante agotamiento de pila.

De manera inversa, si  $M_1$  acepta a  $x$  mediante agotamiento de pila, es sencillo mostrar que la sucesión de movimientos debe ser un movimiento según la regla (1), después una secuencia de movimientos según la regla (2) en la cual  $M_1$  simula la aceptación de  $x$  por  $M_2$ , seguido por la borrado de la pila de  $M_1$  utilizando las reglas (3) y (4). En consecuencia  $x$  debe estar en  $L(M_2)$ .  $\square$

**Teorema 5.2** Si  $L$  es  $N(M_1)$  para algún PDA  $M_1$ , entonces  $L$  es  $L(M_2)$  para algún PDA  $M_2$ .

*Demuestra* Nuestro plan, ahora, es tener a  $M_2$  que simule a  $M_1$  y detectar cuándo  $M_1$  vacía su pila.  $M_2$  accesa un estado final cuando y sólo cuando esto ocurre. Sea  $M_1 = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$  un PDA tal que  $L = L(M_1)$ . Hagamos

$$M_2 = (Q \cup \{q_0', q_f\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q_0', X_0, \{q_f\}),$$

en donde  $\delta'$  está definida de la manera siguiente:

1.  $\delta'(q_0', \epsilon, X_0) = \{(q_0, Z_0, X_0)\}$ .
  2. Para toda  $q$  en  $Q$ ,  $a$  en  $\Sigma \cup \{\epsilon\}$  y  $Z$  en  $\Gamma$ ,
- $$\delta'(q, a, Z) = \delta(q, a, Z).$$
3. Para toda  $q$  en  $Q$ ,  $\delta'(q, \epsilon, X_0)$  contiene a  $(q_f, \epsilon)$ .

La regla (1) ocasiona que  $M_2$  accese a la ID inicial de  $M_1$ , pero  $M_2$  tendrá su propio señalador de fondo de la pila,  $X_0$ , que se encuentra por debajo de los símbolos de la pila de  $M_1$ . La regla (2) permite a  $M_2$  simular a  $M_1$ . Siempre que  $M_1$  borre su pila completa,

$M_2$ , cuando se encuentra simulando a  $M_1$ , borrará su pila completa con excepción del símbolo  $X_0$  que se encuentra en el fondo. La regla (3) ocasiona que  $M_2$ , cuando aparece  $X_0$ , accese un estado final, aceptando, por consiguiente, la entrada  $x$ . La prueba de que  $L(M_1) = N(M_1)$  es similar a la demostración del Teorema 5.1 y se deja como ejercicio.

□

### Equivalecia de los PDAs y los CFLs

**Teorema 5.3** Si  $L$  es un lenguaje libre de contexto, entonces existe un PDA  $M$  tal que  $L = N(M)$ .

**Demostración** Supongamos que  $\epsilon$  no está en  $L(G)$ . El lector puede modificar la construcción para el caso en que  $\epsilon$  esté en  $L(G)$ . Sea  $G = (V, T, P, S)$  una gramática libre de contexto en la forma normal de Greibach que genera a  $L$ . Hagamos

$$M = (\{q\}, T, V, \delta, q, S, \emptyset),$$

en donde  $\delta(q, a, A)$  contiene a  $(q, \gamma)$  siempre que  $A \rightarrow a\gamma$  esté en  $P$ .

El PDA  $M$  simula a la derivación izquierda de  $G$ . Puesto que  $G$  está en la forma normal de Greibach, cada forma oracional de una derivación izquierda consiste en una cadena de terminales  $x$  seguida por una cadena de variables  $\alpha$ .  $M$  almacena el sufijo  $\alpha$  de la forma oracional restante en su pila, después de haber procesado al prefijo  $x$ . De manera formal mostraremos que

$$S \xrightarrow{*} x \alpha \text{ mediante una derivación izquierda si y sólo si } (q, x, S) \xrightarrow{M}^* (q, \epsilon, \alpha). \quad (5.1)$$

Primero suponemos que  $(q, x, S) \xrightarrow{M}^* (q, \epsilon, \alpha)$  y mostramos por inducción sobre  $i$  que  $S \xrightarrow{*} x\alpha$ . La base,  $i = 0$ , es trivial puesto que  $x = \epsilon$  y  $\alpha = S$ . Para la inducción, supóngase que  $i \geq 1$  y hágase  $x = ya$ . Considérese el siguiente paso,

$$(q, ya, S) \xrightarrow{i-1} (q, a, \beta) \xrightarrow{} (q, \epsilon, \alpha). \quad (5.2)$$

Si retiramos a  $a$  del final de la cadena de entrada en las primeras IDs  $i$  de la secuencia (5.2), descubrimos que  $(q, y, S) \xrightarrow{i-1} (q, \epsilon, \beta)$ , ya que  $a$  puede no tener efecto sobre los movimientos de  $M$  hasta que en sea efectivamente consumida de la entrada. Por la hipótesis inductiva  $S \xrightarrow{*} y\beta$ . El movimiento  $(q, a, \beta) \xrightarrow{} (q, \epsilon, \alpha)$  implica que  $\beta = A\gamma$  para alguna  $A$  en  $V$ ,  $A \rightarrow a\eta$  es una producción de  $G$  y  $\alpha = \eta\gamma$ . De ahí que

$$S \xrightarrow{*} y\beta \Rightarrow ya\eta\gamma = x\alpha,$$

y llegamos a la conclusión de que la parte “si” de (5.1) es verdadera.

Ahora supóngase que  $S \xrightarrow{i} x \alpha$  mediante una derivación izquierda. Mostraremos por inducción sobre  $i$  que  $(q, x, S) \xrightarrow{*} (q, \epsilon, \alpha)$ . De nuevo, la base,  $i = 0$ , es trivial. Sea  $i \geq 1$  y supóngase

$$S \xrightarrow{i-1} ya\gamma \Rightarrow ya\eta\gamma,$$

en donde  $x = ya$  y  $\alpha = \eta\gamma$ . Por la hipótesis de inducción,  $(q, y, S) \xrightarrow{*} (q, \epsilon, A\gamma)$  y, por tanto  $(q, ya, S) \xrightarrow{*} (q, a, A\gamma)$ . Como  $A \rightarrow a\eta$  es una producción, se concluye que  $\delta(q, a, A)$  contiene a  $(q, \eta)$ . Por consiguiente

$$(q, x, S) \xrightarrow{*} (q, a, A\gamma) \xrightarrow{} (q, \epsilon, \alpha),$$

y se concluye la parte “sólo si” de (5.1).

Para concluir la demostración, solamente es necesario hacer notar que (5.1) con  $\alpha = \epsilon$  tenemos  $S \xrightarrow{*} x$  si y sólo si  $(q, x, S) \xrightarrow{*} (q, \epsilon, \epsilon)$ . Esto es,  $x$  está en  $L(G)$  si y sólo si  $x$  está en  $N(M)$ . □

**Teorema 5.4** Si  $L$  está en  $N(M)$  para algún PDA  $M$ , entonces  $L$  es un lenguaje libre de contexto.

**Demostración** Sea  $M$  el PDA  $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$ . Sea  $G = (V, \Sigma, P, S)$  una gramática libre de contexto en la que  $V$  es el conjunto de objetos de la forma  $[q, A, p]$ ,  $q$  y  $p$  están en  $Q$  y  $A$  en  $\Gamma$ , más el nuevo símbolo  $S$ .  $P$  es el conjunto de producciones

1.  $S \rightarrow [q_0, Z_0, q]$  para cada  $q$  en  $Q$ ;
2.  $[q, A, q_{m+1}] \rightarrow a[q_1, B_1, q_2] [q_2, B_2, q_3] \dots [q_m, B_m, q_{m+1}]$  para cada  $q, q_1, q_2, \dots, q_{m+1}$  en  $Q$ , cada  $a$  en  $\Sigma \cup \{\epsilon\}$  y  $A, B_1, B_2, \dots, B_m$  en  $\Gamma$ , tal que  $\delta(q, a, A)$  contiene a  $(q_1, B_1 B_2 \dots B_m)$ . (Si  $m = 0$ , entonces la producción es  $[q, A, q_1] \rightarrow a$ .)

Para entender la prueba, es de ayuda saber que las variables y producciones de  $G$  han sido definidas de tal forma que una derivación izquierda en  $G$  de una oración  $x$  es una simulación del PDA  $M$  cuando se alimenta la entrada  $x$ . En particular, las variables que aparecen en cualquier paso de una derivación izquierda de  $G$  corresponden a los símbolos de la pila de  $M$ , en el instante en que  $M$  ha visto tanto de la entrada como cantidad de la gramática que ha sido ya generada. Dicho de otra manera, la intención es que  $[q, A, p]$  derive a  $x$  si y sólo si  $x$  ocasiona que  $M$  borre una  $A$  de su pila mediante una sucesión de movimientos que comience en el estado  $q$  y termine en el estado  $p$ .

Para mostrar que  $L(G) = N(M)$  demostramos por inducción sobre el número de pasos de una derivación de  $G$ , o sobre el número de movimientos de  $M$ , que

$$[q, A, p] \xrightarrow{G}^* x \text{ si y sólo si } (q, x, A) \xrightarrow{M}^* (p, \epsilon, \epsilon). \quad (5.3)$$

Primero mostramos, por inducción sobre  $i$ , que si  $(q, x, A) \xrightarrow{i} (p, \epsilon, \epsilon)$ , entonces  $[q, A, p] \xrightarrow{*} x$ . Si  $i = 1$ , entonces  $\delta(q, x, A)$  debe contener a  $(p, \epsilon)$ . (Aquí  $x$  es  $\epsilon$  o un símbolo de entrada simple). En consecuencia  $[q, A, p] \rightarrow x$  es una producción de  $G$ . Ahora supóngase que  $i > 1$ . Sea  $x = ay$

$$(q, ay, A) \xrightarrow{} (q_1, y, B_1 B_2 \dots B_n) \xrightarrow{i-1} (p, \epsilon, \epsilon).$$

La cadena  $y$  puede escribirse como  $y = y_1 y_2 \dots y_n$ , en donde  $y_j$  tiene el efecto de sacar a  $B_j$  de la pila, posiblemente después de una larga sucesión de movimientos. Es decir, sea  $y_1$  el prefijo de  $y$  al final del cual la pila por primera vez se acorta a  $n - 1$  símbolos. Sea  $y_2$  los símbolos de  $y$  que están después de  $y_1$ , tales que al final de  $y_2$ , la pila se hace, por primera vez, de longitud  $n - 2$  símbolos, y así sucesivamente. En la Fig. 5.5 se presenta el arreglo. Nótese que  $B_1$  no necesita ser el enésimo símbolo de pila a partir del fondo, durante todo el tiempo que  $M$  lee a  $y_1$ , ya que  $B_1$  puede cambiarse si se encuentra en la cima de la pila y sustituido por uno o más símbolos. Sin embargo, ninguno de los  $B_2 B_3 \dots B_n$  se encuentran nunca en la cima mientras  $y_1$  es leída  $y$ , de esta forma, no

pueden ser cambiadas o influir en el cálculo. En general,  $B_j$  permanece sin cambios en la pila mientras  $M$  lee a  $y_1, y_2 \dots y_{j-1}$ .

Existen estados  $q_2, q_3, \dots, q_{n+1}$ , en donde  $q_{n+1} = p$ , tales que

$$(q_j, y_j, B_j) \xrightarrow{*} (q_{j+1}, \epsilon, \epsilon)$$

mediante menos de  $i$  pasos ( $q_j$  es el estado accesado cuando la pila, por primera vez, llega a una longitud de  $n - j + 1$ ). Por tanto la hipótesis inductiva se aplica y

$$[q_j, B_j, q_{j+1}] \xrightarrow{*} y_j \quad \text{para } 1 \leq j \leq n.$$

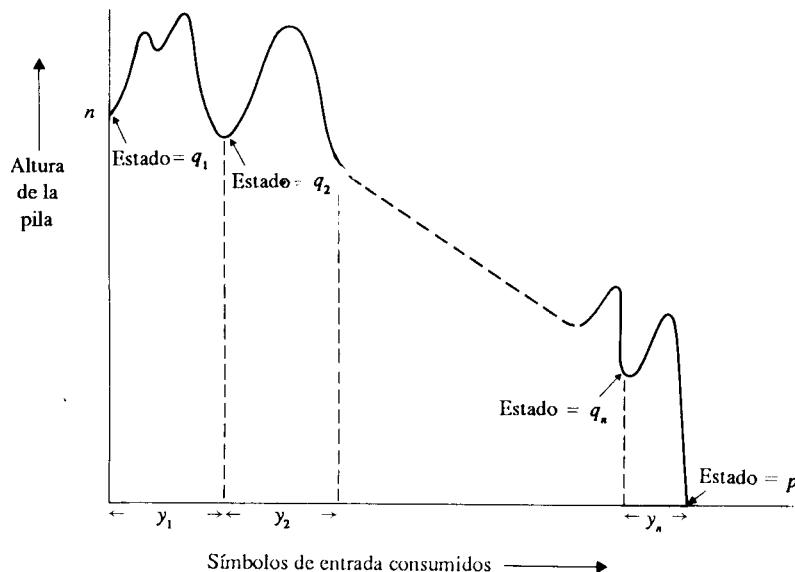


Fig. 5.5 Altura de la pila como función de la entrada consumida.

Si recordamos el movimiento original  $(q, ay, A) \xrightarrow{*} (q_1, y, B_1 B_2 \dots B_n)$ , sabremos que

$$[q, A, p] \Rightarrow a[q_1, B_1, q_2][q_2, B_2, q_3] \dots [q_n, B_n, q_{n+1}],$$

de modo que  $[q, A, p] \xrightarrow{*} ay, y_2 \dots y_n = x$ .

Ahora supóngase que  $[q, A, p] \xrightarrow{*} x$ . Mostramos por inducción sobre  $i$  que  $[(q, x, A) \xrightarrow{*} (p, \epsilon, \epsilon)]$ . La base,  $i = 1$ , se concluye de manera inmediata ya que  $[q, A, p] \xrightarrow{*} x$  debe ser una producción de  $G$  y por consiguiente  $\delta(q, x, A)$  debe contener a  $(p, \epsilon)$ . Nótese que  $x$  es  $\epsilon$  o está en  $\Sigma$ .

Para la inducción, supóngase que

$$[q, A, p] \Rightarrow a[q_1, B_1, q_2] \dots [q_n, B_n, q_{n+1}] \xrightarrow{i-1} x,$$

en donde  $q_{n+1} = p$ . Entonces podemos escribir  $x = ax_1 x_2 \dots x_n$ , en donde  $[q_j, B_j, q_{j+1}] \xrightarrow{*} x_j$  para  $1 \leq j \leq n$ , y con cada derivación llevándose a cabo con menos de  $i$  pasos. Por la

hipótesis de inducción,  $(q_j, x_j, B_j) \xrightarrow{*} (q_{j+1}, \epsilon, \epsilon)$  para  $1 \leq j \leq n$ . Si insertamos  $B_{j+1} \dots B_n$  en el fondo de cada pila en la sucesión anterior de IDs, vemos que

$$(q_j, x_j, B_j B_{j+1} \dots B_n) \xrightarrow{*} (q_{j+1}, \epsilon, B_{j+1} \dots B_n). \quad (5.4)$$

Del primer paso de la derivación de  $x$  a partir de  $[q, A, p]$  sabemos que

$$(q, x, A) \xrightarrow{*} (q_1, x_1 x_2 \dots x_n, B_1 B_2 \dots B_n)$$

es un movimiento legal de  $M$ , así que tomando en cuenta este movimiento y (5.4) para  $j = 1, 2, \dots, n$ , se concluye  $(q, x, A) \xrightarrow{*} (p, \epsilon, \epsilon)$ .

La demostración concluye con la observación de que (5.3), con  $q = q_0$  y  $A = Z_0$  queda

$$[q_0, Z_0, p] \xrightarrow{*} x \quad \text{si y sólo si } (q_0, x, Z_0) \xrightarrow{*} (p, \epsilon, \epsilon).$$

Esta observación, junto con la regla (1) de la construcción de  $G$ , nos dice que

$$S \xrightarrow{*} x \quad \text{si y sólo si } (q_0, x, Z_0) \xrightarrow{*} (p, \epsilon, \epsilon) \text{ para algún estado } p.$$

Esto es,  $x$  está en  $L(G)$  si y sólo si  $x$  está en  $N(M)$ .  $\square$

### Ejemplo 5.3 Sea

$$M = (\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, \delta, q_0, Z_0, \emptyset),$$

en donde  $\delta$  está dada por

$$\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}, \quad \delta(q_1, 1, X) = \{(q_1, \epsilon)\},$$

$$\delta(q_0, 0, X) = \{(q_0, XX)\}, \quad \delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\},$$

$$\delta(q_0, 1, X) = \{(q_1, \epsilon)\}, \quad \delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}.$$

Para construir una CFG  $G = (V, T, P, S)$  que genere a  $N(M)$  hagamos

$$V = \{S, [q_0, X, q_0], [q_0, X, q_1], [q_1, X, q_0], [q_1, X, q_1],$$

$$[q_0, Z_0, q_0], [q_0, Z_0, q_1], [q_1, Z_0, q_0], [q_1, Z_0, q_1]\}$$

y  $T = \{0, 1\}$ . Para construir el conjunto de producciones de manera sencilla, debemos darnos cuenta que algunas variables pueden no aparecer en ninguna derivación que comience con el símbolo  $S$ . Por tanto podemos ahorrar algún esfuerzo si comenzamos con las producciones para  $S$ , entonces agregamos producciones sólo para aquellas variables que aparecen en la derecha de alguna producción que ya se encuentre en el conjunto. Las producciones para  $S$  son

$$S \rightarrow [q_0, Z_0, q_0]$$

$$S \rightarrow [q_0, Z_0, q_1]$$

En seguida agregamos las producciones para la variable  $[q_0, Z_0, q_0]$ . Estas son

$$[q_0, Z_0, q_0] \rightarrow 0[q_0, X, q_0][q_0, Z_0, q_0]$$

$$[q_0, Z_0, q_0] \rightarrow 0[q_0, X, q_1][q_1, Z_0, q_0]$$

Estas producciones son requeridas por  $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$ . Ahora, las producciones para  $[q_0, Z_0, q_1]$  son

$$\begin{aligned}[q_0, Z_0, q_1] &\rightarrow 0[q_0, X, q_0][q_0, Z_0, q_1] \\ [q_0, Z_0, q_1] &\rightarrow 0[q_0, X, q_1][q_1, Z_0, q_1]\end{aligned}$$

Estas son requeridas también por  $\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$ . Las producciones para las variables restantes y los movimientos relevantes del PDA son:

1.  $[q_0, X, q_0] \rightarrow 0[q_0, X, q_0][q_0, X, q_0]$   
ya que  $\delta(q_0, 0, X) = \{(q_0, XX)\}$ .
2.  $[q_0, X, q_1] \rightarrow 1$  ya que  $\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$ .
3.  $[q_1, Z_0, q_1] \rightarrow \epsilon$  ya que  $\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$ .
4.  $[q_1, X, q_1] \rightarrow \epsilon$  ya que  $\delta(q_1, \epsilon, X) = \{(q_1, \epsilon)\}$ .
5.  $[q_1, X, q_1] \rightarrow 1$  ya que  $\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$ .

Debemos recalcar que no existen producciones para las variables  $[q_1, X, q_0]$  y  $[q_1, Z_0, q_0]$ . Como todas las producciones para  $[q_0, X, q_0]$  y  $[q_0, Z_0, q_0]$  tienen a  $[q_1, X, q_0]$  o a  $[q_1, Z_0, q_0]$  en la derecha, ninguna cadena terminal puede ser derivada a partir de  $[q_0, X, q_0]$  ni  $[q_0, Z_0, q_0]$ . Eliminando todas las producciones que involucran una de estas cuatro variables en la izquierda o la derecha, terminamos con las siguientes producciones.

$$\begin{array}{ll} S \rightarrow [q_0, Z_0, q_1], & [q_1, Z_0, q_1] \rightarrow \epsilon, \\ [q_0, Z_0, q_1] \rightarrow 0[q_0, X, q_1][q_1, Z_0, q_1], & [q_1, X, q_1] \rightarrow \epsilon, \\ [q_1, X, q_1] \rightarrow 0[q_0, X, q_1][q_1, X, q_1], & [q_1, X, q_1] \rightarrow 1, \\ [q_0, X, q_1] \rightarrow 1, & \end{array}$$

Resumimos los Teoremas 5.1 hasta 5.4 de la manera siguiente:

1.  $L$  es un lenguaje libre de contexto.
2.  $L = N(M)$  para algún PDA  $M$ .
3.  $L = L(M)$  para algún PDA  $M$ .

## EJERCICIOS

- 5.1 Constrúyase un autómata de apilamiento para cada uno de los lenguajes del Ejercicio 4.1.  
5.2 Constrúyase un PDA equivalente a la siguiente gramática:

$$S \rightarrow aAA, \quad A \rightarrow aS \mid bS \mid a.$$

- 5.3 Complete la demostración del Teorema 5.3 mostrando que cada CFL  $L$  es el conjunto aceptado por algún PDA aun si  $\epsilon$  está en  $L$ . [Sugerencia: Agregue un segundo estado al PDA para  $L - \{\epsilon\}$ .]

5.4 Muestre que si  $L$  es un CFL, entonces existe un PDA  $M$  que acepta a  $L$ , mediante acceso de estado final, tal que  $M$  tiene cuando mucho dos estados, y no hace movimientos.

\*5.5

- a) Muestre que si  $L$  es un CFL, entonces  $L$  es  $L(M)$  para algún PDA  $M$  tal que si  $\delta(q, a, X)$  contiene a  $(p, \gamma)$ , entonces  $|\gamma| \leq 2$ .
- b) Muestre que el  $M$  de la parte (a) puede restringirse aún más, de tal modo que si  $\delta(q, a, X)$  contiene a  $(p, \gamma)$ , entonces  $\gamma$  es  $\epsilon$  (movimiento de inserción),  $X$  (no hay cambio en la pila) o  $YX$  para algún símbolo de pila  $Y$  (movimiento de apilamiento).
- c) ¿Podemos colocar un límite en el número de estados de  $M$  de la parte (a) y aún tener un PDA para cualquier CFL?
- d) ¿Podemos limitar el número de estados de la parte (b)?

5.6 Dé una gramática para el lenguaje  $N(M)$  en donde

$$M = (\{q_0, q_1\}, \{0, 1\}, \{Z_0, X\}, \delta, q_0, Z_0, \emptyset)$$

y  $\delta$  está dada por

$$\begin{array}{ll} \delta(q_0, 1, Z_0) = \{(q_0, XZ_0)\}, & \delta(q_0, \epsilon, Z_0) = \{(q_0, \epsilon)\}, \\ \delta(q_0, 1, X) = \{(q_0, XX)\}, & \delta(q_1, 1, X) = \{(q_1, \epsilon)\}, \\ \delta(q_0, 0, X) = \{(q_1, X)\}, & \delta(q_1, 0, Z_0) = \{(q_0, Z_0)\}. \end{array}$$

5.7 El PDA determinístico (DPDA) no es equivalente al PDA no determinístico. Por ejemplo, el lenguaje

$$L = \{0^n 1^n \mid n \geq 1\} \cup \{0^n 1^{2n} \mid n \geq 1\}$$

es un CFL que no es aceptado por ningún DPDA.

- a) Muestre que  $L$  es un CFL.
- \*\* b) Demuestre que  $L$  no es aceptado por un DPDA.

5.8 Se dice que un lenguaje  $L$  tiene *propiedad de prefijo* si ninguna palabra de  $L$  es un prefijo propio de otra palabra de  $L$ . Muestre que si  $L$  es  $N(M)$  para el DPDA  $M$ , entonces  $L$  tiene la propiedad de prefijo. Lo que se dijo anteriormente es necesariamente verdadero si  $L$  es  $N(M)$  para un PDA determinístico  $M$ ?

\*5.9 Muestre que  $L$  es  $N(M)$  para algún DPDA  $M$  si y sólo si  $L$  es  $L(M')$  para algún DPDA  $M'$  y  $L$  tiene la propiedad de prefijo.

5.10 Un PDA de *dos direcciones* (2pda) es un PDA al que se le permite moverse en cualquier dirección en su entrada. Como el FA de dos direcciones, acepta mediante el movimiento hacia fuera del extremo derecho de su entrada en un estado final. Muéstrese que  $L = \{0^n 1^n 2^n \mid n \geq 1\}$  es aceptado por un 2PDA. Mostraremos en el siguiente capítulo, a propósito, que  $L$  no es un CFL, de modo que los 2 PDAs no son equivalentes a los PDAs.

\*5.11 Escriba un programa que traduzca una expresión regular en un autómata finito.

\*5.12 La gramática

$$E \rightarrow E + E \mid E * E \mid (\cdot) \mid id \quad (5.5)$$

genera el conjunto de expresiones aritméticas con  $+$ ,  $*$ , paréntesis e  $id$  en notación *infix* (operador entre los operandos). La gramática

$$P \rightarrow + PP \mid * PP \mid id$$

genera al conjunto de expresiones aritméticas en notación prefija (el operador antecede a los operandos). Construya un programa que traduzca las expresiones aritméticas en notación infija a notación prefija utilizando la siguiente técnica. Diseñe un PDA determinístico que analice gramaticalmente la expresión infija de acuerdo con la gramática dada en (5.5). Para cada vértice del árbol de análisis gramatical determine la acción necesaria para producir la expresión prefija deseada. [Sugerencia: Vea la solución al ejercicio 5.11.]

**5.13** Construya un compilador para las expresiones aritméticas infijas que produzca un programa en lenguaje ensamblador para evaluar la expresión. Suponga que el lenguaje ensamblador tiene las instrucciones de dirección simple: LOAD  $x$  (copia  $x$  en el acumulador), ADD  $x$  (agrega  $x$  al acumulador), MULT  $x$  (multiplica el contenido del acumulador por  $x$ ) y STO  $x$  (almacena el contenido del acumulador en  $x$ ).

### Soluciones a los ejercicios seleccionados

**5.11** Escribir un programa que traduzca una expresión regular a un autómata finito puede pensarse como la construcción de un compilador rudimentario. Ya hemos visto (Teorema 2.3) que un autómata finito que acepta  $\emptyset$ ,  $\epsilon$ , 0 y 1 puede combinarse para obtener un autómata equivalente a una expresión regular dada. El único problema consiste en analizar gramaticalmente la expresión regular para determinar el orden en el cual combinar el autómata.

Nuestro primer paso es construir una CFG para el conjunto de las expresiones regulares. El paso siguiente consiste en escribir un analizador gramatical y, finalmente, las rutinas generadoras del autómata.

A continuación se da una gramática para las expresiones regulares que agrupe a las subexpresiones de acuerdo con la prioridad convencional de las operaciones. Nótese que  $\epsilon$  se utiliza en lugar del símbolo  $\epsilon$ .

$$\begin{aligned} E &\rightarrow P + E \mid P \\ P &\rightarrow T \cdot P \mid T \\ T &\rightarrow 0 \mid 1 \mid \epsilon \mid \emptyset \mid T^* \mid (E) \end{aligned}$$

La rutina de análisis se construye directamente a partir de la gramática escribiendo un procedimiento para cada variable. Una variable global CADENA contiene inicialmente a la siguiente expresión regular.

```
procedure ENCONTRAR_EXPRESION;
begin
    ENCONTRAR_PRODUCTO;
    while primer símbolo de CADENA es + do
        begin
            borra el primer símbolo de CADENA;
            ENCONTRAR_PRODUCTO
        end;
    end ENCONTRAR_EXPRESION;
    procedure ENCONTRAR_PRODUCTO;
    begin
        ENCONTRAR_TERMINO;
        while primer símbolo de CADENA es • do
```

```
begin
    borra el primer símbolo de CADENA;
    ENCONTRAR_TERMINO
end
end ENCONTRAR_PRODUCTO;
procedure ENCONTRAR TERMINO;
begin
    if primer símbolo de CADENA es 0, 1,  $\epsilon$ , o  $\emptyset$  then
        borra el primer símbolo de CADENA;
    else if primer símbolo de CADENA es ( then begin
        borra primer símbolo de CADENA;
        ENCUENTRA EXPRESION;
    if primer símbolo de CADENA es ) then
        borra primer símbolo de CADENA;
    else error
    end
while primer símbolo de CADENA es * do
    borra el primer símbolo de CADENA
end ENCONTRAR_TERMINO
```

El verdadero programa de análisis gramatical consiste en un solo llamado de procedimiento:

ENCONTRAR\_EXPRESION;

Nótese que los procedimientos recursivos ENCONTRAR\_EXPRESION, ENCONTRAR\_PRODUCTO y ENCONTRAR\_TERMINO no tienen variables locales. En consecuencia, pueden ser implementados mediante una pila que empuje a E, P o T respectivamente, cuando es llamado un procedimiento, e inserte el símbolo cuando el procedimiento regrese. (Aunque ENCONTRAR\_EXPRESION tiene dos llamados a ENCONTRAR\_PRODUCTO, ambas llamadas regresan al mismo punto en ENCONTRAR\_EXPRESION. Por tanto la posición del regreso no necesita ser almacenada. Un comentario similar se hace con referencia a ENCONTRAR\_PRODUCTO). Así pues un PDA determinístico es suficiente para ejecutar el programa que hemos definido.

Con el desarrollo del procedimiento para analizar gramaticalmente una expresión regular, agregamos ahora los planteamientos para obtener un autómata finito. Cada procedimiento se modifica para que regrese un autómata finito. En el procedimiento ENCONTRAR TERMINO, si el símbolo de entrada es 0, 1,  $\epsilon$  o  $\emptyset$  se crea un autómata finito que acepte a 0, 1,  $\epsilon$  o  $\emptyset$  y ENCONTRAR\_TERMINO regresa este autómata. Si el símbolo de entrada es entonces el autómata finito regresado por ENCONTRAR\_EXPRESION es el valor de ENCONTRAR\_TERMINO. En cualquier caso, si el ciclo "while" para \* es ejecutado, el autómata es modificado para aceptar la cerradura.

En el procedimiento ENCONTRAR\_PRODUCTO, el valor de ENCONTRAR\_PRODUCTO es el valor del primer llamado de ENCONTRAR\_TERMINO. Cada vez que la instrucción "while" se ejecuta, el valor de ENCONTRAR\_PRODUCTO se asigna a un autómata que acepte a la concatenación de los conjuntos aceptados por el valor actual de ENCONTRAR\_PRODUCTO y el autómata regresado por el llamado a ENCONTRAR\_TERMINO en el ciclo "while". Planteamientos similares se añaden al procedimiento ENCONTRAR\_EXPRESION.

## NOTAS BIBLIOGRAFICAS

El autómata de apilamiento aparece como construcción formal en Oettinger [1961] y Schutzenberger [1963]. Su equivalencia con las gramáticas libres de contexto fue percibida por Chomsky [1962] y Evey [1963].

Se han estudiado un gran número de dispositivos parecidos. Las máquinas contadoras tienen sólo un símbolo de apilamiento, con excepción del señalador del fondo de pila. Se discuten en Fischer [1966], y Fischer, Meyer y Rosenberg [1968]; véanse también las notas bibliográficas del Capítulo 7. Los transductores de apilamiento son PDAs que pueden imprimir símbolos en cada movimiento. Han sido estudiados por Evey [1963], Fischer [1963], Ginsburg y Rose [1966], Ginsburg y Greibach [1966b] y Lewis y Stearns [1968].

El PDA de dos direcciones mencionado en el Ejercicio 5.10 ha sido estudiado por Hartmanis, Lewis y Stearns [1965]. Sus propiedades de cerradura fueron consideradas por Gray, Harrison e Ibarra [1967] y las caracterizaciones de la clase de los lenguajes aceptados por las variedades determinísticas (2DPDA) y no determinística (2NPDA) fueron dadas por Aho, Hopcroft y Ullman [1968] y Cook [1971c]. Este último contiene el notable resultado de que cualquier lenguaje aceptado por un 2DPDA es reconocible en tiempo lineal en una computadora. Por consiguiente, la existencia de un CFL que requiere más que tiempo lineal para ser reconocido en una computadora, implicaría que existen CFLs no aceptador por los 2DPDAs. Sin embargo, nadie hasta la fecha ha demostrado que tal lenguaje existe. Incidentalmente, el lenguaje  $\{0^n 1^n 2^n \mid n \geq 1\}$  es un ejemplo de un no CFL aceptado por un 2DPDA.

## CAPITULO

## 6

PROPIEDADES  
DE LOS LENGUAJES  
LIBRES DE CONTEXTO

En gran cantidad este capítulo es llevado en forma paralela al Capítulo 3. Daremos primero un lema de sondeo para lenguajes libres de contexto y lo utilizaremos para mostrar que ciertos lenguajes no son libres de contexto. Después consideraremos las propiedades de cerradura de los CFLs y, por último, daremos algoritmos para responder a ciertas interrogaciones acerca de los CFLs.

## 6.1 LEMA DE SONDEO PARA CFLs

El lema de sondeo para conjuntos regulares establece que toda cadena, lo suficientemente larga, de un conjunto regular contiene una cadena corta que puede ser “sondeada”. Es decir, insertar tantas copias de la subcadena como se desee siempre produce una cadena que está en el conjunto regular. El lema de sondeo para CFLs establece que siempre existen dos subcadenas cortas cercanas que pueden repetirse el mismo número de veces y con la frecuencia que se deseé. El planteamiento formal del lema de sondeo es el siguiente.

**Lema 6.1** (Lema de sondeo para lenguajes libres de contexto). Sea  $L$  cualquier CFL. Entonces existe una constante  $n$ , que depende solamente de  $L$ , tal que si  $z$  está en  $L$  y  $|z| \geq n$ , entonces podemos escribir  $z = uvwxy$  tal que

1.  $|vx| \geq 1$ ,
2.  $|vwx| \leq n$  y
3. para toda  $i \geq 0$   $uv^i wx^i$  y está en  $L$ .

**Demostración** Sea  $G$  una gramática en la forma normal de Chomsky que genere a  $L - \{\epsilon\}$ . Obsérvese que si  $z$  está en  $L(G)$  y  $z$  es larga, entonces cualquier árbol de derivación para  $z$  deberá contener una trayectoria larga. De manera más precisa, mostraremos por inducción sobre  $i$  que si el árbol de análisis gramatical de una palabra, generada por una gramática en la forma normal de Chomsky, no tiene una trayectoria mayor que  $i$ , entonces la palabra tiene una longitud no mayor a  $2^{i-1}$ . La base,  $i = 1$ , es trivial, puesto que el árbol debe ser de la forma que se presenta en la Fig. 6.1 (a). Para el paso inductivo, hagamos  $i > 1$ . Sean la raíz y sus hijos de la forma en que se muestra en la Fig. 6.1(b). Si no existen trayectorias de longitud mayor a  $i - 1$  en los árboles  $T_1$  y  $T_2$ , entonces los árboles generan palabras de  $2^{i-1}$  símbolos o menos. Por tanto, el árbol completo genera una palabra de longitud no mayor a  $2^{i-1}$ .

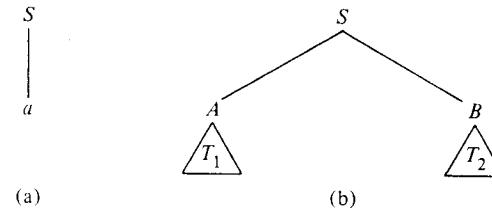


Fig. 6.1 Árboles de derivación.

Hagamos que  $G$  tenga  $k$  variables y  $n = 2^k$ . Si  $z$  está en  $L(G)$  y  $|z| \geq n$ , entonces como  $|z| > 2^{k-1}$ , cualquier árbol de análisis gramatical para  $z$  debe tener una trayectoria con una longitud de al menos  $k + 1$ . Pero una trayectoria tal tiene cuando menos  $k + 2$  vértices y todos éstos, con excepción del último, están etiquetados por variables. Por consiguiente deberá existir alguna variable que aparezca dos veces en la trayectoria.

De hecho podemos decir más. Algunas variables pueden aparecer dos veces cerca del fondo de la trayectoria. En particular, sea  $P$  una trayectoria cuya longitud sea igual o mayor que cualquier trayectoria del árbol. Entonces deben existir dos vértices  $v_1$  y  $v_2$  sobre la trayectoria que satisfacen las condiciones siguientes.

1. Los vértices  $v_1$  y  $v_2$  tienen la misma etiqueta, digamos  $A$ .
2. El vértice  $v_1$  está más cerca de la raíz que el vértice  $v_2$ .
3. La parte de la trayectoria que va de  $v_1$  a la hoja tiene longitud de al menos  $k + 1$ .

Para ver que siempre se pueden encontrar  $v_1$  y  $v_2$ , sólo hay que seguir la trayectoria  $P$  en sentido inverso, a partir de la hoja, manteniendo un registro de las etiquetas que se encontraron. De los  $k + 2$  vértices, solamente la hoja tiene una terminal como etiqueta. Los  $k + 1$  vértices restantes no pueden tener etiquetas con variables distintas.

Ahora el subárbol  $T_1$  con raíz  $v_1$  representa la derivación de una subpalabra cuya longitud es cuando mucho de  $2^k$ . Lo anterior es en efecto así, puesto que no puede existir una trayectoria en  $T_1$  de longitud mayor a  $k + 1$ , ya que  $P$  fue tomada como la trayectoria más larga del árbol completo. Hagamos  $z_1$  el producto del subárbol  $T_1$ . Si  $T_2$  es el subárbol generado por el vértice  $v_2$  y  $z_2$  el producto del subárbol  $T_2$ , entonces podemos escribir

$z_1$  como  $z_3 z_2 z_4$ . Aún más,  $z_3$  y  $z_4$  no pueden ser ambos  $\epsilon$ , ya que la primera producción utilizada en la derivación de  $z_1$  debe ser de la forma  $A \rightarrow BC$  para algunas variables  $B$  y  $C$ . El subárbol  $T_2$  debe estar completamente dentro del subárbol generado por  $B$  o del generado por  $C$ . Lo anterior se ilustra en la Fig. 6.2.

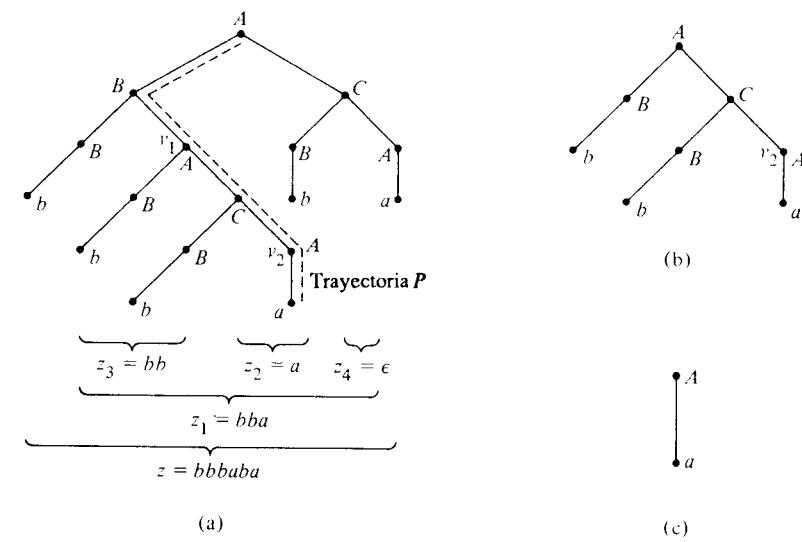


Fig. 6.2 Ilustración de los subárboles  $T_1$  y  $T_2$  del lema 6.1. a) Árbol. b) Subárbol  $T_1$ . c) Subárbol  $T_2$ .

Ahora sabemos que

$$A \xrightarrow[G]{*} z_1 z_2 z_4 \quad y \quad A \xrightarrow{*} z_2, \quad \text{en donde } |z_3 z_2 z_4| \leq 2^k = n.$$

Pero se concluye que  $A \xrightarrow[G]{*} z_1^i z_2 z_3^i$  para cada  $i \geq 0$ . (véase Fig. 6.3) Es claro que la cadena  $z$  se puede escribir como  $uz_3 z_2 z_4 y$ , para algunas  $u$  y  $y$ . Para completar la prueba dejamos  $z_3 = v$ ,  $z_2 = w$ ,  $y z_4 = x$ .  $\square$

### Aplicaciones del lema de sondeo

El lema de sondeo puede ser utilizado para demostrar que una variedad de lenguajes no son libres de contexto, usando el mismo argumento “adversario” que se utilizó en el lema de sondeo para conjuntos regulares.

**Ejemplo 6.1** Considérese el lenguaje  $L_1 = \{a^i b^i c^i \mid i \geq 1\}$ . Supóngase a  $L$  como si fuera libre de contexto y sea  $n$  la constante del Lema 6.1. Considérese también  $z = a^n b^n c^n$ . Escribáse a  $z = uvwxy$  de modo que se satisfagan las condiciones del lema de sondeo. Debemos preguntarnos dónde pudieron estar  $v$  y  $x$ , las cadenas que son sondeadas, en

$a^n b^n c^n$ . Debido a que  $|vwx| \leq n$ , no es posible que  $vx$  tenga  $as$  o  $cs$ , porque la  $a$  que se encuentra más a la derecha está  $n + 1$  posiciones más allá de la  $c$  que está más a la izquierda. Si  $v$  y  $x$  consisten solamente en  $as$ , entonces  $uwy$  (la cadena  $uv^i wx^i$  y con  $i = 0$ ) tiene  $n$   $bs$  y  $n$   $cs$  pero menos de  $n$   $as$ , ya que  $|vx| \geq 1$ . Por tanto,  $uwy$  no es de la forma  $a^i b^j c^j$ . Pero según el lema de sondeo  $uwy$  está en  $L_1$ , lo que constituye una contradicción.

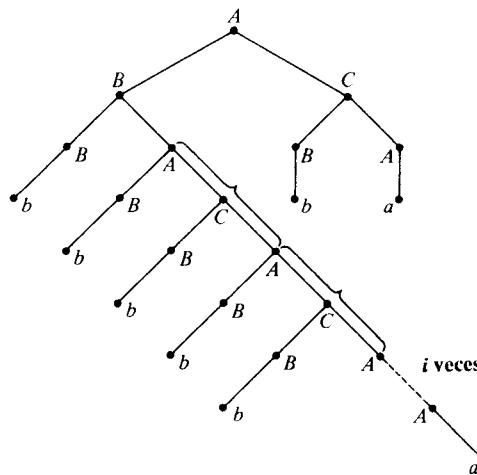


Fig. 6.3 Árbol de derivación de  $uvwxy$ , en donde  $u = b$ ,  $v = bb$ ,  $w = a$ ,  $x = \epsilon$ ,  $y = ba$ .

Los casos en que  $v$  y  $x$  consisten solamente en  $bs$  o sólo en  $cs$  se tratan de manera parecida. Si  $vx$  tiene  $as$  y  $bs$ , entonces  $uwy$  tiene más  $cs$  que  $as$  o  $bs$ , y, de nuevo, no se encuentra en  $L_1$ . Si  $vx$  contiene  $bs$  y  $cs$ , se tiene como resultado una contradicción parecida. Concluimos que  $L_1$  no es un lenguaje libre de contexto.

El lema de sondeo puede utilizarse también para mostrar que ciertos lenguajes parecidos a  $L_1$  no son libres de contexto. Como ejemplos tenemos

$$\{a^i b^i c^j \mid j \geq i\} \quad \text{y} \quad \{a^i b^j c^k \mid i \leq j \leq k\}.$$

En el siguiente ejemplo se ilustra otro tipo de relación que una CFG no puede hacer que se cumpla.

**Ejemplo 6.2** Sea  $L_2 = \{a^i b^j c^i d^j \mid i \geq 1 \text{ y } j \geq 1\}$ . Supóngase que  $L_2$  es un CFL y sea  $n$  la constante del Lema 6.1. Considerese la cadena  $z = a^i b^j c^i d^j$ . Hagamos que  $z = uvwxy$  satisfaga las condiciones del lema de sondeo. Entonces como  $|vwx| \leq n$ ,  $vx$  puede contener cuando mucho dos símbolos diferentes. Aún más, si  $vx$  contiene dos símbolos diferentes deberán ser consecutivos, por ejemplo,  $a$  y  $b$ . Si  $vx$  tiene sólo  $as$ , entonces  $uwy$  tiene menos  $as$  que  $cs$  y no se encuentra en  $L_2$ , lo que es una contradicción. Procedemos de manera similar a  $vx$  consiste solamente en  $bs$ ,  $cs$  o  $ds$ . Ahora supóngase que  $vx$  tiene  $as$  y  $bs$ . Entonces  $uwy$  aún tiene menos  $as$  que  $cs$ . Se presenta una contradicción similar cuando  $vx$  consiste en  $bs$  y  $cs$  o  $cs$  y  $ds$ . Como éstas son las únicas posibilidades, concluimos que  $L_2$  no es libre de contexto.

### Lema de Ogden

Existen ciertos lenguajes que no son libres de contexto para los cuales no es de ayuda el lema de sondeo. Por ejemplo,

$$L_3 = \{a^i b^j c^k d^l \mid \text{para todo } i = 0 \text{ o } j = k = l\}$$

no es libre de contexto. Sin embargo, si escogemos  $z = b^i c^k d^k$  y escribimos  $z = uvwxy$ , entonces siempre es posible elegir  $u$ ,  $v$ ,  $w$ ,  $x$  y  $y$  de manera que  $uv^m wx^m y$  esté en  $L_3$  para toda  $m$ . Por ejemplo, escogemos  $vwx$  que contenga sólo  $bs$ . Si tomamos  $z = a^i b^i c^i d^i$ , entonces  $v$  y  $x$  pueden consistir en solamente  $as$ , en cuyo caso  $uv^m wx^m y$  está de nuevo en  $L_3$  para toda  $m$ .

Lo que necesitamos es una versión más fuerte del lema de sondeo que nos permita centrar nuestra atención en algún número pequeño de posiciones de la cadena y sondearlos. Este tipo de extensión se hace fácil para conjuntos regulares, ya que cualquier secuencia de  $n + 1$  estados de un FA de  $n$  estados debe contener dos veces algún estado y la cadena que interviene puede ser sondada. El resultado para CFLs es mucho más difícil de obtener pero puede demostrarse. Aquí planteamos y demostramos una versión débil de lo que se conoce como lema de Ogden.

**Lema 6.2** (Lema de Ogden). Sea  $L$  un CFL. Entonces existe una constante  $n$  (que, de hecho, puede ser la misma del lema de sondeo) tal que si  $z$  es cualquier palabra de  $L$ , y si señalamos cualesquier  $n$  posiciones de  $z$  o más como "distinguidas", entonces podemos escribir  $z = uvwxy$ , tal es que:

1.  $v$  y  $x$  juntas tienen cuando menos una posición distinguida,
2.  $vwx$  tiene cuando mucho  $n$  posiciones distinguidas, y
3. para toda  $i \geq 0$ ,  $uv^i wx^i y$  está en  $L$ .

**Demostración** Sea  $G$  una gramática en la forma normal de Chomsky que genera al lenguaje  $L - \{\epsilon\}$ . Hagamos que  $G$  tenga  $k$  variables y escójase  $n = 2^k + 1$ . Debemos construir una trayectoria  $P$  en el árbol parecida a la trayectoria  $P$  que se construyó en la demostración del lema de sondeo. Sin embargo, ya que sólo nos preocuparemos aquí por posiciones distinguidas, no podemos interesarnos por cada vértice a lo largo de  $P$ , sino que sólo por los *puntos rama*, que son vértices cuyos dos hijos tienen descendientes distinguidos.

Construyase  $P$  de la forma siguiente: Comience por colocar la raíz en la trayectoria  $P$ . Supóngase que  $r$  es el último vértice colocado en  $P$ . Si  $r$  es una hoja, terminamos. Si  $r$  es solamente un hijo con descendientes distinguidos, añada ese hijo a  $P$  y repítase ahí el proceso. Si ambos hijos de  $r$  tienen descendientes distinguidos, llámese a  $r$  punto rama y agregue a  $P$  el hijo con el mayor número de descendientes distinguidos (rompa de manera arbitraria una atadura). Este proceso se ilustra en la Fig. 6.4.

Se concluye que cada punto rama de  $P$  tiene al menos la mitad de descendientes distinguidos que el punto rama anterior. Como existen al menos  $n$  posiciones distinguidas en  $z$ , y todas ellas son descendientes de la raíz, se concluye que al menos existen  $k + 1$  puntos rama en  $P$ . En consecuencia, entre los últimos  $k + 1$  puntos rama hay dos con la misma etiqueta. Podemos seleccionar  $v_1$  y  $v_2$  como los dos puntos

rama que tienen la misma etiqueta y  $v_1$  más cerca de la raíz que  $v_2$ . La demostración, entonces, se desarrolla de manera idéntica al lema de sondeo.  $\square$

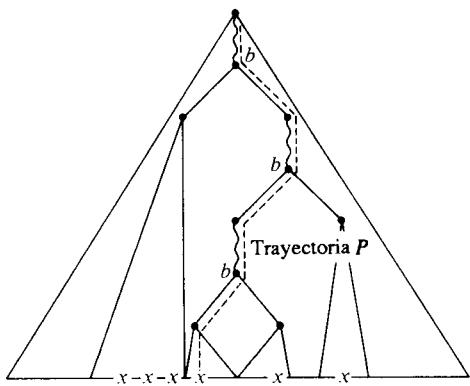


Fig. 6.4 Trayectoria  $P$ . Las posiciones distinguidas están marcadas con  $x$ . Los puntos rama están marcados con  $b$ .

**Ejemplo 6.3** Sea  $L_4 = \{a^i b^j c^k \mid i \neq j, j \neq k \text{ e } i \neq k\}$ . Supóngase que  $L_4$  es un lenguaje libre de contexto. Sea  $n$  la constante del lema de Ogden y considérese la cadena  $z = a^n b^{n+1} c^{n+2n}$ . Hagamos que las posiciones de  $a$  sean distinguidas y que  $z = uvwx$  satisfaga las condiciones del lema de Ogden. Si  $v$  o  $x$  contienen dos símbolos distintos, entonces  $uv^2wx^2y$  no está en  $L_4$ . (Por ejemplo, si  $v$  está en  $a^*b^*$ , entonces  $uv^2wx^2y$  tiene una  $b$  que antecede a una  $a$ .) Ahora, al menos una  $v$  o  $x$ , debe contener  $as$ , ya que solamente  $as$  se encuentran en posiciones distinguidas. Por tanto, si  $x$  está en  $b^*$  o  $c^*$ ,  $v$  debe estar en  $a^*$ . Si  $x$  está en  $a^*$ , entonces  $v$  debe estar en  $a^*$ ; de otra manera una  $b$  o una  $c$  deben anteceder a una  $a$ . Consideraremos con detalle la situación en la que  $x$  está en  $b^*$ . Los otros casos se tratan de manera parecida. Supóngase que  $x$  está en  $b^*$  y  $v$  en  $a^*$ . Se ap =  $|v|$ . Entonces  $1 \leq p \leq n$ , de modo que  $p$  divide a  $n!$  Sea  $q$  el entero tal que  $pq = n!$  Entonces

$$z' = uv^{2q+1}wx^{2q+1}y$$

está en  $L_4$ . Pero  $v^{2q+1} = a^{2pq+p} = a^{2n!+p}$ . Como  $uwy$  contiene exactamente  $(n-p)as$ ,  $z'$  tiene  $(2n! + n)as$ . Sin embargo, puesto que  $v$  y  $x$  no tienen  $cs$ ,  $z'$  tiene también  $(2n! + n)cs$  y en consecuencia no está en  $L_4$ , lo que contribuye una contradicción. Se presenta una contradicción parecida si  $x$  está en  $a^*$  o en  $c^*$ . Por tanto  $L_4$  no es un lenguaje libre de contexto.

Nótese que el lema 6.1 es un caso especial del lema de Ogden en el que todas las posiciones son distinguidas.

## 6.2 PROPIEDADES DE CERRADURA PARA CFLS

Consideraremos ahora algunas operaciones que conservan a los lenguajes libres de contexto. Las operaciones son útiles no sólo para construir ciertos lenguajes libres de contexto o demostrar que son libres de contexto, sino también para demostrar que ciertos lenguajes no son libres de contexto. Puede demostrarse que un lenguaje dado  $L$  no es libre de contexto mediante la construcción, a partir de  $L$ , de un lenguaje que no es libre de contexto utilizando sólo operaciones que conserven a los CFLs.

**Teorema 6.1** Los lenguajes libres de contexto son cerrados con respecto a la unión, concatenación y cerradura de Kleene.

**Demostración** Sea  $L_1$  y  $L_2$  CFLs generados por las CFGs

$$G_1 = (V_1, T_1, P_1, S_1) \quad \text{y} \quad G_2 = (V_2, T_2, P_2, S_2),$$

respectivamente. Puesto que podemos renombrar variables a voluntad sin cambiar el lenguaje generado, suponemos que  $V_1$  y  $V_2$  son disjuntos. Supóngase también que  $S_3$ ,  $S_4$  y  $S_5$  no están en  $V_1$  o en  $V_2$ .

Para  $L_1 \cup L_2$  construyese una gramática  $G_3 = (V_1 \cup V_2 \cup \{S_3\}, T_1 \cup T_2, P_3, S_3)$ , en donde  $P_3$  es  $P_1 \cup P_2$  más las producciones  $S_3 \xrightarrow{*} S_1 \mid S_2$ . Si  $w$  está en  $L_1$ , entonces la derivación  $S_3 \xrightarrow{G_3} S_1 \xrightarrow{G_1} w$  es una derivación en  $G_3$ , porque cada producción de  $G_1$  es una producción de  $G_3$ . De manera parecida, cada palabra de  $L_2$  tiene una derivación en  $G_3$  que comienza con  $S_3 \xrightarrow{*} S_2$ . En consecuencia,  $L_1 \cup L_2 \subseteq L(G_3)$ . Para la parte recíproca, sea  $w$  una palabra en  $L(G_3)$ . Entonces la derivación  $S_3 \xrightarrow{*} w$  comienza con  $S_3 \xrightarrow{G_3} S_1 \xrightarrow{G_1} w$  o con  $S_3 \xrightarrow{G_3} S_2 \xrightarrow{G_2} w$ . En el primer caso, como  $V_1$  y  $V_2$  son disjuntos, sólo los símbolos de  $G_1$  pueden aparecer en la derivación  $S_1 \xrightarrow{*} w$ . Como las únicas producciones de  $P_3$  que involucran solamente símbolos de  $G_1$  son las de  $P_1$ , concluimos que sólo las producciones de  $P_1$  se utilizan en la derivación  $S_1 \xrightarrow{*} w$ . Por consiguiente  $S_1 \xrightarrow{G_1} w$ , y  $w$  está en  $L_1$ . De manera análoga, si la derivación comienza con  $S_3 \xrightarrow{G_3} S_2$ , podemos concluir que  $w$  está en  $L_2$ . De aquí que  $L(G_3) \subseteq L_1 \cup L_2$ , de modo que  $L(G_3) = L_1 \cup L_2$  como se desea.

Para la concatenación, sea  $G_4 = (V_1 \cup V_2 \cup \{S_4\}, T_1 \cup T_2, P_4, S_4)$ , en donde  $P_4$  es  $P_1 \cup P_2$  más la producción  $S_4 \xrightarrow{*} S_1 S_2$ . La demostración para  $L(G_4) = L(G_1)L(G_2)$  es similar a la prueba para la unión y no se incluye.

Para la cerradura, hagamos  $G_5 = (V_1 \cup \{S_5\}, T_1, P_5, S_5)$ , en donde  $P_5$  es  $P_1$  más las producciones  $S_5 \xrightarrow{*} S_1 S_5 \mid \epsilon$ . De nueva cuenta dejamos la demostración de que  $L(G_5) = L(G_1)^*$  al lector.  $\square$

### Sustitución y homomorfismos

**Teorema 6.2** Los lenguajes libres de contexto son cerrados con respecto a la sustitución.

**Demostración** Sea  $L$  una CFL,  $L \subseteq \Sigma^*$  y para cada  $a$  en  $\Sigma$  tomemos  $L_a$  como un CFL. Sea  $L \subseteq L(G)$  y para cada  $a$  en  $\Sigma$  sea  $L_a \subseteq L(G_a)$ . Sin pérdida de generalidad podemos suponer que las variables de  $G$  y de las  $G_a$  son disjuntas. Constrúyase una gramática  $G'$

de la forma siguiente. Las variables de  $G'$  son todas las variables de las  $G$  y de las  $G_a$ ; las terminales de  $G'$  son las terminales de las  $G_a$ . El símbolo inicial de  $G'$  es el símbolo inicial de  $G$ . Las producciones de  $G'$  son todas las producciones de las  $G_a$  junto con aquellas producciones que se forman al tomar una producción  $A \rightarrow \alpha$  de  $G$  y sustituir por  $S_a$ , el símbolo inicial de  $G_a$ , en cada caso de  $a$  en  $\Sigma$  que aparezca en  $\alpha$ .  $\square$

**Ejemplo 6.4** Sea  $L$  el conjunto de palabras que tienen un número igual de  $a$ s y de  $b$ s.  $L_a = \{0^n1^n \mid n \geq 1\}$  y  $L_b = \{ww^k \mid w \text{ está en } (0+2)^*\}$ . Para  $G$  podemos elegir

$$S \rightarrow aShS \mid bSaS \mid \epsilon$$

Para  $G_a$  tómese

$$S_a \rightarrow 0S_a1 \mid 01$$

Para  $G_b$  tómese

$$S_b \rightarrow 0S_b0 \mid 2S_b2 \mid \epsilon$$

Si  $f$  es una sustitución  $f(a) = L_a$  y  $f(b) = L_b$ , entonces  $f(L)$  es generado por la gramática

$$S \rightarrow S_aSS_bS \mid S_bSS_aS \mid \epsilon$$

$$S_a \rightarrow 0S_a1 \mid 01$$

$$S_b \rightarrow 0S_b0 \mid 2S_b2 \mid \epsilon$$

Uno puede observar que, como  $\{a, b\}$ ,  $\{ab\}$  y  $a^*$  son CFLs, la cerradura de los CFLs con respecto a la sustitución implica la cerradura con respecto a la unión, la concatenación y  $*$ . La unión de  $L_a$  y  $L_b$  es simplemente la sustitución de  $L_a$  y  $L_b$  en  $\{a, b\}$  y, de manera similar,  $L_aL_b$  y  $L_a^*$  son las sustituciones en  $\{ab\}$  y  $a^*$ , respectivamente. Como consecuencia, el Teorema 6.1 podría ser considerado como un corolario del Teorema 6.2.

Tomando en cuenta que un homomorfismo es un tipo especial de sustitución, establecemos el siguiente corolario.

**Corolario** Los CFLs son cerrados con respecto a los homomorfismos.

**Teorema 6.3** Los lenguajes libres de contexto son cerrados con respecto al homomorfismo inverso.

**Demostración** Al igual que con los conjuntos regulares, una demostración basada en una máquina, para la cerradura con respecto al homomorfismo inverso es más fácil de comprender. Sean  $h: \Sigma \rightarrow \Delta$  un homomorfismo y  $L$  un CFL. Hagamos  $L' = h^{-1}(L)$ , en donde  $M$  es el PDA ( $Q, \Delta, \Gamma, \delta, q_0, Z_0, F$ ). De manera análoga a la construcción del autómata finito del Teorema 3.5, construimos el PDA  $M'$  que acepta a  $h^{-1}(L)$  de la forma siguiente. Sobre la entrada  $a$ ,  $M'$  genera la cadena  $h(a)$  y simula a  $M$  sobre  $h(a)$ . Si  $M'$  fuera un autómata finito, todo lo que podrá hacer sobre la cadena  $h(a)$  sería cambiar el estado, de modo que  $M'$  podría simular dicho movimiento compuesto en uno de sus movimientos. Sin embargo, en el caso del PDA,  $M'$  podría insertar muchos símbolos en una cadena o, puesto que es no determinístico, hacer movimientos que

empujen un número arbitrario de símbolos sobre la pila. Por tanto  $M'$  no necesariamente puede simular movimientos de  $M$  sobre  $h(a)$  con un movimiento propio (o cualquier número finito de ellos).

Lo que hacemos es asignarle a  $M'$  un buffer en el cual se puede almacenar a  $h(a)$ . Entonces  $M'$  puede simular cualesquier movimientos  $\in$  de  $M$  que desee y consumir los símbolos de  $h(a)$  uno por uno, como si fueran una entrada de  $M$ . Como el buffer es parte del control finito de  $M'$ , no puede permitirse que crezca de manera indiscriminada. Nos aseguramos de que no lo haga dejando a  $M'$  que lea un símbolo de entrada solamente cuando el buffer esté vacío. Por consiguiente, el buffer contiene, todas las veces, un sufijo de  $h(a)$  para alguna  $a$ .  $M'$  acepta su entrada  $w$  si el buffer está vacío y  $M$  se encuentra en un estado final. Es decir,  $M$  ha aceptado a  $h(w)$ . En consecuencia,  $L(M') = \{w \mid h(w) \text{ está en } L\}$ , esto es  $L(M') = h^{-1}(L(M))$ . El ordenamiento se describe en la Fig. 6.5; en seguida se presenta la construcción formal.

Sea  $M' = (Q', \Sigma, \Gamma, \delta', [q_0, \epsilon], Z_0, F \times \{\epsilon\})$ , en donde  $Q'$  consiste en pares  $[q, x]$  tales que  $q$  está en  $Q$  y  $x$  es un sufijo (no necesariamente propio) de alguna  $h(a)$  para  $a$  en  $\Sigma$ .  $\delta'$  se define como:

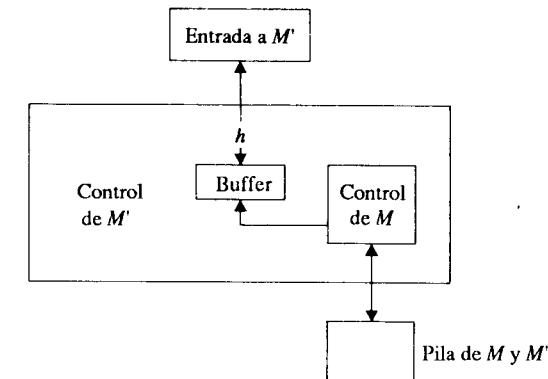


Fig. 6.5 Construcción de un PDA que acepta a  $h^{-1}(L)$ .

1.  $\delta'([q, x], \epsilon, Y)$  contiene a todos los  $([p, x], \gamma)$  tales que  $\delta(q, \epsilon, Y)$  contiene a  $(p, \gamma)$ . Simúlense movimientos  $\in$  de  $M$  independientemente del contenido del buffer.
2.  $\delta'([q, ax], \epsilon, Y)$  contiene a todas las  $([p, x], \gamma)$  tales que  $\delta(q, a, Y)$  contiene a  $(p, \gamma)$ . Simúlense los movimientos de  $M$  sobre la entrada  $a$  en  $\Delta$ , retirando  $a$  del frente del buffer.
3.  $\delta'([q, \epsilon], a, Y)$  contiene a  $([q, h(a)], Y)$  para toda  $a$  en  $\Sigma$  y  $Y$  en  $\Gamma$ . Cárguese el regulador con  $h(a)$ , leyendo  $a$  de la entrada de  $M$ ; el estado y la pila de  $M$  permanecen sin cambios.

Para mostrar que  $L(M') = h^{-1}(L(M))$  primero obsérvese que mediante una aplicación de la regla (3), seguida de aplicaciones de las reglas (1) y (2), si  $(q, h(a), \alpha) \xrightarrow{*} (p, \epsilon, \beta)$ , entonces

$$([q, \epsilon], a, \alpha) \xrightarrow{M'} ([q, h(a)], \epsilon, \alpha) \xrightarrow{M'}^* ([p, \epsilon], \epsilon, \beta).$$

Por consiguiente, si  $M$  acepta a  $h(w)$ , es decir,

$$(q_0, h(w), Z_0) \xrightarrow{M'}^* (p, \epsilon, \beta)$$

para algún  $p$  en  $F$  y  $\beta$  en  $\Gamma^*$ , se concluye que

$$([q_0, \epsilon], w, Z_0) \xrightarrow{M'}^* ([p, \epsilon], \epsilon, \beta),$$

de modo que  $M'$  acepta a  $w$ . Por tanto  $L(M') \supseteq h^{-1}(L(M))$ .

Por el contrario, supóngase que  $M'$  acepta a  $w = a_1 a_2 \dots a_n$ . Entonces, como la regla (3) puede aplicarse solamente con el buffer vacío (segunda componente del estado de  $M'$ ), la sucesión de movimientos de  $M'$  que llevan a la aceptación puede escribirse como

$$\begin{aligned} & ([q_0, \epsilon], a_1 a_2 \dots a_n, Z_0) \xrightarrow{M'}^* ([p_1, \epsilon], a_1 a_2 \dots a_n, \alpha_1), \\ & \xrightarrow{M'} ([p_1, h(a_1)], a_2 a_3 \dots a_n, \alpha_1), \\ & \xrightarrow{M'}^* ([p_2, \epsilon], a_2 a_3 \dots a_n, \alpha_2), \\ & \xrightarrow{M'} ([p_2, h(a_2)], a_3 a_4 \dots a_n, \alpha_2), \\ & \vdots \\ & \xrightarrow{M'}^* ([p_{n-1}, \epsilon], a_n, \alpha_n), \\ & \xrightarrow{M'} ([p_{n-1}, h(a_n)], \epsilon, \alpha_n), \\ & \xrightarrow{M'}^* ([p_n, \epsilon], \epsilon, \alpha_{n+1}), \end{aligned}$$

en donde  $p_n$  está en  $F$ . Las transiciones del estado  $[p_i, \epsilon]$  al estado  $[p_i, h(a_i)]$  se realizan según la regla (3), siendo las otras transiciones mediante las reglas (1) y (2). Por consiguiente,  $(q_0, \epsilon, Z_0) \xrightarrow{M'}^* (p_1, \epsilon, \alpha_1)$  y para toda  $i$ ,

$$(p_i, h(a_i), \alpha_i) \xrightarrow{M'}^* (p_{i+1}, \epsilon, \alpha_{i+1}).$$

Juntando estos movimientos, tenemos

$$(q_0, h(a_1 a_2 \dots a_n), Z_0) \xrightarrow{M'}^* (p_n, \epsilon, \alpha_{n+1}),$$

de modo que  $h(a_1 a_2 \dots a_n)$  está en  $L(M)$ . De aquí que  $L(M') \subseteq h^{-1}(L(M))$ , con lo que concluimos que  $L(M') = h^{-1}(L(M))$ .  $\square$

### Operaciones booleanas

Existen varias propiedades de cerradura de los conjuntos regulares que no poseen los lenguajes libres de contexto. Entre éstas se hace notoria la cerradura con respecto a la intersección y la complementación.

**Teorema 6.4** Los CFLs no son cerrados con respecto a la intersección.

**Demostración** En el Ejemplo 6.1 mostramos que el lenguaje  $L_1 = \{a^i b^j c^i \mid i \geq 1\}$  no es un CFL. Pedimos que  $L_2 = \{a^i b^j c^i \mid i \geq 1 \text{ y } j \geq 1\}$  y  $L_3 = \{a^i b^j c^j \mid i \geq 1 \text{ y } j \geq 1\}$  sean

ambos CFLs. Por ejemplo, para que un PDA reconozca a  $L_2$ , almacena las  $a$ s en su pila y cancela las  $a$ s con las  $b$ s, entonces acepta su entrada después de considerar una o más  $c$ s. De manera alternativa,  $L_2$  es generado por la gramática

$$S \rightarrow AB$$

$$A \rightarrow aAb \mid ab$$

$$B \rightarrow cB \mid c$$

en donde  $A$  genera a  $a^i b^i$  y  $B$  genera a  $c^i$ . Una gramática parecida

$$S \rightarrow CD$$

$$C \rightarrow aC \mid a$$

$$D \rightarrow bDc \mid bc$$

genera a  $L_3$ .

Sin embargo,  $L_2 \cap L_3 = L_1$ . Si los CFLs fueran cerrados con respecto a la intersección,  $L_1$  sería, por tanto, un CFL, lo que contradice el resultado del Ejemplo 6.1.  $\square$

**Corolario** Los CFLs no son cerrados con respecto a la complementación.

**Demostración** Sabemos que los CFLs son cerrados con respecto a la unión. Si fueran cerrados con respecto a la complementación, serían  $L_1 \cap L_2 = L_1 \cup L_2$  cerrados con respecto a la intersección, según la ley de Morgan, contradiciendo así al Teorema 6.4.  $\square$

Aunque la clase de los CFLs no es cerrada con respecto a la intersección, sí lo es con respecto a la intersección con un conjunto regular.

**Teorema 6.5** Si  $L$  es un CFL y  $R$  un conjunto regular, entonces  $L \cap R$  es un CFL.

**Demostración** Sea  $L = L(M)$  para el PDA  $M = (Q_M, \Sigma, \Gamma, \delta_M, q_0, Z_0, F_M)$  y sea  $R = L(A)$  para el DFA  $A = (Q_A, \Sigma, \delta_A, p_0, F_A)$ . Construiremos un PDA  $M'$  mediante “la corrida de  $M$  y  $A$  en paralelo”, como se muestra en la Fig. 6.6.  $M'$  simula los movimientos de  $M$  sobre la entrada  $\epsilon$  sin cambiar el estado de  $A$ . Cuando  $M'$  hace un movimiento sobre el símbolo de entrada  $a$ ,  $M$  simula ese movimiento y también simula el cambio de estado de  $A$  sobre el símbolo de entrada  $a$ .  $M'$  acepta si y sólo si, ambos,  $A$  y  $M$  aceptan. De manera formal, hagamos

$$M' = (Q_A \times Q_M, \Sigma, \Gamma, \delta, [p_0, q_0], Z_0, F_A \times F_M),$$

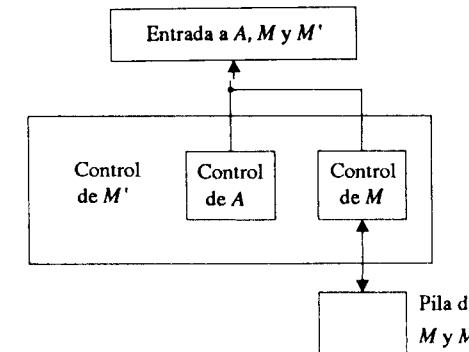


Fig. 6.6 Corrida de un DFA y un PDA en paralelo.

en donde  $\delta$  se define mediante  $\delta([p, q], a, X)$ , contiene a  $([p', q'], \gamma)$  si y sólo si  $\delta_A(p, a) = p'$  y  $\delta_M(q, a, X)$  contiene a  $(q', \gamma)$ . Nótese que  $a$  puede ser  $\epsilon$ , en cuyo caso  $p' = p$ .

Una sencilla inducción sobre  $i$  muestra que

$$([p_0, q_0], w, Z_0) \vdash_M^i ([p, q], \epsilon, \gamma)$$

si y sólo si

$$(q_0, w, Z_0) \vdash_M^i (q, \epsilon, \gamma) \quad \text{y} \quad \delta_A(p_0, w) = p.$$

La base,  $i=0$ , es trivial porque  $p=p_0$ ,  $q=q_0$ ,  $\gamma=Z_0$  y  $w=\epsilon$ . Para la inducción, suponga que es válida la proposición para  $i-1$  y sea

$$([p_0, q_0], xa, Z_0) \vdash_M^{i-1} ([p', q'], a, \beta) \vdash_M^i ([p, q], \epsilon, \gamma),$$

en donde  $w = xa$ , y  $a$  es  $\epsilon$  o un símbolo de  $\Sigma$ . Según la hipótesis inductiva,

$$\delta_A(p_0, x) = p' \quad \text{y} \quad (q_0, x, Z_0) \vdash_M^{i-1} (q', \epsilon, \beta).$$

Mediante la definición de  $\delta$ , el hecho de que  $([p', q'], a, \beta) \vdash_M^i ([p, q], \epsilon, \gamma)$  nos diga que  $\delta_A(p', a) = p$  y  $(q', a, \beta) \vdash_M^i (q, \epsilon, \gamma)$ . Por consiguiente  $\delta_A(p_0, w) = p$  y

$$(q_0, w, Z_0) \vdash_M^i (q, \epsilon, \gamma).$$

La inversa, mostrando que  $(q_0, w, Z_0) \vdash_M^i (q, \epsilon, \gamma)$  y  $\delta_A(p_0, w) = p$  implica que

$$([p_0, q_0], w, Z_0) \vdash_M^i ([p, q], \epsilon, \gamma),$$

es parecida a la anterior y se deja como ejercicio.  $\square$

### Uso de las propiedades de cerradura

Concluimos la presente sección con un ejemplo que ilustrará el uso de las propiedades de cerradura de los lenguajes libres de contexto para demostrar que ciertos lenguajes no son libres de contexto.

**Ejemplo 6.5** Sea  $L = \{ww \mid w \text{ está en } (a+b)^*\}$ . Es decir,  $L$  consiste en todas las palabras cuyas primera y última mitades sean las mismas. Supóngase que  $L$  es libre de contexto. Entonces, según el Teorema 6.5,  $L_1 = L \cap a^*b^*a^*b^*$  también sería un CFL. Pero  $L_1 = \{a^ib^ja^jb^i \mid i \geq 1 \text{ y } j \geq 1\}$ .  $L_1$  es casi igual al lenguaje del Ejemplo 6.2 que se demostró no era libre de contexto, utilizando el lema de sondeo. El mismo argumento muestra que  $L_1$  no es un CFL. Por tanto contradecimos la suposición de que  $L$  es un CFL.

Si no queremos utilizar el lema de sondeo aplicado a  $L_1$ , podemos reducirlo a  $L_2 = \{a^ib^jc^kd^j \mid i \geq 1 \text{ y } j \geq 1\}$ , que es exactamente el mismo lenguaje que se discutió en el Ejemplo 6.2. Sea  $h$  el homomorfismo  $h(a) = h(c) = a$  y  $h(b) = h(d) = b$ . Entonces  $h^{-1}(L_1)$  consiste en todas las palabras de la forma  $x_1x_2x_3x_4$ , en donde  $x_1$  y  $x_3$  tienen la misma longitud y están en  $(a+c)^*$  y  $x_2$  y  $x_4$  son de la misma longitud y se encuentran en  $(b+d)^*$ . Entonces,  $h^{-1}(L_1) \cap a^*b^*c^*d^* = L_2$ . Según los Teoremas 6.3 y 6.5, si  $L_1$  fuera un CFL, también lo sería  $L_2$ . Puesto que se sabe que  $L_2$  no es un CFL, concluimos que  $L_1$  no es un lenguaje libre de contexto.

### 6.3 ALGORITMOS DE DECISION PARA CFLS

Existen muchas preguntas con respecto a los CFLs que podemos responder. Estas incluyen las cuestiones de cuándo un CFL dado está vacío, es finito o infinito y cuando una palabra dada está en un CFL dado. Existen, sin embargo, ciertas preguntas acerca de los CFLs que ningún algoritmo puede responder. Entre éstas podemos incluir las cuestiones de cuándo dos CFGs son equivalentes, cuándo un CFL es cofinito, si el complemento de un CFL dado es también un CFL y si una CFG dada es ambigua. En los dos capítulos siguientes desarrollaremos herramientas para mostrar que no existe un algoritmo para hacer una tarea en particular. En el Capítulo 8 demostraremos, en realidad, que las interrogativas dadas más arriba y otras no tienen algoritmos. En el presente capítulo nos contentaremos con proporcionar algoritmos para algunas de las interrogantes que poseen algoritmos.

Como se tiene con los conjuntos regulares, tenemos varias representaciones para los CFLs, a saber, las gramáticas libres de contexto y los autómatas de apilamiento que aceptan mediante agotamiento de pila o por acceso de un estado final. Como las construcciones que se dieron en el Capítulo 5 son todas efectivas, se puede hacer un algoritmo que utilice una representación para que trabaje por todas las demás. En esta sección utilizaremos la representación CFG.

**Teorema 6.6** Existen algoritmos para determinar si un CFL está (a) vacío (b) es finito o (c) infinito.

**Demostración** El teorema puede demostrarse mediante la misma técnica (Teorema 3.7) que se utilizó para el resultado análogo para conjuntos regulares, haciendo uso del lema de sondeo. Sin embargo, los algoritmos resultantes son altamente insuficientes. En realidad ya hemos dado un mejor algoritmo para probar si un CFL está vacío. Para una CFG  $G = (V, T, P, S)$ , la prueba del Lema 4.1 determina si una variable genera cualquier cadena de terminales. Claramente,  $L(G)$  no está vacío si y sólo si el símbolo inicial  $S$  genera alguna cadena de terminales.

Para probar si  $L(G)$  es finito, utilícese el algoritmo del Teorema 4.5 para encontrar una CFG  $G' = (V, T, P', S)$  en CNF y sin símbolos inútiles, que genera a  $L(G) - \{\epsilon\}$ .  $L(G')$  es finito si y sólo si  $L(G)$  es finito. Una prueba sencilla de la finitud de una gramática CNF sin símbolos inútiles consiste en dibujar una gráfica dirigida con un vértice por cada variable y una arista de  $A$  a  $B$  si existe una producción de la forma  $A \rightarrow BC$  o  $A \rightarrow CB$  para cualquier  $C$ . Entonces el lenguaje generado es finito si y sólo si esta gráfica no tiene ciclos.

Si hubiera un ciclo, digamos  $A_0, A_1, \dots, A_n, A_0$ , entonces

$$A_0 \Rightarrow \alpha_1 A_1 \beta_1 \Rightarrow \alpha_2 A_2 \beta_2 \cdots \Rightarrow \alpha_n A_n \beta_n \Rightarrow \alpha_{n+1} A_0 \beta_{n+1},$$

en donde las  $\alpha$  y  $\beta$  son cadenas de variables, con  $|\alpha_i \beta_i| = i$ . Como no existen símbolos inútiles,  $\alpha_{n+1} \xrightarrow{*} w$  y  $\beta_{n+1} \xrightarrow{*} x$  para algunas cadenas terminales  $w$  y  $x$  que tengan una longitud total de al menos  $n+1$ . Puesto que  $n \geq 0$ ,  $w$  y  $x$  no pueden ambas ser  $\epsilon$ . En seguida, como no hay símbolos inútiles, podemos encontrar cadenas terminales  $y$  y  $z$  tales que  $S \xrightarrow{*} y A_0 z$ , y una cadena terminal  $v$  tal que  $A_0 \xrightarrow{*} v$ . Entonces para toda  $i$

$$S \xrightarrow{*} y A_0 z \xrightarrow{*} yw A_0 xz \xrightarrow{*} yw^2 A_0 x^2 z \xrightarrow{*} \cdots \xrightarrow{*} yw^i A_0 x^i z \xrightarrow{*} yw^i v x^i z.$$

Como  $|wx| > 0$ ,  $yw^i v x^i z$  no puede igualar a  $yw^i v x^j z$  si  $i \neq j$ . Por consiguiente la gramática genera un número infinito de cadenas.

De manera inversa, supóngase que la gráfica no tiene ciclos. Defínase el rango de una variable  $A$  como la longitud de la trayectoria más larga de la gráfica que comienza en  $A$ . La ausencia de ciclos implica que el rango de  $A$  es finito. Observamos, también, que si  $A \rightarrow BC$  es una producción, entonces los rangos de  $B$  y  $C$  deben ser estrictamente menores que el rango de  $A$ , porque para cada trayectoria de  $B$  a  $C$ , existe una trayectoria con longitud mayor en una unidad que parte de  $A$ . Mostramos por inducción en  $r$  que si  $A$  tiene rango  $r$ , entonces ninguna cadena terminal derivada de  $A$  tiene longitud mayor que  $2^r$ .

**Base**  $r = 0$ . Si  $A$  tiene rango 0, entonces su vértice no tiene aristas que salgan. Por tanto, todas las producciones  $A$  tienen terminales en la derecha y  $A$  deriva sólo cadenas de longitud 1.

**Inducción**  $r > 0$ . Si utilizamos una producción de la forma  $A \rightarrow a$ , podemos derivar sólo una cadena de longitud 1. Si comenzamos con  $A \rightarrow BC$ , entonces, como  $B$  y  $C$  son de rango  $r - 1$  o menor, según la hipótesis inductiva, derivan sólo cadenas de longitud  $2^{r-1}$  o menos. Por consiguiente  $BC$  no puede derivar una cadena de longitud mayor a  $2^r$ .

Ya que  $S$  es de rango finito  $r_0$  y, de hecho, es de rango no mayor que el número de variables,  $S$  deriva cadenas de longitud no mayor a  $2^r$ . En consecuencia el lenguaje es finito.  $\square$

#### Ejemplo 6.6 Considérese la gramática

$$S \rightarrow AB$$

$$A \rightarrow BC \mid a$$

$$B \rightarrow CC \mid b$$

$$C \rightarrow a$$

cuyo grafo se muestra en la Fig. 6.7(a). Este grafo no tiene ciclos. Los rangos de  $S$ ,  $A$ ,  $B$  y  $C$  son 3, 2, 1 y 0, respectivamente. Por ejemplo, la trayectoria más larga de  $S$  es  $S \rightarrow A \rightarrow B \rightarrow C \rightarrow C \rightarrow C \rightarrow a$ . Por tanto esta gramática no deriva cadenas de longitud mayor a  $2^3 = 8$  y en consecuencia genera un lenguaje finito. De hecho, la cadena más larga que se genera a partir de  $S$  es

$$S \Rightarrow AB \Rightarrow BCB \Rightarrow CCCB \Rightarrow CCCCC \xrightarrow{*} aaaaa.$$

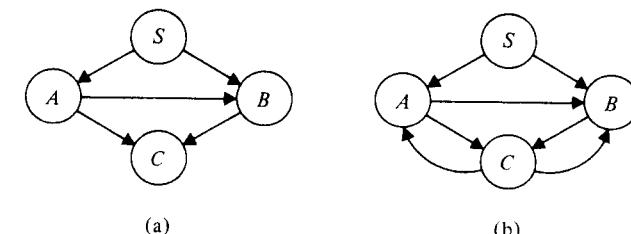


Fig. 6.7 Gráficas correspondientes a gramáticas en CNF.

Si agregamos una producción  $C \rightarrow AB$ , tendremos el grafo de la Fig. 6.7(b). Este nuevo grafo tiene varios ciclos, como  $A, B, C, A$ . Por tanto podemos encontrar una derivación  $A \xrightarrow{*} \alpha_1 A \beta_1$ , en particular  $A \Rightarrow BC \Rightarrow CCC \Rightarrow CABC$ , en donde  $\alpha_1 = C$  y  $\beta_1 = BC$ . Ya que  $C \xrightarrow{*} a$  y  $BC \xrightarrow{*} ba$ , tenemos  $A \xrightarrow{*} aAba$ . Entonces, como  $S \xrightarrow{*} Ab$  y  $A \xrightarrow{*} a$ , tenemos ahora  $S \xrightarrow{*} a'a (ba)^i b$  para cada  $i$ . Por lo tanto el lenguaje es finito.

#### Membresía

Otra cuestión que podemos responder es: dada una CFG  $G = (V, T, P, S)$  y la cadena  $x$  es  $T^*$ , ¿está  $x$  en  $L(G)$ ? Un algoritmo simple pero ineficiente para responder lo anterior consiste en convertir  $G$  a  $G' = (V, T, P', S)$ , una gramática en la forma normal de Greibach que genera a  $L(G) - \{\epsilon\}$ . Puesto que el algoritmo del Teorema 4.3 prueba si  $S \xrightarrow{*} \epsilon$ , no necesitamos interesarlos en el caso en que  $x = \epsilon$ . Por tanto suponemos  $x \neq \epsilon$ , de modo que  $x$  está en  $L(G')$  si y sólo si  $x$  está en  $L(G)$ . Ahora, como cada producción de una gramática GNF agrega exactamente una terminal a la cadena que está siendo generada, sabemos que si  $x$  tiene una derivación en  $G'$ , tiene una con exactamente  $|x|$  pasos. Si ninguna variable de  $G'$  tiene más de  $k$  producciones, entonces existen cuando mucho  $k^{|x|}$  derivaciones izquierdas de cadenas de longitud  $|x|$ . Las trataremos a todas de manera sistemática.

Sin embargo, el algoritmo anterior puede tomar un tiempo que es exponencial con respecto a  $|x|$ . Existen varios algoritmos conocidos que toman un tiempo proporcional al cubo de  $|x|$  y aun menos. Las notas bibliográficas discuten algunos de ellos. Presentaremos aquí un algoritmo simple de tiempo cúbico conocido como algoritmo de Cocke-Younger-Kasami o algoritmo CYK. Está basado en la técnica de programación dinámica discutida en la solución del Ejercicio 3.23. Dadas  $x$  de longitud  $n \geq 1$  y una gramática  $G$ , que supondremos en la forma normal de Chomsky, determinese, para  $i$  y  $j$  y para cada variable  $A$ , si  $A \xrightarrow{*} x_{ij}$ , en donde  $x_{ij}$  es la subcadena  $x$  de longitud  $j$  que comienza en la posición  $i$ .

Procederemos por inducción sobre  $j$ . Para  $j = 1$ ,  $A \xrightarrow{*} x_{ij}$  si y sólo si  $A \rightarrow x_{ij}$  es una producción, ya que  $x_{ij}$  es una cadena de longitud 1. Pasamos ahora a valores más altos de  $j$ , si  $j > 1$ , entonces  $A \xrightarrow{*} x_{ij}$  si y sólo si existe alguna producción  $A \rightarrow BC$  y alguna  $k$ ,  $1 \leq k \leq j$ , tal que  $B$  derive los primeros  $k$  símbolos de  $x_{ij}$  y  $C$  derive los últimos  $j-k$  símbolos de  $x_{ij}$ . Es decir,  $B \xrightarrow{*} x_{ik}$  y  $C \xrightarrow{*} x_{i+k, j-k}$ . Como  $k$  y  $j-k$  son ambos menores que

$j$ , ya sabemos si cada una de las dos últimas derivaciones existe. Podemos, por tanto, determinar si  $A \xrightarrow{*} x_{ij}$ . Finalmente, cuando se alcanza  $j = n$ , debemos determinar si  $S \xrightarrow{*} x_{in}$ . Pero  $x_{in} = x$ , así que  $x$  está en  $L(G)$  si y sólo si  $S \xrightarrow{*} x_{in}$ .

Para plantear de manera precisa el algoritmo CYK, hagamos  $V_{ij}$  el conjunto de variables  $A$  tales que  $A \xrightarrow{*} x_{ij}$ . Nótese que podemos suponer que  $1 \leq i \leq n - j + 1$ , porque no existe una cadena de longitud mayor que  $n - i + 1$  que comience en la posición  $i$ . La Fig. 6.8 da el algoritmo CYK de manera formal.

Los pasos (1) y (2) manejan el caso  $j = 1$ . Como la gramática  $G$  permanece fija, el paso (2) toma una cantidad constante de tiempo. Por consiguiente los pasos (1) y (2) toman un tiempo  $O(n)$ . Los ciclos *for* anidados de las líneas (3) y (4) ocasionan que se ejecuten los pasos (5) a (7) cuando mucho  $n^2$  veces, ya que  $i$  y  $j$  varían en sus respectivos ciclos *for* entre límites que se encuentran separados cuando mucho una cantidad  $n$ . El paso (5) se lleva una cantidad constante de tiempo en cada ejecución, de modo que el

```

begin
1.   for  $i := 1$  to  $n$  do
2.      $V_{i1} := \{A \mid A \rightarrow a$  es una producción y el  $i$ -ésimo símbolo de  $x$  es  $a\}$ ;
3.   for  $j := 2$  to  $n$  do
4.     for  $i := 1$  to  $n - j + 1$  do
         begin
5.        $V_{ij} := \emptyset$ ;
6.       for  $k := 1$  to  $j - 1$  do
7.          $V_{ij} := V_{ij} \cup \{A \mid A \rightarrow BC$  is a production,  $B$  is in  $V_{ik}$  and  $C$ 
           está en  $V_{i+k,j-k}\}$ 
         end
      end
end

```

Fig. 6.8 Algoritmo CYK.

tiempo total hasta el paso (5) es  $O(n^2)$ . El ciclo *for* de la línea (6) ocasiona que el paso (7) sea ejecutado  $n$  o menos tiempo. Puesto que el paso (7) toma un tiempo constante, los pasos (6) y (7) juntos se llevan un tiempo  $O(n)$ . Como se llevan a cabo  $0(n^2)$  veces, el tiempo total hasta el paso (7) es  $O(n^3)$ . Por consiguiente el algoritmo completo es  $O(n^3)$ .

Ejemplo 6.7 Considérese la CFG

$$\begin{aligned} S &\rightarrow AB \mid BC \\ A &\rightarrow BA \mid a \\ B &\rightarrow CC \mid b \\ C &\rightarrow AB \mid a \end{aligned}$$

y la cadena de entrada  $baaba$ . en la Fig. 6.9 se muestra la tabla de las  $V_{ij}$ . El renglón de arriba se llena mediante los pasos (1) y (2) del algoritmo de la Fig. 6.8. Esto es, para las posiciones 1 y 4, que son  $b$ , hacemos  $V_{11} = V_{41} = \{B\}$ , ya que  $B$  es la única variable que deriva a  $b$ . De manera similar,  $V_{21} = V_{31} = V_{51} = \{A, C\}$ , ya que sólo  $A$  y  $C$  tienen producciones con  $a$  en la derecha.

	$b$	$a$	$a$	$b$	$a$
	$i \rightarrow$				
	1	2	3	4	5
1	$B$	$A, C$	$A, C$	$B$	$A, C$
2	$S, A$	$B$	$S, C$	$S, A$	
3	$\emptyset$	$B$	$B$		
4	$\emptyset$	$S, A, C$			
5	$S, A, C$				

Fig. 6.9 Tabla de las  $V_{ij}$ .

Para calcular  $V_{ij}$  para  $j > 1$ , debemos ejecutar el *ciclo for* de los pasos (6) y (7). Debemos balancear  $V_{ik}$  contra  $V_{i+k,j-k}$  para  $k = 1, 2, \dots, j - 1$ , buscando variable  $D$  que estén en  $V_{ik}$  y  $E$  que estén en  $V_{i+k,j-k}$  tales que  $DE$  sea el lado derecho de una o más producciones. El lado izquierdo de estas producciones se unen a  $V_{ij}$ . El patrón en la tabla que corresponde a visitar, una a la vez,  $V_{ik}$  y  $V_{i+k,j-k}$  para  $k = 1, 2, \dots, j - 1$ , consiste en moverse simultáneamente hacia abajo de la columna  $i$  hacia arriba por la diagonal que se extiende desde  $V_{ij}$  hacia la derecha, como se muestra en la Fig. 6.10.

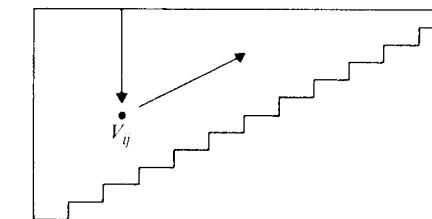


Fig. 6.10 Patrón transversal para el cálculo de  $V_{ij}$ .

Por ejemplo, calculemos  $V_{24}$ , suponiendo que las tres filas de tope en la Fig. 6.9, se encuentran llenas. Comenzamos por fijarnos en  $V_{21} = \{A, C\}$  y  $V_{31} = \{B\}$ . Los posibles lados derechos de  $V_{21}V_{33}$  son  $AB$  y  $CB$ . Sólo el primero de estos es en realidad un lado derecho, y lo es de dos producciones:  $S \rightarrow AB$  y  $C \rightarrow AB$ . Por tanto agregamos  $S$  y  $C$  a  $V_{24}$ . En seguida consideramos  $V_{22}V_{42} = \{B\}$  ( $S, A\} = \{BS, BA\}$ ). Solamente  $BA$  es un lado derecho, así que añadimos el correspondiente lado izquierdo  $A$  a  $V_{24}$ . Finalmente, consideramos a  $V_{23}V_{51} = \{B\}$  ( $A, C\} = \{BA, BC\}$ ).  $BA$  y  $BC$  son lados derechos, con lados izquierdos  $A$  y  $S$ , respectivamente. Estos ya se encuentran en  $V_{24}$ , así que tenemos  $V_{24}$

$= \{S, A, C\}$ . Como  $S$  es miembro de  $V_{15}$ , la cadena  $baaba$  está en el lenguaje generado por la gramática.

## EJERCICIOS

**6.1** Muestre que los siguientes no son lenguajes libres de contexto.

- $\{a^i b^j c^k \mid i < j < k\}$
- $\{a^i b^j \mid j = i^2\}$
- $\{a^i \mid i \text{ es primo}\}$
- el conjunto de las cadenas de  $as$ ,  $bs$  y  $cs$  con un número igual de cada uno
- $\{a^n b^n c^m \mid n \leq m \leq 2n\}$

**6.2** ¿Cuáles de los siguientes son CFLs?

- $\{a^i b^j \mid i \neq j \text{ e } i \neq 2j\}$
- $(a + b)^* - \{(a^n b^n)^* \mid n \geq 1\}$
- $\{ww^R w \mid w \text{ está en } (a + b)^*\}$
- $\{b_i \# b_{i+1} \mid b_i \text{ es el número } i \text{ en binario, } i \geq 1\}$
- $\{wxw \mid w \text{ y } x \text{ están en } (a + b)^*\}$
- $(a + b)^* - \{(a^n b^n)^* \mid n \geq 1\}$

**6.3** Demuestre que los siguientes no son CFLs.

- $\{a^i b^j a^k \mid j = \max\{i, k\}\}$
- $\{a^n b^n c^n \mid i \neq n\}$

[*Sugerencia:* Utilice el lema de Ogden sobre una cadena de la forma  $a^n b^n c^n$ ]

**6.4** Muestre que los CFLs son cerrados bajo las operaciones siguientes:

\*a) El cociente con un conjunto regular, esto es, si  $L$  es un CFL y  $R$  un conjunto regular, entonces  $L/R$  es un CFL.

b) INIT

s\*\* c) CICLO

d) inverso

Véase el Ejercicio 3.4 para las definiciones de INIT y CICLO.

**6.5** Muestre que los CFLs no son cerrados con respecto a las siguientes operaciones:

- MÍN
- MÁX
- $\frac{1}{2}$
- Sustitución inversa
- INV en donde  $INV(L) = \{x \mid x = wyz \text{ y } wy^Rz \text{ está en } L\}$

Las operaciones MÍN, MÁX y  $\frac{1}{2}$  están definidas en los Ejercicios 3.4 y 3.16.

**6.6** Sea  $\Sigma$  un alfabeto. Defínanse los homomorfismos  $h$ ,  $h_1$  y  $h_2$  mediante  $h(a) = h(\bar{a}) = a$ ,  $h_1(a) = a$ ,  $h_1(\bar{a}) = \epsilon$ ,  $h_2(a) = \epsilon$  y  $h_2(\bar{a}) = a$  para cada  $a$  en  $\Sigma^*$ . Para  $L_1 \subseteq \Sigma^*$  y  $L_2 \subseteq \Sigma^*$ , defínase

$$\text{Mezcla}(L_1, L_2) = \{x \mid \text{para alguna } y \text{ en } h^{-1}(x), h_1(y) \text{ está en } L_1 \text{ y } h_2(y) \text{ está en } L_2\}.$$

Esto es, la Mezcla de  $L_1$  y  $L_2$  es el conjunto de palabras formadas mediante la "mezcla" de una palabra de  $L_1$  con una palabra de  $L_2$ . Los símbolos de las dos palabras no necesitan estar alternadas como en una "mezcla perfecta".

a) Muestre que la Mezcla de dos conjuntos regulares es regular.

b) Demuestre que la Mezcla de dos CFLs no es necesariamente un CFL.

c) Demuestre que la Mezcla de un CFL y un conjunto regular es un CFL.

**\*6.7** Un *lenguaje de Dick* es un lenguaje que posee  $k$  tipos de paréntesis balanceados. De manera formal, cada lenguaje de Dick es, para alguna  $k$ ,  $L(G_k)$ , en donde  $G_k$  es la gramática

$$S \rightarrow SS \mid [_1 S]_1 \mid [_2 S]_2 \mid \cdots \mid [_k S]_k \mid \epsilon.$$

Por ejemplo,  $[1][2[1]1][2]2]2]1$  está en el lenguaje de Dick que tiene dos clases de paréntesis. Demuestre que cada CFL  $L$  es  $h(L_b \cap R)$ , en donde  $h$  es un homomorfismo,  $R$  un conjunto regular y  $LD$  un lenguaje de Dick. [*Sugerencia:* Haga que  $L$  sea aceptado mediante agotamiento de pila por un PDA en la forma normal del Ejercicio 5.5(b) en el que los movimientos sólo agregan o suprimen símbolos individuales. Que los paréntesis sean  $[_{abX}]_{abX}$ , en donde  $[_{abX}]$  "significa" que sobre la entrada  $b$ ,  $X$  puede ser suprimido ( $a$  o  $b$  pueden ser  $\epsilon$ ). Entonces el lenguaje de Dick impone la condición de que la pila sea manejada de manera consistente, es decir, si  $X$  se agrega entonces aún será  $X$  cuando sea suprimido. Haga que el conjunto regular  $R$  imponga la condición de que exista una sucesión de estados para los cuales los movimientos de agregar y suprimir son legales para entradas  $a$  y  $b$ , respectivamente. Sea  $h([_{abX}]) = a$  y  $h([_{abX}]) = b$ .]

**\*6.8** Muestre que si  $L$  es un CFL sobre un alfabeto de un símbolo, entonces  $L$  es regular. [*Sugerencia:* sea  $n$  la constante del lema de sondeo para  $L$  y  $L \subseteq 0^*$ . Muestre que para cada palabra de longitud  $n$  o más, digamos  $0^m$ , existen  $p$  y  $q$  no mayores que  $n$  tales que  $0^{p+q}$  está en  $L$  para toda  $i \geq 0$ . Entonces muestre que  $L$  consiste, tal vez, en algunas palabras de longitud menor que  $n$  más un número finito de conjuntos lineales, es decir, conjuntos de la forma  $\{0^{p+q} \mid i \geq 0\}$  para  $p$  y  $q$  fijos,  $q \leq n$ .]

**\*\*6.9** Demuestre que el conjunto de números primos en binario no es un CFL.

**6.10** Muestre que los lenguajes lineales (para la definición véase el Ejercicio 4.20) son cerrados con respecto a
 

- la unión
- homomorfismo
- intersección con un conjunto regular.

**6.11** Demuestre el siguiente lema de sondeo para lenguajes lineales. Si  $L$  es un lenguaje lineal, entonces existe una  $L$  constante tal que si  $z$  en  $L$  tiene longitud  $n$  o mayor, entonces podemos escribir  $z = uvwxy$ , tal que  $|uvxy| \leq n$ ,  $|x| \geq 1$  y para toda  $i \geq 0$   $uv^iwx^i y$  está en  $L$ .

**6.12** Muestre que  $\{a^i b^i c^j d^j \mid i \geq 1 \text{ y } j \geq 1\}$  no es un lenguaje lineal.

**\*6.13** Se dice que un PDA hace una vuelta si accesa una secuencia de IDs

$$(q_1, w_1, \gamma_1) \xrightarrow{} (q_2, w_2, \gamma_2) \xrightarrow{} (q_3, w_3, \gamma_3)$$

$y|\gamma_1$  es estrictamente mayor que  $|\gamma_2|$  y  $|\gamma_3|$ . Esto es, se da una vuelta cuando la longitud de la pila "alcanza un máximo". Se dice que un PDA  $M$  es un PDA de  $k$  vueltas si para cada palabra  $w$  en  $L(M)$ ,  $w$  es aceptada mediante una secuencia de IDs dando no más de  $k$  vueltas. Si un PDA es de  $k$  vueltas para alguna  $k$  finita, se dice que es de vueltas finitas. Si  $L$  es aceptado por un PDA de vueltas finitas,  $L$  es metalineal.

- Muestre que un lenguaje es lineal si y sólo si es aceptado por una PDA de una vuelta.
- Muestre que los lenguajes lineales son cerrados con respecto al homomorfismo inverso.
- Muestre que los lenguajes metalineales son cerrados con respecto a la unión, concatenación, homomorfismo, homomorfismo inverso e intersección con un conjunto regular.

**\*\*6.14** Muestre que el conjunto de las cadenas que tienen un número igual de  $as$  y  $bs$  es un CFL que no es un lenguaje metalineal.

**6.15** Muestre que

- los lenguajes lineales
- los lenguajes metalineales no son cerrados con respecto a\* .

**6.16** Dé un algoritmo para decidirse por alguna de las dos formas oracionales  $\alpha$  y  $\beta$  de una CFG  $G$ , cuando  $\alpha \xrightarrow{*} \beta$ .

**6.17** Utilice el algoritmo CYK para determinar si

- a)  $aaaaaa$       b)  $aaaaaaa$

están en la gramática del Ejemplo 6.7.

**6.18** Sea  $G$  una gramática libre de contexto en CNF.

- Dé un algoritmo para determinar el número de derivaciones distintas de una cadena  $x$ .
- Asocie un costo con cada producción de  $G$ . Dé un algoritmo para producir un análisis gramatical de una cadena  $x$  de costo mínimo. El costo de un análisis es la suma de los costos de las producciones usadas.

[Sugerencia: Modifique el algoritmo CYK que se dio en la Sección 6.3.]

#### Soluciones a los ejercicios seleccionados

**6.4 c)** Sea  $G = (V, T, A, S)$  una CFG en CNF. Para construir  $G$  tal que  $L(G) = \text{CICLO}(L(G))$  considérese un árbol de derivación de una cadena  $x_1x_2$  en la gramática  $G$ . Sígase la trayectoria de  $S$  hasta el símbolo que está más a la izquierda de  $x_2$ . Queremos generar la trayectoria en orden inverso (del fondo a la cima) y símbolos de salida en los lados opuestos de la trayectoria de la cual aparecieron originalmente. Para hacer esto, constrúyase

$$\hat{G} = (V \cup \{\hat{A} \mid A \text{ está en } V\} \cup \{S_0\}, T, \hat{P}, S_0),$$

en donde  $P$  contiene a

1. Todas las producciones de  $P$ ,
2.  $\hat{C} \rightarrow AB$  y  $B \rightarrow CA$  si  $P$  contiene a  $A \rightarrow BC$ ,
3.  $\hat{S} \rightarrow \epsilon$
4.  $S_0 \rightarrow aA$  si  $P$  contiene a  $A \rightarrow a$ ,
5.  $S_0 \rightarrow S$ .

Para ver que  $L(\hat{G}) = \text{CICLO}(L(G))$  muestre por inducción sobre la longitud de una derivación que  $A \xrightarrow{*} A_1A_2 \dots A_n$  si y sólo si para cada  $i$

$$\hat{A}_i \xrightarrow{*} A_{i+1} \dots A_n \hat{A} A_1 \dots A_{i-1}.$$

Entonces

$$S \xrightarrow{*} A_1 \dots A_n \Rightarrow A_1 \dots A_{i-1} a A_{i+1} \dots A_n$$

si

$$\begin{aligned} S_0 &\Rightarrow a \hat{A}_i \xrightarrow{*} a A_{i+1} \dots A_n \hat{S} A_1 \dots A_{i-1} \\ &\Rightarrow a A_{i+1} \dots A_n A_1 \dots A_{i-1}. \end{aligned}$$

En la Fig. 6.11(a) se muestra un árbol de derivación con el árbol correspondiente para  $G$  en la Fig. 6.11(b).

**6.5 a)** Sea  $L$  el CFL  $\{0^i1^j2^k \mid i \leq k \text{ o } j \leq k\}$ .  $L$  es generado por la CFG

$$S \rightarrow AB \mid C, \quad A \rightarrow 0A \mid \epsilon, \quad B \rightarrow 1B2 \mid B2 \mid \epsilon, \quad C \rightarrow 0C2 \mid C2D, \quad D \rightarrow 1D \mid \epsilon$$

$\text{MIN}(L) = \{0^i1^j2^k \mid k = \min(i, j)\}$ . Aseguramos que  $\text{MIN}(L)$  no es un CFL. Supóngase que sí lo es y sea  $n$  la constante del lema de sondeo. Considerese  $z = 0^n1^n2^n = uvwxy$ . Si  $vx$  no contiene 2, entonces  $uwv$  no está en  $\text{MIN}(L)$ . Si  $vx$  tiene un 2, no puede tener un 0, ya que  $|vw| \leq n$ . Por tanto  $uv^2wx^2y$  tiene al menos  $n$  1s y exactamente  $n+1$  2s, al menos  $n$  6; por consiguiente no está en  $\text{MIN}(L)$ .

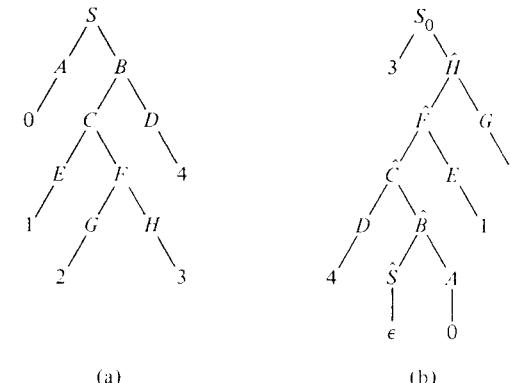


Fig. 6.11 Transformación de árbol utilizada para la solución del Ejercicio 6.4(c).

#### NOTAS BIBLIOGRAFICAS

El lema de sondeo para lenguajes libres de contexto se obtuvo de Bar-Hillel, Perles y Shamir [1961]; el lema de Ogden, en su versión más fuerte, se encuentra en Ogden [1968]. Wise [1976] da una condición necesaria y suficiente para que un lenguaje sea libre de contexto. Parikh [1966] da condiciones necesarias en términos de la distribución de símbolos en las palabras del lenguaje. Los lemas de sondeo para otros tipos de lenguajes están dados en Boasson [1973] y Ogden [1969].

El Teorema 6.2, cerradura con respecto a la sustitución, y el Teorema 6.5, cerradura con respecto a la intersección con un conjunto regular, se tomaron de Bar-Hillel, Perles y Shamir [1961]. El Teorema 6.3 es de Ginsburg y Rose [1963b]. El Teorema 6.4 y su corolario, la no cerradura con respecto a la intersección o complementación, se tomaron de Scheinberg [1960]. El Teorema 6.6, la existencia de un algoritmo para saber si un CFL es finito, también es de Bar-Hillel, Perles y Shamir [1961]. Floyd [1962b] muestra cómo aplicar las propiedades de cerradura para probar lenguajes que fueron construidos para no ser libres de contexto.

El algoritmo CYK fue descubierto originalmente por J. Cocke, pero su primera publicación fue hecha de manera independiente por Kasami [1965] y Younger [1967]. El algoritmo para el reconocimiento de los lenguajes libres de contexto y de análisis gramatical, más práctico y general, se lo debemos a Earley [1970]. Este algoritmo es  $O(n^3)$  en general, pero toma solamente  $O(n^2)$  sobre cualquier CFG no ambigua y es en realidad lineal sobre una amplia variedad de gramáticas útiles. El algoritmo de Valiant [1975a] es asintóticamente el más eficiente, toma  $O(n^2)$ .

pasos, mientras que el algoritmo de Grahas, Harrison y Ruzzo [1976] toma  $O(n^3/\log n)$  pasos. Un resultado relacionado, la membresía para CFGs no ambiguas puede probarse en un tiempo  $O(n^2)$ , se debe a Kasami y Torii [1969] y Early [1970].

El Ejercicio 6.4(a), cerradura de los CFLs con respecto al cociente con un conjunto regular, fue mostrado por Ginsburg y Spanier [1963]. Propiedades de cerradura adicionales de los CFLs pueden encontrarse en Ginsburg y Rose [1963b, 1966]. El ejercicio 6.7, la caracterización de los CFLs mediante lenguajes de Dick, se tomó de Chomsky [1962]. Stanley [1965] probó el resultado más fuerte que consiste en que el lenguaje de Dick utilizado sólo necesita depender del tamaño del alfabeto terminal. La demostración de que los números primos en binario no constituyen un CFL (Ejercicio 6.9) se tomó de Hartmanis y Shank [1968]. Los PDAs de vueltas finitas, que se mencionaron en el Ejercicio 6.13, fueron estudiados por Ginsburg y Spanier [1966]. El Ejercicio 6.8, que se refiere a que los CFL sobre un alfabeto de un símbolo son regulares, fue demostrado por Ginsburg y Rice 1962.

## CAPITULO

## 7

## MAQUINAS DE TURING

En este capítulo introduciremos la máquina de Turing, un modelo matemático simple de una computadora. Independientemente de su simplicidad, la máquina de Turing modela la capacidad de cálculo de una computadora de propósitos generales. La máquina de Turing se estudia debido tanto a la clase de los lenguajes que define (llamados conjuntos de manera recursiva enumerables) como por la clase de funciones enteras que calcula (llamadas funciones parcialmente recursivas). Se introducirán también una variedad de otros modelos de cálculo y se mostrará que son equivalentes a la máquina de Turing en cuanto a la capacidad de cálculo.

## 7.1 INTRODUCCION

El concepto intuitivo de un algoritmo o procedimiento efectivo ha surgido varias veces. En el Capítulo 3 presentamos un procedimiento efectivo para determinar si el conjunto aceptado por un autómata finito estaba vacío, era finito o infinito. Ingenuamente, se podría suponer que para cualquier clase de lenguajes con descripciones finitas, existe un procedimiento efectivo para responder ese tipo de interrogantes. Sin embargo, éste no es el caso. Por ejemplo, no existe un algoritmo para decir si el complemento de un CFL está vacío o no (aunque podamos decir si el CFL mismo está vacío). Nótese que no estamos pidiendo un procedimiento que responda la pregunta para un lenguaje libre de contexto específico, sino, más bien, un procedimiento que responda correctamente la interrogante para todos los CFLs. Está claro que si sólo necesitamos determinar si un CFL específico tiene un complemento vacío, entonces existe un algoritmo que puede respondernos la pregunta. Es decir, existe un algoritmo que dice "sí" y otro que dice

"no", independientemente de sus entradas. Uno de ellos debe ser el correcto. Por supuesto, puede no ser obvio cuál de los dos algoritmos responde a la pregunta correctamente.

A finales del siglo pasado, el matemático David Hilbert elaboró un programa para encontrar un algoritmo para la determinación de la verdad o falsedad de cualquier proposición matemática. En particular, estaba buscando un procedimiento para determinar si una fórmula cualquiera del cálculo de predicados de primer orden, aplicada a los enteros, era verdadera. Como el cálculo de predicados de primer orden es lo suficientemente poderoso para expresar la afirmación de que el lenguaje generado por una gramática libre de contexto es  $\Sigma^*$ , Hilbert tuvo éxito en su empeño y nuestro problema sobre la decisión de si el complemento de un CFL está vacío podría estar resuelto. Sin embargo, en 1931, Kurt Gödel publicó su famoso teorema de la no integridad, que demostró que no puede existir tal procedimiento efectivo. Construyó una fórmula del cálculo de predicados aplicada a los enteros, cuya mera definición establecía que ésta no podría ser aprobada ni desaprobada dentro de este sistema lógico. La formalización de dicho argumento y las consiguientes clarificación y formalización de nuestro concepto intuitivo acerca de un procedimiento efectivo constituye uno de los logros intelectuales más grandes del presente siglo.

Una vez que se hubo formalizado el concepto de procedimiento efectivo, se mostró que no existía un procedimiento efectivo para calcular muchas funciones específicas. En realidad, la existencia de tales funciones se puede ver fácilmente a partir del argumento de conteo. Considérese la clase de las funciones que transforman a los enteros no negativos en  $\{0, 1\}$ . Estas funciones pueden ponerse en correspondencia uno a uno con los reales. Sin embargo, si suponemos que los procedimientos efectivos tienen descripciones finitas, entonces la clase de todos los procedimientos efectivos puede ponerse en correspondencia uno a uno con los enteros. Como no existe una correspondencia uno a uno de los enteros con los reales, deben existir funciones que no tengan su procedimiento efectivo para calcularlas. Simplemente existen demasiadas funciones, un número no contable de ellas, y sólo un número contable de procedimientos. Por consiguiente la existencia de funciones no calculables no es sorprendente. Lo que sí es sorprendente es que algunos problemas y funciones con significancia real en matemáticas, ciencias de la computación y otras disciplinas son no calculables.

Hoy en día, la máquina de Turing se ha convertido en la formalización aceptada de un procedimiento efectivo. Es claro que no se puede demostrar que el modelo de la máquina de Turing es equivalente a nuestro concepto intuitivo de computadora, pero existen fuertes argumentos en favor de esta equivalencia, que se conoce como la hipótesis de Church. En particular, la máquina de Turing es equivalente en poder de cálculo a la computadora digital como la conocemos en la actualidad y también a los conceptos matemáticos más generales de computación.

## 7.2 MODELO DE LA MAQUINA DE TURING

Un modelo formal para un procedimiento efectivo deberá poseer ciertas propiedades. Primero, cada procedimiento deberá poder describir de manera finita. Segundo, el

procedimiento deberá consistir en pasos discretos, cada uno de los cuales puede llevarse a cabo de manera mecánica. Este modelo fue introducido por Alan Turing en 1936. Aquí presentamos una variante del mismo.

El modelo básico, ilustrado en la Fig. 7.1, tiene un control finito, una cinta de entrada que está dividida en celdas y una cabeza de cinta que barre una celda de la cinta a la vez. La cinta tiene una celda que está más a la izquierda, pero se extiende de manera infinita hacia la derecha. Cada celda de la cinta puede contener exactamente un símbolo de un número infinito de símbolos de cinta. Inicialmente, las  $n$  celdas que están más a la izquierda, para alguna  $n \geq 0$  finita, sujetan la entrada, que es una cadena de símbolos escogidos de un subconjunto de los símbolos de la cinta, llamados símbolos de entrada. Cada una del número infinito de celdas restantes sujetan el espacio en blanco, que es un tipo especial de símbolo que no es de entrada.

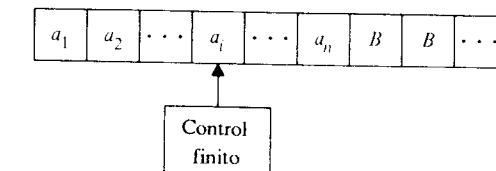


Fig. 7.1 Máquina de Turing básica.

En un movimiento, dependiendo del símbolo barrido por la cabeza de la cinta y del estado del control finito, la máquina de Turing

1. cambia de estado,
2. imprime un símbolo en la celda de la cinta que está siendo barrida, sustituyendo lo que se encontraba ahí escrito y
3. mueve su cabeza una celda hacia la izquierda o la derecha.

Nótese que la diferencia entre una máquina de Turing y un autómata finito de dos direcciones estriba en la capacidad de la primera para cambiar símbolos en su cinta.

De manera formal, una máquina de Turing (TM) se representa por

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

en donde,

$Q$  es el conjunto finito de *estados*,

$\Gamma$  es el conjunto finito de *símbolos de cinta* admisibles,

$B$ , símbolo de  $\Gamma$ , es el *espacio en blanco*,

$\Sigma$ , subconjunto de  $\Gamma$  que no incluye a  $B$ , es el conjunto de los *símbolos de entrada*,

$\delta$  es la *función de movimientos siguientes*, una transformación de  $Q \times \Gamma$  a  $Q \times \Gamma \times \{L, R\}$  ( $\delta$  puede, sin embargo, permanecer indefinida para algunos argumentos),

$q_0$  en  $Q$  es el *estado inicial*,

$F \subseteq Q$  es el conjunto de *estados finales*.

Denotamos a una *descripción instantánea* (ID) de la máquina de Turing,  $M$ , mediante  $\alpha_1 q \alpha_2$ . Aquí,  $q$ , el estado actual de  $M$ , está en  $Q$ ;  $\alpha_1 \alpha_2$  es la cadena de  $\Gamma^*$  que es el contenido de la cinta hasta el símbolo que está más a la derecha que no sea el espacio en blanco o el símbolo que se encuentra a la izquierda de la cabeza, el que esté más a la derecha. (Obsérvese que el espacio en blanco  $B$  puede presentarse en  $\alpha_1 \alpha_2$ .) Para evitar confusiones, suponemos que  $Q$  y  $\Gamma$  son disjuntos. Finalmente, la cabeza de la cinta se supone que barre el símbolo que está más a la izquierda de  $\alpha_2$ , o si  $\alpha_2 = \epsilon$ , la cabeza se encuentra barriendo un espacio en blanco.

Definimos un *movimiento* de  $M$  de la forma siguiente. Sea  $X_1 X_2 \cdots X_{i-1} q X_i \cdots X_n$  una ID. Supóngase que  $\delta(q, X_i) = (p, Y, L)$ , en donde, si  $i - 1 = n$ , entonces  $X_i$  se toma como  $B$ . Si  $i = 1$ , entonces no existe una ID siguiente, pues la cabeza de cinta no puede salir del extremo izquierdo de la cinta. Si  $i > 1$ , entonces escribimos

$$X_1 X_2 \cdots X_{i-1} q X_i \cdots X_n \xrightarrow{M} X_1 X_2 \cdots X_{i-2} p X_{i-1} Y X_{i+1} \cdots X_n. \quad (7.1)$$

Sin embargo, si cualquier sufijo de  $X_{i-1} Y X_{i+1} \cdots X_n$  está completamente en blanco, se elimina de (7.1).

De forma alternativa supóngase que  $\delta(q, X_i) = (p, Y, R)$ , entonces escribimos

$$X_1 X_2 \cdots X_{i-1} q X_i X_{i+1} \cdots X_n \xrightarrow{M} X_1 X_2 \cdots X_{i-1} Y p X_{i+1} \cdots X_n. \quad (7.2)$$

Nótese que en el caso  $i - 1 = n$ , la cadena  $X_1 \dots X_n$  está vacía y el lado derecho de (7.2) es más largo que el izquierdo.

Si dos IDs están relacionadas mediante  $\xrightarrow{M}$ , decimos que la segunda es resultado de la primera mediante un movimiento. Si una ID es resultado de otra mediante algún número finito de movimientos, incluidos los movimientos cero, están relacionadas por el símbolo  $\xrightarrow{*}$ . Cuando no haya lugar a confusión, eliminaremos el subíndice  $M$  de  $\xrightarrow{M}$  o  $\xrightarrow{*}$ .

El *lenguaje aceptado* por  $M$ , representado por  $L(M)$ , es el conjunto de aquellas palabras que se encuentran en  $\Sigma^*$  y que ocasionan que  $M$  accese un estado final cuando se coloca, justificado a la izquierda, sobre la cinta de  $M$ , con  $M$  en el estado  $q_0$ , y con la cabeza de cinta de  $M$  en la celda que está más a la izquierda. De manera formal, el lenguaje aceptado por  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  es

$$\{w \mid w \text{ in } \Sigma^* \text{ y } q_0 w \xrightarrow{*} \alpha_1 p \alpha_2 \text{ para cualquier } p \text{ en } F, \text{ y } \alpha_1 \text{ y } \alpha_2 \text{ en } \Gamma^*\}.$$

Sin pérdida de generalidad. Dado un TM que reconoce al lenguaje  $L$ , podemos suponer que el TM se detiene, es decir, no hace el movimiento siguiente, cuando la entrada es aceptada. Sin embargo, para palabras no aceptadas, es posible que el TM nunca se detenga.

**Ejemplo 7.1** A continuación se da el diseño de un TM  $M$  que acepta al lenguaje  $L = \{0^n 1^n \mid n \geq 1\}$ . Inicialmente, el tipo de  $M$  contiene a  $0^n 1^n$  seguida de un número infinito de espacios en blanco. Una y otra vez,  $M$  sustituye el cero que está más a la izquierda por  $X$ , se mueve a la derecha hasta el siguiente 1 que está más a la izquierda y lo sustituye por  $Y$ , se mueve hacia la izquierda hasta encontrar a la  $X$  que está más a la derecha, entonces se mueve una celda hacia la derecha hasta el 0 que está más a la

izquierda y repite el ciclo. Si, no obstante, cuando está buscando un 1,  $M$  encuentra en su lugar un espacio en blanco, entonces la máquina se detiene sin aceptar la entrada. Si, después de cambiar un 1 por una  $Y$ ,  $M$  ya no encuentra 0s, entonces verifica que no queden más 1s, y si ya no hay, acepta la entrada.

Sea  $Q = \{q_0, q_1, q_2, q_3, q_4\}$ ,  $\Sigma = \{0, 1\}$ ,  $\Gamma = \{0, 1, X, Y, B\}$  y  $F = \{q_4\}$ . De manera informal, cada estado representa una proposición o un grupo de proposiciones de un programa. El estado  $q_0$  es accesado inicialmente y también inmediatamente antes de cada sustitución de un cero que esté más a la izquierda por una  $X$ . El estado  $q_1$  se utiliza para buscar hacia la derecha, se salta los 0s y las  $Y$ s hasta que encuentra el 1 que está más a la izquierda. Si  $M$  encuentra un 1 lo cambia por una  $Y$  y accesa el estado  $q_2$ . El estado  $q_2$  busca hacia la izquierda una  $X$  y accesa el estado  $q_0$  cuando lo encuentra, moviéndose hacia la derecha, hasta el 0 que está más a la izquierda, mientras cambia de estado. Conforme  $M$  busca hacia la derecha en el estado  $q_1$ , si se encuentra una  $B$  o una  $X$  antes de un 1, entonces la entrada es rechazada; sucede que hay demasiados ceros o la entrada no se encuentra en  $0^* 1^*$ .

El estado  $q_0$  posee otro papel. Si, después de que el estado  $q_2$  encuentra a la  $X$  que está más a la derecha, existe una  $Y$  inmediatamente a su derecha, entonces los 0s se han agotado. A partir de  $q_0$ , barriendo a  $Y$ , el estado  $q_3$  es accesado para que lea a  $Y$  y verifique que no quedan 1s. Si las  $Y$ s son seguidas por una  $B$ , el estado  $q_4$  es accesado y se da la aceptación; de otra manera la cadena es rechazada. En la Fig. 7.2 se muestra la función  $\delta$ . La Fig. 7.3 muestra el cálculo de  $M$  sobre la entrada 0011. Por ejemplo, el primer movimiento se explica por el hecho de que  $\delta(q_0, 0) = (q_1, X, R)$ ; El último movimiento se explica por el hecho de que  $\delta(q_3, B) = (q_4, B, R)$ . El lector deberá simular  $M$  sobre algunas entradas rechazadas, como 001101, 001 y 011.

Estado	Símbolo			
	0	1	X	Y
$q_0$	$(q_1, X, R)$	—	—	$(q_3, Y, R)$
$q_1$	$(q_1, 0, R)$	$(q_2, Y, L)$	—	$(q_1, Y, R)$
$q_2$	$(q_2, 0, L)$	—	$(q_0, X, R)$	$(q_2, Y, L)$
$q_3$	—	—	—	$(q_3, Y, R)$
$q_4$	—	—	—	$(q_4, B, R)$

Fig. 7.2 La función  $\delta$ .

$q_0 0011 \xrightarrow{} X q_1 011 \xrightarrow{} X 0 q_1 11 \xrightarrow{} X q_2 0 Y 1 \xrightarrow{} \\ q_2 X 0 Y 1 \xrightarrow{} X q_0 0 Y 1 \xrightarrow{} X X q_1 Y 1 \xrightarrow{} X X Y q_1 1 \xrightarrow{} \\ X X q_2 Y Y \xrightarrow{} X q_2 X Y Y \xrightarrow{} X X q_0 Y Y \xrightarrow{} X X Y q_3 Y \xrightarrow{} \\ X X Y Y q_3 \xrightarrow{} X X Y Y B q_4$

Fig. 7.3 Un cálculo de  $M$ .

### 7.3 LENGUAJES Y FUNCIONES COMPUTABLES

Se dice que un lenguaje aceptado por una máquina de Turing es *enumerable de manera recursiva* (r.e.). El término "enumerable" se deriva del hecho de que son precisamente éstos los lenguajes cuyas cadenas pueden ser enumeradas (listadas) por una máquina de Turing. "De manera recursiva" es un término matemático anterior a la computación y su significado es parecido a lo que los científicos de la computación llamarían "recursión". La clase de los lenguajes r.e., es muy amplia e incluye propiamente a los CFLs.

La clase de los lenguajes r.e., incluye a algunos lenguajes para los cuales no podemos determinar de manera mecánica su membresía. Si  $L(M)$  es uno de estos lenguajes, entonces cada máquina de Turing que reconozca a  $L(M)$  no se detendrá en alguna entrada que no esté en  $L(M)$ . Si  $w$  está en  $L(M)$ ,  $M$  se detendrá en algún momento en  $w$ . Sin embargo, mientras  $M$  está aún corriendo sobre alguna entrada, no podemos nunca decir si  $M$  finalmente aceptará si la dejamos correr el tiempo suficiente, o si  $M$  correrá eternamente.

Es conveniente destacar una subclase de los conjuntos r.e., conocida como *conjuntos recursivos*, y que la constituyen aquellos lenguajes aceptados por al menos una máquina de Turing que se detiene en todas las entradas (nótese que el detenerse puede ir precedido o no por la aceptación). Veremos en el Capítulo 8 que los conjuntos recursivos son una subclase propia de los conjuntos r.e. Nótese también que según el algoritmo de la Fig. 6.8, cada CFL es un conjunto recursivo.

#### La máquina de Turing como una computadora† de funciones enteras

Además de ser una aceptadora de lenguajes, la máquina de Turing puede considerarse como una computadora de funciones que van de los enteros a los enteros. El planteamiento tradicional consiste en representar a los enteros como números *unitarios*; el entero  $i \geq 0$  se representa mediante la cadena  $0^i$ . Si una función tiene  $k$  argumentos,  $i_1, i_2, \dots, i_k$ , entonces estos enteros se colocan inicialmente en la cinta separados por 1s:  $0^{i_1}10^{i_2}1\dots10^{i_k}$ .

Si la TM se detiene (esté o no en un estado de aceptación) con una cinta que consiste en  $0^m$  para alguna  $m$ , entonces decimos que  $f(i_1, i_2, \dots, i_k) = m$ , en donde  $f$  es la función de  $k$  argumentos calculada por esta máquina de Turing. Adviértase que una TM puede calcular una función de un argumento, una función diferente de dos argumentos y así sucesivamente. Nótese también que si la TM  $M$  calcula la función  $f$  de  $k$  argumentos, entonces  $f$  no necesariamente tiene un valor para cada conjunto diferente de  $k$  tuplas de enteros  $i_1, \dots, i_k$ .

Si  $f(i_1, \dots, i_k)$  se define para toda  $i_1, \dots, i_k$ , entonces decimos que  $f$  es una función *totalmente recursiva*. La función  $f(i_1, \dots, i_k)$  calculada por una máquina de Turing se conoce como *función parcialmente recursiva*. En un cierto sentido, las funciones parcialmente recursivas son análogas a los lenguajes r.e., ya que éstas son calculadas

por máquinas de Turing que pueden detenerse o no en una entrada dada. Las funciones totalmente recursivas corresponden a los lenguajes recursivos, ya que son calculadas por TMs que siempre se detienen. Todas las funciones aritméticas corrientes sobre los enteros, como la multiplicación,  $n!$ ,  $\lceil \log_{2^n} n \rceil$  y  $2^n$ , son funciones totalmente recursivas.

**Ejemplo 7.2** La sustracción propia,  $m - n$  se define como  $m - n$  para  $m \geq n$  y cero para  $m < n$ . La TM

$$M = (\{q_0, q_1, \dots, q_6\}, \{0, 1\}, \{0, 1, B\}, \delta, q_0, B, \emptyset)$$

que se define más adelante, y que comienza con  $0^m10^n$  en su cinta, se detiene con  $0^{m-n}$  en su cinta.  $M$  sustituye de manera repetida su 0 del frente por un espacio en blanco, entonces busca hacia la derecha un 1 seguido por un 0 y cambia el 0 por 1. En seguida  $M$  se mueve hacia la izquierda hasta que encuentra un espacio en blanco y entonces repite el ciclo. La repetición termina si

- Buscando hacia la derecha un 0,  $M$  encuentra un espacio en blanco. Entonces, los  $n$  0s, que están en  $0^m10^n$  han sido todos cambiados a 1s y  $n + 1$  de los  $m$  0s se han cambiado a  $B$ .  $M$  sustituye a los  $n + 1$  1s por un 0 y  $n$  Bs, dejando  $m - n$  0s en su cinta.
- Comenzando el ciclo,  $M$  no puede encontrar un 0 que sea cambiado a un espacio en blanco, debido a que los primeros  $m$  0s ya han sido cambiados. Entonces  $n \geq m$ , de modo que  $m - n = 0$ .  $M$  sustituye todos los 1s y 0s restantes por  $B$ .

La función  $\delta$  se describe a continuación.

- $\delta(q_0, 0) = (q_1, B, R)$   
Comienza el ciclo. Sustituye el 0 del frente por B.
- $\delta(q_1, 0) = (q_1, 0, R)$   
 $\delta(q_1, 1) = (q_2, 1, R)$   
Escruta hacia la derecha, buscando el primer 1.
- $\delta(q_2, 1) = (q_2, 1, R)$   
 $\delta(q_2, 0) = (q_3, 1, L)$   
Escruta hacia la derecha saltándose los 1s hasta que encuentra un 0. Cambia ese 0 por 1
- $\delta(q_3, 0) = (q_3, 0, L)$   
 $\delta(q_3, 1) = (q_3, 1, L)$   
 $\delta(q_3, B) = (q_0, B, R)$   
Se mueve hacia la izquierda a un espacio en blanco. Accesa el estado  $q_0$  para repetir el ciclo.
- $\delta(q_2, B) = (q_4, B, L)$   
 $\delta(q_4, 1) = (q_4, B, L)$   
 $\delta(q_4, 0) = (q_4, 0, L)$   
 $\delta(q_4, B) = (q_6, 0, R)$   
Si en el estado  $q_2$  se encuentra una  $B$  antes que un 0, tenemos la situación (i) descrita

† En este caso, el término *computadora* se utiliza para referirse a una máquina que calcula funciones de enteros. (N. del T.)

más arriba. Se accesa el estado  $q_4$  y se mueve hacia la izquierda, cambiando todos los 1 por Bs hasta encontrar una B. Esta B se cambia de nuevo a 0, el estado  $q_6$  es accesado y M se detiene.

6.  $\delta(q_0, 1) = (q_5, B, R)$
- $\delta(q_5, 0) = (q_5, B, R)$
- $\delta(q_5, 1) = (q_5, B, R)$
- $\delta(q_5, B) = (q_6, B, R)$

Si en el estado  $q_0$  se encuentra un 1 en lugar de un 0, el primer bloque de 0s ha sido agotado, como en la situación (ii) anterior. M accesa el estado  $q_5$  para borrar el resto de la cinta, entonces accesa el estado  $q_6$  y se detiene.

Una muestra del cálculo de M sobre la entrada 0010 es:

$$\begin{aligned} q_00010 &\dashv Bq_1010 \dashv B0q_110 \dashv B01q_20 \dashv \\ B0q_311 &\dashv Bq_3011 \dashv q_3B011 \dashv Bq_0011 \dashv \\ BBq_111 &\dashv BB1q_21 \dashv BB11q_2 \dashv BB1q_41 \dashv \\ BBq_41 &\dashv Bq_4 \dashv B0q_6 \end{aligned}$$

Sobre la entrada 0100, M se comporta de la manera siguiente:

$$\begin{aligned} q_00100 &\dashv Bq_1100 \dashv B1q_200 \dashv Bq_3110 \dashv \\ q_3B110 &\dashv Bq_0110 \dashv BBq_510 \dashv BBBq_50 \dashv \\ BBBBq_5 &\dashv BBBBq_6 \end{aligned}$$

## 7.4 TECNICAS PARA LA CONSTRUCCION DE MAQUINAS DE TURING

Diseñar máquinas de Turing mediante la escritura de un conjunto completo de estados y, de una función que dé el movimiento siguiente es una tarea obviamente infructuosa, con el fin de describir construcciones de máquinas de Turing complicadas necesitaremos algunas herramientas conceptuales de “alto nivel”. En la presente sección trataremos acerca de las principales.

### Almacenamiento en el control finito

El control finito puede utilizarse para contener una cantidad finita de información. Para hacer esto el estado se escribe como un par de elementos, uno que ejerce el control y el otro que almacena un símbolo. Debe hacerse énfasis en que este arreglo es con motivos conceptuales solamente. No se ha hecho ninguna modificación a la definición de la máquina de Turing.

**Ejemplo 7.3** Considérese la máquina de Turing M que observa el primer símbolo de entrada, lo registra en su control finito y verifica que el símbolo no aparezca en algún

otro lugar de su entrada. Nótese que M acepta a un conjunto regular, pero M servirá para fines de demostración:

$$M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, [q_0, B], B, F),$$

en donde Q es  $\{q_0, q_1\} \times \{0, 1, B\}$ . Es decir, Q consiste en todos los pares  $[q_0, 0]$ ,  $[q_0, 1]$ ,  $[q_0, B]$ ,  $[q_1, 0]$ ,  $[q_1, 1]$  y  $[q_1, B]$ . El conjunto F es  $\{[q_1, B]\}$ . La intención aquí es que la primer componente del estado controle la acción, mientras que la segunda “recuerde” un símbolo.

Definimos  $\delta$  de la forma siguiente.

- 1 a)  $\delta([q_0, B], 0) = ([q_1, 0], 0, R)$ , b)  $\delta([q_0, B], 1) = ([q_1, 1], 1, R)$ .

Inicialmente,  $q_0$  es la componente de control del estado y M se mueve hacia la derecha. La primer componente de los estados de M se convierte en  $q_1$ , y el primer símbolo visto se almacena en la segunda componente.

- 2 a)  $\delta([q_1, 0], 1) = ([q_1, 0], 1, R)$ , b)  $\delta([q_1, 1], 0) = ([q_1, 1], 0, R)$ .

Si M tiene almacenado un 0 y ve un 1, o viceversa, entonces M continúa su movimiento hacia la derecha.

- 3 a)  $\delta([q_1, 0], B) = ([q_1, B], 0, L)$ , b)  $\delta([q_1, 1], B) = ([q_1, B], 0, L)$ .

M accesa el estado final  $[q_1, B]$  si alcanza un espacio en blanco sin haber encontrado primero una segunda copia del símbolo que está más a la izquierda.

Si M alcanza un espacio en blanco en el estado  $[q_1, 0]$  o en  $[q_1, 1]$ , entonces acepta. Para el estado  $[q_1, 0]$  y el símbolo 0, o para estado  $[q_1, 1]$  y el símbolo 1,  $\delta$  no está definida. Por consiguiente si M encuentra el símbolo de cinta almacenado en su estado, M se detiene sin aceptar.

En general, podemos permitir que el control finito tenga  $k$  componentes que, a excepción de una, almacenen información.

### Pistas múltiples

Podemos imaginar que la cinta de la máquina de Turing está dividida en  $k$  pistas para cualquiera  $k$  finita. En la Fig. 7.4 se muestra este arreglo, con  $k = 3$ . Los símbolos en la cinta se consideran como  $k$ -tuplas, una componente por pista.

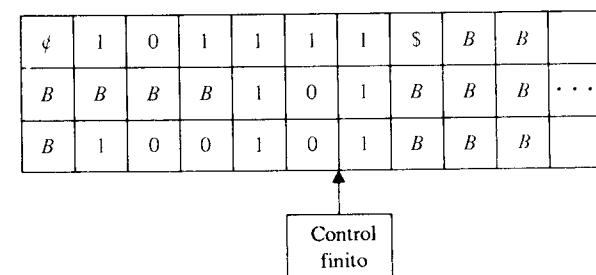


Fig. 7.4 Máquina de Turing de tres pistas.

**Ejemplo 7.4** La cinta de la Fig. 7.4 pertenece a una máquina de Turing que toma una entrada binaria mayor a dos, escrita en la primera pista, y determina si la entrada es un número primo. En la primera pista la entrada está antecedida y precedida por  $\phi$  y  $\$$ . Por tanto los símbolos permisibles son  $[\phi, B, B], [0, B, B], [1, B, B]$  y  $[\$, B, B]$ . Estos símbolos pueden identificarse con  $\phi$ , 0, 1 y  $\$$ , respectivamente, cuando se consideran como símbolos de entrada. El símbolo del espacio en blanco puede ser identificado con  $[B, B]$ .

Para probar si una entrada es un primo, la TM primer escribe el número dos en binario en la segunda pista y copia la primera pista en el tercero. Entonces la segunda pista se sustraen del tercero cuantas veces sea posible, dividiendo efectivamente la tercera pista entre el segundo y dejando el residuo.

Si el residuo es cero, el número que se encuentra en la primera pista no es un primo. Si el residuo no es cero, el número que está en la segunda pista se aumenta en uno. Si la segunda pista iguala a la primera, el número de la primera pista es primo, porque no puede dividirse entre cualquier número que se encuentre propiamente entre uno y sí mismo. Si el segundo es menor que el primero, la operación completa se repite para el nuevo número que está en la segunda pista.

En la Fig. 7.4, la TM hace la prueba para determinar si 47 es primo. La TM divide entre cinco; el 5 ya ha sido sustraído dos veces, de modo que aparece un 37 en la tercera pista.

### Señalamiento de símbolos

Señalar símbolos es un truco útil para visualizar cómo una TM reconoce lenguajes definidos mediante cadenas repetidas, como

$$\{ww \mid w \text{ en } \Sigma^*\}, \quad \{wcy \mid w \text{ y } y \text{ en } \Sigma^*, w \neq y\} \quad \text{o} \quad \{ww^R \mid w \text{ en } \Sigma^*\}.$$

También resulta de utilidad cuando se deben comparar las longitudes de subcadenas, como en los lenguajes

$$\{a^i b^i \mid i \geq 1\} \quad \text{o} \quad \{a^i b^j c^k \mid i \neq j \text{ o } j \neq k\}.$$

Introducimos una pista extra en la cinta que contiene al espacio en blanco o al símbolo  $\checkmark$ . El símbolo  $\checkmark$  aparece cuando el símbolo que se encuentra debajo de él ha sido considerado por la TM en una de sus comparaciones.

**Ejemplo 7.5** Considérese la máquina de Turing  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ , que reconoce al lenguaje  $\{wcwlw \text{ está en } (a+b)^*\}$ . Sea

$$Q = \{[q, d] \mid q = q_1, q_2, \dots, q_9, \quad \text{y} \quad d = a, b, \text{ or } B\}.$$

La segunda componente del estado se utiliza para almacenar un símbolo de entrada,

$$\Sigma = \{[B, d] \mid d = a, b, \text{ or } c\}.$$

El símbolo de entrada,  $[B, d]$  se identifica con  $d$ . Recuérdese que las dos “pistas” son sólo herramientas conceptuales; es decir,  $[B, d]$  es solamente otro nombre que se le da a  $d$ :

$$\Gamma = \{[X, d] \mid X = B \text{ o } \checkmark \quad \text{y} \quad d = a, b, c, \text{ or } B\},$$

$$q_0 = [q_1, B], \quad \text{y} \quad F = \{[q_9, B]\};$$

$[B, B]$  se identifica con  $B$ , el símbolo para el espacio en blanco. Para  $d = a$  o  $b$  y  $e = a$  o  $b$  definimos  $\delta$  de la forma siguiente.

$$1. \quad \delta([q_1, B], [B, d]) = ([q_2, d], [\checkmark, d], R).$$

$M$  verifica el símbolo barrido en la cinta, lo almacena en el control finito y se mueve hacia la derecha.

$$2. \quad \delta([q_2, d], [B, e]) = ([q_2, d], [B, e], R).$$

$M$  continúa moviéndose hacia la derecha, pasando símbolos que no han sido marcados, en busca de  $c$ .

$$3. \quad \delta([q_2, d], [B, c]) = ([q_3, d], [B, c], R).$$

Una vez encontrado  $c$ ,  $M$  accesa un estado cuya primera componente es  $q_3$ .

$$4. \quad \delta([q_3, d], [\checkmark, e]) = ([q_3, d], [\checkmark, e], R).$$

$M$  se mueve hacia la derecha pasando sobre símbolos señalados.

$$5. \quad \delta([q_3, d], [B, d]) = ([q_4, B], [\checkmark, d], L).$$

$M$  encuentra un símbolo sin señalar. Si tal símbolo se corresponde con el símbolo almacenado en el control finito,  $M$  lo marca y comienza a moverse hacia la izquierda. Si el símbolo no corresponde,  $M$  no hace un siguiente movimiento y se detiene sin aceptar.  $M$  también se detiene si en el estado  $q_3$  alcanza a  $[B, B]$  antes de encontrar un símbolo sin señalar.

$$6. \quad \delta([q_4, B], [\checkmark, d]) = ([q_4, B], [\checkmark, d], L).$$

$M$  se mueve hacia la izquierda pasando símbolos señalados.

$$7. \quad \delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L).$$

$M$  encuentra al símbolo  $c$ .

$$8. \quad \delta([q_5, B], [B, d]) = ([q_6, B], [B, d], L).$$

Si el símbolo que se encuentra inmediatamente a la izquierda de  $c$  no está marcado,  $M$  se mueve hacia la izquierda hasta encontrar el símbolo señalado que está más a la derecha.

$$9. \quad \delta([q_6, B], [B, d]) = ([q_6, B], [B, d], L).$$

$M$  prosigue hacia la izquierda.

$$10. \quad \delta([q_6, B], [\checkmark, d]) = ([q_1, B], [\checkmark, d], R).$$

$M$  encuentra un símbolo señalado y se mueve hacia la derecha para tomar otro símbolo y compararlos. La primera componente del estado se convierte en  $q_1$  de nuevo.

$$11. \quad \delta([q_5, B], [\checkmark, d]) = ([q_7, B], [\checkmark, d], R).$$

$M$  estará en el estado  $[q_5, B]$  inmediatamente después de cruzar  $c$  en su movimiento hacia la izquierda. (Véase la Regla 7.) Si aparece un símbolo señalado inmediatamente a la izquierda de  $c$ , todos los símbolos que están a la izquierda de  $c$  han sido marcados.  $M$  debe probar si todos los símbolos que se encuentran a la derecha ya han sido señalados. Si esto sucede, quiere decir que los símbolos han sido comparados de manera apropiada con los que se encuentran a la izquierda de  $c$ , de modo que  $M$  acepta.

$$12. \delta([q_7, B], [B, c]) = ([q_8, B], [B, c], R).$$

$M$  se mueve hacia la derecha sobre  $c$ .

$$13. \delta([q_8, B], [\checkmark, d]) = ([q_8, B], [\checkmark, d], R).$$

$M$  se mueve hacia la derecha pasando sobre los símbolos marcados.

$$14. \delta([q_8, B], [B, B]) = ([q_9, B], [\checkmark, B], L).$$

Si  $M$  encuentra a  $[B, B]$ , el espacio en blanco, se detiene y acepta. Si  $M$  encuentra un símbolo que no está señalado, cuando la primer componente del estado es  $q_8$ , se detiene sin aceptar.

## Corrimiento

Una máquina de Turing puede crear espacio en su cinta mediante el corrimiento de todos los símbolos, que no sean espacios en blanco, un número finito de celdas hacia la derecha. Para hacer esto, la cabeza de la cinta hace una excursión hacia la derecha, almacenando repetidamente los símbolos leídos en su control finito y sustituyéndolos con símbolos leídos de las celdas que están a la izquierda. La TM puede entonces regresar a las celdas vacías e imprimir símbolos seleccionados por ella. Si hay espacio disponible, puede desplazar bloques de símbolos hacia la izquierda de manera similar.

**Ejemplo 7.6.** Construiremos parte de una máquina de Turing,  $M = (Q, \Sigma, \Gamma, \delta, q_0, B)$ , que ocasionalmente puede tener necesidad de correr símbolos, que no sean  $B$ , dos celdas hacia la derecha. Suponemos que la cinta de  $M$  no contiene espacios en blanco entre cadenas o símbolos no vacíos, de modo que cuando alcanza un espacio en blanco reconoce que debe detener el proceso de corrimiento. Hagamos que  $Q$  contenga estados de la forma  $[q, A_1, A_2]$  para  $q = q_1$  o  $q_2$  y  $A_1$  y  $A_2$  en  $\Gamma$ . Sea  $X$  un símbolo especial que no es utilizado por  $M$  excepto en el proceso de corrimiento.  $M$  inicia el proceso de corrimiento en el estado  $[q_1, B, B]$ . Las partes de interés de la función  $\delta$  son las siguientes:

$$1. \delta([q_1, B, B], A_1) = ([q_1, B, A_1], X, R) \text{ para } A_1 \text{ en } \Gamma - \{B, X\}.$$

$M$  almacena el primer símbolo leído en la tercera componente de su estado.  $X$  se imprime en la celda barrida y  $M$  se mueve hacia la derecha.

$$2. \delta([q_1, B, A_1], A_2) = ([q_1, A_1, A_2], X, R) \text{ para } A_1 \text{ y } A_2 \text{ en } \Gamma - \{B, X\}.$$

$M$  corre el símbolo de la tercera componente a la segunda, almacena el símbolo que está leyendo en la tercera componente, imprime una  $X$  y se desplaza hacia la derecha.

$$3. \delta([q_1, A_1, A_2], A_3) = ([q_1, A_2, A_3], A_1, R) \text{ para } A_1, A_2 \text{ y } A_3 \text{ en } \Gamma - \{B, X\}.$$

Ahora  $M$  lee repetidamente un símbolo  $A_3$ , lo almacena en la tercera componente del estado, corre el símbolo que se encontraba antes en la tercera componente,  $A_2$ , a la segunda componente, deposita el símbolo que anteriormente se encontraba en la segunda componente,  $A_1$ , en la celda barrida y se mueve hacia la derecha. Por consiguiente un símbolo será depositado dos celdas a la derecha de su posición original.

$$4. \delta([q_1, A_1, A_2], B) = ([q_1, A_2, B], A_1, R) \text{ para } A_1 \text{ y } A_2 \text{ en } \Gamma - \{B, X\}.$$

Cuando se ve un espacio en blanco sobre la cinta, los símbolos almacenados son depositados sobre la cinta.

$$5. \delta([q_1, A_1, B], B) = ([q_2, B, B], A_1, L).$$

Después de que todos los símbolos se han depositado,  $M$  asigna la primera componente del estado a  $q_2$  y se desplaza hacia la izquierda hasta encontrar una  $X$ , que señala la celda vacía que se encuentra más a la derecha.

$$6. \delta([q_2, B, B], A) = ([q_2, B, B], A, L) \text{ para } A \text{ en } \Gamma - \{B, X\}.$$

$M$  se mueve hacia la izquierda hasta que encuentra una  $X$ . Cuando encuentra una  $X$ ,  $M$  un estado que suponemos está en  $Q$  y reanuda sus otras funciones.

## Subrutinas

De la misma manera como sucede con los programas, la construcción de un diseño “modular” o de “arriba hacia abajo” se facilita si utilizamos subrutinas para definir los procesos elementales. Una máquina de Turing puede simular cualquier tipo de subrutina que se encuentre en los lenguajes de programación, incluyendo procedimientos recursivos y cualesquiera de los mecanismos para pasar parámetros conocidos. Describiremos, aquí, solamente el uso de subrutinas no recursivas y sin parámetros; pero aun éstas son herramientas bastante poderosas.

La idea general consiste en escribir parte de un programa TM que sirva como subrutina; tendrá un estado inicial designado y un estado de retorno también designado que, de manera temporal, no posee movimientos y que será utilizado para efectuar un retorno a la rutina desde la cual se hacen los llamados. Para diseñar una TM que “llame” a la subrutina, se construye un nuevo conjunto de estados para la subrutina, y se especifica un movimiento a partir del estado de retorno. La llamada se efectúa mediante el acceso del estado inicial de la subrutina, y el retorno se lleva a cabo mediante el movimiento a partir del estado de regreso.

**Ejemplo 7.7** El diseño de una TM  $M$  que construya la función totalmente recursiva “multiplicación” se da a continuación.  $M$  comienza con  $0^m 1 0^n$  en su cinta y termina con  $0^m$  antecedida y precedida por espacios en blanco. La idea general consiste en colocar un 1 después de  $0^m 1 0^n$  y después copiar el bloque de  $n$  ceros en el extremo derecho  $m$  veces, borrando, cada vez, uno de los  $m$  ceros. El resultado es  $1 0^n 1 0^m$ . Finalmente se borra el prefijo  $1 0^m$ , dejando  $0^m$ . La parte modular del algoritmo es una subrutina

COPY, que comienza en una ID  $0^m1q_00^n10^i$  y en algún momento accesa una ID  $0^m1q_00^n10^{i+n}$ . En la Fig. 7.5 se define la subrutina COPY. En el estado  $q_1$ , al estar observando un 0,  $M$  lo cambia por un 2 y accesa el estado  $q_2$ . En el estado  $q_2$ ,  $M$  se mueve hacia la derecha hacia el siguiente espacio en blanco, deposita el 0 y comienza a moverse hacia la izquierda en el estado  $q_3$ . En el estado  $q_3$ ,  $M$  se mueve hacia la izquierda hasta un 2. Al alcanzar un 2, se accesa el estado  $q_1$  y el proceso se repite hasta que es encontrado el 1, señalando que el proceso de copiado está completo. El estado  $q_4$  se utiliza para convertir los 2s de nuevo a 0s, y la subrutina se detiene en  $q_5$ .

	0	1	2	B
$q_1$	$(q_2, 2, R)$	$(q_4, 1, L)$		
$q_2$	$(q_2, 0, R)$	$(q_2, 1, R)$		$(q_3, 0, L)$
$q_3$	$(q_3, 0, L)$	$(q_3, 1, L)$	$(q_1, 2, R)$	
$q_4$		$(q_5, 1, R)$	$(q_4, 0, L)$	

Fig. 7.5  $\delta$  para la subrutina COPY.

Para completar el programa de multiplicación, agregamos estados para convertir la ID inicial  $q_00^m10^i$  a  $B0^{m-1}q_10^i1$ . Esto es, se necesitan las reglas

$$\begin{aligned}\delta(q_0, 0) &= (q_6, B, R), \\ \delta(q_6, 0) &= (q_6, 0, R), \\ \delta(q_6, 1) &= (q_1, 1, R).\end{aligned}$$

Se necesitan estados adicionales para convertir una ID  $B^i0^{m-i}1q_00^n10^i$  a  $B^{i+1}0^{m-i-1}1q_10^n10^i$ , que reinicia la subrutina COPY, y para verificar si  $i = m$ , es decir, todos los ceros han sido borrados. En el caso en que  $i = m$ , el  $10^i1$  se borra y el cálculo se detiene en el estado  $q_{12}$ . Estos movimientos se muestran en la Fig. 7.6.

	0	1	2	B
$q_5$	$(q_7, 0, L)$			
$q_7$		$(q_8, 1, L)$		
$q_8$	$(q_9, 0, L)$			$(q_{10}, B, R)$
$q_9$	$(q_9, 0, L)$			$(q_0, B, R)$
$q_{10}$		$(q_{11}, B, R)$		
$q_{11}$	$(q_{11}, B, R)$		$(q_{12}, B, R)$	

Fig. 7.6 Movimientos adicionales para la TM que efectúa la multiplicación.

Nótese que podríamos hacer más de una llamada a la subrutina si la reescribimos utilizando un nuevo conjunto de estado para cada llamada.

## 7.5 MODIFICACIONES DE LAS MAQUINAS DE TURING

Una de las razones para que la máquina de Turing sea aceptada como un modelo general de un cálculo consiste en que el modelo con el que hemos estado tratando es equivalente a muchas versiones modificadas que podrían parecer improvisadas para aumentar la potencia de cálculo. En esta sección daremos demostraciones informales de algunos de estos teoremas de equivalencia.

### Cinta infinita de dos direcciones

Una máquina de Turing con cinta infinita de dos direcciones está representada por  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ , como en el modelo original. Como lo implica su nombre, la cinta es infinita hacia la izquierda y hacia la derecha. Denotamos una ID de un dispositivo tal de igual modo que para una TM infinita de una dirección. Imaginamos, sin embargo, que existe una infinidad de espacios en blanco hacia la izquierda y hacia la derecha de la parte que no está en blanco de la cinta.

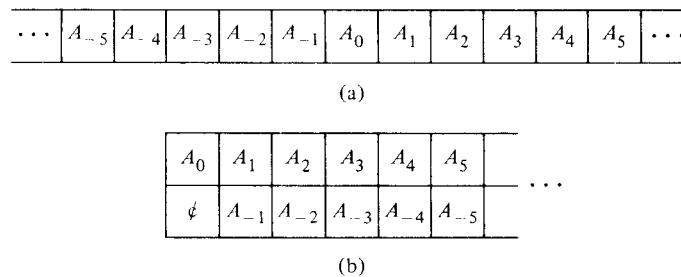
La relación  $\vdash_M$  que asocia a dos IDs si la ID de la derecha se obtiene a partir de la ID de la izquierda mediante un solo movimiento, se define de igual modo que en el modelo original, con la excepción de que si  $\delta(q, X) = (p, Y, L)$ , entonces  $qX\alpha \vdash_M pBY\alpha$  (en el modelo original, no se podría haber hecho ningún movimiento), y si  $\delta(q, X) = (p, B, R)$ , entonces  $qX\alpha \vdash_M p\alpha$  (en el modelo original,  $B$  aparecería a la izquierda de  $p$ ).

La ID inicial es  $q_0w$ . Mientras que en el modelo original existía un extremo izquierdo, en esta versión no existe un borde izquierdo para que la máquina de Turing se “salga”, de modo que puede proseguir hasta donde se desee. La relación  $\vdash_M$  como siempre, asocia a los IDs si la que se encuentra a la derecha puede obtenerse de la que se encuentra a la izquierda mediante un cierto número de movimientos.

**Teorema 7.1**  $L$  es reconocido por una máquina de Turing con una cinta infinita de dos direcciones si y sólo si es reconocido por una TM con una cinta infinita de una dirección.

**Demuestra**ción La demostración de que una TM con una cinta infinita de dos direcciones puede simular a una TM con una cinta infinita de una dirección es fácil. La primera señala la celda que está a la izquierda de la posición inicial de la cabeza y, por consiguiente, simula a la última. Si durante la simulación se alcanza la celda marcada, la simulación termina sin haber aceptación.

De manera inversa, sea  $M_2 = (Q_2, \Sigma_2, \Gamma_2, \delta_2, q_2, B, F_2)$  una TM con una cinta infinita de dos direcciones. Construimos  $M_1$ , una máquina de Turing que simula a  $M_2$  y que tiene una cinta infinita sólo hacia la derecha.  $M_1$  tendrá dos pistas, una para representar las celdas de la cinta de  $M_2$  que están a la derecha de la celda barrida inicialmente, incluyéndola; la otra para representar, en orden inverso, las celdas que están a la izquierda de la celda inicial. La relación entre las cintas de  $M_2$  y  $M_1$  se muestra en la Fig. 7.7, con la celda inicial de  $M_2$  señalada con 0, las celdas que se encuentran a la derecha con 1, 2, ..., y las celdas que están a la izquierda con -1, -2, ... .

Fig. 7.7 a) Cinta de  $M_2$  (b) Cinta de  $M_1$ 

La primera celda de la cinta  $M_1$  contiene al símbolo  $\emptyset$  en la pista inferior, e indica que es la celda que está más a la izquierda. El control finito de  $M_1$  dice si  $M_2$  estará barriendo un símbolo que aparece en la pista superior o en el interior de  $M_1$ .

Debe ser bastante evidente que  $M_1$  puede construirse para simular  $M_2$ , en el sentido de que mientras  $M_2$  se encuentra a la derecha de la posición inicial de su cabeza de entrada,  $M_1$  trabaja en la pista superior. Mientras  $M_2$  se encuentra a la izquierda de la posición inicial de su cabeza de cinta,  $M_1$  trabaja en su pista inferior, moviéndose en la dirección opuesta a la dirección en que  $M_2$  se desplaza. Los símbolos de entrada de  $M_1$  son símbolos que tienen un espacio en blanco en el registro inferior y un símbolo de entrada de  $M_2$  en el registro superior. Este símbolo puede identificarse con el correspondiente símbolo de entrada de  $M_2$ .  $B$  se identifica con  $[B, B]$ .

Daremos ahora una construcción formal de  $M_1 = (Q_1, \Sigma_1, \Gamma_1, \delta_1, q_1, B, F_1)$ . Todos los estados,  $Q_1$ , de  $M_1$  son objetos de la forma  $[q, U]$  o  $[q, D]$ , en donde  $q$  está en  $Q_2$ , más el símbolo  $q_1$ . Nótese que la segunda componente indicará si  $M_1$  está trabajando en el registro superior ( $U$  para indicar arriba) o en el inferior ( $D$  para indicar abajo). Todos los símbolos de cinta de  $\Gamma_1$  son objetos de la forma  $[X, Y]$ , en donde  $X$  y  $Y$  están en  $\Gamma_2$ . Además,  $Y$  puede ser  $\emptyset$ , símbolo que no se encuentra en  $\Gamma_2$ .  $\Sigma_1$  consiste en todos los símbolos  $[a, B]$ , en donde  $a$  está en  $\Sigma_2$ .  $F_1$  es  $\{[q, U], [q, D] \mid q \text{ está en } F_2\}$ . Definimos  $\delta_1$  de la forma siguiente:

1. Para cada  $a$  en  $\Sigma_2 \cup \{B\}$ ,

$$\delta_1(q_1, [a, B]) = ([q, U], [X, \emptyset], R) \quad \text{si} \quad \delta_2(q_2, a) = (q, X, R).$$

Si  $M_2$  se mueve a la derecha en su primer movimiento,  $M_1$  imprime  $\emptyset$  en la pista inferior para señalar el final de la cinta, fija su segunda componente de estado a  $U$  y se mueve hacia la derecha. La primera componente de estado de  $M_1$  contiene al estado de  $M_2$ . En la pista superior,  $M_1$  imprime el símbolo  $X$  que es impreso por  $M_2$ .

2. Para cada  $a$  en  $\Sigma_2 \cup \{B\}$ ,

$$\delta_1(q_1, [a, B]) = ([q, D], [X, \emptyset], R) \quad \text{si} \quad \delta_2(q_2, a) = (q, X, L).$$

Si  $M_2$  se mueve hacia la izquierda,  $M_1$  registra el siguiente estado de  $M_2$  y el símbolo impreso por  $M_2$  como en (1), pero fija la segunda componente de su estado

a  $D$  y se mueve hacia la derecha. De nuevo, se imprime  $\emptyset$  en la pista inferior para señalar el extremo izquierdo de la cinta.

3. Para cada  $[X, Y]$  en  $\Gamma_1$ , con  $Y \neq \emptyset$  y  $A = L$  o  $R$ ,

$$\delta_1([q, U], [X, Y]) = ([p, U], [Z, Y], A) \quad \text{si} \quad \delta_2(q, X) = (p, Z, A).$$

$M_1$  simula a  $M_2$  en la pista superior.

4. Para cada  $[X, Y]$  en  $\Gamma_1$ , con  $Y \neq \emptyset$ ,

$$\delta_1([q, D], [X, Y]) = ([p, D], [X, Z], A) \quad \text{si} \quad \delta_2(q, Y) = (p, Z, \bar{A}).$$

Aquí  $A$  es  $L$  si  $\bar{A}$  es  $R$  y  $\bar{A}$  es  $R$  si  $A$  es  $L$ .  $M_1$  simula a  $M_2$  en la pista inferior de  $M_1$ .

La dirección del movimiento de la cabeza de  $M_1$  es opuesta a la de la cabeza de  $M_2$ .

5.  $\delta_1([q, U], [X, \emptyset]) = \delta_1([q, D], [X, \emptyset])$

$$= ([p, C], [Y, \emptyset], R) \quad \text{si} \quad \delta_2(q, X) = (p, Y, A).$$

Aquí,  $C = U$  si  $A = R$  y  $C = D$  si  $A = L$ .  $M_1$  simula un movimiento de  $M_2$  en una celda que inicialmente era barrida por  $M_2$ . En seguida  $M_1$  trabaja en la pista superior o en la inferior, dependiendo de la dirección en la cual se mueve  $M_2$ . En esta situación,  $M_1$  siempre se moverá hacia la derecha.  $\square$

### Máquinas de Turing multicintas

En la Fig. 7.8 se muestra una máquina de Turing multicintas. Consiste en un control finito con  $k$  cabezas de cinta y  $k$  cintas; cada cinta es infinita en ambas direcciones. En un solo movimiento, dependiendo del estado del control finito y del símbolo barrido por cada una de las cabezas de cinta, la máquina de Turing puede:

1. cambiar de estado;
2. imprimir un nuevo símbolo en cada una de las celdas barridas por sus cabezas de cinta;
3. mover cada una de sus cabezas de cinta, de manera independiente, una celda hacia la izquierda a la derecha, o mantenerlas estacionarias.

En un principio, la entrada aparece en la primera cinta y las otras cintas están en blanco. No definiremos el dispositivo de manera más formal porque el formalismo es muy pesado y significa una generalización de la notación que se utilizó para las TMs de una sola cinta.

**Teorema 7.2** Si el lenguaje  $L$  es aceptado por una máquina de Turing multicintas, entonces es aceptado por una máquina de Turing de una sola cinta.

**Demostración** Sea  $L$  un lenguaje aceptado por  $M_1$ , una TM con  $k$  cintas. Podemos construir  $M_2$ , una TM de una sola cinta con  $2k$  pistas, dos pistas por cada cinta de  $M_1$ . Una pista recoge el contenido de la correspondiente cinta de  $M_1$  y la otra permanece en blanco, excepto por un señalador en la celda que contiene al símbolo barrido por la

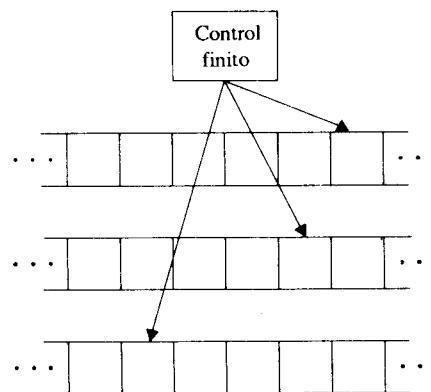


Fig. 7.8 Máquina de Turing multicintas.

correspondiente cabeza de  $M_1$ . El arreglo se ilustra en la Fig. 7.9. El control finito de  $M_2$  almacena el estado de  $M_1$ , junto con una cuenta del número de señaladores de cabeza que están a la derecha de la cabeza de cinta de  $M_2$ .

Cabeza 1		$X$				
Cinta 1	$A_1$	$A_2$	$\dots$		$\dots$	$A_m$
Cabeza 2				$X$		
Cinta 2	$B_1$	$B_2$	$\dots$		$\dots$	$B_m$
Cabeza 3	$X$					
Cinta 3	$C_1$	$C_2$	$\dots$		$\dots$	$C_m$

Fig. 7.9 Simulación de tres cintas mediante una.

Cada movimiento de  $M_1$  es simulado por un recorrido de izquierda a derecha y después de derecha a izquierda de la cabeza de cinta de  $M_2$ . En un principio, la cabeza de  $M_2$  se encuentra en la celda que está más a la izquierda y que contiene un señalador de cabeza. Para simular un movimiento de  $M_1$ ,  $M_2$  hace el recorrido hacia la derecha, visitando cada una de las celdas que tienen señaladores de cabeza y registrando el símbolo barrido por cada cabeza de  $M_1$ . Cuando  $M_2$  cruza un señalador de cabeza, debe actualizar la cuenta de los señaladores de cabeza que están a su derecha. Cuando ya no hay más señaladores de cabeza a la derecha,  $M_2$  ha visto a los símbolos barridos por cada una de las cabezas de  $M_1$ , de modo que  $M_2$  tiene la información suficiente para determinar

el movimiento de  $M_1$ . Ahora  $M_2$  efectúa un recorrido hacia la izquierda hasta que alcanza al señalador de cabeza que se encuentra más a la izquierda. La cuenta de señaladores que están a la derecha permite a  $M_2$  decir cuándo ha avanzado lo suficiente. Conforme  $M_2$  pasa cada señalador de cabeza en el recorrido hacia la izquierda, actualiza el símbolo de cinta de  $M_1$  “barrido” por ese señalador de cabeza, y desplaza a dicho señalador un símbolo hacia la izquierda o derecha para simular el movimiento de  $M_1$ . Si el nuevo estado de  $M_1$  es aceptado, entonces  $M_2$  acepta.  $\square$

Nótese que en la primera simulación de esta sección: aquella que se refiere a la de una TM de cinta infinita con dos direcciones mediante una TM de cinta infinita con una dirección, la simulación se hizo movimiento por movimiento. En la presente, sin embargo, se necesitan muchos movimientos de  $M_2$  para simular un movimiento de  $M_1$ . De hecho, ya que después de  $k$  movimientos, las cabezas de  $M_1$  pueden estar  $2k$  celdas aparte, le toma alrededor de  $\sum_{i=1}^k 2i \approx 2k^2$  movimientos de  $M_2$  para simular  $k$  movimientos de  $M_1$ . (En realidad, se pueden necesitar  $2k$  movimientos más para simular el movimiento de las cabezas hacia la derecha.) Esta disminución cuadrática de la rapidez en la simulación que se da cuando pasamos de una TM multicintas a una TM de una sola cinta es inherentemente necesaria para ciertos lenguajes. En tanto más adelante, en el Capítulo 12, damos una demostración, aquí nos limitaremos a dar un ejemplo de la eficiencia de las TM multicintas.

**Ejemplo 7.8** El lenguaje  $L = \{ww^k \mid w \text{ está en } (0+1)^*\}$  puede ser reconocido por una TM de una sola cinta mediante el movimiento de la cabeza de cinta en una dirección y la otra sobre la entrada, verificando los símbolos de ambos extremos y comparándolos. El proceso es parecido al que se siguió en el Ejemplo 7.5.

Para reconocer  $L$  con una TM de dos cintas, la entrada es copiada en la segunda cinta. La entrada que está en una de las cintas se compara con la inversa en la otra cinta moviendo las cabezas en direcciones opuestas, y se verifica la longitud de la entrada para asegurarse que es par.

Adviértase que el número de movimientos utilizados para reconocer  $L$  mediante una TM de una sola cinta es aproximadamente el cuadrado de la longitud de la entrada, mientras que con una máquina de dos cintas es suficiente un tiempo proporcional a la longitud de la entrada.

### Máquinas de Turing no determinísticas

Una máquina de Turing no determinística es un dispositivo con un control finito y una sola cinta infinita de una dirección. Para un estado dado y un símbolo de cinta barrido por la cabeza, la máquina tiene un número finito de alternativas para el siguiente movimiento. Cada alternativa consiste en un nuevo estado, un símbolo de cinta para imprimir y una dirección del movimiento de la cabeza. Nótese que a una TM no determinística no se le está permitido hacer un movimiento en el cual el siguiente estado sea seleccionado de una alternativa, y el símbolo impreso y/o la dirección del movimiento de la cabeza sean tomados de las otras alternativas. La TM no determinística

acepta su entrada si cualquier sucesión de alternativas de movimiento lleva a un estado de aceptación.

Como sucede con el autómata finito, la adición del no determinismo a la máquina de Turing no le permite al dispositivo aceptar nuevos lenguajes. De hecho, la combinación del no determinismo con cualquiera de las extensiones presentadas o que serán presentadas, como las TMs de cinta infinita de dos direcciones o las multicintas, no nos proporciona un poder adicional. Dejamos tales resultados como ejercicio y demostraremos solamente el resultado básico que se refiere a la simulación de una TM no determinística mediante una determinística.

**Teorema 7.3** Si  $L$  es aceptado por una máquina de Turing no determinística,  $M_1$ , entonces  $L$  es aceptado por alguna máquina de Turing determinística,  $M_2$ .

*Demostración* Para cualesquier estado y símbolo de cinta de  $M_1$ , existe un número finito de alternativas para el movimiento siguiente. Tales alternativas pueden numerarse 1, 2, .... Sea  $r$  el número máximo de alternativas para cualquier par de símbolos de estados de cinta. Entonces cualquier secuencia finita de alternativas puede ser representada por una sucesión de dígitos del 1 hasta  $r$ . No todas estas secuencias pueden representar alternativas de movimientos, ya que, en algunas situaciones, pueden ser menores en número que  $r$ .

$M_2$  tendrá tres cintas. La primera contendrá a la entrada. En la segunda,  $M_2$  generará sucesiones de los dígitos del 1 a  $r$ , de una forma sistemática. Específicamente, las secuencias serán generadas con la más corta al principio. Las sucesiones de igual longitud son generadas en orden numérico.

Para cada sucesión generada en la cinta 2,  $M_2$  copia la entrada en la cinta 3 y entonces simula a  $M_1$  en la cinta 3, utilizando la secuencia de la cinta 2 para dictar los movimientos de  $M_1$ . Si  $M_1$  accesa un estado de aceptación,  $M_2$  también acepta. Si existe una sucesión de alternativas que llevan a la aceptación, ésta, en algún momento, será generada en la cinta 2. Cuando sea simulado,  $M_2$  aceptará. Pero si ninguna sucesión de alternativas de movimiento de  $M_1$  conduce a la aceptación,  $M_2$  tampoco aceptará.  $\square$

### Máquinas de Turing multidimensionales

Consideramos otra modificación de la máquina de Turing que no agrega más potencia: la máquina de Turing multidimensional. El dispositivo posee el usual control finito, pero la cinta consiste en un arreglo de celdas de dimensión  $k$  infinito en todas las  $2k$  direcciones, para algún número fijo  $k$ . Dependiendo del estado y del símbolo barrido, el dispositivo cambia de estado, imprime un nuevo símbolo y mueve su cabeza de cinta en una de las  $2k$  direcciones, ya sea positiva o negativamente, a lo largo de uno de los  $k$  ejes. Al principio, la entrada se encuentra a lo largo de un eje, y la cabeza está en el extremo izquierdo de la entrada.

En cualquier momento, sólo un número finito de filas en cualquier dirección contienen símbolos que no son espacios en blanco. Por ejemplo, considérese la configuración de cintas de la TM de dos dimensiones que se presenta en la Fig. 7.10(a). Dibuje un rectángulo alrededor de los símbolos que no son espacios en blanco, como también se muestra en la Fig. 7.10(a). El rectángulo puede representarse hilera por

hilera en una sola cinta, como se muestra en la Fig. 7.10(b). Los \* separan filas. Una segunda pista puede utilizarse para indicar la posición de la cabeza de cinta de la TM de dos dimensiones.

Demostraremos que una TM de una dimensión puede simular una TM de dos dimensiones, y dejaremos la generalización a más de dos dimensiones como un ejercicio.

**Teorema 7.4** Si  $L$  es aceptado por una TM de dos dimensiones,  $M_2$ , entonces  $L$  es aceptado por una TM de una dimensión,  $M_1$ .

$B$	$B$	$B$	$a_1$	$B$	$B$	$B$
$B$	$B$	$a_2$	$a_3$	$a_4$	$a_5$	$B$
$a_6$	$a_7$	$a_8$	$a_9$	$B$	$a_{10}$	$B$
$B$	$a_{11}$	$a_{12}$	$a_{13}$	$B$	$a_{14}$	$a_{15}$
$B$	$B$	$a_{16}$	$a_{17}$	$B$	$B$	$B$

\*\*BBB $a_1$  BBB\*BB $a_2$   $a_3$   $a_4$   $a_5$  B\* $a_6$   $a_7$   $a_8$   $a_9$  Ba<sub>10</sub> B\*Ba<sub>11</sub> a<sub>12</sub> a<sub>13</sub> Ba<sub>14</sub> a<sub>15</sub>\*BB $a_{16}$  a<sub>17</sub> BBB\*\*

Fig. 7.10 Simulación de dos dimensiones mediante una. a) Cinta en dos dimensiones. b) Simulación de una dimensión.

*Demostración*  $M_1$  representa la cinta de  $M_2$  como en la Fig. 7.10(b).  $M_1$  tiene también una segunda cinta que será utilizada para los fines que describiremos, y las cintas son infinitas de dos direcciones. Supóngase que  $M_2$  hace un movimiento en el cual la cabeza no sale del rectángulo ya representado por la cinta de  $M_1$ . Si el movimiento es horizontal,  $M_1$  simplemente mueve su señalador de cabeza una celda hacia la izquierda o derecha después de haber impreso un nuevo símbolo y haber cambiado el estado de  $M_2$  registrado en el control de  $M_1$ . Si el movimiento es vertical,  $M_1$  utiliza su segunda cinta para contar el número de celdas que se encuentran entre la posición de la cabeza de cinta y el \* que está a la izquierda. Entonces  $M_1$  se mueve hasta el \* de la derecha, si el movimiento es hacia abajo, o hacia el \* de la izquierda si el movimiento es hacia arriba, y coloca el señalador de la cabeza de cinta en la posición correspondiente del nuevo bloque (región comprendida entre los \*) utilizando la cuenta de la segunda cinta.

Considérese ahora la situación en la que la cabeza de  $M_2$  se sale del rectángulo representado por  $M_1$ . Si el movimiento es vertical se añade un nuevo bloque de espacios en blanco a la izquierda o a la derecha, utilizando la segunda cinta para contar la longitud corriente de los bloques. Si el movimiento es horizontal,  $M_1$  utiliza la técnica del "corrimiento" para agregar un espacio en blanco en el extremo izquierdo o derecho de cada bloque, según lo más apropiado. Nótese que los \* dobles señalan los extremos de la región utilizada para contener bloques, de modo que  $M_1$  puede decirnos cuándo ha aumentado todos los bloques. Después de haber creado el espacio para hacer movimientos,  $M_1$  simula el movimiento de  $M_2$  como se describió más arriba.  $\square$

## Máquinas de Turing de varias cabezas

Una máquina de Turing de  $k$  cabezas tiene algún número fijo,  $k$ , de cabezas. Las cabezas están numeradas de 1 hasta  $k$ , y un movimiento de la TM depende del estado y del símbolo que está leyendo cada cabeza. En un movimiento, las cabezas pueden moverse independientemente hacia la izquierda o la derecha, o permanecer estacionarias.

**Teorema 7.5** Si  $L$  es aceptado por alguna TM de  $k$  cabezas  $M_1$ , entonces es aceptado por una TM de una sola cabeza  $M_2$ .

*Demostración* La demostración es parecida a la del Teorema 7.2 para TMs multicintas.  $M_2$  tiene  $k + 1$  registros en su cinta; el último contiene a la cinta de  $M_1$ , y la  $i$ -ésima contiene un señalador que indica la posición de la cabeza de  $i$ -ésimas cinta, para  $1 \leq i \leq k$ . Los detalles se dejan como ejercicio.  $\square$

## Máquinas de Turing fuera de línea

Una máquina de Turing *fueras de línea* es una TM multicintas cuya cinta de entrada es sólo de lectura. Por lo general rodeamos la entrada con señaladores de extremo,  $\$$  en la izquierda y  $\$$  en la derecha. La máquina de Turing no puede mover la cabeza de la cinta de entrada fuera de la región delimitada por  $\$$  y  $\$$ . Debe resultar obvio que la TM fuera de línea es precisamente un caso especial de la TM multicinta y, por consiguiente, no es más poderosa que cualquiera de los modelos que hemos considerado. De manera inversa, una TM fuera de línea puede simular a cualquiera TM  $M$  mediante el uso de una cinta más que  $M$ . La primera cosa que una TM fuera de línea hace es copiar su propia entrada en la cinta extra y entonces simula a  $M$  como si la cinta extra fuera la entrada de  $M$ . La necesidad de las TM fuera de línea se hará evidente en el Capítulo 12, cuando consideremos la limitación de la cantidad de espacio de almacenamiento a menos de la longitud de entrada.

## 7.6 HIPOTESIS DE CHURCH

La hipótesis de que el concepto intuitivo de “función calculable” puede identificarse con la clase de las funciones parcialmente recursivas se conoce como *hipótesis de Church* o como la *tesis de Church-Turing*. Mientras que no podemos esperar una “demostración” de la hipótesis de Church, puesto que el concepto informal de “calculable” sigue siendo un concepto informal, podemos proporcionar evidencia de su carácter de razonable. Ya que nuestra noción intuitiva de “calculable” no pone un límite en el número de pasos o la cantidad de almacenamiento, parecería que las funciones parcialmente recursivas son intuitivamente calculables, aunque alguien podría argumentar que una función no es “calculable” a menos que podamos limitar el cálculo por adelantado o al menos establecer si el cálculo termina o no en algún momento.

Lo que resulta menos claro es si la clase de funciones parcialmente recursivas incluye a todas las funciones “calculables”. Los teóricos de la lógica han presentado muchos otros formalismos como el cálculo  $\lambda$ , sistemas Post y funciones generales

recursivas. Todos estos formalismos han definido la misma clase de funciones, es decir, las funciones parcialmente recursivas. Además, los modelos de computadora abstractos, como la *máquina de acceso directo* (RAM), también dan lugar a la funciones parcialmente recursivas.

La RAM consiste en un número infinito de palabras de memoria, numeradas 0, 1, ..., cada una de las cuales puede contener cualquier entero, y un número finito de registros aritméticos capaces de contener a cualquier entero. Los enteros pueden ser decodificados para aparecer en el tipo usual de instrucciones de computadora. No definiremos el modelo RAM de manera más formal, pero debe quedar claro que si escogemos un conjunto apropiado de instrucciones, la RAM puede simular a cualquier computadora existente. La demostración de que el formalismo de la máquina de Turing es tan poderoso como el formalismo de la RAM se da a continuación. En los ejercicios se discuten algunos otros formalismos.

## Simulación de las máquinas de acceso directo mediante máquinas de Turing

**Teorema 7.6** Una máquina de Turing puede simular a una RAM, siempre y cuando las instrucciones RAM elementales puedan ellas mismas ser simuladas por una TM.

*Demostración* Utilizamos una TM multicintas  $M$  para llevar a cabo la simulación. Una cinta de  $M$  contiene a las palabras de la RAM a las que se les ha dado valores. La cinta se ve de la manera siguiente

$$\# 0 * v_0 \# 1 * v_1 \# 10 * v_2 \# \cdots \# i * v_i \# \cdots,$$

en donde  $v_i$  es el contenido, en binario, de la  $i$ -ésima palabra. En todo momento existirá algún número finito de palabras de la RAM que han sido utilizadas, y  $M$  solamente necesita mantener un registro de los valores hasta la palabra con el número más grande que hasta ese momento se ha utilizado.

La RAM tiene algún número finito de registros aritméticos.  $M$  utiliza una cinta para guardar el contenido de cada registro, una cinta para almacenar el *contador de posición*, que contiene el número de la palabra de la cual se tomará la siguiente instrucción, y una cinta como *un registro de la dirección de memoria* sobre la cual se puede colocar el número de una palabra de memoria.

Supóngase que los primeros 10 bits de una instrucción representan una de las operaciones estándar de una computadora, como LOAD, STORE, ADD, etcétera, y que los bits restantes representan la dirección de un operando. Como no discutiremos los detalles de la instrumentación para todas las instrucciones estándar de una computadora, un ejemplo bastará para poner en claro las técnicas. Supóngase que la posición de la cinta contadora de  $M$  contiene al número  $i$  en binario.  $M$  examina su primera cinta a partir de la izquierda, buscando  $#i *$ . Si encuentra un espacio en blanco antes de hallar  $#i *$ , no existe ninguna instrucción en la palabra  $i$ , de modo que la RAM y  $M$  se detienen. Si se encuentra un  $#i *$ , los bits que siguen después del  $*$  hasta el siguiente  $#$  se examinan. Supóngase que los primeros 10 bits son el código para la instrucción “ADD al registro 2”, y los restantes son algún número  $j$  en binario.  $M$  añade 1 a  $i$  en la posición de la cinta contadora y copia a  $j$  en la cinta de dirección de memoria.

Entonces busca  $\#j *$  en la primera cinta, comenzando de nuevo desde la izquierda (nótese que  $\#0*$  señala el extremo izquierdo). Si no encuentra  $a \#j *$ , suponemos que la palabra  $j$  contiene a 0 y pasamos a la siguiente instrucción de la RAM. Si se encuentra con  $\#j *v\#$ ,  $v$ , se agrega al contenido del registro 2, que está almacenado en su propia cinta. Entonces repetimos el ciclo con la siguiente instrucción.

Obsérvese que aunque la simulación de la RAM utilizó una máquina de Turing multicintas, según el Teorema 7.2, sería suficiente una TM de una sola cinta, aunque la simulación sería más complicada.  $\square$

## 7.7 MAQUINAS DE TURING COMO ENUMERADORES

Hemos visto a las máquinas de Turing como reconocedoras de lenguajes y como calculadoras de funciones sobre los enteros no negativos. Existe una tercera forma útil de considerar a las máquinas de Turing: como dispositivos generadores. Considérese una TM multicintas  $M$  que utiliza una cinta como una *cinta de salida*, en la cual una vez escrito un símbolo, nunca puede ser cambiado, y cuya cabeza de cinta nunca se mueve hacia la izquierda. Supóngase también que sobre la cinta de salida  $M$  escribe cadenas basadas en algún alfabeto  $\Sigma$ , separadas por el símbolo señalador  $\#$ . Podemos definir a  $G(M)$ , el *lenguaje generado por  $M$* , como el conjunto de las  $w$  que están en  $\Sigma^*$ , tales que  $w$  es impresa, en algún momento, entre una pareja de  $\#$ s sobre la cinta de salida.

Nótese que, a menos que  $M$  corra eternamente,  $G(M)$  es finito. También, que no se necesitan que las palabras sean generadas en algún orden específico, o que cualquier palabra en particular sea generada solamente una vez. Si  $L$  es  $G(M)$  para alguna TM  $M$ , entonces  $L$  es un conjunto r.e., y viceversa. Los conjuntos recursivos también tienen una caracterización en términos de generadores; éstos son exactamente los lenguajes cuyas palabras pueden ser generadas en orden creciente, con respecto al tamaño. Tales equivalencias serán demostradas en su oportunidad.

### Caracterización de los conjuntos r.e. mediante generadores

**Lema 7.1** Si  $L$  es  $G(M_1)$  para alguna TM  $M_1$ , entonces  $L$  es un conjunto r.e.

*Demostración* Constrúyase la TM  $M_2$  que contenga una cinta más que  $M_1$ .  $M_2$  simula a  $M_1$  utilizando todas las cintas excepto la de entrada de  $M_2$ . Siempre que  $M_1$  imprima un  $\#$  en su cinta de salida,  $M_2$  compara su entrada con la palabra que se acaba de generar. Si coinciden,  $M_2$  acepta; de otra manera  $M_2$  continúa simulando a  $M_1$ . Es claro que  $M_2$  acepta una entrada  $x$  si y sólo si  $x$  está en  $G(M_1)$ . Por consiguiente,  $L(M_2) = G(M_1)$ .  $\square$

El inverso del lema 7.1 es un poco más difícil. Supóngase que  $M_1$  es un reconocedor para algún conjunto r.e.  $L \subseteq \Sigma^*$ . Nuestro primer intento (no fructífero) de diseñar un generador para  $L$  debe ser generar las palabras de  $\Sigma^*$  en algún orden  $w_1, w_2, \dots$ , correr  $M_1$  sobre  $w_1$  y, si  $M_1$  acepta, generar  $w_1$ . Después correr  $M_1$  sobre  $w_2$ , generando  $w_2$  si  $M_1$  acepta, y así sucesivamente. Este método funciona si se garantiza que  $M_1$  se detenga en todas las entradas. Sin embargo, como veremos en el Capítulo 8, existen lenguajes  $L$  que son r.e., pero no son recursivos. Si éste es el caso, debemos tomar en cuenta la posibilidad de que  $M_1$  nunca se detenga en alguna  $w_i$ . Entonces  $M_2$  nunca considerará

a  $w_{i+1}, w_{i+2}, \dots$ , y de este modo no puede generar ninguna de estas palabras, aun si  $M_1$  las acepta. Por tanto, debemos evitar la simulación de  $M_1$  indefinidamente sobre cualquier palabra. Para hacer esto establecemos un orden para enumerar las palabras de  $\Sigma^*$ . En seguida desarrollaremos un método para generar todos los pares  $(i, j)$  de enteros positivos. La simulación continúa mediante la generación de un par  $(i, j)$  y después simulando  $M_1$  sobre la  $i$ -ésima palabra, para  $j$  pasos.

Establecemos un *orden canónico* para  $\Sigma^*$  de la manera siguiente. Lístense las palabras en orden según su tamaño, con las palabras del mismo tamaño en “orden numérico”. Esto es, hagamos  $\Sigma = \{a_0, a_1, \dots, a_{k-1}\}$ , e imaginemos que  $a_i$  es el “dígito”  $i$  en base  $k$ . Entonces las palabras de longitud  $n$  son los números que van de 0 hasta  $k^n - 1$  escritos en base  $k$ . El diseño de una TM que genere palabras en orden canónico no es difícil y lo dejamos como ejercicio.

**Ejemplo 7.9** Si  $\Sigma = \{0, 1\}$ , el orden canónico es  $\epsilon, 0, 1, 00, 01, 10, 11, 000, 001, \dots$

Nótese que el orden aparentemente más sencillo en el que generamos la representación más corta de 0, 1, 2, ... en base  $k$  no funcionará, ya que nunca generamos palabras como  $a_0a_0a_1$ , que tengan “0s al principio”.

En seguida consideraremos la generación de pares  $(i, j)$  tal que cada par sea generado después de alguna cantidad finita de tiempo. Esta tarea no es tan fácil como parece. El ingenuo planteamiento,  $(1, 1), (1, 2), (1, 3), \dots$ , nunca generará pares en los que  $i > 1$ . En lugar de hacer esto, generaremos pares según el orden de la suma  $i + j$ , y entre pares de igual suma, en orden creciente de  $i$ . Esto es, generamos  $(1, 1), (1, 2), (2, 1), (1, 3), (2, 2), (3, 1), (1, 4), \dots$  El par  $(i, j)$  es el  $[(i+j-1)(i+j-2)/2 + i]$ -ésimo par generado. Así pues este ordenamiento tiene la propiedad deseada que consiste en la existencia de un tiempo finito en el que cualquier par específico  $(i, j)$  es generado.

Es fácil diseñar una TM que genere pares  $(i, j)$  con este orden en binario, y dejamos su construcción al lector. En adelante nos referiremos a la máquina con estas características como *generador de pares*. A propósito, el ordenamiento utilizado por el generador de pares demuestra que los pares de enteros pueden ponerse en correspondencia uno a uno con los enteros mismos, un resultado, al parecer, paradójico descubierto por Georg Kentor cuando demostró que los racionales (que en realidad son el cociente de dos enteros) son de igual número que los enteros.

**Teorema 7.7** Un lenguaje es r.e., si y sólo si es  $G(M_2)$  para alguna TM  $M_2$ .

*Demostración* Tomando en cuenta el Lema 7.1 sólo tenemos que mostrar cómo un conjunto r.e.  $L = L(M_1)$  puede ser generado por una TM  $M_2$ .  $M_2$  simula al generador de pares. Cuando  $(i, j)$  es generado,  $M_2$  produce la  $i$ -ésima palabra  $w_i$  en orden canónico y simula a  $M_1$  sobre  $w_i$  para  $j$  pasos. Si  $M_1$  acepta en el  $i$ -ésimo paso (contando a la ID inicial como paso 1), entonces  $M_2$  genera a  $w_i$ .

Seguramente  $M_2$  no genera palabras que no están en  $L$ . Si  $w$  se encuentra en  $L$ , hagamos  $w$  la  $i$ -ésima palabra en orden canónico para el alfabeto de  $L$ , y que  $M_1$  acepte a  $w$  después de exactamente  $j$  movimientos. Como sólo le toma a  $M_2$  una cantidad finita de tiempo para generar cualquier palabra particular en orden canónico o para simular

$M_1$  en cualquier número de pasos, sabemos que  $M_2$  en algún momento producirá al par  $(i, j)$ . En esa etapa,  $w$  será generada por  $M_2$ . Por consiguiente  $G(M_2) = L$ .  $\square$

**Corolario** Si  $L$  es un conjunto r.e., entonces existe un generador para  $L$  que enumera cada palabra en  $L$  exactamente una vez.

**Demostración** La  $M_2$  descrita más arriba tiene dicha propiedad, ya que genera a  $w_i$  solamente cuando está considerando al par  $(i, j)$ , en donde  $j$  es exactamente el número de pasos que le toma a  $M_1$  para aceptar a  $w_i$ .  $\square$

### Caracterización de los conjuntos recursivos mediante generadores

Mostraremos, ahora, que los conjuntos recursivos son precisamente aquellos cuyas palabras pueden ser generadas en orden canónico.

**Lema 7.2** Si  $L$  es recursiva, entonces existe un generador para  $L$  que imprime las palabras de  $L$  en orden canónico y no imprime otras palabras.

**Demostración** Sea  $L = L(M_1) \subseteq \Sigma^*$ , en donde  $M_1$  se detiene en cada entrada. Constrúyase  $M_2$  que genere a  $L$  de la manera siguiente:  $M_2$  genera (en una cinta de paso las palabras de  $\Sigma^*$ , una a la vez, en orden canónico. Después de generar alguna palabra  $w$ ,  $M_2$  simula a  $M_1$  sobre  $w$ . Si  $M_1$  acepta a  $w$ ,  $M_2$  genera a  $w$ . Ya que está garantizado que  $M_1$  se detendrá, sabemos que  $M_2$  terminará procesando cada palabra después de un tiempo finito  $y$ , por consiguiente, considerará en algún momento cada palabra particular en  $\Sigma^*$ . Es claro que  $M_2$  genera a  $L$  en orden canónico.  $\square$

El inverso del Lema 7.2, es decir, si  $L$  puede ser generado en orden canónico, entonces  $L$  es recursivo, también es válido. Sin embargo, existe una sutileza que no debemos olvidar. En el Lema 7.2 podríamos en realidad construir  $M_2$  a partir de  $M_1$ . Sin embargo, dada una TM  $M$  que genera a  $L$  en orden canónico, sabemos que existe una TM que se detiene y que reconoce a  $L$ , pero no existe un algoritmo que exhiba a esa TM.

Supóngase que  $M_1$  genera a  $L$  en orden canónico. Lo que se puede hacer, de manera natural, es construir una TM  $M_2$  que, sobre la entrada  $w$ , simule a  $M_1$ , hasta que  $M_1$  genere a  $w$  o a una palabra que esté más allá de  $w$  en orden canónico. En el primer caso,  $M_2$  acepta a  $w$  y en el otro  $M_2$  se detiene sin aceptar a  $w$ . Sin embargo si  $L$  es finita,  $M_1$  puede no detenerse nunca después de generar la última palabra de  $L$ , de modo que  $M_1$  no genere a  $w$  ni a ninguna palabra más allá de  $w$ . En esta situación  $M_2$  no se detendría. Este problema se presenta solamente cuando  $L$  es finito, aun cuando sabemos que cada conjunto finito es aceptado por una máquina de Turing que se detiene en todas las entradas. Desafortunadamente, no podemos determinar cuando una TM genera un conjunto finito o, si éste lo es, de qué conjunto finito se trata. Por consiguiente, sabemos que una máquina de Turing que se detiene y acepta a  $L$ , y con lenguaje generado por  $M_1$ , siempre existe, pero no existe un algoritmo que exhiba a dicha máquina de Turing.  $\square$

**Teorema 7.8**  $L$  es recursivo si y sólo si  $L$  es generado en orden canónico.

**Demostración** La parte “sólo si” fue establecida por el Lema 7.2. Para la parte “si”,

cuando  $L$  es infinito, la  $M_2$  descrita arriba es una máquina de Turing que se detiene para  $L$ . Es claro que cuando  $L$  es finito, existe un autómata finito que acepta a  $L$  y, en consecuencia,  $L$  puede ser aceptado por una TM que se detiene en todas las entradas. Nótese que, en general, no podemos presentar una TM en particular que se detenga y que acepte a  $L$ ; sin embargo, el teorema solamente establece que una TM con tales características existe.  $\square$

## 7.8 MAQUINAS DE TURING RESTRINGIDAS EQUIVALENTES AL MODELO BASICO

En la Sección 7.5 consideramos generalizaciones del modelo TM básico. Como ya hemos visto, estas generalizaciones no tienen más poder computacional que el modelo básico. Concluimos este capítulo considerando algunos modelos que en un principio parecen menos poderosos que la TM, pero que de hecho son igual de poderosos. En su mayor parte, estos modelos serán variaciones del autómata de pila que definimos en el Capítulo 5.

De paso, hacemos notar que un autómata de pila es equivalente a una TM no determinística con una entrada de lectura solamente, en la que la cabeza de entrada no puede moverse hacia la izquierda, más una cinta de almacenamiento que posee una restricción más bien peculiar en la cabeza de cinta. Siempre que la cabeza de la cinta de almacenamiento se mueve hacia la izquierda, debe dejar un espacio en blanco. Así, pues, la cinta de almacenamiento, a la derecha de la cabeza, siempre consistirá en espacios en blanco por completo. La cinta de almacenamiento efectivamente es una pila, con el tope a la derecha, y no a la izquierda como las que se presentaron en el Capítulo 5.

### Máquinas de varias pilas

Una *máquina determinística de dos pilas* es una máquina de Turing con una entrada de lectura solamente y dos cintas de almacenamiento. Si una cabeza se mueve hacia la izquierda en cualquiera de las cintas, un espacio en blanco se deja en esa cinta.

**Lema 7.3** Una máquina de Turing de una sola cinta cualquiera puede ser simulada por una máquina determinística de dos pilas.

**Demostración** Los símbolos que se encuentran a la izquierda de la cabeza de la TM que está siendo simulada pueden almacenarse en una pila, mientras que los símbolos que están a la derecha de la cabeza pueden colocarse en la otra pila. En cada pila, los símbolos que están más cerca de la cabeza de la TM son colocados más cerca del tope de la pila que los símbolos que se encuentran más lejos de la cabeza de la TM.  $\square$

### Máquinas contadoras

Podemos demostrar un resultado que es más fuerte que el Lema 7.3. Este se refiere a las *máquinas contadoras*, que son máquinas de Turing fuera de línea cuyas cintas de

almacenamiento son semiinfinitas, y cuyos alfabetos de cinta contienen sólo dos símbolos,  $Z$  y  $B$  (espacio en blanco). Aún más, el símbolo  $Z$ , que sirve como señalador del fondo de la pila, aparece inicialmente en la celda barrida por la cabeza de cinta y puede nunca aparecer en cualquier otra celda. Se puede almacenar un entero  $i$  mediante el movimiento de la cabeza de cinta  $i$  celdas hacia la derecha de  $Z$ . Un número almacenado puede aumentarse o disminuirse moviendo la cabeza de cinta hacia la derecha o hacia la izquierda. Podemos probar si un número es cero verificando si  $Z$  es barrida por la cabeza, pero no podemos probar directamente si dos números son iguales.

En la Fig. 7.11 se presenta un ejemplo de una máquina contadora; es costumbre utilizar  $\emptyset$  y  $\$$  como señaladores de extremos en la entrada. Aquí  $Z$  es el símbolo que no está en blanco en cada cinta. Una descripción instantánea de una máquina contadora puede describirse mediante el estado, el contenido de la cinta de entrada, la posición de la cabeza de entrada y la distancia de las cabezas de almacenamiento al símbolo  $Z$  (que aquí se muestran como  $d_1$  y  $d_2$ ). Llamamos a estas distancias las cuentas en las cintas. La máquina contadora, entonces, puede realmente almacenar sólo unas cuentas en cada cinta y decir si esa cuenta es cero.

**Lema 7.4.** Una máquina de cuatro contadores puede simular a una máquina de Turing cualquiera.

*Demostración* Según el Lema 7.3 es suficiente mostrar que dos cintas contadoras pueden simular una pila. Hagamos que una pila tenga  $k-1$  símbolos de cinta,  $Z_1, Z_2, \dots, Z_{k-1}$ . Entonces podemos

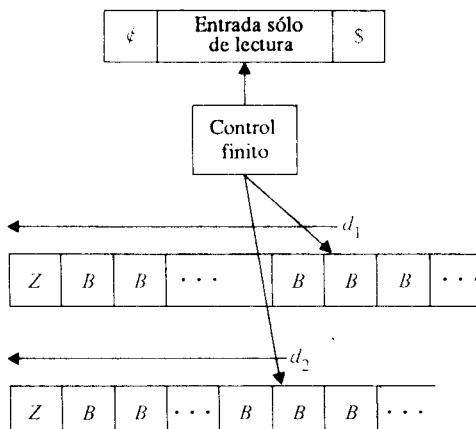


Fig. 7.11 Máquina contadora.

representar a la pila  $Z_{i_1}, Z_{i_2}, \dots, Z_{i_m}$  únicamente mediante la cuenta en base  $k$

$$j = i_m + k i_{m-1} + k^2 i_{m-2} + \dots + k^{m-1} i_1. \quad (7.3)$$

Nótese que no todos los enteros representan una pila; en particular, aquellos cuya representación en base  $k$  contiene al dígito 0 no representan una pila.

Supóngase que el símbolo  $Z_r$  se coloca en el tope (extremo derecho) de la pila  $Z_{i_1}, Z_{i_2}, \dots, Z_{i_m} Z_r$ . La cuenta asociada con  $Z_{i_1}, Z_{i_2}, \dots, Z_{i_m} Z_r$  es  $jk+r$ . Para obtener esta nueva cuenta, la máquina contadora mueve repetidamente la cabeza del primer contador una celda hacia la izquierda y la cabeza de la segunda  $k$  celdas a la derecha. Cuando la cabeza del primer contador alcanza el símbolo que no está en blanco, el segundo contador contiene la cuenta  $jk$ . Es una cuestión simple agregar  $r$  a la cuenta.

Si, en lugar de lo anterior, se saca el símbolo del tope  $Z_{i_m}$  de la pila,  $j$  debería ser sustituida por  $[j/k]$ , la parte entera de  $j/k$ . Disminuimos repetidamente la cuenta en el primer contador por la cantidad de  $k$  y después agregamos uno al segundo contador. Cuando la primera cuenta es cero, la segunda cuenta será  $[j/k]$ .

Para completar la descripción de la simulación, debemos mostrar cómo la máquina de cuatro contadores puede decirnos qué símbolo se encuentra en el tope de cada pila. Si la cuenta  $j$  está almacenada en un contador, la máquina de cuatro contadores puede copiar  $j$  en otro contador, calculando  $j \bmod k$  en su control finito. Adviértase que  $j \bmod k$  es  $i_m$  si  $j$  está dada por (7.3).  $\square$

**Teorema 7.9** Una máquina de dos contadores puede simular a una máquina de Turing cualquiera.

*Demostración* Según el Lema 7.4, es suficiente mostrar la forma de simular cuatro contadores con dos. Hagamos que los cuatro contadores tengan cuentas  $i, j, k$  y  $\ell$ . Un contador puede representar a estos cuatro mediante el número  $n = 2^i 3^j 5^k 7^\ell$ . Como 2, 3, 5 y 7 son números primos,  $i, j, k$  y  $\ell$  pueden recobrarse de manera única a partir de  $n$ .

Para incrementar  $i, j, k$  o  $\ell$  en 1, multiplicamos  $n$  por 2, 3, 5 o 7, respectivamente. Para hacer esto, si tenemos otro contador ajustado a cero, podemos mover la cabeza de este contador 2, 3, 5 o 7 celdas a la derecha cada vez que se mueva la cabeza del primer contador una celda hacia la izquierda. Cuando el primer contador contiene a cero, el segundo contendrá  $2n, 3n, 5n$  o  $7n$ , respectivamente. Para disminuir  $i, j, k$  o  $\ell$  en 1,  $n$  es, por un proceso similar, dividido por 2, 3, 5, o 7, respectivamente.

También debemos mostrar cómo la máquina de dos contadores puede determinar el siguiente movimiento de la máquina de cuatro contadores. La máquina de dos contadores siempre barre la misma celda de la cinta de entrada conforme la máquina de cuatro contadores funciona. El estado de la máquina de cuatro contadores es almacenado en el control finito de la máquina de dos contadores. Por consiguiente, para determinar el movimiento de la máquina de cuatro contadores, la máquina de dos sólo tiene que determinar cuál, si hay alguno, de los  $i, j, k$  y  $\ell$  son 0. Al pasar  $n$  de un contador a otro, el control finito de la máquina de dos contadores puede determinar si  $n$  es divisible entre 2, 3, 5, 7 o cualquier producto de estos números.  $\square$

### Límites en el número de estados y símbolos

Otra forma de restringir una TM es limitando el tamaño del alfabeto de cinta o el número de estados. Si el alfabeto de cinta, el número de cintas y el número de estados se limitan, entonces existe solo un número finito de máquinas de Turing diferentes, de

modo que el modelo restringido es menos poderoso que el original.<sup>†</sup> Si no restringimos el alfabeto de cinta, entonces tres estados y una cinta son suficientes para reconocer a cualquier conjunto r.e.; este resultado se deja como ejercicio. Demostraremos, sin embargo, un resultado concerniente a los alfabetos de cinta limitados.

**Teorema 7.10** Si  $L \subseteq (0+1)^*$  y  $L$  es r.e., entonces  $L$  es aceptado por una TM de una sola cinta con alfabeto de cinta  $\{0, 1, B\}$ .

**Demostración** Sea  $L = L(M_1)$ , en donde  $M_1 = (Q, \{0, 1\}, \Gamma, \delta, q_0, B, F)$ . Supóngase que  $\Gamma$  tiene entre  $2^{k-1} + 1$  y  $2^k$  símbolos, de modo que  $k$  bits son suficientes para codificar cualquier símbolo de cinta de  $M_1$ . Podemos diseñar a  $M_2$ , con alfabeto de cinta  $\{0, 1, B\}$  para simular a  $M_1$ . La cinta de  $M_2$  consistirá en una secuencia de códigos para los símbolos de  $M_1$ . El control finito de  $M_2$  recuerda al estado de  $M_1$  y también recuerda la posición de la cabeza de cinta de  $M_2$ , módulo  $k$ , de forma que  $M_2$  pueda saber cuándo está al principio de un símbolo de cinta codificado de  $M_1$ .

Al comienzo de la simulación de un movimiento de  $M_1$ , la cabeza de  $M_2$  se encuentra en el extremo izquierdo de símbolo de  $M_1$  codificado en binario.  $M_2$  barre los siguientes  $k-1$  símbolos de la derecha, para determinar el movimiento de  $M_1$ . Entonces  $M_1$  sustituye los símbolos barridos para reflejar el movimiento de  $M_1$ , coloca su cabecera de cinta en el extremo izquierdo del código del siguiente símbolo barrido por  $M_1$  y cambia el estado de  $M_1$ . Si tal estado es de aceptación,  $M_2$  acepta; de otra manera,  $M_2$  está listo para simular el siguiente movimiento de  $M_1$ . Se presenta un caso especial cuando  $M_2$  encuentra su cabeza colocada en un espacio en blanco en donde debería estar leyendo el código de un símbolo de  $M_1$ . En este caso,  $M_1$  se acaba de mover a una posición que nunca antes había alcanzado.  $M_2$  debe escribir el código binario del espacio en blanco de  $M_1$  en la celda barrida y las  $k-1$  celdas que están a su derecha, después de lo cual puede simular un movimiento de  $M_1$  como lo hizo antes.  $\square$

Queda todavía por explicar un detalle importante. La entrada de  $M_2$  es una cadena binaria  $w$  en  $(0+1)^*$  que representa a la misma  $w$ , más que una cadena de 0s y 1s codificados que representan a  $w$ . Por tanto, antes de simular a  $M_1$ ,  $M_2$  debe sustituir  $w$  por su código. Para hacer esto,  $M_2$  utiliza el truco del “corrimiento”, utilizando a  $B$  como el símbolo  $X$  descrito en la Sección 7.4, en donde se introdujo el “corrimiento”. Para cada símbolo de entrada, empezando por el que está más a la izquierda, la cadena que se encuentra a la derecha del símbolo es corrida  $k-1$  lugares hacia la derecha y, entonces, el símbolo y los  $k-1$   $B$ s introducidos se sustituyen por el código binario de  $k$  bits correspondiente al símbolo.  $\square$

Podemos aplicar la misma técnica de codificación en binario aun si el alfabeto de entrada no es  $\{0, 1\}$ . En consecuencia planteamos el siguiente corolario y dejamos su demostración como ejercicio.

<sup>†</sup> Sin embargo, existen máquinas de Turing restringidas que son “universales” (véase la Sección 8.3) en el sentido de que dado como entrada un código de una función de transición para alguna TM  $M$  y una entrada  $w$  a  $M$ , la máquina universal acepta si y sólo si  $M$  acepta a  $w$ . Por ejemplo, se sabe que existe una TM universal con una cinta, 5 estados y 7 símbolos de cinta.

**Corolario** Si  $L$  es un conjunto r.e., sobre cualquier alfabeto, entonces  $L$  es aceptado por una TM fuera de línea que tiene una sola cinta además de la entrada, y cuyo alfabeto para la entrada es  $\{0, 1, B\}$ .

**Teorema 7.11** Cada máquina de Turing puede ser simulada por una máquina de Turing fuera de línea que tenga una cinta de almacenamiento con dos símbolos, 0 (espacio en blanco) y 1. La máquina de Turing puede imprimir 0 o 1 sobre un 0, pero no puede imprimir un 0 sobre un 1.

**Demostración** Dejamos la demostración al lector. El “truco” consiste en crear IDs sucesivas de la máquina de Turing original con la cinta de la nueva máquina. Los símbolos de cinta están, por supuesto, codificados en binario. Cada ID es copiada, haciendo los cambios necesarios para simular un movimiento de la máquina vieja.

Además de la codificación binaria del símbolo original. La TM que hace la simulación necesita celdas para indicar la posición de la cabeza en la ID que se está copiando, y celdas para indicar que la representación binaria de un símbolo ya ha sido copiada.  $\square$

## EJERCICIOS

**7.1** Diseñe máquinas de Turing para reconocer los siguientes lenguajes.

- $\{0^n 1^n 0^n \mid n \geq 1\}$ .
- $\{ww^R \mid w \text{ está en } (0+1)^*\}$ .
- El conjunto de las cadenas que tienen un número igual de 0s y 1s.

**7.2** Diseñe máquinas de Turing para calcular las funciones siguientes.

- $[\log_2 n]$
- $n!$
- $n^2$ .

**7.3** Muestre que si  $L$  es aceptado por una TM no determinística de  $k$  cintas y dimensión  $\ell$ , con  $m$  cabezas por cinta, entonces  $L$  es aceptada por una TM determinística con una cinta semiinfinita y una cabeza de cinta.

**7.4** Una función recursiva es una función definida por un conjunto finito de reglas que especifican a la función, para varios argumentos, en términos de variables, constantes enteras no negativas, la función sucesor (agréguese 1), la función misma o una expresión construida a partir de éstas mediante la composición de funciones. Por ejemplo, la función de Ackermann está definida por las reglas:

- $A(0, y) = 1$
- $A(1, 0) = 2$
- $A(x, 0) = x + 2 \text{ para } x \geq 2$
- $A(x + 1, y + 1) = A(A(x, y + 1), y)$ .

a) Evalúe  $A(2, 1)$ .

\*b) ¿Qué función de una variables es  $A(x, 2)$ ?

\*c) Evalúe  $A(4, 3)$ .

**\*7.5** Dé definiciones recursivas para

- $n + m$
- $n - m$
- $nm$
- $n!$

**\*\* 7.6** Muestre que la clase de funciones recursivas es idéntica a la clase de las funciones parcialmente recursivas.

**7.7** Una función es *primitiva recursiva* si es un número finito de aplicaciones de la composición y la *recursión primitiva*<sup>†</sup> aplicadas a la constante 0, la función sucesor o una función proyección  $P_i(x_1, \dots, x_n) = x_i$ .

a) Muestre que cada función primitiva recursiva es una función totalmente recursiva.

\*\*b) Muestre que la función de Ackermann no es primitiva recursiva.

\*\*c) Muestre que sumando el operador minimización,  $\min(f(x))$ , definido como la menor  $x$  tal que  $f(x) = 0$ , se producen todas las funciones parcialmente recursivas.

**7.8** Diseñese una máquina de Turing para enumerar al conjunto  $\{0^n 1^n \mid n \geq 1\}$ .

**\*\*7.9** Muestre que cada conjunto r.e., es aceptado por una TM con sólo dos estados que no son de aceptación y uno de aceptación.

**\* 7.10** Complete la demostración del Teorema 7.11, concerniente a que los símbolos de cinta 0 (espacios en blanco) y 1, en donde no hay un 1 en el que se haya sobreimpreso un 0, son suficientes para que una TM de línea menor acepte a cualquier lenguaje r.e.

**7.11** Considere un modelo de TM fuera de línea que no puede escribir en cualquier cinta, sino que tiene tres cuentas que pueden situarse en la cinta auxiliar. Muestre que el modelo puede aceptar a cualquier lenguaje r.e.

## NOTAS BIBLIOGRAFICAS

La máquina de Turing es un invento de Turing [1936]. Formulaciones alternativas pueden encontrarse en Kleene [1936], Church [1936], o Post [1936]. Para una discusión de la hipótesis de Church, véase Kleene [1952], Davis [1958] o Rogers [1967]. Otros formalismos equivalentes a las funciones parcialmente recursivas incluyen al cálculo  $\lambda$  (Church [1941]), funciones recursivas (Kleene [1952]) y los sistemas de Post (Post [1943]).

Las TMs de línea menor son discutidas por Hartmanis, Lewis y Stearns [1965]. Un resultado importante con respecto a las TMs de varias cabezas (el que se refiere a que pueden ser simulados sin pérdida de tiempo con una cabeza por cinta) se encuentra en Hartmanis y Stearns [1965]. El caso que no fue cubierto por el último artículo, cuando la máquina de varias cabezas corre en tiempo *real* (un número de movimientos proporcional a la longitud de la entrada), fue tratado por Fischer, Meyer y Rosenberg [1972] y Leong y Seiferas [1977]; contiene lo más reciente sobre reducción de la complejidad de esa construcción. Las RAMs fueron consideradas de manera formal por Cook y Recknow [1973].

<sup>†</sup> Una recursión primitiva es una definición de  $f(x_1, \dots, x_n)$  mediante

$$\begin{aligned} f(x_1, \dots, x_n) &= \text{if } x_n = 0 \text{ then} \\ &\quad g(x_1, \dots, x_{n-1}) \\ &\text{else} \\ &\quad h(x_1, \dots, x_n, f(x_1, \dots, x_{n-1}, x_n - 1)) \end{aligned}$$

en donde  $g$  y  $h$  son funciones primitivas recursivas.

El Teorema 7.9, que se refiere a que dos contadores pueden simular a una TM, fue demostrado por Minsky [1961]; la demostración que se dio aquí fue tomada de Fischer [1966]. El Teorema 7.11, sobre TMs que solamente pueden imprimir 1s sobre 0s, se tomó de Wang [1957]. El Ejercicio 7.9, sobre la limitación del número de estados, fue tomado de Shannon [1956]. De hecho, como el último artículo supone la aceptación mediante una detención, más que mediante el acceso de un estado final, muestra que sólo se necesitan dos estados.

Diferentes textos proporcionan una introducción a la teoría de las máquinas de Turing y las funciones recursivas. Entre éstos se encuentran Davis [1958, 1965], Rogers [1967], Yasuhara [1971], Jones [1973], Brainerd y Landweber [1974], Hennie [1977] y Machtey y Young [1978].

# 8

## IRRESOLUBILIDAD

Consideraremos ahora las clases de los lenguajes recursivos y recursivamente enumerables. El aspecto más interesante de este estudio concierne a los lenguajes cuyas cadenas son interpretadas como codificaciones de ejemplos de problemas. Considérese el problema que consiste en determinar si una máquina de Turing cualquiera acepta a una cadena vacía. Este problema puede formularse como un problema de lenguaje mediante la codificación de las TMs como cadenas de 0s y 1s. El conjunto de todas las cadenas que codifican a las TMs que aceptan a la cadena vacía es un lenguaje que es recursivamente enumerable pero no recursivo. De lo anterior concluimos que no puede existir un algoritmo para decidir cuáles TMs aceptan a la cadena vacía y cuáles no.

En este capítulo mostraremos que muchas interrogantes acerca de las TMs, así como algunos concernientes a los lenguajes libres de contexto y otros formalismos, no tienen algoritmo para su respuesta. Además, introduciremos algunos conceptos fundamentales de la teoría de las funciones recursivas, incluyendo la jerarquía de los problemas inducidos por la consideración de máquinas de Turing con “óráculos”.

### 8.1 PROBLEMAS

De manera informal utilizamos la palabra *problema* para referirnos a una interrogante como podría ser: “¿Es ambigua una CFG dada?” En el caso del *problema de la ambigüedad*, expresado más arriba, una instancia del problema es una CFG particular. En general una *instancia* de un problema es una lista de argumentos, cada uno de éstos para cada parámetro del problema. Si restringimos nuestra atención a problemas con

respuesta si-no y a instancias codificadas del problema mediante cadenas sobre algún alfabeto finito, podemos transformar la cuestión de la existencia de un algoritmo para resolver un problema, en la cuestión de si un lenguaje en particular es recursivo o no. Podría parecer que estamos desecharando muchos problemas importantes al considerar solamente problemas con respuestas si-no; pero, de hecho, éste no es el caso. Muchos de los problemas generales tienen versiones si-no cuya demostración es tan difícil como la del problema general.

Considérese el problema de la ambigüedad para las CFGs. Llámese a la versión si-no AMB. Una versión más general del problema, llamada FIND, requiere la producción de una palabra con dos o más análisis gramaticales si una existe y la respuesta "no" en cualquier otro caso. Se puede utilizar un algoritmo para FIND para resolver a AMB. Si FIND produce una palabra  $w$ , entonces la respuesta es "si"; si FIND responde "no", entonces la respuesta es "no". De manera inversa, dado un algoritmo para AMB podemos producir un algoritmo para FIND. El algoritmo aplica primero AMB a la gramática  $G$ . Si AMB contesta "no" nuestro algoritmo responde "no". Si AMB contesta "si", el algoritmo comienza sistemáticamente a generar todas las palabras sobre el alfabeto terminal de  $G$ . Tan pronto como una palabra  $w$  es generada, ésta se prueba para ver si tiene dos o más árboles de análisis gramatical. Nótese que el algoritmo no comienza generando palabras a menos que  $G$  sea ambigua, de modo que alguna palabra  $w$ , finalmente, será encontrada e impresa. Por consiguiente tenemos, de hecho, un algoritmo. La parte del algoritmo que prueba  $w$  para ver si tiene dos o más derivaciones se deja como ejercicio.

El proceso mediante el cual construimos un algoritmo para un problema (como FIND), utilizando un supuesto algoritmo por otro (AMB), se conoce como *reducción* (de FIND a AMB). En general, cuando reducimos el problema  $A$  al problema  $B$  estamos mostrando que  $B$  es al menos tan difícil como  $A$ . Por consiguiente, en este caso como en muchos otros, el problema si-no AMB no es más fácil que la versión más general del problema. Más adelante mostraremos que no existe un algoritmo para AMB. Mediante la reducción de AMB a FIND concluimos que tampoco existe un algoritmo para FIND, ya que la existencia de un algoritmo para FIND implica la existencia de un algoritmo para AMB, lo que constituye una contradicción.

Veamos un punto más instructivo concerniente a la codificación de la gramática  $G$ . Como todas las máquinas de Turing tienen un alfabeto fijo, no podemos tomar a la notación de cuatro parámetros,  $G = (V, T, P, S)$ , como la codificación de  $G$  sin modificación. Podemos codificar conjuntos de cuatro parámetros como cadenas binarias de la manera siguiente. Hagamos que los metasímbolos del conjunto, es decir, los paréntesis izquierdo y derecho, llaves, comas y  $\rightarrow$ , sean codificados por 1, 10, 100, ...,  $10^5$ , respectivamente. Codifiquemos el  $i$ -ésimo símbolo de gramática (en cualquier orden escogido) mediante  $10^{i+5}$ . En esta codificación no podemos decir los símbolos exactos que se utilizaron para los terminales y los no terminales. Por supuesto, que si renombramos a los no terminales no afectamos al lenguaje generado, de modo que sus símbolos no son de importancia. Aunque por lo general consideramos a las identidades de las terminales como importantes, en este problema el símbolo real que se utiliza para los terminales no es pertinente, ya que el renombrar a los terminales no afecta a la ambigüedad o inambigüedad de una gramática.

## Problemas resolubles e irresolubles

Se dice que un problema cuyo lenguaje es recursivo es *resoluble*. De lo contrario, el problema es *irresoluble*. Esto es, un problema es irresoluble si no existe un algoritmo que tome como una entrada una instancia del problema y determine si la respuesta a tal instancia es "si" o "no".

Una consecuencia, que no es intuitiva, de la definición de "irresoluble" es que los problemas que sólo tienen una instancia son trivialmente resolubles. Considérese el siguiente problema basado en la conjetura de Fermat. ¿No existe solución en enteros positivos a la ecuación  $x^i + y^i = z^i$  si  $i \geq 3$ ? Nótese que  $x, y, z$  e  $i$  no son parámetros sino variables acotadas en el planteamiento del problema. Existe una máquina de Turing que acepta a cualquier entrada y otra que la rechaza. Una de éstas responde a la conjetura de Fermat de manera correcta, si bien no sabemos cuál de ellas. De hecho, ni siquiera puede existir una resolución de la conjetura utilizando los axiomas de la aritmética. Esto es, la conjetura de Fermat puede ser verdadera, aunque no existe una demostración aritmética del hecho. La posibilidad (aunque no la certeza) de que éste sea el caso surge del teorema de inintegridad de Gödel, que establece que cualquier sistema formal consistente, lo suficientemente poderoso como para abarcar a la teoría de números, debe tener proposiciones que son verdaderas pero no demostrables dentro del sistema.

No debería preocupar al lector el hecho de que un enigma como la conjetura de Fermat sea "resoluble". La teoría de la no resolubilidad tiene que ver con la existencia o no existencia de algoritmos para resolver problemas que tengan una infinidad de instancias.

## 8.2 PROPIEDADES DE LOS LENGUAJES RECURSIVOS Y NUMERABLES DE MANERA RECURSIVA

Muchos de los teoremas que se presentan en este capítulo se demuestran mediante la reducción de un problema a otro. Estas reducciones involucran la combinación de varias máquinas de Turing para formar una máquina compuesta. El estado de la TM compuesta tiene una componente para cada máquina componente individual. De manera similar la máquina compuesta tiene cintas separadas para cada máquina individual. Los detalles de la máquina compuesta por lo general son tediosos y no aportan más claridad al asunto. Por consiguiente hemos decidido describir las construcciones de manera informal.

Dado un algoritmo (una TM que siempre se detiene), podemos permitirle a la TM compuesta que lleve a cabo una acción si el algoritmo acepta y que haga otra acción si no acepta. No podríamos hacer lo anterior si nos hubiesen dado una TM cualquiera en lugar de un algoritmo, ya que si la TM no aceptara, podría correr eternamente y la máquina compuesta nunca iniciaría la siguiente tarea. En los diagramas, una flecha que se encuentra dentro de una caja con la etiqueta "inicio" indica una señal de inicio. Las cajas que no poseen la señal de "inicio", se suponen que comienzan a operar cuando la máquina compuesta lo hace. Los algoritmos tienen dos salidas, "si" y "no", que pueden utilizarse como señales de inicio o como respuesta por la máquina compuesta. Cualquier TM tiene sólo una salida "si", que puede utilizarse con los mismos fines.

Ponemos ahora nuestra atención en algunas propiedades de cerradura básicas de las clases de conjuntos recursivos y r.e.

**Teorema 8.1** El complemento de un lenguaje recursivo es recursivo.

**Demostración** Sean  $L$  un lenguaje recursivo y  $M$  una máquina de Turing que se detiene en todas las entradas y acepta a  $L$ . Construimos  $M'$  a partir de  $M$  de tal forma que si  $M$  accesa un estado final en la entrada  $w$ , entonces  $M'$  se detiene sin aceptar. Si  $M$  se detiene sin aceptar,  $M'$  accesa un estado final. Puesto que uno de estos dos eventos

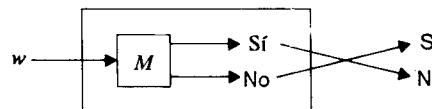


Fig. 8.1 Construcción que muestra que los lenguajes recursivos son cerrados con respecto a la complementación.

ocurre,  $M'$  es un algoritmo. Es claro que  $L(M')$  es el complemento de  $L$  y en consecuencia el complemento de  $L$  es un lenguaje recursivo. La Fig. 8.1 representa la construcción de  $M'$  a partir de  $M$ .

**Teorema 8.2** La unión de dos lenguajes recursivos es recursiva. La unión de dos lenguajes enumerables de manera recursiva es enumerable de manera recursiva.

**Demostración** Sean  $L_1$  y  $L_2$  lenguajes recursivos aceptados por los algoritmos  $M_1$  y  $M_2$ . Construimos  $M$ , que primero simula a  $M_1$ . Si  $M_1$  acepta, entonces  $M$  acepta. Si  $M_1$  rechaza, entonces  $M$  simula a  $M_2$  y acepta si y sólo si  $M_2$  acepta. Ya que tanto  $M_1$  como  $M_2$  son algoritmos, está garantizado que  $M$  se detenga. Claramente  $M$  acepta a  $L_1 \cup L_2$ .

Para los lenguajes enumerables de manera recursiva la anterior construcción no funciona, puesto que  $M_1$  puede no detenerse. En su lugar,  $M$  puede simular de manera simultánea a  $M_1$  y a  $M_2$  en cintas separadas. Si alguna acepta, entonces  $M$  acepta. La Fig. 8.2 muestra las dos construcciones del teorema.

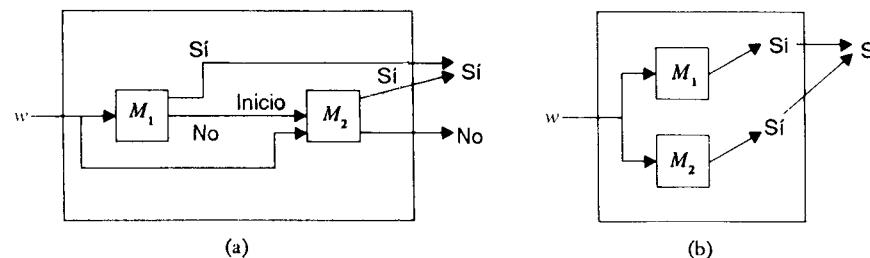


Fig. 8.2 Construcción para la unión.

**Teorema 8.3** Si un lenguaje  $L$  y su complemento  $\bar{L}$  son ambos enumerables de manera recursiva, entonces  $L$  (y en consecuencia  $\bar{L}$ ) es recursivo.

**Demostración** Sean  $M_1$  y  $M_2$  máquinas que aceptan a  $L$  y a  $\bar{L}$ , respectivamente. Considérese  $M$ , como se muestra en la Fig. 8.3, para simular al mismo tiempo a  $M_1$  y a  $M_2$ .  $M$  acepta a  $w$  si  $M_1$  acepta a  $w$  y rechaza a  $w$  si  $M_2$  la acepta. Como  $w$  está en  $L$  o en  $\bar{L}$ , sabemos que exactamente uno,  $M_1$  o  $M_2$ , aceptará. Por tanto,  $M$  siempre dirá "sí" o "no", pero nunca dirá ambos. Adviértase que no hay un límite *a priori* en el tiempo que se tomará antes de que  $M_1$  o  $M_2$  acepte, pero existe la certeza de que uno u otro lo hará. Ya que  $M$  es un algoritmo que acepta a  $L$ , se concluye que  $L$  es recursivo.  $\square$

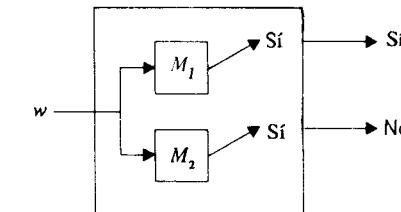


Fig. 8.3 Construcción para el Teorema 8.3.

Los Teoremas 8.1 y 8.3 tienen una importante consecuencia. Sean  $L$  y  $\bar{L}$  un par de lenguajes complementarios. Entonces sucederá una de las siguientes:

1. ambos,  $L$  y  $\bar{L}$ , son recursivos,
2. ni  $L$  ni  $\bar{L}$  son r.e., o
3. uno de los dos,  $L$  o  $\bar{L}$ , es r.e., pero no recursivo; el otro no es r.e.

Una importante técnica para mostrar un problema soluble consiste en demostrar por diagonalización que el complemento del lenguaje para ese problema no es r.e. Por tanto se debe poder aplicar el caso (2) o (3) anteriores. Esta técnica juega un papel esencial en la demostración de nuestro primer problema irresoluble.

### 8.3 MAQUINAS DE TURING UNIVERSALES Y UN PROBLEMA IRRESOLUBLE

Utilizaremos ahora la diagonalización para probar que un problema particular es irresoluble. El problema es: ¿Acepta la máquina de Turing  $M$  la entrada  $w$ ? Aquí, ambos,  $M$  y  $w$ , son parámetros del problema. Para formalizar el problema como un lenguaje restringiremos  $w$  al alfabeto  $\{0, 1\}$  y  $M$  tendrá un alfabeto de cinta  $\{0, 1, B\}$ . Como el problema restringido es irresoluble, el problema más general seguramente será irresoluble. Decidimos trabajar con la versión más restringida para simplificar la codificación de las instancias del problema como cadenas.

#### Códigos para la máquina de Turing

Para empezar, codificaremos las máquinas de Turing con alfabetos restringidos, como cadenas sobre  $\{0, 1\}$ . Sea

$$M = (Q, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$$

una máquina de Turing con alfabeto de entrada  $\{0, 1\}$  y el espacio en blanco como el único símbolo de cinta adicional. Aún más, supondremos que  $Q = \{q_1, q_2, \dots, q_n\}$  es el conjunto de estados y que  $q_2$  es el único estado final. El Teorema 7.10 nos asegura que si  $L \subseteq (0+1)^*$  es aceptado por cualquier TM, entonces es aceptado por una con alfabeto  $\{0, 1, B\}$ . También, que no hay necesidad de más de un estado final en cualquier TM, ya que una vez que ha sido aceptado, la máquina puede también detenerse.

Es conveniente llamar a los símbolos 0, 1 y B por los sinónimos  $X_1, X_2$  y  $X_3$ , respectivamente. También le asignaremos a las direcciones L y R los sinónimos  $D_1$  y  $D_2$ , respectivamente. Entonces un movimiento genérico  $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$  queda codificado por la cadena binaria

$$0^i 10^j 10^k 10^l 10^m. \quad (8.1)$$

Un código binario para la máquina de Turing  $M$  es

$$111 \text{ código}_1 11 \text{ código}_2 11 \dots 11 \text{ código}_n 111, \quad (8.2)$$

en donde cada código<sub>i</sub> es una cadena de la forma (8.1), y cada movimiento de  $M$  está codificado por uno de los códigos<sub>j</sub>. No es necesario que los movimientos se den en algún orden específico, pues cada TM en realidad tiene muchos códigos. Cualquiera de estos códigos para  $M$  será representado por  $\langle M \rangle$ .

Cada cadena binaria puede ser interpretada como el código para, a lo sumo, una TM; muchas de las cadenas binarias no constituyen código alguno para ninguna TM. Para ver que la decodificación es única, nótese que ninguna cadena de la forma (8.1) tiene dos 1s seguidos, de modo que los código<sub>j</sub> pueden encontrarse directamente. Si una cadena no comienza ni termina con tres 1s exactamente, tiene tres 1s en algún lugar que no sea el final o tiene dos pares de 1s sin que haya cinco bloques de 0s entre ellos, entonces la cadena no representa a ninguna TM.

El par constituido por  $M$  y  $w$  se representa mediante una cadena de la forma (8.2) seguida por  $w$ . Cualquiera de estas cadenas será denotada por  $\langle M, w \rangle$ .

**Ejemplo 8.1** Sea  $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$  con los movimientos:

$$\begin{aligned} \delta(q_1, 1) &= (q_3, 0, R), \\ \delta(q_3, 0) &= (q_1, 1, R), \\ \delta(q_3, 1) &= (q_2, 0, R), \\ \delta(q_3, B) &= (q_3, 1, L). \end{aligned}$$

Así pues, una cadena representada por  $\langle M, 1011 \rangle$  es

11010010001010011000101010010011

00010010010100110001000100101111011

Nótese que muchas cadenas diferentes también son códigos para el par  $\langle M, 1011 \rangle$ , y cualquiera de éstas de ser referida por la notación  $\langle M, 1011 \rangle$ .

### Un lenguaje no r.e.

Supóngase que tenemos una lista de  $(0+1)^*$  en orden canónico (véase la Sección 7.7), en donde  $w_i$  es la  $i$ -ésima palabra y  $M_j$  es la TM cuyo código, como en (8.2), es el entero

	1	2	3	4	...
1	0	1	1	0	...
2	1	1	0	0	...
3	0	0	1	0	...
4	0	1	0	1	...
:	:	:	:	:	...

Fig. 8.4 Tabla hipotética que indica la aceptación de palabras por las TMs.

$j$  escrito en binario. Imagínese una tabla infinita que diga, para todo  $i$  y  $j$ , si  $w_i$  está en  $L(M_j)$ . La Fig. 8.4 sugiere dicha tabla;<sup>†</sup> 0 significa que  $w_i$  no está en  $L(M_j)$  y 1 significa que sí lo está.

Construimos un lenguaje  $L_d$  utilizando las entradas diagonales de la tabla para determinar la membresía de  $L_d$ . Para garantizar que ninguna TM acepta a  $L_d$ , insistimos en que  $w_i$  está en  $L_d$  si y sólo si la entrada  $(i, i)$  es 0, esto es, si  $M_i$  no acepta a  $w_i$ . Supóngase que alguna TM  $M_j$  ha aceptado a  $L_d$ . En este caso nos enfrentamos con la siguiente contradicción. Si  $w_j$  está en  $L_d$ , entonces la entrada  $(j, j)$  es 0, implicando que  $w_j$  no está en  $L(M_j)$  y contradiciendo el hecho de que  $L_d = L(M_j)$ . Por otra parte, si  $w_j$  no está en  $L_d$ , entonces la entrada  $(j, j)$  es 1, lo que implica que  $w_j$  está en  $L(M_j)$ , que de nuevo contradice a  $L_d = L(M_j)$ . Como  $w_j$  está o no en  $L_d$ , concluimos que nuestra hipótesis,  $L_d = L(M_j)$ , es falsa. Por consiguiente, ninguna TM de la lista acepta a  $L_d$  y, según el Teorema 7.10, ninguna TM acepta al lenguaje  $L_d$ .

Hemos demostrado

**Lema 8.1**  $L_d$  no es r.e.

### El lenguaje universal

Definimos  $L_u$ , el “lenguaje universal”, como  $\{\langle M, w \rangle \mid M \text{ acepta a } w\}$ . Llamamos a  $L_u$  “universal” ya que la cuestión de si cualquier cadena particular  $w$  de  $(0+1)^*$  es aceptada por cualquier máquina de Turing particular  $M$  es equivalente a la cuestión de

<sup>†</sup> En realidad como todas las TM de baja numeración aceptan al conjunto vacío, la parte correcta de la tabla mostrada tiene sólo 0s.

si  $\langle M', w \rangle$  está en  $L_u$ , en donde  $M'$  es la TM con alfabeto de cinta  $\{0, 1, B\}$  equivalente a la  $M$  que se construyó en el Teorema 7.10.

**Teorema 8.4**  $L_u$  es numerable de manera recursiva.

**Demostración** Exhibiremos una TM de tres cintas  $M_1$  que acepta a  $L_u$ . La primer cinta de  $M_1$  es la cinta de entrada, y la cabeza de entrada sobre esa cinta se utiliza para buscar los movimientos de la TM  $M$  cuando se ha dado el código  $\langle M, w \rangle$  como entrada. Nótese que los movimientos de  $M$  se encuentran entre los primeros dos bloques de 1s. La segunda cinta de  $M_1$  simulará a la cinta de  $M$ . El alfabeto de  $M$  es  $\{0, 1, B\}$ , de modo que cada símbolo de la cinta de  $M$  puede estar contenido en una celda de la segunda cinta de  $M_1$ . Obsérvese que si no restringimos el alfabeto de  $M$ , tendríamos que utilizar muchas celdas de la cinta de  $M_1$  para simular una de las celdas de  $M$ , pero la simulación podría haberse llevado a cabo con un poco más de trabajo. La tercera cinta contiene el estado de  $M$ , con  $q_i$  representado por 0'. El comportamiento de  $M_1$  es el siguiente:

1. Verifica el formato de la cinta 1 para ver que posee un prefijo de la forma (8.2) y que no existen dos códigos que comiencen con  $0^i 1 0^j 1$  para las mismas  $i$  y  $j$ . También verifica si  $0^i 1 0^j 1 0^k 1 0^l 1 0^m$  es un código, entonces  $1 \leq j \leq 3$ ,  $1 \leq k \leq 3$  y  $1 \leq m \leq 2$ . La cinta 3 puede utilizarse como una cinta de registro para facilitar la comparación de los códigos.
2. Pone en funcionamiento a la cinta 2 para contener a  $w$ , la parte de la entrada que se encuentra más allá del segundo bloque de tres 1s. Pone en funcionamiento a la cinta 3 para contener a un solo 0, que representa a  $q_i$ . Las tres cabezas de cinta están colocadas en los símbolos que están más a la izquierda. Estos símbolos pueden estar señalados de modo que las cabezas puedan encontrar su camino de regreso.
3. Si la cinta 3 contiene a 00, el código para el estado final, se detiene y acepta.
4. Sea  $X_j$  el símbolo barrido por la cinta 2 y sea  $0^t$  el contenido corriente de la cinta 3. Barre la cinta 1 a partir del extremo izquierdo hasta los segundos 111, buscando una subcadena que comience con  $1 1 0^i 1 0^j 1$ . Si no se encuentra dicha cadena, se detiene y rechaza;  $M$  no tienen un siguiente movimiento y no ha aceptado. Si sí encuentra a tal código, hace que éste sea  $0^i 1 0^j 1 0^k 1 0^l 1 0^m$ . Después coloca a  $0^t$  en la cinta 3, imprime  $X_j$  en la celda de cinta barrida por la cabeza 2 y mueve dicha cabeza en la dirección  $D_m$ . Nótese que hemos verificado en (1) que  $1 \leq i \leq 3$  y  $1 \leq m \leq 2$ . Se traslada al paso (3).

Es fácil verificar que  $M_1$  acepta a  $\langle M, w \rangle$  si y sólo si  $M$  acepta a  $w$ . También es cierto que si  $M$  corre indefinidamente sobre  $w$ ,  $M_1$  también lo hará sobre  $\langle M, w \rangle$  y si  $M$  se detiene en  $w$  sin aceptar,  $M_1$  hace lo mismo sobre  $\langle M, w \rangle$ .  $\square$

La existencia de  $M_1$  es suficiente para demostrar el Teorema 8.4. Sin embargo, según los teoremas 7.2 y 7.10, podemos encontrar una TM con una cinta semiinfinita y alfabeto  $\{0, 1, B\}$  que acepta a  $L_u$ . Llamamos a esta TM particular  $M_u$ , la máquina de Turing universal, ya que hace el trabajo de cualquier TM con alfabeto de entrada 0, 1.

Por el Lema 8.1, el lenguaje diagonal  $L_d$  es no r.e., y por tanto no es recursivo. Así pues, según el Teorema 8.1,  $L_d$  no es recursivo. Nótese que  $L_d = \{w_i \mid M_i \text{ acepta a } w_i\}$ . Podemos demostrar que el lenguaje universal  $L_u = \{\langle M, w \rangle \mid M \text{ acepta a } w\}$  no es

recursivo, mediante la reducción de  $L_d$  a  $L_u$ . En consecuencia,  $L_u$  es un ejemplo de un lenguaje que es r.e., pero que no es recursivo. De hecho,  $L_d$  es otro ejemplo de este tipo de lenguaje.

**Teorema 8.5**  $L_u$  es no recursivo.

**Demostración** Supóngase que  $A$  es un algoritmo que reconoce a  $L_u$ . Entonces podríamos reconocer a  $L_d$  de la manera siguiente. Dada la cadena  $w$  de  $(0 + 1)^*$ , determinésemos mediante un cálculo sencillo el valor de  $i$  tal que  $w = w_i$ . El entero  $i$  en binario es el código para alguna TM  $M_i$ . Aliméntese  $\langle M_i, w_i \rangle$  al algoritmo  $A$  y acéptese a  $w$  si y sólo si  $M_i$  acepta a  $w_i$ . La construcción se muestra en la Fig. 8.5. Es fácil comprobar que el algoritmo construido acepta a  $w$  si y sólo si  $w = w_i$  y  $w_i$  está en  $L(M_i)$ . Por tanto tenemos un algoritmo para  $L_d$ . Como no existe tal algoritmo, sabemos que nuestra hipótesis, que el algoritmo  $A$  para  $L_u$  existe, es falsa. De aquí que  $L_u$  es r.e., pero no recursivo.  $\square$

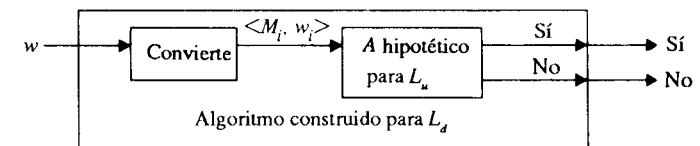


Fig. 8.5 Reducción de  $L_d$  a  $L_u$

#### 8.4 TEOREMA DE RICE Y ALGUNOS OTROS PROBLEMAS IRRESOLUBLES

Tenemos ahora un ejemplo de un lenguaje r.e. que es no recursivo. El problema asociado “¿Acepta  $M$  a  $w$ ? ” es irresoluble, y podemos utilizar este hecho para mostrar que otros problemas son irresolubles. En esta sección daremos varios ejemplos de problemas irresolubles que se refieren a conjuntos r.e. En las siguientes tres secciones discutiremos algunos problemas irresolubles tomados de fuera del dominio de las TMs.

**Ejemplo 8.2** Considere el problema: “¿Es  $L(M) \neq \emptyset$ ? ” Sea  $\langle M \rangle$  el código para  $M$  como en (8.2). Entonces definimos

$$L_{ne} = \{\langle M \rangle \mid L(M) \neq \emptyset\} \quad \text{y} \quad L_e = \{\langle M \rangle \mid L(M) = \emptyset\}.$$

Nótese que  $L_e$  y  $L_{ne}$  son complementos uno del otro, ya que cada cadena binaria denota alguna TM; aquellas cadenas con un formato erróneo representan a la TM que no tiene movimientos. Todas estas cadenas se encuentran en  $L_e$ . Decimos que  $L_{ne}$  es r.e., pero no recursivo, y que  $L_e$  no es r.e.

Mostramos que  $L_{ne}$  es r.e., mediante la construcción de una TM  $M$  que reconoce códigos de las TMs que aceptan conjuntos no vacíos. Dada la entrada  $\langle M_i \rangle$ ,  $M$  adivina de manera no determinística una cadena  $x$  aceptada por  $M_i$  y verifica que  $M_i$  efectivamente acepte a  $x$  mediante la simulación de  $M_i$  sobre la entrada  $x$ . Este paso puede también llevarse a cabo de manera determinística si utilizamos el generador de pares descrito en la Sección 7.7. Para el par  $(j, k)$  simúlese  $M_i$  sobre la  $j$ -ésima cadena binaria (en orden canónico) para  $k$  pasos. Si  $M_i$  acepta, entonces  $M$  acepta a  $\langle M_i \rangle$ .

Ahora debemos mostrar que  $L_e$  es no recursivo. Supóngase que sí lo es. Entonces podemos construir un algoritmo para  $L_u$ , violando el Teorema 8.5. Sea A un algoritmo hipotético que acepte a  $L_e$ . Existe un algoritmo B que, dado  $\langle M, w \rangle$ , construye una TM  $M'$  que acepta a  $\emptyset$  si  $M$  no acepta a  $w$  y que acepta a  $(0+1)^*$  si  $M$  acepta a  $w$ . El plano de  $M'$  se presenta en la Fig. 8.6.  $M'$  ignora su entrada  $x$  y en su lugar simula a  $M$  sobre la entrada  $w$ , aceptando si  $M$  acepta.

Nótese que  $M'$  no es B. Más bien B es como un compilador que toma  $\langle M, w \rangle$  como "programa fuente" y produce a  $M'$  como "programa objeto". Hemos descrito lo que B debe hacer, pero no la forma en que lo hace. La construcción de B es simple. Toma a  $\langle M, w \rangle$  y

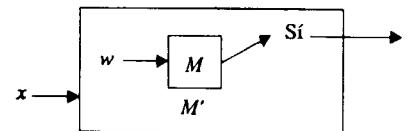


Fig. 8.6 La TM  $M'$ .

aisla a  $w$ . Digamos que  $w = a_1 a_2 \dots a_n$  es de longitud  $n$ . B crea  $n + 3$  estados  $q_1, q_2, \dots, q_{n+3}$  con movimientos

$$\begin{aligned} \delta(q_1, X) &= (q_2, \$, R) \text{ para cualquier } X \text{ (imprime el señalador)}, \\ \delta(q_i, X) &= (q_{i+1}, a_{i-1}, R) \text{ para cualquier } X \neq B \text{ y } 2 \leq i \leq n+1 \text{ (imprime a } w\text{)}, \\ \delta(q_{n+2}, X) &= (q_{n+2}, B, R) \text{ para } X \neq B \text{ (borra la cinta)}, \\ \delta(q_{n+2}, B) &= (q_{n+3}, B, L), \\ \delta(q_{n+3}, X) &= (q_{n+3}, X, L) \text{ para } X \neq \$ \text{ (encuentra al señalador)}. \end{aligned}$$

Una vez producidos los códigos para estos movimientos, B entonces añade  $n + 3$  a los índices de los estados de  $M$  e incluye el movimiento

$$\delta(q_{n+3}, \$) = (q_{n+4}, \$, R) /* \text{ comienza } M^*/$$

y todos los movimientos de  $M$  en su TM generada. La TM resultante tiene un símbolo de cinta extra  $\$$ , pero según el Teorema 7.10, podemos construir  $M'$  con alfabeto de cinta  $\{0, 1, B\}$  y, seguramente, podemos hacer  $q_2$  el estado de aceptación. Este paso completa el algoritmo B, y su salida es la deseada  $M'$  de la Fig. 8.6.

Supóngase ahora que existe el algoritmo A que acepta a  $L_e$ . Entonces construimos un algoritmo C para  $L_u$  como se muestra en la Fig. 8.7. Si  $M$  acepta a  $w$ , entonces  $L(M') \neq \emptyset$ ; de modo que A dice "no" y C dice "sí". Si  $M$  no acepta a  $w$ , entonces  $L(M') = \emptyset$ , A dice "sí" y C dice "no". Como, según el Teorema 8.5, el algoritmo C no existe, A tampoco puede existir. Por consiguiente  $L_e$  es no recursivo. Si  $L_{ne}$  fuera recursivo, por el Teorema 8.1,  $L_e$  también lo sería. Por tanto  $L_{ne}$  es r.e., pero no recursivo. Si  $L_e$  fuera r.e., entonces  $L_e$  y  $L_{ne}$ , según el Teorema 8.3, serían recursivos. Como consecuencia de lo anterior,  $L_e$  es no r.e.

Ejemplo 8.3 Considérese el lenguaje

$$L_r = \{\langle M \rangle \mid L(M) \text{ es recursivo}\}$$

y

$$L_{nr} = \{\langle M \rangle \mid L(M) \text{ es no recursivo}\}$$

Adviértase que  $L_r$  no es  $\{\langle M \rangle \mid M \text{ se detiene en todas las entradas}\}$ , aunque incluye al último lenguaje. Una TM  $M$  podría aceptar un lenguaje recursivo aún pensando que  $M$  misma podría entrar en un ciclo sin salir nunca de él con algunas palabras que no estén en  $L(M)$ ; sin embargo, alguna otra TM equivalente a  $M$  debe detenerse. Exigimos que ninguno  $L_r$  ni  $L_{nr}$  sean r.e.

Supóngase que  $L_r$  sea r.e. Entonces podríamos construir una TM para  $L_u$  que sabemos que no existe. Sea  $M_r$  una TM que acepta

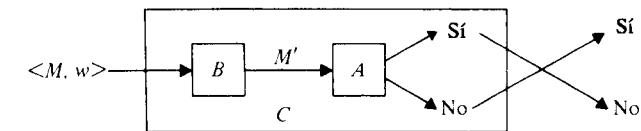


Fig. 8.7 Algoritmo construido para  $L_u$  suponiendo que el algoritmo A para  $L_e$  existe.

a  $L_r$ . Podemos construir un algoritmo A que toma a  $\langle M, w \rangle$  como entrada y produce como salida a la TM  $M'$  tal que

$$L(M') = \begin{cases} \emptyset & \text{si } M \text{ no acepta a } w, \\ L_u & \text{si } M \text{ acepta a } w. \end{cases}$$

Nótese que  $L_u$  es no recursivo, de modo que  $M'$  acepta un lenguaje recursivo si y sólo si  $M$  no acepta a  $w$ . En la Fig. 8.8 se muestra el plano de  $M'$ . Como en el ejemplo anterior, hemos descrito la salida de A. Dejamos la construcción de A al lector.

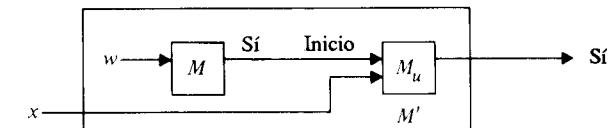


Fig. 8.8 La TM  $M'$ .

Dados A y  $M_r$ , podríamos construir una TM que acepte a  $\bar{L}_u$ , misma que se muestra en la Fig. 8.9, y se comporta de la siguiente forma: Sobre la entrada  $\langle M, w \rangle$  la TM utiliza a A para producir  $M'$ , utiliza a  $M_r$  para determinar si el conjunto aceptado por  $M'$  es recursivo, y acepta si y sólo si  $L(M')$  es recursivo. Pero  $L(M')$  es recursivo si y sólo si  $L(M') = \emptyset$ , lo cual significa que  $M$  no acepta a  $w$ . Por consiguiente la TM de la Fig. 8.9 acepta a  $\langle M, w \rangle$  si y sólo si  $\langle M, w \rangle$  está en  $\bar{L}_u$ .

Ahora pongamos nuestra atención en  $\bar{L}_u$ . Supóngase que tenemos una TM  $M_u$  que acepta a  $L_u$ . Entonces podemos utilizar  $M_u$  y un algoritmo  $B$ , que debe ser construido por el lector, para aceptar a  $\bar{L}_u$ . B toma a  $\langle M, w \rangle$  como entrada y produce como salida una TM  $M'$  tal que

$$L(M') = \begin{cases} \Sigma^* & \text{si } M \text{ acepta a } w \\ L_u & \text{si } M \text{ no acepta a } w. \end{cases}$$

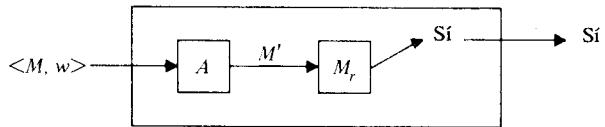


Fig. 8.9 TM hipotética para  $\bar{L}_u$

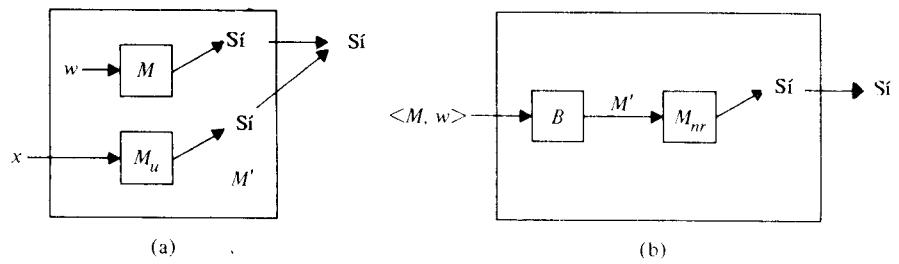


Fig. 8.10 Construcciones utilizadas en la demostración de que  $\bar{L}_u$  no es r.e. a)  $M'$ , b) TM para  $\bar{L}_u$

Por tanto  $M'$  acepta a un lenguaje recursivo si y sólo si  $M$  acepta a  $w$ .  $M'$ , que debe ser producida por  $B$ , se muestra en la Fig. 8.10(a) y en la Fig. 8.10(b) se muestra una TM que acepta a  $\bar{L}_u$  dados  $B$  y  $M_{nr}$ . La TM de la Fig. 8.10(b) acepta a  $\langle M, w \rangle$  si y sólo si  $L(M')$  es no recursivo, o de manera equivalente, si y sólo si  $M$  no acepta a  $w$ . Esto es, la TM acepta a  $\langle M, w \rangle$  si y sólo si  $\langle M, w \rangle$  está en  $\bar{L}_u$ . Puesto que ya hemos mostrado que no existe una TM así, la hipótesis de que  $M_u$  existe es falsa. Llegamos a la conclusión de que  $\bar{L}_u$  es no r.e.

### Teorema de Rice para conjuntos de índice recursivos

Los ejemplos anteriores muestran que no podemos decidir si el conjunto aceptado por una máquina de Turing está vacío o es recursivo. La técnica para demostrarlo puede utilizarse también para mostrar que no podemos decidir si el conjunto aceptado es finito, infinito, regular, libre de contexto, tiene un número par de cadenas o satisface a muchos otros predicados. ¿Qué es entonces lo que podemos decidir acerca del conjunto aceptado por una TM? Solamente los predicados triviales, como “¿Acepta la TM un conjunto r.e.?”, que son verdaderos para todas las TMs o falsos para todas ellas.

En lo que sigue discutiremos lenguajes que representan propiedades de los lenguajes r.e. Es decir, los lenguajes son conjuntos de códigos TM tales que la membresía de  $\langle M \rangle$  en el lenguaje depende solamente de  $L(M)$ , no de  $M$  misma. Más adelante consideraremos lenguajes de los códigos de TM que dependen de la TM misma, tales como “ $M$  tiene 27 estados”, que pueden ser satisfechos por algunas TMs que acepten un lenguaje dado, pero no por todas.

Sea  $\mathcal{L}$  un conjunto de lenguajes r.e., cada uno un conjunto de  $(0 + 1)^*$ . Se dice que  $\mathcal{L}$  es una *propiedad de los lenguajes r.e.* Un conjunto  $L$  tiene propiedad  $\mathcal{L}$  si  $L$  es un elemento de  $\mathcal{L}$ . Por ejemplo, la propiedad de ser infinito es  $\{L \mid L \text{ es infinito}\}$ .  $\mathcal{L}$  es una propiedad trivial si  $\mathcal{L}$  es el vacío o  $\mathcal{L}$  consiste en todos los lenguajes r.e. Sea  $L_{\mathcal{L}}$  el conjunto  $\{\langle M \rangle \mid L(M) \text{ está en } \mathcal{L}\}$ .

**Teorema 8.6.** (Teorema de Rice) Cualquier propiedad no trivial de los lenguajes r.e. es irresoluble.

*Demostración* Sin pérdida de generalidad supongamos que  $\emptyset$  no está en  $\mathcal{L}$  (de otra manera considérese  $\mathcal{L}'$ ). Ya que  $\mathcal{L}$  es no trivial, existe  $L$  con propiedad  $\mathcal{L}$ . Sea  $M_L$  una TM que acepte a  $L$ . Supóngase que  $\mathcal{L}$  es resoluble. Entonces existe un algoritmo  $M_{\mathcal{L}}$  que acepta a  $L_{\mathcal{L}}$ . Utilizamos a  $M_L$  y  $M$  para construir un algoritmo para  $L_u$  de la manera siguiente: Primero construyese un algoritmo  $A$  que toma  $\langle M, w \rangle$  como entrada y produzca a  $\langle M' \rangle$  como salida, en el que  $L(M')$  está en  $\mathcal{L}$  si y sólo si  $M$  acepta a  $w$  ( $\langle M, w \rangle$  está en  $L_u$ ).

En la Fig. 8.11 se presenta el diseño de  $M'$ . Primero  $M'$  ignora su entrada y simula a  $M$  sobre  $w$ . Si  $M$  no acepta a  $w$ , entonces  $M'$  no acepta a  $x$ . Si  $M$  acepta a  $w$ , entonces  $M'$  simula a  $M_L$  sobre  $x$ , aceptando a  $x$  si y sólo si  $M_L$  acepta a  $x$ . Por consiguiente  $M'$  acepta a  $\emptyset$  o a  $L$  dependiendo de si  $M$  acepta a  $w$  o no.

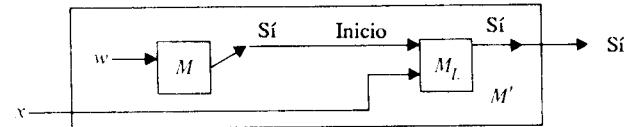


Fig. 8.11  $M'$  utilizada en el teorema de Rice.

Podemos utilizar la  $M_{\mathcal{L}}$  hipotética para determinar si  $L(M')$  está en  $\mathcal{L}$ . Ya que  $L(M')$  está en  $\mathcal{L}$  si y sólo si  $\langle M, w \rangle$  está en  $L_u$ , tenemos un algoritmo para reconocer a  $L_u$ , lo que constituye una contradicción. Por consiguiente  $\mathcal{L}$  debe ser irresoluble. Nótese cómo esta demostración generaliza lo que se expuso en el Ejemplo 8.2  $\square$

El Teorema 8.6 tiene una gran variedad de consecuencias, algunas de las cuales se resumen en el siguiente corolario.

**Corolario** Las siguientes propiedades de los conjuntos r.e. son irresolubles.

- a) vacuidad,
- b) finitud,
- c) regularidad,
- d) libertad de contexto.

### Teorema de Rice para conjuntos de índice enumerables de manera recursiva.

La condición bajo la cual un conjunto  $L_{\mathcal{L}}$  es r.e. es mucho más complicada. Debemos demostrar que  $L_{\mathcal{L}}$  es r.e. si y sólo si  $\mathcal{L}$  satisface las siguientes tres condiciones.

1. Si  $L$  está en  $\mathcal{L}$  y  $L \subseteq L'$ , para algún  $L'$  r.e., entonces  $L'$  está en  $\mathcal{L}$  (*propiedad de contención*).
2. Si  $L$  es un lenguaje infinito en  $\mathcal{L}$ , entonces existe un subconjunto finito de  $L$  en  $\mathcal{L}$ .
3. El conjunto de lenguajes finitos en  $\mathcal{L}$  es *enumerable*, en el sentido de que existe una máquina de Turing que genera la cadena (posiblemente) infinita código<sub>1</sub># código<sub>2</sub># ..., en donde código<sub>i</sub> es un código para el  $i$ -ésimo lenguaje finito de  $\mathcal{L}$  (en cualquier orden). El código para el lenguaje finito  $\{w_1, w_2, \dots, w_n\}$  es justamente  $w_1, w_2, \dots, w_n$ .

Demostraremos esta caracterización con una serie de lemas.

**Lema 8.2** Si  $\mathcal{L}$  no posee la propiedad de contención, entonces  $L_{\mathcal{L}}$  no es r.e.

*Demuestra* Generalizamos la demostración de que  $L_{\mathcal{L}}$  es no r.e. Hagamos que  $L_1$  esté en  $\mathcal{L}$ ,  $L_1 \subseteq L_2$ , y que  $L_2$  no se encuentre en  $\mathcal{L}$ . [Para el caso en donde  $\mathcal{L}$  era los conjuntos no recursivos, escogimos  $L_1 = L_u$  y  $L_2 = (\mathbf{0} + 1)^*$ .] Constrúyase el algoritmo  $A$  que tome a  $\langle M, w \rangle$  como entrada y que produzca como TM de salida a  $M'$  con el comportamiento que se muestra en la Fig. 8.12, en donde  $M_1$  y  $M_2$  aceptan a  $L_1$  y  $L_2$ , respectivamente. Si  $M$  acepta a  $w$ , entonces  $M_2$  se pone en funcionamiento y  $M'$  acepta a  $x$  siempre que  $x$  esté en  $L_1$  o  $L_2$ . Si  $M$  no acepta a  $w$ , entonces  $M_2$  nunca se pone en funcionamiento, así que  $M'$  acepta a  $x$  y sólo si  $x$  está en  $L_1$ . Como  $L_1 \subseteq L_2$ ,

$$L(M') = \begin{cases} L_2 & \text{si } M \text{ acepta a } w \\ L_1 & \text{si } M \text{ no acepta a } w. \end{cases}$$

Por tanto  $L(M')$  está en  $\mathcal{L}$  si y sólo si  $M$  no acepta a  $w$ .

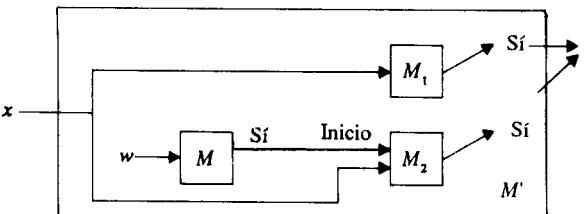


Fig. 8.12 La TM  $M'$

De nuevo cuenta dejamos al lector diseñar el “compilador”  $A$  que toma como entrada  $\langle M, w \rangle$  y los conecta con las máquinas de Turing fijas  $M_1$  y  $M_2$  para construir la  $M'$  que se muestra en la Fig. 8.12. Una vez que se ha construido  $A$ , podemos utilizar una TM  $M_u$  para  $L_u$  que acepte a  $\bar{L}_u$  como se ilustra en la Fig. 8.13. Esta TM acepta a  $\langle M, w \rangle$  si y sólo si  $M'$  acepta un lenguaje que esté en  $\mathcal{L}$ , de manera equivalente, si y sólo si  $M$  no acepta a  $w$ . Como una TM así no existe, sabemos que  $M_u$  no puede existir, de modo que  $L_u$  es no r.e.  $\square$

Volvemos ahora nuestra atención a la segunda propiedad de los conjuntos de índice enumerables de manera recursiva.

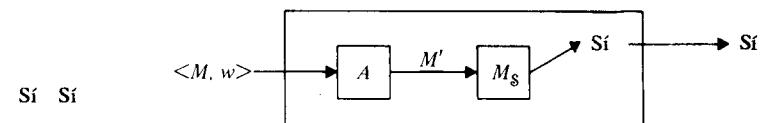


Fig. 8.13 TM hipotética para aceptar a  $\bar{L}_u$

**Lema 8.3** Si  $\mathcal{L}$  tiene un lenguaje infinito  $L$  tal que ningún subconjunto finito de  $L$  esté en  $\mathcal{L}$ , entonces  $L_{\mathcal{L}}$  es no r.e.

*Demuestra* Supóngase que  $L_{\mathcal{L}}$  es r.e. Debemos demostrar que  $\bar{L}_u$  sería r.e. de la manera siguiente. Sea  $M_1$  una TM que acepta a  $L$ . Constrúyase un algoritmo  $A$  que tome como entrada al par  $\langle M, w \rangle$  y produzca como salida a una TM  $M'$  que acepte a  $L$  si  $w$  se encuentra en  $L(M)$  y acepta, en cualquier otro caso, a algún subconjunto finito de  $L$ . Como se muestra en la Fig. 8.14,  $M'$  simula a  $M_1$  en su entrada  $x$ . Si  $M_1$  acepta a  $x$ , entonces  $M'$  simula a  $M$  sobre  $w$  para  $|x|$  movimientos. Si  $M$  no puede aceptar a  $w$  después de  $|x|$  movimientos, entonces  $M'$  acepta a  $x$ . Dejamos el diseño del algoritmo  $A$  como ejercicio.

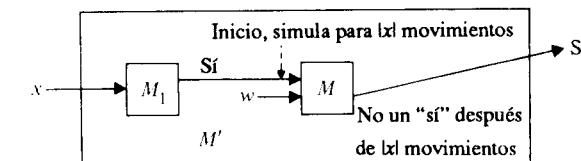


Fig. 8.14 Construcción de  $M'$

Si  $w$  está en  $L(M)$ , entonces  $M$  acepta a  $w$  después de cierto número de movimientos, digamos  $j$ . Entonces  $L(M') = \{x \mid x \text{ está en } L \text{ y } |x| \leq j\}$ , que es un subconjunto finito de  $L$ . Si  $w$  no está en  $L(M)$ , entonces  $L(M') = L$ . De aquí que si  $M$  no acepta a  $w$ ,  $L(M')$  está

en  $\mathcal{L}$ , y si  $M$  acepta a  $w, L(M')$ , siendo un subconjunto finito de  $L$ , no está en  $\mathcal{L}$ , según la hipótesis del lema. Un argumento, que ya se ha generalizado, demuestra que si  $L_{\neq}$  es r.e., también lo es  $L_u$ . Ya que el último no es r.e., concluimos que el primero tampoco es r.e.  $\square$

Finalmente, consideraremos la tercera propiedad de los conjuntos de índice r.e.

**Lema 8.4** Si  $L_{\neq}$  es r.e., entonces la lista de códigos binarios para los conjuntos finitos que están en  $\mathcal{L}$  es enumerable.

*Demostración* Utilizaremos el generador de pares descrito en la Sección 7.7. Cuando se genera  $(i, j)$ , tratamos  $i$  como el código binario para un conjunto finito, suponiendo que 0 es el código para la coma, 10 el código para cero y 11 el código para uno. De manera clara, podemos construir un TM  $M^{(i)}$  (esencialmente un autómata finito) que acepta exactamente las palabras que están en el lenguaje finito representadas por  $i$ . Entonces simulando el enumerador de  $L_{\neq}$  para  $j$  pasos. Si ha impreso a  $M^{(i)}$ , imprimimos el código para el conjunto finito representado por  $i$ , esto es, la representación binaria de  $i$  misma, seguido por un símbolo delimitador #. En cualquier circunstancia, después de la simulación, regresamos el control al generador de pares, el cual genera el par que sigue a  $(i, j)$ .  $\square$

**Teorema 8.7**  $L_{\neq}$  es r.e. si y sólo si

1. Si  $L$  está en  $\mathcal{L}$  y  $L \subseteq L'$ , para algún r.e.  $L'$ , entonces  $L'$  está en  $\mathcal{L}$ .
2. Si  $L$  es un conjunto infinito de  $\mathcal{L}$ , entonces existe un subconjunto finito  $L'$  de  $L$  que está en  $\mathcal{L}$ .
3. El conjunto de lenguajes finitos que están en  $\mathcal{L}$  es enumerable.

*Demostración* La parte “sólo si” está constituida por los Lemas 8.2, 8.3 y 8.4. Para la parte “si”, supóngase que son válidos (1), (2) y (3). Construimos una TM  $M_1$  que reconoce a  $\langle M \rangle$  si y sólo si  $L(M)$  está en  $\mathcal{L}$  de la manera siguiente.  $M_1$  genera a los pares  $(i, j)$  utilizando el generador de pares. Como respuesta a  $(i, j)$ ,  $M_1$  simula a  $M_2$ , que es un enumerador de los conjuntos finitos de  $\mathcal{L}$ , para  $i$  pasos. Sabemos que  $M_2$  existe por la condición (3). Sea  $L_1$  el último conjunto impreso completamente por  $M_2$ . [Si no existe un conjunto completamente impreso, genérese el siguiente par  $(i, j)$ .] Entonces simúlese  $M$  para  $j$  pasos en cada palabra de  $L_1$ . Si  $M$  acepta a todas las palabras de  $L_1$ , entonces  $M_1$  acepta a  $\langle M \rangle$ . Si no,  $M_1$  genera al siguiente par  $(i, j)$ .

Utilizamos las condiciones (1) y (2) para demostrar que  $L(M_1) = L_{\neq}$ . Supóngase que  $L$  se encuentra en  $L_{\neq}$  y sea  $M$  cualquier TM con  $L(M) = L$ . Según la condición (2), existe un conjunto finito  $L' \subseteq L$  en  $\mathcal{L}$  (si  $L$  es finito tómese  $L' = L$ ). Hagamos que  $L'$  sea generado después de  $i$  pasos de  $M_2$ , y sea  $j$  el máximo número de pasos que le toma a  $M$  aceptar una palabra de  $L'$  (si  $L' = \emptyset$ , hágase  $j = 1$ ). Entonces cuando  $M_1$  genera  $(i, j)$ , si no antes,  $M_1$  aceptará a  $\langle M \rangle$ .

De manera inversa, supóngase que  $M_1$  acepta a  $\langle M \rangle$ . Entonces existe algún par  $(i, j)$  tal que, dentro de  $j$  pasos,  $M$  acepta cada palabra de algún lenguaje finito  $L'$  tal que  $M_2$  genera a  $L'$  dentro de sus primeros  $i$  pasos. Entonces  $L'$  se encuentra en  $\mathcal{L}$  y  $L \subseteq L(M)$ . Según la condición (1),  $L(M)$  está en  $\mathcal{L}$ , de modo que  $\langle M \rangle$  se encuentra en  $L_{\neq}$ . Concluimos que  $L(M_1) = L_{\neq}$ .  $\square$

El Teorema 8.7 tiene una gran variedad de consecuencias. Resumiremos algunas de ellas como corolarios y dejaremos las otras como ejercicios.

**Corolario 1** Las siguientes propiedades de los conjuntos r.e. son no r.e.

- a.  $L = \emptyset$ .
- b.  $L = \Sigma^*$ .
- c.  $L$  es recursivo.
- d.  $L$  es no recursivo.
- e.  $L$  es un singletón (posee exactamente un miembro).
- f.  $L$  es un conjunto regular.
- g.  $L - L_u \neq \emptyset$ .

*Demostración* En cada caso se viola la condición (1), excepto en (b), donde la condición que se viola es la (2), y (g), en donde se viola la condición (3).  $\square$

**Corolario 2** Las siguientes propiedades de los conjuntos r.e. son r.e.

- a.  $L \neq \emptyset$ .
- b.  $L$  contiene al menos diez miembros.
- c.  $w$  está en  $L$  para alguna palabra fija  $w$ .
- d.  $L \cap L_u \neq \emptyset$ .

### Problemas acerca de las máquinas de Turing

¿Dice el Teorema 8.6 que todo lo concerniente a las máquinas de Turing es irresoluble? La respuesta es no. Ese teorema tiene que ver sólo con las propiedades de los lenguajes aceptados, no con las propiedades de la máquina de Turing misma. Por ejemplo, la pregunta “¿Tiene una máquina de Turing dada un número par de estados?” claramente es resoluble. Cuando se trabaja con las propiedades de las máquinas de Turing mismas, debemos hacer uso de nuestra ingenuidad. Damos a continuación dos ejemplos.

**Ejemplo 8.4** Es irresoluble si una máquina de Turing con alfabeto  $\{0, 1, B\}$  nunca imprime tres 1s consecutivos en su cinta. Para cada máquina de Turing  $M_i$  construimos  $M_i'$ , la que sobre una cinta en blanco simula a  $M_i$  sobre la cinta en blanco. Sin embargo  $M_i'$  utiliza 01 para codificar a 0 y 10 para codificar a 1. Si la cinta de  $M_i$  tiene un 0 en la celda  $j$ ,  $M_i'$  tiene 01 en las celdas  $2j-1$  y  $2j$ . Si  $M_i$  cambia un símbolo,  $M_i'$  cambia el correspondiente 1 a 0, después el 0 apareado a 1. Uno puede diseñar a  $M_i$  de manera sencilla de forma que  $M_i'$  nunca tenga tres 1s consecutivos en su cinta. Ahora modificamos aún más a  $M_i$  de modo tal que si  $M_i$  acepta,  $M_i'$  imprime tres unos consecutivos y se detiene. Por consiguiente  $M_i'$  imprime tres 1s consecutivos si y sólo si  $M_i$  acepta a  $\epsilon$ . Según el Teorema 8.6, es irresoluble si una TM acepta a  $\epsilon$ , ya que el predicado “ $\epsilon$  está en  $L$ ” no es trivial. Por tanto la pregunta de si una máquina de Turing cualquiera imprime alguna vez tres 1s consecutivos es irresoluble.

**Ejemplo 8.5** Es resoluble en caso de que una máquina Turing de una sola cinta que se ponga a funcionar en una cinta en blanco barra cada celda cuatro veces o más. Si la

máquina de Turing nunca barre una celda cuatro veces o más, entonces cada *secuencia cruzada* (sucesión de estados en la cual se cruza la frontera entre celdas, suponiendo que los estados cambian antes de que la cabeza se mueva) es de longitud de cuando mucho tres. Pero existe un número finito de secuencias cruzadas distintas de longitud tres o menos. Por consiguiente, la máquina de Turing permanece dentro de un número acotado de celdas de cinta, en cuyo caso las técnicas del autómata finito responderán a la pregunta, o alguna secuencia cruzada se repite. Pero si alguna secuencia cruzada se repite, entonces la TM se mueve hacia la derecha siguiendo algún patrón fácilmente detectable, y la cuestión es de nuevo resoluble.

### 8.5. IRRESOLUBILIDAD DEL PROBLEMA DE CORRESPONDENCIA DE POST

Los problemas irresolubles surgen en una variedad de áreas. En las siguientes tres secciones exploraremos algunos de los problemas más interesantes de la teoría del lenguaje y desarrollaremos técnicas para demostrar qué problemas particulares son irresolubles. Comenzaremos con el Problema de Correspondencia de Post, que es una valiosa herramienta para establecer si otros problemas son irresolubles.

Una instancia del *Problema de Correspondencia de Post (PCP)* consiste en dos listas,  $A = w_1, \dots, w_k$  y  $B = x_1, \dots, x_k$ , de cadenas sobre algún alfabeto  $\Sigma$ . Esta instancia del PCP tiene una solución si existe cualquier secuencia de enteros  $i_1, i_2, \dots, i_m$ , con  $m \geq 1$ , tal que

$$w_{i_1}, w_{i_2}, \dots, w_{i_m} = x_{i_1}, x_{i_2}, \dots, x_{i_m}.$$

La secuencia  $i_1, \dots, i_m$  es una solución a esta instancia del PCP.

**Ejemplo 8.6** Sea  $\Sigma = \{0, 1\}$ . Sean  $A$  y  $B$  las listas de tres cadenas cada una, como se definen en la Fig. 8.15. En este caso PCP tiene una solución. Sea  $m = 4$ ,  $i_1 = 2$ ,  $i_2 = 1$ ,  $i_3 = 1$  e  $i_4 = 3$ . Entonces

$$w_2 w_1 w_1 w_3 = x_2 x_1 x_1 x_3 = 10111110.$$

	Lista A	Lista B
$i$	$w_i$	$x_i$
1	1	111
2	10111	10
3	10	0

Fig. 8.15 Una instancia del PCP.

**Ejemplo 8.7** Sea  $\Sigma = \{0, 1\}$ . Sean  $A$  y  $B$  las listas de tres cadenas que se presentan en la Fig. 8.16.

	Lista A	Lista B
$i$	$w_i$	$x_i$
1	10	101
2	011	11
3	101	011

Fig. 8.16 Otra instancia del PCP.

Supóngase que esta instancia del PCP tiene una solución  $i_1, i_2, \dots, i_m$ . Es claro que,  $i_1 = 1$ , ya que ninguna cadena que comienza con  $w_2 = 011$  puede igualar a la cadena que comienza con  $x_2 = 11$ ; ninguna cadena que comienza con  $w_3 = 101$  puede igualar a una cadena que comienza con  $x_3 = 011$ .

Escribimos la cadena de la lista  $A$  encima de la correspondiente cadena de  $B$ . Hasta aquí tenemos

$$\begin{array}{l} 10 \\ 101 \end{array}$$

La siguiente selección de  $A$  debe comenzar con 1. Por consiguiente  $i_2 = 1$  o  $i_2 = 3$ . Pero  $i_2 = 1$  no funcionará, puesto que ninguna cadena que comienza con  $w_1 w_1 = 1010$  puede igualar a una cadena que comience con  $x_1 x_1 = 101101$ . Con  $i_2 = 3$  tenemos

$$\begin{array}{l} 10101 \\ 101011 \end{array}$$

Como la cadena de la lista  $B$  de nuevo excede a la cadena de la lista  $A$  por el símbolo simple 1, un argumento parecido al anterior muestra que  $i_3 = i_4 = \dots = 3$ . Por tanto existe solamente una sucesión de alternativas que genera cadenas compatibles, y para esta sucesión, la cadena de  $B$  es siempre un carácter más larga. Por consiguiente esta instancia del PCP no tiene solución.

### Una versión modificada del PCP

Demostraremos que el PCP es irresoluble mostrando que si fuera resoluble tendríamos un algoritmo para  $L_u$ . Primero, mostraremos que si el PCP es resoluble, una versión modificada del PCP también sería resoluble.

El *Problema de Correspondencia de Post Modificado (MPCP)* es el siguiente: Dadas dos listas  $A$  y  $B$ , de  $k$  cadenas cada una y tomadas de  $\Sigma^*$ , digamos

$$A = w_1, w_2, \dots, w_k \quad y \quad B = x_1, x_2, \dots, x_k,$$

¿Existe una sucesión de enteros  $i_1, i_2, \dots, i_r$ , tal que

$$w_1 w_{i_1} w_{i_2} \cdots w_{i_r} = x_1 x_{i_1} x_{i_2} \cdots x_{i_r}?$$

La diferencia entre el MPCP y el PCP reside en que en el MPCP se requiere una solución para empezar con la primera cadena de cada lista.

**Lema 8.5** Si el PCP fuera no resoluble, entonces el MPCP sería resoluble. Es decir, el MPCP se reduce al PCP.

**Demostración** Sea

$$A = w_1, w_2, \dots, w_k \quad \text{y} \quad B = x_1, x_2, \dots, x_k$$

una instancia del MPCP. Convertimos esta instancia del MPCP a una instancia del PCP que tiene una solución si y sólo si muestra instancia del MPCP tiene una solución. Si el PCP fuera resoluble, seríamos, entonces, capaces de resolver el MPCP, demostrando el Lema. 8.5.

Sea  $\Sigma$  el alfabeto más pequeño que contiene a todos los símbolos de las listas  $A$  y  $B$ , y sean  $\$$  y  $\#$  símbolos que no se encuentran en  $\Sigma$ . Hagamos que  $y_i$  se obtenga de  $w_i$  mediante la inserción del símbolo  $\$$  después de cada carácter de  $w_i$  y hagamos que  $z_i$  se obtenga de  $x_i$  mediante la inserción del símbolo  $\#$  adelante de cada carácter de  $x_i$ . Creéñase las nuevas palabras.

$$\begin{aligned} y_0 &= \$y_1, & z_0 &= z_1, \\ y_{k+1} &= \$\$, & z_{k+1} &= \$\$. \end{aligned}$$

Sean  $C = y_0, y_1, \dots, y_{k+1}$  y  $D = z_0, z_1, \dots, z_{k+1}$ . Porejemplo, las listas  $C$  y  $D$  que se construyeron a partir de las listas  $A$  y  $B$  del Ejemplo 8.6 se muestran en la Fig. 8.17.

	Lista A		Lista B		PCP		
	$i$	$w_i$	$x_i$	$i$	$y_i$	$z_i$	
MPCP	1	1	111	0	$\$1\$$	$\$1\$1\$1$	PCP
	2	10111	10	1	$1\$$	$\$1\$1\$1$	
	3	10	0	2	$1\$0\$1\$1\$1\$$	$\$1\$0$	
				3	$\$0\$$	$\$0$	
				4	$\$\$$	$\$\$$	

Fig. 8.17 Instancias correspondientes del MPCP y del PCP.

En general, las listas  $C$  y  $D$  representan una instancia del PCP. Pedimos que esta instancia del PCP tenga solución si y sólo si la instancia del MPCP representada por las listas  $A$  y  $B$  tiene solución. Para ver esto, nótese que si  $1, i_1, i_2, \dots, i_r$  es una solución para el MPCP con listas  $A$  y  $B$ , entonces

$$0, i_1, i_2, \dots, i_r, k+1$$

es una solución al PCP con listas  $C$  y  $D$ . De la misma forma, si  $i_1, i_2, \dots, i_r$  es una solución para el PCP con listas  $C$  y  $D$ , entonces  $i_1 = 0$  e  $i_r = k+1$  ya que  $y_0$  y  $z_0$  son las únicas palabras con el mismo índice que comienzan con el mismo símbolo, y  $y_{k+1}$  y  $z_{k+1}$  son las únicas palabras con el mismo índice que terminan con el mismo símbolo. Sea  $j$  el entero más pequeño tal que  $i_j = k+1$ . Entonces  $i_1, i_2, \dots, i_r$  es también una solución, ya que el símbolo  $\$$  ocurre solamente como el último símbolo de  $y_{k+1}$  y  $z_{k+1}$ , y, para ninguna  $\ell$ , en donde  $1 \leq \ell \leq j$ , se tiene  $i_\ell = k+1$ . Es claro que  $1, i_1, i_2, \dots, i_{j-1}$  es una solución del MPCP para las listas  $A$  y  $B$ .

Si existe un algoritmo para resolver a PCP, podemos construir un algoritmo para resolver a MPCP mediante la conversión de cualquier instancia de MCP a PCP como se hizo más arriba.  $\square$

### Irresolubilidad del PCP

**Teorema 8.8** El PCP es irresoluble.

**Demostración** Tomando en cuenta el Lema 8.5, sólo es suficiente mostrar que si el MPCP fuera resoluble entonces sería resoluble la cuestión de si una TM acepta a una palabra dada. Esto, es reducimos  $L_w$  a MPCP, el cual, según el Lema 8.5, se reduce al PCP. Para cada  $M$  y  $w$  construimos una instancia del MPCP que tiene una solución si y sólo si  $M$  acepta a  $w$ . Hacemos esto mediante la construcción de una instancia del MPCP que, si tiene una solución, tiene una que comienza con  $\#q_0 w \# \alpha_1 q_1 \beta_1 \# \cdots \# \alpha_k q_k \beta_k \#$ , en donde las cadenas que se encuentran entre  $\#$ s sucesivos son IDs sucesivas de un cálculo de  $M$  con entrada  $w$ , y  $q_k$  es un estado final.

Formalmente, los pares de cadenas que forman las listas  $A$  y  $B$  de la intancia de MPCP se dan enseguida. Ya que, excepto para el primer par, que se utilizará primero, el orden de los pares es irrelevante para la existencia de una solución, los pares se darán sin número de índice. Suponemos que no hay movimientos a partir de un estado final.

El primer par es:

$$\begin{array}{ll} \text{Lista A} & \text{Lista B} \\ \# & \#q_0 w \# \end{array}$$

Los pares restantes se agrupan de la manera siguiente:

*Grupo I*

$$\begin{array}{ll} \text{Lista A} & \text{Lista B} \\ X & X \\ \# & \# \end{array} \quad \text{para cada } X \text{ en } \Gamma.$$

*Grupo II.* Para cada  $q$  en  $Q - F$ ,  $p$  en  $Q$  y  $X, Y$  y  $Z$  en  $\Gamma$ :

$$\begin{array}{lll} \text{Lista A} & \text{Lista B} & \\ qX & Yp & \text{if } \delta(q, X) = (p, Y, R) \\ ZqX & pZY & \text{if } \delta(q, X) = (p, Y, L) \\ q\# & Yp\# & \text{if } \delta(q, B) = (p, Y, R) \\ Zq\# & pZY\# & \text{if } \delta(q, B) = (p, Y, L) \end{array}$$

*Grupo III.* Para cada  $q$  en  $F$  y  $X$  y  $Y$  en  $\Gamma$ :

Lista A	Lista B
$XqY$	$q$
$Xq$	$q$
$qY$	$q$

*Grupo IV*

Lista A	Lista B
$q\#$	#

para cada  $q$  en  $F$ .

Digamos que  $(x, y)$  es una *solución parcial* del MPCP con lista  $A$  y  $B$  si  $x$  es un prefijo de  $y$ , y  $x$  y  $y$  son la concatenación de las correspondientes cadenas de las listas  $A$  y  $B$ , respectivamente. Si  $xz = y$ , entonces llámese a  $z$  el *residuo* de  $(x, y)$ .

Supóngase que de la ID  $q_0w$  existe una secuencia válida de  $k$  IDs más. Pedimos que exista una solución parcial

$$(x, y) = (\#q_0 w \# \alpha_1 q_1 \beta_1 \# \cdots \# \alpha_{k-1} q_{k-1} \beta_{k-1} \#, \\ \# q_0 w \# \alpha_1 q_1 \beta_1 \# \cdots \# \alpha_k q_k \beta_k \#).$$

Aún más, que ésta sea la única solución parcial cuya cadena más larga es de longitud  $|y|$ .

Es fácil demostrar la proposición anterior por inducción en  $k$ . Para  $k = 0$  ésta es trivial, ya que el par  $(\#, \#q_0w\#)$  deberá escogerse primero.

Supóngase que la proposición es verdadera para alguna  $k$  y que  $q_k$  no se encuentra en  $F$ . Podemos mostrar fácilmente que es verdadera para  $k + 1$ . El residuo del par  $(x, y)$  es  $z = \alpha_k q_k \beta_k \#$ . Los siguientes pares deben escogerse de tal forma que sus cadenas de la lista A forman a  $z$ . No importa qué símbolos aparezcan a la derecha e izquierda de  $q_k$ , existe cuando mucho un par del Grupo II que permitirá a la solución parcial continuar más allá de  $q_k$ . Este par representa, de una manera natural, el movimiento de  $M$  a partir de la ID  $\alpha_k q_k \beta_k$ . Los otros símbolos de  $z$  fuerzan alternativas del Grupo I. Ninguna otra alternativa le permitirá a  $z$  ser compuesta con elementos de la lista A.

Por tanto, podemos obtener una nueva solución parcial  $(y, y \alpha_{k+1} q_{k+1} \beta_{k+1} \#)$ . Es fácil ver que  $\alpha_{k+1} q_{k+1} \beta_{k+1}$  es la ID que puede ser alcanzada por  $M$  en un movimiento a partir de  $y \alpha_k q_k \beta_k$ . También, no existe otra solución parcial cuya longitud de la segunda cadena iguale a  $|y \alpha_{k+1} q_{k+1} \beta_{k+1} \#|$ .

Además, si  $q_k$  está en  $F$ , es fácil encontrar pares de los grupos I y III que, cuando están precedidos por la solución parcial  $(x, y)$  y seguidos por el par del Grupo IV, proporcionan una solución al MPCP con listas A y B.

Por consiguiente si  $M$ , iniciada en la ID  $q_0w$ , alcanza un estado de aceptación, la instancia del MPCP con listas A y B tiene una solución. Si  $M$  no alcanza un estado de aceptación, ningún par de los Grupos III o IV puede ser utilizado. Por consiguiente, pueden existir soluciones parciales, pero la cadena de B debe exceder en longitud a la cadena de A, de modo que ninguna solución es posible.

Concluimos que la instancia del MPCP tiene una solución si y sólo si  $M$  con entrada  $w$  se detiene en un estado de aceptación. Puesto que la construcción dada más arriba puede efectuarse para una  $M$  y  $w$  cualesquiera, se concluye que si existiera un

algoritmo para resolver el MPCP, entonces debería haber un algoritmo para reconocer a  $L_w$ , contradiciendo lo establecido por el Teorema 8.5.  $\square$

**Ejemplo 8.8** Sea

$$M = (\{q_1, q_2, q_3\}, \{0, 1, B\}, \{0, 1\}, \delta, q_1, B, \{q_3\}),$$

y definamos  $\delta$  mediante:

$q_i$	$\delta(q_i, 0)$	$\delta(q_i, 1)$	$\delta(q_i, B)$
$q_1$	$(q_2, 1, R)$	$(q_2, 0, L)$	$(q_2, 1, L)$
$q_2$	$(q_3, 0, L)$	$(q_1, 0, R)$	$(q_2, 0, R)$
$q_3$	—	—	—

Sea  $w = 01$ . Construimos una instancia del MPCP con listas A y B. El primer par es  $\#$  para la lista A y  $\#q_101\#$  para la lista B. Los pares restantes son

*Grupo I*

Lista A	Lista B
0	0
1	1
#	#

*Grupo II*

Lista A	Lista B	
$q_10$	$1q_2$	de $\delta(q_1, 0) = (q_2, 1, R)$
$0q_11$	$q_200$	de $\delta(q_1, 1) = (q_2, 0, L)$
$1q_11$	$q_210$	
$0q_1\#$	$q_201\#$	de $\delta(q_1, B) = (q_2, 1, L)$ <sup>†</sup>
$1q_1\#$	$q_211\#$	
$0q_20$	$q_300$	de $\delta(q_2, 0) = (q_3, 0, L)$
$1q_20$	$q_310$	
$q_21$	$0q_1$	de $\delta(q_2, 1) = (q_1, 0, R)$
$q_2\#$	$0q_2\#$	de $\delta(q_2, B) = (q_2, 0, R)$

*Grupo III*

Lista A	Lista B
$0q_30$	$q_3$
$0q_31$	$q_3$
$1q_30$	$q_3$
$1q_31$	$q_3$
$0q_3$	$q_3$

<sup>†</sup> Puesto que B nunca es impreso, podemos omitir los pares en los que B está a la derecha del estado. Los pares del Grupo III también omiten aquellos con B en uno o ambos lados del estado.

$1q_3$	$q_3$
$q_30$	$q_3$
$q_31$	$q_3$

**Grupo IV**

Lista A	Lista B
$q_3\#\#$	#

Nótese que  $M$  acepta la entrada  $w = 01$  mediante la sucesión de IDs:

$$q_101, 1q_21, 10q_1, 1q_201, q_3101.$$

Veamos si existe una solución al MPCP que hemos construido. El primer par da una solución parcial ( $\#, \#q_101\#$ ). Una inspección de los pares indica que la única manera de obtener una solución parcial más larga es utilizando enseguida el par  $(q_10, 1q_2)$ . La solución parcial resultante es  $(\#q_10, \#q_101\#1q_2)$ . El residuo es ahora  $1\#1q_2$ . Los siguientes tres pares seleccionados deben ser  $(1, 1)$ ,  $(\#, \#)$  y  $(1, 1)$ . La solución parcial se convierte en  $(\#q_101\#1, \#q_101\#1q_21\#1)$ . Y ahora el residuo es  $q_21\#1$ . Continuando con el argumento, vemos que la única solución parcial, cuya longitud de su segunda cadena es 14, es  $(x, x0q_1\#1)$ , en donde  $x = \#q_101\#1q_21\#1$ .

Aquí, aparentemente tenemos una alternativa, porque el siguiente par utilizado puede ser  $(0, 0)$  o  $(0q_1\#, q_201\#)$ . En el primer caso tenemos como solución parcial a  $(x0, x0q_1\#10)$ . Pero esta solución parcial es un “extremo muerto”. No se le puede agregar ningún par para formar otra solución parcial, así que, seguramente no puede llevarnos a una solución.

De manera similar, continuamos forzados, por nuestro deseo de alcanzar una solución, a escoger un par específico para seguir con cada solución parcial. Finalmente, alcanzamos la solución parcial  $(y, y1\#q_310)$ , en donde

$$y = \#q_101\#1q_21\#10q_1\#1q_20.$$

Como  $q_3$  es un estado final, podemos ahora utilizar los pares de los Grupos I, III y IV para encontrar una solución a la instancia del MPCP. La selección de pares es

$$\begin{aligned} &(1, 1), (\#, \#), (q_31, q_3), (0, 0), (1, 1), (\#, \#), (q_30, q_3), \\ &(1, 1), (\#, \#), (q_31, q_3), (\#, \#), (q_3\#\#, \#). \end{aligned}$$

Por consiguiente, la palabra más corta que puede ser compuesta con las correspondientes cadenas tomadas de las listas  $A$  y  $B$ , comenzando con el par 1, es

$$\#q_101\#1q_21\#10q_1\#1q_201\#q_3101\#q_301\#q_3\#\#.$$

**Una aplicación del PCP**

El problema de correspondencia de Post puede utilizarse para demostrar que una amplia variedad de problemas son irresolubles. Aquí sólo daremos una aplicación: la irresolubilidad de la ambigüedad para las gramáticas libres de contexto. El lector

deberá consultar los ejercicios que se encuentran al final del capítulo para ver aplicaciones adicionales.

**Teorema 8.9** Es irresoluble la cuestión de si una CFG cualquiera es ambigua.

**Demostración** Sean

$$y \quad A = w_1, w_2, \dots, w_n \quad y \quad B = x_1, x_2, \dots, x_n$$

dos listas de palabras sobre un alfabeto finito  $\Sigma$ . Sean  $a_1, a_2, \dots, a_n$  los nuevos símbolos. Y hagamos

$$L_A = \{w_{i_1} w_{i_2} \cdots w_{i_m} a_{i_m} a_{i_{m-1}} \cdots a_{i_1} \mid m \geq 1\}$$

y

$$L_B = \{x_{i_1} x_{i_2} \cdots x_{i_m} a_{i_m} a_{i_{m-1}} \cdots a_{i_1} \mid m \geq 1\}.$$

Hagamos, también, que  $G$  sea la CFG

$$\{S, S_1, S_2\}, \Sigma \cup \{a_1, \dots, a_n\}, P, S,$$

en donde  $P$  contiene a las producciones  $S \rightarrow S_A, S \rightarrow S_B$  y para  $1 \leq i \leq n, S_A \rightarrow w_i S_A a_i, S_A \rightarrow w_i a_i, S_B \rightarrow x_i S_B a_i$  y  $S_B \rightarrow x_i a_i$ . La gramática  $G$  genera al lenguaje  $L_A \cup L_B$ .

Si la instancia  $(A, B)$  del PCP tiene una solución, digamos  $i_1, i_2, \dots, i_m$ , entonces existe una palabra  $x_{i_1} x_{i_2} \cdots x_{i_m} a_{i_m} a_{i_{m-1}} \cdots a_{i_1}$  en  $L_A$  que iguala a la palabra  $w_{i_1} w_{i_2} \cdots w_{i_m} a_{i_m} a_{i_{m-1}} \cdots a_{i_1}$  en  $L_B$ . Esta palabra tiene una derivación izquierda que comienza  $S \rightarrow S_A$  y otra que comienza  $S \rightarrow S_B$ . De aquí que en este caso  $G$  sea ambigua.

De manera inversa, supóngase que  $G$  es ambigua. Puesto que las  $as$  determinan las producciones utilizadas, es fácil demostrar que cualquier palabra derivada de  $S_A$  tiene solamente una derivación izquierda de  $S_A$ . De manera similar, ninguna palabra derivada de  $S_B$  tiene más de una derivación izquierda que parte de  $S_B$ . Por consiguiente debe darse el hecho de que alguna palabra tenga derivaciones izquierdas que parten de  $S_A$  y  $S_B$ . Si esta palabra es  $ya_{i_m} a_{i_{m-1}} \cdots a_{i_1}$ , en donde  $y$  está en  $\Sigma^*$ , entonces  $i_1, i_2, \dots, i_m$  es una solución al PCP.

Por tanto  $G$  es ambigua si y sólo si la instancia  $(A, B)$  del PCP tiene una solución. Hemos, de esta manera, reducido el PCP al problema de la ambigüedad para las CFGs. Esto es, si hubiera un algoritmo para el último problema, podríamos construir un algoritmo para el PCP, el cual, según el Teorema 8.8., no existe. Por tanto el problema de la ambigüedad para las CFGs es irresoluble. □

## 8.6. CALCULOS VALIDOS Y NO VALIDOS DE TMS: UNA HERRAMIENTA PARA PROBAR PROBLEMAS CFL IRRESOLUBLES

Mientras que los PCPs pueden reducirse fácilmente la mayoría de los problemas irresolubles conocidos con respecto a los CFLs, existe un método más directo, que resulta instructivo. En esta sección demostraremos reducciones directas del problema de la membresía para las TMs a diferentes problemas concernientes a los CFLs. Para

hacerlo, necesitamos introducir los conceptos de cálculos válidos y no válidos de la máquina de Turing.

Un *cálculo válido* de una máquina de Turing  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ , para los fines de la presente sección, es una cadena  $w_1 \# w_2^R \# w_3 \# w_4^R \dots$  tal que:

1. cada  $w_i$  es una ID de  $M$ , una cadena de  $\Gamma^* Q \Gamma^*$  que no termina con  $B$ ,
2.  $w_1$  es una ID inicial que tiene la forma  $q_0 x$  para  $x \in \Sigma^*$ ,
3.  $w_n$  es una ID final, es decir, una ID que está en  $\Gamma^* F \Gamma^*$  y
4.  $w_i \vdash_M w_{i+1}$  para  $1 \leq i < n$ .

Suponemos, sin pérdida de generalidad, que  $Q$  y  $\Gamma$  son disjuntos y que  $\#$  no está en  $Q$  ni en  $\Gamma$ .

El *conjunto de cálculos no válidos* de una máquina de Turing es el complemento del conjunto de cálculos válidos con respecto al alfabeto  $\Gamma \cup Q \cup \{\#\}$ .

Los conceptos de cálculos válidos y no válidos resultan de utilidad para demostrar que muchas de las propiedades de los CFLs son irresolubles. La razón es que el conjunto de cálculos no válidos es un CFL y el conjunto de cálculos válidos es la intersección de dos CFLs.

**Lema 8.6** El conjunto de cálculos válidos de una máquina de Turing  $M$  es la intersección de dos CFLs,  $L_1$  y  $L_2$ , las gramáticas de dichos CFLs pueden construirse de manera efectiva a partir de  $M$ .

*Demostración* Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  una TM. Ambos CFLs,  $L_1$  y  $L_2$ , consistirán en cadenas de la forma  $x_1 \# x_2 \# \dots \# x_m \#$ . Utilizamos  $L_1$  para imponer la condición  $x_i \vdash (x_{i+1})^R$  para  $i$  impar, y  $L_2$  para imponer la condición  $x_i^R \vdash x_{i+1}$  para  $i$  par.  $L_2$  también impone la condición de que  $x_1$  es una ID inicial. El hecho es que  $x_m$  sea una ID final o su inversa es impuesto por  $L_1$  o  $L_2$ , dependiendo de si  $m$  es impar o par, respectivamente. Entonces  $L_1 \cap L_2$  es el conjunto de cálculos válidos de  $M$ .

Para comenzar, sea  $L_3$  el conjunto  $\{y \# z^R \mid y \vdash_M z\}$ . Es fácil construir un PDA  $P$  que acepte a  $L_3$ .  $P$  lee  $y$ , la entrada hasta  $\#$ , verificando en su control finito que  $y$  sea de la forma  $\Gamma^* Q \Gamma^*$ . En el proceso,  $P$  coloca sobre su pila la ID  $z$ , con  $y \vdash_M z$ , en donde  $y$  es la entrada antes del  $\#$ . Esto es, cuando la entrada a  $P$  es un símbolo de  $\Gamma$ ,  $P$  incluye el símbolo en la pila. Si la entrada es un estado  $q$  de  $Q$ ,  $P$  almacena a  $q$  en el control finito y lee el siguiente símbolo de entrada, digamos  $X$  (si el siguiente símbolo es  $\#$ , tómese  $X$  como  $B$ ). Si  $\delta(q, X) = (p, Y, R)$ , entonces  $P$  coloca a  $Yp$  en la pila. Si  $\delta(q, X) = (p, Y, L)$ , dejemos que  $Z$  esté en el tope de la pila. Entonces  $P$  sustituye a  $Z$  por  $pZY$  (pero si la última entrada leída fue  $\#$ , y  $Y = B$ , sólo se deberá sustituir  $Z$  por  $pZ$  o por  $p$  si  $Z$  también es  $B$ ). Despues de leer el  $\#$ ,  $P$  compara cada símbolo de entrada con el símbolo del tope de la pila. Si difieren,  $P$  no tiene un siguiente movimiento y de esta manera muere. Si son iguales,  $P$  expulsa al símbolo que se encuentra en el tope de la pila. Cuando la pila queda vacía,  $P$  acepta.

Ahora, hagamos  $L_4 = (L_3 \#)^* (\{\epsilon\} \cup \Gamma^* F \Gamma^* \#)$ . Según los Teoremas 5.4 y 6.1, existe un algoritmo para construir una CFG para  $L_1$ . De una manera parecida, podemos construir un PDA para  $L_4 = \{y^R \# z \mid y \vdash_M z\}$ . La construcción de  $G_2$  para

$$L_2 = q_0 \Sigma^* \# (L_4 \#)^* (\{\epsilon\} \cup \Gamma^* F \Gamma^* \#)$$

entonces se hace sencilla, y según el Teorema 6.1 existe un algoritmo para construir una CFG  $G_2$  para  $L_2$ . Ahora  $L_1 \cap L_2$  es el conjunto de cálculos válidos de  $M$ . Esto es, si  $x_1 \# x_2 \# \dots \# x_m \#$  se encuentra en  $L_1 \cap L_2$ , entonces  $L_1$  requiere que  $x_i \vdash_M (x_{i+1})^R$  para impar;  $L_2$  requiere que  $x_1$  sea inicial y  $x_i^R \vdash_M x_{i+1}$  para  $i$  par.  $L_1$  impone el hecho de que la última ID tenga un estado de aceptación para  $m$  impar y  $L_2$  lo impone para  $m$  par.  $\square$

**Teorema 8.10** Es irresoluble, para las CFGs cualesquiera,  $G_1$  y  $G_2$  el hecho de si  $L(G_1) \cap L(G_2)$  es el vacío.

*Demostración* Según el Lema 8.6, podemos construir, a partir de  $M$ , las gramáticas  $G_1$  y  $G_2$  de forma que  $L(G_1) \cap L(G_2)$  es el conjunto de cálculos válidos de  $M$ . Si existe un algoritmo  $A$  para decir si la intersección de los lenguajes de dos CFGs es el vacío, podemos construir un algoritmo  $B$  para decir si  $L(M) = \emptyset$ , para una TM  $M$  cualquiera. Simplemente diseñamos  $B$  para construir  $G_1$  y  $G_2$  a partir de  $M$  como se hizo en el Lema 8.6, entonces aplicamos el Algoritmo  $A$  para decir si  $L(G_1) \cap L(G_2)$  es el vacío. Si la intersección es el vacío, entonces no existen cálculos válidos de  $M$ , de modo que  $L(M) = \emptyset$ . Si la intersección no es el vacío,  $L(M) \neq \emptyset$ . Es decir, el problema de la vacuidad para conjuntos r.e. se reduce al problema de la intersección para CFGs.

El Algoritmo  $B$  no puede existir, sin embargo, ya que  $L(M) = \emptyset$  es irresoluble según el Teorema 8.6. Por consiguiente,  $A$  no existe, de manera que es irresoluble el hecho de si la intersección de dos CFLs es el vacío.  $\square$

Aunque se necesita que dos lenguajes libres de contexto representen a los cálculos válidos de una máquina de Turing, el conjunto de cálculos válidos es en sí mismo un CFL. La razón reside en que no necesitamos más garantizar, para cada  $i$ , de manera simultánea que  $w_i \vdash w_{i+1}$ . Sólo necesitamos adivinar dónde se presenta un error. Esto es, debemos verificar, para una  $i$ , que  $w_i \vdash w_{i+1}$  es falsa.

**Lema 8.7** El conjunto de cálculos no válidos de una máquina de Turing  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  es un CFL.

*Demostración* Si una cadena  $w$  es un cálculo no válido, entonces se cumple una de las condiciones siguientes:

1.  $w$  no es de la forma  $x_1 \# x_2 \# \dots \# x_m \#$ , en la que  $x_i$  es una ID de  $M$ .
2.  $x_1$  no es inicial; es decir,  $x_1$  no se encuentra en  $q_0 \Sigma^*$ .
3.  $x_m$  no es final; es decir,  $x_m$  no está en  $\Gamma^* F \Gamma^*$ .
4.  $x_i \vdash_M (x_{i+1})^R$  es falsa para alguna  $i$  impar.
5.  $x_i^R \vdash_M x_{i+1}$  es falsa para alguna  $i$  par.

El conjunto de cadenas que satisfacen 1, 2 y 3 es regular, y un FA que lo acepte se puede construir fácilmente. Los conjuntos de cadenas que satisfacen 4 y 5 son, cada uno, CFLs. Demostraremos esta afirmación para (4); un argumento parecido se aplica para 5. Un PDA  $P$  para (4) selecciona de manera no determinística alguna  $x_i$  que está precedida por un número par de  $\#$ s, y mientras lee  $x_i$  almacena en su pila la ID  $z$  tal que  $x_i \vdash z$ , con el extremo derecho de  $z$  en el tope de la pila. Después de encontrar a  $\#$  en la entrada,  $P$  compara  $z$  con la siguiente  $x_{i+1}$ . Si  $z \neq x_{i+1}$ , entonces  $P$  barre el resto de su entrada y acepta.

El conjunto de cálculos no válidos es la unión de dos CFLs y un conjunto regular. Según el Teorema 6.1, este conjunto es un CFL, y se puede construir una gramática para este lenguaje de manera efectiva.  $\square$

**Teorema 8.11** Es irresoluble, para cualquier CFG  $G$ , el hecho de si  $L(G) = \Sigma^*$ .

*Demostración* Dada una TM arbitraria  $M$ , podemos construir de manera efectiva una CFG  $G$  con alfabeto terminal  $\Sigma$ , tal que  $L(G) = \Sigma^*$  si y sólo si  $L(M) = \emptyset$ . Es decir, según el Lema 8.7 podemos construir una CFG  $G$  que genere los cálculos no válidos de  $M$ . Por tanto si, para una  $G$  arbitraria,  $L(G) = \Sigma^*$  fuera irresoluble, entonces podríamos decidir, para una  $M$  cualquiera, si  $L(M) = \emptyset$ , lo que constituye una contradicción.  $\square$

### Otras consecuencias de la caracterización de los cálculos mediante CFLs

Muchos otros resultados se desprenden del Teorema 8.11.

**Teorema 8.12** Sean  $G_1$  y  $G_2$  CFGs arbitrarias y  $R$  un conjunto regular cualquiera. Los siguientes problemas son irresolubles.

- 1)  $L(G_1) = L(G_2)$ .
- 2)  $L(G_2) \subseteq L(G_1)$ .
- 3)  $L(G_1) = R$ .
- 4)  $R \subseteq L(G_1)$ .

*Demostración* Hágase  $G_2$  una gramática que genere a  $\Sigma^*$ , en donde  $\Sigma$  es el alfabeto terminal de  $G_1$ . Entonces (1) y (2) son equivalentes a  $L(G_1) = \Sigma^*$ . Tómese  $R = \Sigma^*$ , y (3) y (4) son equivalentes a  $L(G_1) = \Sigma^*$ . Por consiguiente el problema irresoluble de si un CFL es  $\Sigma^*$  se reduce a (1) hasta (4), y cada uno de estos problemas es también irresoluble.  $\square$

Nótese que, según los Teoremas 5.3 y 5.4, uno puede convertir de manera efectiva entre PDAs y CFGs, de modo que los Teoremas 8.10, 8.11 y 8.12 siguen siendo verdaderos si los CFLs están representados por PDAs en lugar de CFGs. También, el conjunto regular  $R$  del Teorema 8.12 puede estar representado por un DFA, un NFA o una expresión regular, como se quiera.

Se debería observar también que la cuestión  $L(G) \subseteq R$  es resoluble. La razón es que  $L(G) \subseteq R$  si y sólo si  $L(G) \cap \bar{R} = \emptyset$ . Pero  $L(G) \cap \bar{R}$  es un CFL y, por tanto, su vacuidad es resoluble.

Existen algunas propiedades adicionales de los lenguajes libres de contexto que podemos demostrar que son irresolubles, observando que si una TM tiene cálculos válidos sobre un conjunto infinito de entradas, su conjunto de cálculos válidos no es, en general, un CFL. Sin embargo, modificamos primero cada máquina de Turing  $M$ , de manera trivial añadiendo dos estados extra cuyo único propósito es asegurarnos que  $M$  haga al menos dos movimientos en cada cálculo. Esto se puede hacer sin que, de otra manera, se modifique el cálculo efectuado por  $M$ . El objeto de la modificación es forzar a cada cálculo válido a que contenga al menos tres IDs y asegurar, por tanto, que el conjunto de cálculos válidos sea un CFL si y sólo si  $M$  acepta a un conjunto finito.

**Lema 8.8** Sea  $M$  una máquina de Turing que hace al menos tres movimientos en cada entrada. El conjunto de cálculos válidos de  $M$  es un CFL si y sólo si el conjunto aceptado por  $M$  es un conjunto finito.

*Demostración* Si el conjunto aceptado por  $M$  es finito, el conjunto de cálculos válidos de  $M$  es finito y en consecuencia un CFL. Supóngase que el conjunto aceptado por  $M$  es infinito y el conjunto  $L$  de cálculos válidos es un CFL. Como  $M$  acepta un conjunto infinito, existe un cálculo válido.

$$w_1 \# w_2^R \# w_3 \# \dots$$

en el que las  $w_i$  son IDs y  $|w_2|$  es mayor que la constante  $n$  del lema de Ogden. Señálese los símbolos de  $w_2$  como distinguidos. Entonces podemos “sondear” a  $w_2$  sin sondear a  $w_1$  y  $w_3$ , obteniendo así un cálculo no válido que debe estar en  $L$ . Concluimos que los cálculos válidos no forman un CFL.  $\square$

**Teorema 8.13.** Para las CFGs arbitrarias  $G_1$  y  $G_2$ , es irresoluble la cuestión de si

1.  $\overline{L(G_1)}$  es un CFL;
2.  $L(G_1) \cap L(G_2)$  es un CFL.

*Demostración*

1. Dada una máquina de Turing cualquiera  $M$ , modifíquese a  $M$  sin cambiar el conjunto aceptado, de modo que haga al menos dos movimientos en cada entrada. Constrúyase la CFG  $G$  que genera a los cálculos no válidos.  $\overline{L(G)}$  es un CFL si y sólo si  $M$  acepta un conjunto finito.
2. Procédase como en la parte (1), pero constrúyanse  $G_1$  y  $G_2$  tales que  $L(G_1) \cap L(G_2)$  sea el conjunto de cálculos válidos de  $M$ .  $\square$

## 8.7 TEOREMA DE GREIBACH

Existe una similitud sorprendente entre las demostraciones de irresolubilidad en la teoría del lenguaje. Esto sugiere la existencia de un teorema análogo al Teorema de Rice para las clases de lenguajes como los CFL, y de hecho existe.

Enfoquemos nuestra atención en la clase de lenguajes  $\mathcal{C}$ , como los CFLs, y en un sistema particular (como las CFGs o los PDAs) para interpretar cadenas de lenguajes finitos como los nombres de los lenguajes. Considérese una clase  $\mathcal{C}$  de los lenguajes con la propiedad de que, dados los nombres (por ejemplo, gramáticas) de los lenguajes  $L_1$  y  $L_2$  de  $\mathcal{C}$  y un nombre (por ejemplo, un autómata finito) para un conjunto regular  $R$ , podemos construir efectivamente nombres para  $RL_1$ ,  $L_1R$  y  $L_1 \cup L_2$ . Entonces decimos que la clase  $\mathcal{C}$  es *efectivamente cerrada* con respecto a la concatenación con los conjuntos regulares y la unión. Más aún, supóngase que  $L = \Sigma^*$  es irresoluble para la clase  $\mathcal{C}$ , como es el caso para los CFLs. El siguiente teorema muestra que una amplia variedad de problemas son irresolubles para la clase  $\mathcal{C}$ .

**Teorema 8.14** (*Teorema de Greibach*) Sea  $\mathcal{C}$  una clase de lenguajes que es efectivamente cerrada con respecto a la concatenación con conjuntos regulares y la unión, y

para la cual “ $= \Sigma^*$ ” es irresoluble para cualquier  $\Sigma$  fijo lo suficientemente grande. Sea  $P$  cualquier propiedad no trivial† que es verdadera para todos los conjuntos regulares y que se conserva con respecto a  $/a$ , en donde  $a$  es un solo símbolo. (Esto es, si  $L$  tiene la propiedad  $P$ , entonces también la tiene  $L/a = \{w \mid wa \text{ está en } L\}$ .) Entonces  $P$  es irresoluble para  $\mathcal{C}$ .

**Demostración** Sea  $L_0 \subseteq \Sigma^*$  un miembro de  $\mathcal{C}$  para el cual  $P(L_0)$  es falsa, en los lugares donde  $\Sigma^*$  es lo suficientemente grande, de modo que “ $= \Sigma^*$ ” es irresoluble. Para cualquier  $L \subseteq \Sigma^*$  de  $\mathcal{C}$  constrúyase  $L_1 = L_0 \# \Sigma^* \cup \Sigma^* \# L$ .  $L_1$  está en  $\mathcal{C}$ , ya que  $\mathcal{C}$  es efectivamente cerrado con respecto a la concatenación con conjuntos regulares y con respecto a la unión. Ahora, si  $L = \Sigma^*$ , entonces  $L_1 = \Sigma^* \# \Sigma^*$ , que es un conjunto regular y, por tanto  $P(L_1)$  es verdadera. Si  $L \neq \Sigma^*$ , entonces existen palabras que no están en  $L$ . Por consiguiente  $L/\#w = L_0$ . Como  $P$  se conserva con respecto al cociente con un símbolo simple, se conserva con respecto al cociente con la cadena  $\#w$ , mediante inducción en  $|w|$ . Por consiguiente  $P(L_1)$  debe ser falsa, o de otro modo  $P(L_0)$  sería verdadero, lo que es contrario a nuestra suposición. Por tanto  $P(L_1)$  es verdadera si y sólo si  $L = \Sigma^*$ . De este modo, “ $= \Sigma^*$ ” para  $\mathcal{C}$  se reduce a la propiedad  $P$  para  $\mathcal{C}$ , y de aquí que  $P$  es irresoluble para  $\mathcal{C}$ .  $\square$

### Aplicaciones del teorema de Greibach

El teorema 8.14 puede utilizarse para mostrar, por ejemplo, que es irresoluble el hecho de si el lenguaje generado por una CFG es regular o no. Nótese que esta cuestión es diferente a preguntarse si el lenguaje generado es igual a algún conjunto regular particular  $R$ , que fue lo que se pidió en el Teorema 8.12.

**Teorema 8.15** Sea  $G$  una CFG cualquiera. Es irresoluble la cuestión de si  $L(G)$  es regular.

**Demostración** Los CFLs son efectivamente cerrados con respecto a la concatenación con conjuntos regulares y con respecto a la unión. Sea  $P$  la propiedad de que  $L$  es regular.  $P$  es no trivial para los CFLs, es verdadera para todos los conjuntos regulares y se conserva bajo el cociente con un solo símbolo, según el Teorema 3.6. Nótese que los conjuntos regulares son efectivamente cerrados con respecto al cociente con otro conjunto regular, aunque el Teorema 3.6 no exige esto (véase la discusión que sigue al teorema). Por consiguiente, según el Teorema 8.14,  $P$  es irresoluble para los CFLs.  $\square$

El Teorema 8.15 nos permite mostrar que una propiedad es irresoluble mediante la demostración de que la propiedad se conserva con respecto al cociente con un solo símbolo. Esta última tarea es, con frecuencia, relativamente sencilla como, por ejemplo, cuando se demuestra que la ambigüedad inherente es irresoluble.

**Lema 8.9** Sea  $P$  la propiedad de que un CFL no es inherentemente ambiguo. Entonces  $P$  se conserva con respecto al cociente con un solo símbolo.

† Técnicamente, una propiedad es justamente un subconjunto de  $\mathcal{C}$ . Decimos “ $L$  tiene la propiedad  $P$ ” o “ $P(L)$ ” para significar que  $L$  es miembro de  $P$ .

**Demostración.** Sean  $G = (V, T, P, S)$  una CFG no ambigua y

$$G_a = (V \cup \{[A/a] \mid A \text{ está en } V\}, T, P_a, [S/a]).$$

en donde  $P_a$  contiene a

1. todas las producciones de  $P$ ,
2.  $[A/a] \xrightarrow{*} \alpha$  si  $A \xrightarrow{*} \alpha a$  está en  $P$ ,
3.  $[A/a] \xrightarrow{*} \alpha [B/a]$  si  $A \xrightarrow{*} \alpha B\beta$  está en  $P$  y  $\beta \xrightarrow{*} \epsilon$ .

Pugnamos por que  $L(G_a) = L(G)/a$  y que  $G_a$  sea no ambigua. Para ver lo anterior, primero demuéstrese mediante una sencilla inducción que

1.  $[S/a] \xrightarrow{*} \alpha$  si y sólo si  $S \xrightarrow{*} \alpha a$  y
2.  $[S/a] \xrightarrow{*} \alpha [A/a]$  si y sólo si  $S \xrightarrow{*} \alpha A$ .

El hecho de que  $L(G_a) = L(G)/a$  se concluye de manera inmediata. Supóngase que  $G_a$  es ambigua. Entonces deberán existir dos derivaciones izquierdas

1.  $[S/a] \xrightarrow{*} \beta \xrightarrow{*} \alpha \xrightarrow{*} x$  y
2.  $[S/a] \xrightarrow{*} \gamma \xrightarrow{*} \alpha \xrightarrow{*} x$  en donde  $\beta \neq \gamma$ .

Pero entonces, en  $G$  tenemos dos derivaciones izquierdas de la cadena  $xa$ , lo que constituye una contradicción. Por tanto  $G_a$  debe ser no ambigua. Concluimos que la no ambigüedad se conserva con respecto al cociente con un solo símbolo.  $\square$

**Teorema 8.16** La ambigüedad inherente para los CFLs es irresoluble.

**Demostración** Según el Teorema 4.7,  $P$  es no trivial. Según el Lema 8.9 se conserva con respecto al cociente con un solo símbolo. Es fácil demostrar que  $P$  es verdadera para todos los conjuntos regulares. Esto es, cada conjunto regular tiene una CFG no ambigua. (El lector deberá adelantarse hasta el Teorema 9.2 para ver una construcción de una CFG no ambigua a partir de un DFA arbitrario.) Por consiguiente, según el Teorema 8.14, la ambigüedad inherente para los CFLs es irresoluble.  $\square$

## 8.8 INTRODUCCION A LA TEORIA DE LAS FUNCIONES RECURSIVAS

En la Sección 7.3 mencionamos que cada máquina de Turing puede considerarse como si se calculara una función de los enteros a los enteros, y también como una reconocedora de lenguajes. Para cada máquina de Turing y cada  $k$ , existe una función  $f_M^{(k)}(i_1, i_2, \dots, i_k)$  que toma a  $k$  enteros como argumentos y produce una respuesta entera o no está definida para tales argumentos. Si  $M$  comenzó con  $0^{i_1} 1^{i_2} \dots 0^{i_k}$  en su cinta se detiene con  $0^j$  en su cinta, y entonces decimos que  $f_M^{(k)}(i_1, \dots, i_k) = j$ . Si  $M$  no se detiene con una cinta que consiste en un bloque de 0s y con todas las demás celdas en blanco, entonces  $f_M^{(k)}(i_1, \dots, i_k)$  no está definida. Nótese que la misma máquina de Turing puede considerarse como una reconocedora de lenguaje, una calculadora para una función con un argumento, una calculadora para una función diferente con dos argumentos, y así sucesivamente.

Si  $i$  es un código entero para una TM  $M$ , como se describió en la Sección 8.3 y  $k$  es un parámetro obvio, entonces escribiremos con frecuencia  $f_i$  en lugar de  $f_M^{(k)}$ .

Recuérdese que una función calculada por una máquina de Turing se conoce como función (parcialmente) recursiva. Si sucede que la función está definida para todos los valores de sus argumentos, entonces también se le conoce como una función totalmente recursiva.

Las construcciones de las máquinas de Turing dadas antes en este capítulo y la previa pueden expresarse como funciones totalmente recursivas de una sola variable. Esto es, podemos considerar un algoritmo A que toma como entrada el código binario para una TM  $M$  y produce como salida el código binario para otra TM  $M'$ , como una función  $g$  de una variable. En particular, sea  $i$  el entero que representa a  $M$  y  $j$  el entero que representa a  $M'$ . Entonces  $g(i) = j$ . Técnicamente, la TM  $B$  que calcula a  $g$  no es A, sino, más bien, una que convierte su entrada unitaria a binaria, simula a A y entonces convierte su entrada a unitaria.

### El teorema $S_{mn}$

Nuestro primer teorema, conocido como *teorema  $S_{mn}$* , dice que dada una función parcialmente recursiva  $g(x, y)$  de dos variables, existe un algoritmo que se puede utilizar para construir, a partir de una TM para  $g$  y un valor para  $x$ , otra TM que con la entrada  $y$  calcule a  $g(x, y) = g(x, y)$ .

**Teorema 8.17** Sea  $g(x, y)$  una función parcialmente recursiva. Entonces existe una función totalmente recursiva  $\sigma$  de una variable, tal que  $f_{\sigma(x)}(y) = g(x, y)$  para todas  $x$  y  $y$ . Esto es, si  $\sigma(x)$  se trata como el entero que representa a alguna TM  $M_x$ , entonces  $f_{M_x}(y) = g(x, y)$ .

**Demuestra** Hagamos que  $M$  calcule a  $g$ . Sea A una TM que, dada la entrada  $x$  escrita en unitario, construye una TM  $M_x$  que, cuando se tiene la entrada  $y$ , la corre hacia la derecha y escribe 0\*1 en su lado izquierdo;  $M_x$  entonces regresa su cabeza al extremo izquierdo y simula a  $M$ . La salida de A es la representación unitaria de un entero  $< M_x >$  que representa a  $M_x$ . Entonces A calcula una función totalmente recursiva  $\sigma$  y  $f_{\sigma(x)}(y) = g(x, y)$ . En la demostración, nótense que para cada  $x$ ,  $\sigma(x)$  es un entero que representa a la  $M_x$  dada arriba, y para cada  $x$ ,  $M_x$  es diseñada para producir  $g(x, y)$  cuando se le da la entrada  $y$ . Puesto que  $f_{\sigma(x)}$  es la función calculada por  $M_x$ , se llega a la igualdad  $f_{\sigma(x)}(y) = g(x, y)$ .  $\square$

### Teorema de recursión

El segundo teorema, conocido como *teorema de recursión*, establece que cada función totalmente recursiva  $\sigma$  que transforma *índices* (enteros que representan máquinas de Turing) de funciones parcialmente recursivas en índices de funciones parcialmente recursivas, tiene un punto fijo  $x_0$  tal que  $f_{x_0}(y) = f_{\sigma(x_0)}(y)$  para todas las  $y$ . En otras palabras, si de alguna manera modificamos a todas las máquinas de Turing, existe siempre una máquina de Turing  $M_{x_0}$  para la cual la máquina de Turing modificada  $M_{\sigma(x_0)}$  calcula la misma función que la máquina de Turing no modificada. De entrada esto parece imposible, puesto que podemos modificar a cada máquina de Turing para que agregue un 1 a la función calculada originalmente. Uno se ve tentado a decir que  $f(y) + 1 \neq f(y)$ .

Pero nótense que si  $f(y)$  está indefinida en todos lados, entonces  $f(y) + 1$  iguala a  $f(y)$  para toda  $y$ .

**Teorema 8.18** Para cualquier función totalmente recursiva  $\sigma$  existe una  $x_0$  tal que  $f_{x_0}(x) = f_{\sigma(x_0)}(x)$  para toda  $x$ .

**Demuestra** Para cada entero  $i$  constrúyase una TM que, sobre la entrada  $x$ , calcule  $f_i(i)$  y después simula, mediante una TM universal, a la  $f_i(i)$ -ésima TM sobre  $x$ . Sea  $g(i)$  el índice de la TM construida de esa manera. Entonces, para toda  $i$  y  $x$ ,

$$f_{g(i)}(x) = f_{f_i(i)}(x). \quad (8.3)$$

Obsérvese que  $g(i)$  es una función totalmente recursiva aun si  $f_i(i)$  no está definida. Sea  $j$  un índice de la función  $\sigma g$ . Esto es,  $j$  es un código entero para una TM que, dada la entrada  $i$ , calcula a  $g(i)$  y después aplica  $\sigma$  a  $g(i)$ . Entonces para  $x_0 = g(j)$  tenemos

$$\begin{aligned} f_{x_0}(x) &= f_{g(j)}(x) \\ &= f_{f_j(j)}(x) && \text{por (8.3)} \\ &= f_{\sigma(g(j))}(x) && \text{ya que } f_j \text{ es la función } \sigma g \\ &= f_{\sigma(x_0)}(x). \end{aligned}$$

Por consiguiente  $x_0$  es un punto fijo de la transformación  $\sigma$ . Esto es, la TM  $x_0$  y la TM  $\sigma(x_0)$  calculan la misma función.  $\square$

### Aplicaciones de los teoremas de recursión y $S_{mn}$

**Ejemplo 8.9** Sea  $M_1, M_2, \dots$  cualquier enumeración de todas las máquinas de Turing. No necesitamos que esta enumeración sea la “estándar” introducida en la Sección 8.3; sólo requerimos que cualquiera que sea la representación utilizada para una TM, podamos, mediante un algoritmo, convertir de una representación cualquiera a la notación de siete parámetros que introdujimos en la Sección 7.2 y viceversa. Entonces podemos utilizar el teorema de recursión para mostrar que para alguna  $i$ ,  $M_i$  y  $M_{i+1}$  calculan, ambos, la misma función.

Sea  $\sigma(i)$  la función totalmente recursiva definida de la siguiente manera. Enumérense las TMs  $M_1, M_2, \dots$  hasta llegar a una con código entero  $i$ , como en (8.2). Nótense que los estados de la TM deben considerarse en todos los órdenes posibles para ver si  $i$  es un código para esta TM, ya que en la notación introducida en la Sección 8.3, el orden en el cual se escriben los movimientos para los diferentes estados afecta al código. Una vez que se tiene que  $M_j$  tiene código  $i$ , enumérese una TM más,  $M_{j+1}$ , y hágase  $\sigma(i)$  el código para  $M_{j+1}$ . Entonces el teorema de recursión aplicado a esta  $\sigma$  nos dice que existe alguna  $x_0$  para la cual  $M_{x_0}$  y  $M_{x_0} + 1$  definen a la misma función de una variable.

**Ejemplo 8.10** Dado un sistema formal  $F$ , como la teoría de conjuntos, podemos exhibir una máquina de Turing  $M$  tal que no existe demostración en  $F$  del hecho de que  $M$  iniciada en cualquier entrada particular se detenga, y tampoco una demostración del

hecho de que no se detenga. Constrúyase  $M$ , una TM que calcule una función de dos entradas  $g(i, j)$ , tal que

$$g(i, j) = \begin{cases} 1 & \text{si existe una demostración en } F \text{ del hecho de que } f_i(j) \\ & \text{no está definida; esto es, existe una demostración del hecho de que la } i\text{-ésima TM no se detiene cuando se le da la entrada } j; \\ & \text{no definida en cualquier otro caso.} \end{cases}$$

$M$  enumera las demostraciones de  $F$  en algún orden, imprimiendo 1 si encuentra una demostración de que la  $i$ -ésima TM no se detiene en una entrada  $j$ . Aún más, podemos construir  $M$  de manera tal que si  $g(i, j) = 1$ , entonces  $M$  se detiene, y  $M$  no se detiene en cualquier otra circunstancia. Según el teorema  $S_m$  existe  $\sigma$  tal que

$$f_{\sigma(i)}(j) = g(i, j).$$

Según el teorema de recursión, podemos construir de manera efectiva un entero  $i_0$  tal que

$$f_{i_0}(j) = f_{\sigma(i_0)}(j) = g(i_0, j).$$

Pero  $g(i_0, j) = 1$  y está, por tanto, definida si y sólo si existe una prueba en  $F$  del hecho de que  $f_{i_0}(j)$  está indefinida. Por consiguiente si  $F$  es consistente (es decir, no puede haber pruebas de una proposición y su negación), puede no haber prueba en  $F$  del hecho de que la  $i_0$ -ésima TM se detiene o no en cualquier entrada particular  $j$ .

## 8.9 CALCULOS DE ORACULO

Uno se ve tentado a preguntarse qué pasaría si el problema vacuidad o cualquier otro irresoluble, fuera resoluble. ¿Podríamos entonces calcular cualquier cosa? Debemos tener cuidado al contestar la interrogación. Si comenzamos por suponer que el problema de vacuidad es resoluble, tendremos un conjunto contradictorio de suposiciones y podemos no llegar a conclusión y podemos no llegar a conclusión alguna. Evitamos este problema al definir una máquina de Turing con oráculo.

Sea  $A$  un lenguaje,  $A \subseteq \Sigma^*$ . Una *máquina de Turing con oráculo A* es una máquina de Turing de una sola cinta que posee tres estados especiales  $q_n$ ,  $q$ , y  $q_n$ . El estado  $q$ , se utiliza para preguntarse si una cadena está o no en el conjunto  $A$ . Cuando la máquina de Turing accesa el estado  $q$ , solicita una respuesta a la pregunta: “¿Se encuentra la cadena de símbolos no en blanco a la derecha de la cabeza de cinta  $A$ ?”. La respuesta es proporcionada al cambiar el estado de la máquina de Turing, en el movimiento siguiente, a uno de los estados  $q_y$  o  $q_n$ , dependiendo de si la respuesta es si o no.<sup>†</sup> El cálculo continúa de manera normal hasta la siguiente oportunidad en que se accesa  $q$ , cuando el “oráculo” hace otra pregunta.

<sup>†</sup> Nótese que la TM puede recordar su estado anterior mediante la escritura de ese estado en su cinta justo antes de accesar a  $q$ .

Obsérvese que si  $A$  es un conjunto recursivo, entonces el oráculo puede ser simulado por otra máquina de Turing, y el conjunto aceptado por la TM con oráculo  $A$  es enumerable de manera recursiva. Por el otro lado, si  $A$  no es un conjunto recursivo y se tiene disponible un oráculo para proporcionar la respuesta correcta, entonces la TM con oráculo  $A$  puede aceptar un conjunto que no es enumerable de manera recursiva. Representamos a la máquina de Turing  $M$  con oráculo  $A$  mediante  $M^A$ . Un conjunto  $L$  es *enumerable de manera recursiva con respecto a A* si  $L = L(M^A)$  para alguna TM  $M$ . Un conjunto  $L$  es *recursivo con respecto a A* si  $L = L(M^A)$  para alguna TM  $M^A$  que siempre se detiene. Dos conjuntos de oráculos son *equivalentes* si cada uno de ellos es recursivo en el otro.

### Una jerarquía de los problemas irresolubles

Podemos, ahora, replantear la cuestión que surgió al principio de la sección como: “¿Qué conjuntos pueden ser reconocidos dado un oráculo para el problema de vacuidad?” Es claro que no todos los conjuntos pueden ser r.e. con respecto al problema de vacuidad, ya que existe un número no contable de conjuntos y solamente un número contable de TMs. Considérese el conjunto de oráculos  $S_1 = \{\langle M \rangle \mid L(M) = \emptyset\}$ , que no es un conjunto r.e., (recuérdese que  $\langle M \rangle$  es el código binario para la TM  $M$ ). Ahora considérense las TMs con oráculo  $S_1$ . Estas máquinas tienen un problema que se detiene que no es recursivo en  $S_1$ . Mediante la definición de un oráculo para los problemas de vacuidad para las TMs con oráculo  $S_1$ , y así sucesivamente, podemos desarrollar una jerarquía infinita de problemas irresolubles. De manera más específica, defínase.

$$S_{i+1} = \{\langle M \rangle \mid L^S(M) = \emptyset\}.$$

$S_{i+1}$  es un oráculo para resolver el problema de vacuidad para cálculos con respecto a  $S_i$ . Podemos, ahora, clasificar algunos problemas irresolubles (pero no todos los problemas de este tipo) mostrando su equivalencia a un conjunto  $S_i$  para alguna  $i$  particular.

**Teorema 8.19** El problema de la membresía para las TMs sin oráculos es equivalente a  $S_1$ .

*Demostración* Constrúyase  $M^{S_1}$  que, dado  $\langle M, w \rangle$  en su entrada, construya el código para la TM  $M'$  que acepta a  $\emptyset$  si  $w$  no se encuentra en  $L(M)$  y acepta a  $(0 + 1)^*$  en cualquier otro caso. La construcción de  $M'$  se dio en el Ejemplo 8.2.  $M^{S_1}$  entonces accesa al estado  $q$ , con el código para  $M$  a la derecha de su cabeza y acepta si y sólo si se accesa  $q_n$ . Por consiguiente, el problema de la membresía para las TMs sin oráculo es recursivo en  $S_1$ .

De manera inversa, podemos demostrar que existe una máquina de Turing con el problema de la membresía como oráculo, que reconoce a  $S_1$ . (Estrictamente hablando, el oráculo es  $L_u$ .) Para demostrar que  $S_1$  es recursivo en  $L_u$ , construye una TM  $M_2$  que, dado  $\langle M \rangle$ , construye una nueva TM  $M'$  que funciona de la manera siguiente:  $M'$  ignora su entrada; en su lugar,  $M'$  utiliza al generador de pares para generar a todos los pares  $(i, j)$ . Cuando  $(i, j)$  es generado,  $M'$  simula a  $M$  para  $i$  pasos sobre la  $j$ -ésima palabra de entrada para  $M$ ; las palabras se numeran en el orden usual. Si  $M$  acepta,  $M'$  acepta

su propia entrada. Si  $L(M) = \emptyset$ , entonces  $L(M') = \emptyset$ . Si  $L(M) \neq \emptyset$ , entonces  $M'$  acepta a todas sus entradas,  $\epsilon$  inclusive. Por tanto,  $M_2^{L_2}$  puede interrogar a su oráculo sobre si  $M'$  acepta a  $\epsilon$ , es decir, sobre si  $\langle M', \epsilon \rangle$  está en  $L_2$ . Si esto sucede,  $M_2$  rechaza a  $M$ . De otra forma  $M_2$  acepta a  $M$ . Por consiguiente  $S_1$  es recursivo en  $L_2$ .  $\square$

En seguida, considérese el problema de si  $L(M) = \Sigma^*$ , en donde  $\Sigma$  es el alfabeto de entrada para la TM  $M$ . En un cierto sentido, este problema es más “difícil” que el de la membresía o la vacuidad, porque, como veremos más adelante, el problema “ $= \Sigma^*$ ” es equivalente a  $S_2$ , mientras que el problema de vacuidad y el de la membresía son equivalentes a  $S_1$ . Mientras que esta diferencia no significa nada en términos prácticos, puesto que todos estos problemas son irresolubles, los resultados con respecto al grado comparativo de dificultad sugieren que, cuando consideramos versiones restringidas de los problemas, el problema “ $= \Sigma^*$ ” en realidad es más difícil que el de la membresía o de la vacuidad. Para gramáticas libres de contexto, los problemas de vacuidad de membresía son resolubles, mientras que, según el Teorema 8.11, el problema sobre si  $L(G) = \Sigma^*$  es irresoluble. Como otro ejemplo, considérense las expresiones regulares. Los problemas de vacuidad y membresía son, cada uno, resolubles de manera eficiente, toman un tiempo que es un polinomio con respecto a la longitud de la expresión, mientras que el problema sobre si una expresión regular dada  $r$  es equivalente a  $\Sigma^*$  se ha demostrado casi con toda certeza que requiere un tiempo que es exponencial en la longitud de  $r$ .†

**Teorema 8.20** El problema sobre si  $L(M) = \Sigma^*$  es equivalente a  $S_2$ .

**Demostración** Construimos una TM  $M_3^{S_2}$  que toma una TM  $M$  cualquiera y construye a partir de ella a  $\hat{M}^{S_1}$ , una TM con oráculo  $S_1$  que se comporta de la forma siguiente:  $\hat{M}^{S_1}$  enumera las palabras  $x$ , y para cada  $x$  utiliza el oráculo  $S_1$  para decir si  $M$  acepta a  $x$ . La técnica mediante la cual  $S_1$  puede ser utilizado para responder la cuestión de la membresía fue cubierta por el Teorema 8.19.  $\hat{M}^{S_1}$  acepta a su propia entrada si cualquier  $x$  no es aceptada por  $M$ . Por consiguiente,

$$L(\hat{M}^{S_1}) = \begin{cases} \emptyset & \text{si } L(M) = \Sigma^*, \\ \Sigma^* & \text{en cualquier otro caso.} \end{cases}$$

$M_3^{S_2}$  con entrada  $M$  construye a  $M_1^{S_1}\dagger\dagger$ , entonces pregunta a su propio oráculo,  $S_2$ , si  $L(\hat{M}^{S_1}) = \emptyset$ . Si esto es cierto,  $M_3^{S_2}$  acepta a  $M$ , y  $M_3^{S_2}$  rechaza en cualquier otro caso. Por tanto  $M_3^{S_2}$  acepta a  $\{\langle M \rangle \mid L(M) = \Sigma^*\}$ .

Ahora debemos mostrar que  $S_2$  es recursivo en el problema “ $= \Sigma^*$ ”. Esto es, sea  $L_*$  el conjunto de códigos de las máquinas de Turing ordinarias que aceptan a todas las cadenas sobre su alfabeto de entrada. Entonces existe una TM  $M_4^{L_*}$  que acepta a  $S_2$ .

Antes de construir a  $M_4^{L_*}$ , primero definimos un cálculo válido de una TM  $M^{S_1}$  que utiliza el oráculo  $S_1$ . Un cálculo válido es una sucesión de IDs, justo como se definió para las máquinas de Turing ordinarias. Sin embargo, si una ID tiene el estado  $q$ , y la

† Técnicamente, el problema es “completo en el espacio polinomial”; Véase el Capítulo 13.

†† Nótese que  $S_1$  no es parte de  $\hat{M}$ . En realidad  $M_3^{S_2}$  construye las transiciones de estado de la máquina oráculo  $M$ , la cual trabajará correctamente dado  $S_1$  como oráculo.

siguiente ID tiene el estado  $q_n$ , entonces  $M^{S_1}$  ha preguntado al oráculo si alguna TM  $M$  acepta a  $\emptyset$  y obtenido la respuesta de “no”. Para demostrar que esta respuesta es correcta, insertamos un cálculo válido de la TM ordinaria  $N$ , mostrando que  $N$  acepta alguna entrada particular. Si el siguiente estado es  $q$ , sin embargo, no insertamos ningún cálculo de  $N$ .

Ahora, describamos cómo  $M_4^{L_*}$  se comporta con la entrada  $M^{S_1}$ .  $M_4^{L_*}$  crea una TM ordinaria  $M'$  que acepta a todos los cálculos no válidos de  $M^{S_1}$ . Para comprobar que una cadena no es un cálculo válido,  $M'$  verifica si el formato es inválido (como se hizo en el Lema 8.7) o si una ID de  $M^{S_1}$  no sigue, en un movimiento, a la ID anterior de  $M^{S_1}$  en la sucesión, o si un cálculo de una TM ordinaria  $N$  insertada entre dos IDs de  $M^{S_1}$  con estados  $q$ , y  $q_n$  no es válido.

La única parte difícil que se tiene que verificar es cuando una ID de  $M^{S_1}$  tiene estado  $q_j$ , y la siguiente ID tiene estado  $q_i$ . Entonces  $M'$  debe determinar si “sí” es la respuesta correcta, de modo que estas dos IDs no se siguen en secuencia. Sea  $N$  la TM acerca de la cual se hace la interrogación.  $M'$  utiliza el generador de pares y, cuando se genera  $(i, j)$ , simula a  $N$  para  $i$  pasos en la  $j$ -ésima entrada. Si  $N$  acepta,  $M'$  determina que  $L(N) \neq \emptyset$ , de modo que “sí” es la respuesta equivocada. Por tanto el cálculo no es válido y  $M'$  acepta a este cálculo.

Ahora bien,  $M'$  acepta a todas las cadenas sobre su alfabeto de entrada si y sólo si  $L(M^{S_1}) = \emptyset$ , esto es,  $M^{S_1}$  no tiene cálculos válidos.  $M_4^{L_*}$  puede interrogar a su oráculo sobre si  $M'$  acepta o no a  $\Sigma^*$ . El código para  $M^{S_1}$  está en  $S_2$ , si y sólo si  $L(M') = \Sigma^*$ . Por consiguiente  $S_2$  es recursivo en  $L_*$ .  $\square$

## Reducibilidad de Turing

A lo largo del presente capítulo, hemos tratado con un concepto al que llamamos “reducibilidad”, con el cual reducimos el lenguaje  $L_1$  a  $L_2$ , encontrando un algoritmo que transforme cadenas de  $L_1$  a cadenas de  $L_2$  que no se encuentren en  $L_2$ . Este concepto de reducibilidad con frecuencia se conoce como *reducibilidad muchos a uno*, y mientras que representa todo lo que necesitamos, no es el concepto más general. Una técnica más general se conoce como *reducibilidad de Turing*, y consiste simplemente en demostrar que  $L_1$  es recursivo en  $L_2$ .

Si  $L_1$  es reducible muchos a uno en  $L_2$ , entonces, seguramente,  $L_1$  es Turing reducible a  $L_2$ . Para su demostración, supóngase que  $f$  es una función calculable mediante una TM que siempre se detiene, tal que  $f(x)$  está en  $L_2$  si y sólo si  $x$  está en  $L_1$ . Considérese entonces la TM con oráculo  $M^{L_2}$  que, dada la entrada  $x$ , calcula  $f(x)$  y después accesa el estado  $q_f$  con  $f(x)$  a la derecha de su cabeza.  $M^{L_2}$  acepta si y sólo si enseguida accesa a  $q_f$ . Seguramente  $L(M^{L_2}) = L_1$ , así que  $L_1$  es Turing reducible a  $L_2$ . La parte inversa es falsa, y se sugiere una demostración en los ejercicios.

Si  $L_1$  es Turing reducible a  $L_2$  y  $L_1$  es no irresoluble, entonces  $L_2$  también es irresoluble. Porque si  $L_2$  fuera recursivo, entonces la TM oráculo  $M^{L_2}$ , tal que  $L(M^{L_2}) = L_1$ , puede ser simulada por una TM ordinaria que siempre se detiene. Por tanto uno podría utilizar una reducción de Turing para demostrar que  $L_2$  es no irresoluble, dado que  $L_1$  es irresoluble, aun en circunstancias en las que no exista una reducción muchos a uno de  $L_1$  a  $L_2$ , o es difícil encontrarla.

El concepto de reducibilidad muchos a uno tiene sus virtudes, sin embargo. Si  $L_1$  es reducible muchos a uno a  $L_2$ , y  $L_1$  no es r.e., podemos concluir que  $L_2$  es no r.e. Aunque esta conclusión no puede ser alcanzada con la reducibilidad de Turing. Por ejemplo,  $L_u$  es un lenguaje no r.e. que es Turing reducible al lenguaje r.e.  $L_u$ . Podemos reconocer a  $L_u$  dado  $L_u$  como oráculo, si preguntamos si  $\langle M, w \rangle$  está en  $L_u$  y aceptando si y sólo si la respuesta es no.

Vemos que la forma más difícil de reducibilidad (muchos a uno) nos permite llegar a conclusiones que no podemos alcanzar con la forma más sencilla de reducibilidad (Turing). En el Capítulo 13, en donde estudiaremos la reducibilidad acotada, veremos más ejemplos de cómo las formas más difíciles de reducción producen conclusiones que no se pueden obtener mediante formas más simples.

## EJERCICIOS

**8.1** Supóngase que los alfabetos de cinta de todas las máquinas de Turing son seleccionados a partir de algún conjunto infinito de símbolos  $a_1, a_2, \dots$ . Muestre la forma en que cada TM puede codificarse como una cadena binaria.

**8.2** ¿Cuáles de las siguientes propiedades de los conjuntos r.e. son ellas mismas r.e.?

- a)  $L$  contiene al menos dos cadenas.
- b)  $L$  es infinito
- c)  $L$  es un lenguaje libre de contexto.
- d)  $L = L^R$ .

**8.3** Demuestre que es irresoluble el hecho de si una TM se detiene en todas las entradas.

**8.4** Un sistema de Estampilla es un conjunto finito  $P$  de pares  $(\alpha, \beta)$  tomados de algún alfabeto finito y unas cadenas inicial  $\gamma$ . Decimos que  $\alpha \xrightarrow{*} \delta \beta$  si  $(\alpha, \beta)$  es un par. Definimos  $\xrightarrow{*}$  como la cerradura reflexiva y transitiva de  $\Rightarrow$ , como se hizo para las gramáticas. Muéstrese que, para un sistema de estampilla  $(P, \gamma)$  y una cadena  $\delta$  dados, es no soluble el hecho de si  $\gamma \xrightarrow{*} \delta$ . [Sugerencia: Para cada TM  $M$  sea  $\gamma$  la ID inicial de  $M$  con cinta en blanco, seguida por un señalador #, y seleccione los pares de manera que cualquier ID deba convertirse en la siguiente ID después de una sucesión de aplicaciones de las reglas, a menos que esa ID tenga un estado de aceptación, en cuyo caso la ID puede volverse, en algún momento,  $\epsilon$ . Entonces pregúntese si  $\gamma \xrightarrow{*} \epsilon$ .]

**8.5** Muestre que no existe un algoritmo que, dada una TM  $M$  que define una función parcialmente recursivas  $f$  de una variable, produzca una TM  $M'$  que define una función diferente de una sola variable.

**\*8.6** Para las máquinas de Turing ordinarias  $M$  muestre que

- a) el problema de determinar si  $L(M)$  es finito es equivalente a  $S_2$ ;
- b) el problema de determinar si  $L(M)$  es un conjunto regular es equivalente a  $S_3$ .

**8.7** Demuéstrese que los siguientes problemas acerca de programas en los lenguajes de programación real son irresolubles.

- a) Si un programa dado puede entrar en un ciclo y no salir de él para alguna entrada.
- b) Si un programa dado nunca produce una salida.
- c) Si dos programas producen la misma salida en todas las entradas.

**8.8** Utilice el Teorema 8.14 para demostrar que las siguientes propiedades de los CFLs son irresolubles.

- a)  $L$  es un lenguaje lineal.
- b)  $L$  es un CFL.

**S8.9** Muestre que el Teorema 8.14 se aplica a los lenguajes lineales. [Sugerencia: Consulte el Teorema 9.2 para una demostración sobre el hecho de que cada conjunto regular tiene una gramática lineal. La parte difícil consiste en mostrar que “ $= \Sigma^*$ ” es irresoluble para lenguajes lineales].

**\*8.10** Demuestre que las siguientes propiedades de los lenguajes lineales son irresolubles. Puede utilizar el hecho de que cada conjunto regular es un lenguaje lineal.

- a)  $L$  es un conjunto regular.
- b)  $L$  es un lenguaje lineal.
- c)  $L$  es un CFL
- d)  $L$  no tiene una CFG lineal no ambigua.

**\*8.11** Muestre que para el CFL  $L$ , es irresoluble el hecho de si  $L = L^R$ .

**\*8.12**

- a) Muestre que si  $L_1$  es reducible muchos a uno a  $L_2$  y  $L_2$  es (i) recursivo en  $L_3$  o (ii) r.e. en  $L_3$ , entonces  $L_1$  es recursivo r.e. en  $L_3$ , respectivamente.
- b) Muestre que  $L_u$  es Turing reducible a  $S_1$ .
- c) Muestre que  $L_u$  no es reducible muchos a uno a  $S_1$ . [Sugerencia: Utilice la parte (a).]

**8.13** Decimos que  $L_1$  se reduce mediante una “tabla de verdad” a  $L_2$  si:

1. Si existen  $k$  algoritmos que transforman a cada cadena  $x$  sobre el alfabeto de  $L_1$  a cadenas sobre el alfabeto de  $L_2$ . Sea  $g_i(x)$  el resultado de aplicar el  $i$ -ésimo algoritmo a  $x$ .
2. Existe una función booleana  $f(y_1, \dots, y_k)$  tal que si  $y_i$  es verdadera cuando  $g_i(x)$  está en  $L_2$  y  $y_i$  es falsa en cualquier otro caso, entonces  $f(y_1, \dots, y_k)$  es verdadera si y sólo si  $x$  está en  $L_1$ .

Por ejemplo, sea  $L_1$  el conjunto de cadenas con igual número de 0s y 1s y sea  $L_2$  el conjunto de cadenas con no menos 0s que 1s. Sea  $g_1(x) = x$  y  $g_2(x)$  la función formada a partir de  $x$  mediante la sustitución de 0s por 1s y viceversa. Sea  $f(y_1, y_2) = y_1 \wedge y_2$ . Entonces  $f(y_1, y_2)$  es verdadera si y sólo si  $g_1(x)$  y  $g_2(x)$ , ambos, no tienen menos 0s que 1s; esto es,  $x$  tiene un número igual de 0s y 1s. Por consiguiente  $L_1$  se reduce mediante una tabla de verdad a  $L_2$ .

- a) Muestre que si  $L_1$  se reduce mediante una tabla de verdad a  $L_2$ , entonces  $L_1$  es Turing reducible a  $L_2$ .
- b) Muestre que si  $L_1$  es reducible muchos a uno a  $L_2$ , entonces  $L_1$  se reduce mediante una tabla de verdad a  $L_2$ .

c) Muestre que  $\bar{L}_u$  se reduce mediante una tabla de verdad a  $S_1$ .

**8.14** Considérese una TM multicinta con oráculo que, cuando consulta a su oráculo, se refiere al contenido completo de una cinta designada, digamos la última. Muestre que este modelo es equivalente al oráculo TM como se definió en la Sección 8.9.

**8.15** Demuestre que el PCP es resoluble para palabras sobre un alfabeto de un solo símbolo.

**8.16** Demuestre que el PCP es equivalente a  $S_1$ .

\***8.17.** Muestre que el PCP es irresoluble si las cadenas se restringen a tener longitud uno o dos. ¿Qué sucede si las cadenas son restringidas a tener una longitud de exactamente dos?

\***8.18** Sea  $\sigma$  una función totalmente recursiva que transforma índices de funciones parcialmente recursivas. Dé un algoritmo para enumerar un conjunto infinito de puntos fijos de  $\sigma$ ; esto es, infinitamente tantas  $i$  tales que  $f_i(y) = f_{\sigma(i)}(y)$  para toda  $y$ .

\***8.19** ¿Existe una enumeración efectiva de las máquinas de Turing  $M_1, M_2, \dots$  tales que no haya tres TMs consecutivas que calculen la misma función?

#### Soluciones a los ejercicios seleccionados

**8.3** Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$  una TM. Construimos otra TM  $M'$ , tal que  $M'$  se detenga en  $x$  si y sólo si  $M$  acepta a  $x$ . Por consiguiente tendremos que demostrar que la cuestión de si una TM se detiene en todas las entradas que reduce a la cuestión de si una TM acepta todas las entradas, de la cual sabemos que es irresoluble. De paso, también mostraremos, mediante esta construcción, que una cuestión como “¿Se detiene una TM en una entrada dada?” o “¿Una TM se detiene en alguna entrada?”, es también irresoluble.

$M'$  se diseña para comportarse de la manera siguiente. Primero, corre su entrada una posición hacia la derecha, colocando un señalador de extremo izquierdo  $\$$  en la celda que se encuentra más a la izquierda. Despues  $M'$  simula a  $M$ . Si  $\delta(q, X)$  no está definido y sucede que (i)  $q$  no es de aceptación y  $X$  es cualquier símbolo de  $\Gamma \cup \{\$\}$  [ Nótese que  $\delta(q, \$)$  seguramente no está definida] o (ii)  $q$  es de aceptación y  $X$  es  $\$$ , entonces  $M'$ , barriendo a  $X$  en el estado  $q$ , se mueve hacia la derecha y accesa al estado  $p_1$ . En el estado  $p_1$ , barriendo cualquier símbolo,  $M'$  se mueve hacia la izquierda y accesa al estado  $p_2$ ; en ese estado,  $M'$  se mueve hacia la derecha y accesa de nuevo al estado  $p_1$ . Por consiguiente,  $M'$  entra en un ciclo sin salir de él si  $M$  se detiene en un estado de no aceptación o se sale del extremo izquierdo de la cinta en cualquier estado. Si  $M$  accesa un estado de aceptación, sin barrer a  $\$$ , entonces  $M'$  se detiene. De esta manera,  $M'$  se detiene si y sólo si  $M$  acepta su entrada, como se desea.

**8.9** Primero debemos demostrar que los lenguajes lineales son cerrados con respecto a la unión y la concatenación con conjuntos regulares. Consideraremos el Teorema 9.2, que nos da una demostración de que cada conjunto regular es generado por la CFG cuyas producciones todas son de las formas  $A \rightarrow Bw$  y  $A \rightarrow w$  para las no terminales  $A$  y  $B$  y la cadena de terminales  $w$ . Cualquier gramática con estas características seguramente es lineal. La demostración de que los lenguajes lineales son cerrados con respecto a la unión es justamente como la del Teorema 6.1. Para la concatenación con un conjunto regular, sea  $G_1 = (V_1, T_1, P_1, S_1)$  una gramática lineal y  $G_2 = (V_2, T_2, P_2, S_2)$  una gramática con todas las producciones de las formas  $A \rightarrow Bw$  y  $A \rightarrow w$ . Suponga que  $V_1$  y  $V_2$  son disjuntos. Sea

$$G = (V_1 \cup V_2, T_1 \cup T_2, P, S_2),$$

en donde  $P$  consiste en

- i) todas las producciones  $A \rightarrow Bw$  de  $P_2$ ,
- ii) la producción  $A \rightarrow S_1w$  cuando  $A \rightarrow w$  es una producción de  $P_2$  y
- iii) todas las producciones de  $P_1$

Entonces es fácil ver que  $L(G)$  es  $L(G_1)L(G_2)$ , ya que todas las derivaciones de  $G$  son de la forma  $S_2 \xrightarrow{*} S_1x \xrightarrow{*} yx$ , en donde  $S_2 \xrightarrow{*} x$  y  $S_1 \xrightarrow{*} y$ . Como los conjuntos regulares y los lenguajes lineales son cerrados con respecto a la inversión, la concatenación en la izquierda mediante un conjunto regular se concluye de manera parecida.

Ahora debemos demostrar que “ $= \Sigma^*$ ” es irresoluble para los lenguajes lineales. La demostración es casi paralela a la del Lema 8.7 y del Teorema 8.11, el resultado análogo para las CFGs generales. La diferencia importante es que debemos redefinir la forma de los cálculos válidos de modo que el conjunto de cálculos no válidos sea una CFG lineal. Definamos un cálculo válido de TM  $M$  como la cadena

$$w_1 \# w_2 \# \cdots \# w_{n-1} \# w_n \# \# w_n^R \# w_{n-1}^R \# \cdots \# w_2^R \# w_1^R, \quad (8.4)$$

en donde cada  $w_i$  es una ID,  $w_i \xrightarrow{M} w_{i+1}$  para  $1 \leq i < n$ ,  $w_1$  es una ID inicial y  $w_n$  es una final. Entonces no es difícil construir una gramática lineal para las cadenas que no son de la forma (8.4), siguiendo las ideas del Lema 8.7. Entonces la parte análoga al Teorema 8.11 muestra que “ $= \Sigma^*$ ” es irresoluble para gramáticas lineales.

#### NOTAS BIBLIOGRAFICAS

La irresolubilidad de  $\bar{L}_u$  es el resultado básico de Turing [1936]. Los Teoremas 8.6 y 8.7, que caracterizan a los conjuntos recursivos y r.e., se tomaron de Rice [1953, 1956]. El problema de correspondencia de Post fue demostrado como irresoluble en Post [1946] y la demostración de irresolubilidad que se utilizó aquí se tomó de Floyd [1964]. Los Lemas 8.6 y 8.7, que relacionan a los cálculos de las máquinas de Turing con las CFGs, son de Hartmanis [1967].

Los artículos fundamentales sobre las propiedades irresolubles de los CFLs se tomaron de Bar Hillel, Perles y Shamir [1961] y de Ginsburg y Rose [1963a]. Sin embargo, el Teorema 8.9, sobre la ambigüedad, fue demostrado independientemente por Cantor [1962], Floyd [1962a], Chomsky y Schultzenberger [1963]. El Teorema 8.16, irresoluble de la ambigüedad inherente, se tomó de Ginsburg y Ullian [1966]. Las gramáticas lineales y sus propiedades de decisión han sido estudiadas por Greibach [1963, 1966] y Gross [1964]. El planteamiento utilizado en la solución al Ejercicio 8.9 es de Baker y Book 1974.

El teorema de Greibach se tomó de Greibach [1968]. Una generalización de éste aparece en Hunt y Rosenkrantz [1974], la cual incluye una solución al Ejercicio 8.11. Hopcroft y Ullman [1968a] muestran que para ciertas clases de lenguajes definidos por autómatas, la resolubilidad de la membresía y de vacuidad están relacionadas. Los teoremas  $S_m$  y de recursión se tomaron de Kleene [1952]. El ejemplo 8.10, sobre la no existencia de demostraciones de detenimiento o no de toda las TMs, es de Hartmanis y Hopcroft [1976].

Hartmanis y Hopcroft [1968] son los autores del artículo básico que relaciona los problemas acerca de los CFLs con la jerarquía de los problemas irresolubles. Los Teoremas 8.19 y 8.20, así como el Ejercicio 8.6, se tomaron de ese artículo. Resultados adicionales de este tipo han sido

obtenidos por Cudia y Singletary [1968], Cudia [1970], Hartmanis [1969] y Reedy y Savitch [1975]. El Ejercicio 8.4, sobre los sistemas de estampilla, se tomó de Minsky [1961].

## CAPITULO

## 9

## LA JERARQUIA DE CHOMSKY

De las tres clases principales de lenguajes que hemos estudiado, los conjuntos regulares, los lenguajes libres de contexto y los lenguajes enumerables de manera recursiva, hemos caracterizado gramaticalmente sólo a los CFLs. En este capítulo daremos definiciones gramaticales de los conjuntos regulares y de los lenguajes r.e. También introduciremos una nueva clase de lenguajes, que se encuentran entre los CFs y los lenguajes r.e., y daremos máquinas y caracterizaciones gramaticales para esta nueva clase. Con frecuencia a estas cuatro clases de lenguajes se les conoce como *jerarquía de Chomsky*, ya que fue Noam Chomsky quien definió a estas clases como modelos potenciales de lenguajes naturales.

## 9.1 GRAMATICAS REGULARES

Si todas las producciones de una CFG son de la forma  $A \rightarrow wB$  o  $A \rightarrow w$ , en donde  $A$  y  $B$  son variables y  $w$  es una cadena (posiblemente vacía) de terminales, entonces decimos que la gramática es *lineal por la derecha*. Si todas las producciones son de la forma  $A \rightarrow Bw$  o  $A \rightarrow w$ , la llamamos *lineal por la izquierda*. Las gramáticas lineales por la derecha o la izquierda se conocen como *gramáticas regulares*.

**Ejemplo 9.1** El lenguaje  $0(10)^*$  es generado por una gramática lineal por la derecha

$$S \rightarrow 0A \quad (9.1)$$

$$A \rightarrow 10A \mid \epsilon$$

y por la gramática lineal por la izquierda

$$S \rightarrow 10 \mid 0 \quad (9.2)$$

## Equivalencia de las gramáticas regulares y los autómatas finitos

Las gramáticas regulares caracterizan a los conjuntos regulares, en el sentido de que un lenguaje es regular si y sólo si tiene una gramática lineal por la izquierda y si y sólo si tiene una gramática lineal por la derecha y si sólo si tiene una gramática lineal por la derecha. Estos resultados se demuestran en los dos siguientes teoremas.

**Teorema 9.1** Si  $L$  tiene una gramática regular, entonces  $L$  es un conjunto regular.

**Demostración** Primero, supóngase que  $L = L(G)$  para alguna gramática lineal por la derecha  $G = (V, T, P, S)$ . Construimos un NFA con movimientos  $\epsilon$ ,  $M = (Q, T, \delta, [S], \{[\epsilon]\})$  que simula las derivaciones de  $G$ .

$Q$  consiste en los símbolos  $[\alpha]$  tales que  $\alpha$  es  $S$  o un sufijo (no necesariamente propio) del lado derecho de alguna producción de  $P$ .

Definimos  $\delta$  mediante:

1. Si  $A$  es una variable, entonces  $\delta([A], \epsilon) = \{[\alpha] \mid A \rightarrow \alpha \text{ es una producción}\}$ .
2. Si  $a$  se encuentra en  $T$  y  $\alpha$  en  $T^* \cup T^*V$ , entonces  $\delta([a\alpha], a) = \{[\alpha]\}$ .

Entonces una inducción sencilla sobre la longitud de una derivación o sucesión de movimientos muestra que  $\delta([S], w)$  contiene a  $[\alpha]$  si y sólo si  $S \xrightarrow{*} xA \Rightarrow xy\alpha$ , en donde  $A \rightarrow y\alpha$  es una producción y  $xy = w$ , o si  $\alpha = S$  y  $w = \epsilon$ . Puesto que  $[\epsilon]$  es el único estado final,  $M$  acepta a  $w$  si y sólo si  $S \xrightarrow{*} xA \Rightarrow w$ . Pero como cada derivación de una cadena terminal tiene al menos un paso, vemos que  $M$  acepta a  $w$  si y sólo si  $G$  genera a  $w$ . De aquí que cada gramática lineal por la derecha genera a un conjunto regular.

Ahora hagamos  $G' = (V, T, P', S)$  una gramática lineal por la izquierda. Sea  $G' = (V, T, P', S)$ , en donde  $P'$  consiste en las producciones de  $G$  con el lado derecho invertido, esto es,

$$P' = \{A \rightarrow \alpha \mid A \rightarrow \alpha^R \text{ está en } P\}.$$

Si invertimos las producciones de una gramática lineal por la izquierda conseguimos una gramática lineal por la derecha, y viceversa. Por tanto,  $G'$  es una gramática lineal por la derecha, y es fácil probar que  $L(G')^R = L(G)^R$ . Según el párrafo anterior,  $L(G')$  es un conjunto regular. Pero los conjuntos regulares son cerrados con respecto a la inversión (Ejercicio 3.4g), así que  $L(G')^R = L(G)$  es también un conjunto regular. Por consiguiente cada gramática lineal por la derecha o por la izquierda define a un conjunto regular.  $\square$

**Ejemplo 9.2** El NFA construido por el Teorema 9.1 a partir de la gramática (9.1) se muestra en la Fig. 9.1

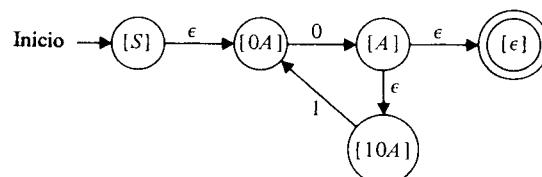


Fig. 9.1 NFA que acepta a  $0(10)^*$ .

Ahora considérese la gramática (9.2). Si invertimos sus producciones obtenemos

$$S \rightarrow 01S \mid 0$$

La construcción del Teorema 9.1 para esta gramática produce un NFA de la Fig. 9.2 (a). Si invertimos las aristas de ese NFA e intercambiamos los estados inicial y final, obtenemos otro NFA para  $0(10)^*$ .

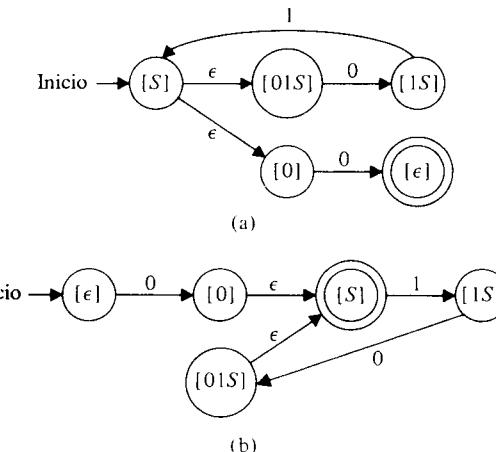


Fig. 9.2 Construcción de un NFA para  $0(10)^*$  a partir de una gramática lineal por la izquierda.

**Teorema 9.2** Si  $L$  es un conjunto regular, entonces  $L$  es generado por alguna gramática lineal por la derecha.

**Demostración** Sea  $L = L(M)$  para el DFA  $M = (Q, \Sigma, \delta, q_0, F)$ . Supóngase, primero que  $q_0$  no es un estado final. Entonces  $L = L(G)$  para la gramática lineal por la derecha  $G = (Q, \Sigma, P, q_0)$ , en donde  $P$  consiste en la producción  $p \rightarrow aq$  siempre que  $\delta(p, a) = q$  y también  $p \rightarrow a$  siempre que  $\delta(p, a)$  sea un estado final. Entonces es claro que  $\delta(p, w) = q$  si y sólo si  $p \xrightarrow{*} wq$ . Si  $wa$  es aceptada por  $M$ , sea  $\delta(q_0, w) = p$ , que implica que  $q_0 \xrightarrow{*} wp$ . También,  $\delta(p, a)$  es final, de modo que  $p \rightarrow a$  es una producción. Por consiguiente  $q_0 \xrightarrow{*} wa$ . De manera inversa, sea  $q_0 \xrightarrow{*} x$ . Entonces  $x = wa$  y  $q_0 \xrightarrow{*} wp \Rightarrow wa$  para algún estado (variable)  $p$ . Entonces  $\delta(q_0, w) = p$  y  $\delta(p, a)$  es final. Por tanto  $x$  se encuentra en  $L(M)$ . En consecuencia  $L(M) = L(G) = L$ .

Sea, ahora,  $q_0$  un estado de  $F$ , de manera que  $\epsilon$  se encuentra en  $L$ . Notamos que la gramática definida más arriba genera a  $L - \{\epsilon\}$ . Podemos modificar a  $G$  mediante la adición de un nuevo símbolo de inicio  $S$  con producciones  $S \rightarrow q_0 | \epsilon$ . La gramática resultante sigue siendo lineal por la derecha y genera a  $L$ .

Para producir una gramática lineal por la izquierda para  $L$ , comenzamos con un NFA para  $L^R$  y entonces invertimos el lado derecho de todas las producciones de la gramática lineal por la derecha resultante.  $\square$

**Ejemplo 9.3** En la Fig. 9.3 vemos un DFA para  $0(10)^*$ .

La gramática lineal por la derecha de este DFA es

$$\begin{array}{l} A \rightarrow 0B \mid 1D \mid 0 \\ B \rightarrow 0D \mid 1C \\ C \rightarrow 0B \mid 1D \mid 0 \\ D \rightarrow 0D \mid 1D \end{array}$$

Como  $D$  es inútil podemos eliminarla, obteniendo la gramática

$$\begin{array}{l} A \rightarrow 0B \mid 0 \\ B \rightarrow 1C \\ C \rightarrow 0B \mid 0 \end{array}$$

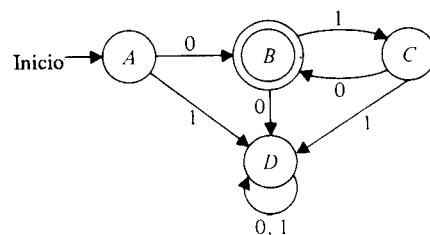


Fig. 9.3 DFA para  $0(10)^*$ .

## 9.2 GRAMATICAS NO RESTRINGIDAS

La familia más grande de gramáticas en la jerarquía de Chomsky permite producciones de la forma  $\alpha \rightarrow \beta$ , en donde  $\alpha$  y  $\beta$  son cadenas arbitrarias de símbolos de gramática, en donde  $\alpha \neq \epsilon$ . Estas gramáticas se conocen como *gramáticas semi-Thue, tipo 0, de estructura de frase o no restringidas*. Continuaremos utilizando la notación de 4 parámetros  $G = (V, T, P, S)$  para las gramáticas no restringidas. Decimos que  $\gamma \alpha \delta \Rightarrow \gamma \beta \delta$  siempre que  $\alpha \rightarrow \beta$  sea una producción. Como se hizo antes,  $\Rightarrow^*$  representa la cerradura reflexiva y transitiva de la relación  $\Rightarrow$ :

$$L(G) = \{w \mid w \text{ está en } T^* \text{ y } S^* \Rightarrow^* w\},$$

exactamente como se tiene para las gramáticas libres de contexto.

**Ejemplo 9.4** Una gramática que genera al conjunto  $\{a^i \mid i \text{ es una potencia positiva de } 2\}$ ; se da a continuación:

- |                         |                              |
|-------------------------|------------------------------|
| 1. $S \rightarrow ACaB$ | 5. $aD \rightarrow Da$       |
| 2. $Ca \rightarrow aaC$ | 6. $AD \rightarrow AC$       |
| 3. $CB \rightarrow DB$  | 7. $aE \rightarrow Ea$       |
| 4. $CB \rightarrow E$   | 8. $AE \rightarrow \epsilon$ |

$A$  y  $B$  sirven como señaladores de extremo izquierdo y derecho para las formas oracionales;  $C$  es un señalador que se mueve a través de la cadena de  $as$  que se encuentra entre  $A$  y  $B$ , doblando su número mediante la producción (2). Cuando  $C$  alcanza al señalador izquierdo  $B$ , se convierte en  $D$  o  $E$  mediante la producción (3) o (4). Si se escoge una  $D$ , esa  $D$  se traslada hacia la izquierda mediante la producción (5) hasta que se alcanza el señalador del extremo de la izquierda  $A$ . En este punto  $D$  se convierte en  $C$  de nuevo mediante la producción (6) y el proceso se reinicia. Si se escoge una  $E$ , entonces se consume el señalador del extremo de la derecha. La  $E$  se traslada hacia la izquierda a través de la producción (7) y consume al señalador izquierdo, dejando una cadena de  $2^i as$  para alguna  $i > 0$ . Podemos demostrar, por inducción en el número de pasos de la derivación, que si nunca se utiliza la producción (4), entonces cualquiera de las formas oracionales es uno de los siguientes:

- i)  $S$ ,
- ii) de la forma  $Aa^i Ca^j B$ , en donde  $i + 2j$  es una potencia positiva de 2, o
- iii) de la forma  $Aa^i Da^j B$ , en donde  $i + j$  es una potencia positiva de 2.

Cuando utilizamos la producción (4) nos quedamos con una forma oracional  $A a^i E$ , en la cual  $i$  es una potencia positiva de 2. Entonces los únicas pasos posibles de una derivación son  $i$  aplicaciones de (7) que producirán  $AEa^i$ , seguida de una aplicación de (8), produciéndose la oración  $a^i$ , en donde  $i$  es una potencia positiva de 2.

## Equivalencia de las gramáticas tipo 0 y las máquinas de Turing

Probaremos, en los siguientes dos teoremas, que las gramáticas no restringidas caracterizan a los lenguajes r.e. El primer teorema establece que cada lenguaje tipo 0 genera un conjunto r.e. Una demostración sencilla consistiría en dar un algoritmo para enumerar a todas las cadenas generadas por una gramática tipo 0. En lugar de hacer lo anterior, construiremos una máquina de Turing que reconoce las oraciones generadas por una gramática tipo 0, ya que esta construcción será de utilidad más adelante, para hacer una demostración parecida acerca de las gramáticas sensibles al contexto (la clase restante de la jerarquía de Chomsky).

**Teorema 9.3** Si  $L$  es  $L(G)$  para una gramática no restringida  $G = (V, T, P, S)$ , entonces  $L$  es un lenguaje r.e.

**Demostración** Construyamos una máquina de Turing no determinística de dos cintas  $M$  para reconocer a  $L$ . La primera cinta de  $M$  es la entrada sobre la que se puede colocar una cadena  $w$ . La segunda se utiliza para contener a una forma oracional  $\alpha$  de  $G$ .  $M$  inicia  $\alpha$  a  $S$ . Entonces  $M$  de manera repetida hace lo siguiente:

1. Selecciona de manera no determinística una posición  $i$  en  $\alpha$ , de manera que cualquier  $i$  que se encuentre entre 1 y  $|\alpha|$  puede ser escogida. Esto es, comienza en la izquierda y, repetidamente, escoge moverse hacia la derecha o seleccionar la presente posición.
2. De manera no determinística selecciona una producción  $\beta \rightarrow \gamma$  de  $G$ .

3. Si  $\beta$  aparece iniciando en la posición  $i$  de  $\alpha$ , sustituye a  $\beta$  por  $\gamma$  en esa posición, utilizando la técnica de “corrimiento” de la Sección 7.4, corriéndose, tal vez, hacia la izquierda si  $|\gamma| < |\beta|$ .
4. Compara la forma oracional resultante con la  $w$  de la cinta 1. Si coinciden, acepta;  $w$  es una oración de  $G$ . Si no, se regresa al paso (1).

Es fácil demostrar que todas las formas oracionales, y solamente éstas, de  $G$  aparecen en la cinta 2 cuando se pone en funcionamiento el paso (4), después de una secuencia de alternativas. Por consiguiente  $L(M) = L(G) = L$ , así que  $L$  es r.e.  $\square$

**Teorema 9.4** Si  $L$  es un lenguaje r.e., entonces  $L = L(G)$  para alguna gramática no restringida  $G$ .

*Demostración* Sea  $L$  un lenguaje aceptado por la máquina de Turing  $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ . Constrúyase una gramática  $G$  que genere “de manera no determinística” dos copias de una representación de alguna palabra de  $\Sigma^*$  y después simula la acción de  $M$  en una copia. Si  $M$  acepta a la palabra, entonces  $G$  convierte la segunda copia en una cadena terminal. Si  $M$  no acepta, la derivación nunca tiene como resultado una cadena terminal.

De manera formal hagamos

$$G = (V, \Sigma, P, A_1), \quad \text{en donde } V = ((\Sigma \cup \{\epsilon\}) \times \Gamma) \cup \{A_1, A_2, A_3\}$$

y las producciones de  $P$  son:

1.  $A_1 \rightarrow q_0 A_2$
2.  $A_2 \rightarrow [a, a] A_2$  para cada  $a$  en  $\Sigma$ .
3.  $A_2 \rightarrow A_3$
4.  $A_3 \rightarrow [\epsilon, B] A_3$
5.  $A_3 \rightarrow \epsilon$
6.  $q[a, X] \rightarrow [a, Y] p$  para cada  $a$  en  $\Sigma \cup \{\epsilon\}$ , cada  $q$  en  $Q$  y cada  $X$  y  $Y$  en  $\Gamma$ , tal que  $\delta(q, X) = (p, Y, R)$ .
7.  $[b, Z] q [a, X] \rightarrow p[b, Z][a, Y]$  para cada  $X, Y$  y  $Z$  en  $\Gamma$ ,  $a$  y  $b$  en  $\Sigma \cup \{\epsilon\}$  y  $q$  en  $Q$ , tal que  $\delta(q, X) = (p, Y, L)$ .
8.  $[a, X] q \rightarrow qaq$ ,  $q[a, X] \rightarrow qaq$  y  $q \rightarrow \epsilon$  para cada  $a$  en  $\Sigma \cup \{\epsilon\}$ ,  $X$  en  $\Gamma$  y  $q$  en  $F$ .

Utilizando las reglas 1 y 2, tenemos

$$A_1 \xrightarrow{*} q_0[a_1, a_1][a_2, a_2] \cdots [a_n, a_n]A_2,$$

en donde  $a_i$  se encuentra en  $\Sigma$  para cada  $i$ . Supóngase que  $M$  acepta a la cadena  $a_1 a_2 \dots a_n$ . Entonces, para alguna  $m$ ,  $M$  no utiliza más de  $m$  celdas hacia la derecha para su entrada. Utilizando la regla 3, después la regla 4  $m$  veces y, finalmente, la regla 5, tenemos

$$A_1 \xrightarrow{*} q_0[a_1, a_1][a_2, a_2] \cdots [a_n, a_n][\epsilon, B]^m.$$

De aquí en adelante, sólo las reglas 6 y 7 se pueden utilizar hasta que se genere un estado de aceptación. Nótese que las primeras componentes de las variables de  $(\Sigma \cup \{\epsilon\}) \times \Gamma$  nunca son cambiadas. Podemos demostrar, por inducción en el número de movimientos hechos por  $M$ , que si

$$q_0 a_1 a_2 \cdots a_n \xrightarrow{*} M X_1 X_2 \cdots X_{r-1} q X_r \cdots X_s, \quad (9.3)$$

entonces

$$\begin{aligned} q_0[a_1, a_1][a_2, a_2] \cdots [a_n, a_n][\epsilon, B]^m &\xrightarrow{*} \\ [a_1, X_1][a_2, X_2] \cdots [a_{r-1}, X_{r-1}]q[a_r, X_r] \cdots [a_{n+m}, X_{n+m}], \end{aligned} \quad (9.4)$$

en donde  $a_1, a_2, \dots, a_n$  se encuentran en  $\Sigma$ ,  $a_{n+1} = a_{n+2} = \dots = a_{n+m} = \epsilon$ ,  $X_1, X_2, \dots, X_{n+m}$  se encuentran en  $\Gamma$  y  $X_{n+1} = X_{n+2} = \dots = X_{n+m} = B$ .

La hipótesis inductiva es trivialmente verdadera para cero movimientos, ya que  $r = 1$  y  $s = n$ . Supóngase que es verdadera para  $k - 1$  movimientos y sea

$$q_0 a_1 a_2 \cdots a_n \xrightarrow{k-1} M X_1 X_2 \cdots X_{r-1} q X_r \cdots X_s \xrightarrow{*} Y_1 Y_2 \cdots Y_{t-1} p Y_t \cdots Y_u.$$

Según la hipótesis inductiva,

$$\begin{aligned} q_0[a_1, a_1] \cdots [a_n, a_n][\epsilon, B]^m &\xrightarrow{*} [a_1, X_1] \cdots [a_{r-1}, X_{r-1}]q[a_r, X_r] \cdots [a_{n+m}, X_{n+m}], \\ \text{en donde las as y las Xs satisfacen las condiciones de (9.4).} \end{aligned}$$

Si  $t = r + 1$ , entonces el  $k$ -ésimo movimiento de  $M$  es hacia la derecha, de modo que  $\delta(q, X_r) = (p, Y_r, R)$ . Mediante la regla (6),  $q[a_r, X_r] \rightarrow [a_r, Y_r] p$  es una producción de  $G$ . Por consiguiente

$$\begin{aligned} q_0[a_1, a_1] \cdots [a_n, a_n][\epsilon, B]^m &\xrightarrow{*} \\ [a_1, Y_1] \cdots [a_{t-1}, Y_{t-1}]p[a_t, Y_t] \cdots [a_{n+m}, Y_{n+m}], \end{aligned} \quad (9.5)$$

en donde  $Y_i = B$  para  $i > u$ .

Si  $t = r - 1$ , entonces el  $k$ -ésimo movimiento de  $M$  es hacia la izquierda y demostramos (9.5) utilizando la regla (7) y las observaciones que consisten en que  $r > 1$  y  $\delta(q, X_r) = (p, Y_r, L)$ .

Mediante la regla (8), si  $p$  se encuentra en  $F$ , entonces

$$[a_1, Y_1] \cdots [a_{t-1}, Y_{t-1}]p[a_t, Y_t] \cdots [a_{n+m}, Y_{n+m}] \xrightarrow{*} a_1 a_2 \cdots a_n.$$

Por consiguiente, hemos demostrado que si  $w$  se encuentra en  $L(M)$ , entonces  $A_1 \xrightarrow{*} w$ , de manera que  $w$  está en  $L(G)$ .

Para la inversa, que  $w$  se encuentra en  $L(G)$  implica que  $w$  se encuentra en  $L(M)$ , una inducción parecida a la anterior demuestra que (9.4) implica a (9.3). Dejamos esta parte como ejercicio. Entonces nos damos cuenta de que no existe una forma de eliminar el estado de  $M$  a partir de las formas oracionales de  $G$  sin utilizar la regla (8). Por consiguiente  $G$  no puede derivar una cadena terminal sin simular un cálculo aceptado de  $M$ . Según la regla (8), la cadena derivada debe ser la primera componente

de las variables de  $(\Sigma \cup \{\epsilon\}) \times \Gamma$ , las cuales nunca cambian conforme se van simulando los movimientos de  $M$ .  $\square$

### 9.3 LENGUAJES SENSIBLES AL CONTEXTO

Supóngase que sobre las producciones  $\alpha \rightarrow \beta$  de una gramática de estructura de fase imponemos la restricción de que  $\beta$  sea al menos de la misma longitud de  $\alpha$ . En este caso llamamos a la gramática resultante *sensible al contexto* y a su lenguaje, *lenguaje sensible al contexto* (CSG y CSL, respectivamente). El término “sensible al contexto” viene de una forma normal para estas gramáticas, en la cual cada producción es de la forma  $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ , con  $\beta = \epsilon$ . Las producciones de la última forma parecen más producciones libres de contexto, pero permiten la sustitución de la variable  $A$  por la cadena  $\beta$  solamente en el “contexto”  $\alpha_1 - \alpha_2$ . Dejamos esta forma normal como ejercicio.

Casi cualquier lenguaje que uno pueda concebir es sensible al contexto; las únicas pruebas conocidas de que ciertos lenguajes no son CLSs están, a fin de cuentas, basadas en la diagonalización. Estos lenguajes incluyen a  $L_u$  del Capítulo 8 y los lenguajes a los cuales se puede reducir  $L_u$ , por ejemplo, los lenguajes del capítulo 8 que, se demostró son irresolubles. En la Sección 9.4 demostraremos que existen lenguajes recursivos que no son CSLs, y en el Capítulo 12 refinaremos un poco más esta afirmación. En ambos casos las demostraciones se llevan a cabo mediante diagonalización.

**Ejemplo 9.5** Considérese de nuevo la gramática del Ejemplo 9.4. Existen dos producciones que violan la definición de gramática sensible al contexto. Estas son  $CB \rightarrow E$  y  $AE \rightarrow \epsilon$ . Podemos crear una CSG para el lenguaje  $\{a^i | i \geq 1\}$  si nos damos cuenta que  $A, B, C, D$  y  $E$  no son más que señaladores, que, a fin de cuentas, desaparecerán. En lugar de utilizar símbolos separados para los señaladores, podemos incorporar estos señaladores a las  $a$ s mediante la creación de variables “compuestas” como  $[CaB]$ , que son un solo símbolo que aparece en lugar de la cadena, en este caso  $CaB$ .

El conjunto completo de símbolos que necesitamos para emular la gramática del Ejemplo 9.4 es  $[ACaB], [Aa], [ACa], [ADa], [AEa], [Ca], [Da], [Ea], [aCB], [CaB], [aDB], [aE], [DaB]$  y  $[aB]$ . Las producciones de nuestras gramáticas sensibles al contexto, las cuales agrupamos de acuerdo con la producción del Ejemplo 9.4 que emulan, son:

- 1)  $S \rightarrow [ACaB]$
- 2)  $[Ca]a \rightarrow aa[Ca]$   
 $[Ca][aB] \rightarrow aa[CaB]$   
 $[ACa]a \rightarrow [Aa]a[Ca]$   
 $[ACa][aB] \rightarrow [Aa]a[CaB]$   
 $[ACaB] \rightarrow [Aa][aCB]$   
 $[CaB] \rightarrow a[aCB]$
- 3)  $[aCB] \rightarrow [aDB]$
- 4)  $[aCB] \rightarrow [aE]$
- 5)  $a[Da] \rightarrow [Da]a$   
 $[aDB] \rightarrow [Dab]$   
 $[Aa][Da] \rightarrow [ADa]a$   
 $a[Dab] \rightarrow [Da][aB]$   
 $[Aa][DaB] \rightarrow [ADa][aB]$
- 6)  $[ADa] \rightarrow [ACa]$
- 7)  $a[Ea] \rightarrow [Ea]a$   
 $[aE] \rightarrow [Ea]$   
 $[Aa][Ea] \rightarrow [AEa]a$
- 8)  $[AEa] \rightarrow a$

Es sencillo demostrar que  $S \xrightarrow{*} \alpha$  de la gramática del Ejemplo 9.4 si y sólo si  $S \xrightarrow{*} \alpha'$  en la presente CSG con una, en donde  $\alpha'$  está formada a partir de  $\alpha$  mediante el agrupamiento con una  $a$  de todos los señaladores (desde  $A$  hasta  $E$ ) que aparecen entre ella y la  $a$  que está a su izquierda, y también agrupando en la primera  $a$  a cualquier señalador que esté a su izquierda, y con la última  $a$  a todo señalador que esté a su derecha. Por ejemplo, si  $\alpha = AaaCaB$ , entonces  $\alpha'$  es  $[Aa]a[CaB]$ .

### Autómatas lineales acotados

Introduciremos, ahora, una caracterización de máquina de los CSLs. Un *autómata lineal acotado* (LBA) es una máquina de Turing no determinística que satisface las siguientes dos condiciones.

1. Su alfabeto de entrada incluye dos símbolos especiales  $\epsilon$  y  $\$$ , los *señaladores de extremo izquierdo* y *derecho*, respectivamente.
2. El LBA no tiene movimientos a la izquierda de  $\epsilon$  o a la derecha de  $\$$ , ni puede imprimir otro símbolo sobre  $\epsilon$  y  $\$$ .

El autómata lineal acotado es simplemente una máquina de Turing que, en lugar de tener una cinta potencialmente infinita sobre la cual hacer los cálculos, está restringida a la porción de la cinta que contiene a la entrada  $x$  más los dos cuadrados de la cinta que contienen a los señaladores de extremo. En el Capítulo 12 veremos que restringir la máquina de Turing a una cantidad de cinta que, en cada entrada, está limitada o acotada por alguna función lineal de la longitud de la entrada tendría como resultado una idéntica capacidad de cálculo que la máquina de Turing restringida a la porción de la cinta que contiene a la entrada; de aquí el nombre “autómata lineal acotado”.

Un LBA será representado por  $M = (Q, \Sigma, \Gamma, \delta, q_0, \epsilon, \$, F)$ , en donde  $Q, \Sigma, \Gamma, \delta, q_0$  y  $F$  se definen de la misma manera que para una TM no determinística;  $\epsilon$  y  $\$$  son símbolos de  $\Sigma$ , los señaladores de extremo izquierdo y derecho.  $L(M)$ , el *lenguaje aceptado* por  $M$ , es

$\{w | w \text{ está en } (\Sigma - \{\epsilon, \$\})^* \text{ y } q_0 \epsilon w \$ \xrightarrow{M} \alpha q \beta \text{ para alguna } q \text{ en } F\}$ .

Nótese que los señaladores de extremo se encuentran en la cinta de entrada inicialmente pero no se consideran parte de la palabra que va a ser aceptada o rechazada. Puesto que un LBA no puede moverse fuera de la entrada, no hay necesidad de suponer que existe cinta en blanco a la derecha de  $\$$ .

### Equivalencia de los LBAs y las CSGs

Demostraremos ahora que, excepto por el hecho de que un LBA puede aceptar a  $\epsilon$  mientras que un CSG no puede generar a  $\epsilon$ , los LBAs aceptan exactamente a los CSLs.

**Teorema 9.5** Si  $L$  es un CSL, entonces  $L$  es aceptado por algún LBA.

**Demostración** La demostración es casi la misma que la del Teorema 9.3. La única diferencia reside en que, mientras que la TM del Teorema 9.3 genera formas oracio-

nales de una gramática no restringida en una segunda cinta, el LBA utiliza un segundo registro en su cinta de entrada. Presentado con  $\varphi w\$$  en su cinta, el LBA comienza escribiendo el símbolo  $S$  en un segundo registro debajo del símbolo que está más a la izquierda de  $w$ . Si  $w = \epsilon$ , el LBA se detiene sin aceptar. En seguida el LBA adivina repetidamente una producción y una posición en la forma oracional escrita en el segundo registro. Aplica la producción, corriendo la porción de la forma oracional hacia la derecha siempre que la forma oracional se expande. Sin embargo, si la nueva forma oracional es más larga que  $w$ , si el LBA se detiene sin que haya aceptación. Por consiguiente, el LBA aceptará a  $w$  si existe una derivación  $S \xrightarrow{*} w$  tal que ninguna forma oracional intermedia sea más larga que  $w$ . Pero como el lado derecho de cualquier producción de una CSG tiene una longitud igual o mayor que la de la parte izquierda, no podría existir una derivación  $S \xrightarrow{*} \alpha \xrightarrow{*} w$ , en donde  $\alpha$  es más larga que  $w$ . Por tanto el LBA acepta a todas las palabras generadas por la CSG y solamente a éstas.  $\square$

**Teorema 9.6** Si  $L = L(M)$  para el LBA  $M = (Q, \Sigma, \Gamma, \delta, q_0, \varphi, \$, F)$ , entonces  $L - \{\epsilon\}$  es un CSL.

*Demostración* La prueba hace una construcción paralela a la de una gramática no restringida a partir de la TM del Teorema 9.4. La diferencia radica en que los señaladores de extremo, en la cinta del LBA, deben incorporarse a los símbolos de cinta adyacentes, y el estado debe, de manera similar, incorporarse al símbolo barrido por la cabeza de cinta. La razón para que suceda esto es que si la CSG simulara al LBA utilizando símbolos separados para los señaladores de extremo, o para los estados, podría no borrar tales símbolos después, ya que podría necesitar acortar una forma oracional, y el lado derecho de cada producción de la CSG es al menos de la misma longitud que el izquierdo. La generación de una sucesión de pares, cuyo primer componente forma la cadena terminal  $a_1 a_2 \dots a_n$  y la segunda componente forma la cinta de LBA, es llevada a cabo por las producciones.

$$\begin{aligned} A_1 &\rightarrow [a, q_0 \varphi a] A_2, & A_1 &\rightarrow [a, q_0 \varphi a \$], \\ A_2 &\rightarrow [a, a] A_2, & A_2 &\rightarrow [a, a \$], \end{aligned}$$

para toda  $a$  en  $\Sigma - \{\varphi, \$\}$ .

Las reglas para simular al LBA son parecidas a las reglas 6 y 7 del Teorema 9.4 y se dejan como ejercicio.

Si  $q$  es un estado final, entonces tenemos la producción

$$[a, \alpha q \beta] \rightarrow a$$

para toda  $a$  en  $\Sigma - \{\varphi, \$\}$  y todas las  $\alpha$  y  $\beta$  posibles (esto es,  $\alpha$  y/o  $\beta$  podrían incluir a  $\varphi, \$$  y un símbolo de cinta). Nótese que el número de producciones definidas es finito. Permitimos también la eliminación de la segunda componente de una variable si se encuentra adyacente a una terminal, mediante

$$[a, \alpha] b \rightarrow ab,$$

$$b[a, \alpha] \rightarrow ba$$

para cualesquiera  $a$  y  $b$  en  $\Sigma - \{\varphi, \$\}$  y todas las  $\alpha$ s posibles.

Las producciones mostradas de manera explícita, claramente, son sensibles al contexto. Las producciones que simulan al LBA pueden inducirse de manera sencilla, a que conserven la longitud, de modo que la gramática resultante sea una CSG. La demostración de que cualquier palabra  $w$ , con excepción de  $\epsilon$ , es aceptada por  $M$  si y sólo si es generada por la gramática, sigue un desarrollo paralelo a la del Teorema 9.4, y no la presentamos aquí. Adviértase que no hay manera de que la gramática establezca la entrada  $\varphi \$$  del LBA o simule a  $M$  en esa entrada. Por consiguiente  $\epsilon$  no es generado por la gramática esté o no en  $L(M)$ .  $\square$

## 9.4 RELACIONES ENTRE TIPOS DE LENGUAJES

Los cuatro tipos o clases de lenguaje: r.e., CSLs, CFLs y conjuntos regulares, se conocen con frecuencia como lenguajes del tipo 0, 1, 2 y 3, respectivamente. Podemos demostrar que, excepto para el caso de la cadena vacía, los lenguajes tipo  $i$  propiamente incluyen a los lenguajes tipo  $(i+1)$  para  $i = 0, 1$  y  $2$ . Necesitamos primero demostrar que cada CSL es recursivo y, de hecho, que existen lenguajes recursivos que no son CSLs.

### CSLs y los conjuntos recursivos

**Teorema 9.7** Cada CSL es recursivo.

*Demostración* Dadas una CSG  $G = (V, T, P, S)$  y una palabra  $w$  de  $\Sigma^*$  de longitud  $n$ , podemos probar si  $w$  se encuentra en  $L(G)$  de la manera siguiente. Construyase un grafo cuyos vértices son las cadenas de  $(V \cup T)^*$  de longitud  $n$  o menor. Póngase un arco de  $\alpha$  a  $\beta$  si  $\alpha \Rightarrow \beta$ . Entonces las trayectorias del grafo corresponden a las derivaciones de  $G$ , y  $w$  se encuentra en  $L(G)$  si y sólo si existe una trayectoria que va del vértice para  $S$  al vértice para  $w$ . Utilícese cualquiera entre los diferentes algoritmos para encontrar trayectorias (véase Aho, Hopcroft y Ullman [1974]) para decidir si existe o no dicha trayectoria.  $\square$

**Ejemplo 9.6** Considérese la CSG del Ejemplo 9.5 y la entrada  $w = aa$ . Una forma de probar las trayectorias del grafo es comenzar con la cadena  $S$ , y en el  $i$ -ésimo paso encontrar las cadenas de longitud  $n$  o menores que tienen una trayectoria que parte de  $S$  y cuya longitud sea  $i$  o menor. Si tenemos el conjunto para  $i-1$ , llamémosle  $\mathcal{L}$ , entonces el conjunto para  $i$  será  $\mathcal{L} \cup \{\beta \mid \alpha \Rightarrow \beta \text{ para alguna } \alpha \text{ en } \mathcal{L} \text{ y } |\beta| \leq n\}$ . En nuestro ejemplo tenemos los siguientes conjuntos:

$$\begin{aligned} i = 0: & \{S\} \\ i = 1: & \{S, [ACaB]\} \\ i = 2: & \{S, [ACaB], [Aa][aCB]\} \\ i = 3: & \{S, [ACaB], [Aa][aCB], [Aa][aDB], [Aa][aE]\} \\ & \vdots \end{aligned}$$

$$\begin{aligned} i = 6: \quad & \{S, [ACaB], [Aa][aCB], [Aa][aDB], [Aa][aE], \\ & [Aa][DaB], [Aa][Ea], [ADA][aB], [AEa]a, \\ & [ACA][aB], aa\} \end{aligned}$$

Puesto que para  $i = 6$  descubrimos que  $aa$  se puede alcanzar desde  $S$ , no necesitamos seguir más adelante. En general, como el número de formas oracinales de longitud  $n$  o menor es finito, para cualquier gramática fija y  $n$  constante, sabemos que, finalmente, llegaremos a un punto en el que ya no se agregan nuevas formas oracionales. Puesto que el conjunto para  $i$  depende solamente del conjunto para  $i - 1$ , no añadiremos nunca nuevas cadenas, así que si aún no hemos producido a  $w$ , nunca lo haremos. En tal caso,  $w$  no se encuentra en el lenguaje.

Para demostrar que el CSLs es un subconjunto propio de los lenguajes recursivos demostraremos algo un poco más general. En particular, demostraremos que cualquier clase de lenguaje que puede ser enumerado de manera efectiva, mediante el listado de las máquinas de Turing que se detienen en todas las entradas, para cada miembro de la clase, es una subclase propia de los lenguajes recursivos.

**Lema 9.1** Sea  $M_1, M_2, \dots$  la enumeración de algún conjunto de máquinas de Turing que se detienen en todas las entradas. Entonces existe algún lenguaje recursivo que no es  $L(M_i)$  para cualquier  $i$ .

*Demostración* Sea  $L$  el subconjunto de  $(0 + 1)^*$  tal que  $w$  se encuentre en  $L$  si y sólo si  $M_i$  no acepta a  $w$ , en donde  $i$  es un entero cuya representación binaria es  $w$ .  $L$  es recursivo, ya que dada  $w$  podemos generar  $M_i$  y probar si  $w$  se encuentra o no en  $L(M_i)$ . Pero ninguna TM de la lista acepta a  $L$ . Supóngase que  $L$  fuera  $L(M_j)$  y que  $x$  sea la representación binaria de  $j$ . Si  $x$  está en  $L$ , entonces  $x$  no está en  $L(M_j)$ , y si  $x$  no se encuentra en  $L$ , entonces  $x$  está en  $L(M_j)$ . Por consiguiente,  $L \neq L(M_j)$  como se supuso. De aquí que  $L$  es un lenguaje recursivo que no está en  $L(M_j)$  para cualquier  $j$ .  $\square$

**Teorema 9.8** Existe un lenguaje recursivo que no es sensible al contexto.

*Demostración* Según el Lema 9.1 sólo necesitamos demostrar que podemos enumerar TMs que se detienen para los CSLs sobre el alfabeto  $\{0, 1\}$ . Dado algún código binario, tomemos la representación de cuatro parámetros para las CSGs con alfabeto terminal  $\{0, 1\}$ . Por ejemplo, podríamos hacer que 0, 1, coma,  $\rightarrow$ ,  $\{ \cdot \}$ ,  $( \cdot )$  y sean representados por 10, 100, ...,  $10^8$ , respectivamente, y que la  $i$ -ésima variable se represente con  $10^{i+8}$ . Sea  $M_j$  la máquina de Turing que construye el algoritmo del Teorema 9.7 y que reconoce al lenguaje de la CSG con código binario  $j$ . Es claro que  $M_j$  siempre se detendrá, no importa si su entrada es aceptada o no. Entonces el teorema se concluye de manera inmediata a partir del Lema 9.1.  $\square$

### Teorema de la jerarquía

**Teorema 9.9** (a) Los conjuntos regulares están contenidos apropiadamente en los lenguajes libres de contexto. (b) Los CFLs que no contienen a la cadena vacía están

contenidos apropiadamente en los lenguajes sensibles al contexto. (c) Los CSLs están contenidos apropiadamente en los conjuntos r.e.

*Demostración* La parte (a) se concluye del hecho de que cada gramática es una CFG y el conjunto  $\{0^n 1^n \mid n \geq 1\}$  es un ejemplo de una CFL que no es regular. La parte (b) se demuestra advirtiendo que cada CFG en la forma normal de Chomsky es una CSG.  $\{a^{2i} \mid i \geq 1\}$  es un CSL el cual es fácil demostrar que no es CFL mediante el lema de sondeo. Para la parte (c) cada CSG es, seguramente, una gramática no restringida. La contención propia se concluye del Teorema 9.8.  $\square$

## EJERCICIOS

9.1 Constrúyanse gramáticas lineales por la derecha y por la izquierda para los lenguajes

- a)  $(0 + 1)^* 00(0 + 1)^*$
- b)  $0^*(1(0 + 1)^*)$
- c)  $((01 + 10)^* 11)^* 00^*$

9.2 Demuestre la siguiente forma normal para gramáticas lineales por la derecha y el resultado análogo para las gramáticas lineales por la izquierda: Si  $L$  es un conjunto regular, entonces  $L - \{\epsilon\}$  es generado por una gramática en la cual todas las producciones son de la forma  $A \rightarrow aB$  o  $A \rightarrow a$  para las terminales  $a$  y las variables  $A$  y  $B$ .

9.3 Una gramática libre de contexto se dice que es *simple* si se encuentra en la forma normal de Greibach y, para cada variable  $A$  y cada terminal  $a$ , existe cuando mucho una cadena  $\alpha$  tal que  $A \rightarrow a\alpha$  es una producción. Un lenguaje es *simple* si tiene una gramática simple. Por ejemplo,  $L = \{0^n 1^n \mid n \geq 1\}$  tiene la gramática simple:

$$\begin{aligned} S &\rightarrow 0A \\ A &\rightarrow 0AB \mid 1 \\ B &\rightarrow 1 \end{aligned}$$

Nótese que la gramática más natural GNF para  $L$ ,

$$\begin{aligned} S &\rightarrow 0SB \mid 0B \\ B &\rightarrow 1 \end{aligned}$$

no es simple, ya que existen dos producciones  $S$  cuyo lado derecho comienza con 0. Demuestre que cada conjunto regular que no contiene  $a \in$  es un lenguaje simple. [Sugerencia: Utilice una representación de DFA para el conjunto regular.]

\*9.4 Se dice que una CFG es *autofijable* si existe alguna variable útil  $A$  tal que  $A \xrightarrow{*wAx}$ , y ni  $w$  ni  $x$  son  $\epsilon$ . Demuestre que un CFL es regular si y sólo si tiene una CFG que no sea autofijable. [Sugerencia: Es fácil demostrar que ninguna gramática regular es autofijable. Para la parte “si”, demuéstrese que una gramática que no es autofijable puede ponerse en la forma normal de Greibach sin hacerla autofijable. Despues muestre que para cada gramática no autofijable GNF existe una constante  $k$  tal que ninguna forma oracional izquierda tenga más de  $k$  variables. Finalmente, demuestre, a partir de lo anterior, que la gramática no autofijable GNF puede ser convertida a una gramática regular.]

\*9.5 Dé gramáticas no restringidas para

- |   |                                    |
|---|------------------------------------|
| a) $\{ww \mid w \text{ está en } (0 + 1)^*\}$ | b) $\{0^2 \mid i \geq 1\}$         |
| c) $\{0^i \mid i \text{ no es primo}\}$       | d) $\{0^i 1^j 2^k \mid i \geq 1\}$ |

**9.6** Dé gramáticas sensibles al contexto para los lenguajes del Ejercicio 9.5, excluyendo  $a \in \{a\}$ .

**9.7** Se dice que un CSL es *determinístico* si es aceptado por algún LBA determinístico. Demuestre que el complemento de un CSL determinístico es también un CSL determinístico. [Sugerencia: Demuestre que, para cada LBA determinístico existe un LBA equivalente que se detiene en cada entrada.] Incidentalmente, éste es abierto cuando cada CSL es un CSL determinístico, y si los CSLs son cerrados con respecto a la complementación. Obviamente, una respuesta positiva a la primer cuestión implicaría una respuesta positiva a la segunda.

\*9.8

- a) Demuestre que cada lenguaje libre de contexto es aceptado por un LBA determinístico.
- b) Demuestre que la cerradura booleana de los CFLs está contenida en la clase de los conjuntos aceptados por los LBAs determinísticos.
- c) Demuestre que la contención en (b) es propia. [Sugerencia: Considérese a los lenguajes sobre el alfabeto de un símbolo.]

\*9.9 Demuestre que cada CSL es generado por una gramática en la cual todas las producciones son de la forma  $\alpha A\beta \rightarrow \alpha\gamma\beta$ , en donde  $A$  es una variable,  $\alpha$ ,  $\beta$  y  $\gamma$  son cadenas de símbolos de gramática y  $\gamma \neq \epsilon$ .

\*9.10 Muestre que los CSLs son cerrados con respecto a las operaciones siguientes:

- |                         |  |
|-------------------------|--|
| a) unión                | b) concatenación   |
| c) intersección         | d) sustitución   |
| e) homomorfismo inverso | f) cerradura positiva (recuerde que $L^+ = \bigcup_{i=1}^{\infty} L^i$ ) |

\*9.11 Demuestre que los conjuntos r.e. son cerrados con respecto a las siguientes operaciones: los mismos incisos (a) hasta (e) del Ejercicio 9.10  
f) cerradura de Kleene.

9.12

- a) Demuestre que todas las propiedades irresolubles de los CFLs que se mencionaron en las Secciones 8.5, 8.6 y 8.7 son irresolubles para los CSLs, con la excepción de que “ $=\Sigma$ ” es trivialmente resoluble porque ningún CSL contiene  $a \in \epsilon$ .
- b) Demuestre que “ $=\Sigma$ ” es irresoluble para los CSLs.

S 9.13 Demuestre que es no soluble la cuestión de si un CSL dado es el vacío.

\*S9.14 Demuestre que cada conjunto r.e. es  $h(L)$ , en donde  $h$  es un homomorfismo y  $L$  un CSL.

#### Soluciones a los ejercicios seleccionados

**9.10** Las demostraciones son semejantes a las pruebas de los Teoremas 6.1, 6.2 y 6.3 para CFLs. Sin embargo, existe un problema que tenemos que resolver. Considérese la construcción de la concatenación. Supóngase que

$$G_1 = (V_1, T_1, P_1, S_1) \quad y \quad G_2 = (V_2, T_2, P_2, S_2)$$

son CSGs que generan a  $L_1$  y  $L_2$ , respectivamente. En el Teorema 6.1 para CFGs construimos la gramática

$$G_4 = (V_1 \cup V_2 \cup \{S_4\}, T_1 \cup T_2, P_1 \cup P_2 \cup \{S_4 \rightarrow S_1 S_2\}, S_4)$$

para generar a  $L_1 L_2$ . Esta construcción es correcta para las CFGs, siempre que  $V_1$  y  $V_2$  sean disjuntos.

Para los CSGs, sin embargo, podríamos tener producciones  $\alpha \rightarrow \beta$  en  $P_1$  o  $P_2$  que fuesen aplicables en una forma oracional de  $G_4$ , digamos  $\gamma\delta$ , en donde  $S_1 \xrightarrow{*} \gamma$  y  $S_2 \xrightarrow{*} \delta$ , en una posición tal que  $\alpha$  quede sobre el límite entre  $\gamma$  y  $\delta$ . Por tanto, debemos derivar una cadena que no esté en  $L_1 L_2$ .

Suponiendo  $V_1 \cap V_2 = \emptyset$  no representa una ayuda, ya que  $\alpha$  podría consistir solamente en terminales y, por supuesto, no podemos suponer que  $T_1 \cap T_2 = \emptyset$ . Lo que necesitamos es una forma normal para las CSGs que permita sólo variables en el lado izquierdo de las producciones. Este lema es fácil de demostrar. Sea  $G = (V, T, P, S)$  una CSG. Constrúyase  $G' = (V, T, P', S)$ , en donde  $V$  consista en  $V$  más las variables  $Aa$  para cada  $a$  en  $T$ .  $P'$  consiste en las producciones  $Aa \rightarrow a$  para cada  $a$  y la producción  $\alpha' \rightarrow \beta'$  para cada  $\alpha \rightarrow \beta$  en  $P$ , en donde  $\alpha'$  es  $\alpha$  en la que cada vez que aparece  $a$  es sustituida por  $A_a$  y  $\beta'$  está relacionada de manera parecida con  $\beta$ .

Ahora, si suponemos que  $G_1$  y  $G_2$  tienen conjuntos disjuntos de variables y se encuentran en la forma normal descrita más arriba, la construcción del Teorema 6.1 para la unión y la concatenación conduce a los CSLs.

La cerradura positiva presenta otro problema. Si, en analogía con el Teorema 6.1, construimos

$$G_5 = (V_1 \cup \{S_5\}, T_1 \cup \{S_5 \rightarrow S_1 S_5 | S_1\}, S_5),$$

no hemos evitado el problema del potencial para la aplicación de una producción  $\alpha \rightarrow \beta$  en tal forma que deje a las cadenas sobre los límites; estas cadenas son derivadas de dos o más instancias de  $S_1$ . Lo que podemos hacer es crear una gramática  $G'_1$ , la cual es  $G_1$  con cada variable  $A$  sustituida por un nuevo símbolo  $A'$ . Entonces construimos la gramática  $G_5 = (V_5, T_1, P'_5, S_5)$ , en la cual  $V_5$  consiste en las variables de  $G_1$  y  $G'_1$  más los símbolos  $S_5$  y  $S'_5$ ;  $P'_5$  consiste en las producciones de  $G_1$  y  $G'_1$  más

$$\begin{aligned} S_5 &\rightarrow S_1 S'_5 | S_1 \\ S'_5 &\rightarrow S'_1 S_5 | S'_1 \end{aligned}$$

Como ningún CSL contiene  $a \in \epsilon$ , nunca podemos tener símbolos derivados de dos instancias de  $S_1$  o dos instancias de  $S'_1$  adyacentes, y podemos estar seguros de que cada producción de  $G_5$  es aplicada a una cadena derivada de una instancia de  $S_1$  o  $S'_1$ .

El homomorfismo inverso, la intersección y la sustitución se pueden manejar mejor con demostraciones basadas en máquinas. Sea  $L$  un CSL aceptado por el LBA  $M$  y  $h$  un homomorfismo. Supóngase que  $|h(a)| \leq k$ ,  $\alpha$  para cualquier  $a$ . Entonces podemos construir el LBA  $M'$  para  $h^{-1}(L)$  de la manera siguiente:  $M'$  toma su entrada  $x$  y calcula  $h(x)$ , almacenando  $k$  símbolos por celda. Existe espacio suficiente, ya que  $|h(x)| \leq k|x|$ . Entonces  $M'$  simula a  $M$  en  $h(x)$ , aceptando si  $M$  acepta.

Para la intersección, sea  $L_1$  y  $L_2$  CSLs aceptados por los LBAs  $M_1$  y  $M_2$ . Constrúyase el LBA  $M_3$  que trata su entrada como si fuese escrita en dos registros. Esto es, identificamos el símbolo de entrada  $a$  con  $[a, a]$ . En el primer registro,  $M_3$  simula a  $M_1$ . Si alguna secuencia de alternativas de movimientos de  $M_1$  hace que ésta acepte,  $M_3$  empieza a simular a  $M_2$  en el segundo registro, aceptando si  $M_2$  acepta. Por consiguiente  $M_3$  acepta  $L_1 \cap L_2$ .

Para la sustitución en el CSL  $L \subseteq \Sigma$  de los CSLs  $L_a$  para símbolos  $a$  de  $\Sigma$ , constrúyase un LBA que funciona de la manera siguiente. Dada la entrada  $a_1 a_2 \dots a_n$ , adivínese de forma no determinística cuáles son las posiciones que terminan las cadenas de algún  $L_a$  y señálense. Si dividimos que  $a_1 a_{i_1} \dots a_j$  está en algún  $L_a$  en particular, simúlese en LBA para  $L_a$  sobre esa subcadena. Si  $a_1 a_{i_1} \dots a_j$  está en  $L_a$ , sustitúyase por  $a$ . Si todas nuestras conjecturas son correctas,

tómese la cadena resultante de  $\Sigma^*$  y simúlese un LBA para  $L$  en él, aceptando a  $a_1 a_2 \dots a_n$ , si ese LBA acepta.

**9.13** Es fácil diseñar un LBA que acepte los cálculos válidos de una máquina de Turing dada. Por consiguiente, el problema de vacuidad para las máquinas de Turing es reducible a la cuestión de si un CSL dado es el vacío o no.

**9.14** Sea  $L_1$  un conjunto r.e. y  $c$  un símbolo que no está en el alfabeto de  $L_1$ . Sea  $M_1$  una TM que acepta a  $L_1$  y defínase

$$L_2 = \{wc \mid M_1 \text{ acepta a } w \text{ mediante una sucesión de movimientos en los cuales la cabeza nunca se mueve más de } i \text{ posiciones hacia la derecha de } w\}.$$

Entonces  $L_2$  es aceptado por un LBA que simula a  $M_1$ , tratando a  $c$  como el espacio en blanco y deteniéndose si siempre va más allá de la sucesión de  $cs$  en su entrada. Sólo tenemos que demostrar que  $L_1 = h(L_2)$  para algún homomorfismo  $h$ . Sea  $h(a) = a$  para todos los símbolos del alfabeto de  $L$  y  $h(c) = \epsilon$ .

Combinando el Ejercicio 9.14 con el Teorema 9.9, observamos que los CSLs no son cerrados con respecto a los homomorfismos. Esto puede parecer paradójico, ya que el Ejercicio 9.10 exigía que los CSLs fueran cerrados con respecto a la sustitución. Sin embargo, el homomorfismo no es un caso especial de la sustitución por un CSL, ya que el CSL puede no contener  $a \in$ . En particular, para la  $h$  definida más arriba,  $h(c) = \epsilon$  no es un CSL. Los CSLs son, sin embargo, cerrados con respecto a los homomorfismos que no transforman a cualquier símbolo en  $\epsilon$ .

## NOTAS BIBLIOGRAFICAS

La jerarquía de Chomsky fue definida en Chomsky [1956, 1959]. Chomsky y Miller [1958] demostraron la equivalencia de las gramáticas regulares y los conjuntos regulares. Kuroda [1964] demostró la equivalencia de los LBAs y las CSGs. Anteriormente, Myhill [1960] había definido a los LBAs determinísticos y Landweber [1963] demostró que los lenguajes de los LBAs determinísticos están contenidos en los CSLs. Chomsky [1959] demostró que los conjuntos r.e. son equivalentes a los lenguajes generados por las gramáticas tipo 0. Fische [1969] de algunas caracterizaciones interesantes de los CSLs. Hibbard [1974] discute una restricción da la CSGs que produce a los lenguajes libres de contexto. Propiedades de cerradura adicionales de los CSLs se estudian en Ginsburg y Griebach [1966b] y en Wegbreit [1969]. Las propiedades de decisión básicas de los CSLs se dan en Landweber [1964].

## CAPITULO

# 10

## LENGUAJES DETERMINISTICOS LIBRES DE CONTEXTO

Hasta aquí tenemos modelos de máquinas que definen a cada clase de lenguajes en la jerarquía de Chomsky. En los extremos de la jerarquía, las máquinas (autómatas finitos y máquinas de Turing) no muestran diferencia alguna en su capacidad de aceptación, entre sus modelos determinísticos y no determinísticos. Para los autómatas lineales acotados, no se sabe si las variedades determinísticas o no determinísticas aceptan o no a la misma clase de lenguajes. Sin embargo, para los autómatas de pila sabemos que los PDAs aceptan una familia de lenguajes, los *lenguajes determinísticos libres de contexto* (DCFLs), que se encuentran propiamente entre los conjuntos regulares y los lenguajes libres de contexto.

Resulta que la sintaxis de muchos lenguajes de programación puede describirse mediante los DCFLs. Aún más, los sistemas compiladores de escritura modernos, por lo general, requieren que la sintaxis del lenguaje, para el cual producirán el compilador, sea descrita por una gramática libre de contexto de forma restringida. Estas formas restringidas casi invariablemente generan solamente DCFLs. Estudiaremos la que probablemente sea la más importante de tales formas definidas: las gramáticas *LR*. Las gramáticas *LR* tienen la propiedad de que generen exactamente a los DCFLs.

Si se va a utilizar un sistema compilador de escritura, es necesario por lo general que el diseñador del lenguaje escoga una sintaxis para este lenguaje que lo convierta en un DCFL. Por consiguiente resulta útil ser capaz de determinar si un lenguaje propuesto es de hecho un DCFL. Si sí es, uno puede, con frecuencia, probarlo mediante la producción de un DPDA o una gramática *LR* que defina al lenguaje. Pero si el lenguaje  $L$  no es un DCFL, ¿Cómo vamos a demostrarlo? Si  $L$  no es un CFL en absoluto, podríamos utilizar, tal vez, el lema de sondeo. Sin embargo, con frecuencia  $L$  será un

CFL pero no DCFL. No existe un lema de sondeo que sea específicamente para los DCFLs, de modo que debemos retornar a las propiedades de cerradura. Afortunadamente, los DCFLs son cerrados con respecto a cierto número de operaciones, como la complementación, que en general no conservan a los CFLs. Por tanto, si  $L$  es un CFL pero su complemento no, entonces  $L$  no es un DCFL.

Las Secciones 10.1 hasta 10.4 desarrollan varias propiedades de cerradura de los DCFLs. La Sección 10.5 cubre de manera breve las propiedades de decisión. Las Secciones 10.6 y 10.7 tratan sobre las gramáticas  $LR$ .

### 10.1 FORMAS NORMALES PARA DPDA

Recuérdese que PDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  es determinístico si:

1. siempre que  $\delta(q, a, X)$  sea no vacía para alguna  $a$  en  $\Sigma$ , entonces  $\delta(q, \epsilon, X)$  es vacío y
2. para cada  $q$  en  $Q$ ,  $a$  en  $\Sigma \cup \{\epsilon\}$  y  $X$  en  $\Gamma$ ,  $\delta(q, a, X)$  contiene cuando mucho un elemento.

La regla (1) evita que exista la alternativa de escoger entre utilizar la entrada siguiente o hacer un movimiento  $\epsilon$ . La Regla (2) evita una alternativa sobre la misma entrada. Para los PDAs determinísticos, de aquí en adelante, escribiremos  $\delta(q, a, X) = (p, \gamma)$  más que  $\delta(q, a, X) = \{(p, \gamma)\}$ .

Como en el caso de los PDAs en general, podemos poner a los DPDA en una *forma normal* en donde las únicas operaciones de pila consisten en borrar el símbolo de la cima o insertar un símbolo. Esta forma será demostrada en los siguientes dos lemas. El primer lema muestra que el DPDA no necesita nunca colocar más de un símbolo por movimiento, ya que puede colocar una cadena de símbolos uno a la vez, utilizando movimientos  $\epsilon$ . El segundo lema muestra que los DPDA no necesitan nunca cambiar el símbolo del tope de la pila. Los cambios se evitan mediante el almacenamiento del símbolo en el tope de la pila en el control finito y registrando los cambios que se hacen ahí. El lector que tiene un dominio sobre tales ideas puede saltarse hasta el principio de la siguiente sección.

**Lema 10.1** Cada DCFL es  $L(M)$  para un DPDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  tal que si  $\delta(q, a, X) = (p, \gamma)$ , entonces  $|\gamma| \leq 2$ .

*Demuestra* Si  $\delta(q, a, X) = (r, \gamma)$  y  $|\gamma| > 2$ , sea  $\gamma = Y_1 Y_2 \cdots Y_n$  en donde  $n \geq 3$ . Ahora se crean estados que no son de aceptación  $p_1, p_2, \dots, p_{n-2}$  y se redefinie

$$\delta(q, a, X) = (p_1, Y_{n-1} Y_n).$$

Defínase entonces

$$\delta(p_i, \epsilon, Y_{n-i}) = (p_{i+1}, Y_{n-i-1} Y_{n-i})$$

para  $1 \leq i \leq n-3$  y  $\delta(p_{n-2}, \epsilon, Y_2) = (r, Y_1, Y_2)$ . Por consiguiente, en el estado  $q$ , sobre la entrada  $a$ , con  $X$  en el tope de la pila, el DPDA revisado todavía sustituye a  $X$  con  $Y_1 Y_2 \cdots Y_n = \gamma$  y accesa al estado  $r$ , pero ahora toma  $n-1$  movimientos para hacerlo.  $\square$

**Lema 10.2** Cada DCFL es  $L(M)$  para un DPDA  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  tal que si  $\delta(q, a, X) = (p, \gamma)$ , entonces  $\gamma$  es  $\epsilon$  (una inserción),  $X$  (no hay movimiento de pila) o de la forma  $YX$  (una presión) para algún símbolo de pila  $Y$ .

*Demuestra* Supóngase que  $L = L(M')$ , en donde  $M' = (Q', \Sigma, \Gamma', \delta', q'_0, X_0, F')$  satisface las condiciones del Lema 10.1. Construimos  $M$  para simular a  $M'$ , mientras mantenemos el símbolo del tope de la pila de  $M'$  en el control de  $M$ . De manera formal, hagamos

$$Q = Q' \times \Gamma', q_0 = [q'_0, X_0], \quad F = F' \times \Gamma' \quad \text{y} \quad \Gamma = \Gamma' \cup \{Z_0\},$$

en donde  $Z_0$  es un nuevo símbolo que no se encuentra en  $\Gamma'$ . Defínase  $\delta$  mediante:

- i) Si  $\delta'(q, a, X) = (p, \epsilon)$ , entonces para toda  $Y$ ,  $\delta([q, X], a, Y) = ([p, Y], \epsilon)$ . Si  $M'$  inserta su pila,  $M$  inserta su pila, recogiendo el símbolo insertado para su control.
- ii) Si  $\delta'(q, a, X) = (p, Y)$ , entonces para toda  $Z$ ,  $\delta([q, X], a, Z) = ([p, Y], Z)$ . Si  $M'$  cambia su símbolo en el tope de la pila,  $M$  registra el cambio en su propio control pero no altera su pila.
- iii) Si  $\delta'(q, a, X) = (p, YZ)$ , entonces, para toda  $W$ ,  $\delta([q, X], a, W) = ([p, Y], ZW)$ . Si la pila de  $M'$  crece,  $M$  empuja un símbolo en su pila.

Es fácil demostrar, por inducción sobre el número de movimientos hechos, que

$$(q'_0, w, X_0) \vdash_{M'}^* (q, \epsilon, X_1 X_2 \cdots X_n)$$

si y sólo si

$$([q'_0, X_0], w, Z_0) \vdash_M^* ([q, X_1], \epsilon, X_2 X_3 \cdots X_n Z_0).$$

Por consiguiente  $L(M) = L(M')$ .  $\square$

### 10.2 CERRADURA DE LOS DCFLs CON RESPECTO A LA COMPLEMENTACION

Para demostrar que el complemento de un DCFL es también un DCFL, nos gustaría utilizar el planteamiento que se usó en el Teorema 3.2 para demostrar la cerradura de los conjuntos regulares con respecto a la complementación. Esto es, dado un DPDA  $M$  nos gustaría intercambiar estados finales y no finales y entonces ser capaces de exigir que el DPDA resultante acepte al complemento de  $L(M)$ .

Existen dos dificultades que complican el planteamiento anterior. La primera dificultad reside en que el DPDA original no debe moverse nunca más allá de algún punto sobre una cadena de entrada, ya que al leer la entrada  $w$  o alcanza una ID en la cual no es posible hacer ningún movimiento, o hace una infinidad de movimientos sobre la entrada  $\epsilon$  y nunca utiliza otro símbolo de entrada. En cualquier caso, el DPDA no acepta ninguna entrada que tenga a  $w$  como prefijo, y por consiguiente un DPDA que acepte al complemento deberá aceptar cada cadena con prefijo  $w$ . Sin embargo, si solamente cambiamos estados finales y no finales, el DPDA resultante aún no se moverá más allá de  $w$  y por lo tanto no aceptará cadenas con el prefijo  $w$ .

La segunda dificultad es debida al hecho de que después de ver una oración  $x$ , el DPDA puede hacer varios movimientos sobre la entrada  $\epsilon$ . El DPDA puede encontrarse en estados finales después de algunos de estos movimientos y en estados no

finales después de otros. En este caso, intercambiar los estados finales y no finales trae como consecuencia que el DPDA todavía acepte a  $x$ .

### Cómo forzar a los DPDA a barrer su propia entrada

Para eliminar la primera dificultad, demostramos un lema que establece que, dado un DPDA  $M$ , siempre podemos encontrar un DPDA equivalente  $M'$  que nunca accesará una ID a partir de la cual no utilizará nunca otro símbolo de entrada.

**Lema 10.3** Sea  $M$  un DPDA. Existe un DPDA equivalente  $M'$  tal que sobre cada entrada,  $M'$  barre a la entrada completa.

*Demuestra* Podemos suponer, sin pérdida de generalidad, que para cada ID accesible y cada símbolo de entrada,  $M$  tiene un siguiente movimiento. De otra manera uno puede añadir un señalador de extremo en la pila para evitar que  $M$  borre la pila completa y por tanto que se detenga, sin barrer, la entrada completa. Además, se puede agregar un “estado muerto”,  $d$  de manera que para cualquier combinación de estado, símbolo de entrada y símbolo de pila para la cual  $M$  no tiene un siguiente movimiento, ya sea utilizando el símbolo de entrada o una entrada  $\epsilon$ , ocurra una transferencia al estado  $d$ . Sobre cualquier símbolo de entrada, la única transición del estado  $d$  es al estado  $d$ , y no se da ningún cambio en la pila. Por supuesto,  $d$  no es un estado de aceptación.

Ahora, si para cada ID y símbolo de entrada,  $M$  tiene un siguiente movimiento, entonces la única forma en la que  $M$  nunca alcanzaría el extremo de su entrada es si en alguna ID,  $M$  hace una infinidad de movimientos sobre la entrada  $\epsilon$ . Si en el estado  $q$ , con  $Z$  en el tope de la pila,  $M$  hace una infinidad de movimientos  $\epsilon$  sin borrar el símbolo  $Z$ , entonces hagamos que  $M$  accese en su lugar el estado muerto  $d$ . Este cambio no puede afectar al lenguaje aceptado, a menos que  $M$  haya entrado a un estado de aceptación en algún momento durante la sucesión infinita de movimientos  $\epsilon$ . Sin embargo, en este caso, introducimos un nuevo estado final  $f$ , haciendo  $\delta(q, \epsilon, Z) = (f, Z)$  y  $\delta(f, \epsilon, Z) = (d, Z)$ .

Formalmente, proponemos la siguiente construcción. Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ . Definimos

$$M' = (Q \cup \{q'_0, d, f\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q'_0, X_0, F \cup \{f\}),$$

en donde:

1.  $\delta'(q'_0, \epsilon, X_0) = (q_0, Z_0 X_0)$ .  $X_0$  señala el fondo de la pila.
2. Si para alguna  $q$  en  $Q$ ,  $a$  en  $\Sigma$  y  $Z$  en  $\Gamma$ ,  $\delta(q, a, Z)$  y  $\delta(q, \epsilon, Z)$  son ambas el vacío, entonces

$$\delta'(q, a, Z) = (d, Z).$$

También para toda  $q$  en  $Q$  y  $a$  en  $\Sigma$ ,

$$\delta'(q, a, X_0) = (d, X_0).$$

Accesa al estado muerto si no es posible, hacer un movimiento.

$$3. \delta'(d, a, Z) = (d, Z) \text{ para toda } a \text{ en } \Sigma \text{ y } Z \text{ en } \Gamma \cup \{X_0\}.$$

4. Si, para  $q$  y  $Z$  y toda  $i$ , existen  $q_i$  y  $\gamma_i$  para las cuales se cumple  $(q, \epsilon, Z) \xrightarrow{M'} (q_i, \epsilon, \gamma_i)$ , entonces  $\delta'(q, \epsilon, Z) = (d, Z)$ , con la condición de que ninguna  $q_i$  sea estado final y que  $\delta'(q, \epsilon, Z) = (f, Z)$  siempre que una o más de las  $q_i$  sea final. ( Nótese que no hemos pedido que podamos determinar cuándo  $\delta'(q, \epsilon, Z)$  deba ser  $(d, Z)$  o  $(f, Z)$ . Sin embargo, existe sólo un número finito de tales decisiones que deben tomarse. Para cada conjunto posible de alternativas existe un DPDA. Uno de estos DPDA es será el que se desea. Posteriormente deberemos demostrar que la construcción es efectiva.)
5.  $\delta'(f, \epsilon, Z) = (d, Z)$  para toda  $Z$  en  $\Gamma \cup \{X_0\}$ .
6. Para cualquier  $q$  en  $Q$ ,  $a$  en  $\Sigma \cup \{\epsilon\}$  y  $Z$  en  $\Gamma$ , si  $\delta'(q, a, Z)$  no ha sido definida mediante la regla (2) o la (4), entonces  $\delta'(q, a, Z) = (q, a, Z)$ .

El argumento que precede a la construcción formal debería ser suficiente para convencernos de que  $L(M') = L(M)$ . Para demostrar que  $M'$  utiliza todas sus entradas, supóngase que, para algún prefijo propio  $x$  de  $xy$ ,

$$(q'_0, xy, X_0) \xrightarrow{M'}^* (q, y, Z_1 Z_2 \cdots Z_k X_0),$$

y de la ID  $(q, y, Z_1 Z_2 \cdots Z_k X_0)$ , nunca se consume un símbolo de  $y$ . Según la regla (2) no es posible que  $M'$  se detenga. Mediante la regla (4) no es posible que  $M'$  haga una sucesión infinita de movimientos  $\epsilon$  sin borrar a  $Z_i$ . Por consiguiente  $M'$  debe, en algún momento, borrar a  $Z_i$ . De manera similar,  $M'$  deberá borrar a  $Z_2, \dots, Z_k$  y, finalmente, accesar una ID  $(q', y, X_0)$ . Según la regla (2)  $(q', y, X_0) \xrightarrow{M'}^* (d, y', X_0)$ , en donde  $y = ay'$  y  $a$  se encuentra en  $\Sigma$ . Por tanto,  $M'$  no deja de leer más allá de  $x$  como se supuso y  $M'$  satisface las condiciones del lema.  $\square$

Observemos ahora que la construcción de la regla (4) del Lema 10.3 puede hacerse efectiva. Supóngase, sin pérdida de generalidad, que  $M$  es una forma normal. Calcularemos más información de la que en realidad se necesita. En particular, determinaremos, para cada  $q$  y  $p$  de  $Q$  y  $Z$  en  $\Gamma$ , si

1.  $(q, \epsilon, Z) \xrightarrow{M'}^* (p, \epsilon, Z)$ ,
2.  $(q, \epsilon, Z) \xrightarrow{M'}^* (p, \epsilon, \epsilon)$ ,
3.  $(q, \epsilon, Z) \xrightarrow{M'}^* (p, \epsilon, \gamma)$ , para alguna  $\gamma$  en  $\Gamma^*$ .

Para cada  $q$  y  $Z$  podemos determinar a partir de (3) si  $M$  accesa un estado que consume al símbolo siguiente de  $z$  sin borrar a  $Z$ .<sup>†</sup> Si no, entonces de la regla (2) podemos determinar si  $M$  borra a  $Z$ . Si ninguno de los dos sucesos ocurre, entonces  $M'$  debe accesar el estado muerto mediante la regla (2), o la regla (4) se aplica y de nuevo (3) nos dice si  $\delta'(q, \epsilon, Z)$  es  $(d, Z)$  o  $(f, Z)$ .

Constrúyanse las tablas de valores booleanos  $T_1, T_2$  y  $T_3$  tales que, para  $i = 1, 2$  y  $3$ ,  $T_i(q, Z, p)$  es *verdadero* si y sólo si la proposición ( $i$ ) es *verdadera* para  $q, Z$  y  $p$ . Inicialmente, todas las tablas son falsas y se llenan inductivamente. La base reside en

<sup>†</sup> Nótese que mediante la construcción del Lema 10.2, el solo estado  $p$  determina si se va a realizar un movimiento no  $e$  sobre la entrada.

establecer  $T_3(q, Z, p) = \text{verdadero}$  si  $\delta(q, \epsilon, Z) = (p, YZ)$ ,  $T_1(q, Z, p) = T_3(q, Z, p) = \text{verdadero}$  si  $\delta(q, \epsilon, Z) = (p, Z)$  y  $T_2(q, Z, p) = \text{verdadero}$  si  $\delta(q, \epsilon, Z) = (p, \epsilon)$ . Las inferencias inductivas son:

1. Siempre que  $\delta(q, \epsilon, Z) = (r, YZ)$ , entonces

- a) si  $T_2(r, Y, s)$  y  $T_2(s, Z, p)$  son verdaderos, hágase  $T_2(q, Z, p) = \text{verdadero}$ ;
- b) si  $T_2(r, Y, s)$  y  $T_1(s, Z, p)$  son verdaderos, hágase  $T_1(q, Z, p) = \text{verdadero}$ ;
- c) si  $T_2(r, Y, s)$  y  $T_3(s, Z, p)$  son verdaderos, o  $T_1(r, Y, s)$  y  $T_3(s, Y, p)$  son verdaderos, hágase  $T_3(q, Z, p) = \text{verdadero}$ ;
- d) si  $T_3(r, Y, p)$  es verdadero, hágase  $T_3(q, Z, p) = \text{verdadero}$ .

2. Siempre que  $\delta(q, \epsilon, Z) = (r, Z)$  entonces

- a) si  $T_1(r, Z, p)$  es verdadero, hágase  $T_1(q, Z, p) = \text{verdadero}$ ;
- b) si  $T_2(r, Z, p)$  es verdadero, hágase  $T_2(q, Z, p) = \text{verdadero}$ ;
- c) si  $T_3(r, Z, p)$  es verdadero, hágase  $T_3(q, Z, p) = \text{verdadero}$ ;

Dejamos como ejercicio la construcción de un algoritmo eficiente para llenar las entradas verdaderas en las tablas y demostrar que las únicas entradas verdaderas son aquellas que se concluyen a partir de la base y las reglas (1) y (2) dadas arriba.

### Cerradura y complementación

Ahora estamos listos para demostrar que los DCFLs son cerrados con respecto a la complementación. Para hacerlo debemos tratar con el segundo problema mencionado al principio de la presente sección; la posibilidad de que después de leer la entrada  $w$ , el DPDA haga una sucesión de movimientos  $\epsilon$ , accesando estados finales y no finales. La solución consiste en modificar el DPDA mediante la edición de una segunda componente al estado. La segunda componente registra si ha sido accesado un estado final del DPDA original desde la última vez que se utilizó una entrada *verdadera* ( $\neq \epsilon$ ) en un movimiento. Si no, el DPDA que acepta al complemento accesa un estado final de los suyos, justo antes de que se encuentre listo para utilizar al siguiente símbolo de entrada verdadero.

**Teorema 10.1** El complemento de un DCFL es un DCFL.

*Demostración* Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  un DPDA que satisface las condiciones del Lema 10.3. Sea  $M' = (Q', \Sigma, \Gamma, \delta', q'_0, Z_0, F')$  un DPDA que simula a  $M$ , en donde

$$Q' = \{[q, k] \mid q \text{ está en } Q \text{ y } k = 1, 2, \text{ o } 3\}.$$

Hagamos  $F' = \{[q, 3] \mid q \text{ está en } Q\}$ , y

$$q'_0 = \begin{cases} [q_0, 1] & \text{si } q_0 \text{ está en } F; \\ [q_0, 2] & \text{si } q_0 \text{ no está en } F. \end{cases}$$

El propósito de  $k$  en  $[q, k]$  es registrar, entre entradas verdaderas, si  $M$  ha accesado o no un estado de aceptación. Si  $M$  ha accesado un estado de aceptación desde la última entrada verdadera, entonces  $k = 1$ . Si  $M$  no ha accesado un estado de aceptación desde la última entrada verdadera, entonces  $k = 2$ . Si  $k = 1$  cuando  $M$  lee un símbolo de entrada verdadero, entonces  $M'$  simula el movimiento de  $M$  y cambia  $k$  a 1 o 2, dependiendo

de si el nuevo estado de  $M$  se encuentra o no en  $F$ . Si  $k = 2$ ,  $M'$  primero cambia  $k$  a 3 y después simula el movimiento de  $M$ , cambiando  $k$  a 1 o 2, dependiendo de si el nuevo estado de  $M$  se encuentra o no en  $F$ . Por consiguiente,  $\delta'$  está definida de la manera siguiente, para  $q$  y  $p$  en  $Q$  y  $a$  en  $\Sigma$ .

1. Si  $\delta(q, \epsilon, Z) = (p, \gamma)$ , entonces, para  $k = 1$  o 2,

$$\delta'([q, k], \epsilon, Z) = ([p, k'], \gamma),$$

en donde  $k' = 1$  si  $k = 1$  o  $p$  se encuentra en  $F$ ; en cualquier otro caso  $k' = 2$ .

2. Si  $\delta(q, a, Z) = (p, \gamma)$ , para  $a$  en  $Z$ , entonces

$$\delta'([q, 2], \epsilon, Z) = ([q, 3], Z)$$

y

$$\delta'([q, 1], a, Z) = \delta'([q, 3], a, Z) = ([p, k], \gamma)$$

en donde  $k = 1$  o 2 para  $p$  en  $F$  o para  $p$  no en  $F$ , respectivamente.

Pedimos que  $L(M')$  sea el complemento de  $L(M)$ . Supóngase que  $a_1 a_2 \dots a_n$  se encuentra en  $L(M)$ . Entonces  $M$  accesa un estado de aceptación después de utilizar a  $a_n$  como una entrada. En este caso, la segunda componente del estado de  $M'$  será 1 antes de que sea posible para  $M'$  utilizar una entrada verdadera después de  $a_n$ . Por tanto  $M$  no acepta (acceder un estado cuya segunda componente es 3) mientras  $a_n$  sea la última entrada verdadera utilizada.

Si  $a_1 a_2 \dots a_n$  no se encuentra en  $L(M)$ , según el Lema 10.3,  $M'$  no tendrá, después de un cierto tiempo de haber leído a  $a_n$ , movimientos  $\epsilon$  para hacer y tendrá que utilizar un símbolo de entrada verdadero. Pero, en este momento, la segunda componente del estado de  $M'$  es 2, ya que  $a_1 a_2 \dots a_n$  no se encuentra en  $L(M)$ . Mediante la regla (2),  $M'$  aceptará antes de intentar utilizar un símbolo de entrada verdadero.  $\square$

Antes de concluir esta sección establecemos el siguiente corolario.

**Corolario** Cada CFL determinístico es aceptado por algún DPDA que, en un estado de aceptación, puede no hacer ningún movimiento sobre la entrada  $\epsilon$ .

*Demostración* La proposición se encuentra de manera implícita en la demostración del Teorema 10.1. Nótese que en un estado final (uno en el cual  $k = 3$ ) ningún movimiento  $\epsilon$  es posible.  $\square$

Es posible utilizar el Teorema 10.1 para mostrar que ciertos lenguajes no son DCFLs.

**Ejemplo 10.1** El lenguaje  $L = \{0^i 1^j 2^k \mid i = j \text{ o } j = k\}$  es un CFL generado por la gramática

$$S \rightarrow AB \mid CD \quad A \rightarrow 0A1 \mid \epsilon$$

$$B \rightarrow 2B \mid \epsilon \quad C \rightarrow 0C \mid \epsilon \quad D \rightarrow 1D2 \mid \epsilon$$

Sin embargo,  $L$  no es un DCFL. Si sí lo fuera, entonces  $\bar{L}$  sería un DCFL y en consecuencia un CFL. Según el Teorema 6.5,  $L_i = \bar{L} \cap 0^* 1^* 2^*$  sería un CFL. Pero  $L_i =$

$\{0^i 1^j 2^k \mid i \neq j \text{ y } j \neq k\}$ . Una demostración, utilizando el lema de Odgen, parecida a la del Ejemplo 6.3 muestra que  $L_1$  no es un CFL, de modo que  $L$  no es un DCFL.

### 10.3 MAQUINAS DE PREDICCION

Para establecer un cierto número de otras propiedades de cerradura de los DCFLs, necesitamos una construcción en la cual los símbolos de pila del DPDA  $M$  sean modificados para que contengan información con respecto a cierto autómata finito  $A$ . La información asociada con el símbolo en el tope de la pila dice, para cada estado  $q$  de  $M$  y  $p$  de  $A$ , si existe alguna cadena de entrada que ocasione que  $M$  acepte cuando se le inicia en el estado  $q$  con su pila actual, y que simultáneamente ocasione que  $A$  acepte si se le inició en el estado  $p$ .

De manera formal, sea  $M = (Q_M, \Sigma, \Gamma, \delta_M, q_0, Z_0, F_M)$  un DPDA en forma normal y  $A = (Q_A, \Sigma, \Delta, \delta_A, p_0, F_A)$ . Entonces  $\pi(M, A)$ , la máquina de predicción para  $M$  y  $A$ , se define mediante  $(Q_M, \Sigma, \Gamma \times \Delta, \delta, q_0, X_0, F_M)$ , en donde  $\Delta$  es el conjunto de subconjuntos de  $Q_M \times Q_A$ . La intención es que si  $\pi(M, A)$  se encuentra en una  $ID(r, x, [Z, \mu]y)$ , entonces  $\mu$  consiste en exactamente los pares  $(q, p)$  tales que existe una  $w$  en  $\Sigma^*$  para la cual  $\delta_A(p, w)$  se encuentra en  $F_A$ , y  $(q, w, Z\beta) \vdash_M^*(s, \epsilon, \alpha)$  para alguna  $s$  en  $F_M$  y  $\alpha$  y  $\beta$  en  $\Gamma^*$ , en donde  $\beta$  es la cadena de las primeras componentes de  $y$ .

Para definir a  $\delta$  y  $X_0$  necesitamos una notación adicional. Sea  $M_{q, Z}$  con  $q$  y  $Z$  el estado inicial y el símbolo de inicio, respectivamente. Hagamos que  $A_p$  sea  $A$  con  $p$  el estado inicial. Entonces, según nuestra notación usual,

$$L(M_{q, Z}) = \{w \mid (q, w, Z) \vdash_M^*(s, \epsilon, \gamma) \text{ para alguna } s \text{ en } F_M \text{ y } \gamma \text{ en } \Gamma^*\}.$$

y

$$L(A_p) = \{w \mid \delta_A(p, w) \text{ está en } F_A\}.$$

Hagamos  $N_r(M_{q, Z})$  el conjunto de cadenas que ocasiona que  $M_{q, Z}$  borre su pila y accese el estado  $r$ , esto es,

$$N_r(M_{q, Z}) = \{w \mid (q, w, Z) \vdash_M^*(r, \epsilon, \epsilon)\}.$$

Es seguro que  $L(M_{q, Z})$  es un DCFL y que  $L(A_p)$  es un conjunto regular. También es verdad que  $N_r(M_{q, Z})$  es un DCFL. Para efecto de la demostración, modifíquese  $M$  para que coloque un señalador  $Y_0$  en el fondo de la pila y después que simule a  $M$  en el estado  $q$ , con la pila  $ZY_0$ . Si  $Y_0$  llega al tope de la pila, entonces acepta si el estado es  $r$  y, si no sucede esto, entonces rechaza. Finalmente, sea  $L_r(A_p) = \{w \mid \delta_A(p, w) = r\}$ . Es claro que  $L_r(A_p)$  es regular.

Ahora estamos en posición de definir a  $\delta(r, a, [Z, \mu])$ , para  $r$  en  $Q_M$ ,  $a$  en  $\Sigma \cup \{\epsilon\}$ ,  $Z$  en  $\Gamma$ , y  $\mu$  en  $\Delta$ , de la manera siguiente:

1. Si  $\delta_M(r, a, Z) = (s, \epsilon)$ , entonces  $\delta(r, a, [Z, \mu]) = (s, \epsilon)$ . Nótese que  $\mu$  no influye en la acción de  $\pi(M, A)$ , excepto en la regla (3), dada más abajo, en donde tiene influencia sobre la segunda componente del símbolo de pila presionada.
2. Si  $\delta_M(r, a, Z) = (s, Z)$ , entonces  $\delta(r, a, [Z, \mu]) = (s, [Z, \mu])$ .

3. Si  $\delta_M(r, a, Z) = (s, YZ)$ , entonces  $\delta(r, a, [Z, \mu]) = (s, [Y, v][Z, \mu])$ , en donde  $v$  consiste en aquellos pares  $(q, p)$  tales que cumplen con

- a)  $L(M_{q, Y}) \cap L(A_p)$  es no vacío, o
- b) existe alguna  $t$  en  $Q_M$  y alguna  $u$  en  $Q_A$  tales que

$$N_t(M_{q, Y}) \cap L_u(A_p)$$

es no vacío y  $(t, u)$  se encuentra en  $\mu$ .

Nótese que  $L(M_{q, Y})$  y  $N_t(M_{q, Y})$  son CFLs y que  $L(A_p)$  y  $L_u(A_p)$  son regulares, de modo que según los Teoremas 6.5 y 6.6, podemos determinar si los lenguajes mencionados en (a) y (b) son vacíos o no.

Finalmente, sea  $X_0 = [Z_0, \mu_0]$ , en donde

$$\mu_0 = \{(q, p) \mid L(M_{q, Z_0}) \cap L(A_p) \neq \emptyset\}.$$

**Lema 10.4**  $\pi(M, A)$  como se definió arriba tiene la propiedad de que

$$(q_0, x, [Z_0, \mu_0]) \vdash_{\pi(M, A)}^*(r, y, [Z_1, \mu_1][Z_2, \mu_2] \cdots [Z_n, \mu_n])$$

si y sólo si

- a)  $(q_0, x, Z_0) \vdash_M^*(r, y, Z_1 Z_2 \cdots Z_n)$ , y
- b) para  $1 \leq i \leq n$ ,

$$\mu_i = \{(q, p) \mid \text{para alguna } w, (q, w, Z_i Z_{i+1} \cdots Z_n) \vdash_M^* (s, \epsilon, \gamma) \text{ para alguna } s \text{ en } F_M \text{ y } \gamma \text{ en } \Gamma^*, y \delta_A(p, w) \text{ en } F_A\}.$$

**Demostración** La parte (a) es obvia, ya que  $\pi(M, A)$  simula a  $M$ , llevando consigo a la segunda componente de los símbolos de pila, pero sin permitirles influir en nada que no sea otra segunda componente de símbolos de pila.

Demostraremos (b) por inducción en  $i$ , comenzando con  $i = n$  y procediendo hacia abajo. La base,  $i = n$ , es sencilla.  $Z_n$  debe ser  $Z_0$ , ya que  $M$  se encuentra en forma normal. La definición de  $X_0$  más la regla (2) de la definición de  $\delta$  nos dan la base.

Para la inducción, supóngase que el resultado es verdadero par  $i + 1$ . Entonces  $\mu_i$  se construyó a partir de  $\mu_{i+1}$  conforme  $v$  se construía a partir de  $\mu$  en la regla (3). Supóngase que existe alguna  $w$  tal que

$$(q, w, Z_i Z_{i+1} \cdots Z_n) \vdash_M^* (s, \epsilon, \gamma)$$

para  $s$  en  $F_M$  y  $\delta_A(p, w)$  en  $F_A$ . Entonces existen dos casos, dependiendo de si  $Z_i$  es o no borrado. Si no sucede esto, entonces  $w$  se encuentra en  $L(M_{q, Z_i})$  y también en  $L(A_p)$ , así que según la regla (3a),  $(q, p)$  se encuentra en  $\mu_i$ . Si  $Z_i$  es borrado, hagamos  $w = w_1 w_2$ , en donde

$$(q, w_1, Z_i) \vdash_M^* (t, \epsilon, \epsilon) \quad y \quad (t, w_2, Z_{i+1} Z_{i+2} \cdots Z_n) \vdash_M^* (s, \epsilon, \gamma)$$

para alguna  $s$  en  $F_M$ . Sea también  $\delta_A(p, w_1) = u$ , de modo que  $\delta_A(u, w_2)$  se encuentra en  $F_A$ . Entonces  $w_1$  se encuentra en  $N_t(M_{q, Z_i})$  y también en  $L_u(A_p)$ . Mediante la hipótesis inductiva,  $(t, u)$  se encuentra en  $\mu_{i+1}$ . Por consiguiente, según la regla (3b),  $(q, p)$  está en  $\mu_i$ .

De manera inversa, si  $(q, p)$  se encuentra en  $\mu_i$  según la regla (3a), entonces existe una  $w$  tal que  $\delta_A(p, w)$  está en  $F_A$  y  $(q, w, Z_i Z_{i+1} \cdots Z_n) \stackrel{*}{\vdash_M} (s, \epsilon, \gamma)$  para  $s$  en  $F_M$ , mediante una secuencia de movimientos en los que  $Z_i$  nunca es borrado. Si  $(q, p)$  se encuentra en  $\mu_i$  mediante la regla (3b) entonces existe  $w_1$ , en  $\Sigma^*$ ,  $t$  en  $Q_M$  y  $u$  en  $Q_A$  tales que  $(q, w_1, Z_i) \stackrel{*}{\vdash_M} (t, \epsilon, \epsilon)$ ,  $\delta_A(p, w_1) = u$ , y  $(t, u)$  se encuentra en  $\mu_{i+1}$ . Según la hipótesis de inducción, existe  $w_2$  en  $\Sigma^*$  tal que  $(t, w_2, Z_{i+1} Z_{i+2} \cdots Z_n) \stackrel{*}{\vdash_M} (s, \mathcal{C}, g)$  para alguna  $s$  en  $F_M$  y  $d_A(u, w_2)$  en  $F_A$ . Por consiguiente  $(q, w_1 w_2, Z Z_{i+1} \cdots Z_n) \stackrel{*}{\vdash_M} (s, \mathcal{C}, g)$  y  $d_A(p, w_1 w_2)$  se encuentra en  $F_A$ , de modo que  $(q, p)$  pertenece a  $\mu_i$ . Esto completa la inducción y la demostración del lema.  $\square$

**Ejemplo 10.2** Sea

$$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{X, Z_0\}, \delta_M, q_0, Z_0, \{q_3\}),$$

en donde

$$\begin{aligned} \delta_M(q_0, 0, Z_0) &= (q_0, XZ_0), & \delta_M(q_1, 0, X) &= (q_2, \epsilon), \\ \delta_M(q_0, 0, X) &= (q_0, XX), & \delta_M(q_2, 0, X) &= (q_2, \epsilon), \\ \delta_M(q_0, 1, X) &= (q_1, XX), & \delta_M(q_2, \epsilon, Z_0) &= (q_3, \epsilon), \\ \delta_M(q_1, 1, X) &= (q_1, XX), \end{aligned}$$

Hágase también  $A = (\{p_0, p_1\}, \{0, 1\}, \delta_A, p_0, \{p_0\})$ , en donde

$$\begin{aligned} \delta_A(p_0, 0) &= p_1, & \delta_A(p_0, 1) &= p_0, \\ \delta_A(p_1, 0) &= p_0, & \delta_A(p_1, 1) &= p_1. \end{aligned}$$

Obsérvese que

$$L(M) = L(M_{q_0, Z_0}) = \{0^i 1^j 0^k \mid i + j = k, i > 0 \text{ y } j > 0\}.$$

También que  $L(M_{q_1}, Z_0) = \emptyset$  y que  $L(M_{q_2}, Z_0) = L(M_{q_3}, Z_0) = \{\epsilon\}$ .

$$L(A) = L(A_{p_0}) = (1 + 01^*0)^*;$$

esto es, las cadenas con un número par de 0s, y  $L(A_{p_1}) = 1^*0(1 + 01^*0)^*$  esto es, las cadenas con un número impar de 0s. Por tanto  $L(M_{q_0}, Z_0) \cap L(A_{p_0})$  contiene cadenas como 00110000, y  $L(M_{q_0}, Z_0) \cap L(A_{p_1})$  contiene cadenas como 01110000.  $L(M_{q_2}, Z_0) \cap L(A_{p_0})$  y  $L(M_{q_3}, Z_0) \cap L(A_{p_0})$ , cada una, contienen a  $\epsilon$ , pero las otras cuatro intersecciones de la forma  $L(M_{q_i}, Z_0) \cap L(A_{p_j})$  se encuentran vacías. Por consiguiente el símbolo inicial de  $\pi(M, A)$  es  $[Z_0, \mu_0]$ , en donde

$$\mu_0 = \{(q_0, p_0), (q_0, p_1), (q_2, p_0), (q_3, p_0)\}.$$

Calculemos ahora

$$\delta(q_0, 0, [Z_0, \mu_0]) = (q_0, [X, v][Z_0, \mu_0]).$$

Para llevar esto a cabo, necesitamos deducir que  $L(M_{q_i}, x) = \emptyset$  para  $i = 0, 1$  o  $2$ , ya que no podemos aceptar sin tener un  $Z_0$  en la pila y no podemos escribir  $Z_0$  si no se en-

contraba en ésta originalmente. Así pues, no existe contribución a  $v$  de la regla (3a). Sin embargo,  $L(M_{q_3}, x) \cap L(A_{p_0}) = \{\epsilon\}$ , de modo que agregamos  $(q_3, p_0)$  a  $v$ .

Considérese la regla (3b).

$$N_{q_2}(M_{q_0, x}) = \{0^i 1^j 0^k \mid i + j = k - 1 \text{ and } j > 0\},$$

$$N_{q_2}(M_{q_1, x}) = \{1^j 0^k \mid j = k - 1\},$$

y

$$N_{q_2}(M_{q_2, x}) = \{0\}.$$

Los otros conjuntos de la forma  $N_{q_i}(M_{q_j, x})$  están vacíos. También  $L_{p_i}(A_{p_j})$  consiste en todas las cadenas que tienen un número par de 0s si  $i = j$ , y en todas las cadenas que tienen un número impar de 0s si  $i \neq j$ .

Como  $N_{q_i}(M_{q_j, x})$  es no vacío sólo si  $i = 2$  y  $j = 0, 1$ , o 2 solamente podemos aplicar como éxito la regla (3b) si el par  $(q_2, p_0)$  se selecciona de  $\mu_0$ . Vemos que tanto  $N_{q_2}(M_{q_0, x}) \cap L_{p_0}(A_{p_0})$  como  $N_{q_2}(M_{q_1, x}) \cap L_{p_0}(A_{p_1})$  son no vacíos, produciendo, para  $v, (q_0, p_0)$  y  $(q_0, p_1)$ . De manera similar,  $N_{q_2}(M_{q_1, x}) \cap L_{p_0}(A_{p_0})$  y  $N_{q_2}(M_{q_1, x}) \cap L_{p_1}(A_{p_1})$  son no vacíos, produciendo, para  $v, (q_1, p_0)$  y  $(q_1, p_1)$ . También  $N_{q_2}(M_{q_3, x}) \cap L_{p_0}(A_{p_1})$  es no vacío, produciendo, para  $v, (q_2, p_1)$ , pero

$$N_{q_2}(M_{q_2, x}) \cap L_{p_0}(A_{p_0}) = \emptyset.$$

Por consiguiente

$$v = \{(q_0, p_0), (q_0, p_1), (q_1, p_0), (q_1, p_1), (q_2, p_1), (q_3, p_0)\}.$$

## 10.4 PROPIEDADES DE CERRADURA ADICIONALES DE LOS DCFLs

Utilizando la idea desarrollada en la sección anterior, podemos demostrar unas cuantas propiedades de cerradura de los lenguajes determinísticos libres de contexto. Antes de seguir adelante presentaremos un lema más técnico. El lema afirma que podemos definir la aceptación para un DPDA mediante una combinación del estado y del símbolo en el tope de la pila; el lenguaje definido de esta manera es aún un lenguaje determinístico.

**Lema 10.5** Sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$  un DPDA. Hagamos que  $B$  sea cualquier subconjunto de  $Q \times \Gamma$ , esto es, pares formados con estados y símbolos de pila. Defínase.

$$L = \{w \mid (q_0, w, Z_0) \stackrel{*}{\vdash_M} (q, \epsilon, Z\gamma) \text{ para algún } (q, Z) \text{ en } B\}.$$

Entonces  $L$  es un DCFL.

**Demostración** Definimos un DPDA  $M'$  que acepta a  $L$ , de la manera siguiente.

$$M' = (Q', \Sigma, \Gamma, \delta', q_0, Z_0, F'),$$

en donde

$$Q' = \{q, q', q'' \mid q \text{ en } Q\} \quad \text{y} \quad F' = \{q'' \mid q \text{ en } Q\}.$$

$M'$  hace los mismos movimientos que  $M$ , excepto que  $M'$  se mueve de un estado no primado a uno que tiene solamente una prima y después, sobre la entrada  $\epsilon$ ; se regresa de nuevo al correspondiente estado no primado, ya sea directamente o mediante una versión con dos primas. Este último caso se aplica solamente si el par formado por el estado y el símbolo en el tope de la pila se encuentra en  $B$ .

Formalmente,

1. si  $\delta(q, a, Z) = (p, \gamma)$ , entonces  $\delta'(q, a, Z) = (p', \gamma)$ ;
2.  $\delta'(q', \epsilon, Z) = (q, Z)$  siempre que  $(q, Z)$  no se encuentre en  $B$ ;
3.  $\delta'(q', \epsilon, Z) = (q'', Z)$  y  $\delta'(q'', \epsilon, Z) = (q, Z)$  si  $(q, Z)$  se encuentra en  $B$ .  $\square$

### Cociente con un conjunto regular

Recuérdese que el cociente de  $L_1$  con respecto a  $L_2$ , denotado por  $L_1/L_2$ , es

$$\{x \mid \text{existe } w \text{ en } L_2 \text{ tal que } xw \text{ está en } L_1\}.$$

En el Ejercicio 6.4 exigimos que los CFLs fueran cerrados con respecto al cociente con un conjunto regular. (Véase el Teorema 11.3 para una demostración). Demostraremos ahora un resultado parecido para los DCFLs.

**Teorema 10.2** Sea  $L$  un DCFL y  $R$  un conjunto regular. Entonces  $L/R$  es un DCFL.

*Demostración* Sea  $L = L(M)$  para el DPDA  $M$  que siempre barre su entrada completa. Sea  $R = L(A)$  para el autómata finito  $A$ . Supóngase que  $M = (Q_M, \Sigma, \Gamma, \delta_M, q_0, Z_0, F_M)$  y  $A = (Q_A, \Sigma, \Delta, p_0, F_A)$ . Entonces sea

$$M' = (Q_M, \Sigma, \Gamma \times \Delta, \delta, q_0, [Z_0, \mu_0], F_M)$$

una  $\pi(M, A)$ , la máquina de predicción para  $M$  y  $A$ . Hagamos  $B$  el subconjunto de  $Q_M \times (\Gamma \times \Delta)$  que contiene a todos los pares  $(q, [Z, \mu])$  tales que  $(q, p_0)$  se encuentra en  $\mu$ .

Entonces, según el Lema 10.5,

$$L_1 = \{x \mid (q_0, x, [Z_0, \mu_0]) \vdash_{M'}^* (q, \epsilon, [Z, \mu]\gamma) \text{ y } (q, p_0) \text{ está en } \mu\}$$

es un DCFL. Según el Lema 10.4,

$$L_1 = \{x \mid \text{para alguna } w \text{ en } \Sigma^*, (q_0, x, Z_0) \vdash_M^* (q, \epsilon, Z\gamma') \\ y (q, w, Z\gamma') \vdash_M^* (s, \epsilon, \beta),\}$$

en donde  $s$  está en  $F_M$ ,  $\gamma'$  es la primera componente de  $\gamma$  y  $\delta_A(p_0, w)$  está en  $F_A$ .

De manera equivalente,

$$L_1 = \{x \mid \text{para alguna } w \text{ en } \Sigma^*, xw \text{ está en } L(M) \text{ y } w \text{ en } L(A)\}.$$

Esto es,  $L_1 = L/R$ . Por tanto  $L/R$  es un DCFL.  $\square$

### MIN y MAX

Mostraremos ahora dos operaciones que preservan a los DCFLs, pero no a un CFL cualquiera.

Recuérdese que para cada lenguaje  $L$ :

$$\text{MIN}(L) = \{x \mid x \text{ está en } L \text{ y ninguna } w \text{ de } L \text{ es prefijo propio de } x\},$$

y

$$\text{MAX}(L) = \{x \mid x \text{ está en } L \text{ y } x \text{ no es un prefijo propio de ninguna palabra de } L\}.$$

**Ejemplo 10.3** Sea

$$L = \{0^i 1^j 0^k \mid i, j, k > 0, i + j \geq k\}.$$

Entonces  $\text{MIN}(L) = 00*11*0$  y  $\text{MAX}(L) = \{0^i 1^j 0^{i+j} \mid i, j \geq 0\}$ .

**Teorema 10.3** Si  $L$  es un DCFL, entonces  $\text{MIN}(L)$  y  $\text{MAX}(L)$  son DCFLs.

*Demostración* Sea  $M = (Q_M, \Sigma, \Gamma, \delta_M, q_0, Z_0, F_M)$  un DPDA que acepta a  $L$  y que siempre barre a su entrada completa. Modifíquese a  $M$  para que no haga movimientos en un estado final. Entonces el DPDA resultante  $M'$  acepta a  $\text{MIN}(L)$ . Para la demostración, si  $w$  se encuentra en  $\text{MIN}(L)$ , entonces sea

$$(q_0, w, Z_0) = I_0 \vdash_{M'} I_1 \vdash_{M'} \cdots \vdash_{M'} I_m \quad (10.1)$$

la secuencia de IDs accesadas por  $M$ , donde  $I_m = (q, \epsilon, \gamma)$  para alguna  $\gamma$ , y  $q$  en  $F_M$ . Aún más, puesto que  $w$  se encuentra en  $\text{MIN}(L)$ , ninguna de las  $I_0, I_1, \dots, I_{m-1}$  tiene un estado de aceptación. Por consiguiente (10.1) es también un cálculo de  $M'_1$ , de modo que  $w$  se encuentra en  $L(M)$ .

De manera inversa, si  $(q_0, w, Z_0) = I_0 \vdash_{M'_1} I_1 \vdash_{M'_1} \cdots \vdash_{M'_1} I_m$  es un cálculo aceptado de  $M'_1$ , entonces ninguno de los  $I_0, I_1, \dots, I_{m-1}$  tiene un estado de aceptación. Por consiguiente  $w$  se encuentra en  $\text{MIN}(L)$ .

Para el caso de MAX debemos hacer uso de la máquina de predicción. Sea  $A = (Q_A, \Sigma, \delta_A, p_0, F_A)$  el FA simple de la Fig. 10.1 que acepta a  $\Sigma^*$ . Sea  $M = (Q_M, \Sigma, \Gamma \times \Delta, \delta, q_0, [Z_0, \mu_0], F_M)$  la  $\pi(M, A)$ . Y sea  $B = \{(q, [Z, \mu]) \mid q \text{ está en } F_M \text{ y } (q, p_0) \text{ no se encuentra en } \mu\}$ . Entonces, según el Lema 10.5,

$$L_1 = \{x \mid (q_0, x, Z_0) \vdash_M^* (q, \epsilon, \gamma)$$

para alguna  $q$  en  $F_M$ , y para ninguna  $w \neq \epsilon$  sucede que  $(q, w, \gamma) \vdash^* (s, \epsilon, \beta)$  para  $s$  en  $F_M\}$

es un DCFL. Pero  $L_1 = \text{MAX}(L)$ , en consecuencia  $\text{MAX}(L)$  es un DCFL.  $\square$

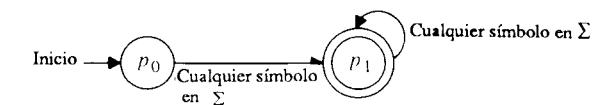


Fig. 10.1 El autómata  $A$ .

**Ejemplo 10.4** Utilicemos el Teorema 10.3 para mostrar un CFL que no es DCFL. Sea  $L_1 = \{0^i 1^j 2^k \mid k \leq i \text{ o } k \leq j\}$ . Entonces  $L_1$  es un CFL generado por la gramática

$$\begin{aligned} S &\rightarrow AB \mid C \\ A &\rightarrow 0A \mid \epsilon \\ B &\rightarrow 1B2 \mid 1B \mid \epsilon \\ C &\rightarrow 0C2 \mid 0C \mid D \\ D &\rightarrow 1D \mid \epsilon \end{aligned}$$

Supóngase que  $L_1$  es un DCFL. Entonces  $L_2 = \text{MAX}(L_1)$  sería un DCFL y por tanto un CFL. Pero  $L_2 = \{0^i 1^j 2^k \mid k = \max(i, j)\}$ . Supóngase que  $L_2$  es un CFL. Sea  $n$  la constante del lema de sondeo y considérese  $z = uvwxy = 0^n 1^n 2^n$ . Si ni  $v$  ni  $x$  tienen un 2, entonces  $z' = uv^2 w x^2 y$  tiene  $n$  2s y al menos  $(n+1)$  0s o, cuando menos  $(n+1)$  1s. Por consiguiente  $z'$  no se encontraría en  $L_2$  como se supuso.

Considérese, ahora, el caso en el que  $vx$  tiene un 2. Si  $v$  o  $x$  tiene más de un símbolo, entonces  $z' = uv^2 w x^2 y$  no es de la forma  $0^i 1^j 2^k$  y no sería  $L_2$ . Por tanto 0 o 1 no se encuentra presente en  $vx$ . Por consiguiente  $wxy$  tiene menos de  $n$  2s pero tiene  $n$  0s o  $n$  1s y no se encuentra en  $L_2$ . Concluimos que  $L_2$  no es un CFL, de modo que  $L_1$  no es un DCFL.

### Otras propiedades de cerradura

Como regla general, solamente aquellas propiedades de cerradura de los CFLs que se mencionaron en la Sección 6.2, y que se demostraron utilizando la caracterización del PDA, pueden trasladarse a los DCFLs. En particular podemos establecer lo siguiente.

**Teorema 10.4** Los DCFLs son cerrados con respecto a (a) homomorfismo inverso y (b) la intersección con un conjunto regular.

*Demostración* Los argumentos utilizados en los Teoremas 6.3 y 6.5 funcionan para los DPDAs.  $\square$

**Teorema 10.5** Los DCFLs no son cerrados con respecto a (a) homomorfismo (b) unión, (c) concatenación o (d) cerradura de Kleene.

*Demostración* Véase el Ejercicio 10.4 y su solución.  $\square$

## 10.5 PROPIEDADES DE DECISION DE LOS DCFLs

Un cierto número de problemas que son irresolubles para los CFLs, son solubles para los DCFLs.

**Teorema 10.6** Sea  $L$  un DCFL y  $R$  un conjunto regular. Los siguientes problemas son resolvibles.

1. ¿Es  $L = R$ ?
2. ¿Es  $R \subseteq L$ ?

3. ¿Es  $\bar{L} = \emptyset$ ?

4. ¿Es  $\bar{L}$  un CFL?

5. ¿Es  $\bar{L}$  regular?

### Demostración

1.  $L = R$  si y sólo si  $L_1 = (L \cap \bar{R}) \cup (\bar{L} \cap R)$  es el vacío. Puesto que los DCFLs son cerrados efectivamente con respecto a la complementación y la intersección con un conjunto regular y como los CFLs son cerrados efectivamente con respecto a la unión,  $L_1$  es un CFL, y el problema del vacío para los CFLs es resoluble.

2.  $R \subseteq L$  si y sólo si  $\bar{L} \cap R = \emptyset$ . Puesto que  $\bar{L} \cap R$  es un CFL,  $\bar{L} \cap R = \emptyset$  es resoluble.

3. Como los DCFLs son cerrados efectivamente con respecto a la complementación,  $\bar{L}$  es un DCFL y, en consecuencia,  $\bar{L} = \emptyset$  es resoluble.

4. La propiedad  $\bar{L}$  es un CFL es trivial para los DCFL y, en consecuencia, es resoluble.

5. La regularidad para los DCFLs es resoluble. La demostración es larga y remitimos al lector a Stearns [1967] o Valiant [1975b].  $\square$

### Propiedades irresolubles de los DCFLs

Algunas otras propiedades irresolubles de los CFLs permanecen irresolubles aun cuando se restringen a los DCFLs. Muchos de estos problemas pueden demostrarse irresolubles observando que los lenguajes  $L_1$  y  $L_2$  de la Sección 8.6, cuya intersección son los cálculos válidos de una máquina de Turing  $M$ , son DCFLs.

**Teorema 10.7** Sean  $L$  y  $L'$  DCFLs arbitrarios. Entonces los siguientes problemas son irresolubles.

1. ¿Es  $L \cap L' = \emptyset$ ?
2. ¿Es  $L \subseteq L'$ ?
3. ¿Es  $L \cap L'$  un DCFL?
4. ¿Es  $L \cap L'$  un CFL?
5. ¿Es  $L \cup L'$  un DCFL?

*Demostración* Dada una TM cualquiera  $M$ , en el Lema 8.6, mostraremos la manera de construir los lenguajes  $L_1$  y  $L_2$  tales que  $L_1 \cap L_2 = \emptyset$  si y sólo si  $L(M) = \emptyset$ . Resulta fácil demostrar que  $L_1$  y  $L_2$  son DCFLs si presentamos DPDAs que los acepten. Por consiguiente (1) se concluye de manera inmediata si tomamos en cuenta el hecho de que la cuestión de si  $L(M) = \emptyset$  es irresoluble. Partiendo de que las DCFLs son cerrados con respecto al complemento, y de que  $L \subseteq L'$  si y sólo si  $L \cap L' = \emptyset$ , (2) se concluye de (1).

Para demostrar (3), (4) y (5), modifíquese cada TM  $M$  para que efectúe al menos dos movimientos antes de aceptar, como se hizo en el Lema 8.8. Entonces  $L_1 \cap L_2$  es un conjunto finito (en cuyo caso seguramente es un CFL y un DCFL) o no es un CFL, dependiendo de si  $L(M)$  es finito o no. Por tanto la solubilidad de (3) o (4) implicaría la solubilidad de la finitud para  $L(M)$ , que, sabemos, es una propiedad irresoluble.

Puesto que los DCFLs son cerrados con respecto a la complementación, tomar la decisión de si  $L \cup L'$  es un DCFL es equivalente a decidir si  $L \cap L'$  es un DCFL. Por consiguiente (5) se concluye de (3).  $\square$

**Teorema 10.8** Sea  $L$  un CFL cualquiera La cuestión de si  $L$  es un DCFL es irresoluble.

*Demuestra*ción Hagamos que  $L$  sea el CFL de los cálculos no válidos de una TM  $M$  cualquiera que hace al menos dos movimientos en cada entrada.  $L$  es regular, y, por tanto, un DCFL si y sólo si  $M$  acepta un conjunto finito.  $\square$

Finalmente observamos que la cuestión de si dos DCFLs son equivalentes es un problema importante de la teoría de los lenguajes que aún permanece sin resolver.

## 10.6 GRAMATICAS LR(0)

Recuérdese que uno de los motivos para estudiar los DCFLs es su capacidad para describir la sintaxis de los lenguajes de programación. Varios sistemas compiladores de escritura requieren especificaciones sintácticas en la forma de CFGs restringidas, las cuales permiten solamente la representación de DCFLs. Más aún, el analizador gramatical producido por tales sistemas compiladores de escritura es esencialmente un DPDA. En la presente sección introduciremos un tipo restringido de CFG conocido como gramática  $LR(0)$ . Esta clase de gramática es la primera de una familia que se conoce de manera colectiva como gramáticas  $LR$ . Por cierto, cabe hacer notar que  $LR(0)$  significa “barido de izquierda a derecha (*left-to-right*, en inglés) de la entrada produciéndose una derivación derecha y utilizando 0 símbolos de adelanto† sobre la entrada”.

Las gramáticas  $LR(0)$  definen exactamente a los DCFLs que poseen la propiedad de prefijo (se dice que  $L$  tiene la *propiedad de prefijo* si, siempre que  $w$  esté en  $L$ , ningún prefijo propio de  $w$  está en  $L$ ). Nótese que la propiedad de prefijo no es una restricción severa, ya que la introducción de un señalador de extremo convierte a cualquier DCFL en un CFL con la propiedad de prefijo. Por consiguiente,  $L\$ = \{w\$ \mid w \text{ está en } L\}$  es un DCFL con la propiedad de prefijo, siempre que  $L$  sea un DCFL.

Mientras que la restricción  $LR(0)$  es muy severa para proporcionar gramáticas convenientes y naturales a muchos lenguajes de programación, la condición  $LR(0)$  captura la esencia de sus generalizaciones más útiles, mismas que serán discutidas en la Sección 10.8, y que se han utilizado con éxito en varios sistemas generadores de analizadores de gramática.

### Items LR

Para introducir las gramáticas  $LR(0)$  necesitamos algunas definiciones preliminares. En primer lugar, un ítem para una CFG dada es una producción en la que se tiene un punto en alguna parte del lado derecho, incluyendo el principio o el final. En el caso de una producción  $\epsilon, B \rightarrow \epsilon, B \rightarrow \cdot$  es un ítem

† Adelanto, *lookahead* en inglés, Véase la Sección 10.8. (N. del T.)

**Ejemplo 10.5** Introduciremos ahora una gramática que utilizaremos en una serie de ejemplos.

$$S' \rightarrow Sc \quad S \rightarrow SA \mid A \quad A \rightarrow aSb \mid ab \quad (10.2)$$

Esta gramática, con símbolo inicial  $S$ , genera cadenas de “paréntesis balanceados”, y trata a  $a$  y  $b$  como el paréntesis izquierdo y derecho, respectivamente, y  $c$  como un señalador de extremo. Los ítems de la gramática (10.2) son

$$\begin{array}{lll} S' \rightarrow \cdot Sc & S \rightarrow \cdot SA & A \rightarrow \cdot aSb \\ S' \rightarrow S \cdot c & S \rightarrow S \cdot A & A \rightarrow a \cdot Sb \\ S' \rightarrow Sc \cdot & S \rightarrow SA \cdot & A \rightarrow aS \cdot b \\ & S \rightarrow \cdot A & A \rightarrow aSb \cdot \\ & S \rightarrow A \cdot & A \rightarrow \cdot ab \\ & A \rightarrow a \cdot b & \\ & A \rightarrow ab \cdot & \end{array}$$

De aquí en adelante utilizaremos los símbolos  $\xrightarrow{*}$  y  $\xrightarrow{r m}$  para denotar derivaciones derechas y pasos simples en las derivaciones derechas, respectivamente. Una forma oracional derecha es una forma oracional que puede derivarse mediante una derivación derecha. Un *pivote* (*handle* en inglés) de una forma oracional derecha  $\gamma$  para la CFG  $G$  es una subcadena  $\beta$ , tal que

$$S \xrightarrow{r m} \delta Aw \xrightarrow{r m} \delta\alpha\beta w$$

y  $\delta\beta w = \gamma$ . Es decir un pivote, de  $\gamma$  es una subcadena que puede introducirse en el último paso de una derivación derecha de  $\gamma$ . Nótese que en este contexto, la posición de  $\beta$  dentro de  $\gamma$  es importante.

Un *prefijo viable* de una forma oracional derecha  $\gamma$  es cualquier prefijo de  $\gamma$  que termine no más a la derecha que el extremo derecho de un pivote de  $\gamma$ .

**Ejemplo 10.6** En la gramática (10.2) existe una derivación derecha

$$S' \Rightarrow Sc \Rightarrow SAc \Rightarrow SaSbc.$$

Por tanto  $SaSbc$  es una forma oracional derecha y su pivote es  $aSb$ . Nótese que en cualquier gramática inambigua en la que no haya símbolos inútiles, como el caso de la gramática (10.2), la derivación más a la derecha de una forma oracional derecha dada es única, de modo que su pivote también es único. Por consiguiente podemos hablar de “el pivote”, más que de “un pivote”. Los prefijos viables de  $SaSbc$  son  $\epsilon, S, Sa, SaS$  y  $SaSb$ .

Decimos que un ítem  $A \rightarrow \alpha \cdot \beta$  es *válido* para un prefijo viable  $\gamma$  si existe una derivación derecha

$$S \xrightarrow{r m} \delta Aw \xrightarrow{r m} \delta\beta w,$$

y  $\delta\alpha = \gamma$ . El conocimiento de cuáles ítems son válidos para un prefijo viable dado nos ayuda a encontrar una derivación derecha en forma inversa, de la manera siguiente. Se dice que un ítem es *completo* si el punto es el símbolo que se encuentra más a la derecha en el ítem. Si  $A \rightarrow \alpha \cdot$  es un ítem completo válido para  $\gamma$  entonces vemos que  $A \rightarrow \alpha$  podría ser utilizada en el último paso y que la forma oracional derecha anterior en la derivación de  $\gamma w$  es  $\delta Aw$ .

Por supuesto que no podemos más que sospechar lo anterior ya que  $A \rightarrow \alpha \cdot$  puede ser válida para  $\gamma$  debido a una derivación derecha  $S \xrightarrow{*} \delta Aw' \Rightarrow \gamma w'$ . Es claro que podrían existir dos o más ítems completos válidos para  $\gamma$ , o existir un pivote de  $\gamma w$  que incluya a símbolos de  $w$ . De manera intuitiva, una gramática está definida como  $LR(0)$  si en cada situación de éstas  $\delta Aw$  es, de hecho, la forma oracional derecha anterior para  $\gamma w$ . En ese caso, podemos iniciar con una cadena de terminales  $x$  que se encuentre en  $L(G)$  y, en consecuencia, que sea una forma oracional derecha de  $G$ , y trabajar en sentido inverso a las formas oracionales derechas anteriores hasta que lleguemos a  $S$ . Tendremos entonces una derivación derecha de  $x$ .

**Ejemplo 10.7** Considérese la gramática (10.2) y la forma oracional derecha  $abc$ . Como

$$S' \xrightarrow{rm} Ac \Rightarrow abc.$$

vemos que  $A \rightarrow ab \cdot$  es válida para el prefijo viable  $ab$ . Vemos también que  $A \rightarrow a \cdot b$  es válida para el prefijo viable  $a$  y que  $A \rightarrow \cdot ab$  es válida para el prefijo viable  $\epsilon$ . Puesto que  $A \rightarrow ab \cdot$  es un ítem completo, debemos ser capaces de deducir que  $Ac$  es la forma oracional derecha anterior para  $abc$ .

### Conjuntos de cálculo para ítems válidos

La definición de las gramáticas  $LR(0)$  y el método para la aceptación de  $L(G)$  por la gramática  $LR(0)$   $G$  mediante un DPDA, cada uno, dependen del conocimiento del conjunto de ítems válidos para cada prefijo viable  $\gamma$ . Resulta que para cada CFG  $G$ , el conjunto de prefijos viables es un conjunto regular, y este conjunto regular es aceptado por un NFA cuyos estados son los ítems para  $G$ . Aplicando la construcción de subconjunto a este NFA obtenemos un DFA cuyo estado, en respuesta al prefijo viable  $\gamma$ , es el conjunto de ítems válidos de  $\gamma$ .

El NFA  $M$  que reconoce a los prefijos viables para la CFG  $G = (V, T, P, S)$  se define de la manera siguiente. Sea  $M = (Q, V \cup T, \delta, q_0, Q)$ , en donde  $Q$  es el conjunto de ítems para  $G$  más el estado  $q_0$ , que no es un ítem. Defínase.

1.  $\delta(q_0, \epsilon) = \{S \rightarrow \cdot \alpha \mid S \rightarrow \alpha \text{ es una producción}\},$
2.  $\delta(A \rightarrow \alpha \cdot B\beta, \epsilon) = \{B \rightarrow \cdot \gamma \mid B \rightarrow \gamma \text{ es una producción}\},$
3.  $\delta(A \rightarrow \alpha \cdot X\beta, X) = \{A \rightarrow \alpha X \cdot \beta\}.$

La regla (2) permite la expansión de una variable  $B$ , que aparece inmediatamente a la derecha del punto. La regla (3) permite el movimiento del punto sobre cualquier símbolo de gramática  $X$ , si  $X$  es el siguiente símbolo de entrada.

**Ejemplo 10.8** El NFA para la gramática (10.2) se muestra en la Fig. 10.2

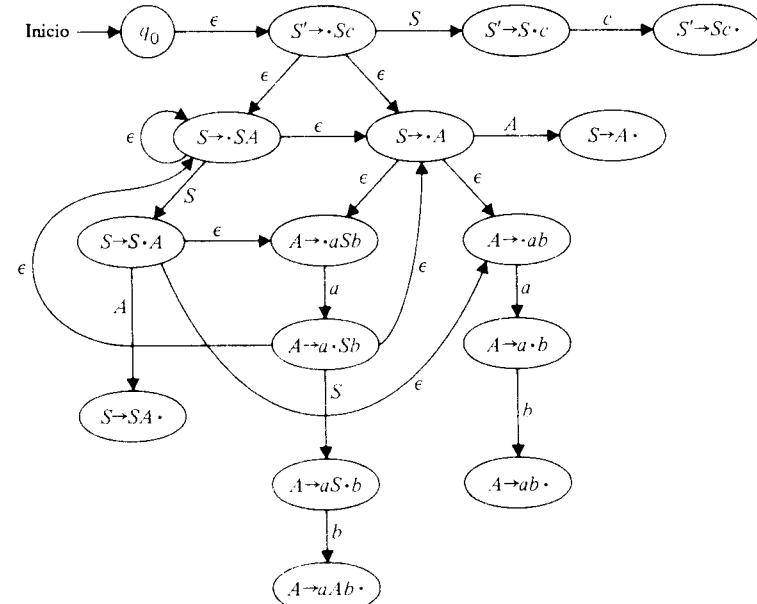


Fig. 10.2 NFA que reconoce los prefijos viables de la Gramática (10.2).

**Teorema 10.9** El NFA  $M$  definido más arriba tiene la propiedad de que  $\delta(q_0, \gamma)$  contiene a  $A \rightarrow \alpha \cdot \beta$  si y sólo si  $A \rightarrow \alpha \cdot \beta$  es válido para  $\gamma$ .

### Demostración

*Sólo si:* Debemos mostrar que cada ítem  $A \rightarrow \alpha \cdot \beta$  contenido en  $\delta(q_0, \gamma)$  es válido para  $\gamma$ . Procederemos mediante inducción en la longitud de la trayectoria etiquetada con  $\gamma$  más corta de  $q_0$  a  $A \rightarrow \alpha \cdot \beta$  en el diagrama de transiciones de  $M$ . La base (longitud 1) es sencilla. Las únicas trayectorias de longitud uno que parten de  $q_0$  tienen etiqueta  $\epsilon$  y van a ítems de la forma  $S \rightarrow \cdot \alpha$ . Cada uno de estos ítems es válido para  $\epsilon$  debido a la derivación derecha  $S \xrightarrow{rm} \alpha$ .

Para la inducción, supóngase que el resultado es verdadero para trayectorias más cortas que  $k$ , y hagamos que exista una trayectoria de longitud  $k$ , etiquetada con  $\gamma$ , que va de  $q_0$  a  $A \rightarrow \alpha \cdot \beta$ . Existen dos casos, dependiendo de si la etiqueta de la última arista es  $\epsilon$  o no.

**CASO1** La última arista tiene etiqueta  $X$ , para  $X$  en  $V \cup T$ . La arista debe partir de un estado  $A \rightarrow \alpha \cdot X\beta$ , en el cual  $\alpha = \alpha'X$ . Entonces, por la hipótesis inductiva,  $A \rightarrow \alpha' \cdot X\beta$  es válido para  $\gamma'$ , en donde  $\gamma = \gamma'X$ . Por consiguiente, existe una derivación derecha

$$S \xrightarrow{rm} \delta Aw \xrightarrow{rm} \delta\alpha' X\beta w,$$

en donde  $\delta\alpha' = \gamma'$ . Esta misma derivación muestra que  $A \rightarrow \alpha' X \cdot \beta$  (el cual es  $A \rightarrow \alpha \cdot \beta$ ) es válido para  $\gamma$ .

**CASO 2** La última arista tiene etiqueta  $\in$ . En este caso  $\alpha$  debe ser  $\in$ , y  $A \rightarrow \alpha \cdot \beta$  es en realidad  $A \rightarrow \cdot \beta$ . El ítem del estado anterior es de la forma  $B \rightarrow \alpha_1 \cdot A\beta_1$ , y también es válido para  $\gamma$ . Por consiguiente, existe una derivación

$$S \xrightarrow{*_{rm}} \delta Bw \Rightarrow \delta\alpha_1 A\beta_1 w,$$

en la cual  $\gamma = \delta\alpha_1 \cdot$  Sea  $\beta_1 \xrightarrow{*} x$  para alguna cadena terminal  $x$ . Entonces la derivación

$$S \xrightarrow{*_{rm}} \delta Bw \xrightarrow{*_{rm}} \delta\alpha_1 A\beta_1 w \xrightarrow{*_{rm}} \delta\alpha_1 Axw \xrightarrow{*_{rm}} \delta\alpha_1 \beta xw$$

puede escribirse como

$$S \xrightarrow{*_{rm}} \delta\alpha_1 Axw \xrightarrow{*_{rm}} \delta\alpha_1 \beta xw.$$

Por tanto,  $A \rightarrow \cdot \beta$  es válido para  $\gamma$ , ya que  $\gamma = \delta\alpha_1$ .

*Si:* Supóngase que  $A \rightarrow \alpha \cdot \beta$  es válido para  $\gamma$ . Entonces

$$S \xrightarrow{*_{rm}} \gamma_1 Aw \xrightarrow{*_{rm}} \gamma_1 \alpha \beta w, \quad (10.3)$$

en donde  $\gamma_1 \alpha = \gamma$ . Si podemos demostrar que  $\delta(q_0, \gamma_1)$ , contiene a  $A \rightarrow \cdot \alpha \beta$ , entonces, mediante la regla (3), sabemos que  $\delta(q_0, \gamma)$  contiene a  $A \rightarrow \alpha \cdot \beta$ . Por tanto podemos demostrar por inducción en la longitud de la derivación (10.3) que  $\delta(q_0, \gamma_1)$  contiene a  $A \rightarrow \cdot \alpha \beta$ .

La base, un paso, se concluye de la regla (1). Para la inducción, considérese el paso en  $S \xrightarrow{*_{rm}} \gamma_1, Aw$  en el cual se introdujo la  $A$  mostrada de manera explícita. Esto es, escríbase  $S \xrightarrow{*_{rm}} \gamma_1, Aw$  'como'

$$S \xrightarrow{*_{rm}} \gamma_2 Bx \xrightarrow{*_{rm}} \gamma_2 \gamma_3 A\gamma_4 x \xrightarrow{*_{rm}} \gamma_2 \gamma_3 Ayx,$$

en donde  $\gamma_2 \gamma_3 = \gamma_1$ , y  $yx = w$ . Entonces, según la hipótesis inductiva aplicada a la derivación.

$$S \xrightarrow{*_{rm}} \gamma_2 Bx \Rightarrow \gamma_2 \gamma_3 A \gamma_4 x,$$

sabemos que  $B \rightarrow \cdot \gamma_3 A\gamma_4$  está en  $\delta(q_0, \gamma_2)$ . Mediante la regla (3),  $B \rightarrow \gamma_3 \cdot A\gamma_4$  se encuentra en  $\delta(q_0, \gamma_2 \gamma_3)$  y según la regla (2),  $A \rightarrow \cdot \alpha \beta$  está en  $\delta(q_0, \gamma_2 \gamma_3)$ . Puesto que  $\gamma_2 \gamma_3 = \gamma_1$ , hemos demostrado la hipótesis inductiva.  $\square$

### Definición de la gramática LR(0)

Ahora ya estamos preparados para definir una gramática  $LR(0)$ . Decimos que  $G$  es una *gramática  $LR(0)$*  si

1. su símbolo inicial no aparece en el lado derecho de cualquier producción, y

2. para cada prefijo viable  $\gamma$  de  $G$ , siempre que  $A \rightarrow \alpha \cdot$  sea un ítem válido completo para  $\gamma$ , entonces ningún otro ítem completo ni ningún ítem que posea una terminal a la derecha del punto es válido para  $\gamma$ .†

No existe prohibición alguna contra el hecho de que varios ítems incompletos sean válidos para  $\gamma$ , mientras que ningún ítem completo es válido.

El teorema 10.9 proporciona un método para calcular los conjuntos de ítems válidos para cualquier prefijo viable. Solamente conviértase el NFA cuyos estados son ítems a un DFA. En el DFA, la trayectoria con etiqueta  $\gamma$  que parte del estado inicial lleva al estado que es el conjunto de ítems válidos para  $\gamma$ . Por tanto constrúyase el DFA e inspecciónese cada estado para ver si se da una violación de la condición  $LR(0)$ .

**Ejemplo 10.9** En la Fig. 10.3 se muestra el DFA construido a partir del NFA de la Fig. 10.2, en el cual se eliminaron el estado muerto (conjunto vacío de ítems) y las transiciones al estado muerto. Estos estados, con excepción de  $I_0, I_1, I_3$ , y  $I_6$ , consisten en un solo ítem completo. Los estados con más de un ítem no poseen ítems completos, y es seguro que  $S'$ , el símbolo inicial, no aparece en el lado derecho de cualquier producción. De aquí que la gramática (10.2) sea  $LR(0)$ .

### 10.7 GRAMATICAS LR(0) Y LOS DPAs

Demostraremos ahora que cada gramática  $LR(0)$  genera un DCFL y que cada DCFL con la propiedad de prefijo tiene una gramática  $LR(0)$ . Ya que cada lenguaje con una gramática  $LR(0)$ , como se demostrará, tiene la propiedad de prefijo, tenemos una caracterización exacta de los DCFLs; a saber,  $L$  es un DCFL si y sólo si  $LS$  tiene una gramática  $LR(0)$ .

#### DPDAs a partir de gramáticas $LR(0)$

La manera en que construimos un DPDA a partir de una gramática  $LR(0)$  difiere de la forma en la cual construimos un PDA (no determinístico) a partir de un CFL arbitrario en el Teorema 5.3. En este último teorema determinamos una derivación izquierda de la palabra que se hallaba en la entrada del PDA, utilizamos la pila para contener al sufijo de una forma oracional izquierda comenzando con la variable que está más a la izquierda. En esta ocasión determinaremos una derivación a la extrema derecha, en sentido inverso, utilizando la pila para contener un prefijo viable de una forma oracional derecha, incluyendo a todas las variables de dicha forma oracional, permitiéndole al residuo de la forma aparecer en la entrada.

Con el fin de describir claramente este proceso, resulta de utilidad desarrollar una nueva notación para las IDs de un PDA. Representamos a la pila con su tope en el extremo derecho, más que a la izquierda. Para distinguir a la nueva notación de la vieja,

† Los únicos ítems que pueden ser válidos de manera simultánea a  $A \rightarrow \alpha \cdot$  son las producciones con una no terminal a la derecha del punto, y esto puede ocurrir sólo si  $\alpha = \epsilon$ ; de otra manera, se puede demostrar que ocurre otra violación de las condiciones  $LR(0)$ .

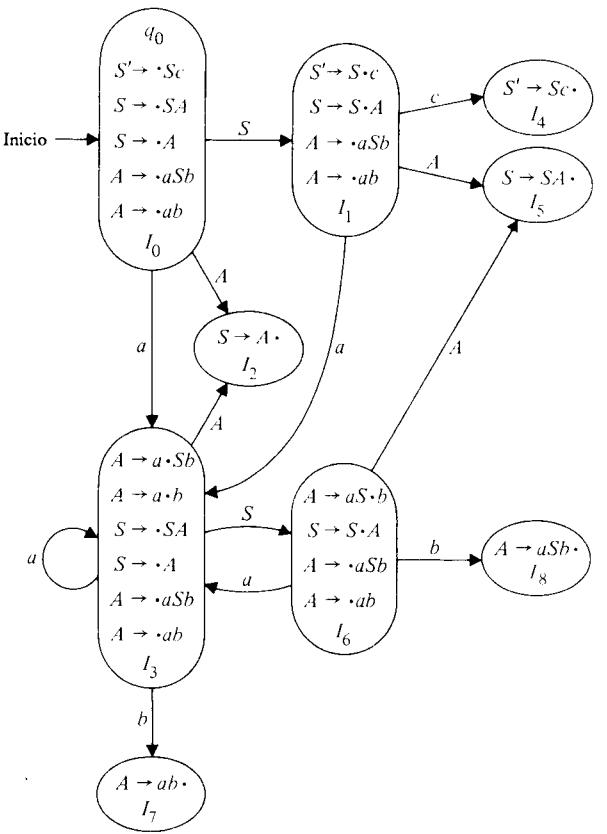


Fig. 10.3 DFA cuyos estados son los conjuntos de ítems válidos.

utilizamos paréntesis cuadrados en lugar de los redondos:  $[q, \alpha, w]$  en nuestro sinónimo de  $(q, w, \alpha^R)$ .

Para simular las derivaciones derechas de una gramática  $LR(0)$  no solo mantenemos un prefijo viable en la pila, sino que, sobre cada símbolo, mantenemos un estado del DFA que reconoce prefijos viables. Si un prefijo viable  $X_1 X_2 \cdots X_k$  se encuentra sobre la pila, entonces el contenido de la pila completa será  $s_0 X_1 s_1 \cdots X_k s_k$ , en donde  $s_i$  es  $\delta(q_0, X_1 \cdots X_i)$  y  $\delta$  es la función de transición del DFA. El estado de encima  $s_k$  proporciona los ítems válidos para  $X_1 X_2 \cdots X_k$ .

Si  $s_k$  contiene a  $A \rightarrow \alpha \cdot$ , entonces  $A \rightarrow \alpha \cdot$  es válido para  $X_1 \cdots X_k$ . Por consiguiente  $\alpha$  es un sufijo de  $X_1 \cdots X_k$ , digamos  $\alpha = X_{i+1} \cdots X_k$  (nótese que  $\alpha$  puede ser  $\epsilon$ , en cuyo caso  $i = k$ ). Aún más, existe alguna  $w$  tal que  $X_1 \cdots X_k w$  es una forma oracional derecha, y existe una derivación.

$$S \xrightarrow{rm} X_1 \cdots X_i Aw \xrightarrow{rm} X_1 \cdots X_k w.$$

Por consiguiente, para obtener la forma oracional derecha anterior a  $X_1 \cdots X_k w$  en una derivación derecha *reducimos*  $\alpha$  a  $A$ , sustituyendo  $X_{i+1} \cdots X_k$  en el tope de la pila por  $A$ . Esto es, mediante una secuencia de movimientos de exclusión (utilizando estados

diferentes de modo que el DPDA pueda recordar lo que está haciendo) seguida por un movimiento que coloca a  $A$  y al estado de cobertura correcto de la pila, nuestro DPDA accederá una sucesión de IDs

$$[q, s_0 X_1 \cdots s_{k-1} X_k s_k, w] \xrightarrow{*} [q, s_0 X_1 \cdots s_{i-1} X_i s_i A s, w], \quad (10.4)$$

en donde  $s = \delta(s_i, A)$ . Adviértase que si la gramática es  $LR(0)$ ,  $s_k$  contiene solamente a  $A \rightarrow \alpha \cdot$ , a menos que  $\alpha = \epsilon$ , en cuyo caso  $s_k$  debe contener algunos ítems incompletos. Sin embargo, según la definición del  $LR(0)$ , ninguno de estos ítems tiene una terminal colocada a la derecha del punto, o es completa. Por consiguiente, para cualquier  $y$  tal que  $X_1 \cdots X_k y$  es una forma oracional derecha,  $X_1 \cdots X_k Ay$  debe ser la forma oracional derecha anterior, de manera que la reducción de  $\alpha$  a  $A$  es correcta, sin importar la entrada actual.

Considérese, ahora, el caso en el cual  $s_k$  contiene sólo ítems incompletos. Entonces la forma oracional anterior a  $X_1 \cdots X_k w$  podría no estar formada mediante la reducción de un sufijo de  $X_1 \cdots X_k$  alguna variable, sino habría un ítem completo válido para  $X_1 \cdots X_k$ . Debe existir una terminación pivote a la derecha de  $X_k$  en  $X_1 \cdots X_k w$ , ya que  $X_1 \cdots X_k$  es un prefijo viable. Por tanto la única acción apropiada para el DPDA es correr el siguiente símbolo de entrada sobre la pila. Esto es,

$$[q, s_0 X_1 \cdots s_{k-1} X_k s_k, ay] \xrightarrow{*} [q, s_0 X_1 \cdots s_{k-1} X_k s_k at, y], \quad (10.5)$$

en donde  $t = \delta(s_k, a)$ . Si  $t$  no es el conjunto vacío de ítems,  $X_1 \cdots X_k a$  es un prefijo viable. Si  $t$  está vacío, demostraremos que no existe ninguna forma oracional derecha previa posible para  $X_1 \cdots X_k ay$ , de modo que la entrada original no se encuentra en el lenguaje de la gramática, y el DPDA “muere” en lugar de efectuar el movimiento (10.5). Resumimos las observaciones anteriores en el siguiente teorema:

**Teorema 10.10** Si  $L$  es  $L(G)$  para una gramática  $LR(0) G$ , entonces  $L$  es  $N(M)$  para un DPDA  $M$ .

**Demostración** A partir de  $G$ , constrúyase el DFA  $D$ , con función de transición  $\delta$ , que reconoce a los prefijos viables de  $G$ s. Hagamos que los símbolos de pila de  $M$  sean los símbolos de gramática de  $G$  y los estados de  $D$ .  $M$  tiene estado  $q$ , que es su estado inicial, junto con los estados adicionales que fueron utilizados para llevar a cabo las reducciones mediante sucesiones de movimientos como los (10.4). Suponemos que el lector puede especificar el conjunto de estados para cada reducción y las transiciones  $\in$  que se necesitan para llevar a cabo la reducción. También dejamos al lector la especificación de la función de transición de  $M$  que se necesita para instrumentar los movimientos indicados por (10.4) y (10.5).

Hemos indicado con anterioridad por que, si  $G$  es  $LR(0)$ , las reducciones son el único camino posible para obtener la forma oracional derecha previa cuando el estado del DFA en la cima de la pila de  $M$ s contiene un ítem completo. Pugnamos por que cuando  $M$  comienza con  $w$  en  $L(G)$  en su entrada y solamente  $s_0$  en su pila, construya una derivación extrema derecha para  $w$  en orden inverso. El único punto que todavía requiere demostración corresponde a que cuando se requiere un corrimiento, como en (10.5), debido a que el estado DFA en el tope que se encuentra en la pila de  $M$ s tiene

solamente ítems incompletos, entonces podría no existir un pivote entre los símbolos de gramática  $X_1 \cdots X_k$  que se encontraban en la pila en ese momento. Si existiera tal pivote, entonces algún estado DFA sobre la pila, por debajo de la cima, tendría un ítem completo.

Supóngase que existe un estado tal que contiene a  $A \rightarrow \alpha \bullet$ . Nótese que cada estado, cuando se coloca por primera vez en la pila, ya sea mediante (10.4) o (10.5), se encuentra en el tope de la pila. Por lo tanto, inmediatamente requerirá la reducción de  $\alpha$  a  $A$ . Si  $\alpha \neq \epsilon$ , entonces  $\{A \rightarrow \alpha \bullet\}$  se retira de la pila y no puede introducirse en ella. Si  $\alpha = \epsilon$ , entonces la reducción de  $\epsilon$  a  $A$  se efectúa mediante (10.4), ocasionando que  $A$  sea colocada en la pila por encima de  $X_1 \cdots X_k$ . En este caso, siempre habrá una variable por arriba de  $X_k$  en la pila siempre que  $X_1 \cdots X_k$  ocupe las posiciones del fondo de la pila. Pero  $A \rightarrow \epsilon$  en la posición  $k$  podría no ser el pivote de ninguna forma oracional derecha  $X_1 \cdots X_k \beta$ , en donde  $\beta$  contiene una variable.

Un último punto que se refiere a la aceptación por parte de  $G$ . Si el estado del tope de la pila es  $\{S \rightarrow \alpha \bullet\}$ , en el que  $S$  es el símbolo inicial de  $G$ , entonces  $G$  expulsa su pila, aceptando. En este caso hemos completado la inversa de una derivación derecha de la entrada original. Adviértase que como  $S$  no aparece en la parte derecha de ninguna producción, resulta imposible que exista un ítem de la forma  $A \rightarrow S \bullet \alpha$  válido para el prefijo viable  $S$ . Por consiguiente no habrá nunca necesidad de correr símbolos de entrada adicionales cuando  $S$  aparece solo en la pila. Dicho de otra manera,  $L(G)$  tiene siempre la propiedad de prefijo si  $G$  es  $LR(0)$ .

Así pues, hemos demostrado que si  $w$  se encuentra en  $L(G)$ ,  $M$  halla una derivación derecha de  $w$ , reduce  $w$  a  $S$  y acepta. de manera inversa, si  $M$  acepta a  $w$ , la secuencia de formas oracionales derechas representadas por las IDs de  $M$  proporciona una derivación de  $w$  a partir de  $S$ . Por consiguiente  $N(M) = L(G)$ .  $\square$

**Corolario** Cada gramática  $LR(0)$  es no ambigua.

**Demostración** El argumento anterior muestra que la derivación derecha de  $w$  es única.  $\square$

**Ejemplo 10.10** Considérese el DFA de la Fig. 10.3. Sean  $0, 1, \dots, 8$  los nombres de los estados correspondientes a los conjuntos de ítems  $I_0, I_1, \dots, I_8$ , respectivamente. Hagamos que la entrada sea  $aababb$ . El DPDA  $M$  construido de la misma forma que en el Teorema 10.10 realiza la sucesión de movimiento que se lista en la Fig. 10.4.

Por ejemplo, en la línea (1), el estado 0 se encuentra en el tope de la pila. No existe ningún ítem completo en el conjunto  $I_0$ , así que nos recorremos. El primer símbolo de entrada es  $a$ , y existe una transición de  $I_0$  a  $I_3$  con etiqueta  $a$ . Por consiguiente en la línea (2) la pila es  $0a3$ . En la línea (9), 5 se encuentra en el tope de la pila.  $I_5$  consiste en el ítem completo  $S \rightarrow SA$ . Sacamos  $SA$  de la pila, dejando  $0a3$ . Entonces colocamos a  $S$  en la pila. Existe una transición de  $I_3$  a  $I_6$  con etiqueta  $S$ , de modo que colocamos 6 sobre  $S$ , y producimos la pila  $0a3S6$  en la línea (10).

### Gramáticas $LR(0)$ a partir de DPDA

Comenzaremos ahora nuestro estudio del resultado recíproco: si  $L$  es  $N(M)$  para un DPDA  $M$  entonces  $L$  tiene una gramática  $LR(0)$ . De hecho, la gramática del Teorema

	Pila	Entrada restante	Comentarios
1)	0	$aababb$	ID inicial
2)	$0a3$	$ababb$	Comentarios
3)	$0a3a3$	$babb$	Comentarios
4)	$0a3a3b7$	$abbc$	Comentarios
5)	$0a3A2$	$abbc$	Se reduce mediante $A \rightarrow ab$
6)	$0a3S6$	$abbc$	Se reduce mediante $S \rightarrow A$
7)	$0a3S6a3$	$bbc$	Comentarios
8)	$0a3S6a3b7$	$bc$	Comentarios
9)	$0a3S6A5$	$bc$	Se reduce mediante $A \rightarrow ab$
10)	$0a3S6$	$bc$	Se reduce mediante $S \rightarrow SA$
11)	$0a3S6b8$	$c$	Comentarios
12)	$0A2$	$c$	Se reduce mediante $A \rightarrow aSb$
13)	$0S1$	$c$	Se reduce mediante $S \rightarrow A$
14)	$0S1c4$	—	Comentarios
15)	—	—	Acepta

Fig. 10.4 Sucesión de movimientos del DPDA  $M$ .

5.4 en  $LR(0)$ , siempre que  $M$  sea determinístico, pero resulta más sencillo demostrar que una modificación de dicha gramática es  $LR(0)$ . El cambio que efectuamos consiste en poner al principio del lado derecho de cada producción un símbolo que diga cuál movimiento PDA da lugar a dicha producción.

De manera formal, sea  $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$  un DPDA. Definimos la gramática  $G_M = (V, \Sigma, P, S)$  tal que  $L(G_M) = N(M)$ .  $V$  consiste en el símbolo  $S$ , los símbolos  $[qXp]$  para  $q$  y  $p$  en  $Q$  y  $X$  en  $\Gamma$ , y en los símbolos  $A_{qay}$  para  $q$  en  $Q$ ,  $a$  en  $\Sigma \cup \{\epsilon\}$  y  $Y$  en  $\Gamma$ .  $S$  y los  $[qXp]$  juegan el mismo papel que en el Teorema 5.3. El símbolo  $A_{qay}$  indica que la producción se obtiene del movimiento de  $M$  en  $\delta(q, a, Y)$ . Las producciones de  $G_M$  son como sigue (se han eliminado los símbolos y las producciones inútiles):

- 1)  $S \rightarrow [q_0 Z_0 p]$  para toda  $p$  en  $Q$ .
- 2) Si  $\delta(q, a, Y) = (p, \epsilon)$ , entonces existe una producción  $[qYp] \rightarrow A_{qay}$ .
- 3) Si  $\delta(q, a, Y) = (p_1, X_1 X_2 \dots X_k)$  para  $k \geq 1$ , entonces para cada secuencia de estados  $p_2, p_3, \dots, p_{k+1}$  existe una producción

$$[qYp_{k+1}] \rightarrow A_{qay}[p_1 X_1 p_2] \cdots [p_k X_k p_{k+1}].$$

- 4) Para toda  $q, a$  y  $Y$ ,  $A_{qay} \rightarrow a$ .

Considérese una derivación derecha en  $G_M$ . Inicial como  $S \Rightarrow [q_0 Z_0 p]$  para algún estado  $p$ . Supóngase, para cuestiones de argumento, que  $\delta(q_0, a, Z_0) = (r, XYZ)$ . Entonces las únicas producciones para  $[q_0 Z_0 p]$  que derivan cadenas que comienzan con  $a$  ( $a$  puede ser  $\epsilon$ ) tienen lado derecho  $A_{qoaZ_0}[rXs][sYt][tZp]$  para algunos estados  $s$  y  $t$ . Supóngase que la derivación derecha en algún momento deriva alguna cadena  $w$  a partir de  $[tZp]$ . Entonces, si  $\delta(s, b, Y) = (u, VW)$ , podríamos continuar la derivación derecha como

$$S \xrightarrow{*} A_{qoaZ_0}[rXs][sYt]w \xrightarrow{rm} A_{qoaZ_0}[rXs]A_{sby}[uVv][vWt]w. \quad (10.6)$$

Considérese ahora los movimientos hechos por  $M$  antes de leer la entrada  $w$ . La entrada correspondiente a la derivación (10.6) es de la forma  $ax_1bx_2x_3w$ , en donde  $[rXs] \xrightarrow{*} x_1$ ,  $[uVv] \xrightarrow{*} x_2$ , y  $[vWt] \xrightarrow{*} x_3$ . La secuencia correspondiente de movimientos es de la forma†

$$\begin{aligned}
 (q_0, ax_1bx_2x_3w, Z_0) &\vdash (r, x_1bx_2x_3w, XYZ) \\
 &\vdash^* (s, bx_2x_3w, YZ) \\
 &\vdash (u, x_2x_3w, VWZ) \\
 &\vdash (v, x_3w, WZ) \\
 &\vdash (t, w, Z). \tag{10.7}
 \end{aligned}$$

Si comparamos (10.6) y (10.7) nos damos cuenta de que los símbolos de pila ( $Z$  en particular) que permanecen en la pila al final de (10.7) son los símbolos que no aparecen (con dos estados unidos en una variable entre paréntesis cuadrados) en el prefijo viable más largo de (10.6). Los símbolos de pila expulsados de la pila de (10.7), a saber,  $X$ ,  $V$  y  $W$ , son los símbolos que aparecen en el prefijo viable de (10.6). Esta situación tiene sentido, ya que los símbolos que están en el extremo izquierdo de una forma oracional derivan un prefijo de una oración, y dicho prefijo es leído en primer lugar por el PDA.

En general, dado cualquier prefijo viable  $\alpha$  de  $G_M$ , podemos encontrar una ID correspondiente  $I$  de  $M$  en la cual la pila contiene a todos, y únicamente a ellos, los símbolos de pila que fueron introducidos en una derivación derecha de alguna  $\alpha w$  y más tarde sustituidos por una cadena de terminales. Más aún  $I$  se obtiene cuando  $M$  ha leído cualquier cadena derivada de  $\alpha$ . En el caso en que  $M$  sea determinístico, podemos argumentar que las derivaciones de las formas oracionales derechas con prefijo  $\alpha$  tienen una forma específica y traducen tales limitaciones a las derivaciones en restricciones al conjunto de ítems de  $\alpha$ .

**Lema 10.6** Si  $M$  es un DPDA y  $G_M$  es la gramática construida a partir de  $M$ , como se hizo más arriba, entonces, siempre que  $[qXp] \xrightarrow{*} w$ , existe un cálculo único  $(q, w, X) \vdash^* (p, \epsilon, \epsilon)$ . Es más, la secuencia de movimientos hecha por  $M$  corresponde al inverso de la secuencia en la cual la  $A_{\alpha Y}$ s con subíndices son sustituidas por  $a$ , en donde  $A_{\alpha Y}$  se considera como “correspondientes” a un movimiento en el que el estado es  $s$ ,  $Y$  se encuentra en el tope de la pila, y se utiliza la entrada  $a$ .

**Demostración** La existencia de un cálculo tal fue demostrada en el Teorema 5.3. Su unicidad se concluyó del hecho de que  $M$  es determinístico. Para mostrar la correspondencia entre los movimientos de  $M$  y la inversa de la secuencia de las expansiones de las  $A$ s con subíndices, efectuamos una inducción sencilla sobre la longitud de una derivación. La parte clave del paso inductivo es cuando la primera expansión se lleva a cabo según la regla (3):

$$[qXp] \Rightarrow A_{\alpha X}[p_1 X_1 p_2][p_2 X_2 p_3] \cdots [p_k X_k p].$$

Entonces la  $A_{\alpha X}$ , mostrada explícitamente, se expandirá después de que sean expandidas todas las  $A$ s con subíndice derivadas de las otras variables.

Como el primer movimiento de  $M$  es,

$$(q, w, X) \vdash (p_1, w', X_1 X_2 \cdots X_k),$$

en donde  $w = aw'$ , corresponde a  $A_{\alpha X}$ , tenemos demostrada parte de la inducción. Lo que resta de la inducción se concluye de observar que en los movimientos de  $M$ ,  $X_1, X_2, \dots, X_k$  se eliminan de la pila en orden, mediante el uso de las entradas  $w_1, w_2, \dots, w_k$ , en donde  $w_1 w_2 \cdots w_k = w'$ , mientras que en la derivación derecha de  $w$  a partir de  $[qXp]$ , la derivación de  $w_1$ , a partir de  $[p_1 X_1 p_2]$  sigue a la derivación de  $w_2$  a partir de  $[p_2 X_2 p_3]$ , y así sucesivamente. Puesto que todas las derivaciones son más cortas que  $[qXp] \xrightarrow{*} w$ , podemos utilizar la hipótesis inductiva para completar la demostración. □

Ahora, para cada variable  $[qXp]$  de  $G_M$ , fijemos una cadena particular  $w_{qXp}$  derivada de  $[qXp]$ .†. Sea  $h$ , el homomorfismo de las variables de  $G_M$  a  $\Sigma^*$  definido por

$$h(A_{\alpha Y}) = a, \quad h([qXp]) = w_{qXp}.$$

Hagamos que  $N(A_{\alpha Y}) = 1$  y  $N([qXp])$  sean el número de movimientos en el cálculo correspondiente a  $[qXp] \xrightarrow{*} w_{qXp}$ . Extiéndase  $N$  a  $V^*$  mediante  $N(B_1 B_2 \dots B_k) = \sum_{i=1}^k N(B_i)$ . Finalmente, representamos un movimiento  $\delta(q, a, Y)$  de  $M$  mediante el triplete  $(qaY)$ . Sean  $m$  el homomorfismo de  $V^*$  a los movimientos definidos por

1.  $m(A_{\alpha Y}) = (qaY);$
2.  $m([qXp])$  es el inverso de la sucesión de subíndices de las  $A$ s expandidas en la derivación de  $w_{qXp}$  a partir de  $[qXp]$ . Según el Lema 10.6,  $m([qXp])$  es también la secuencia de movimientos  $(q, w_{qXp}, X) \vdash^* (p, \epsilon, \epsilon)$ .

Ahora podemos completar nuestra caracterización de las gramáticas  $LR(0)$ .

**Lema 10.7** Sea  $\gamma$  un prefijo viable de  $G_M$ . ( Nótese, mediante la construcción de  $G_M$ ,  $\gamma$  se encuentra en  $V^*$ ). Entonces  $(q_0, h(\gamma), Z_0) \xrightarrow{N(\gamma)} (p, \epsilon, \beta)$  para algunas  $p$  y  $\beta$ , a través de la secuencia de movimientos  $m(\gamma)$ .

**Demostración** Puesto que  $\gamma$  es un prefijo viable, existe alguna  $\gamma$  en  $\Sigma^*$  tal que  $\gamma\gamma$  es una forma oracional derecha. Entonces, para algún estado  $r$ ,  $[q_0 Z_0 r] \xrightarrow{N(\gamma)} h(\gamma) \gamma$ . Según el Lema 10.6, las últimas  $N(\gamma)$  expansiones de  $A$ s en dicha derivación se llevan a cabo después de que la forma oracional derecha  $\gamma\gamma$  se ha alcanzado. También según el Lema 10.6 existe una sucesión única de movimientos  $(q_0, h(\gamma)\gamma, Z_0) \vdash^* (r, \epsilon, \epsilon)$  y los primeros  $N(\gamma)$  de éstos deben ser  $m(\gamma)$ . □

Ahora estamos listos para demostrar que  $G_M$  es  $LR(0)$ . Ya que, obviamente, el símbolo inicial no aparece en ningún lado derecho, es suficiente demostrar que cada conjunto de ítems con un ítem completo  $B \rightarrow \beta \bullet$  no contiene a ningún otro ítem completo y a ningún ítem de la forma  $A_{\alpha Y} \rightarrow \bullet a$  para  $a$  en  $\Sigma$ . Demostraremos estos hechos en dos lemas.

† Suponemos que  $G_M$  no tiene símbolos inútiles, así que  $w_{qXp}$  existe.

† Nótese que hemos regresado a nuestra notación original para los IDs.

**Lema 10.8** Si  $I$  es un conjunto de ítems de  $G_M$  y  $B \rightarrow \beta \cdot$  se encuentra en  $I$ , entonces no existe el ítem  $A_{qaY} \rightarrow \cdot a$  en  $I$ .

*Demostración* Sea  $I$  el conjunto de ítems para el prefijo viable  $\gamma$ .

**CASO 1** Si  $B \rightarrow \beta$  es una producción formada a partir de la regla (1), entonces  $\gamma = \beta$  y  $\gamma$  es una variable simple  $[q_0 Z_0 p]$ , ya que  $S$  no aparece en ningún lado derecho. Si  $A_{qaY} \rightarrow \cdot a$  es válido para  $\gamma$ , entonces existe una derivación  $S \xrightarrow{*} \gamma A_{qaY} \gamma \xrightarrow{*} \gamma aY$ . Sin embargo, ninguna forma oracional derecha comienza con una variable  $[q_0 Z_0 p]$ , a menos que sea el primer paso de una derivación; toda las formas oracionales derechas siguientes comienzan con una  $A$  con subíndices, hasta la última, que comienza con una terminal. Por consiguiente  $\gamma$  no podría estar seguido por  $A_{qaY}$  en una forma oracional derecha.

**CASO 2** Si  $B \rightarrow \beta$  es introducida mediante las reglas (2) o (3), entonces podemos, de nuevo, argumentar que  $\gamma' \beta A_{qaY}$  es un prefijo viable, en el cual  $\gamma = \gamma' \beta$ . Sin embargo, en cualquier derivación derecha, cuando  $B \rightarrow \beta$  se aplica, el último símbolo de  $\beta$  es expandido, de manera inmediata, por las reglas (2), (3) o (4), de modo que  $\beta$  no podría aparecer sin cambios en una forma oracional derecha, seguida de  $A_{qaY}$ .

**CASO 3** Si  $B \rightarrow \beta$  es  $A_{pbZ} \rightarrow b$  introducida por la regla (4) y  $A_{qaY} \rightarrow \cdot a$  es válida para  $\gamma$ , entonces  $b$  debe ser  $\epsilon$ , de otra forma  $\gamma A_{qaY}$ , que es un prefijo viable, tiene una terminal en ella. Como  $A_{peZ} \rightarrow \cdot$  es válido para  $\gamma$ , se concluye que  $\gamma A_{peZ}$  es un prefijo viable. Por consiguiente, según el Lema 10.7 aplicado a  $\gamma A_{qaY}$  y  $\gamma A_{peZ}$ , los primeros  $N(\gamma) + 1$  movimientos hechos por  $M$  cuando se le da la entrada  $h(\gamma)a$  son ambas  $m(\gamma)$  ( $p \in Z$ ) y  $m(\gamma)$  ( $qaY$ ), contradiciendo el determinismo de  $M$ . (Adviértase que en la primera de estas secuencias, una  $a$  no se consume.)  $\square$

**Lema 10.9** Si  $I$  es un conjunto de ítems de  $G_M$ , y  $B \rightarrow \beta \cdot$  se encuentra en  $I$ , entonces no existe otro ítem  $C \rightarrow \alpha \cdot$  en  $I$ .

*Demostración* De nuevo hagamos que  $\gamma$  sea un prefijo viable con el conjunto de ítems válidos  $I$ .

**CASO 1** Ninguna,  $B \rightarrow \beta$  o  $C \rightarrow \alpha$  es una producción introducida por la regla (4). Entonces la forma de las producciones de los tipos (2) y (3), y el hecho de que las producciones del tipo (1) se aplican solamente en el primer paso, nos dice que, como  $\alpha$  y  $\beta$  son las dos sufijos de  $\gamma$ , debemos tener  $\beta = \alpha$ . Si estas producciones son del tipo (1),  $B = C = S$ , de modo que los dos ítems son en realidad el mismo. Si las producciones son del tipo (2) o (3), es fácil verificar que  $B = C$ . Por ejemplo, si  $\alpha = \beta = A_{qaY}$ , entonces las producciones son del tipo (2), y  $B$  y  $C$  son cada una  $[qYp]$  para alguna  $p$ . Pero la regla (2) requiere que  $\delta(q, a, Y) = (p, \epsilon)$ , así que el determinismo de  $M$  nos asegura que  $p$  es única.

**CASO 2**  $B \rightarrow \beta$  y  $C \rightarrow \alpha$  son producciones del tipo (4). Entonces  $\gamma B$  y  $\gamma C$  son prefijos viables, y el Lema 10.7 proporciona una contradicción al determinismo de  $M$ . Esto es, si  $\beta = \alpha = \epsilon$ , entonces los primeros  $N(\gamma) + 1$  movimientos de  $M$  sobre la entrada  $h(\gamma)$  deben ser  $m(\gamma B)$  y también  $m(\gamma C)$ . Si  $\beta = a \neq \epsilon$ , y  $\alpha = b \neq \epsilon$ , entonces  $a = b$ , y los

primeros  $N(\gamma) + 1$  movimientos de  $M$  sobre la entrada  $h(\gamma) a$  deben ser  $m(\gamma B)$  y  $m(\gamma C)$ . Si  $\beta = a \neq \epsilon$  y  $\alpha = \epsilon$ , entonces los primeros  $N(\gamma) + 1$  movimientos de  $M$  sobre la entrada  $h(\gamma) a$  nos conduce a una contradicción parecida.

**CASO 3**  $B \rightarrow \beta$  se produce de la regla (1), (2) o (3) y  $C \rightarrow \alpha$  se tiene de la regla (4), o viceversa. Entonces  $\gamma C$  es una forma oracional derecha, y  $\gamma$  termina en  $\beta$ . Podemos reglamentar esta posibilidad como en los casos (1) y (2) del Lema 10.8.  $\square$

**Teorema 10.11** Si  $M$  es un DPDA, entonces  $G_M$  es una gramática  $LR(0)$ .

*Demostración* Es inmediata a partir de los Lemas 10.8 y 10.9.  $\square$

Podemos completar ahora nuestra caracterización de las gramáticas  $LR(0)$ .

**Teorema 10.12** Un lenguaje  $L$  posee una gramática  $LR(0)$  si y sólo si  $L$  es un DCFL con la propiedad de prefijo.

*Demostración*

Si: Supóngase que  $L$  es un DCFL con la propiedad de prefijo. Entonces  $L$  es  $L(M')$  para un DPDA  $M'$ . Podemos hacer que  $M'$  acepte a  $L$  mediante el agotamiento de su pila, colocando un señalador de fondo de pila en  $M'$  y ocasionalmente que  $M'$  accese un nuevo estado que borre la pila siempre que accese un estado final. Como  $L$  tiene la propiedad de prefijo, no cambiamos el lenguaje aceptado, y  $L$  es aceptado mediante el agotamiento de la pila, por parte del nuevo DPDA,  $M$ . Por consiguiente,  $L = L(G_M)$ , y la conclusión deseada se sigue del Teorema 10.11.

Sólo si: El Teorema 10.10 establece que  $L$  es  $N(M)$  para un DPDA,  $M$ '. Podemos hacer uso de la construcción del Teorema 5.2 para demostrar que  $L$  es  $L(M')$  para un DPDA,  $M'$ . El hecho de que  $L$  tiene la propiedad de prefijo se concluye del hecho de que un DPDA “muere” cuando agota su pila.  $\square$

**Corolario**  $L\$$  tiene gramática  $LR(0)$  si y sólo si  $L$  es un DCFL, en donde  $\$$  no es un símbolo del alfabeto de  $L$ .

*Demostración* Seguramente,  $L\$$  tiene la propiedad de prefijo. Si  $L\$$  es un DCFL, entonces, según el Teorema 10.2,  $L = L\$/\$$  es un DCFL. Inversamente, si  $L$  es un DCFL es sencillo construir un DPDA para  $L\$$ .  $\square$

## 10.8 GRAMATICAS LR(k)

Resulta interesante hacer notar que si agregamos un símbolo de “adelanto” mediante la determinación del conjunto de terminales seguidas, en el cual pueda ser posible llevar a cabo la reducción mediante  $A \rightarrow \alpha$ , entonces podemos utilizar DPDA para reconocer los lenguajes de una clase más amplia de gramáticas. Estas gramáticas se conocen como gramáticas  $LR(1)$  debido al símbolo de adelanto. Se sabe que todos los CFLs determinísticos, y solamente éstos, tienen gramáticas  $LR(1)$ . Esta clase de gramáticas tiene una gran importancia en el diseño de compiladores, ya que son lo

suficientemente amplias para incluir la sintaxis de casi todos los lenguajes de programación, si bien son, por otra parte, lo suficientemente restrictivas como para tener analizadores de gramática eficientes que sean, esencialmente, DPDAs.

Resulta que al agregar más de un símbolo de adelanto para guiar la alternativa de reducciones no añada a la clase de los lenguajes definibles, aunque para cualquier  $k$  existen gramáticas, conocidas como gramáticas  $LR(k)$ , que pueden ser analizadas con  $k$  símbolos de adelanto, pero sin que tengan  $k - 1$  símbolos de adelanto.

Demos, de manera breve, la definición de las gramáticas  $LR(1)$  y un ejemplo de éstas, sin demostrar ninguna de las discusiones anteriores. La extensión clave de las gramáticas  $LR(0)$  consiste en que un ítem  $LR(1)$  es un ítem  $LR(0)$  seguido de un *conjunto de adelanto* formado por las terminales y/o el símbolo especial  $\$$ , que sirve para representar el extremo derecho de una cadena. Por consiguiente, la forma general de un ítem  $LR(1)$  es

$$A \rightarrow \alpha \cdot \beta, \{a_1, a_2, \dots, a_n\}.$$

Decimos que el ítem  $LR(1)$   $A \rightarrow \alpha \cdot \beta, \{a\}$  es *válido* para el prefijo viable  $\gamma$  si existe una derivación extrema derecha  $S \xrightarrow{*} \delta A y \xrightarrow{*} \delta \alpha \beta y$ , en la que  $\delta \alpha = \gamma$ , y

- i)  $a$  es el primer símbolo de  $y$ , o
- ii)  $y = \epsilon$  y  $a$  es  $\$$ .

También,  $A \rightarrow \alpha \cdot \beta, \{a_1, a_2, \dots, a_n\}$  es válido para  $\gamma$  si, para cada  $i$ ,  $A \rightarrow \alpha \cdot \beta, \{a_i\}$  es válido para  $\gamma$ .

Al igual que los ítems  $LR(0)$ , el conjunto de ítems  $LR(1)$  forma los estados de prefijos viables que reconocen al NFA, y podemos calcular el conjunto de ítems válidos para cada prefijo viable mediante la conversión de este NFA a un DFA. Las transiciones de dicho NFA se definen de la manera siguiente.

1. Existe una transición en  $X$  de  $A \rightarrow \alpha \cdot X \beta, \{a_1, a_2, \dots, a_n\}$  a  $A \rightarrow \alpha X \cdot \beta, \{a_1, a_2, \dots, a_n\}$ .
2. Existe una transición en  $\epsilon$  de  $A \rightarrow \alpha \cdot X \beta, \{a_1, a_2, \dots, a_n\}$  a  $B \rightarrow \cdot \gamma$ ,  $T$  si  $B \rightarrow \gamma$  es una producción y  $T$  el conjunto de terminales y/o  $\$$  tal que  $b$  se encuentra en  $T$  si y sólo si sucede uno de los casos siguientes
  - i)  $\beta$  deriva una cadena terminal que comienza con  $b$ , o
  - ii)  $\beta \xrightarrow{*} \epsilon$ , y  $b$  es  $a_i$  para alguna  $1 \leq i \leq n$ .
3. Existe un estado inicial  $q_0$  con transiciones en  $\epsilon$  a  $S \rightarrow \cdot \alpha, \{\$\}$  para cada producción  $S \rightarrow \alpha$ .

#### Ejemplo 10.11 Considérese la gramática

$$S \rightarrow A \quad A \rightarrow BA \mid \epsilon \quad B \rightarrow aB \mid b \quad (10.8)$$

que genera al conjunto regular,  $(a^* b)^*$ . El NFA para la gramática (10.8) se muestra en la Fig 10.5, y en la Fig. 10.6 se presenta el correspondiente DFA. El NFA de la Fig. 10.5 no es del tipo usual, puesto que presenta el hecho de que no tiene dos ítems que difieran solamente en los conjuntos de adelanto. En general, podemos ver dos ítems que poseen la misma producción con punto.

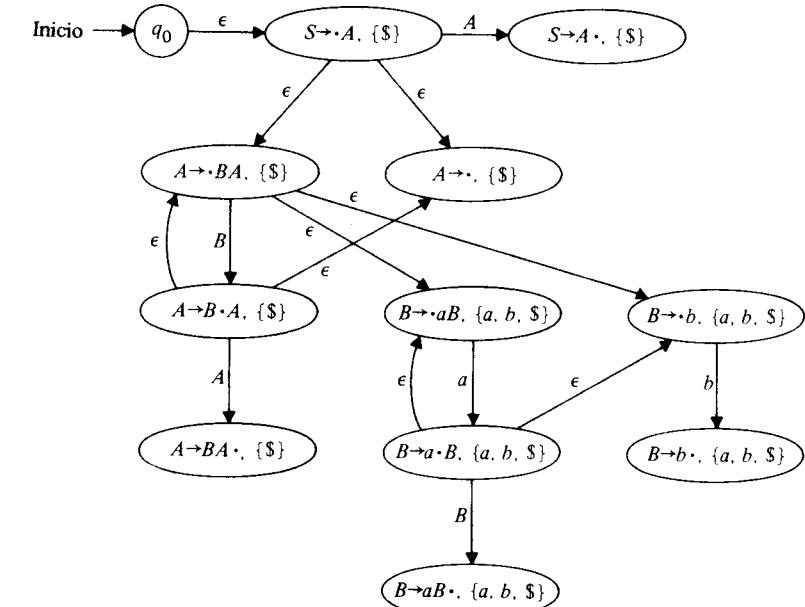


Fig. 10.5 NFA para los ítems  $LR(1)$ .

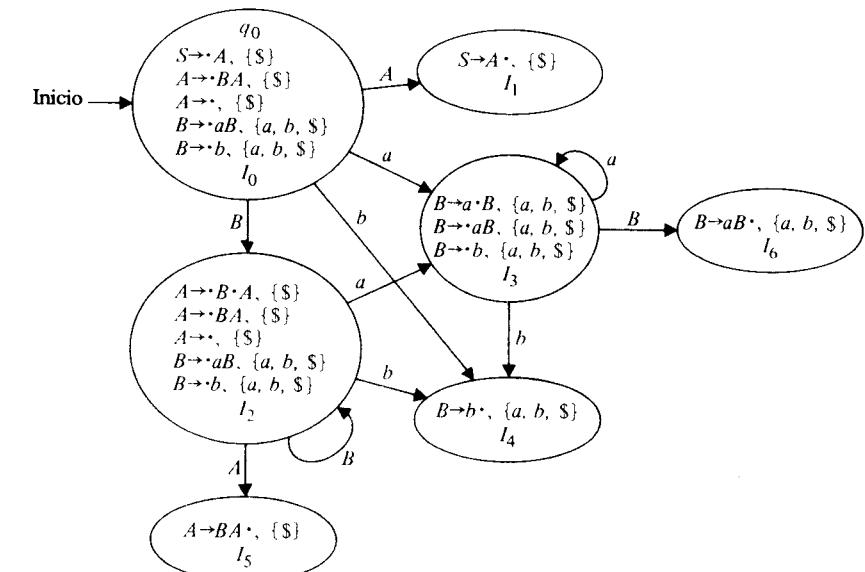


Fig. 10.6 DFA para los ítems  $LR(1)$ .

Para ver de qué manera se construye la Fig. 10.5, considérese el ítem  $S \rightarrow \cdot A, \$$ . Tiene transiciones  $\epsilon$  a ítems de la forma  $A \rightarrow \cdot AB, T \rightarrow \cdot A, T$ , pero ¿qué deberá ser  $T$ ? En la regla (2) dada más arriba,  $\beta$  es  $\epsilon$ , de modo que (2i) no produce símbolos para  $T$ . La regla (2ii) nos dice que  $\$$  se encuentra en  $T$ , así que  $T = \{\$\}$ . Considerese ahora el ítem  $A \rightarrow \cdot BA, \$$ . Existen transiciones  $\epsilon$  de  $B \rightarrow \cdot aB, U$  y  $B \rightarrow \cdot b, U$  para alguna  $U$ . Aquí,  $\beta = A$ . Resulta sencillo verificar que  $A$  deriva cadenas que comienzan con  $a$  y  $b$ , de modo que  $a$  y  $b$  se encuentran en  $U$ .  $A$  deriva también  $\epsilon$ , así que  $\$$  también está en  $U$ , ya que es el conjunto de adelanto de  $A \rightarrow \cdot BA, \$$ . Por consiguiente  $U = \{a, b, \$\}$ .

Se dice que una gramática es  $LR(1)$  si

1. el símbolo inicial no aparece en ningún lado derecho y
2. siempre que el conjunto de ítems  $I$  válido para algún prefijo viable incluya algún ítem completo  $A \rightarrow \alpha \cdot, \{\alpha_1, \alpha_2, \dots, \alpha_n\}$ , entonces
  - i) ninguna  $\alpha_i$  aparece inmediatamente a la derecha del punto en cualquier ítem de  $I$  y
  - ii) si  $B \rightarrow \beta \cdot, \{\beta_1, \beta_2, \dots, \beta_k\}$  es otro ítem completo de  $I$ , entonces  $\alpha_i \neq \beta_j$  para cualquier  $1 \leq i \leq n$  y  $1 \leq j \leq k$

**Ejemplo 10.12** Considerese la Fig. 10.6. El conjunto de ítems  $I_1, I_4, I_5, I_6$  consiste en solamente un ítem y, así, satisface (2). El conjunto  $I_0$  tiene un ítem completo,  $A \rightarrow \cdot, \$$ . Pero  $\$$  no aparece a la derecha de un punto en cualquier ítem de  $I_0$ . Un señalamiento parecido se aplica a  $I_2$ , e  $I_3$  no tiene ítems completos. Por consiguiente la gramática (10.8) es  $LR(1)$ . Nótese que esta gramática no es  $LR(0)$ ; su lenguaje no posee la propiedad de prefijo.

El autómata que acepta a un lenguaje  $LR(1)$  es parecido a un DPDA, excepto que al primero se le permite utilizar el siguiente símbolo de entrada al hacer sus decisiones, aún si efectúa un movimiento que no consuma su entrada. Podemos simular un autómata así mediante un DPDA ordinario si colocamos  $\$$  al final de la entrada. Entonces el DPDA puede mantener el símbolo siguiente o a  $\$$  en su estado para indicar el símbolo barrido. La pila de nuestro autómata es como la pila del DPDA que reconoce a la gramática  $LR(0)$ : posee símbolos de gramática que se alternan y conjuntos de ítems. Las reglas mediante las cuales decide si reducir o correr un símbolo de entrada en la pila son:

1. Si el conjunto de ítems en el tope tiene al ítem completo  $A \rightarrow \alpha \cdot, \{\alpha_1, \alpha_2, \dots, \alpha_n\}$  en donde  $A \neq S$ , redúzcase mediante  $A \rightarrow \alpha$  si el símbolo de entrada actual está en  $\{\alpha_1, \alpha_2, \dots, \alpha_n\}$ .
2. Si el conjunto de ítems en el tope tiene un ítem  $S \rightarrow \alpha \cdot, \$$ , entonces redúzcase mediante  $S \rightarrow \alpha$  y acéptese si el símbolo actual es  $\$$ , esto es, se ha alcanzado el extremo de la entrada.
3. Si el conjunto de ítems en el tope tiene un ítem  $A \rightarrow \alpha \cdot aB, T$ , y  $a$  es el símbolo de entrada actual, entonces córrase.

Nótese que la definición de la gramática  $LR(1)$  nos garantiza que cuando más una de las reglas anteriores, se aplicará a cualquier símbolo de entrada o  $\$$  particular. Tenemos la costumbre de resumir estas decisiones mediante una tabla cuyos renglones corresponden al conjunto de ítems y cuyas columnas son las terminales y  $\$$ .

**Ejemplo 10.13** En la Fig. 10.7 se muestra la tabla para la gramática 10.8, construida a partir de la Fig. 10.6. Las entradas que están en blanco representan un error; la entrada no se encuentra en el lenguaje. En la Fig. 10.8 se muestra la sucesión de acciones que lleva acabo el analizador de gramática sobre la entrada  $aabb$ . El número  $i$  en la pila representa al conjunto de ítems  $I_i$  de la Fig. 10.6. El conjunto propio de ítems con los que se cubre un símbolo de gramática dado se determina a partir de las transiciones DFA (Fig. 10.6) exactamente como se hizo para una gramática  $LR(0)$ .

	$a$	$b$	$\$$
$I_0$	Corrimiento	Corrimiento	Se reduce mediante $A \rightarrow \epsilon$ Acepta
$I_1$			
$I_2$	Corrimiento	Corrimiento	Se reduce mediante $A \rightarrow \epsilon$
$I_3$	Corrimiento	Corrimiento	
$I_4$	Se reduce mediante $B \rightarrow b$	Se reduce mediante $B \rightarrow b$	Se reduce mediante $B \rightarrow b$
$I_5$			Se reduce mediante $A \rightarrow BA$
$I_6$	Se reduce mediante $B \rightarrow aB$	Se reduce mediante $B \rightarrow aB$	Se reduce mediante $B \rightarrow aB$

Fig. 10.7 Tabla de decisiones para la Gramática (10.8).

Pila	Entrada restante	Comentarios
0	$aabb\$$	Inicial
0a3	$abb\$$	Corrimiento
0a3a3	$bb\$$	Corrimiento
0a3a3b4	$b\$$	Corrimiento
0a3a3B6	$b\$$	Se reduce mediante $B \rightarrow b$
0a3B6	$b\$$	Se reduce mediante $B \rightarrow aB$
0B2	$b\$$	Se reduce mediante $B \rightarrow aB$
0B2b4	$\$$	Corrimiento
0B2B2	$\$$	Se reduce mediante $B \rightarrow b$
0B2B2A5	$\$$	Se reduce mediante $A \rightarrow \epsilon$
0B2A5	$\$$	Se reduce mediante $A \rightarrow BA$
041	$\$$	Se reduce mediante $A \rightarrow BA$
—	$\$$	Se reduce mediante $S \rightarrow A$ y acepta

Fig. 10.8 Acción del analizador  $LR(1)$  sobre la entrada  $aabb$ .

## EJERCICIOS

**10.1** Demuestre que la forma normal del Lema 10.2 vale para los PDAs no determinísticos.

**10.2**

a) Demuestre que cada DCFL es aceptado por un DPDA cuyos únicos movimientos  $\epsilon$  son movimientos de expulsión.

b) Demuestre que el DPDA de la parte (a) puede construirse de modo que satisfaga la forma normal del Lema 10.2.

**\*10.3** Dé un algoritmo eficiente para instrumentar la regla (4) del Lema 10.3.

**\*S10.4** Demuestre que los DCFLs no son cerrados con respecto a la unión, la concatenación cerradura de Kleene u homomorfismos.

**\*\*10.5** Demuestre que los lenguajes siguientes no son DCFLs.

$$\text{Sa)} \{ww^R \mid w \text{ está en } (0+1)^*\} \quad \text{b)} \{0^n1^n \mid n \geq 1\} \cup \{0^n1^{2n} \mid n \geq 1\}$$

**\*\*10.6** Demuestre que

$$\{0^i1^j2^i \mid i, j \geq 1\} \cup \{0^i1^j2^j \mid i, j \geq 1\}$$

es un DCFL, pero que no es aceptado por ningún DPDA sin movimientos  $\epsilon$ .

**10.7** Muestre que si  $L$  es un DCFL, entonces  $L$  es aceptado por un DPDA que si acepta a  $a_1a_2\dots a_n$ , lo hace inmediatamente después de consumir a  $a_n$  (sin que haya lugar a movimientos  $\epsilon$  posteriores). {Sugerencia: Utilícese la máquina de predicción.}

**10.8** ¿Se aplica el Teorema de Greibach (Teorema 8.14) a los DCFLs?

**10.9** Construya los conjuntos no vacíos de ítems para las gramáticas siguientes. ¿Cuáles de éstas son  $LR(0)$ ?

a)  $S' \rightarrow S$   
 $S \rightarrow aSa \mid bSb \mid c$

b)  $S' \rightarrow S$   
 $S \rightarrow aSa \mid bSb \mid \epsilon$

Sc)  $S \rightarrow E_1$   
 $E_1 \rightarrow T_3 E_1 \mid T_1$   
 $E_2 \rightarrow T_3 E_2 \mid T_2$   
 $T_1 \rightarrow a\$ \mid (E_2 \$$   
 $T_2 \rightarrow a) \mid (E_2)$   
 $T_3 \rightarrow a + \mid (E_2 +$

**10.10** Muestre la sucesión de pilas utilizadas por el DPDA que se construyó a partir de la gramática 10.9(c) cuando la entrada es  $a + (a + a)\$$ .

**10.11** Construya los conjuntos no vacíos de ítems  $LR(1)$  para las siguientes gramáticas. ¿Cuáles de éstas son  $LR(1)$ ?

a)  $S \rightarrow A$   
 $A \rightarrow AB \mid \epsilon$   
 $B \rightarrow aB \mid b$

b)  $S \rightarrow E$   
 $E \rightarrow E + T \mid T$   
 $T \rightarrow a \mid (E)$

**10.12** Repita el Ejercicio 10.10 para la gramática 10.11 (b).

**10.13** Sea  $G$  una gramática  $LR(0)$  con  $A \rightarrow \alpha \cdot, \alpha \neq \epsilon$ , válido para algún prefijo viable  $\gamma$ . Demuestre que ninguna otra producción puede ser válida para  $\gamma$ .

### Soluciones a los ejercicios seleccionados

**10.4** Sean  $L_1 = \{0^i1^j2^i \mid i, j \geq 0\}$  y  $L_2 = \{0^n1^n \mid n \geq 0\}$ . Es fácil mostrar que  $L_1$  y  $L_2$  son DCFLs. Sin embargo,  $L_1 \cup L_2$  es el CFL que demostramos no era un DCFL en el Ejemplo 10.1. Por consiguiente los DCFLs no son cerrados con respecto a la unión.

Para la concatenación, sea  $L_3 = aL_1 \cup L_2$ . Entonces  $L_3$  es un DCFL, ya que la presencia o ausencia del símbolo  $a$  nos dice dónde buscar una palabra de  $L_1$  o de  $L_2$ . Seguramente,  $a^*$  es un DCFL. Sin embargo  $a^*L_3$  no es un DCFL. Si lo fuera, entonces  $L_4 = a^*L_3 \cap a0^*1^*2^*$  sería un DCFL, según el Teorema 10.4. Pero  $L_4 = aL_1 \cup aL_2$ . Si  $L_4$  es un DCFL, aceptado por el DPDA  $M$ , entonces podríamos reconocer a  $L_1 \cup L_2$  mediante la simulación de  $M$  sobre la entrada (imaginaria)  $a$  y después sobre la entrada real. Como  $L_1 \cup L_2$  no es un DCFL, ni lo es  $L_4$ , y por tanto los DCFLs no son cerrados con respecto a la concatenación.

La demostración para la cerradura es parecida, si hacemos  $L_5 = \{a\} \cup L_3$ ,  $L_5$  es un DCFL, pero  $L_5^*$  no, según una demostración similar a la hecha arriba.

Para el homomorfismo, sea  $L_6 = aL_1 \cup bL_2$ , que es un DCFL. Sea  $L$  el homomorfismo que transforma  $b$  a  $a$  y transforma a otros símbolos en sí mismos. Entonces  $h(L_6) = L_4$ , así pues, los DCFLs no son cerrados con respecto al homomorfismo.

**10.5(a)** Supóngase que  $L_1 = \{ww^R \mid w \text{ está en } (0+1)^*\}$  es un DCFL. Entonces, según el Teorema 10.4, también lo sería

$$L_2 = L_1 \cap (01)^*(10)^*(01)^*(10)^*.$$

Ahora

$$L_2 = \{(01)^i(10)^j(01)^j(10)^i \mid i, j \geq 0, i \text{ y } j \text{ no ambos } 0\}.$$

Por el Teorema 10.3,  $L_3 = \text{MIN}(L_2)$  es un DCFL. Pero

$$L_3 = \{(01)^i(10)^j(01)^j(10)^i \mid 0 \leq j < i\},$$

puesto que si  $j \geq i$ , se tiene un prefijo en  $L_2$ . Sea  $L$  el homomorfismo  $h(a) = 01$  y  $h(b) = 10$ . Entonces

$$L_4 = h^{-1}(L_3) = \{a^jb^ja^jb^i \mid 0 \leq j < i\}$$

es un DCFL, según el Teorema 10.4. Sin embargo el lema de sondeo con  $z = a^{n+1}b^nb^na^{n+1}$  muestra que  $L_4$  no es ni siquiera un CFL.

**10.9 (c)** Antes de abordar este proyecto, describamos el lenguaje de la gramática. Primero describimos “expresión” y “término” de manera recursiva, de la manera siguiente.

1. Un término es un símbolo simple  $a$  que representa a cualquier “argumento de una expresión aritmética o de una expresión que está entre paréntesis.”
2. Una expresión consiste en uno o más términos conectados mediante signos más.

Entonces el lenguaje de esta gramática es el conjunto de todas las expresiones seguidas por un señalador de extremo, \$. \$,  $E_1$  y  $T_1$  generan expresiones y términos seguidos por un \$.  $E_2$  y  $T_2$  generan expresiones y términos seguidos por un paréntesis derecho y  $T_3$  genera un término seguido por un signo más.

Resulta que las gramáticas  $LR(0)$  que definen expresiones aritméticas, de las cuales nuestra gramática es un ejemplo sencillo, son bastante complicadas, en comparación con una gramática  $LR(1)$  para el mismo lenguaje [véase el Ejercicio 10.11 (b)]. Por esta razón los sistemas compiladores de escritura nunca requieren que la sintaxis de un lenguaje sea descrita por una gramática  $LR(0)$ ; Las gramáticas  $LR(1)$ , o un subconjunto de éstas, son preferibles. Sin embargo,

$I_0$	$I_5$	$I_{10}$	$I_{14}$
$S \rightarrow \cdot E_1$	$T_3 \rightarrow (\cdot E_2 +$	$E_2 \rightarrow T_3 \cdot E_2$	$T_3 \rightarrow (E_2 + \cdot$
$E_1 \rightarrow \cdot T_3 E_1$	$T_1 \rightarrow (\cdot E_2 \$$	$E_2 \rightarrow \cdot T_3 E_2$	
$E_1 \rightarrow \cdot T_1$	$E_2 \rightarrow \cdot T_3 E_2$	$E_2 \rightarrow \cdot T_2$	$I_{15}$
$T_3 \rightarrow \cdot a +$	$E_2 \rightarrow \cdot T_2$	$T_3 \rightarrow \cdot a +$	$T_1 \rightarrow (E_2 \$ \cdot$
$T_3 \rightarrow \cdot (E_2 +$	$T_3 \rightarrow \cdot a +$	$T_3 \rightarrow \cdot (E_2 +$	
$T_1 \rightarrow \cdot a \$$	$T_3 \rightarrow \cdot (E_2 +$	$T_2 \rightarrow \cdot a)$	$I_{16}$
$T_1 \rightarrow \cdot (E_2 \$$	$T_2 \rightarrow \cdot a)$	$T_2 \rightarrow \cdot (E_2)$	$E_2 \rightarrow T_3 E_2$
		$T_2 \rightarrow \cdot (E_2)$	
$I_1$	$I_6$	$I_{11}$	$I_{17}$
$S \rightarrow E_1 \cdot$	$E_1 \rightarrow T_3 E_1 \cdot$	$E_2 \rightarrow T_2 \cdot$	$T_2 \rightarrow a) \cdot$
$I_2$	$I_7$	$I_{12}$	$I_{18}$
$E_1 \rightarrow T_3 \cdot E_1$	$T_3 \rightarrow a + \cdot$	$T_3 \rightarrow a \cdot +$	$T_3 \rightarrow (E_2 \cdot +$
$E_1 \rightarrow \cdot T_3 E_1$		$T_2 \rightarrow a \cdot )$	$T_2 \rightarrow (E_2 \cdot )$
$E_1 \rightarrow \cdot T_1$			
$T_3 \rightarrow \cdot a +$			
$T_3 \rightarrow \cdot (E_2 +$			
$T_1 \rightarrow \cdot a \$$			
$T_1 \rightarrow \cdot (E_2 \$$			
$I_3$	$I_8$	$I_{13}$	$I_{19}$
$E_1 \rightarrow T_1 \cdot$	$T_1 \rightarrow a \$ \cdot$	$T_3 \rightarrow (\cdot E_2 +$	$T_2 \rightarrow (E_2) \cdot$
		$T_2 \rightarrow (\cdot E_2)$	
		$E_2 \rightarrow \cdot T_3 E_2$	
		$E_2 \rightarrow \cdot T_2$	
		$T_3 \rightarrow \cdot a +$	
		$T_3 \rightarrow \cdot (E_2 + \cdot$	
		$T_2 \rightarrow \cdot a)$	
		$T_2 \rightarrow \cdot (E_2)$	
$I_4$	$I_9$		
$T_3 \rightarrow a \cdot +$	$T_3 \rightarrow (E_2 \cdot +$		
$T_1 \rightarrow a \cdot \$$	$T_1 \rightarrow (E_2 \cdot \$$		

Fig. 10.9 Conjuntos de ítems para el Ejercicio 10.9(c).

la presente gramática servirá como un Ejercicio útil. El DSFA que acepta prefijos viables tiene 20 estados, sin contar el estado muerto. En la Fig. 10.9 colocamos en una tabla a estos conjuntos de ítems. La Fig. 10.10 da la tabla de transiciones para el DFA; los espacios en blanco indican transiciones al estado muerto.

Una inspección de los conjuntos de ítems nos dice que ciertos conjuntos, a saber, 1, 3, 6, 7, 8, 11, 14, 15, 16, 17 y 19, consisten en un solo ítem completo, mientras que los restantes no contienen ítems completos. Por consiguiente la gramática es  $LR(0)$ .

## NOTAS BIBLIOGRAFICAS

Los autómatas de pila determinísticos fueron estudiados por primera vez por Fischer [1963], Schutzenberg [1963], Haines [1965] y Ginsburg y Greibach [1966a]. El Lema 10.3, sobre el hecho de que los DPDA's pueden ser construidos de modo que consuman completamente su entrada, se tomó de Schutzenberg [1963]; el Teorema 10.1, cerradura con respecto a la complementación, fue observado, de manera independiente, por diferentes autores. La mayoría de las propiedades de cerradura y decisión, Teoremas 10.2 a 10.8, fueron demostrados por primera vez por Ginsburg y Greibach [1966a]. El hecho que consiste en que es resoluble si un DCFL es regular constituye una excepción, lo que fue demostrado por Stearns [1967]. La construcción de la máquina de predicción fue tomada de Hopcroft y Ullman [1969b].

Las gramáticas  $LR(k)$  y la equivalencia de los DPDA's con las gramáticas  $LR(1)$  se tomaron de Knuth [1965]. El último trabajo dio lugar a una sucesión de artículos que trataron sobre las subclases de las CFGs que tienen algoritmos para análisis gramatical eficientes. La historia de

	$a$	$+$	(	)	\$	$E_1$	$E_2$	$T_1$	$T_2$	$T_3$
0	4		5			1		3		2
1										
2	4		5				6	3		2
3										
4		7			8					
5	12		13				9		11	10
6										
7										
8										
9		14			15					
10	12		13				16		11	10
11										
12		7		17						
13	12		13				18		11	10
14										
15										
16										
17										
18		14		19						
19										

Fig. 10.10 Tabla de transiciones del DFA que reconoce prefijos viables.

este desarrollo se encuentra descrita en Aho y Ullman [1972, 1973]. Graham [1970] muestra que un cierto número de otras clases de gramáticas definen exactamente a los CFLs.

Después del trabajo de Knuth [1965], una serie de artículos examinaron la clase de las gramáticas  $LR(1)$  como una subclase útil para la cual se podían construir analizadores de gramática de tamaño razonable. Korenjak [1969] fue el primero en establecer uno de estos métodos, aunque los métodos de mayor uso en la actualidad son dos subclases de gramáticas  $LR(1)$ , conocidas como  $SLR(1)$  (la S por “simple”) y  $LALR(1)$  (LA por adelante, “lookahead” en inglés), debidas a DeRemer [1969, 1971]. A modo de comparación, un lenguaje de programación típico, como ALGOL, tiene un analizador  $LR(1)$  (un DFA que reconoce un prefijo viable) con varios miles de estados; se necesitan todavía más estados para un analizador  $LR(0)$ . Como la tabla de transiciones debe ser parte del analizador de gramática de un lenguaje, no resulta factible almacenar un analizador gramatical tan grande en la memoria principal de la computadora, incluso si la tabla es compacta. Sin embargo, los mismos lenguajes tienen analizadores  $SLR(1)$  o  $LALR(1)$  de unos cuantos cientos de estados, lo que es apropiado para la compactación. Véase Aho y Johnson [1974] o Aho y Ullman [1977], en donde se da una descripción de la forma en que los analizadores  $LR$  se diseñan y utilizan.

Una gran parte de la investigación se ha enfocado sobre la cuestión abierta de si la equivalencia es soluble para los DPDAs. Korenjak y Hopcroft [1966] demostraron que la equivalencia es soluble para una subclase de los DCFLs, conocidos como lenguajes “simples”.† Estos están definidos mediante gramáticas que se encuentran en la forma normal de Greibach, y tales que no tienen ninguna dos producciones  $A \rightarrow a\alpha$  y  $A \rightarrow a\beta$ . La resolubilidad de la equivalencia fue extendida a las gramáticas  $LL(k)$  de Lewis y Stearns [1968], las cuales son un subconjunto propio de las gramáticas  $LR(k)$ , por Rosenkrantz y Stearns [1970]. Valiant [[1973] demostró que la equivalencia es soluble para los DPDAs de vueltas finitas (véase el Ejercicio 6.13 para su definición), entre otras clases; véase también Valiant [1974], Beeri [1976] y Taniguchi y Kasami [1976]. Friedman [1977] demostró que la equivalencia para los DPDAs es resoluble si y sólo si es resoluble para los “esquemas de recursión monádicos”, que en términos de autómatas pueden considerarse como DPDAs de un solo estado que pueden basar su movimiento siguiente en el símbolo de entrada actual, sin que se consuma dicho símbolo.

De manera adicional, se ha hecho trabajo en la extensión de la no resolubilidad de la contención para los DCFLs a pequeños subconjuntos del mismo tipo de lenguajes. El trabajo tuvo su culminación con el de Friedman [1976], quien demostró que la contención es irresoluble incluso para los lenguajes simples de Korenjak y Hopcroft [1966].

En Ginsburg y Greibach se encuentra una solución al Ejercicio 10.5 (b), y el Ejercicio 10.6 está basado en Cole [1969].

## CAPITULO

## 11

PROPIEDADES DE CERRADURA  
DE FAMILIAS DE  
LENGUAJES

Existen sorprendentes similitudes entre las propiedades de cerradura de los conjuntos regulares, los lenguajes libres de contexto, los conjuntos r.e. y otras clases. No solamente son parecidas las propiedades de cerradura, sino que también lo son las técnicas de demostración que se utilizan para establecer estas propiedades. En este capítulo haremos un planteamiento general y estudiaremos a todas las familias de lenguajes que poseen ciertas propiedades de cerradura. Lo anterior nos proporcionará una nueva visión de la estructura subyacente de las propiedades de cerradura y simplificará el estudio de nuevas clases de lenguajes.

## 11.1 TRIOS Y TRIOS COMPLETOS

Recuérdese que un lenguaje es un conjunto de cadenas de longitud finita sobre algún alfabeto finito. Una *familia de lenguajes* es una colección de lenguajes que contiene al menos un lenguaje no vacío. Un *trío* es una familia de lenguajes cerrados con respecto a la intersección con un conjunto regular, al homomorfismo inverso y al homomorfismo libre de movimientos  $\in$  (hacia adelante). [Decimos que un homomorfismo  $h$  es *libre de movimientos* si  $h(a) \neq \in$  para cualquier símbolo  $a$ ]. Si la familia de lenguajes es cerrada con respecto a todos los homomorfismos, con respecto al homomorfismo inverso y a la intersección con un conjunto regular, entonces diremos que se trata de un *trío completo*.

**Ejemplo 11.1** Los conjuntos regulares, los lenguajes libres de contexto y los conjuntos r.e. son trios completos. Los lenguajes sensibles al contexto y los conjuntos

† Estos no están relacionados de ninguna forma sustancial con las gramáticas  $LR$  “simples”.

recursivos son tríos pero no tríos completos, puesto que no son cerrados con respecto a cualquier homomorfismo. De hecho, cerrar a los CSLs o a los conjuntos recursivos con respecto a cualquier homomorfismo produce a los conjuntos r.e. (véase el Ejercicio 9.14 y su solución).

En el Teorema 3.3 se demuestra que los conjuntos regulares son cerrados con respecto a la intersección; por consiguiente son cerrados con respecto a “la intersección con un conjunto regular”. El Teorema 3.5 muestra la cerradura de los conjuntos regulares con respecto a los homomorfismos y al homomorfismo inverso, completando la prueba de que los conjuntos regulares forman un trío completo. Los corolarios de los Teoremas 6.2, 6.3 y 6.5 muestran que los CFLs son un trío completo. El Ejercicio 9.10 y su intersección nos proporcionan una demostración de que los CSLs son cerrados con respecto al homomorfismo inverso, a la intersección (y por tanto con respecto a la intersección con un conjunto regular) y a la sustitución (por tanto con respecto al homomorfismo libre de movimientos  $\in$ ; pero no con respecto a todos los homomorfismos, ya que  $\in$  no está permitido en un CSL). Por consiguiente los CSLs forman un trío, pero no un trío completo.

Demostraremos que los conjuntos recursivos son un trío, y dejaremos como ejercicio la demostración de que los conjuntos r.e. forman un trío completo. Sea  $h$  un homomorfismo y  $L$  un lenguaje recursivo reconocido por el algoritmo  $A$ . Entonces  $h^{-1}(L)$  es reconocido por el algoritmo  $B$ , el cual simplemente aplica  $A$  a  $h(w)$ , en donde  $w$  es la entrada de  $B$ . Sea  $g$  un homomorfismo libre de movimientos  $\in$ . Entonces  $g(L)$  es reconocido por el algoritmo  $C$  el cual, dada la entrada  $w$  de longitud  $n$ , enumera todas las palabras  $x$  de longitud de hasta  $n$  sobre el alfabeto de dominio de  $g$ . Para cada  $x$  tal que  $g(x) = w$ , el algoritmo  $A$  se aplica a  $x$ , y si  $x$  se encuentra en  $L$ , el algoritmo  $C$  acepta a  $w$ . Nótese que ya que  $g$  está libre de movimientos  $\in$ ,  $w$  no puede ser  $g(x)$  si  $|x| > |w|$ . Finalmente, si  $R$  es un conjunto regular aceptado por el DFA  $M$ , podemos construir el algoritmo  $D$  que acepte a la entrada  $w$  si y sólo si  $A$  acepta a  $w$  y  $M$  lo hace también.

Concluimos esta sección haciendo la observación de que cada trío completo contiene a todos los conjuntos regulares. Por consiguiente los conjuntos regulares conforman el trío completo más pequeño. También, llegamos a la conclusión de que los conjuntos regulares libres de  $\in$  son el trío más pequeño. (Un lenguaje es *libre de  $\in$*  si  $\in$  no se encuentra en el lenguaje.)

**Lema 11.1** Cada trío completo contiene a todos los conjuntos regulares; cada trío contiene a todos los conjuntos regulares libres de  $\in$ .

**Demostración** Sea  $\mathcal{C}$  un trío,  $\Sigma$  un alfabeto y  $R \subseteq \Sigma^*$  un conjunto regular libre de  $\in$ . Puesto que  $\mathcal{C}$  contiene al menos un lenguaje no vacío, hagamos que  $L$  se encuentre en  $\mathcal{C}$  y  $w$  en  $L$ . Defínase  $\Sigma' = \{a' | a \text{ se encuentra en } \Sigma\}$  y sea  $h$  el homomorfismo que transforma a cada  $a$  de  $\Sigma$  en  $\in$  y a cada  $a'$  de  $\Sigma'$  en  $w$ . Entonces  $L_1 = h^{-1}(L)$  se encuentra en  $\mathcal{C}$  debido a que  $\mathcal{C}$  es un trío. Como  $w$  se encuentra en  $L$ ,  $L_1$  contiene a todas las cadenas de  $\Sigma' \Sigma^*$ , y también a otras. Sea  $g$  el homomorfismo  $g(a') = g(a) = a$  para toda  $a$  en  $\Sigma$ . Entonces, puesto que  $g$  es libre de movimientos  $\in$ , sabemos que  $L_2 = g(L_1)$  se encuentra en  $\mathcal{C}$  y está en  $\Sigma^*$  o en  $\Sigma^+$ , dependiendo de si  $L_1$  contiene o no a  $\in$ . Por

consiguiente  $L_2 \cap R = R$  se encuentra en  $\mathcal{C}$ , demostrándose así nuestra afirmación de que cada trío contiene a todos los conjuntos regulares libres de  $\in$ .

Si  $\mathcal{C}$  es un trío completo, podemos modificar la demostración anterior haciendo  $g'(a') = \in$  y  $g'(a) = a$  para toda  $a$  en  $\Sigma$ . Entonces  $L_2 = g'(L_1) = \Sigma^*$ . Si  $R$  es cualquier conjunto regular,  $L_2 \cap R = R$  se encuentra en  $\mathcal{C}$ .  $\square$

Dejamos como ejercicio demostrar que los conjuntos regulares conforman un trío y por consiguiente el trío más pequeño. Adviértase que no forman un trío completo, debido a que no son cerrados con respecto a todos los homomorfismos.

## 11.2 TRANSFORMACIONES DE MAQUINAS SECUENCIALES GENERALIZADAS

Al estudiar las propiedades de cerradura, uno observa inmediatamente que ciertas propiedades se concluyen automáticamente de otras. Así pues, para establecer un conjunto de propiedades de cerradura para una clase de lenguajes uno necesita solamente establecer un conjunto de propiedades a partir de las cuales se concluyen las otras. En esta sección estableceremos un cierto número de propiedades de cerradura que se concluyen de las propiedades básicas de los tríos y los tríos completos.

La primera operación que consideraremos es una generalización de un homomorfismo. Considérese una máquina de Mealy a la cual se le permite emitir cualquier cadena, incluyendo  $\in$ , en un movimiento. Este dispositivo se conoce como una máquina secuencial generalizada, y la transformación que define se conoce como la transformación de una máquina secuencial generalizada.

De manera más formal una *máquina secuencial generalizada* (GSM) es un conjunto de seis parámetros  $M = (Q, \Sigma, \Delta, \delta, q_0, F)$ , en donde  $Q$ ,  $\Sigma$  y  $\Delta$  son los *estados*, el *alfabeto de entrada* y el *alfabeto de salida*, respectivamente,  $\delta$  es una transformación de  $Q \times \Sigma$  a los subconjuntos finitos de  $Q \times \Delta^*$ ,  $q_0$  es el *estado inicial* y  $F$  es el conjunto de *estados finales*. La interpretación de  $(p, w)$  en  $\delta(q, a)$  es que  $M$  en el estado  $q$  con símbolo de entrada  $a$ , como una posible alternativa de movimiento, puede accesar al estado  $p$  y emitir a la cadena  $w$ .

Extendemos el dominio de  $\delta$  a  $Q \times \Sigma^*$  de la manera siguiente.

$$1. \delta(q, \in) = \{(q, \in)\}.$$

$$2. \text{para } x \text{ en } \Sigma^* \text{ y } a \text{ en } \Sigma,$$

$$\delta(q, xa) = \{(p w) | w = w_1 w_2 \text{ y para alguna } p', \\ (p', w_1) \text{ está en } \delta(q, x) \text{ y } (p, w_2) \text{ está en } \delta(p', a)\}.$$

Una GSM está *libre de movimientos  $\in$*  si  $\delta$  transforma  $Q \times \Sigma$  en los conjuntos finitos de  $Q \times \Delta^*$ . Hagamos que  $M(x)$ , donde  $M$  que es la GSM definida más arriba, denote el conjunto

$$\{y | (p, y) \text{ está en } \delta(q_0, x) \text{ para alguna } p \text{ en } F\}.$$

Si  $L$  es un lenguaje sobre  $\Sigma$ , hagamos que  $M(L)$  represente a

$$\{y | y \text{ está en } M(x) \text{ para alguna } x \text{ en } L\}.$$

Decimos que  $M(L)$  es una *transformación GSM*. Si  $M$  es libre de  $\epsilon$ , entonces  $M(L)$  es una *transformación GSM libre de  $\epsilon$* . Nótese que  $L$  es un parámetro de la transformación, no un lenguaje dado.

Hagamos también

$$M^{-1}(x) = \{y \mid M(y) \text{ contiene a } x\},$$

y

$$M^{-1}(L) = \{y \mid x \text{ está en } M(y) \text{ para alguna } x \text{ en } L\}.$$

Decimos que  $M^{-1}$  es una *transformación GSM inversa*. No es necesariamente verdadero que  $M^{-1}(M(L)) = M(M^{-1}(L)) = L$ , de modo que  $M^{-1}$  no es un inverso verdadero.

**Ejemplo 11.2** Sea

$$M = (\{q_0, q_1\}, \{0, 1\}, \{a, b\}, \delta, q_0, \{q_1\}).$$

Definimos  $\delta$  mediante

$$\delta(q_0, 0) = \{(q_0, aa), (q_1, b)\},$$

$$\delta(q_0, 1) = \{(q_0, a)\},$$

$$\delta(q_1, 0) = \emptyset,$$

$$\delta(q_1, 1) = \{(q_1, \epsilon)\}.$$

Podemos dibujar un GSM como un autómata finito, con una arista etiqueta con  $a/w$  del estado  $q$  al estado  $p$  si  $\delta(q, a)$  contiene a  $(p, w)$ . En la Fig. 11.1 se muestra el diagrama para la  $M$  anterior. De manera intuitiva, como los 0s son entrada para  $M$ , ésta tiene la alternativa de emitir dos  $a$ s o de emitir una  $b$ . Si  $M$  emite la  $b$ , se traslada al estado  $q_1$ . Si 1 es entrada de  $M$  y  $M$  se encuentra en el estado  $q_0$ , sólo puede producir como salida una  $a$ . En el estado  $q_1$ ,  $M$  se muere si la entrada es 0, pero puede permanecer en el estado  $q_1$ , sin producir salida alguna cuando la entrada es 1.

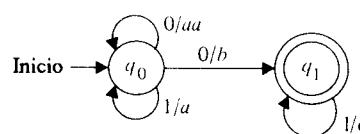


Fig. 11.1 Una GSM.

Sea  $L = \{0^n 1^n \mid n \geq 1\}$ . Entonces

$$M(L) = \{a^{2n}b \mid n \geq 0\}.$$

Debido a que los 0s son leídos por  $M$ , ésta emite dos  $a$ s por 0, hasta que en algún momento adivine que debería emitir el símbolo  $b$  e ir al estado  $q_1$ . Si los 1s no siguen

inmediatamente después en la entrada,  $M$  muere. 0 si  $M$  prefiere permanecer en el estado  $q_0$  cuando lee el primer 1, nunca puede alcanzar el estado  $q_1$  si la entrada es de la forma  $0^* 1^*$ . Por consiguiente la única entrada que  $M$  hace cuando se le da la entrada  $0^* 1^*$  es  $a^{2n-2}b$ .

Si  $L_1 = \{a^{2n}b \mid n \geq 0\}$ , entonces

$$M^{-1}(L_1) = \{w01^* \mid i \geq 0 \text{ y } w \text{ tiene un número par de 1s}\}.$$

Nótese que  $M^{-1}(M(L)) \supsetneq L$ .

La transformación GSM es una herramienta útil para expresar un lenguaje en términos de un segundo lenguaje que tenga esencialmente la misma estructura, pero diferentes aditamentos externos. Por ejemplo,  $L_1 = \{a^n b^n \mid n \geq 1\}$  y  $L_2 = \{a^n b a^n \mid n \geq 1\}$  en algún sentido tienen la misma estructura, pero difieren un poco.  $L_1$  y  $L_2$  pueden expresarse fácilmente en términos uno del otro mediante transformaciones GSM. La Figura 11.2(a) muestra una GSM que transforma a  $L_1$  en  $L_2$ , y la Fig. 11.2(b) muestra una GSM que transforma a  $L_2$  en  $L_1$ .

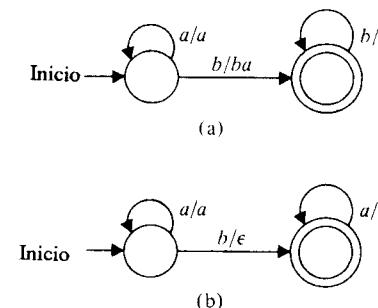


Fig. 11.2 Dos GSMs.

### Cerradura con respecto a las transformaciones GSM

Un factor clave con respecto a las transformaciones GSM consiste en que pueden ser expresados en términos de homomorfismos y la intersección con los conjuntos regulares. Por consiguiente cualquier clase de lenguajes cerrados con respecto a las operaciones citadas es cerrado con respecto a las transformaciones GSM.

**Teorema 11.1** Cada trío completo es cerrado con respecto a las transformaciones GSM. Cada trío es cerrado con respecto a las transformaciones GSM libres de  $\epsilon$ .

**Demostración** Sea  $\mathcal{C}$  un trío completo,  $L$  un miembro de  $\mathcal{C}$  y  $M = (Q, \Sigma, \Delta, \delta, q_0, F)$  una GSM. Debemos demostrar que  $M(L)$  se encuentra en  $\mathcal{C}$ . Hagamos

$$\Delta_1 = \{[q, a, x, p] \mid \delta(q, a) \text{ contiene a } (p, x)\}$$

y sean  $h_1, h_2$  los homomorfismos de  $\Delta_1^*$  a  $\Sigma^*$  y de  $\Delta_1^*$  a  $\Delta^*$  definidos mediante  $h_1([q, a, x, p]) = a$  y  $h_2([q, a, x, p]) = x$ . Sea  $R$  el conjunto regular de todas las cadenas de  $\Delta_1^*$  tales que

1. la primera componente del primer símbolo es  $q_0$ , el estado inicial de  $M$ ;
2. la última componente del último símbolo es un estado final de  $M$ ;
3. la última componente de cada símbolo es la misma que la primera componente del símbolo exitoso.

Es fácil verificar que  $R$  es un conjunto regular. Un DFA puede verificar la condición (3) recordando el símbolo anterior en su estado y comparándolo con el símbolo actual.

Si  $\epsilon$  no está en  $L$ , entonces  $M(L) = h_2(h_1^{-1}(L) \cap R)$ . Es decir  $h_1^{-1}$  transforma la entrada de  $M$  a una cadena que tiene codificado en ella, para cada símbolo de la cadena de entrada, una transición de estado posible de  $M$  en el símbolo de entrada y una cadena de salida correspondiente. El conjunto regular fuerza a la secuencia de estados a que sean una sucesión posible de transiciones de estado de  $M$ . Finalmente,  $h_2$  borra la entrada y las transiciones de estado, dejando solamente la cadena de salida. De manera formal,

$$h_1^{-1}(L) = \{[p_1, a_1, x_1, q_1][p_2, a_2, x_2, q_2] \cdots [p_k, a_k, x_k, q_k] \mid a_1 a_2 \cdots a_k \\ \text{está en } L, \text{ los } p_i \text{ son arbitrarios y } (q_i, x_i) \text{ se encuentra en } \delta(p_i, a_i)\}.$$

Haciendo la intersección de  $h_1^{-1}(L)$  con  $R$  se tiene

$$L = \{[q_0, a_1, x_1, q_1][q_1, a_2, x_2, q_2] \cdots [q_{k-1}, a_k, x_k, q_k] \mid a_1 a_2 \cdots a_k \\ \text{está en } L, q_k \text{ está en } F \text{ y } \delta(q_{i-1}, a_i) \text{ contiene a } (q_i, x_i) \text{ para toda } i\}$$

De donde, por definición,  $h_2(L')$  es  $M(L)$ .

Si  $\epsilon$  está en  $L$  y  $q_0$  no es un estado final, entonces  $h_2(L')$  es todavía  $M(L)$  pero si  $\epsilon$  se encuentra en  $L$  y  $q_0$  es un estado final, entonces  $M(\epsilon) = \epsilon$ , de modo que debemos modificar la construcción anterior para asegurarnos de que  $\epsilon$  se encuentre en el lenguaje resultante. Sea  $L'' = h_1^{-1}(L) \cap (R + \epsilon)$ . Entonces  $L'' = L' \cup \{\epsilon\}$ , ya que  $\epsilon$  se encuentra en  $h_1^{-1}(L)$ . Por consiguiente  $h_2(L'') = M(L)$ . Puesto que cada trío completo es cerrado con respecto a la intersección con un conjunto regular, con respecto al homomorfismo y al homomorfismo inverso.  $M(L)$  se encuentra en  $\mathcal{C}$ .

La demostración para los tríos y las transformaciones GSM libres de  $\epsilon$  se desarrolla de una manera similar. Puesto que la GSM nunca emite  $a \in \Sigma$ , la  $x$  de  $[q, a, x, p]$  nunca es  $\epsilon$ , y en consecuencia  $h_2$  es un homomorfismo libre de  $\epsilon$ .  $\square$

### Borrado limitado y transformaciones GSM inversas

Los tríos no necesariamente son cerrados con respecto a los homomorfismos que tienen como resultado un borrado arbitrario. Sin embargo, los tríos son cerrados con respecto a ciertos homomorfismos que permiten el borrado, siempre y cuando éste sea limitado. Se dice que una clase de lenguajes es cerrada con respecto al *borrado limitado k* si para cualquier lenguaje  $L$  de la clase y cualquier homomorfismo  $h$  tal que  $h$  nunca transforma a más de  $k$  símbolos consecutivos, de cualquier oración  $x$  en  $L$ , a  $\epsilon$ ,  $h(L)$  se encuentra en la clase. Esta es cerrada con respecto al *borrado limitado k* si es cerrada con respecto al *borrado limitado k* para toda  $k$ . Nótese que si  $h(a)$  es  $\epsilon$  para alguna  $a$ , entonces el hecho de que  $h$  sea un borrado limitado en  $L$  depende de  $L$ .

**Lema 11.2** Cada trío es cerrado con respecto al borrado limitado.

*Demostración* Sea  $\mathcal{C}$  un trío,  $L \subseteq \Sigma^*$  un miembro de  $\mathcal{C}$  y  $h$  un homomorfismo que es  $k$  limitado en  $L$ . Hagamos que

$$\Sigma_2 = \{[x] \mid x \text{ está en } \Sigma_1^*, |x| \leq k+1 \text{ y } h(x) \neq \epsilon\}.$$

Sean  $h_1$  y  $h_2$  los homomorfismos definidos por

$$h_1([a_1 a_2 \cdots a_m]) = a_1 a_2 \cdots a_m \text{ y } h_2([a_1 a_2 \cdots a_m]) = h(a_1 a_2 \cdots a_m).$$

Puesto que  $[a_1 a_2 \cdots a_m]$  se encuentra solamente en  $\Sigma_2$  si  $h(a_1 a_2 \cdots a_m) \neq \epsilon$ ,  $h_2$  es libre de  $\epsilon$ . Entonces  $h_2(h_1^{-1}(L))$  se encuentra en  $\mathcal{C}$  ya que éste es cerrado con respecto a los homomorfismos libres de  $\epsilon$  y con respecto a todos los homomorfismos inversos. Resulta sencillo verificar que  $h_2(h_1^{-1}(L)) = h(L)$ .  $\square$

**Teorema 11.2** Cada trío es cerrado con respecto a las transformaciones GSM inversas.

*Demostración* Sea  $\mathcal{C}$  un trío,  $L$  un miembro de  $\mathcal{C}$  y  $M = (Q, \Sigma, \Delta, \delta, q_0, F)$  una GSM. Sin pérdida de generalidad supóngase que los conjuntos  $\Sigma$  y  $\Delta$  son disjuntos. Si no, sustitúyanse los símbolos de  $\Delta$  por nuevos símbolos y restitúyalos al final de la construcción mediante un homomorfismo libre de  $\epsilon$  que transforme a cada símbolo nuevo al símbolo viejo correspondiente. Sea  $h_1$  el homomorfismo que transforma  $(\Sigma \cup \Delta)^*$  a  $\Delta^*$  definido por

$$h_1(a) = \begin{cases} a & \text{para } a \text{ en } \Delta \\ \epsilon & \text{para } a \text{ en } \Sigma. \end{cases}$$

Hagamos  $L_1 = h_1^{-1}(L)$ . Entonces  $L_1$  es el conjunto de cadenas de  $\Sigma^* b_1 \Sigma^* b_2 \cdots \Sigma^* b_n \Sigma^*$ , tal que  $b_1 b_2 \cdots b_n$  se encuentra en  $L$ .

Sea  $R$  el conjunto regular que consiste en todas las palabras de la forma  $a_1 x_1 a_2 x_2 \cdots a_m x_m$  tales que

1. las  $as$  se encuentran en  $\Sigma$ ,
2. las  $xs$  se encuentran en  $\Delta^*$ ,
3. existen estados  $q_0, q_1, \dots, q_m$  tales que  $q_m$  se encuentra en  $F$  y, para  $1 \leq i \leq m$ ,  $\delta(q_{i-1}, a_i)$  contiene a  $(q_i, x_i)$ .

Nótese que  $x_i$  puede ser  $\epsilon$ . El lector puede demostrar fácilmente que  $R$  es un conjunto regular mediante la construcción de un autómata finito no determinístico que acepte a  $R$ . Este NFA adivina la sucesión de estados  $q_1, q_2, \dots, q_m$ .

Ahora  $L_1 \cap R$  es el conjunto de todas las palabras que se encuentran en  $R$  de la forma  $a_1 x_1 a_2 x_2 \cdots a_m x_m$ ,  $m \geq 0$ , en donde las  $as$  se encuentran en  $\Sigma$ , las  $xs$  en  $\Delta^*$ ,  $x_1 x_2 \cdots x_m$  en  $L$  y  $\delta(q_0, a_1 a_2 \cdots a_m)$  contiene a  $(p, x_1 x_2 \cdots x_m)$ , para alguna  $p$  en  $F$ . Ninguna de las  $x_i$  es de longitud mayor que  $k$ , en donde  $k$  es la longitud de la  $x$  más larga tal que  $(p, x)$  se encuentra en  $\delta(q, a)$  para algunas  $p$  y  $q$  en  $Q$  y  $a$  en  $\Sigma$ .

Finalmente, sea  $h_2$  el homomorfismo que transforma  $a$  en  $a$  para cada  $a$  en  $\Sigma$ , y  $a$  en  $\epsilon$  para cada  $b$  en  $\Delta$ . Entonces

$$M^{-1}(L) = h_2(L_1 \cap R)$$

se encuentra en  $\mathcal{C}$  según el lema 11.2, puesto que  $h_2$  nunca hace que más de  $k$  símbolos consecutivos sean transformados a  $\epsilon$ .  $\square$

### 11.3 OTRAS PROPIEDADES DE CERRADURA PARA TRIOS

Los trios y los trios completos son cerrados con respecto a muchas otras operaciones. En esta sección presentaremos varias de tales propiedades.

**Teorema 11.3** Cada trío completo es cerrado con respecto al cociente con un conjunto regular.

**Demostración** Sean  $\mathcal{C}$  un trío completo,  $L \subseteq \Sigma_1^*$  un miembro de  $\mathcal{C}$  y  $R \subseteq \Sigma_1^*$  un conjunto regular. Para cada  $a$  en  $\Sigma_1$  sea  $a'$  un nuevo símbolo y hagamos que  $\Sigma'_1$  sea el conjunto de tales símbolos. Sean  $h_1$  y  $h_2$  los homomorfismos de  $(\Sigma_1^* \cup \Sigma'_1)^*$  a  $\Sigma_1$  definidos por  $h_1(a) = h_1(a') = a$ ,  $h_2(a) = \epsilon$  y  $h_2(a') = a$ . Entonces  $L/R = h_2(h_1^{-1}(L) \cap (\Sigma'_1)^* R)$ , y en consecuencia  $L/R$  se encuentra en  $\mathcal{C}$ . Esto es,  $h_1^{-1}(L)$  consiste en las palabras de  $L$  con cada símbolo primado o sin primas independientemente. Por consiguiente  $h_1^{-1}(L) \cap (\Sigma'_1)^* R$  consiste en aquellas palabras  $xy$  tales que  $x$  consiste solamente en símbolos con primas y  $y$  se encuentra en  $R$ , y si  $z$  es  $x$  en la que se eliminaron las primas, entonces  $zy$  se encuentra en  $L$ . Se concluye que

$$h_2(h_1^{-1}(L) \cap (\Sigma'_1)^* R)$$

consiste en todas las cadenas  $z$  definidas de la forma descrita arriba, esto es  $L/R$ .  $\square$

**Teorema 11.4** Los trios son cerrados con respecto a la sustitución por conjuntos regulares libres de  $\epsilon$ , y los trios completos son cerrados con respecto a la sustitución por conjuntos regulares.

**Demostración** Sea  $\mathcal{C}$  un trío,  $L \subseteq \Sigma_1^*$  un miembro de  $\mathcal{C}$  y  $s: \Sigma_1^* \rightarrow \Sigma_2^*$  una sustitución tal que para cada  $a$  en  $\Sigma_1^*$ ,  $s(a)$  es regular. Por el momento supondremos que  $\Sigma_1$  y  $\Sigma_2$  son disjuntos, y que  $s(a)$  no contiene a  $\epsilon$ .

Sea  $x$  una cadena de  $L$ . Mediante un homomorfismo inverso podemos insertar cualquier cadena de  $\Sigma_2$  entre los símbolos de  $x$ . A través de la intersección con un conjunto regular podemos asegurar que la cadena insertada después del símbolo  $a$  se encuentra en  $s(a)$ . Entonces mediante un borrado limitado podemos eliminar los símbolos de  $x$ , dejando una cadena de  $s(x)$ .

De manera más precisa sea  $h_1: (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma_1^*$  el homomorfismo definido por  $h_1(a) = a$  para  $a$  en  $\Sigma_1$  y  $h_1(a) = \epsilon$  para  $a$  en  $\Sigma_2$  y sea  $h_2: (\Sigma_1 \cup \Sigma_2)^* \rightarrow \Sigma_2^*$  el homomorfismo definido por  $h_2(a) = \epsilon$  para  $a$  en  $\Sigma_1$  y  $h_2(a) = a$  para  $a$  en  $\Sigma_2$ . Entonces

$$s(L) = h_2(h_1^{-1}(L) \cap \left( \bigcup_{a \in \Sigma_1} as(a) \right)^*)$$

Ahora

$$\left( \bigcup_{a \in \Sigma_1} as(a) \right)^*$$

es un conjunto regular, ya que cada  $s(a)$  lo es. Puesto que  $s(a)$  es libre de  $\epsilon$ ,  $h_2$  borra cuando más cada uno de los otros símbolos, de modo que, según el lema 11.2,  $s(L)$  se encuentra en  $\mathcal{C}$ . La demostración que se refiere a que los trios completos son cerrados con respecto a la sustitución por conjuntos regulares es idéntica a la anterior excepto por el hecho de que  $s$  puede no ser libre de  $\epsilon$ . Si  $\Sigma_1$  y  $\Sigma_2$  no son disjuntos, sustitúyase cada símbolo de  $\Sigma_2$  por un nuevo símbolo y síganse las operaciones anteriores a través de un homomorfismo libre de  $\epsilon$  para restituir los viejos símbolos.  $\square$

### 11.4 FAMILIAS ABSTRACTAS DE LENGUAJES

Muchas de las familias de lenguajes que hemos estudiado poseen propiedades de cerradura que no se infieren de las operaciones de los trios o de los trios completos. Predominan entre éstas la unión, la concatenación y la cerradura de Kleene. Es por esta razón que se han estudiado a fondo las consecuencias de otros dos conjuntos de propiedades de cerradura; y de hecho el estudio se hizo mucho antes que los trios y los trios completos. Definimos una clase de los lenguajes como una *familia abstracta de lenguajes* (AFL) si es un trío completo y es cerrada con respecto a la unión, la concatenación y la cerradura positiva (recuérdese que  $L^+$ , la cerradura positiva de  $L$ , es  $\bigcup_{i=1}^{\infty} L^i$ ). Llámese a una clase de lenguajes un AFL *completo* si es un trío completo y es cerrado con respecto a la unión, la concatenación y la cerradura de Kleene.

Por ejemplo. Demostramos en los Capítulos 3 y 6 que los conjuntos regulares y los lenguajes libres de contexto son AFLs completos. Los conjuntos r.e. también son un AFL completo, y dejamos su demostración con ejercicio. Los CSLs forman un AFL, pero no es completo ya que no son cerrados con respecto a los homomorfismos generales (véase los Ejercicios 9.10 y 9.14).

Vimos que los conjuntos regulares son el trío completo más pequeño. También forman un AFL completo y por lo tanto son el AFL completo más pequeño. Los conjuntos regulares libres de  $\epsilon$  son el AFL más pequeño, así como también el trío más pequeño.

El siguiente teorema establece que los AFLs son cerrados con respecto a la sustitución en conjuntos regulares. Esto es, para cada símbolo de un alfabeto, asociamos un lenguaje de un AFL  $\mathcal{C}$ . Entonces sustituyendo cada símbolo en cada cadena de algún conjunto regular mediante el lenguaje asociado produce un lenguaje en  $\mathcal{C}$ .

**Teorema 11.5** Sea  $\mathcal{C}$  un AFL que contiene algún lenguaje que, a su vez, contiene a  $\epsilon$ , y sea  $R \subseteq \Sigma^*$  un conjunto regular. Hagamos que  $s$  sea una sustitución definida por  $s(a) = L_a$  para cada  $a$  en  $\Sigma$ , en donde  $L_a$  es un miembro de  $\mathcal{C}$ . Entonces  $s(R)$  se encuentra en  $\mathcal{C}$ .

**Demostración** La demostración se lleva a cabo mediante inducción en el número de operadores de una expresión regular que representa a  $R$ . Si existen cero operadores, entonces la expresión regular debe ser una de entre  $\emptyset$ ,  $\epsilon$  o  $a$ , para  $a$  en  $\Sigma$ . Si ésta es  $a$ , entonces el resultado de la sustitución es  $L_a$ , que se encuentra en  $\mathcal{C}$ . Si la expresión regular es  $\emptyset$ , el resultado de la sustitución es  $\emptyset$ , el cual se encuentra en  $\mathcal{C}$  según el lema 11.1. Si la expresión regular es  $\epsilon$ , el resultado de la sustitución es  $\{\epsilon\}$ . Exigimos que

$\{\epsilon\}$  se encuentra en  $\mathcal{C}$  porque alguna  $L$  que contiene  $\epsilon \in$  se encuentra en  $\mathcal{C}$ , y  $L \cap \{\epsilon\} = \{\epsilon\}$  se encuentra en  $\mathcal{C}$  mediante la cerradura con respecto a la intersección con un conjunto regular.

El paso de inducción es sencillo. Los AFLs son cerrados con respecto a la unión y la concatenación; y podemos demostrar fácilmente la cerradura con respecto a  $*$ , dado  $L$  de  $\mathcal{C}$  que contiene  $\epsilon \in$ . Esto es, ya hemos demostrado que  $\{\epsilon\}$  se encuentra en  $\mathcal{C}$ . Si  $L_1$  es cualquier lenguaje de  $\mathcal{C}$ , entonces  $L_1^*$  se encuentra en  $\mathcal{C}$ , de modo que  $L_1^* = L_1^+ \cup \{\epsilon\}$  se encuentra en  $\mathcal{C}$ . Por consiguiente, el AFL  $\mathcal{C}$  es cerrado con respecto a  $\cup$ ,  $\cdot$  y  $*$ , de lo cual se concluye el paso inductivo. Por tanto  $\mathcal{C}$  es cerrado con respecto a la sustitución en un conjunto regular.  $\square$

En general, los AFLs no son cerrados con respecto a la sustitución de lenguajes de la familia en otros lenguajes de la familia, aunque la mayoría de los AFLs más comunes, como los CFLs, los conjuntos recursivos y los conjuntos r.e., lo son. Sin embargo, cualquier AFL cerrado con respecto a  $\cap$  es cerrado con respecto a la sustitución. La demostración es parecida a la del Teorema 11.5 y se deja como ejercicio. También dejamos como ejercicio demostrar el hecho de que todos los AFLs, aún aquellos que no poseen un lenguaje que contenga  $\epsilon \in$ , son cerrados con respecto a la sustitución en los conjuntos regulares libres de  $\epsilon$ .

## 11.5 INDEPENDENCIA DE LAS OPERACIONES AFL

La definición de un AFL requiere de seis propiedades de cerradura. Sin embargo, para demostrar que una familia de lenguajes es un AFL, no se necesitan demostrar las seis propiedades, ya que no son independientes. Por ejemplo, cualquier familia de lenguajes que son cerrados con respecto a  $\cup$ ,  $*$ ,  $h$  libre de  $\epsilon$ ,  $h^{-1}$  y  $\cap R$  es necesariamente cerrada con respecto a  $\cdot$ . De manera similar,  $\cup$  se concluye de las otras cinco operaciones y lo mismo se cumple para  $\cap R$ . Sólo demostraremos la dependencia de  $\cdot$ .

**Teorema 11.6** Cualquier familia de lenguajes cerrados con respecto a  $\cup$ ,  $*$ ,  $h$  libre de  $\epsilon$ ,  $h^{-1}$  y  $\cap R$  es cerrado con respecto a  $\cdot$ .

**Demostración** Sea  $\mathcal{C}$  la familia de lenguajes del teorema, y hagamos que  $L_1 \subseteq \Sigma^*$  y  $L_2 \subseteq \Sigma^*$  se encuentren en  $\mathcal{C}$ . Podemos suponer, sin pérdida de generalidad, que  $\epsilon \in$  no se encuentra en  $L_1$  o en  $L_2$ . Esta suposición queda justificada por el hecho de que

$$L_1 L_2 = (L_1 - \{\epsilon\})(L_2 - \{\epsilon\}) \cup L'_1 \cup L'_2,$$

en donde  $L'_1$  es  $L_1$  si  $\epsilon \in$  se encuentra en  $L_2$  y  $\emptyset$  en cualquier otro caso;  $L'_2$  es  $L_2$  si  $\epsilon \in$  se encuentra en  $L_1$  y  $\emptyset$  en cualquier otro caso. Como  $\mathcal{C}$  es cerrado con respecto a la unión, si podemos demostrar que  $(L_1 - \{\epsilon\})(L_2 - \{\epsilon\})$  se encuentra en  $\mathcal{C}$ , habremos demostrado que  $L_1 L_2$  se encuentra en  $\mathcal{C}$ .

Sean  $a$  y  $b$  símbolos que no se encuentran en  $\Sigma$ . Como  $\mathcal{C}$  es un trío, el Teorema 11.1 nos dice que  $\mathcal{C}$  es cerrado con respecto a las transformaciones GSM libres de  $\epsilon$ . Hagamos que  $M_1$  sea la GSM que imprime a  $a$ , seguida por su primer símbolo de

entrada, y después copia su entrada; sea  $M_2$  otra GSM que imprime a  $b$  con su primer símbolo de entrada, y después copia su entrada. Entonces, como  $\epsilon \in$  no se encuentra en  $L_1$  o en  $L_2$ ,  $M_1(L_1) = aL_1$  y  $M_2(L_2) = bL_2$ , y ambos se encuentran en  $\mathcal{C}$ . Según las cerraduras con respecto a  $\cup$ ,  $*$  y  $\cap R$ ,

$$(aL_1 \cup bL_2)^* \cap a\Sigma^*b\Sigma^* = aL_1 bL_2$$

se encuentra en  $\mathcal{C}$ . Defínase  $g$  como el homomorfismo  $g(a) = g(b) = \epsilon$ , y  $g(c) = c$  para toda  $c$  en  $\Sigma$ . Entonces  $g$  es un homomorfismo de borrado limitado 2, ya que se supone que  $L_1$  y  $L_2$  son libres de  $\epsilon$ . Según el Lema 11.2,  $g(aL_1 bL_2) = L_1 L_2$  se encuentra en  $\mathcal{C}$ .  $\square$

	Trío	Trío completo	AFL	AFL completo
$\cup$			✓	✓
$\cdot$			✓	✓
$+$			✓	✓
$*$				✓
$h^{-1}$	✓	✓	✓	✓
$h$ libre de $\epsilon$	✓	✓	✓	✓
$h$			✓	✓
$\cap R$	✓	✓	✓	✓
Transformaciones GSM libres de $\epsilon$				
Transformaciones GSM	✓	✓	✓	✓
Transformaciones GSM inversas	✓	✓	✓	✓
Borrado limitado	✓	✓	✓	✓
Cociente con un conjunto regular			✓	✓
Inicio			✓	✓
Sustitución en conjuntos regulares			✓	✓
Sustitución por conjuntos regulares libres de $\epsilon$	✓	✓	✓	✓
Sustitución por conjuntos regulares			✓	✓

Fig. 11.13 Resumen de las propiedades de cerradura.

† Utilizamos  $\cap R$  para denotar "intersección con un conjunto regular",  $h$  para "homomorfismo" y  $h^\dagger$  para "homomorfismo inverso". El punto se utiliza para concatenación.

	Conjuntos regulares	CFLs	DCFLs	CSLs	Conjuntos recursivos	Conjuntos r.e.
Complementación	✓		✓	?	: ✓	
Intersección	✓			✓	✓	✓
Sustitución	✓	✓		✓		✓
MÍN	✓		✓	?	✓	✓
MÁX	✓		✓			
CICLO	✓	✓		✓	✓	✓
Inversión	✓	✓		✓	✓	✓

Fig. 11.4 Algunas otras propiedades de cerradura.

Pregunta	Conjuntos regulares	DCFLs	CFLs	CSLs	Conjuntos recursivos	Conjuntos r.e.
¿Está $w$ en $L$ ?	D	D	D	D	D	U
¿Es $L = \emptyset$ ?	D	D	D	U	U	U
¿Es $L = \Sigma^*$ ?	D	D	U	U	U	U
¿Es $L_1 = L_2$ ?	D	?	U	U	U	U
¿Es $L_1 \subseteq L_2$ ?	D	U	U	U	U	U
¿Es $L_1 \cap L_2 = \emptyset$ ?	D	U	U	U	U	U
¿Es $L = R$ ? En donde $R$ es un conjunto regular dado		D	D	U	U	U
¿Es $L$ regular?	T	D	U	U	U	U
¿Es la intersección de dos lenguajes un lenguaje del mismo tipo?		T	U	U	T	T
¿Es el complemento de un lenguaje también un lenguaje del mismo tipo?		T	U	U	?	T
						U

Fig. 11.5 Algunas propiedades de decisión.

## 11.6 RESUMEN

En la Fig. 11.3 listamos algunas de las operaciones bajo las cuales los tríos, los tríos completos, los AFLs y los AFLs completos son cerrados. Todas las propiedades han sido demostradas en este capítulo o son ejercicios. Recuérdese que los conjuntos regulares, los CFLs y los conjuntos r.e. son AFLs completos; los CSLs y los conjuntos recursivos son AFLs. Sin embargo, los DCFLs ni siquiera son tríos.

Algunas otras operaciones no encajan en la teoría de los tríos y los AFLs. En la Fig. 11.4 hacemos un resumen de las propiedades de cerradura de seis clases de lenguajes con respecto a estas operaciones. La cuestión de si los CSLs son cerrados con respecto al complemento es un problema que ha estado abierto durante mucho tiempo y es equivalente a su cerradura con respecto a MIN.

Mientras que este capítulo ha tratado sobre las propiedades de cerradura y no sobre las propiedades de decisión, hemos alcanzado un buen punto para hacer un resumen de tales propiedades para las seis clases de lenguajes mencionadas en la Fig. 11.4. En la Fig. 11.5 mostramos si cada una de diez propiedades importantes es resoluble para las seis clases. D significa resoluble, U irresoluble, T resoluble de manera trivial (debido a que la respuesta es siempre "sí") y ? significa que la respuesta no se conoce. Los resultados que se presentan en la Fig. 11.5 se demostraron en varios capítulos, principalmente en los Capítulos 3, 6, 8 y 10.

## EJERCICIOS

\*S11.1 Demuestre que los lenguajes lineales son un trío completo pero no un AFL.

11.2 Muestre que los conjuntos regulares libres de  $\epsilon$  son un AFL.

11.3 Muestre que un trío completo es cerrado con respecto a INIT, SUB y FIN, en donde

$$\text{SUB}(L) = \{x \mid wxy \text{ no se encuentra en } L \text{ para algunas } w \text{ y } y\},$$

y

$$\text{FIN}(L) = \{x \mid wx \text{ está en } L \text{ para alguna } w\}.$$

11.4 Demuestre que no cada AFL es cerrado con respecto a \*, h, INIT, SUB, FIN, cociente con un conjunto regular o la sustitución por conjuntos regulares.

\*11.5 Muestre que no cada trío completo es cerrado con respecto a  $\cup$ ,  $\cdot$ ,  ${}^*$ ,  ${}^+$  o la sustitución en conjuntos regulares. [Sugerencia: Los lenguajes lineales son suficientes para todas las operaciones, excepto la unión. (Para demostrar que ciertos lenguajes no son lineales utilice el Ejercicio 6.11). Para mostrar la no cerradura con respecto a la unión, encuentre dos tríos completos  $\mathcal{C}_1$  y  $\mathcal{C}_2$  que contengan a los lenguajes  $L_1$  y  $L_2$ , respectivamente, tales que  $L_1 \cup L_2$  no se encuentra ni en  $\mathcal{C}_1$  ni en  $\mathcal{C}_2$ . Muestre que  $\mathcal{C}_1 \cup \mathcal{C}_2$  es también un trío completo].

11.6 Demuestre que cada una de las propiedades de cerradura y no cerradura de la Fig. 11.4 (ya se ha pedido la demostración de algunas en ejercicios anteriores o demostradas en otros teoremas).

\*11.7 La *intercalación* de dos lenguajes  $L_1$  y  $L_2$ , denotada por  $\text{IL}(L_1, L_2)$ , es

$\{w_1x_1w_2x_2\cdots w_kx_k \mid k \text{ es arbitraria}, w_1w_2\cdots w_k \text{ está en } L_1 \text{ y } x_1x_2\cdots x_k \text{ se encuentra en } L_2\}.$ †

Muestre que si  $\mathcal{C}$  es cualquier trío,  $L$  se encuentra en  $\mathcal{C}$ , y si  $R$  es un conjunto regular, entonces  $\text{IL}(L, R)$  se encuentra en  $\mathcal{C}$ .

11.8 ¿Son los conjuntos siguientes cerrados con respecto a  $\text{IL}$ ?

- a) conjuntos regulares    b) CFLs    c) CSLs    d) conjuntos recursivos
- e) conjuntos r.e.

11.9 Un *transductor*  $A$  es una GSM que puede hacer movimientos (producir una salida y cambiar de estado) sobre una entrada  $\epsilon$ . Muestre que cada trío completo es cerrado con respecto a las transducciones  $A$ .

11.10 Encuentre un GSM que transforme  $ai$  en el conjunto  $\{a^ib^k \mid i \leq j + k \leq 2i\}$  para toda  $i$ .

\*11.11 Demuestre que cualquier clase de los lenguajes cerrados con respecto a  $\bullet$ ,  $h, h^{-1}$  y  $\cap R$  es cerrada con respecto a la unión.

\*11.12 Demuestre que cualquier clase de los lenguajes cerrados con respecto a  $h$ ,  $h^{-1}$ ,  $\bullet$  y  $\cup$  es cerrada con respecto a  $\cup R$ .

\*11.13 De ejemplos de clases de lenguajes cerrados con respecto a

- a)  $\cup, \bullet, h$  libre de  $\epsilon, h^{-1}$  y  $\cap R$ , pero no con respecto a  $\bullet$ ;
- b)  $\cup, \bullet, +, h$  libre de  $\epsilon$  y  $\cap R$ , pero no con respecto a  $h^{-1}$ ;
- c)  $\cup, \bullet, +, h^{-1}$  y  $\cap R$ , pero no con respecto a  $h$  libre de  $\epsilon$ .

\*11.14 Muestre que un AFL es cerrado con respecto a la complementación si y sólo si es cerrado con respecto a MIN.

\*11.15 Una *gramática de contexto disperso*,  $G = (V, T, P, S)$  tiene producciones de la forma  $(A_1, \dots, A_n) \rightarrow (\alpha_1, \dots, \alpha_n)$ , en donde cada  $\alpha_i$  se encuentra en  $(V \cup T)^*$ . Si  $(A_1, \dots, A_n) \rightarrow (\alpha_1, \dots, \alpha_n)$  se encuentra en  $P$ , entonces escribimos

$$\beta_1 A_1 \beta_2 A_2 \cdots \beta_n A_n \beta_{n+1} \Rightarrow \beta_1 \alpha_1 \beta_2 \alpha_2 \cdots \beta_n \alpha_n \beta_{n+1}.$$

Sea  $\stackrel{*}{\Rightarrow}$  la cerradura reflexiva y transitiva de  $\Rightarrow$ . El lenguaje generado por  $G$  es  $\{x \mid x \text{ está en } T^* \text{ y } S \stackrel{*}{\Rightarrow} x\}$ .

a) Demuestra que los lenguajes de contexto disperso forman un AFL.

b) ¿Qué clase de lenguajes es generada por las gramáticas de contexto disperso si permitimos las producciones en las que las  $ai$  sean posiblemente  $\epsilon$ ?

\*\*11.16 Se dice que un AFL  $\mathcal{C}$  es *principal* si existe un lenguaje  $L$  tal que  $\mathcal{C}$  es el menor AFL que contiene a  $L$ .

a) ¿Forman los CFLs un AFL principal?

† Nótese que algunas ws y xs pueden ser  $\epsilon$ .

b) Demuestre que el menor AFL que contiene a  $\{a^n b^n \mid n \geq 0\}$  está contenido propiamente en los CFLs.

c) Sea  $\mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2, \dots$  una secuencia infinita de AFLs tales que  $\mathcal{C}_{i+1} \subseteq \mathcal{C}_i$  para toda  $i > 0$ . Demuestre que la unión de las  $\mathcal{C}_i$ s forman un AFL que no es principal.

d) Dé un ejemplo de un AFL no principal.

\*11.17 Muestre que si un AFL es cerrado con respecto a la intersección, entonces es cerrado con respecto a la sustitución.

### Soluciones a los ejercicios seleccionados

11.1 Para demostrar que los lenguajes lineales son cerrados con respecto al homomorfismo, hagamos que  $G$  sea una gramática no lineal y  $h$  un homomorfismo. Si cada producción  $A \rightarrow wBx$  o  $A \rightarrow y$  es sustituida por  $A \rightarrow h(w)Bh(x)$  o  $A \rightarrow h(y)$ , respectivamente, entonces la gramática resultante genera a  $h(L(G))$ . Para demostrar la cerradura con respecto  $h^{-1}$  y  $\cap R$ , podríamos utilizar demostraciones basadas en máquinas parecidas a las demostraciones para los CFLs, puesto que, según el resultado del Ejercicio 6.13(a), los lenguajes lineales son caracterizados por PDAs de una vuelta. En lugar de esto daremos demostraciones basadas en gramáticas

Sean  $G = (V, T, P, S)$  una CFG lineal y  $M = (Q, T, \delta, q_0, F)$  un DFA. Constrúyase la gramática lineal  $G' = (V', T, P', S')$  que genere a  $L(G) \cap L(M)$ . Hagamos  $V' = \{[qAp] \mid q \text{ y } p \text{ se encuentran en } Q \text{ y } A \text{ en } V\} \cup \{S'\}$ . Defínase, entonces,  $P'$  con las producciones

1.  $S' \rightarrow [q_0Sp]$  para toda  $p$  en  $F$ ,
2.  $[qAp] \rightarrow w[rBs]x$  siempre que  $A \rightarrow wBx$  se encuentre en  $P$ ,  $\delta(q, w) = r$  y  $\delta(s, x) = p$  y
3.  $[qAp] \rightarrow y$  siempre que  $A \rightarrow y$  se encuentre en  $P$  y  $\delta(q, y) = p$ .

Una inducción sencilla sobre la longitud de la derivación muestra que  $[qAp] \xrightarrow{*} w$  si y sólo si  $A \xrightarrow{*} w$  y  $\delta(q, w) = p$ . Por consiguiente  $S' \xrightarrow{*} w$  si y sólo si  $S \xrightarrow{*} w$  y  $\delta(q_0, w)$  es un estado final. Por tanto  $L(G') = L(G) \cap L(M)$ .

Hagamos, ahora,  $G = (V, T, P, S)$  que sea una gramática lineal y  $h: \Sigma^* \rightarrow T^*$  un homomorfismo. Supóngase que  $k$  es tal que para toda  $a$  en  $\Sigma$ ,  $|h(a)| \leq k$  y si  $A \rightarrow wBx$  o  $A \rightarrow w$  está en  $P$ , entonces  $|w| \leq k$  y  $|x| \leq k$ . Sea  $G'' = (V'', \Sigma, P'', [S])$ , en donde  $V''$  consiste en todos los símbolos  $[wAx]$  tales que  $A$  está en  $V$  y  $w$  y  $x$  se encuentran en  $T^*$  y cada uno es de longitud de cuando más  $2k - 1$ . También se encuentran en  $V''$  los símbolos  $[y]$ , en donde  $|y| \leq 3k - 1$ . De manera intuitiva  $G''$  simula una derivación de  $G$  en su variable hasta que la cadena de terminales que se encuentra a la derecha o la izquierda de la variable de  $G$  sea de longitud de al menos  $k$ . Entonces  $G''$  produce una terminal  $a$  a la derecha o la izquierda y elimina  $h(a)$  del lugar en donde se encontraba almacenado en la variable.

Las producciones de  $P''$  son:

1. Si  $A \rightarrow w_1Bx_1$  se encuentra en  $P$ , entonces, para toda  $w_2$  y  $x_2$  de longitud cuando más de  $k - 1$ ,  $[w_2w_1Ax_1] \rightarrow [w_2w_1Bx_1x_2]$  se encuentra en  $P''$ . Si  $A \rightarrow y$  se encuentra en  $P$ , entonces  $[w_2Ax_2] \rightarrow [w_2yx_2]$  está en  $P''$ .
  2. para  $a$  en  $\Sigma$ ,
- $$[h(a)w_1Ax_1] \rightarrow a[w_1Ax_1], \quad [w_1Ax_1h(a)] \rightarrow [w_1Ax_1]a, \quad y \quad [h(a)y] \rightarrow a[y].$$
3.  $[\epsilon] \rightarrow \epsilon$ .

Mediante inducción sobre la longitud de la derivación se concluye que

$$[S] \xrightarrow[G]{*} w_1[w_2 Ax_2]x_1$$

si y sólo si

$$S \xrightarrow[G]{*} h(w_1)w_2 Ax_2 h(x_1).$$

Por consiguiente  $[S] \xrightarrow[G]{*} v$  si y sólo si  $S \xrightarrow[G]{*} h(v)$ , y por tanto  $L(G') = h^{-1}(L(G))$ .

Para demostrar que los lenguajes lineales no son AFL, demostramos que no son cerrados con respecto a la concatenación. Es seguro que  $\{ab^i | i \geq 1\}$  y  $\{c^j d^j | j \geq 1\}$  son lenguajes lineales, pero su concatenación no lo es, según el Ejercicio 6.12.

## NOTAS BIBLIOGRAFICAS

El estudio de familias abstractas de lenguajes fue iniciado por Ginsburg y Greibach [1969], quienes demostraron los teoremas 11.1 a 11.5 y el Lema 11.1. La importancia central del trío en esta teoría la hizo notar Ginsburg [1975]. El Teorema 11.6 sobre la independencia de los operadores aparece en Greibach y Hopcroft [1969]; aquí también se puede encontrar una solución al Ejercicio 11.13. El concepto de borrado limitado se debe también a Greibach y Hopcroft [1969]. El hecho de que los AFLs cerrados con respecto a la intersección son cerrados con respecto a la sustitución fue demostrado por primera vez por Ginsburg y Hopcroft [1970]. Existe una enorme cantidad de literatura concerniente a las familias abstractas de lenguajes; mencionamos solamente el trabajo de Ginsburg y Greibach [1970], que trata sobre los AFLs principales (Ejercicios 11.16) y el de Greibach [1970], quien intenta trabajar la sustitución dentro de la teoría. Un resumen y referencias adicionales se pueden encontrar en Ginsburg [1975].

Desde su origen, la teoría de las familias de lenguajes ha sido conectada con la teoría de los autómatas. Ginsburg y Greibach [1969] demostraron que una familia de lenguajes es un AFL completo si y sólo si está definido por una familia de autómatas no determinística con una entrada en una sola dirección. Por supuesto que el concepto de una "familia de autómatas" debe definirse de manera apropiada, pero, aproximadamente, cada una de estas familias está caracterizada por un conjunto de reglas mediante las cuales puede accesar o actualizar su almacenamiento. La parte "si" fue demostrada en forma independiente por Hopcroft y Ullman [1967b]. Chandler [1969] caracterizó a las familias de autómatas determinísticos con una entrada de una dirección en términos de las propiedades de cerradura, y Aho y Ullman [1970] hicieron lo mismo para los autómatas determinísticos con una entrada de dos direcciones. Curiosamente no se conoce ninguna caracterización para los autómatas no determinísticos de dos direcciones.

También se han hecho intentos para codificar una teoría de las gramáticas, principalmente las subfamilias de las CFGs. Gabriellian y Ginsburg [1974] y Cremers y Ginsburg [1975] escribieron los artículos básicos en esta área.

La GSM fue definida por Ginsburg [1962], y el estudio de las transformaciones GSM y sus propiedades se inició con Ginsburg y Rose [1963b]. Una importante cuestión, aún no resuelta, se refiere a la prueba de la equivalencia de dos transductores secuenciales. El hecho de que la equivalencia es resoluble para las máquinas de Moore (y por tanto para las máquinas de Mealy, las cuales son generalizadas por las GSMS) era conocido desde Moore [1956]. Griffiths [1968] demostró que el problema de la equivalencia para las GSMS libres de  $\epsilon$ . es irresoluble, mientras

que Bird [1973] dio un algoritmo de decisión para la equivalencia de los autómatas de dos cintas, que son más generales que las GSMS determinísticas.

Las gramáticas de contexto disperso (Ejercicio 11.15) se discuten en Greibach y Hopcroft [1969].



# CAPITULO

# 12

## TEORIA DE COMPLEJIDAD COMPUTACIONAL

La teoría de los lenguajes clasifica a los conjuntos según su complejidad estructural. Por tanto los conjuntos regulares se consideran como “más simples que los CFLs, porque el autómata finito tiene una estructura menos compleja que un PDA. Otra clasificación, conocida como complejidad computacional, está basada en la cantidad de tiempo, espacio u otro recurso que se necesita para reconocer a un lenguaje en algún dispositivo de cómputo universal, como la máquina de Turing.

Aunque la complejidad computacional se refiere principalmente al tiempo y al espacio, existen muchas otras medidas posibles, como el número de inversiones de la dirección de movimiento de la cabeza de la cinta en una TM de una sola cinta. De hecho, uno puede definir una medida de la complejidad de manera abstracta y demostrar muchos de los resultados en un marco más general. Decidimos presentar los resultados para los ejemplos específicos del tiempo y el espacio, ya que este planteamiento presenta las demostraciones de manera más intuitiva. En la Sección 12.7 hacemos un breve esbozo de un planteamiento más abstracto.

### 12.1 DEFINICIONES

#### Complejidad espacial

Considérese la máquina de Turing fuera de línea  $M$  de la Fig 12.1.  $M$  tiene una cinta de entrada de lectura solamente con señaladores de extremo y cintas de almacenamiento semiinfinitas  $k$ . Si, para cada palabra de entrada de longitud  $n$ ,  $M$  barre cuando más  $S(n)$  celdas en cualquier cinta de almacenamiento, entonces se dice que  $M$  es una *máquina de Turing con límite espacial  $S(n)$*  o de *complejidad espacial  $S(n)$* . El lenguaje reconocido por  $M$  también se dice que es de complejidad espacial  $S(n)$ .

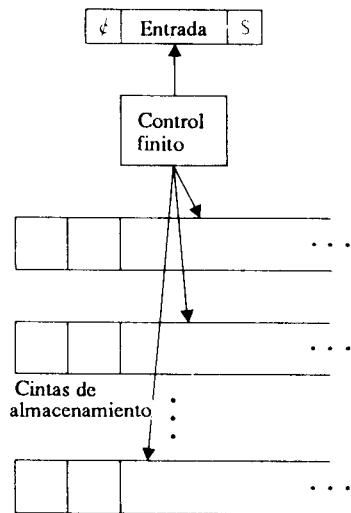


Fig. 12.1 Máquina de Turing multicintas con entrada de lectura solamente.

Nótese que la máquina de Turing no puede reescribir sobre la entrada y que solamente la longitud de las cintas de almacenamiento utilizadas cuenta en el cálculo del límite de la cinta. Esta restricción nos permite considerar límites de cinta con un crecimiento menor al lineal. Si la TM pudiera reescribir sobre la cinta de entrada, entonces la longitud de la entrada tendría que incluirse al calcular el límite espacial. Por consiguiente ningún límite espacial podría ser menor que el lineal.

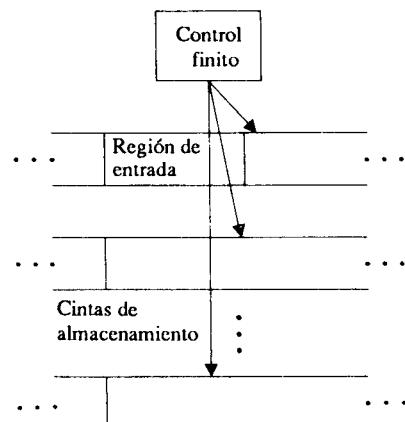


Fig. 12.2 Máquina de Turing multicintas.

### Complejidad temporal

Considérese la TM multicintas  $M$  de la Fig. 12.2. La TM tiene  $k$  cintas infinitas de dos direcciones, una de las cuales contiene a la cinta de entrada. En todas las cintas, incluyendo a la cinta de entrada, se puede escribir. Si, para cada palabra de entrada de longitud  $n$ ,  $M$  hace cuando más  $T(n)$  movimientos antes de detenerse, entonces decimos que  $M$  es una *máquina de Turing con límite temporal  $T(n)$*  o de *complejidad temporal  $T(n)$* . El lenguaje reconocido por  $M$  se dice que es de complejidad temporal  $T(n)$ .

Los dos modelos diferentes para la complejidad espacial y temporal fueron seleccionados con el propósito de hacer simples ciertas demostraciones, además de que es posible llevar a cabo algunas variaciones en los modelos. Por ejemplo, si  $S(n) \geq n$ , entonces podemos utilizar la TM de una sola cinta como nuestro modelo, sin cambiar la clase de los lenguajes aceptada en el espacio  $S(n)$ . No podemos, sin embargo, cuando se está discutiendo la complejidad temporal, utilizar la TM de una sola cinta, o utilizar TMs con cualquier número fijo de cintas, sin posiblemente perder algunos lenguajes de los lenguajes aceptados por el tiempo  $T(n)$ .

### Ejemplo 12.1 Considérese el lenguaje

$$L = \{ w c w^R \mid w \text{ está en } (0+1)^* \}.$$

El lenguaje  $L$  es de complejidad temporal  $n+1$ , ya que existe una máquina de Turing  $M_1$ , con dos cintas, que copia la entrada que se encuentra a la izquierda de  $c$  en la segunda cinta. Después, cuando se ha encontrado una  $c$ ,  $M_1$  mueve su segunda cabeza de cinta hacia la izquierda, a través de la cadena que acaba de ser copiada y, de manera simultánea, continúa moviendo su cabeza de cinta de entrada hacia la derecha. Los símbolos que se encuentran bajo las dos cabezas son comparados conforme las cabezas se mueven. Si todas las parejas de símbolos coinciden y si, además, el número de símbolos que se encuentran a la derecha e izquierda de  $c$  son el mismo, entonces  $M_1$  acepta. Resulta sencillo ver que  $M_1$  hace cuando mucho  $n+1$  movimientos si la entrada es de longitud  $n$ .

Existe otra máquina de Turing,  $M_2$ , de complejidad espacial  $\log_2 n$  que acepta a  $L$ .  $M_2$  utiliza dos cintas de almacenamiento como contadores binarios. Primero, la entrada es verificada para cerciorarse de que solamente una  $c$  aparece, y de que hay un número igual de símbolos a la derecha e izquierda de  $c$ . En seguida se comparan las palabras de la derecha con las de la izquierda, símbolo por símbolo, utilizando los contadores para encontrar símbolos correspondientes. Si éstos no concuerdan,  $M_2$  se detiene sin aceptar. Si todos los símbolos corresponden,  $M_2$  acepta.

### Suposiciones especiales acerca de las funciones de complejidad espacial y temporal

Debería resultar obvio que cada TM utiliza al menos una celda en todas las entradas, de modo que si  $S(n)$  es una medida de la complejidad espacial, podemos suponer que  $S(n) \geq 1$  para toda  $n$ . Planteamos la útil hipótesis de que cuando hablamos de “complejidad espacial  $S(n)$ ”, lo que realmente queremos decir es  $\max(1, \lceil S(n) \rceil)$ . Por ejemplo,

en el Ejemplo 12.1 decimos que la TM  $M_2$  es de “complejidad espacial  $\log_2 n$ ”. Esto no tiene significado para  $n = 0$  o 1, a menos que se acepte que “ $\log_2 n$ ” es una abreviatura de  $\max(1, \lceil \log_2 n \rceil)$ .

De manera similar, es razonable suponer que cualquier función de la complejidad temporal  $T(n)$  es al menos  $n + 1$ , porque éste es el tiempo necesario para leer justamente la entrada y verificar que el extremo ha sido alcanzado mediante la lectura del primer espacio en blanco.<sup>†</sup> Por tanto, establecemos la norma de que “complejidad temporal  $T(n)$ ” significa  $\max(n + 1, \lceil T(n) \rceil)$ . Por ejemplo, el valor de la complejidad temporal  $n \log_2 n$  en  $n = 1$  es 2, no 0, y en  $n = 2$ , su valor es 3.

### Complejidades temporal y espacial no determinísticas

Los conceptos de máquinas de Turing con límites temporal y espacial se aplican también a las máquinas no determinísticas. Una TM no determinística es de complejidad temporal  $T(n)$  sin ninguna secuencia de alternativas de movimientos ocasiona que la máquina haga más de  $T(n)$  movimientos. Es de complejidad espacial  $S(n)$  si ninguna secuencia de movimientos le permite barrer más de  $S(n)$  celdas sobre cualquier cinta de almacenamiento.

### Clases de complejidad

La familia de lenguajes de complejidad espacial  $S(n)$  se denota por ESPACIOD( $S(n)$ ); los lenguajes de complejidad espacial no determinística  $S(n)$  se llaman de manera colectiva ESPACION( $S(n)$ ). La familia de lenguajes de complejidad temporal  $T(n)$  se representa por TIEMPOD( $T(n)$ ) y la familia de lenguajes de complejidad temporal no determinística  $T(n)$  se denota como TIEMPON( $T(n)$ ). Todas estas familias de lenguajes se conocen como *clases de complejidad*. Por ejemplo, el lenguaje  $L$  del Ejemplo 12.1 se encuentra en TIEMPOD( $n$ )<sup>††</sup> y en ESPACIOD( $\log_2 n$ ). Por consiguiente  $L$  también se encuentra en TIEMPON( $n$ ) y ESPACION( $\log_2 n$ ), tanto como en clases más grandes como TIEMPOD( $n^2$ ) o ESPACION( $\sqrt{n}$ ).

## 12.2 ACCELERACION LINEAL, COMPRESION DE CINTA Y REDUCCIONES EN EL NUMERO DE CINTAS

Puesto que el número de estados y el tamaño del alfabeto de cinta de una máquina de Turing pueden ser arbitrariamente grandes, la cantidad de espacio que se necesita para reconocer a un conjunto siempre se puede comprimir mediante un factor constante. Esto se hace posible codificando varios símbolos de cinta en uno. De manera parecida, uno puede acelerar un cálculo mediante un factor constante. Por consiguiente en términos de complejidad se trata de la rapidez funcional de crecimiento (por ejemplo, lineal, cuadrática, exponencial), esto es, factores constantes importantes pueden ser

despreciados. Por ejemplo, hablaremos de complejidad  $\log n$  sin especificar la base de los logaritmos, ya que  $\log_b n$  y  $\log_c n$  difieren por un factor constante, a saber,  $\log_c b$ . En esta sección estableceremos las características básicas concernientes a la aceleración lineal y a la compresión, y también consideraremos el número de cintas en la complejidad.

### Compresión de cinta

**Teorema 12.1** Si  $L$  es aceptado por una máquina de Turing con límite espacial  $S(n)$  y  $k$  cintas de almacenamiento, entonces, para cada  $c > 0$ ,  $L$  es aceptado por una TM con límite espacial  $cS(n)$ .<sup>†</sup>

*Demostración* Sea  $M_1$  una máquina de Turing fuera de línea y con límite espacial  $S(n)$  que acepta a  $L$ . La demostración consiste en construir una nueva máquina de Turing  $M_2$  que simula a  $M_1$ , en donde, para alguna constante  $r$ , cada celda de cinta de almacenamiento de  $M_2$  contiene un símbolo que representa el contenido de  $r$  celdas adyacentes de la correspondiente cinta de  $M_1$ . El control finito de  $M_2$  puede mantener el rastro de cuál de las celdas de  $M_1$ , entre las representadas, está siendo barrida por  $M_1$ .

La construcción detallada de las reglas de  $M_2$  a partir de las reglas de  $M_1$  se deja al lector. Sea  $r$  una constante tal que  $rc \geq 2$ .  $M_2$  puede simular a  $M_1$  sin utilizar más de  $\lceil S(n)/r \rceil$  celdas en cualquier cinta. Si  $(n) \geq r$ , este número no es mayor que  $cS(n)$ . Si  $S(n) < r$ , entonces  $M_2$  puede almacenar en una celda el contenido de cualquier cinta. Por consiguiente  $M_2$ , en el último caso, utiliza solamente una celda. □

**Corolario** Si  $L$  se encuentra en ESPACION( $S(n)$ ), entonces  $L$  se encuentra en ESPACION( $cS(n)$ ), en donde  $c$  es cualquier constante mayor que cero.

*Demostración* Si la  $M_1$  descrita arriba es no determinística, hagamos que  $M_2$  sea no determinístico en la construcción anterior. □

### Reducción del número de cintas para las clases de complejidad espacial

**Teorema 12.2** Si un lenguaje es aceptado por una TM con límite espacial  $S(n)$  y  $k$  cintas de almacenamiento, entonces es aceptado por una TM con límite espacial  $S(n)$  y una sola cinta de almacenamiento.

*Demostración* Sea  $M_1$  una TM con límite espacial  $S(n)$  y  $k$  cintas de almacenamiento que acepta a  $L$ . Podemos construir una nueva TM  $M_2$ , con una sola cinta de almacenamiento que simula las cintas de almacenamiento de  $M_1$  en  $k$  registros. La técnica se utilizó en el Teorema 7.2.  $M_2$  no utiliza más de  $S(n)$  celdas. □

De aquí en adelante supondremos que cualquier TM con límite espacial  $S(n)$  tiene sólo una cinta de almacenamiento, y si  $S(n) \geq n$ , entonces se trata de una TM de una sola cinta, más que de una TM fuera de línea con una cinta de almacenamiento y una cinta de entrada.

<sup>†</sup> Nótese que, sin embargo, existen TMs que aceptan o rechazan sin leer toda su entrada. Decidimos eliminarlas de nuestra consideración.

<sup>††</sup> Recuérdese que  $n$  realmente quiere decir  $\max(n + 1, n) = n + 1$  para la complejidad temporal.

## Aceleración lineal

Antes de considerar los límites temporales introduciremos la siguiente notación. Sea  $f(n)$  una función de  $n$ . La expresión  $\sup_{n \rightarrow \infty} f(n)$  se considera como el límite cuando  $n \rightarrow \infty$  del menor límite superior de  $f(n)$ ,  $f(n+1)$ ,  $f(n+2)$ , ... De la misma manera,  $\inf_{n \rightarrow \infty} f(n)$  es el límite cuando  $n \rightarrow \infty$  del mayor límite inferior de  $f(n)$ ,  $f(n+1)$ ,  $f(n+2)$ , ... Si  $f(n)$ , converge a un límite cuando  $n \rightarrow \infty$ , entonces dicho límite es tanto  $\inf_{n \rightarrow \infty} f(n)$  como  $\sup_{n \rightarrow \infty} f(n)$ .

**Ejemplo 12.2** Sea  $f(n) = 1/n$  para  $n$  par y  $f(n) = n$  para  $n$  impar. El menor límite superior de  $f(n)$ ,  $f(n+1)$ , ... es claramente  $\infty$ , para cualquier  $n$ , debido a que los términos para  $n$  impar. De aquí que  $\sup_{n \rightarrow \infty} f(n) = \infty$ . Sin embargo, debido a los términos para  $n$  par también es cierto que  $\inf_{n \rightarrow \infty} f(n) = 0$ .

Como otro ejemplo, supóngase que  $f(n) = n/(n+1)$ . Entonces el menor límite superior de  $n/(n+1)$ ,  $(n+1)/(n+2)$ , ... es 1 para cualquier  $n$ . Por tanto

$$\sup_{n \rightarrow \infty} \frac{n}{n+1} = 1.$$

El mayor límite inferior de  $n/(n+1)$ ,  $(n+1)/(n+2)$ , ... es  $n/(n+1)$  y  $\lim_{n \rightarrow \infty} n/(n+1) = 1$ , de modo que  $\inf_{n \rightarrow \infty} n/(n+1) = 1$ , también.

**Teorema 12.3** Si  $L$  es aceptado por una máquina de Turing con límite temporal  $T(n)$  y  $k$  cintas,  $M_1$ , entonces  $L$  es aceptado por una máquina de Turing con límite temporal  $cT(n)$  y  $k$  cintas,  $M_2$ , para cualquier  $c > 0$ , siempre que  $k > 1$  y si  $\inf_{n \rightarrow \infty} T(n)/n = \infty$ .

**Demuestra** Se puede construir una TM,  $M_2$  para simular a  $M_1$  de la manera siguiente. Primero  $M_2$  copia la entrada en una página de almacenamiento, codificando  $m$  símbolos en uno. (El valor de  $m$  será determinado más adelante.) A partir de este punto,  $M_2$  utiliza su cinta de almacenamiento como cinta de entrada y usa la vieja entrada como cinta de almacenamiento.  $M_2$  codificará el contenido de las cintas de almacenamiento de  $M_1$  mediante la combinación de  $m$  símbolos en uno. Durante el curso de la simulación,  $M_2$  simula un gran número de movimientos de  $M_1$  en un *paso básico* que consiste en ocho movimientos de  $M_2$ . Llámese a las celdas barridas corrientemente por cada una de las cabezas de  $M_2$  *celdas destino*. El control finito de  $M_2$  registra, para cada cinta, cual de los  $m$  símbolos de  $M_1$ , representado por cada celda destino, es barrido por la correspondiente cabeza de  $M_2$ .

Para comenzar un paso básico,  $M_2$  mueve cada cabeza hacia la izquierda una vez, dos veces a la derecha y de nuevo una sola vez hacia la izquierda, registrando los símbolos que están a la izquierda y a la derecha de las celdas destino en su control finito. Se requieren cuatro movimientos de  $M_2$ , después de los cuales  $M_2$  ha regresado a sus celdas destino.

En seguida,  $M_2$  determina el contenido de todas las celdas de las cintas de  $M_1$  representadas por las celdas destino y sus celdas adyacentes de la izquierda y la derecha, en el momento en que alguna cabeza de cinta  $M_1$  abandona por primera vez la región representada por las celdas destino y las celdas adyacentes de la izquierda y

la derecha. ( Nótese que este cálculo, hecho por  $M_2$ , no toma tiempo. Se construye dentro de las reglas de transición de  $M_2$ ). Si  $M_1$  acepta antes de que alguna de sus cabezas de cinta abandone la región representada,  $M_2$  acepta. Si  $M_1$  se detiene,  $M_2$  lo hace también. En cualquier otro caso,  $M_2$  visita entonces, sobre cada cinta, a las dos celdas vecinas de la celda destino, cambiando los símbolos que se encuentran en éstas y los de la destino, si fuera necesario.  $M_2$  coloca cada una de sus cabezas en la celda que representa al símbolo que la correspondiente cabeza de  $M_1$  se encuentra barriendo al final de los movimientos simulados. Cuando mucho se necesitan cuatro movimientos de  $M_2$ .

Le toma a  $M_1$  al menos  $m$  movimientos para trasladar una cabeza fuera de la región representada por una celda destino y sus celdas adyacentes. Por consiguiente, en ocho movimientos,  $M_2$  ha simulado al menos  $m$  movimientos de  $M_1$ . Escojase  $m$  tal que  $cm \geq 16$ .

Si  $M_1$  lleva a cabo  $T(n)$  movimientos, entonces  $M_2$  simula a éstos en, cuando mucho,  $8\lceil T(n)/m \rceil$  movimientos. También,  $M_2$  debe copiar y codificar su entrada ( $m$  celdas en una), entonces regresa la cabeza de la cinta de entrada simulada al extremo izquierdo. Esto toman  $\lceil n/m \rceil$  movimientos, para hacer un total de

$$n + \lceil n/m \rceil + 8\lceil T(n)/m \rceil \quad (12.1)$$

movimientos. Como  $\lceil x \rceil < x + 1$  para cualquier  $x$ , (12.1) tiene como límite superior

$$n + n/m + 8T(n)/m + 2. \quad (12.2)$$

Ahora hemos supuesto que  $\inf_{n \rightarrow \infty} T(n)/n = \infty$ , de modo que para cada constante  $d$  existe una  $n_d$  tal que, para toda  $n \geq n_d$ ,  $T(n)/n \geq d$ , o dicho de otra manera,  $n \leq T(n)/d$ . Por consiguiente siempre que  $n \geq 2$  (de modo que  $n+2 \leq 2n$ ) y  $n \geq n_d$ , (12.2) tendrá como límite superior a

$$T(n) \left[ \frac{8}{m} + \frac{2}{d} + \frac{1}{md} \right]. \quad (12.3)$$

Aún no hemos especificado a  $d$ . Recordando que  $m$  fue escogida de modo que  $cm \geq 16$ , escójase  $d = m/4 + \frac{1}{8}$  y sustitúyase, en (12.3),  $16/c$  por  $m$ . Entonces, para toda  $n \geq \max(2, n_d)$  el número de movimientos hechos por  $M_2$  no excede a  $cT(n)$ .

Para reconocer el número finito de palabras de longitud menor que el máximo de  $2 + n_d$ ,  $M_2$  solamente utiliza su control finito, tomando  $n+1$  movimientos para leer su entrada y alcanzar el espacio en blanco que señala el final de la entrada. Por consiguiente, la complejidad temporal de  $M_2$  es  $cT(n)$ . Recuérdese que para la complejidad temporal,  $cT(n)$  quiere decir  $\max(n+1, \lceil cT(n) \rceil)$ . □

**Corolario** Si  $\inf_{n \rightarrow \infty} T(n)/n = \infty$  y  $c > 0$ , entonces

$$\text{TIEMPOD}(T(n)) = \text{TIEMPOD}(cT(n)).$$

**Demuestra** El Teorema 12.3 representa una demostración directa para cualquier lenguaje aceptado por una DTM con 2 o más cintas en tiempo  $T(n)$ . Es claro que si  $L$  es aceptado por una TM de una cinta, es aceptado por una TM de dos cintas de la misma complejidad temporal. □

El Teorema 12.3 no se aplica si  $T(n)$  es una constante múltiplo de  $n$ , porque en este caso  $\inf_{n \rightarrow \infty} T(n)/n$  es una constante, no el infinito. Sin embargo, la construcción del teorema 12.3, con un análisis más cuidadoso del límite temporal de  $M_2$ , muestra el siguiente resultado.

**Teorema 12.4** Si  $L$  es aceptado por una TM con límite temporal  $cn$  y  $k$  cintas, para  $k > 1$  y para alguna constante  $c$ , entonces para cada  $\epsilon > 0$ ,  $L$  es aceptado por una TM con límite temporal  $((1 + \epsilon)n)$  y  $k$  cintas.

*Demostración* Tómese  $m = 1/16 \epsilon$  en la demostración del Teorema 12.3.  $\square$

**Corolario** Si  $T(n) = cn$  para alguna  $c > 1$ , entonces  $\text{TIEMPON}(T(n)) = \text{TIEMPON}((1 + \epsilon)n)$ , para cualquier  $\epsilon > 0$ .

**Corolario** (de los teoremas 12.3 y 12.4)

- Si  $\inf_{n \rightarrow \infty} T(n)/n = \infty$ , entonces  $\text{TIEMPON}(T(n)) = \text{TIEMPON}(cT(n))$ , para cualquier  $c > 0$ .
- Si  $T(n) = cn$ , para alguna constante  $c$ , entonces  $\text{TIEMPON}(T(n)) = \text{TIEMPON}((1 + \epsilon)n)$ , para cualquier  $\epsilon > 0$ .

*Demostración* Las demostraciones son parecidas a las de los Teoremas 12.3 y 12.4.  $\square$

### Reducción del número de cintas para las clases de complejidad temporal

Veamos ahora qué es lo que le sucede a la complejidad temporal cuando nos restringimos a una cinta. Un lenguaje como  $L = \{wcw^R/w \in (a+b)^*\}$  puede ser reconocido por un tiempo lineal por una máquina de dos cintas, como lo vimos en el Ejemplo 12.1. Sin embargo, en una máquina de una sola cinta,  $L$  requiere un tiempo  $cn^2$  para alguna  $c > 0$ . (Los ejercicios dan sugerencias de cómo se puede llevar a cabo la demostración). Ello permite que sólo una cinta pueda elevar al cuadrado el tiempo necesario para reconocer un lenguaje. Que esto es lo peor que puede suceder se expresa en el teorema siguiente.

**Teorema 12.5** Si  $L$  está en  $\text{TIEMPON}(T(n))$ , entonces  $L$  es aceptado en un tiempo  $T^2(n)$  por una TM de una sola cinta.

*Demostración* En la construcción del Teorema 7.2, pasando de una TM multicintas a una TM de una sola cinta,  $M_2$  utiliza cuando mucho  $6T^2(n)$  pasos para simular  $T(n)$  pasos de  $M_1$ . Según el teorema 12.3, podemos acelerar a  $M_1$  para que corra en un tiempo  $T(n)/\sqrt{6}$ . Entonces  $M_2$  es una TM de una sola cinta que acepta a  $L$  en  $T^2(n)$  pasos.  $\square$

**Corolario** Si  $L$  está en  $\text{TIEMPON}(T(n))$ , entonces  $L$  es aceptado por una NTM de una sola cinta de complejidad temporal no determinística  $T^2(n)$ .

*Demostración* La demostración es parecida a la del teorema.  $\square$

Si nos restringimos a dos cintas, la pérdida de tiempo es considerablemente menor que si nos restringimos a una sola cinta, hecho que se demuestra mediante el siguiente teorema.

**Teorema 12.6** Si  $L$  es aceptado por una máquina de Turing con límite temporal  $T(n)$  y  $k$  cintas,  $M_1$ , entonces  $L$  es aceptado por una TM de dos cintas de almacenamiento,  $M_2$  en un tiempo  $T(n) \log T(n)$ .

*Demostración* La primera cinta de almacenamiento de  $M_2$  tendrá dos registros para cada cinta de almacenamiento de  $M_1$ . Por conveniencia, enfocaremos nuestra atención en los dos registros correspondientes a una cinta en particular de  $M_1$ . Las otras cintas de  $M_1$  son simuladas exactamente de la misma forma. La segunda cinta se utiliza solamente para tachar y transportar bloques de datos en la cinta 1.

Una cinta particular de  $M_1$ , conocida como  $B_0$ , contendrá a los símbolos de almacenamiento barridos por cada una de las cabezas de  $M_1$ . Más que desplazar señaladores de cabeza,  $M_2$  transportará datos a través de  $B_0$  en dirección opuesta a la del movimiento de la cabeza de  $M_1$  que está siendo simulado. Por consiguiente  $M_2$  puede simular cada movimiento de  $M_1$  mirando solamente  $B_0$ . Hacia la derecha de la celda  $B_0$  habrá bloques  $B_1, B_2, \dots$  de longitud que crece de manera exponencial; esto es,  $B_i$  es de longitud  $2^{i-1}$ . De forma similar, hacia la izquierda de  $B_0$  existen bloques  $B^{-1}, B^{-2}, \dots$ , con  $B^{-i}$  de longitud  $2^{i-1}$ . Suponemos que hay señaladores entre bloques, aunque en realidad no parezcan hasta que el bloque ha sido utilizado.

Hagamos que  $a_0$  denote el contenido de la celda barrida inicialmente por esta cabeza de cinta de  $M_1$ . El contenido de las celdas que se encuentran a la derecha de ésta es  $a_1, a_2, \dots$ , y el de las celdas que están a la izquierda  $a_{-1}, a_{-2}, \dots$ , respectivamente. Los valores de las  $a_i$  pueden variar cuando tengan acceso a  $B_0$ ; pero no son sus valores, sino sus posiciones en los registros de la cinta 1 de  $M_2$  lo que nos importa. Inicialmente el registro superior de  $M_2$  para la cinta de  $M_1$  en cuestión se supone que está vacío, mientras que el registro inferior se supone contiene a ...,  $a_{-2}, a_{-1}, a_0, a_1, a_2, \dots$  Estas se sitúan en los bloques ...,  $B_{-2}, B_{-1}, B_0, B_1, B_2, \dots$ , como se muestra en la Fig. 12.3.

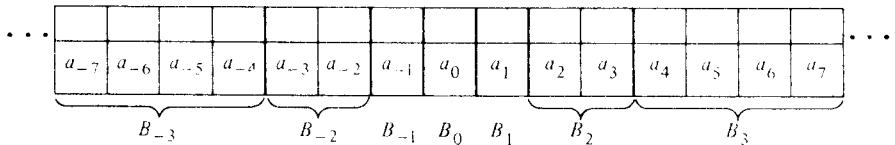


Fig. 12.3 Bloques en la cinta 1.

Como ya se dijo, los datos serán corridos a través de  $B_0$  y, tal vez, se modifiquen mientras lo hacen. Después de la simulación de cada movimiento de  $M_1$ , se hará válido lo siguiente:

- Para cualquier  $i > 0$ ,  $B_i$  está completo (ambos registros) y  $B_{-1}$  está vacío o  $B_i$  está vacío y  $B_{-1}$  está completo, o los registros inferiores de  $B_i$  y  $B_{-1}$  están llenos, mientras que los registros superiores están vacíos.
- El contenido de cualquiera,  $B_i$  o  $B_{-1}$  representan celdas consecutivas de la cinta de  $M_1$  representada. Para  $i > 0$ , el registro superior representa celdas que se encuentran a la izquierda de las celdas del registro inferior; para  $i < 0$ , el registro superior representa a celdas que están a la derecha de las celdas del registro inferior.
- Para  $i < j$ ,  $B_i$  representa celdas que se encuentran a la izquierda de las celdas de  $B_j$ .

4.  $B_0$  siempre tiene sólo su registro inferior lleno, y su registro superior está señalado de manera especial.

Para ver cómo se transfieren los datos imagínese que la cabeza de cinta de  $M_1$  en cuestión se mueve hacia la derecha. Entonces  $M_2$  debe correr los correspondientes datos hacia la derecha. Para llevar a cabo esto,  $M_2$  mueve la cabeza de la cinta 1 de  $B_0$ , en donde se encuentra, y se desplaza a la derecha hasta que encuentra el primer bloque, digamos  $B_i$ , que no tiene sus dos registros completos. Despues  $M_2$  copia todos los datos de  $B_0, B_1, \dots, B_{i-1}$  en la cinta 2 y la almacena en el registro inferior de  $B_1, B_2, \dots, B_{i-1}$  más el registro inferior de  $B_i$ , suponiendo que el registro inferior de  $B_i$  aún no está lleno. Si éste ya se encuentra completo, el registro superior de  $B_i$  se utiliza en su lugar. En cualquier caso, existe espacio justo para distribuir los datos. Adviértase, también, que los datos pueden tomarse y almacenarse en esta nueva posición en un tiempo proporcional a la longitud de  $B_i$ .

En seguida, en un tiempo proporcional a la longitud de  $B_i$ ,  $T_1$  puede encontrar a  $B_{i-1}$  (esto se facilita si se utiliza la cinta 2 para medir la distancia de  $B_i$  a  $B_0$ ). Si  $B_{i-1}$  se encuentra completamente lleno,  $T_1$  recoge el registro superior de  $B_{i-1}$  y lo almacena en la cinta 2. Si  $B_{i-1}$  está lleno a la mitad, el registro inferior se coloca en la cinta 2. En cualquier caso, lo que se copió en la cinta 2, se transfiere en seguida a los registros inferiores de  $B_{-(i-1)}, B_{-(i-2)}, \dots, B_0$ . (Según la Regla 1, estos registros tienen que estar vacíos, ya que  $B_1, B_2, \dots, B_{i-1}$  están completos). De nuevo, nótense que hay espacio justo para almacenar los datos, y todas las operaciones anteriores pueden efectuarse en un tiempo proporcional a la longitud de  $B_i$ . Adviértase también que los datos pueden distribuirse de manera que se satisfagan las reglas (1), (2) y (3), anteriores.

A todo lo descrito en el párrafo anterior se le conoce como operación  $B_i$ . El caso en el cual la cabeza de  $M_1$  se mueve hacia la derecha es análogo. El contenido posterior de los bloques, conforme  $M_1$  mueve la cabeza de cinta en cuestión cinco celdas a la izquierda, se muestra en la Fig. 12.4.

Vemos que, para cada cinta de  $M_1, M_2$  debe llevar a cabo una operación  $B_i$  cuando mucho una vez cada  $2^{i-1}$  movimientos de  $M_1$ , ya que se lleva este tiempo para llenar a  $B_1, B_2, \dots, B_{i-1}$ , que se encuentran llenos a la mitad después de una operación  $B_i$ . También, una operación  $B_i$  no puede llevarse a cabo por primera vez hasta que  $M_1$  realiza su movimiento  $2^{i-1}$ . Por consiguiente, si  $M_1$  opera en un tiempo  $T(n)$ ,  $M_2$  efectuará solamente operaciones  $B_i$ , para aquellas  $i$  tales que  $i \leq \log_2 T(n) + 1$ .

Hemos visto que existe una  $m$  constante, tal que  $M_2$  utiliza cuando mucho  $m \cdot 2^i$  movimientos para llevar a cabo una operación  $B_i$ . Si  $M_1$  hace  $T(n)$  movimientos,  $M_2$  efectúa cuando más.

$$T_1(n) = \sum_{i=1}^{\log_2 T(n) + 1} m \cdot 2^i \frac{T(n)}{2^{i-1}} \quad (12.4)$$

Movimientos cuando se encuentra simulando una cinta de  $M_1$ .

De la ecuación (12.4) obtenemos

$$T_1(n) = 2mT(n)[\log_2 T(n) + 1], \quad (12.5)$$

y de (12.5),

$$T_1(n) < 4mT(n) \log_2 T(n).$$

$B_{-3}$	$B_{-2}$	$B_{-1}$	$B_0$	$B_1$	$B_2$	$B_3$
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
$a_0$	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$
$a_7$						
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
				$a_0$	$a_1$	$a_2$
$a_3$	$a_4$	$a_5$	$a_6$	$a_7$		
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
				$a_0$	$a_1$	$a_2$
$a_5$	$a_6$	$a_7$				
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
				$a_0$	$a_1$	$a_2$
$a_6$	$a_7$					
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
				$a_0$	$a_1$	$a_2$
$a_7$						
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
				$a_0$	$a_1$	$a_2$
$a_4$	$a_5$	$a_6$	$a_7$			
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
				$a_0$	$a_1$	$a_2$
$a_3$	$a_4$	$a_5$	$a_6$	$a_7$		
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
				$a_0$	$a_1$	$a_2$
$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$	
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
				$a_0$	$a_1$	$a_2$
$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$	$a_7$
$a_{-7}$	$a_{-6}$	$a_{-5}$	$a_{-4}$	$a_{-3}$	$a_{-2}$	$a_{-1}$
				$a_0$	$a_1$	$a_2$
$a_0$	$a_1$	$a_2$	$a_3$	$a_4$	$a_5$	$a_6$

Fig. 12.4 Contenido de los bloques de  $M_1$ .

El lector deberá ser capaz de ver que  $M_2$  opera en un tiempo proporcional a  $T_1(n)$ , aun cuando  $M_1$  hace movimientos en los que utilice diferentes cintas de almacenamiento, en lugar de utilizar sólo la cinta en la que nos hemos concentrado. Según el Teorema 12.3, podemos modificar  $M_2$  para que corra en no más de  $T(n) \log_2 T(n)$  pasos.  $\square$

**Corolario** Si  $L$  es aceptado por una NTM de  $k$  cintas y complejidad temporal  $T(n)$ , entonces  $L$  es aceptado por una NTM de dos cintas y complejidad temporal  $T(n) \log T(n)$ .

**Demostración** La demostración es análoga a la del teorema. □

### 12.3 TEOREMAS DE JERARQUIAS

De manera intuitiva, si se tiene más tiempo o espacio, uno debería ser capaz de reconocer más lenguajes o calcular más funciones. Sin embargo, los teoremas de la aceleración lineal y la compresión nos dicen que tenemos que aumentar el espacio o el tiempo disponible en no más de un factor constante. ¿Pero qué sucede si multiplicamos el espacio o el tiempo por una función que crece lentamente, como  $\log \log n$ ? ¿Es posible que no podamos, entonces, reconocer cualesquiera nuevos lenguajes? ¿Existe un límite temporal o espacial  $f(n)$  tal que cada lenguaje recursivo se encuentra en  $\text{TIEMPOD}(f(n))$ , o tal vez en  $\text{ESPACIOD}(f(n))$ ?

La respuesta a la última interrogante es “no”, como se demostrará en el teorema siguiente. Sin embargo, la respuesta a la primera depende de si empezamos o no con una función “bien comportada”. En esta sección daremos definiciones apropiadas de “bien comportado” y demostraremos que para las funciones bien comportadas, se agregan pequeñas cantidades de tiempo y espacio extra.

En la Sección 12.6 consideraremos funciones recursivas totales arbitrarias y las clases de complejidad que definen. En esa sección veremos que se presenta un comportamiento extraño. Existen “huecos” en cualquier jerarquía de complejidad, esto es, existe una función  $T(n)$  para la cual  $\text{TIEMPOD}(T^2(n)) = \text{TIEMPOD}(T(n))$ , y en general, para cualquier función totalmente recursiva  $f$ , existe una complejidad temporal  $T_f(n)$  para la cual  $\text{TIEMPOD}(T_f(n)) = \text{TIEMPOD}(f(T_f(n)))$ . Proposiciones semejantes valen para el espacio y, de hecho, para cualquier medida razonable de complejidad computacional. También veremos que existen lenguajes  $L$  para los cuales no existe el “mejor” reconocedor; en lugar de esto existe una secuencia infinita de TMs que reconocen a  $L$ , cada una de las cuales corre mucho más rápido que la anterior.

**Teorema 12.7** Dado cualquier límite temporal (límite espacial) totalmente recursivo  $T(n)$ , existe un lenguaje recursivo  $L$  que no se encuentra en  $\text{TIEMPOD}(T(n))$  o en  $\text{ESPACIOD}(T(n))$ , respectivamente.

**Demostración** Demostraremos el resultado para el caso del tiempo; el argumento para el espacio es parecido. El argumento consiste básicamente en una diagonalización. Puesto que  $T(n)$  es totalmente recursivo, existe una TM que se detiene en  $M$ , que lo calcula. Construimos  $\hat{M}$  de manera que acepte al lenguaje  $L \subseteq (0 + 1)^*$  que es recursivo pero que no se encuentra en  $\text{TIEMPOD}(T(n))$ . Sea  $x_i$  la  $i$ -ésima cadena del ordenamiento canónico de  $(0 + 1)^*$ . En el Capítulo 8 ordenamos TMs de una sola cinta como alfabeto de cinta  $\{0, 1, B\}$ . Podemos ordenar de maneras parecidas a las TMs multicintas con alfabetos de cinta arbitrarios constituyendo sus funciones de transición por cadenas binarias. El único punto de importancia es que los nombres de los símbolos de cinta, como los de los estados, no importan, de modo que podemos suponer que todas las TMs, cuyo alfabeto de entrada es  $\{0, 1\}$ , tienen alfabeto de cinta  $0, 1, B, X_4, X_5, \dots$  hasta alguna  $X_n$  finita, y entonces codificar 0, 1 y  $B$  mediante 0, 00 y 000, y codificar  $X_i$  mediante  $0^i, i \geq 4$ . Permitimos también un número cualquiera de 1s en el frente del código, para que  $M$  represente a  $\hat{M}$ , de manera que  $M$  tenga códigos arbitrariamente largos.

Estamos, pues, en libertad de hablar de  $M_i$ , la  $i$ -ésima TM multicintas. Definimos ahora  $L = \{x_i \mid M_i \text{ no acepta a } x_i \text{ dentro de } T(|x_i|) \text{ movimientos}\}$ . Exigimos que  $L$  sea recursivo. Para reconocer a  $L$ , llévese a cabo el siguiente algoritmo, que puede, seguramente, ser instrumentado en una máquina de Turing. Dada la entrada  $w$  de longitud  $n$ , simúlese  $M$  en  $n$  para calcular  $T(n)$ . Despues determine  $i$  tal que  $w = x_i$ . El entero  $i$ , escrito en binario, es la función de transición de alguna TM multicintas  $M_i$  (si  $i$  en binario no está en una forma apropiada para la función de transición, entonces  $M_i$  no tiene movimientos). Simúlese  $M_i$  sobre  $w$  para  $T(n)$  movimientos, aceptando si  $M_i$  se detiene sin aceptar o si corre para más de  $T(n)$  movimientos y no acepta.

Para ver que  $L$  no está en el  $\text{TIEMPOD}(T(n))$ , supóngase que  $L = L(M_i)$  y que  $M_i$  tiene límite temporal  $T(n)$ . ¿Se encuentra  $x_i$  en  $L$ ? Si es así,  $M_i$  acepta a  $x_i$  en  $T(n)$  pasos, en donde  $n = |x_i|$ . En consecuencia, por definición de  $L$ ,  $x_i$  no se encuentra en  $L$ , lo que constituye una contradicción. Si  $x_i$  no se encuentra en  $L$ , entonces  $M_i$  no acepta a  $x_i$ , de modo que por definición de  $L$ ,  $x_i$  se encuentra en éste, lo que de nuevo representa una contradicción. Ambas suposiciones nos llevan a una contradicción, así pues, la suposición de que  $M_i$  tiene límite temporal  $T(n)$  debe ser falsa. □

Si  $T'(n) \geq T(n)$  para toda  $n$ , se concluye, de manera inmediata, de la definición de clase de complejidad temporal, que  $\text{TIEMPOD}(T(n)) \subseteq \text{TIEMPOD}(T'(n))$ . Si  $T(n)$  es una función totalmente recursiva, el Teorema 12.7 implica que existe un conjunto recursivo  $L$  que no está en  $\text{TIEMPOD}(T(n))$ . Hagamos que  $\hat{T}(n)$  sea el tiempo de corrido de alguna máquina de Turing que acepta a  $L$  y sea  $T'(n) = \max\{T(n), \hat{T}(n)\}$ . Entonces  $\text{TIEMPOD}(T(n)) \subsetneq \text{TIEMPOD}(T'(n))$ , ya que  $L$  se encuentra en el último, pero no en el primero. Por consiguiente, sabemos que existe una jerarquía infinita de clases de complejidad temporal determinística. Un resultado parecido vale para las clases de complejidad espacial determinística, y para las clases temporal y espacial no determinísticas.

El Teorema 12.7 demuestra que, para cualquier complejidad recursiva, temporal o espacial,  $f(n)$ , existe una  $f'(n)$  tal que algún lenguaje se encuentra en la clase de complejidad definida por  $f'(n)$ , pero no en la definida por  $f(n)$ . Demostraremos ahora que, para una función bien comportada,  $f(n)$ , solamente se necesita un pequeño incremento en la rapidez de crecimiento de  $f(n)$  para producir una nueva clase de complejidad. Los Teoremas 12.8 y 12.9 tratan sobre el incremento que se necesita con el fin de obtener una nueva clase de complejidad determinística. Estos teoremas se utilizarán más adelante para establecer límites inferiores a la complejidad de varios problemas. Resultados parecidos para las clases no determinísticas son mucho más difíciles de obtener; en la Sección 12.5 tocaremos una densa jerarquía para el espacio no determinístico.

#### Una jerarquía espacial

Introduciremos ahora nuestra noción de una función de complejidad espacial “bien comportada”. Se dice que una función  $S(n)$  es *espacialmente construible* si existe una Máquina de Turing  $M$  que tiene un límite espacial  $S(n)$  y, para cada  $n$ , existe una entrada de longitud  $n$  sobre la cual  $M$  en realidad utiliza  $S(n)$  celdas de cinta. El conjunto

de funciones espacialmente construibles incluye a  $\log n$ ,  $n^k$ ,  $2^n$  y  $n!$ .  $S_1(n)$  y  $S_2(n)$  son espacialmente construibles, entonces también lo son  $S_1(n)S_2(n)$ ,  $2^{S_1(n)}$ , y  $S_1(n)^{S_2(n)}$ . Por tanto, el conjunto de funciones espacialmente construibles es muy rico.

Nótese que la  $M$  del párrafo anterior no necesita utilizar el espacio  $S(n)$  en todas sus entradas de longitud  $n$ , justo en alguna entrada de esa longitud. Si para toda  $n$ , de hecho  $M$  utiliza exactamente  $S(n)$  celdas en cualquier entrada de longitud  $n$ , entonces decimos que  $S(n)$  es *espacialmente construible de manera completa*. Cualquier función espacialmente construible  $S(n) \geq n$  es espacialmente construible de manera completa (ejercicio).

Con el fin de simplificar el resultado que se da más adelante, demostraremos el siguiente lema.

**Lema 12.1** Si  $L$  es aceptado por una TM con límite espacial  $S(n) \geq \log_2 n$ , entonces  $L$  es aceptado por una TM con límite espacial  $S(n)$  que se detiene en todas las entradas.

**Demostración** Sea  $M$  una máquina de Turing fuera de línea con límite espacial  $S(n)$ ,  $s$  estados y  $t$  símbolos de cinta que acepta a  $L$ . Si  $M$  acepta, lo hace mediante una sucesión de cuando más  $(n+2)sS(n)t^{S(n)}$  movimientos, ya que de otra manera alguna ID se repite. Esto es, existen  $n+2$  posiciones de entrada de la cabeza,  $s$  estados,  $S(n)$  posiciones de la cabeza de cinta y  $t^{S(n)}$  contenidos de cinta de almacenamiento. Si se añade un registro adicional como contador de movimientos,  $M$  puede apagarse a sí misma después de  $(4st)^{S(n)} \geq (n+2)sS(n)t^{S(n)}$  movimientos. En realidad  $M$  establece un contador de longitud  $\log n$ , y cuenta en base  $4st$ . Siempre que  $M$  barre una nueva celda que se encuentra más allá de las celdas que contienen al contador, incrementa la longitud de éste. Por consiguiente si  $M$  efectúa un ciclo habiendo utilizado solamente  $i$  celdas de cinta, entonces el contador detectará esto cuando la cuenta alcance  $(4st)^{\max(i, \log_2 n)}$ , que es al menos  $(n+2)sS(n)t^{S(n)}$ .  $\square$

**Teorema 12.8** Si  $S_2(n)$  es una función espacialmente construible de manera completa,

$$\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0,$$

y  $S_1(n)$  y  $S_2(n)$  son, cada uno, de al menos  $\log_2 n$ , entonces existe un lenguaje que se encuentra en  $\text{ESPACIOD}(S_2(n))$  pero no en  $\text{ESPACIOD}(S_1(n))$ .

**Demostración** El teorema se demuestra mediante diagonalización. Considérese una enumeración de máquinas de Turing fuera de línea con alfabeto de entrada  $\{0, 1\}$  y una cinta de almacenamiento, basadas en el código binario de la Sección 8.3, pero que tienen un prefijo de 1s permitido, de modo que cada TM tiene codificaciones arbitrariamente largas. Construimos una TM,  $M$ , que utiliza espacio  $S_2(n)$  y no coincide, en al menos una entrada, con cualquiera de las TMs con límite espacial  $S_1(n)$ .

Sobre la entrada  $w$ ,  $M$  comienza señalando  $S_2(n)$  celdas sobre una cinta, en donde  $n$  es la longitud de  $w$ . Puesto que  $S_2(n)$  es espacialmente construible de manera completa, esto se puede hacer mediante la simulación de una TM que utilice exactamente  $S_2(n)$  celdas en cada entrada de longitud  $n$ . En lo sucesivo, si  $M$  intenta abandonar las celdas señalada,  $M$  se detiene y rechaza a  $w$ . Esto nos garantiza que  $M$  tiene límite espacial  $S_2(n)$ .

En seguida  $M$  comienza una simulación, sobre la entrada  $w$ , de la TM  $M_w$ , la TM codificada por la cadena binaria  $w$ . Si  $M_w$  tiene límite espacial  $S_1(n)$  y  $t$  símbolos de cinta, entonces la simulación requiere de espacio  $\lceil \log_2 t \rceil S_1(n)$ .  $M$  acepta a  $w$  sólo si  $M$  puede completar la simulación en un espacio  $S_2(n)$  y  $M_w$  se detiene sin aceptar a  $x$ . Como  $M$  tiene límite espacial  $S_2(n)$ ,  $L(M)$  se encuentra en  $\text{ESPACIOD}(S_2(n))$ .  $L(M)$  no se encuentra en  $\text{ESPACIOD}(S_1(n))$ . Porque, supóngase que existe una TM con límite espacial  $S_1(n)$ ,  $\hat{M}$ , con  $t$  símbolos de cinta que acepta a  $L(M)$ . Según el Lema 12.1, podemos asumir que  $\hat{M}$  se detiene en todas las entradas Debido a que  $\hat{M}$  aparece infinitad de veces en la enumeración y

$$\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0,$$

existe una  $w$  lo suficientemente larga,  $|w| = n$ , tal que  $\lceil \log_2 t \rceil S_1(n) < S_2(n)$  y  $M_w$  es  $\hat{M}$ . Sobre la entrada  $w$ ,  $M$  tiene espacio suficiente para simular a  $M_w$  y aceptar si y sólo si  $M_w$  rechaza. Por consiguiente,  $L(M_w) \neq L(M)$ , lo que es una contradicción. Por tanto  $L(M)$  se encuentra en  $\text{ESPACIOD}(S_2(n))$  pero no en  $\text{ESPACIOD}(S_1(n))$ .  $\square$

Mientras que la mayoría de las funciones comunes son espacialmente contribuibles de manera completa, necesitamos solamente la constructibilidad espacial para hacer que el Teorema 12.8 valga. Establecemos, por tanto, el siguiente corolario.

**Corolario** El Teorema 12.8 es válido aun si  $S_2(n)$  es espacialmente construible pero no espacialmente construible de manera completa.

**Demostración** Sea  $M_1$  una TM que construye a  $S_2(n)$  sobre alguna entrada. Sea  $\Sigma$  el alfabeto de entrada de  $M_1$ . Diseñamos a  $M$  de modo que acepte un lenguaje sobre el alfabeto  $\Sigma \times \{0, 1\}$ . Esto es, la entrada de  $M$  es tratada como si tuviera dos registros: el primero se utiliza como entrada de  $M_1$ , el segundo como el código de una TM con alfabeto de entrada  $\Sigma \times \{0, 1\}$ . La única modificación al diseño de  $M$  es que ésta debe dejar de utilizar los bloques de las cintas 1 y 2 mediante la simulación de  $M_1$  en el primer registro de  $M$ . Podemos demostrar que  $M$  no coincide con cualquier TM con límite espacial  $S_1(n)$ ,  $\hat{M}$ , sobre una entrada cuya longitud,  $n$ , es lo suficientemente grande, y cuyo primer registro es una cadena de  $\Sigma^n$  que ocasiona que  $M_1$  utilice  $S_2(n)$  celdas, y cuyo segundo registro es una codificación de  $\hat{M}$ .  $\square$

Dejamos como ejercicio la demostración de que la condición  $S_2(n) \geq \log_2 n$  en el Teorema 12.8 y su corolario no son realmente necesarios. La demostración no es una diagonalización, sino que depende de la demostración de que

$$\{wc^i w \mid w \text{ está en } (\mathbf{a} + \mathbf{b})^*, \quad |w| = S_2(n) \quad \text{y} \quad i = n - 2S_2(n)\}$$

es aceptado en espacio  $S_2(n)$ , pero no en el espacio  $S_1(n)$  si

$$\inf_{n \rightarrow \infty} \frac{S_1(n)}{S_2(n)} = 0,$$

y  $S_2(n) < \log_2 n$ .

Nótese que si  $\inf_{n \rightarrow \infty} [S_1(n)/S_2(n)] = 0$  y  $S_1(n) \leq S_2(n)$  para toda  $n$ , entonces

**ESPACIOD**( $S_1(n)$ )  $\subsetneq$  **ESPACIOD**( $S_2(n)$ ).

Sin embargo, si no tenemos la condición  $S_1(n) \leq S_2(n)$ , entonces es posible que **ESPACIOD**( $S_1(n)$ ) y **ESPACIOD**( $S_2(n)$ ), cada uno, tengan lenguajes que no están contenidos en el otro.

### Una jerarquía temporal

La jerarquía temporal determinística no es tan rígida como la jerarquía espacial. La razón para que esto sea así reside en que una TM que, diagonaliza sobre TMs multicintas, tiene algún número fijo de cintas. Para simular a TM con un número mayor de cintas hacemos uso de la simulación de dos cintas de una TM multicintas; por tanto, introducimos una desaceleración logarítmica. Antes de dar la construcción introduciremos el concepto de constructibilidad temporal.

Se dice que una función  $T(n)$  es *temporalmente construible* si existe una máquina de Turing multicintas con límite temporal  $T(n)$  que diagonaliza sobre máquinas de Turing multicintas, de modo que, para cada  $n$ , existe alguna entrada sobre la cual  $M$  en realidad efectúa  $T(n)$  movimientos. Justamente, como se tiene para las funciones espacialmente construibles, existe una jerarquía rica de funciones temporalmente construibles. Decimos que  $T(n)$  es *temporalmente construible de manera completa* si existe una TM que utilice un tiempo  $T(n)$  sobre todas las entradas de longitud  $n$ . De nuevo, la mayoría de las funciones comunes son temporalmente construibles de manera completa.

**Teorema 12.9** Si  $T_2(n)$  es una función temporalmente construible de manera completa y

$$\inf_{n \rightarrow \infty} \frac{T_1(n) \log T_1(n)}{T_2(n)} = 0,$$

entonces existe un lenguaje en **TIEMPOD**( $T_2(n)$ ) pero que no está en **TIEMPOD**( $T_1(n)$ ).

**Demostración** La demostración es parecida a la del Teorema 12.8, y solamente daremos un pequeño bosquejo de la construcción necesaria. Se construye una TM con límite temporal  $T_2(n)$ ,  $M$ , para que funcione de la siguiente forma.  $M$  trata la entrada  $w$  como un código de una máquina de Turing  $\hat{M}$  y simula a  $\hat{M}$  sobre la entrada  $w$ . Aquí surge una dificultad, ya que  $M$  tiene algún número fijo de cintas, así que, para algunas  $ws$ ,  $\hat{M}$  tendrá más cintas que  $M$ . Afortunadamente, según el Teorema 12.6, solamente se necesitan dos cintas para simular a cualquier  $\hat{M}$ , aunque la simulación cuesta un factor de  $\log T_1(n)$ . También, ya que  $\hat{M}$  puede tener muchos símbolos de cinta, lo que debe ser codificado en algún número fijo de símbolos, la simulación de  $T_1(n)$  movimientos de  $\hat{M}$  por parte de  $M$  requiere un tiempo  $cT_1(n) \log T_1(n)$ , en donde  $c$  es una constante que depende de  $\hat{M}$ .

Con el fin de asegurarnos que la simulación de  $\hat{M}$  tiene límite  $T_2(n)$ ,  $M$  ejecuta de manera simultánea los pasos de una TM (utilizando cintas adicionales) que toma exactamente un tiempo  $T_2(n)$  en todas sus entradas de longitud  $n$ . Esta es la razón por la cual  $T_2(n)$  debe ser temporalmente construible de manera completa. Después de  $T_2(n)$

pasos,  $M$  se detiene.  $M$  acepta a  $w$  solamente si se acepta la simulación de  $\hat{M}$  y está rechazada de a  $w$ . La codificación de  $\hat{M}$  se diseña de la misma forma que el teorema anterior, de modo que cada  $\hat{M}$  tiene codificaciones arbitrariamente largas. Por consiguiente, si  $\hat{M}$  es una máquina de Turing con límite temporal  $T_1(n)$ , existirá una  $w$  lo suficientemente larga que codifique a  $\hat{M}$  de forma que

$$cT_1(|w|) \log T_1(|w|) \leq T_2(|w|),$$

y la simulación deberá llevar a una conclusión. En este caso,  $w$  está en **L**( $M$ ) si y sólo si  $w$  no se encuentra en **L**( $\hat{M}$ ). Así **L**( $M$ )  $\neq$  **L**( $\hat{M}$ ) para toda  $\hat{M}$  con límite de tiempo  $T_1(n)$ . Por consiguiente **L**( $M$ ) está en **TIEMPOD**( $T_2(n)$ ) – **TIEMPOD**( $T_1(n)$ ).  $\square$

**Ejemplo 12.3** Sean  $T_1(n) = 2^n$  y  $T_2(n) = n^2 2^n$ . Entonces

$$\inf_{n \rightarrow \infty} \frac{T_1(n) \log_2 T_1(n)}{T_2(n)} = \inf_{n \rightarrow \infty} \frac{1}{n} = 0.$$

Por tanto se aplica el Teorema 12.9, y **TIEMPOD**( $2^n$ )  $\neq$  **TIEMPOD**( $n^2 2^n$ ). Puesto que  $T_1(n) \leq T_2(n)$  para toda  $n$ , podemos concluir que **TIEMPOD**( $2^n$ )  $\subsetneq$  **TIEMPOD**( $n^2 2^n$ ).

### 12.4 RELACIONES ENTRE MEDIDAS DE COMPLEJIDAD

Existen varias relaciones sencillas, y una que no es tan obvia, entre las complejidades de un lenguaje dado  $L$ , de acuerdo con las cuatro medidas de complejidad que hemos definido. Las relaciones sencillas se establecen en un teorema.

#### Teorema 12.10

- a) Si  $L$  está en **TIEMPOD**( $f(n)$ ), entonces  $L$  está en **ESPACIOD**( $f(n)$ ).
- b) Si  $L$  está en **ESPACIOD**( $f(n)$ ) y  $f(n) \geq \log_2 n$ , entonces existe alguna constante  $c$ , dependiendo de  $L$ , tal que  $L$  está en **TIEMPOD**( $c^{f(n)}$ ).
- c) Si  $L$  está en **TIEMPON**( $f(n)$ ), entonces existe alguna constante  $c$ , dependiendo de  $L$ , tal que  $L$  está en **TIEMPON**( $c^{f(n)}$ ).

#### Demostración

- a) Si la TM  $M$  no hace más de  $f(n)$  movimientos, no puede barrer más de  $f(n) + 1$  celdas sobre cualquier cinta. Mediante la modificación de  $M$  para que contenga dos símbolos por celda, podemos disminuir los requerimientos de almacenamiento a  $\lceil [f(n) + 1]/2 \rceil$ , que es cuando mucho  $f(n)$ .
- b) Obsérvese que si la TM  $M_1$  tiene  $s$  estados y  $t$  símbolos de cinta, y utiliza cuando mucho un espacio  $f(n)$ , entonces el número de IDs diferentes de  $M_1$ , con entrada de longitud  $n$  es, como máximo  $s(n+2)f(n)t^{f(n)}$ . Como  $f(n) \geq \log_2 n$ , existe alguna constante  $c$  tal que, para toda  $n \geq 1$ ,  $c^{f(n)} \geq s(n+2)f(n)t^{f(n)}$ .

A partir de  $M_1$ , constrúyase una TM multicintas  $M_2$  que utilice una cinta para contar a  $c^{f(n)}$ , y otras dos cintas para simular a  $M_1$ . Si  $M_1$  no ha aceptado cuando la cuenta alcance  $c^{f(n)}$ ,  $M_2$  se detiene sin aceptar. Despues de este número de movi-

mientos,  $M_1$  debe haber repetido alguna ID y, por tanto, nunca va a aceptar. Es claro que  $M_2$  tiene límite temporal  $c^{f(n)}$ .

- c) Sea  $M_1$  una TM no determinística con límite temporal  $f(n)$ ,  $s$  estados,  $t$  símbolos de cinta y  $k$  cintas. El número de IDs posibles de  $M_1$ , dada la entrada de longitud  $n$ , es de cuando mucho  $s(f(n) + 1)^k t^{k f(n)}$ , el producto del número de estados, posiciones de cabeza y contenido de cinta. Por consiguiente  $d = s(t+1)^{3k}$  satisface.

$$d^{f(n)} \geq s(f(n) + 1)^k t^{k f(n)} \text{ para toda } n \geq 1.$$

Una TM multicintas determinística puede determinar si  $M_1$  acepta la entrada  $w$ , de longitud  $n$ , mediante la construcción de una lista de todas las IDs de  $M_1$  que son accesibles a partir de la ID inicial. Este proceso puede efectuarse en el límite temporal mediante el cuadrado de la longitud de la lista. Como la lista de IDs accesibles tiene una longitud que no es mayor que  $d^{f(n)}$  veces la longitud de una ID, que puede ser codificada en  $1 + k(f(n) + 1)$  símbolos, el tiempo está limitado por  $c^{f(n)}$ , para alguna constante  $c$ .  $\square$

**Teorema 12.11 (teorema de Savitch)** Si  $L$  está en  $\text{ESPACION}(S(n))$  entonces  $L$  está en  $\text{ESPACIOD}(S^2(n))$  siempre y cuando  $S(n)$  sea espacialmente construible y  $S(n) \geq \log_2 n$ .

*Demuestra* Se  $L = L(M_1)$ , en donde  $M_1$  es una TM no determinística con límite espacial  $S(n)$ . Para alguna constante  $c$ , cuando mucho existen  $c^{S(n)}$  IDs para una entrada de longitud  $n$ . Por consiguiente, si  $M_1$  acepta su entrada, lo hace mediante una secuencia de cuando más  $c^{S(n)}$  movimientos, ya que ninguna ID se repite en el cálculo más corto de  $M_1$  que conduce a la aceptación.

Hagamos que  $I_1 \xrightarrow{(i)} I_2$  signifique que la ID  $I_2$  pueda ser alcanzada desde  $I_1$  mediante una secuencia de cuando mucho  $2^i$  movimientos. Para  $i \geq 1$ , podemos determinar si  $I_1 \xrightarrow{(i)} I_2$  probando cada  $I'$  para verificar si  $I_1 \xrightarrow{(i-1)} I'$  y  $I' \xrightarrow{(i-1)} I_2$ . Por consiguiente, el espacio que se necesita para determinar si podemos llegar de una ID a otra en  $2^i$  movimientos es igual al espacio que se necesita para determinar si podemos llegar de una ID a otra en  $2^{i-1}$  movimientos. Obsérvese que el espacio utilizado para probar si una ID es alcanzable a partir de otra en  $2^{i-1}$  movimientos puede ser utilizado de nuevo para cada una de tales pruebas.

En la Fig. 12.5 se dan los detalles para probar si  $w$  se encuentra en  $L(M_1)$ . El algoritmo de la Fig. 12.5 puede instrumentarse en una máquina de Turing  $M_2$  que utiliza una cinta como pila de registros de activación† para hacer los llamados a PRUEBA. Cada llamado tiene un registro de activación en el cual se colocan los valores de los parámetros  $I_1, I_2$  e  $i$ , así como también el valor de la variable local  $I'$ . Como  $I_1, I_2$  e  $I'$  son IDs con no más de  $S(n)$  celdas, podemos representar a cada una de ellas en un espacio  $S(n)$ . La posición de entrada de la cabeza en binario utiliza  $\log n \leq S(n)$  celdas. Nótese que la cinta de entrada en todas las IDs está fija y es la misma que la entrada de  $M_2$ , de modo que no necesitamos copiar la entrada en cada ID. El parámetro  $i$  puede

† Un “registro de activación” es el área utilizada por los datos que pertenecen a un llamado de un procedimiento.

codificarse en binario utilizando cuando más  $m S(n)$  celdas. Por consiguiente cada registro de activación se lleva un espacio  $O(S(n))$ .

```

begin
  sea  $|w| = n$  y  $m = \lceil \log_2 c \rceil$ 
  sea  $I_0$  una ID inicial de  $M_1$  con entrada  $w$ ;
  for cada ID final  $I_f$  de longitud de cuando más  $S(n)$  do
    if PRUEBA ( $I_0, I_f, m S(n)$ ) then acept;
  end;

  procedure PRUEBA ( $I_1, I_2, i$ );
    if  $i = 0$  y  $(I_1 = I_2 \circ I_1 \mid= I_2)$  then return true;
    if  $i \geq 1$  then
      for cada ID  $I'$  de longitud de cuando más  $S(n)$  do
        if PRUEBA ( $I_1, I', i - 1$ ) y PRUEBA ( $I', I_2, i - 1$ ) then
          return true;
    return false
  end PRUEBA

```

Fig. 12.5 Algoritmo para simular a  $M_1$ .

Como el tercer parámetro disminuye una unidad cada vez que se llama a PRUEBA, la llamada inicial tiene  $i = mS(n)$ , y no se hace ninguna llamada cuando  $i$  alcanza el valor cero, el número máximo de registros de activación en la pila es  $O(S(n))$ , por tanto el espacio total utilizado es  $O(S^2(n))$  y, según el Teorema 12.1, podemos rediseñar a  $M_2$  para hacer que el espacio sea exactamente  $S^2(n)$ .  $\square$

#### Ejemplo 12.4

$$\begin{aligned} \text{ESPACION}(\log n) &\subseteq \text{ESPACIOD}(\log^2 n) \\ \text{ESPACION}(n^2) &\subseteq \text{ESPACIOD}(n^4) \quad \text{y} \quad \text{ESPACION}(2^n) \subseteq \text{ESPACIOD}(4^n). \end{aligned}$$

Nótese que, para  $S(n) \geq n$ , el teorema de Savitch vale aún si  $S(n)$  es espacialmente construible más que si es espacialmente construible de manera completa.  $M_2$  comienza simulando una TM  $M$  que construye a  $S(n)$ , sobre cada entrada de longitud  $n$ , tomando la cantidad más grande de espacio utilizado como  $S(n)$  y usando esta longitud para representar el espacio de los registros de activación. Obsérvese, sin embargo, que si no tenemos manera de calcular  $S(n)$  en el espacio par  $S^2(n)$ , entonces no podemos hacer un ciclo a través de todos los valores posibles de  $I_f$  o de  $I'$  sin alcanzar a alguno que tome demasiado espacio.

#### 12.5 LEMAS DE TRASLACION Y LAS JERARQUIAS NO DETERMINISTICAS

En los teoremas 12.8 y 12.9 vimos que las jerarquías de espacio y tiempo determinísticos son muy densas. Esto podría parecer que las correspondientes jerarquías para las máquinas no determinísticas necesitarían un incremento de un cuadrado para el espacio

y una exponencial para el tiempo, para simular una máquina no determinística con propósitos de diagonalización. Sin embargo, se puede utilizar un argumento translacional para dar una jerarquía mucho más densa para las máquinas no determinísticas. Ilustraremos la técnica para el caso del espacio.

### Lema de traslación

El primer paso consiste en demostrar que la contención se traslada hacia arriba. Por ejemplo, supóngase que es verdad (lo que no es cierto) que  $\text{ESPACION}(n^3) \subseteq \text{ESPACION}(n^2)$ . Esta relación podría trasladarse hacia arriba si reemplazamos  $n$  por  $n^2$ , produciéndose

$$\text{ESPACION}(n^6) \subseteq \text{ESPACION}(n^4).$$

**Lema 12.2** Sean  $S_1(n)$ ,  $S_2(n)$  y  $f(n)$  funciones espacialmente construibles de manera completa, en las que  $S_2(n) \geq n$  y  $f(n) \geq n$ . Entonces

$$\text{ESPACION}(S_1(n)) \subseteq \text{ESPACION}(S_2(n)).$$

implica que

$$\text{ESPACION}(S_1(f(n))) \subseteq \text{ESPACION}(S_2(f(n))).$$

*Demostración* Sea  $L_1$  un lenguaje aceptado por  $M_1$ , una TM no determinística con límite espacial  $S_1(f(n))$ . Sea

$$L_2 = \{x \$^i \mid M_1 \text{ acepta a } x \text{ en el espacio } S_1(|x| + i)\},$$

en donde  $\$$  es un nuevo símbolo que no se encuentra en el alfabeto de  $L_1$ . Entonces  $L_2$  es aceptado por una TM  $M_2$  de la manera siguiente. Sobre la entrada  $x \$^i$ ,  $M_2$  señala  $S_1(|x| + i)$  celdas, lo cual puede hacer, puesto que  $S_1$  es completamente construible. Entonces  $M_2$  simula a  $M_1$  sobre  $x$ , aceptando si y sólo si  $M_1$  acepta sin utilizar más de  $S_1(|x| + i)$  celdas. Es claro que  $M_2$  tiene límite espacial  $S_1(n)$ .

Lo que hemos hecho es tomar  $L_1$  en  $\text{ESPACION}(S_1(f(n)))$  y llenar las cadenas con  $\$$ s de tal manera que la versión llenada de  $L_2$  está en  $\text{ESPACION}(S_1(n))$ . Ahora bien, según la hipótesis de que  $\text{ESPACION}(S_1(n)) \subseteq \text{ESPACION}(S_2(n))$ , existe una TM no determinística con límite espacial  $S_2(n)$ ,  $M_3$ , que acepta a  $L_2$ .

Finalmente construimos  $M_4$  que acepta al conjunto original  $L_1$  dentro del espacio  $S_2(f(n))$ .  $M_4$  señala  $f(n)$  celdas y después  $S_2(f(n))$  celdas, lo cual puede hacer, puesto que  $f$  y  $S_2$  son completamente construibles. Como  $S_2(n) \geq n$ ,  $f(n) \leq S_2(f(n))$ , de modo que  $M_4$  no ha utilizado más de  $S_2(f(n))$  celdas.

En seguida  $M_4$ , sobre la entrada  $x$ , simula a  $M_3$  sobre  $x \$^i$ , para  $i = 0, 1, 2, \dots$ . Para hacer esto,  $M_4$  debe mantener registro de la posición de la cabeza de  $M_3$  sobre  $x \$^i$ . Si la cabeza de  $M_3$  se encuentra dentro de  $x$ , la cabeza de  $M_4$  se encuentra en el punto correspondiente sobre su entrada. Siempre que la cabeza de  $M_3$  se mueve hacia los  $\$$ s,  $M_4$  registra la localización en un contador. La longitud del contador es, todo lo más, de  $\log i$ .

Si durante la simulación,  $M_3$  acepta, entonces  $M_4$  acepta. Si  $M_3$  no acepta, entonces  $M_4$  aumenta  $i$  hasta que el contador ya no encaja más en  $S_2(f(|x|))$  celdas de cinta.

Entonces  $M_4$  se detiene. Ahora, si  $x$  se encuentra en  $L_1$ , entonces  $x \$^i$  se encuentra en  $L_2$ , para  $i$  que satisface  $S_1(|x| + i) = S_1(f(|x|))$ . Como  $f(n) \geq n$ , esta igualdad es satisfecha por  $i = f(|x|) - |x|$ . Por consiguiente el contador requiere un espacio  $\log(f(|x|) - |x|)$ . Ya que  $S_2(f(|x|)) \geq f(|x|)$ , se concluye que el contador coincidirá. Por tanto  $x$  se encuentra en  $L(M_4)$  si y sólo si  $x \$^i$  se encuentra en  $L(M_3)$ , para alguna  $i$ . En consecuencia  $L(M_4) = L_1$  y  $L_1$  se encuentra en  $\text{ESPACION}(S_2(f(n)))$ .  $\square$

Nótese que podemos relajar la condición de que  $S_2(n) \geq n$ , requiriendo solamente que  $S_2(n) \geq \log_2 n$ , siempre y cuando  $S_2(f(n))$  sea espacialmente construible de manera completa. Entonces  $M_4$  puede dejar de utilizar  $S_2(f(n))$  celdas sin dejar de utilizar  $f(n)$  celdas. Como  $S_2(f(n)) \geq \log f(n)$ , todavía queda espacio para el contador de  $M_4$ .

Esencialmente, el mismo argumento que se dio en la demostración del lema 12.2 muestra los resultados análogos para  $\text{ESPACIOD}$ ,  $\text{TIEMPOD}$  y  $\text{TIEMPON}$ .

**Ejemplo 12.5** Utilizando el resultado para la traslación análoga para el tiempo determinístico, podemos demostrar que  $\text{TIEMPOD}(2^n) \not\subseteq \text{TIEMPOD}(n 2^n)$ . Nótese que este resultado no se concluye del Teorema 12.9, ya que

$$\inf_{n \rightarrow \infty} \frac{2^n \log 2^n}{n 2^n} = 1.$$

Supóngase que

$$\text{TIEMPOD}(n 2^n) \subseteq \text{TIEMPOD}(2^n).$$

Entonces, haciendo  $S_1(n) = n 2^n$ ,  $S_2(n) = 2^n$  y  $f(n) = 2^n$ , obtenemos

$$\text{TIEMPOD}(2^n 2^n) \subseteq \text{TIEMPOD}(2^{2n}). \quad (12.6)$$

De manera similar, haciendo  $f(n) = n + 2^n$  obtenemos

$$\text{TIEMPOD}(n + 2^n) 2^{2n} \subseteq \text{TIEMPOD}(2^n 2^{2n}). \quad (12.7)$$

Haciendo una combinación de (12.6) con (12.7) obtenemos

$$\text{TIEMPOD}((n + 2^n) 2^{2n}) \subseteq \text{TIEMPOD}(2^{2n}). \quad (12.8)$$

Sin embargo,

$$\inf_{n \rightarrow \infty} \frac{2^{2n} \log 2^{2n}}{(n + 2^n) 2^n 2^{2n}} = \inf_{n \rightarrow \infty} \frac{1}{n + 2^n} = 0.$$

Por consiguiente, el Teorema 12.9 implica que (12.8) es falsa, de modo que nuestra suposición de que  $\text{TIEMPOD}(n 2^n) \subseteq \text{TIEMPOD}(2^n)$  debe ser falsa. Como  $\text{TIEMPOD}(2^n) \subseteq \text{TIEMPOD}(n 2^n)$ , concluimos que  $\text{TIEMPOD}(2^n) \not\subseteq \text{TIEMPOD}(n 2^n)$ .

**Ejemplo 12.6** El lema de traslación puede ser utilizado para demostrar que  $\text{ESPACION}(n^3)$  está contenido propiamente en  $\text{ESPACION}(n^4)$ . Supóngase por el contrario que  $\text{ESPACION}(n^4) \subseteq \text{ESPACION}(n^3)$ . Entonces, haciendo  $f(n) = n^3$ , obtenemos  $\text{ESPACION}(n^{12}) \subseteq \text{ESPACION}(n^9)$ . De manera similar, haciendo

$f(n) = n^4$ , obtenemos  $\text{ESPACION}(n^{16}) \subseteq \text{ESPACION}(n^{12})$ ,  $f(n) = n^5$  da  $\text{ESPACION}(n^{20}) \subseteq \text{ESPACION}(n^{15})$ . Juntando todo lo anterior se produce  $\text{ESPACION}(n^{20}) \subseteq \text{ESPACION}(n^9)$ . Sin embargo, por el teorema 12.11, sabemos que  $\text{ESPACION}(n^9) \subseteq \text{ESPACION}(n^{18})$  y por el teorema 12.8, que  $\text{ESPACIOD}(n^{18}) \subsetneq \text{ESPACIOD}(n^{20})$ . Por consiguiente, si combinamos estos resultados, obtenemos

$$\begin{aligned}\text{ESPACION}(n^{20}) &\subseteq \text{ESPACION}(n^9) \subseteq \text{ESPACIOD}(n^{18}) \\ &\subsetneq \text{ESPACIOD}(n^{20}) \subseteq \text{ESPACION}(n^{20})\end{aligned}$$

lo que constituye una contradicción. Por consiguiente, nuestra suposición de que  $\text{ESPACION}(n^4) \subseteq \text{ESPACION}(n^3)$  es equivocada y llegamos a la conclusión de que  $\text{ESPACION}(n^3) \subsetneq \text{ESPACION}(n^4)$ .

### Una jerarquía espacial no determinística

El Ejemplo 12.6 puede generalizarse para mostrar una jerarquía densa para el espacio no determinístico en el intervalo de los polinomios.

**Teorema. 12.12** Si  $\epsilon > 0$  y  $r \geq 0$ , entonces

$$\text{ESPACION}(n^r) \subsetneq \text{ESPACION}(n^{r+\epsilon}).$$

*Demostración* Si  $r$  es cualquier número real no negativo, podemos encontrar enteros positivos  $s$  y  $t$  tales que  $r \leq s/t$  y  $r + \epsilon \geq (s+1)/t$ . En consecuencia es suficiente demostrar, para todos los enteros positivos  $s$  y  $t$ , que

$$\text{ESPACION}(n^{s/t}) \subsetneq \text{ESPACION}(n^{(s+1)/t}).$$

Por el contrario supóngase que

$$\text{ESPACION}(n^{(s+1)/t}) \subseteq \text{ESPACION}(n^{s/t}).$$

Entonces, según el lema 12.2 con  $f(n) = n^{(s+i)/t}$ , tenemos

$$\text{ESPACION}(n^{(s+1)(s+i)}) \subseteq \text{ESPACION}(n^{s(s+i)}) \quad (12.9)$$

para  $i = 0, 1, \dots, s$ . Como  $s(s+i) \leq (s+1)(s+i-1)$ , para  $i \geq 1$ , sabemos que

$$\text{ESPACION}(n^{s(s+i)}) \subseteq \text{ESPACION}(n^{(s+1)(s+i-1)}). \quad (12.10)$$

Utilizando (12.9) y (12.10) de manera alternada, tendremos

$$\begin{aligned}\text{ESPACION}(n^{(s+1)(2s)}) &\subseteq \text{ESPACION}(n^{s(2s)}) \\ &\subseteq \text{ESPACION}(n^{(s+1)(2s-1)}) \subseteq \text{ESPACION}(n^{s(2s-1)}) \\ &\subseteq \dots \subseteq \text{ESPACION}(n^{(s+1)s}) \subseteq \text{ESPACION}(n^{s^2})\end{aligned}$$

Esto es,

$$\text{ESPACION}(n^{2s^2+2s}) \subseteq \text{ESPACION}(n^{s^2}).$$

Sin embargo, según el Teorema de Savitch,

$$\text{ESPACION}(n^{s^2}) \subseteq \text{ESPACION}(n^{2s^2}),$$

y según el Teorema 12.8,

$$\text{ESPACIOD}(n^{2s^2}) \subsetneq \text{ESPACIOD}(n^{2s^2+2s}).$$

Es claro que

$$\text{ESPACIOD}(n^{2s^2+2s}) \subseteq \text{ESPACION}(n^{2s^2+2s}).$$

Combinando estos resultados, obtenemos

$$\text{ESPACION}(n^{2s^2+2s}) \subsetneq \text{ESPACION}(n^{2s^2+2s}),$$

lo que constituye una contradicción. Concluimos que nuestra suposición

$$\text{ESPACION}(n^{(s+1)/t}) \subseteq \text{ESPACION}(n^{s/t})$$

está equivocada. Como la contención en la dirección opuesta es obvia, concluimos que

$$\text{ESPACION}(n^{s/t}) \subsetneq \text{ESPACION}(n^{(s+1)/t})$$

para cualesquier enteros positivos  $s$  y  $t$ .  $\square$

Se pueden demostrar jerarquías densas parecidas para el espacio no determinístico en intervalos mayores que los polinomiales, y dejamos algunos de tales resultados como ejercicios. El Teorema 12.12 no se generaliza de manera inmediata al tiempo no determinístico, debido al papel clave que desempeña el teorema de Savitch, del cual no se conoce uno correspondiente para el tiempo. Sin embargo, un análogo para el tiempo del Teorema 12.12 fue establecido por Cook [1973a].

### 12.6 PROPIEDADES DE LAS MEDIDAS DE COMPLEJIDAD GENERALES: TEOREMAS DE ESPACIO, ACCELERACIÓN Y UNIÓN

En esta sección discutiremos algunas propiedades intuitivas de las medidas de complejidad. Se demostrarán solamente para la complejidad espacial, pero veremos en la siguiente sección que se aplican a todas las medidas de complejidad.

Los Teoremas 12.8 y 12.9 indican que las jerarquías espacial y temporal son muy densas. Sin embargo, en ambos teoremas se requiere que las funciones sean construibles. ¿Puede esto ser descartado? La respuesta es no: las jerarquías espacial y temporal determinísticas tienen espacios arbitrariamente grandes entre ellas.

Decimos que una proposición con parámetro  $n$  es verdadera *casi en todas partes* (a.e., por las siglas en inglés: *almost everywhere*) si es verdadera para todos los valores de  $n$ , excepto para un número finito de ellos. Decimos que una proposición es verdadera *infinitamente frecuente* (i.o., por las siglas en inglés: *infinitely often*) si es verdadera para un número infinito de  $n$ s. Nótese que ambas proposiciones y su negación pueden ser verdaderas i.o.

**Lema 12.3** Si  $L$  es aceptado por una TM  $M$  con límite espacial a.e.  $S(n)$ , entonces  $L$  es aceptado por una TM con límite espacial  $S(n)$ .

*Demostración* Utilícese el control finito para aceptar o rechazar cadenas de longitud  $n$  para el número finito de  $n$ , en donde  $M$  no tiene límite  $S(n)$ . Adviértase que la

construcción no es efectiva, ya que en ausencia de un límite temporal no podemos decir cuál de estas palabras son aceptadas por  $M$ .  $\square$

**Lema 12.4** Existe un algoritmo para determinar, dadas la TM  $M$ , una entrada de longitud  $n$  y un entero  $m$ , si  $m$  es el máximo número de celdas de cinta utilizadas por  $M$ , sobre alguna entrada de longitud  $n$ .

*Demostración* Para cada  $m$  y  $n$  existe un límite  $t$  sobre el número de movimientos que  $M$  puede efectuar sobre una entrada de longitud  $n$ , sin utilizar más de  $m$  celdas de cualquier cinta de almacenamiento o repitiendo una ID. Simúlense todas las secuencias que tengan más de  $t$  movimientos, comenzando con cada entrada de longitud  $n$ .  $\square$

**Teorema 12.13 (Teorema de Espacio de Borodin).** Dada cualquier función totalmente recursiva  $g(n) \geq n$ , existe una función totalmente recursiva  $S(n)$  tal que  $\text{ESPACIOD}(S(n)) = \text{ESPACIOD}(g(S(n)))$ . En otras palabras, existe un “hueco” o espacio entre los límites espaciales  $S(n)$  y  $g(S(n))$ , dentro del cual se encuentra la complejidad espacial mínima de ningún lenguaje.

*Demostración* Sean  $M_1, M_2, \dots$  una enumeración de TMs. Sea  $S_i(n)$  el número máximo de celdas de cinta utilizadas por  $M_i$  sobre cualquier entrada de longitud  $n$ . Si  $M_i$  siempre se detiene, entonces  $S_i(n)$  es una función total y es la complejidad espacial de  $M_i$ , pero si  $M_i$  no se detiene sobre alguna entrada de longitud  $n$ , entonces  $S_i(n)$  queda indefinida.<sup>†</sup> Construimos  $S(n)$  de forma tal que para cada  $k$  se cumple que

1.  $S_k(n) \leq S(n)$  a.e., o
2.  $S_k(n) \geq g(S(n))$  i.o

Esto es, ninguna  $S_k$  se encuentra entre  $S_k(n)$  y  $g(S(n))$  para casi toda  $n$ .

Al construir  $S(n)$  para algún valor de  $n$ , restringimos nuestra atención al conjunto finito de TMs  $M_1, M_2, \dots, M_n$ . El valor para  $S_n$  se selecciona de manera que, para ninguna  $i$  entre 1 y  $n$ ,  $S_i$  se encuentre entre  $S(n)$  y  $g(S(n))$ . Si pudiéramos calcular el valor finito más grande de  $S_i(n)$  para  $1 \leq i \leq n$ , entonces podríamos establecer  $S(n)$  igual a dicho valor. Sin embargo, puesto que algunas  $S_i(n)$  están indefinidas, no podemos calcular el valor más grande. En lugar de esto, hacemos de manera inicial  $j = 1$  y vemos si existe alguna  $M_j$  en nuestro conjunto finito para la cual  $S_j(n)$  se encuentre entre  $j + 1$  y  $g(j)$ . Si existe alguna  $S_j(n)$  con tales características, entonces asignamos  $j$  a  $S_j(n)$  y repetimos el proceso. Si no, hágase  $S(n)$  igual a  $j$  y hemos terminado. Como existe un número finito de TMs bajo consideración, y según el Lema 12.4, podemos decir si  $S(n) = m$ , para cualquier  $m$  fija, el proceso, en algún momento, calculará un valor para  $j$  tal que, para  $1 \leq i \leq n$ , se cumple que  $S_i(n) \leq j$  o  $S_i(n) > g(j)$ . Asígnese este valor de  $j$  a  $S(n)$ .

Supóngase que existe algún lenguaje  $L$  en  $\text{ESPACIOD}(g(S(n)))$  pero que no está en  $\text{ESPACIOD}(S(n))$ . Entonces  $L = L(M_k)$  para alguna  $k$  en donde  $S_k(n) \leq g(S(n))$ , para toda  $n$ . Mediante la construcción de  $S(n)$ , para toda  $n \geq k$ ,  $S_k(n) \leq S(n)$ . Esto es,  $S_k(n)$

<sup>†</sup> Identificamos un valor no definido con el infinito, de modo que un valor no definido es mayor que cualquier valor definido.

$\leq S(n)$  a.e., y de aquí que, según el Lema 12.3,  $L$  está en  $\text{ESPACIOD}(S(n))$ , lo que es una contradicción. Concluimos que  $\text{ESPACIOD}(S(n)) = \text{ESPACIOD}(g(S(n)))$ .  $\square$

El Teorema 12.13 y su análogo para las otras tres medidas de complejidad tienen un cierto número de consecuencia altamente intuitivas, como las que damos a continuación.

**Ejemplo 12.7** Existe una función totalmente recursiva  $f(n)$  tal que

$$\text{TIEMPOD}(f(n)) = \text{TIEMPON}(f(n)) = \text{ESPACIOD}(f(n)) = \text{ESPACION}(f(n)).$$

Es claro que  $\text{TIEMPOD}(f(n))$  está contenido en  $\text{TIEMPON}(f(n))$  y  $\text{ESPACIOD}(f(n))$ . De manera similar, tanto  $\text{TIEMPON}(f(n))$  como  $\text{ESPACIOD}(f(n))$  están contenidas en  $\text{ESPACION}(f(n))$ . Según el Teorema 12.10 para toda  $f(n) \geq \log_2 n$ , si  $L$  está en  $\text{ESPACION}(f(n))$ , entonces existe una constante  $c$ , que depende solamente de  $L$ , tal que  $L$  se encuentra en  $\text{TIEMPOD}(c^{f(n)})$ . Por consiguiente,  $L = L(M)$  para alguna TM  $M$  cuya complejidad temporal está limitada por arriba por  $f(n)^{f(n)}$  a.e. Para el análogo del TIEMPOD del Lema 12.3,  $L$  está en  $\text{TIEMPOD}(f(n)^{f(n)})$ . Finalmente, el análogo de TIEMPOD del Teorema 12.13, con  $g(x) = x^x$ , establece la existencia de  $f(n)$  para la cual  $\text{TIEMPOD}(f(n)) = \text{TIEMPOD}(f(n)^{f(n)})$ , lo que demuestra el resultado.

De manera similar, si se tienen dos modelos universales de cálculo, pero uno de ellos es muy simple y lento, digamos una máquina de Turing que hace un movimiento por siglo, y el otro es muy rápido, digamos una máquina de acceso aleatorio con poderosas instrucciones de construcción para la multiplicación, exponentiación, etcétera, que efectúa un millón de operaciones por segundo, se puede demostrar fácilmente que existe una  $T(n)$  totalmente recursiva tal que cualquier función calculable en el tiempo  $T(n)$  en un modelo, es calculable en un tiempo  $T(n)$  en el otro.

### Teorema de la aceleración

Otro fenómeno curioso con respecto a las medidas de complejidad es el que se refiere a que existen funciones que no tienen mejores programas (máquinas de Turing). Ya hemos visto que cada TM permite una aceleración lineal en el tiempo y una compresión lineal en el espacio. Demostraremos ahora que existen lenguajes que no tienen un “mejor” programa. Esto es, reconocedores para esos lenguajes que puedan acelerarse indefinidamente. Trabajaremos solamente con el espacio y demostraremos que existe un lenguaje  $L$  tal que, para cualquier máquina de Turing que acepta a  $L$ , siempre existe otra máquina de Turing que acepta a  $L$  y utiliza, por ejemplo, solamente la raíz cuadrada del espacio utilizado por la primera. Por supuesto que este nuevo reconocedor puede ser sustituido por un reconocedor aún más rápido y así sucesivamente, *ad infinitum*.

La idea fundamental de la demostración es muy sencilla. Mediante diagonalización se construye  $L$ , de modo que  $L$  no pueda ser reconocida rápidamente por cualquier máquina “pequeña”, ésto es, una máquina con un índice entero pequeño que la codifica. Conforme aumentan los índices de la máquina, el proceso de diagonalización permite un reconocimiento cada vez más rápido de  $L$ . Dada cualquier máquina que reconozca a  $L$ , ésta tiene algún índice fijo y por tanto puede reconocer a  $L$  solamente con la

rapidez dada por el índice. Sin embargo, las máquinas con índices más grandes pueden reconocer a  $L$  de manera arbitrariamente más rápida.

**Teorema 12.14** (*Teorema de Aceleración de Blum*) Sea  $r(n)$  cualquier función totalmente recursiva. Existe un lenguaje recursivo  $L$  tal que, para cualquier máquina de Turing  $M_i$  que acepta a  $L$ , existe una máquina de Turing  $M_j$  que acepta a  $L$  y tal que  $r(S_j(n)) \leq h(n)$ , para casi toda  $n$ .

**Demostración** Sin pérdida de generalidad, supóngase que  $r(n)$  es una función monótonamente no decreciente y espacialmente construible de manera completa, con  $r(n) \geq n^2$  (véase Ejercicio 12.9). Defínase  $h(n)$  mediante

$$h(1) = 2, \quad h(n) = r(h(n-1)).$$

Entonces  $h(n)$  es una función espacialmente construible de manera completa, como el lector puede demostrar fácilmente.

Sea  $M_1, M_2, \dots$  una enumeración de todas las TMs fuera de línea análoga a la de la Sección 8.3 para TMs de una sola cinta. En particular, suponemos que el código para  $M_i$  tiene longitud  $\log_2 i$ . Construimos  $L$  de manera que

1. si  $L(M_i) = L$ , entonces  $S_i(n) \geq h(n-i)$  a.e.;
2. para cada  $k$ , existe una máquina de Turing  $M_j$  tal que  $L(M_j) = L$  y  $S_j(n) \leq h(n-k)$ .

Las condiciones dadas arriba sobre  $L$ , nos asegura que, para cada  $M_i$  que acepta a  $L$ , existe una  $M_j$  que acepta a  $L$  con

$$S_i(n) \geq r(S_j(n)) \quad \text{a.e.}$$

Para ver lo anterior, selecciónese  $M_j$  de forma tal que  $S_j(n) \leq h(n-i-1)$ . Según (2),  $M_j$  existe. Entonces según (1),

$$S_i(n) \geq h(n-i) = r(h(n-i-1)) \geq r(S_j(n)) \quad \text{a.e.}$$

Construyamos ahora  $L \subseteq 0^*$  que satisface a (1) y (2). Para  $n = 0, 1, 2, \dots$  en turno, especificamos si  $0^n$  se encuentra en  $L$ . Durante el proceso, ciertas  $M_i$  son señaladas como "canceladas". Una TM cancelada seguramente no acepta a  $L$ . Sea  $\sigma(n)$  el menor entero  $j \leq n$ , tal que  $S_j(n) < h(n-j)$  y  $M_j$  no es cancelada por  $i = 0, 1, \dots, n-1$ . Cuando consideramos a  $n$ , si  $\sigma(n)$  existe,  $M_{\sigma(n)}$  se señala como cancelada. Entonces se coloca en  $L$  a  $0^n$  si y sólo si  $\sigma(n)$  existe y  $0^n$  no es aceptado por  $M_{\sigma(n)}$ .

En seguida demostraremos que  $L$  satisface la condición (1), a saber: si  $L(M_i) = L$ , entonces  $S_i(n) \geq h(n-i)$  a.e. Sea  $L(M_i) = L$ . Al construir  $L$ , todas las TMs  $M_j$ , para  $j < i$ , que nunca son canceladas, se cancelan después de considerar algún número finito de  $n$ s, digamos arriba de  $n_0$ . Nótese que  $n_0$  no puede ser calculado de manera efectiva, pero sin embargo existe. Supóngase que  $S_i(n) < h(n-i)$  para alguna  $n > \max(n_0, i)$ . Cuando consideramos a  $n$ , ninguna  $M_j$ ,  $j < i$ , se cancela. Por consiguiente  $\sigma(n) = i$ , y  $M_i$  sería cancelada si no fue cancelada previamente. Pero una TM que es cancelada es seguro que no acepta a  $L$ . Por tanto  $S_i(n) \geq h(n-i)$  para  $n > \max(n_0, i)$ , es decir,  $S_i(n) \geq h(n-i)$  a.e.

Para demostrar la condición (2) mostraremos que existe, para una  $k$  dada, una TM  $M = M_j$  tal que  $L(M) = L$  y  $S_j(n) \leq h(n-k)$ , para toda  $n$ . Para determinar si  $0^n$  se encuentra en  $L$ ,  $M$  debe simular a  $M_{\sigma(n)}$  sobre  $0^n$ . Para saber de qué  $\sigma(n)$  se trata,  $M$  debe determinar cuáles  $M_i$ s ya han sido canceladas por  $0^i$  para  $i < n$ . Sin embargo, la construcción de la lista de las TMs canceladas directamente requiere el ver si  $M_i$  utiliza más de un espacio  $h(i)$ , para  $0 \leq i \leq n$  y  $1 \leq i \leq n$ . Para  $i < k + \ell - n$ , esto requiere de más de un espacio  $h(n-k)$ .

La solución consiste en observar que cualquier TM  $M_i$ ,  $i \leq k$ , que nunca se cancela, es cancelada cuando consideramos alguna  $\ell$  menor que una  $n_1$  particular. Para cada  $\ell \leq n_1$ , incorpórese dentro del control finito de  $M$  si  $0^\ell$  se encuentra en  $L$ , y también incorpórese una lista de todas las TMs  $M_i$  canceladas por cualquier  $\ell \leq n_1$ . Por consiguiente, ningún espacio en absoluto es necesario para  $M$ , si  $n \leq n_1$ . Si  $n > n_1$ , para calcular  $\sigma(n)$  y simular  $M_{\sigma(n)}$  sobre  $0^n$ , solamente será necesario simular a las TMs  $M_i$  sobre la entrada  $0^\ell$ , en donde  $n_1 < \ell \leq n$  y  $k < i \leq n$ , para ver si  $M_i$  es cancelada por  $\ell$ .

Para probar si  $M_i$  es cancelada por  $\ell$ , sólo necesitamos simular  $M_i$  utilizando  $h(\ell-i)$  celdas de  $M_i$ , que es menor que  $h(n-k)$ , como  $\ell \leq n$  e  $i > k$ . Puesto que  $n > n_1$ , debe darse que  $\sigma(n)$ , si existe, sea mayor que  $k$ . Por consiguiente, simular  $M_{\sigma(n)}$  sobre  $0^n$  toma  $h(n-\sigma(n))$  celdas de  $M_{\sigma(n)}$ , que es menor que  $h(n-k)$  celdas.

Por último, debemos demostrar que  $M$  puede hacerse operar dentro de un espacio  $h(n-k)$ . Solamente necesitamos simular las TMs  $M_i$ , para  $k < i \leq n$ , sobre las entradas  $0^\ell$ ,  $n_1 \leq \ell \leq n$ , para ver si son canceladas, de modo que necesitamos representar no más de  $h(n-k-1)$  celdas de la cinta de  $M_i$ , para cualquier simulación. Puesto que  $i \leq n$ , el código entero para  $M_i$  tiene longitud no mayor de  $\log_2 n$ . Por consiguiente cualquier símbolo de cinta de  $M_i$  puede ser codificado utilizando  $\log_2 n$  celdas de  $M$ . Como  $r(x) \geq x^2$ , sabemos que  $h(x) \leq 2^{2x}$ . También, por la definición de  $h$ ,  $h(n-k) \geq [h(n-k-1)]^2 \geq 2^{2n-k-1}h(n-k-1)$ . Como  $2^{2n-k-1} \geq \log_2 n$  a.e., un espacio  $h(n-k)$  es suficiente para la simulación, para casi toda  $n$ .

Además del espacio que se necesita para simular a las TMs, se necesita espacio para mantener la lista de las TMs canceladas. Esta lista consiste en cuando mucho  $n$  TMs, cada una con un código de longitud de como máximo  $\log_2 n$ . El espacio  $n \log n$  que se necesita para mantener la lista de las TMs canceladas es también menor que  $h(n-k)$  a.e. Según el Lema 12.3,  $M$  puede ser modificada para que reconozca palabras  $0^n$ , en donde  $n \log_2 n < h(n-k)$  o  $2^{2n-k-1} < \log_2 n$ , en su control finito. La TM que resulta es de complejidad espacial  $h(n-k)$  para toda  $n$ , y es la  $M$  que se desea.  $\square$

### Teorema de unión

El último teorema de esta sección, conocido como *teorema de unión*, tiene que ver con la asignación de los nombres de las clases de complejidad. Como medio de introducción, sabemos que cada polinomio, como  $n^2$  o  $n^3$ , define una clase de complejidad espacial (tanto como las clases de complejidad de los otros tres tipos). Sin embargo, ¿forma el espacio polinomial una clase de complejidad? Es decir, ¿existe una  $S(n)$  tal que  $\text{ESPACIOD}(S(n))$  contiene a todos los conjuntos reconocibles en un límite espacial polinomial y no a otros conjuntos? Es claro que  $S(n)$  debe ser casi en todas partes mayor que cualquier polinomio, pero también debe ser lo suficientemente

pequeña para que uno no pueda ajustar otra función que sea el espacio utilizado por alguna TM que está entre la función y los polinomios, en donde “ajustar” debe tomarse como un término técnico cuyo significado se define de manera precisa en el siguiente teorema.

**Teorema 12.15** Sea  $\{f_i(n) \mid i = 1, 2, \dots\}$  una colección recursivamente enumerable de funciones recursivas. Esto es, existe una TM que enumera una lista de TMs, la primera calcula a  $f_1$ , la segunda a  $f_2$ , y así sucesivamente. También supóngase que para cada  $i$  y  $n$ ,  $f_i(n) < f_{i+1}(n)$ . Entonces existe una  $S(n)$  recursiva tal que

$$\text{ESPACIOD}(S(n)) = \bigcup_{i \geq 1} \text{ESPACIOD}(f_i(n)).$$

*Demostración* Construimos una función  $S(n)$  que satisface las siguientes dos condiciones:

1. Para cada  $i$ ,  $S(n) \geq f_i(n)$  a.e.
2. Si  $S_j(n)$  es la complejidad espacial exacta de alguna TM  $M_j$ , y, para cada  $i$ ,  $S_j(n) > f_i(n)$  i.o., entonces  $S_j(n) > S(n)$ , para alguna  $n$  (y de hecho para un número infinito de  $n$ s).

La primera condición nos asegura que

$$\bigcup_i \text{ESPACIOD}(f_i(n)) \subseteq \text{ESPACIOD}(S(n)).$$

La segunda condición asegura que  $\text{ESPACIOD}(S(n))$  contiene solamente a aquellos conjuntos que se encuentran en  $\text{ESPACIOD}(f_i(n))$ , para alguna  $i$ . Juntas, las condiciones implican que

$$\text{ESPACIOD}(S(n)) = \bigcup_i \text{ESPACIOD}(f_i(n)).$$

Haciendo  $S(n) = f_n(n)$  nos asegura que se cumple con la condición (1). Sin embargo, puede no satisfacer la condición (2). Puede existir una TM  $M_j$  cuya complejidad espacial  $S_j(n)$  sea mayor que cada  $f_i(n)$  i.o., pero menor que  $f_n(n)$ , para toda  $n$ . Por consiguiente pueden existir conjuntos en  $\text{ESPACIOD}(f_n(n))$  que no se encuentren en  $\bigcup_i \text{ESPACIOD}(f_i(n))$ . Para salvar este problema construimos a  $S(n)$  de modo que ésta se meta debajo de cada  $S_j(n)$  que sea i.o., mayor que cada  $f_i(n)$ , y, de hecho,  $S(n)$  se meterá debajo de  $S_j(n)$  para una infinidad de  $n$ s. Esto se lleva a cabo adivinando para cada TM  $M_j$ , una  $i_j$ , tal que  $f_{i_j}(n) \geq S_j(n)$  a.e. La “adivinación” no es determinística; más bien está sujeta a una revisión determinística de la manera siguiente. Si en algún punto descubrimos que la suposición no es correcta, suponemos un valor mayor para  $i_j$ , y, para alguna  $n$  particular, definimos  $S(n)$  como la menor  $S_j(n)$ . Si sucede que  $S_j$  crece más rápido que cualquier  $f_i$ ,  $S$  será infinitamente más frecuente menor que  $S_j$ . Por otro lado, si alguna  $f_i$  es casi en todos lados mayor que  $S_j$ , en algún momento supondremos una de tales  $f_i$  y se detiene asignando valores de  $S$  menores de  $S_j$ .

En la Fig. 12.6 damos un algoritmo que genera a  $S(n)$ . Se mantiene una lista, llamada LISTA de “adivinaciones”, de la forma “ $i_j = k$ ” para varios enteros  $j$  y  $k$ . Para cada  $j$ , habrá cuando mucho una suposición  $k$  sobre la LISTA en cualquier momento. Como se tiene en el teorema anterior,  $M_1, M_2, \dots$  es una enumeración de todas las TMs fuera de línea,  $S_j(n)$  es la cantidad máxima de espacio utilizado por  $M_j$  sobre cualquier entrada de longitud  $n$ . Recuérdese que  $S_j(n)$  puede estar indefinida (infinita) para algunos valores de  $n$ .

```

begin
1. LISTA := lista vacía
2. for n = 1, 2, 3, ... do
3.   if para toda "i_j = k" en LISTA, f_k(n) ≥ S_j(n) then
4.     añádase "i_j = n" a LISTA y defínase S(n) = f_n(n)
      else
        begin
5.          entre todas las adivinaciones de LISTA tales que f_k(n) < S_j(n), sea "i_j = k" la adivinación
            con la menor k y, dada esa k, la menor j;
6.          defínase S(n) = f_k(n);
7.          sustitúyase "i_j = k" por "i_j = n" en LISTA;
8.          añádase "i_j = n" a LISTA
      end
end

```

Fig. 12.6 Definición de  $S(n)$ .

Para demostrar que

$$\text{ESPACIOD}(S(n)) = \bigcup_i \text{ESPACIOD}(f_i(n))$$

primero demostraremos que  $S(n)$  satisface las condiciones (1) y (2). Considérese la condición (1). Para ver que, para cada  $m$ ,  $S(n) \geq f_m(n)$  a.e., obsérvese que a  $S(n)$  se le asigna un valor solamente en las líneas (4) y (6) de la Fig. 12.6. Siempre que  $S(n)$  esté definida en la línea (4), para  $n \geq m$ , el valor de  $S(n)$  es de al menos  $f_m(n)$ . Por consiguiente, para los valores de  $S(n)$  definidos en la línea (4),  $S(n) \geq f_m(n)$ , excepto para el conjunto finito de  $n$  menores que  $m$ . Ahora considérese los valores de  $S(n)$  definidos en la línea (6). Cuando  $n$  alcanza a  $m$ , LISTA tendrá algún número finito de adivinaciones. Cada una de éstas puede, de manera subsecuente, ocasionar que un valor de  $S(n)$ , para alguna  $n > m$ , sea menor que  $f_m(n)$ . Sin embargo, cuando esto llega a suceder la línea (7) hace que dicha adivinación sea sustituida por una adivinación “ $i_j = p$ ”, para alguna  $p \geq m$ , y esta adivinación, si se le selecciona en la línea (5), no ocasiona que  $S(n)$  sea menor que  $f_m(n)$ , ya que  $f_p(n) \geq f_m(n)$ , siempre que  $p \geq m$ . Por tanto, de la línea (6), existen solamente un número finito de  $n$  mayores que  $m$  (cuando mucho la longitud de LISTA cuando  $n = m$ ), para los cuales  $S(n) < f_m(n)$ . Puesto que sólo existe un número finito de  $n$ s menores que  $m$ ,  $S(n) \geq f_m(n)$  a.e.

En seguida debemos demostrar la condición (2), que establece que si existe una TM  $M_j$  tal que, para cada  $i$ ,  $S_j(n) > f_i(n)$  i.o., entonces  $S_j(n') > S(n')$ , para un número infinito de  $n'$ . En todo momento, después de  $n = j$ , LISTA tendrá una adivinación para  $i_j$ , y LISTA es siempre finita. Para  $n = j$  colocamos a " $i_j = j$ " en LISTA. Como  $S_j(n) > f_i(n)$  i.o., existirá un número arbitrariamente grande de valores subsecuentes de  $n$  para los cuales no vale la condición del paso (3). En cada uno de estos momentos, o nuestra adivinación " $i_j = j$ " es seleccionada en la línea (5) o se selecciona alguna otra adivinación del número finito que se encuentra en LISTA, cuando  $n = j$ . En el último caso, esa suposición es sustituida por " $i_p = q$ ", con  $q > j$ . Todas las suposiciones que se agregaron a LISTA son también de la forma " $i_p = q$ ", para  $q > j$ , así que, en algún momento, nuestra " $i_j = j$ " será seleccionada en el paso (5), y para este valor de  $n$  tenemos  $S_j(n) \geq f_i(n) = S(n)$ . Por consiguiente la condición (2) es verdadera.

Por último, debemos demostrar que las condiciones (1) y (2) implican

$$\text{ESPACIOD}(S(n)) = \bigcup_i \text{ESPACIOD}(f_i(n)).$$

Supóngase que  $L$  se encuentra en  $\bigcup_i \text{ESPACIOD}(f_i(n))$ . Entonces  $L$  se encuentra en  $\text{ESPACIOD}(f_m(n))$ , para alguna  $m$  particular. Según la condición (1),  $S(n) \geq f_m(n)$  a.e., por tanto, según el lema 12.3,  $L$  está en  $\text{ESPACIOD}(S(n))$ . Supóngase ahora que  $L$  se encuentra en  $\text{ESPACIOD}(S(n))$ . Sea  $L = L(M_j)$ , en donde  $S_j(n) \leq S(n)$ , para toda  $n$ . Si para ninguna  $i$ ,  $L$  está en  $\text{ESPACIOD}(f_i(n))$ , entonces, según el lema 12.3, para cada  $i$ , cada TM  $M_k$  que acepta a  $L$  tiene  $S_k(n) \geq f_i(n)$  i.o. Por tanto, según la condición (2), existe alguna  $n$  para la cual  $S_k(n) > S(n)$ . Si hacemos  $k = j$  se produce una contradicción.  $\square$

**Ejemplo 12.8** Sea  $f_i(n) = n^i$ . Entonces seguramente que podemos enumerar una secuencia de TMs  $M_1, M_2, \dots$  tal que  $M_i$ , presentada con la entrada  $0^n$ , escribe  $0^{n^i}$  sobre su cinta y se detiene. Por consiguiente, el Teorema 12.15 dice que existe alguna  $S(n)$  tal que

$$\text{ESPACIOD}(S(n)) = \bigcup_i \text{ESPACIOD}(n^i).$$

Como cualquier polinomio  $p(n)$  es menor o igual que alguna  $n^i$  a.e.,  $\text{ESPACIOD}(S(n))$  es la unión sobre todos los polinomios  $p(n)$  de  $\text{ESPACIOD}(p(n))$ . Esta unión, que en el próximo capítulo llamaremos ESPACIOP, y que juega un papel clave en la teoría de los problemas no tratables, es, por consiguiente, considerado como una clase de complejidad espacial determinística.

## 12.7 TEORIA DE COMPLEJIDAD AXIOMATICA

El lector puede haber observado que muchos teoremas de este capítulo no dependen del hecho de que estemos midiendo la cantidad de tiempo o espacio utilizado, sino sólo en el hecho de que medimos algún recurso que se está consumiendo conforme el cálculo se efectúa. De hecho, uno podría postular axiomas que gobiernan los recursos y dar un desarrollo axiomático completo de la teoría de la complejidad. En esta sección hacemos un breve bosquejo de este planteamiento.

## Los axiomas de Blum

Sea  $M_1, M_2, \dots$  una enumeración de máquinas de Turing que definen entre ellas a cada función parcialmente recursiva. Debido a razones técnicas, consideramos a las  $M_i$  como calculadoras de las funciones parcialmente recursivas  $\phi_i$ , más que como reconocedoras de conjuntos. La razón es que, desde el punto de vista de la notación, es más sencillo medir la complejidad como función de la entrada más que de la longitud de la entrada. Sea  $\phi_i(n)$  la función de una variable calculada por  $M_i$  y hagamos que  $\Phi_1(n), \Phi_2(n), \dots$  sea un conjunto de funciones parcialmente recursivas que satisfacen a los dos siguientes axiomas (*axiomas de Blum*).

**Axioma 1**  $\Phi_i(n)$  está definida si y sólo si  $\phi_i(n)$  está definida.

**Axioma 2** La función  $R(i, n, m)$  definida como 1 si  $\Phi_i(n) = m$  y 0 en cualquier otro caso, es una función totalmente recursiva.

La función  $\Phi_i(n)$  da la complejidad del cálculo de la  $i$ -ésima máquina de Turing sobre la entrada  $n$ . El Axioma 1 requiere que  $\Phi_i(n)$  esté definida si y sólo si la  $i$ -ésima máquina de Turing se detiene en la entrada  $n$ . Por consiguiente una  $\Phi_i$  posible sería el número de pasos de la  $i$ -ésima máquina de Turing. La cantidad de espacio utilizado es otra alternativa, siempre y cuando definamos el espacio utilizado como infinito si la TM entra en un ciclo.

El Axioma 2 requiere que podamos determinar si la complejidad de la  $i$ -ésima máquina de Turing sobre la entrada  $n$  es  $m$ . Por ejemplo, si nuestra medida de complejidad es el número de pasos que se llevan en el cálculo, entonces, dados  $i, n$  y  $m$ , podemos simular  $M_i$  sobre  $0^n$  para  $m$  pasos y ver si se detiene. El Lema 12.4 y su análogo representan exigencias que el Axioma 2 cumple para las cuatro medidas con las cuales hemos estado tratando.

**Ejemplo 12.9** La complejidad espacial determinística satisface los axiomas de Blum, siempre y cuando digamos que  $\Phi_i(n)$  está indefinida si  $M_i$  no se detiene sobre la entrada  $0^n$ , aun cuando el espacio utilizado por  $M_i$  sobre  $0^n$  pueda ser limitado. La complejidad temporal determinística, de manera similar, satisface los axiomas de Blum si decimos que  $\Phi_i(n)$  está indefinida cuando  $M_i$  corre sin detenerse o se detiene sin que haya ningún  $0^k$  sobre su cinta. Para calcular  $R(i, n, m)$ , simplemente simúlense  $M_i$  para  $m$  pasos sobre la entrada  $0^n$ .

Podemos establecer que el tiempo y el espacio no determinísticos satisfacen los axiomas si hacemos una definición inteliente de lo que significa, para una NTM, calcular una función. Por ejemplo, podemos decir que  $\phi_i(n) = j$  si y sólo si existe alguna sucesión de alternativas para  $M_i$  con entrada  $0^n$  que se detiene con  $0^j$  sobre la cinta, y no existe una sucesión de alternativas que lleve a detenerse con algún  $0^k$ ,  $k \neq j$ , sobre la cinta.

Si definimos  $\Phi_i(n) = \phi_i(n)$ , no satisfacemos el Axioma 2. Supóngase que  $R(i, n, m)$  es recursiva. Entonces existe un algoritmo para decir si  $M_i$  con entrada  $0^n$  se detiene con  $0^m$  sobre su cinta. Dada cualquier TM  $M$ , podemos construir  $\hat{M}$  para simular a  $M$ . Si  $M$  se detiene con cualquier cinta,  $\hat{M}$  borra su propia cinta. Si  $i$  es un índice para  $\hat{M}$ ,

entonces  $R(i, n, 0)$  es verdadera si y sólo si  $M$  se detiene sobre la entrada  $0^n$ . Por consiguiente si  $R(i, n, m)$  fuera recursiva, podríamos decir si una TM  $M$  dada se detiene sobre una entrada dada, lo cual es no resoluble (véase el Ejercicio 8.3).

### Relaciones recursivas entre las medidas de complejidad

Muchos de los teoremas sobre la complejidad pueden demostrarse únicamente a partir de los dos axiomas. En particular, el hecho de que existen funciones arbitrariamente complejas, el teorema de la aceleración, el teorema de espacio, y el de la unión pueden demostrarse de esta manera. Demostraremos solamente un teorema aquí para ilustrar las técnicas. El teorema que seleccionamos es el que establece que todas las medidas están relacionadas de manera recursiva. Esto es, dadas dos medidas de complejidad,  $\Phi$  y  $\hat{\Phi}$ , existe una función totalmente recursiva  $r$  tal que la complejidad de la TM  $M_i$  en una medida,  $\hat{\Phi}_i(n)$ , es cuando mucho  $r(n, \Phi_i(n))$ . Por ejemplo, los Teoremas 12.10 y 12.11 establecen que, para las cuatro medidas de complejidad con las cuales hemos estado tratando, cuando mucho una función exponencial relacionaba a cualquier par de tales medidas de complejidad. En cierto sentido, las funciones que son fáciles en una medida son “fáciles” en cualquier otra medida, aunque el término “fácil” debe entenderse moderadamente, porque  $r$  podría ser una función que crece muy rápido, como la función de Ackermann.

**Teorema 12.16** Sean  $\Phi$  y  $\hat{\Phi}$  dos medidas de complejidad. Entonces existe una función recursiva  $r$  tal que, para toda  $i$ ,

$$r(n, \Phi_i(n)) \geq \hat{\Phi}_i(n) \quad \text{a.e.}$$

**Demostración** Sea

$$r(n, m) = \max_i \{\hat{\Phi}_i(n) \mid i \leq n \quad \text{y} \quad \Phi_i(n) = m\}.$$

La función  $r$  es recursiva, ya que  $\Phi_i(n) = m$  puede ser probada con el Axioma 2. En el caso de que sea igual a  $m$ , entonces  $\phi_i(n)$  y  $\hat{\Phi}_i(n)$  deben estar definidas, según el Axioma 1, y 2, en consecuencia el máximo puede ser calculado. Es claro que  $r(n, \Phi_i(n)) \geq \hat{\Phi}_i(n)$  para toda  $n \geq i$ , puesto que, para  $n \geq i$ ,  $r(n, \hat{\Phi}_i(n))$  es cuando menos  $\hat{\Phi}_i(n)$ .  $\square$

Aunque el planteamiento axiomático es elegante y nos permite demostrar los resultados en un marco más general, no consigue capturar al menos un aspecto que es importante de nuestra noción intuitiva de complejidad. Si construimos una máquina de Turing  $M_k$  que primero ejecute  $M_i$  sobre  $n$  y después ejecute  $M_j$  sobre el resultado, esperaríamos que la complejidad de  $M_k$  sea sobre  $n$  fuera de al menos la de  $M_i$  sobre  $n$ . Sin embargo, existen medidas de complejidad en las que éste no es el caso. En otras palabras, haciendo un cálculo podemos reducir la complejidad de lo que ya hemos hecho. Dejamos la construcción de una medida de complejidad tal como ejercicio.

### EJERCICIOS

**12.1** El concepto de secuencia cruzada—la secuencia de estados en la que el límite entre dos celdas está cruzado—se definió en la Sección 2.6, en relación con los autómatas finitos de dos direcciones. Sin embargo, el concepto se aplica correctamente a las TMs de una sola cinta. Demuestre las siguientes propiedades básicas de las secuencias cruzadas.

- a) El tiempo que le toma a una TM de una sola cinta  $M$  sobre la entrada  $w$  es la suma de las longitudes de las secuencias cruzadas entre cada dos celdas de la cinta de  $M$ .
- b) Supóngase que  $M$  es una TM de una sola cinta que, si acepta su entrada, lo hace a la derecha de las celdas sobre las cuales su entrada fue escrita originalmente. Muestre que si  $M$  acepta a la entrada  $w_1 w_2$ , y la secuencia cruzada entre  $w_1$  y  $w_2$  es la misma que entre  $x_1$  y  $x_2$  cuando  $M$  tiene la entrada  $x_1 x_2$ , entonces  $M$  acepta a  $x_1 w_2$ .

**\*12.2** Utilice el Ejercicio 12.1 para mostrar que los lenguajes

- S a)  $\{w^i w^k \mid w \text{ se encuentra en } (a+b)^*\}$     b)  $\{wcw \mid w \text{ se encuentra en } (a+b)^*\}$

cada uno requiere de  $kn^2$  pasos sobre alguna entrada de longitud impar lo suficientemente grande  $n$ , para alguna constante  $k > 0$ . Por consiguiente el límite del Teorema 12.5 es en un sentido el mejor posible.

- \*12.3** El concepto de secuencia cruzada puede adaptarse a las TMs fuera de línea si sustituimos el concepto de “estado” por el estado, contenido de las cintas de almacenamiento y posiciones de la cabeza de las cintas de almacenamiento. El Teorema 12.8, la jerarquía espacial, se refería solamente a complejidades espaciales de  $\log n$  o más. Demuestre que lo mismo es válido para la  $S_2(n)$  espacialmente construible de manera completa que está por debajo de  $\log n$ . [Sugerencia: Utilizando un argumento general basado en las secuencias cruzadas, muestre que  $\{wcw \mid w \text{ se encuentra en } (a+b)^* \text{ y } |w| = 2^{s^2(i+2)^{kw^2}}\}$  se encuentra en  $\text{ESPACIOD}(S_2(n))$  pero no en  $\text{ESPACIOD}(S_1(n))$ .]

- \*12.4** Muestre, utilizando argumentos generales basados en secuencias cruzadas, que si  $L$  no es un conjunto regular y  $L$  está en  $\text{ESPACIOD}(S(n))$ , entonces  $S(n) \geq \log \log n$  i.o. Muestre el mismo resultado  $\in$  para el espacio no determinístico. Por tanto, para los espacios determinístico y no determinístico existe un “hueco” entre  $1$  y  $\log \log n$ .

**12.5** Demuestre que el Lema 12.2, el “lema de traslación”, se aplica a

- a) espacio determinístico
- b) tiempo determinístico, y
- c) tiempo no determinístico.

**12.6** Muestre que  $\text{TIEMPOD}(2^{2^n} + n)$  incluye de manera propia a  $\text{TIEMPOD}(2^{2^n})$ .

**12.7** Muestre que  $\text{ESPACION}(c + \epsilon)^n$  incluye de manera propia a  $\text{ESPACION}(c^n)$  para cualquier  $c > 1$  y  $\epsilon > 0$ .

**12.8** ¿Cuál, si existe, es la relación entre cada uno de los siguientes pares de clases de complejidad?

- a)  $\text{ESPACIOD}(n^2)$  y  $\text{ESPACIOD}(f(n))$ , en donde  $f(n) = n$  para  $n$  impar y  $n^3$  para  $n$  par.
- b)  $\text{TIEMPOD}(2^n)$  y  $\text{TIEMPOD}(3^n)$
- c)  $\text{ESPACION}(2^n)$  y  $\text{ESPACIOD}(5^n)$
- d)  $\text{ESPACIOD}(n)$  y  $\text{TIEMPOD}(\lceil \log_2 n \rceil)$

**12.9** Muestre que si  $r$  es cualquier función totalmente recursiva, entonces existe una  $r'$  monótonamente no decreciente, espacialmente construible de manera completa, tal que  $r'(n) \geq r(n)$ , y  $r'(x) \geq x^2$  para todos los enteros  $x$ . [Sugerencia: Considérese la complejidad espacial de cualquier TM que calcule a  $r$ .]

**12.10** Muestre que existe una función totalmente recursiva  $S(n)$  tal que  $L$  está en  $\text{ESPACIOD}(S(n))$  si y sólo si  $L$  es aceptada por alguna TM con límite espacial  $c^n$ , para  $c > 1$ .

**12.11** Supóngase que utilizamos los axiomas para la teoría de complejidad computacional, como si perteneciera a los lenguajes más que a las funciones. Esto es, Sea  $M_1, M_2, \dots$  una enumeración de máquinas de Turing y  $L_i$  el lenguaje aceptado por  $M_i$ . Sustitúyase el Axioma 1 por:

Axioma 1':  $\Phi_i(n)$  está definida si y sólo si  $M_i$  se detiene en todas las entradas de longitud  $n$ . Demuestre de nuevo el Teorema 12.16 utilizando los Axiomas 1' y 2.

**12.12** Demuestre que los teoremas de aceleración y de espacio valen para ESPACION, TIEMPOD y TIEMPON. [Sugerencia: Utilícese el Teorema 12.16 y los teoremas de aceleración y espacio para ESPACIOD].

**12.13** Demuestre que las funciones siguientes son espacial y temporalmente construibles de manera completa:

- a)  $n^2$       b)  $2^n$       c)  $n!$

**12.14** Demuestre que las funciones siguientes son especialmente construibles de manera completa:

- a)  $\sqrt{n}$       b)  $\log_2 n$

**\*\*S 12.15** c) Alguna función que tenga límite superior  $\log_2 \log_2 n$  y cuyo límite inferior sea  $c \log_2 \log_2 n$ , para alguna  $c > 0$ , infinitamente frecuente.

Demuestre que si  $T_2(n)$  es temporalmente construible y

$$\inf_{n \rightarrow \infty} \frac{T_1(n) \log T_1(n)}{T_2(n)} = 0,$$

entonces existe un lenguaje aceptado por una TM de una sola cinta y límite temporal  $T_2(n)$ , pero no por una máquina de una sola cinta y límite temporal  $T_1(n)$ . [Sugerencia: Para simular una TM de una sola cinta  $M_i$  mediante una máquina de una sola cinta, muévese la descripción de  $M_i$  de forma que se encuentre siempre cerca de la cabeza de la cinta. De manera similar, llévese un "control" para decir cuando  $M_i$  ha excedido su límite temporal].

**\*\*12.16** Muestre que si  $T_2(n)$  es temporalmente construible y

$$\inf_{n \rightarrow \infty} \frac{T_1(n) \log^* T_1(n)}{T_2(n)} = 0,$$

entonces, para cada  $k$ , existe un lenguaje aceptado por una TM de  $k$  cintas y límite temporal  $T_2(n)$ , pero no lo es por ninguna TM de  $k$  cintas y límite temporal  $T_1(n)$ , en donde  $\log^*(m)$  es el número de veces que debemos tomar el logaritmo base dos de  $m$  para llegar a 1 o menos. Por ejemplo,  $\log^*(3) = 2$  y  $\log^*(2^{65536}) = 5$ . Nótese que este ejercicio implica al Ejercicio 12.15.

**12.17** Demuestre que, para cualquier medida de complejidad,  $\Phi$  que satisface los axiomas de Blum no puede existir una función totalmente recursiva  $f$  tal que  $\Phi_i(n) \leq f(n, \phi_i(n))$ . Esto es, uno no puede limitar la complejidad de una función en términos de su valor.

**\*\*12.18** El teorema de aceleración implica que, para funciones recursivas arbitrariamente grandes,  $r$ , podemos encontrar un lenguaje  $L$  para las cuales existe una secuencia de TMs  $M_1, M_2, \dots$ , cada una de las cuales acepta a  $L$ , tal que el espacio utilizado por  $M_i$  es al menos  $r$  aplicada al espacio utilizado por  $M_{i+1}$ . Sin embargo, no hemos dado un algoritmo para encontrar dicha secuencia; solamente demostramos que ésta debe existir. Demuestre que la aceleración no es efectiva en el sentido de que si, para cada TM que acepta a  $L$ , existe una

$M_i$  sobre la lista que utilice menos espacio, entonces la lista de TMs no es recursivamente enumerable.

**\*12.19** ¿Cuáles de las siguientes son medidas de complejidad?

- a)  $\Phi_i(n) =$  el número de cambios de estado hechos por  $M_i$  sobre la entrada  $n$ .
- b)  $\Phi_i(n) =$  el número máximo de movimientos hechos por  $M_i$  sobre la entrada  $n$ , sin que haya lugar a cambios de estado.
- c)  $\Phi_i(n) = 0$  para toda  $i$  y  $n$ .
- d)  $\Phi_i(n) = \begin{cases} 0 & \text{si } \phi_i(n) \text{ está definida,} \\ \text{indefinida} & \text{en cualquier otro caso} \end{cases}$

**\*\*12.20** (teorema de la Honestidad para el espacio) Muestre que existe una función totalmente recursiva,  $r$ , tal que, para cada clase de complejidad espacial  $\mathcal{C}$  existe una función  $S(n)$  tal que  $\text{ESPACIOD}(S(n)) = \mathcal{C}$  y  $S(n)$  es calculable en espacio  $r(S(n))$ .

**\*12.21** El Teorema 12.7 muestra que dada  $S(n)$ , existe un conjunto  $L$  tal que cualquier TM que reconozca a  $L$  utiliza más de  $S(n)$  espacio i.o. Fortalezca este resultado para demostrar que existe un conjunto  $L'$  tal que cualquier TM que reconozca a  $L'$  utiliza más de  $S(n)$  espacio a.e.

**\*\*12.22** Sea  $\Phi$  una medida de complejidad y  $c(i, j)$  una función recursiva tal que cuando  $\phi_i(n)$  y  $\phi_j(n)$  están definidas, entonces también lo está  $\phi_{c(i,j)}(n)$ . Demuestre que existe una función recursiva  $h$  tal que

$$\Phi_{c(i,j)}(n) \leq h(n, \Phi_i(n), \Phi_j(n)) \quad \text{a.e.}$$

**\*\*12.23** Muestre que si  $f(n)$  es espacialmente construible de manera completa entonces  $\text{TIEMPOD}(f(n) \log f(n)) \subseteq \text{ESPACIOD}(f(n))$ .

**12.24** Presente una TM que acepte un conjunto infinito, el cual no tenga ningún subconjunto regular infinito.

**\*12.25** Considere TMs de una sola cinta que utilicen una unidad constante de tinta cada vez que cambian un símbolo sobre la cinta.

- a) Demuestre un teorema de "velocidad" lineal para la tinta.
- b) Dé una definición apropiada de "función de tinta construible de manera completa".
- c) ¿Qué incremento en la cantidad de tinta se necesita para obtener una nueva clase de complejidad?

**12.26** Se dice que una máquina de Turing es *olvidadiza* si la posición de la cabeza en cada unidad de tiempo depende solamente de la longitud de la entrada y no de la entrada real. Demuestre que si  $L$  es aceptado por una TM con límite temporal  $T(n)$  y  $k$  cintas, entonces  $L$  es aceptado por una TM olvidadiza  $T(n) \log T(n)$  de dos cintas.

**\*12.27** Sea  $L \subseteq (0 + 1)^*$  un conjunto aceptado por alguna TM con límite temporal  $T(n)$ . Demuestre que, para cada  $n$ , existe un circuito booleano, con entradas  $x_1, \dots, x_n$ , que tiene, cuando mucho,  $T(n) \log T(n)$  compuertas de dos entradas y que produce la salida 1 si y sólo si los valores de  $x_1, \dots, x_n$  corresponden a una cadena de  $L$ . Los valores de  $x_1, \dots, x_n$  corresponden a la cadena  $x$  si  $x_i$  tiene valor verdadero siempre que el  $i$ -ésimo símbolo de  $x$  sea 1, y  $x_i$  tenga valor falso siempre que el  $i$ -ésimo símbolo de  $x$  sea 0. [Sugerencia: Simúlese una TM olvidadiza].

**\*\*12.28.** Los ciclos de los programas consisten en variables que toman valores enteros y proposiciones. Una proposición es una de las formas que se dan a continuación.

1.  $\langle \text{variable} \rangle := \langle \text{variable} \rangle$
2.  $\langle \text{variable} \rangle := \langle \text{variable} \rangle + 1$
3.  $\text{for } i := 1 \text{ to } \langle \text{variable} \rangle \text{ do proposición;}$
4.  $\text{begin } \langle \text{proposición} \rangle; \langle \text{proposición} \rangle; \dots \langle \text{proposición} \rangle \text{ end;}$

En (3) el valor de la variable está limitado antes del ciclo, como en PL/I.

- a) Demuestre que los programas en ciclos siempre terminan.
- b) Demuestre que cada programa en ciclos calcula una función recursiva primitiva.
- c) Demuestre que cada función recursiva primitiva es calculada por algún programa en ciclos.
- d) Demuestre que una TM que tiene un tiempo de corrida que es una función recursiva primitiva puede calcular solamente una función recursiva primitiva.

\*12.29. Sea  $F$  un sistema de prueba formal en el cual podemos demostrar teoremas relacionados con las TMs de una sola cinta. Defínase una clase de complejidad.

$$C_{T(n)} = \{L(M_i) \mid \text{existe una demostración en } F \text{ sobre que } T_i(n) \leq T(n), \text{ para toda } n\}.$$

¿Puede reforzarse la jerarquía temporal del Ejercicio 12.16 para que entre en la complejidad probable? [Sugerencia: Sustitúyase el reloj por una demostración sobre el hecho de que  $T_i(n) \leq T(n)$ .]

#### Soluciones a los ejercicios seleccionados

12.2(a) Considérese una cadena  $wcw^R$  de longitud  $n$  y sea  $\ell_{w^R_i}$  la longitud de la secuencia cruzada entre las posiciones  $i$  e  $i+1$ , para  $1 \leq i < n/2$ , hecha por alguna TM  $M$  con  $s$  estados. Supóngase que el promedio de  $\ell_{w^R_i}$  sobre todas las palabras  $w$  de longitud  $(n-1)/2$  es  $p(i)$ . Entonces, para al menos la mitad de todas las  $w$ s,  $\ell_{w^R_i} \geq 2p(i)$ . El número de  $w$ s es  $2^{(n-1)/2}$ , de modo que hay, por lo menos  $2^{(n-3)/2}$   $w$ s para las que  $\ell_{w^R_i} \leq 2p(i)$ . Como el número de secuencias cruzadas de longitud  $2p(i)$  o menor es

$$\sum_{j=0}^{2p(i)} s^j \leq s^{2p(i)+1},$$

debe existir al menos  $2^{(n-3)/2}/s^{2p(i)+1}$   $w$ s con la misma secuencia cruzada entre las posiciones  $i$  e  $i+1$ . Existen  $2^{(n-1)/2-i}$  secuencias de  $as$  y  $bs$  que pueden aparecer en las posiciones  $i+1$  hasta  $(n-1)/2$  de tales palabras, de modo que si

$$\frac{2^{(n-3)/2}}{s^{2p(i)+1}} > 2^{(n-1)/2-i} \quad (12.11)$$

Entonces dos palabras que tengan la misma secuencia cruzada difieren en algún lugar entre las primeras  $i$  posiciones. Así pues, según el Ejercicio 12.1(b),  $M$  acepta una palabra que no debería aceptar.

Por consiguiente (12.11) es falsa, y  $s^{2p(i)+1} \geq 2^{i-1}$ . En consecuencia,

$$p(i) \geq \frac{i-1}{2 \log_2 s} - \frac{1}{2}$$

Seguramente existe alguna palabra  $w$  tal que, cuando se le presenta con  $wcw^R$ ,  $M$  toma al menos un tiempo promedio. Según el Ejercicio 12.1(a), este promedio es de al menos

$$\sum_{i=1}^{(n-1)/2} p(i) \geq \sum_{i=1}^{(n-1)/2} \frac{i-1}{2 \log_2 s} - \frac{n-1}{4} \geq \frac{1}{4 \log_2 s} \left( \frac{n-3}{2} \right) \left( \frac{n-1}{2} \right) - \frac{n-1}{4}$$

12.14(c) Podemos diseñar una TM fuera de línea,  $M$ , de complejidad espacial  $S(n)$  para probar, para  $i = 2, 3, \dots$ , si su entrada de longitud  $n$  es divisible entre cada  $i$ , deteniéndose tan pronto como encontramos un valor de  $i$  que no divide a  $n$ . Como la prueba sobre si  $i$  divide a  $n$  sólo necesita  $\log_2 i$  celdas de almacenamiento,  $S(n)$  es el logaritmo de la  $i$  más grande tal que  $2, 3, \dots, i$ , todos, dividen a  $n$ . Si hacemos  $n = k!$ , sabemos que  $S(n) \geq \log_2 k$ . Como  $k! \leq k^k$ , sabemos que

$$\log_2 n \leq k \log_2 k$$

y

$$\log_2 \log_2 n \leq \log_2 k + \log_2 \log_2 k \leq 2 \log_2 k.$$

Por consiguiente para aquellos valores de  $n$  que son  $k!$ , para alguna  $k$ , se concluye que

$$S(n) \geq \frac{1}{2} \log_2 \log_2 n.$$

Debemos demostrar que, para toda  $n$ ,  $S(n) \geq 1 + \log_2 \log_2 n$ . Es suficiente demostrar que la  $n$  más pequeña para la cual  $S(n) \geq k$ , que es el mínimo común múltiplo (MCM) de  $2, 3, \dots, 2^{k-1} + 1$ , es al menos  $2^{2k-1}$ . [Esto es, necesitamos el hecho de que MCM ( $2, 3, \dots, i$ )  $\geq 2^{i-1}$ ]. Una demostración requiere resultados de la teoría de números que no estamos preparados para deducir, en particular el que se refiere a que la probabilidad de que el entero  $i$  sea primo es asintóticamente  $1/\ln i$ , en donde  $\ln$  es el logaritmo natural (véase Hardy y Wright [1938]). Puesto que MCM ( $2, 3, \dots, i$ ) es al menos el producto de los primos que se encuentran entre  $2$  e  $i$ , es sencillo mostrar un límite inferior del orden de  $e^i$  para MCM ( $2, 3, \dots, i$ ), para  $i$  grande.

#### NOTAS BIBLIOGRAFICAS

El estudio de la complejidad temporal, puede decirse, comenzó con Hartmanis y Stearns [1965], en donde encontraremos los Teoremas 12.3, 12.4, 12.5 y 12.9. El estudio serio de la complejidad espacial dio comienzo con el trabajo de Hartmanis, Lewis y Stearns [1965], y Lewis, Stearns y Hartmanis [1965]; los Teoremas 12.1, 12.2 y 12.8 se tomaron del primero. Seiferas [1977a, b] presenta algunos de los más recientes resultados sobre las jerarquías de complejidad. Un cierto número de artículos anteriores estudian aspectos parecidos en computación. En Grzegorczyk [1953], Axt [1959] y Ritchie [1963] encontramos jerarquías de las funciones recursivas. Yamada [1962] estudia la clase de funciones calculables en tiempo real [ $T(n) = n$ ]. Rabin [1963] mostró que dos cintas pueden hacer más que una en el tiempo real, un resultado que ha sido generalizado por Aanderaa [1974] a  $k$  contra  $k-1$  cintas.

El Teorema 12.6, que muestra que una desaceleración logarítmica es suficiente cuando uno pasa de muchas cintas a dos, se tomó de Hennie y Stearns [1966]. El Teorema 12.11, la relación cuadrática entre los tiempos determinísticos y no determinísticos, aparece en Savitch [1970]. Los lemas traslacionales fueron tratados por primera vez por Ruby y Fischer [1965], mientras que el Teorema 12.12, la jerarquía espacial no determinística, se debe a Ibarra [1972]. La jerarquía temporal no determinística a la que se aludió en el texto, es de Cook [1973a]. Las mejores jerarquías no determinísticas conocidas se encuentran en Seiferas, Fischer y Meyer [1973]. Book y Greibach [1970] caracterizaron los lenguajes  $\bigcup_{n=0}^{\infty} \text{TIEMPON}(cn)$ .

El estudio de las medidas de complejidad abstractas se originó con Blum [1967]. El Teorema 12.13, el teorema de espacio, se tomó de Borodin [1972] y (en esencia) de Trakhtenbrot [1964]; una versión más consistente se debe a Constable [1972]. ( Nótese que éstos y todos los artículos mencionados en este párrafo tratan sobre las medidas de complejidad de Blum, no únicamente sobre el espacio, como hicimos en el texto). El Teorema 12.14, teorema de la aceleración, es del trabajo de Blum [1967], y el teorema de la unión es de McCreight y Meyer [1969]. El Teorema 12.16, sobre las relaciones recursivas entre medidas de complejidad, se tomó de Blum [1967]. El teorema de honestidad, que se mencionó en el Ejercicio 12.20 es de

McCreight y Meyer [1969]. El planteamiento simplificado de la complejidad abstracta que se utiliza en este libro está basado en las ideas de Hartmanis y Hopcroft [1971].

Las secuencias cruzadas, discutidas en los Ejercicios 12.1 y 12.2, se tomaron de Hennie [1965]. La generalización de las secuencias cruzadas que se utilizó en los Ejercicios 12.3 y 12.4 se desarrolló en Hopcroft y Ullman [1969a], aunque el Ejercicio 12.4, en el caso determinístico, es de Hartmanis, Lewis y Stearns [1965]. El Ejercicio 12.14(c) se tomó de Freedman y Ladner [1975]. El ejercicio 12.16, una jerarquía más densa cuando las TMs se restringen a tener exactamente  $k$  cintas, es del trabajo de Paul [1977]. El Ejercicio 12.18, que muestra que la aceleración no puede hacerse efectiva, se tomó de Blum [1971]. El Ejercicio 12.22 se tomó de Hartmanis y Hopcroft [1971]. El Ejercicio 12.23 es de Hopcroft, Paul y Valiant [1975]. Véase también Paul, Tarjan y Celoni [1976] en donde se hallará una demostración de que el método de Hopcroft *et al.*, no puede ser extendido. Las máquinas de Turing olvidadizas y los Ejercicios 12.26 y 12.27 se deben a M. Fischer y N. Pippenger Los programas en ciclos y <sup>31</sup> Ejercicio 12.28 se tomaron del trabajo de Ritchie [1963] y del de Meyer y Ritchie [1967].

## CAPITULO

## 13

PROBLEMAS  
NO TRATABLES

En el Capítulo 8 descubrimos que se pueden plantear problemas que no son resolubles en una computadora. En este capítulo veremos que, entre los problemas resolubles, existen algunos tan difíciles que, para todos los fines prácticos, no pueden resolverse en su generalidad completa con una computadora. Algunos de tales problemas, aunque sean resolubles, se ha demostrado que requieren un tiempo exponencial para su resolución. Otros requieren por necesidad un tiempo exponencial; si existiera una forma más rápida de resolverlos que la exponencial, entonces un gran número de problemas importantes de matemáticas, ciencias de la computación y de otros campos (problemas para los cuales se ha buscado en vano una buena solución durante muchos años) podrían resolverse con medios sustancialmente mejores que los ya conocidos.

## 13.1 TIEMPO Y ESPACIO POLINOMIALES

Los lenguajes reconocibles en un tiempo determinístico exponencial forman una clase natural e importante, la clase  $\bigcup_{i \geq 1} \text{TIEMPO}(n^i)$ , la cual denotaremos con  $\mathcal{P}$ . Es un concepto intuitivamente atractivo el hecho de que  $\mathcal{P}$  sea la clase de problemas que pueden ser resueltos de manera eficiente. Aunque se podría objetar que un algoritmo de  $n^{57}$  pasos no es muy eficiente, en la práctica encontramos que los problemas de  $\mathcal{P}$  por lo general tienen soluciones en un tiempo polinomial de grado inferior.

Existe un cierto número de problemas importantes que no parecen estar en  $\mathcal{P}$  pero que tienen algoritmos no determinísticos eficientes. Estos problemas caen dentro de la clase  $\bigcup_{i \geq 1} \text{TIEMPON}(n^i)$ , la cual denotaremos por  $\mathcal{NP}$ . Un ejemplo es el problema circuital de Hamilton: *¿Tiene un grafo un ciclo en el que cada vértice del grafo aparece*

exactamente una vez? No parece que haya un algoritmo en tiempo polinomial determinístico que reconozca aquellos grafos con circuitos de Hamilton. Sin embargo, existe un algoritmo no determinístico sencillo; adivine las aristas del ciclo y verifique que de verdad forman un circuito de Hamilton.

La diferencia entre  $\mathcal{P}$  y  $\mathcal{NP}$  es parecida a la diferencia entre encontrar de manera eficiente una demostración de una proposición (como “este grafo tiene un circuito de Hamilton”) y verificar de manera eficiente una demostración (es decir, confirmar que determinado circuito es de Hamilton). Intuitivamente sentimos que verificar una demostración dada es más sencillo que encontrar una demostración, pero no sabemos esto a ciencia cierta.

Las otras dos clases naturales son

$$\text{ESPACIOP} = \bigcup_{i \geq 1} \text{ESPACIOD}(n^i)$$

y

$$\text{ESPACION} = \bigcup_{i \geq 1} \text{ESPACION}(n^i)$$

Nótese que según el Teorema de Savitch (Teorema 12.11)  $\text{ESPACIOP} = \text{ESPACION}$ , ya que  $\text{ESPACION}(n^i) \subseteq \text{ESPACIOD}(n^{2i})$ . Obviamente,  $\mathcal{P} \subseteq \mathcal{NP} \subseteq \text{ESPACIOP}$ , aunque no se sabe si alguna de estas contenencias son propias. Más aún, como veremos, no es posible que las herramientas matemáticas que se necesitan para resolver las cuestiones de una manera o de otra hayan sido desarrolladas.

Dentro de  $\text{ESPACIOP}$  tenemos dos jerarquías de clases de complejidad:

$$\text{ESPACIOD}(\log n) \subsetneq \text{ESPACIOD}(\log^2 n) \subsetneq \text{ESPACIOD}(\log^3 n) \subsetneq \dots$$

y

$$\text{ESPACION}(\log n) \subsetneq \text{ESPACION}(\log^2 n) \subsetneq \text{ESPACION}(\log^3 n) \subsetneq \dots$$

Es claro que  $\text{ESPACIOD}(\log^k n) \subseteq \text{ESPACION}(\log^k n)$  y por consiguiente, según el teorema de Savitch,

$$\bigcup_{k \geq 1} \text{ESPACION}(\log^k n) = \bigcup_{k \geq 1} \text{ESPACIOD}(\log^k n).$$

Aunque uno puede demostrar que

$$\mathcal{P} \neq \bigcup_{k \geq 1} \text{ESPACIOD}(\log^k n),$$

la contención de una de las clases en la otra se desconoce. Sin embargo

$$\text{ESPACIOD}(\log n) \subseteq \mathcal{P} \subseteq \mathcal{NP} \subseteq \text{ESPACIOP},$$

y al menos una de las contenencias es propia, ya que  $\text{ESPACIOD}(\log n) \subsetneq \text{ESPACIOP}$ , según el teorema de jerarquía espacial.

### Reducibilidades limitadas

Recuérdese que en el Capítulo 8 demostramos que un lenguaje  $L$  era irresoluble tomando un lenguaje  $L'$  que se sabía era irresoluble y *reduciéndolo* a  $L$ . Esto es, presentábamos una transformación  $g$  calculada por una TM que siempre se detiene, tal que, para todas las cadenas  $x$ ,  $x$  se encuentra en  $L'$  si y sólo si  $g(x)$  se encuentra en  $L$ . Entonces si  $L$  fuera recursiva,  $L$  no podría ser reconocida mediante el cálculo de  $g(x)$  y decidiendo si  $g(x)$  se encuentra en  $L$ .

Restringiendo a  $g$  a que sea una función fácilmente calculable, podemos establecer si  $L$  se encuentra o no en alguna clase tal como  $\mathcal{P}$ ,  $\mathcal{NP}$  o  $\text{ESPACIOP}$ . Nos centraremos ahora de manera especial, en dos tipos de reducibilidad: la reducibilidad en tiempo polinomial y la reducibilidad en espacio logarítmico. Decimos que  $L'$  es *reducible en un tiempo polinomial* a  $L$  si existe una TM con límite temporal polinomial que, para cada entrada  $x$ , produzca una salida  $y$  que se encuentre en  $L$  si y sólo si  $x$  se encuentra en  $L'$ .

**Lema 13.1** Sea  $L'$  reducible en tiempo polinomial a  $L$ . Entonces

a)  $L'$  se encuentra en  $\mathcal{NP}$  si  $L$  está en  $\mathcal{NP}$ ,

b)  $L'$  se encuentra en  $\mathcal{P}$  si  $L$  se encuentra en  $\mathcal{P}$ .

*Demuestra*ción Las demostraciones de (a) y (b) son parecidas. Demostraremos solamente (b). Supóngase que la reducción tiene un límite temporal  $p_1(n)$  y que  $L$  es reconocible en un tiempo  $p_2(n)$ , en donde  $p_1$  y  $p_2$  son polinomios. Entonces  $L'$  puede ser reconocido en un tiempo polinomial de la manera siguiente. Dada la entrada  $x$  de longitud  $n$ , prodúzcase  $y$  utilizando la reducción en tiempo polinomial. Como la reducción tiene límite temporal  $p_1(n)$  y, cuando mucho, un solo símbolo puede ser impreso por movimiento, se concluye que  $|y| \leq p_1(n)$ . Entonces, podemos probar si  $y$  se encuentra en  $L'$  en un tiempo  $p_2(p_1(n))$ , por tanto, el tiempo total que se lleva decir si  $x$  se encuentra en  $L$  es  $p_1(n) + p_2(p_1(n))$ , que es un polinomio en  $n$ . En consecuencia,  $L$  está en  $P$ .  $\square$

Un *transductor espacio logarítmico* es una TM fuera de línea que siempre se detiene, que tiene un almacenamiento de tachado  $\log n$  y una cinta de salida de sólo escritura, sobre la cual la cabeza nunca se mueve hacia la izquierda. Decimos que  $L'$  es *reducible en un espacio logarítmico* a  $L$  si existe un transductor espacio logarítmico el cual, dada la entrada  $x$ , produce una cadena de salida  $y$  que se encuentra en  $L$  si y sólo si  $x$  se encuentra en  $L'$ .

**Lema 13.2** Si  $L'$  es reducible en un espacio logarítmico a  $L$ , entonces

a)  $L'$  se encuentra en  $\mathcal{P}$  si  $L$  está en  $\mathcal{P}$ ,

b)  $L$  está en  $\text{ESPACION}(\log^k n)$ , si  $L$  se encuentra en  $\text{ESPACION}(\log^k n)$ ,

c)  $L'$  se encuentra en  $\text{ESPACIOD}(\log^k n)$  si  $L$  se encuentra en  $\text{ESPACIOD}(\log^k n)$ .

*Demuestra*ción

a) Es suficiente mostrar que una reducción en espacio logarítmico no puede tomar más de un tiempo polinomial, de modo que el resultado se concluye del Lema 13.1(b).

En la demostración, nótense que el contenido de la cinta de salida no puede influir en el cálculo, de modo que el producto del número de estados, contenido de cinta de almacenamiento y posiciones de las cabezas de cinta de entrada y de almacenamiento constituye un límite superior para el número de movimientos que se pueden hacer antes de que el transductor espacio logarítmico accese un ciclo, lo cual sería una contradicción a la suposición de que siempre se detiene. Si la cinta de almacenamiento tiene una longitud de  $\log n$ , se puede ver con facilidad que el límite es un polinomio en  $n$ .

Existe una sutileza involucrada en las demostraciones de (b) y (c). Demostraremos solamente (c), siendo la demostración de (b) esencialmente la misma que la de (c).

c) Sea  $M_1$  el transductor espacio logarítmico que reduce  $L'$  a  $L$ , y sea  $M_2$  una TM con límite espacial  $\log^k n$  que acepta en  $L$ . Sobre la entrada  $x$  de longitud  $n$ ,  $M_1$  produce una salida de longitud anotada por  $n^c$  para alguna constante  $c$ . Puesto que la salida no puede escribirse en un espacio  $\log^k n$ ,  $M_1$  y  $M_2$  no pueden ser simuladas mediante el almacenamiento de la salida de  $M_1$  sobre una cinta. En lugar de esto, la salida de  $M_1$  puede alimentarse directamente a  $M_2$ , un símbolo por vez. Esto funciona mientras  $M_2$  se mueve hacia la derecha sobre su entrada. De moverse  $M_2$  a la izquierda  $M_1$  debe ser reiniciada para determinar el símbolo de entrada de  $M_2$ , ya que la salida de  $M_1$  no ha sido salvada.

Construimos  $M_3$  para aceptar a  $L'$  de la manera siguiente. Una cinta de almacenamiento de  $M_3$  contiene la posición de entrada de  $M_2$  en base  $2^c$ . Como la posición de entrada no puede sobrepasar  $n^c$ , este espacio puede almacenarse en un espacio  $\log n$ . Las otras cintas de almacenamiento de  $M_3$  simulan a las cintas de almacenamiento de  $M_1$  y  $M_2$ . Supóngase que, en algún momento la cabeza de entrada de  $M_2$  se encuentra en la posición  $i$ , y que  $M_2$  hace un movimiento hacia la izquierda o la derecha.  $M_3$  ajusta el estado y las cintas de almacenamiento de  $M_2$  de acuerdo con lo anterior. Entonces  $M_3$  reinicia la simulación de  $M_1$  a partir del comienzo y espera hasta que  $M_1$  ha producido  $i-1$  o  $i+1$  símbolos de salida, si la cabeza de entrada de  $M_2$  se ha movido hacia la izquierda o hacia la derecha, respectivamente. El último símbolo de salida producido es el nuevo símbolo barrido por la cabeza de  $M_2$ , de modo que  $M_3$  está lista para simular el siguiente movimiento de  $M_2$ . Como casos especiales, si  $i=1$  y  $M_2$  se mueve hacia la izquierda, suponemos que, en seguida,  $M_2$  barre el señalador de extremo de la izquierda, y si  $M_1$  se detiene antes de producir  $i+1$  símbolos de salida (cuando  $M_2$  se mueve hacia la derecha), suponemos que, en seguida,  $M_2$  barre el señalador de extremo de la derecha.  $M_3$  acepta a su propia entrada cuando  $M_2$  acepta su entrada simulada. Por consiguiente  $M_3$  es una TM con límite espacial  $\log^k n$  que acepta en  $L'$ . □

**Lema 13.3** La composición de dos reducciones en espacio logarítmico (resp. tiempo polinomial) es una reducción en espacio logarítmico (resp. tiempo polinomial).

*Demostración* Es una sencilla generalización de las construcciones que se presentaron en los Lemas 13.1 y 13.2. □

### Problemas completos

Como ya lo hemos mencionado, nadie sabe si  $\mathcal{NP}$  incluye a lenguajes que no estén en  $\mathcal{P}$ , de modo que la cuestión de la contención propia está abierta. Una forma de encontrar un lenguaje que esté en  $\mathcal{NP}-\mathcal{P}$  consiste en buscar un problema “más difícil” que esté en  $\mathcal{NP}$ . De manera intuitiva, un lenguaje  $L_0$  es un problema más difícil si cada lenguaje de  $\mathcal{NP}$  es reducible a  $L_0$  mediante una reducción fácilmente calculable. Dependiendo del tipo exacto de reducibilidad, podemos concluir ciertos hechos con relación a  $L_0$ . Por ejemplo, si toda  $\mathcal{NP}$  es reducible en espacio logarítmico a  $L_0$ , podemos concluir que si  $L_0$  estuviera en  $\mathcal{P}$ , entonces  $\mathcal{P}$  sería igual a  $\mathcal{NP}$ . De manera similar, si  $L_0$  estuviera en  $\text{ESPACIOD}(\log n)$ , entonces  $\mathcal{NP} = \text{ESPACIOD}(\log n)$ . Si toda  $\mathcal{NP}$  fuera reducible en un tiempo polinomial a  $L_0$ , entonces todavía podríamos concluir que si  $L_0$  estuviera en  $\mathcal{P}$ , entonces  $\mathcal{P}$  sería igual a  $\mathcal{NP}$ , pero no podríamos concluir que la proposición  $L_0$  está en  $\text{ESPACIOD}(\log n)$ .  $\mathcal{NP} = \text{ESPACIOD}(\log n)$ .

De los ejemplos anteriores, vemos que el concepto de “más difícil” puede depender de la clase de reducibilidad de que se trate. Esto es, puede haber lenguajes  $L_0$  tales que todos los lenguajes de  $\mathcal{NP}$  tengan reducciones en tiempo polinomial a  $L_0$ , pero no todos tengan reducciones en espacios logarítmicos a  $L_0$ . Más aún, las reducciones en espacio logarítmico y tiempo polinomial no agotan los tipos de reducciones que podemos considerar. Teniendo esto en cuenta, definimos el concepto de problemas más difíciles (*completos*), para una clase general de lenguajes. Con respecto a un tipo particular de reducción. Es claro que lo siguiente se puede generalizar a un tipo arbitrario de reducción.

Sea  $\mathcal{C}$  la clase de los lenguajes. Decimos que el lenguaje  $L$  es *completo para  $\mathcal{C}$  con respecto a las reducciones en tiempo polinomial (espacio logarítmico)* si  $L$  se encuentra en  $\mathcal{C}$ , y cada lenguaje de  $\mathcal{C}$  es reducible en tiempo polinomial (espacio logarítmico) a  $L$ . Decimos que  $L$  es *difícil para  $\mathcal{C}$  con respecto a las reducciones en tiempo polinomial (espacio logarítmico)* si cada lenguaje de  $\mathcal{C}$  es reducible en tiempo polinomial (espacio logarítmico) a  $L$ , pero  $L$  no está necesariamente en  $\mathcal{C}$ . Dos casos especiales son de gran importancia e introduciremos abreviaturas para ellos.  $L$  es *NP-completo (NP-difícil)* si  $L$  es completo (difícil) para  $\mathcal{NP}$  con respecto a las reducciones en espacio logarítmico. †  $L$  es *ESPACIOP-completo (ESPACIOP-difícil)* si  $L$  es completo (difícil) para  $\text{ESPACIOP}$  con respecto a las reducciones en tiempo polinomial.

Con el fin de mostrar que un primer lenguaje  $L_0$  es *NP-completo*, debemos dar una reducción en espacio logarítmico de cada lenguaje de  $\mathcal{NP}$  a  $L_0$ . Una vez que se tenga un problema *NP-completo*  $L_0$ , podemos demostrar que otro lenguaje  $L_1$  de  $\mathcal{NP}$  es *NP-completo* mediante la presentación de una reducción en espacio logarítmico de  $L_0$  a  $L_1$ , ya que la composición de dos reducciones en espacio logarítmico es una reducción en espacio logarítmico, según el Lema 13.3. Esta misma técnica se utilizará para establecer problemas completos para otras clases.

† Muchos autores utilizan el término “*NP-completo*” para significar “completo para  $\mathcal{NP}$  con respecto a las reducciones en tiempo polinomial”, o en algunos casos “con respecto a las reducciones de Turing en tiempo polinomial”.

### 13.2 ALGUNOS PROBLEMAS NP-COMPLETOS

La significancia de la clase de los problemas *NP*-completos reside en que incluye a muchos problemas que son naturales y han sido examinados seriamente en busca de soluciones eficientes. No se sabe si alguno de estos problemas tiene una solución en tiempo polinomial. El hecho de que si cualquiera de estos problemas estuviera en  $\mathcal{P}$ , todos los demás también lo estarían, refuerza el concepto de que es improbable que tengan soluciones en tiempo polinomial. Más aún, si se demuestra que un nuevo problema es *NP*-completo, entonces tenemos el mismo grado de confiabilidad de que el nuevo problema es difícil que el que se tiene para los problemas clásicos.

El primer problema que demostraremos que es *NP*-completo, que a la sazón viene a ser, históricamente, el primero de tales problemas, es el carácter de satisfacción de las expresiones booleanas. Comenzaremos definiendo el problema de manera precisa.

#### El problema de la satisfactoriedad

Una *expresión booleana* es una expresión compuesta por variables, paréntesis y los operadores  $\wedge$  (AND lógico),  $\vee$  (OR lógico) y  $\neg$  (negación). La precedencia de estos operadores es  $\neg$  más alto, después  $\wedge$  y después  $\vee$ . Las variables toman valores 0 (falso) y 1 (verdadero); y también las expresiones. Si  $E_1$  y  $E_2$  son expresiones booleanas, entonces el valor de  $E_1 \wedge E_2$  es 1 si ambos,  $E_1$  y  $E_2$ , tienen valor 1 y 0 en cualquier otro caso. El valor de  $E_1 \vee E_2$  en 1 si  $E_1$  o  $E_2$  tiene valor 1 y 0 en cualquier otro caso. El valor de  $\neg E_1$  es 1 si  $E_1$  es 0 y 0 si  $E_1$  es 1. Una expresión es *satisfaciente* si existe alguna asignación de 0s y 1s a las variables que da a la expresión el valor 1. *El problema de la satisfactoriedad* consiste en determinar, dada una expresión booleana, si ésta es satisfaciente.

Podemos representar el problema de la satisfactoriedad como un lenguaje  $L_{sat}$  de la manera siguiente. Sean las variables de alguna expresión  $x_1, x_2, \dots, x_m$ , para alguna  $m$ . Codifíquese  $x_i$  como el símbolo  $x$  seguido por  $i$  escrito en binario. El alfabeto de  $L_{sat}$  es, por tanto,

$$\{\wedge, \vee, \neg, (,), x, 0, 1\}.$$

Es fácil ver que la longitud de la versión codificada de una expresión de  $n$  símbolos es de no más  $\lceil n \log_2 n \rceil$ , ya que cada símbolo diferente a una variable se codifica mediante un símbolo, no existen más de  $\lceil n/2 \rceil$  variables diferentes en una expresión de longitud  $n$  y el código de una variable requiere de no más de  $1 + \lceil \log_2 n \rceil$  símbolos. De aquí en adelante trataremos a la palabra en  $L_{sat}$  que representa una expresión de longitud  $n$  como si la palabra misma fuera de longitud  $n$ . Nuestros resultados no dependerán del hecho de si utilizamos  $n$  o  $n \log n$  para la longitud de la palabra, ya que  $\log(n \log n) \leq 2 \log n$ , y trataremos con reducciones en espacio logarítmico.

Se dice que una expresión booleana se encuentran en *forma normal conjuntiva*<sup>†</sup> (*CNF*) si es de la forma  $E_1 \wedge E_2 \wedge \dots \wedge E_k$ , y cada  $E_i$  conocido como *cláusula* (o *conjunto*), es de la forma  $\alpha_{i1} \vee \alpha_{i2} \vee \dots \vee \alpha_{in}$ , en donde cada  $\alpha_{iy}$  es una *literal*, esto es,  $x$  o  $\neg x$ , para alguna variable  $x$ . Generalmente escribimos  $x$  en lugar de  $\neg x$ . Por ejemplo,  $(x_1 \vee x_2) \wedge (x_1 \vee x_3 \vee x_4) \wedge x_3$ , se encuentra en CNF. Se dice que la expresión se encuentra en 3-CNF

<sup>†</sup> "Conjuntiva" es un adjetivo que se refiere al operador lógico AND (*conjunction*). El término *disyuntivo* se aplica de manera similar al operador lógico OR.

si cada cláusula tiene exactamente tres literales distintas. El ejemplo de arriba no se encuentra en 3-CNF porque las cláusulas primera y tercera tienen menos de tres literales.

#### La satisfactoriedad es *NP*-completa

Comenzaremos por dar una reducción en espacio logarítmico de cada lenguaje de  $\mathcal{NP}$  a  $L_{sat}$ .

**Teorema 13.1** El problema de la satisfactoriedad es *NP*-completo.

*Demostración* La parte sencilla de la demostración consiste en que  $L_{sat}$  se encuentra en  $\mathcal{NP}$ . Para determinar si una expresión de longitud  $n$  es satisfaciente, adivíñese de manera no determinística valores para todas las variables y después evalúese la expresión. Por tanto,  $L_{sat}$  se encuentra en  $\mathcal{NP}$ .

Para demostrar que cada lenguaje de  $\mathcal{NP}$  es reducible a  $L_{sat}$ , para cada NTM  $M$  que está limitada en el tiempo por un polinomio  $p(n)$ , damos un algoritmo en espacio logarítmico que toma como entrada una cadena  $x$  y produce una fórmula booleana  $E_x$  que es satisfaciente si y sólo si  $M$  acepta a  $x$ . Describiremos, ahora, a  $E_x$ .

Sea  $\# \beta_0 \# \beta_1 \# \dots \# \beta_{p(n)}$  un cálculo de  $M$ , en donde cada  $\beta_i$  es una ID que consiste en exactamente  $p(n)$  símbolos. Si ocurre la aceptación antes del movimiento  $p(n)$ , permitimos que se repita la ID que ha aceptado, de manera que cada cálculo tiene exactamente  $p(n) + 1$  IDs.

En cada ID agrupamos el estado con el símbolo barrido para formar un solo símbolo compuesto. Además, el símbolo compuesto en la  $i$ -ésima ID contiene un entero  $m$  que indica el movimiento mediante el cual la  $(i + i)$ -ésima ID se concluye de la  $i$ -ésima. Se asignan números a los movimientos mediante el ordenamiento arbitrario del conjunto finito de alternativas que  $M$  puede tener dados un estado y un símbolo de cinta.

Para cada símbolo que puede aparecer en un cálculo y para cada  $i$ ,  $0 \leq i < (p(n) + 1)^2$ , creamos una variable booleana  $c_{ix}$  para indicar si el  $i$ -ésimo símbolo del cálculo es  $X$ . (El símbolo que está en el lugar 0 es el # inicial.) La expresión  $E_x$  que deberemos construir será verdadera para una asignación dada a las  $c_{ix}$ s si y sólo si las  $c_{ix}$ s que son verdaderas corresponden a un cálculo válido. La expresión  $E_x$  establece lo siguiente:

1. Las  $c_{ix}$ s que son verdaderas corresponden a una cadena de símbolos, en que exactamente una  $c_{ix}$  es verdadera para cada  $i$ .
2. La ID  $\beta_0$  es una ID inicial de  $M$  con entrada  $x$ .
3. La última ID contiene un estado final.
4. Cada ID se concluye de la anterior mediante el movimiento de  $M$  que está indicado.

La fórmula  $E_x$  es el AND lógico de cuatro fórmulas, cada una de las cuales hace cumplir una de las condiciones anteriores. La primera fórmula, que establece que, para cada  $i$  entre 0 y  $(p(n) + 1)^2$ , exactamente una  $C_{ix}$  es verdadera, es

$$\bigwedge_i \left[ \bigvee_x C_{ix} \wedge \neg \left( \bigvee_{x \neq y} (C_{ix} \wedge C_{iy}) \right) \right].$$

Para un valor dado de  $i$ , el término  $\bigvee_{x \in F} C_{ix}$  fuerza al menos a un  $C_{ix}$  que sea verdadero y  $\neg \bigwedge_{x \neq y} (C_{ix} \wedge C_{iy})$  fuerza cuando más a uno a ser verdadero.

Sea  $x = a_1 a_2 \cdots a_n$ . La segunda fórmula, que expresa el hecho de que  $\beta_0$  es una ID inicial, es a su vez el operador AND de los siguientes términos.

i)  $c_{0\#} \wedge c_{p(n)+1,\#}$ . Los símbolos que se encuentran en las posiciones 0 y  $(p(n) + 1)$  son #.

ii)  $c_{1Y_1} \vee c_{1Y_2} \vee \cdots \vee c_{1Y_k}$ , en donde  $Y_1, Y_2, \dots, Y_k$  son todos símbolos compuestos que representan al símbolo de cinta  $a_1$ , el estado inicial  $q_0$  y el número de un movimiento legal de  $M$  en el estado  $q_0$  que se encuentra leyendo al símbolo  $a_1$ . Esta cláusula establece que el primer símbolo de  $\beta_0$  es correcto.

iii)  $\bigwedge_{2 \leq i \leq n} c_{iai}$ . Desde el segundo hasta el  $n$ -ésimo símbolo de  $\beta_0$  son correctos.

iv)  $\bigwedge_{n+1 \leq i \leq p(n)} c_{ib}$ . Los símbolos restantes de  $\beta_0$  son espacios en blanco.

La tercera fórmula dice que la última ID tiene un estado de aceptación. Puede escribirse de la forma

$$\bigvee_{p(n)(p(n)+1) < i < (p(n)+1)^2} \left( \bigvee_{X \in F} c_{iX} \right),$$

en donde  $F$  es el conjunto de símbolos compuestos que incluye un estado final.

Para ver cómo escribir la cuarta fórmula, que establece que cada ID  $\beta_i$ ,  $i \geq 1$ , se concluye de  $\beta_{i-1}$  mediante el movimiento que aparece en el símbolo compuesto de  $\beta_{i-1}$ , obsérvese que podemos deducir, esencialmente, cada símbolo de  $\beta_i$  a partir del símbolo correspondiente de  $\beta_{i-1}$  y los símbolos que se encuentran en cualquier lado (uno de ellos podría ser #). Esto es, el símbolo de  $\beta_i$  es el mismo que el símbolo correspondiente de  $\beta_{i-1}$  a menos que dicho símbolo tenga el estado y el movimiento, o uno de los símbolos adyacentes tenga el estado y el movimiento, y el movimiento ocasione que la posición de la cabeza se corra al lugar en que estaba el símbolo  $\beta_i$ . Nótese que si este símbolo de  $\beta_i$  es el que representa al estado, también representaría un movimiento legal arbitrario de  $M$ , así que puede existir más de un símbolo legal. Nótese también que si la ID anterior tiene un estado de aceptación, la ID anterior y la actual son iguales.

Por consiguiente, podemos especificar de manera sencilla un predicado  $f(W, X, Y, Z)$  que es verdadero si y sólo si el símbolo  $Z$  pudiera aparecer en la posición  $j$  de alguna ID dado que  $W, X$  y  $Y$  son símbolos que se encuentran en las posiciones  $j - 1$ ,  $j$  y  $j + 1$  de la ID anterior [ $W$  es # si  $j = 1$  y  $Y$  es # si  $j = p(n)$ ]. Resulta conveniente, también, declarar a  $f(W, \#, X, \#)$  como verdadero, de modo que podamos tratar a los señalados que están entre las IDs de la misma forma que tratamos a los símbolos que están dentro de las IDs. Podemos ahora expresar la cuarta fórmula como

$$\bigwedge_{p(n) < j < (p(n)+1)^2} \left( \bigvee_{\substack{W, X, Y, Z \text{ tal \\ que } f(W, X, Y, Z)}} (c_{j-p(n)-2, W} \wedge c_{j-p(n)-1, X} \wedge c_{j-p(n), Y} \wedge c_{jZ}) \right).$$

Es fácil, dado un cálculo de aceptación de  $M$  sobre  $x$ , encontrar valores verdaderos para las  $c_{ix}$ s que hacen a  $E_x$  verdadera. Sólo hágase  $c_{ix}$  verdadero si y solo si el  $i$ -ésimo símbolo del cálculo es  $X$ . De manera inversa, dada una asignación de valores verda-

deros que hacen que  $E_x$  sea verdadera, las cuatro fórmulas anteriores garantizan que existe un cálculo de aceptación de  $M$  sobre  $x$ . Adviértase que aún pensando que  $M$  es no determinística, el hecho de que una alternativa de movimiento se ha incorporado a cada ID garantiza que el estado siguiente, el símbolo impreso y la dirección de movimiento de la cabeza de una ID a la siguiente serán todos consistentes con alguna de las alternativas de  $M$ .

Aún más, las fórmulas que componen a  $E_x$  son de longitud  $O(p^2(n))$ , y son lo suficientemente simples para que una TM de espacio logarítmico pueda generarlas dada una  $x$  en su entrada. La TM sólo necesita capacidad de almacenamiento suficiente para contar hasta  $(p(n) + 1)^2$ . Como el logaritmo de un polinomio en  $n$  es alguna constante multiplicada por  $\log n$ , esto se puede hacer con un almacenamiento  $O(\log n)$ . Por consiguiente, hemos mostrado que cada lenguaje en  $NP$  es reducible en espacio logarítmico a  $L_{sat}$ , demostrando que  $L_{sat}$  es  $NP$ -completo.  $\square$

Acabamos de demostrar que el problema de la satisfactoriedad para las expresiones booleanas es  $NP$ -completo. Esto significa que se podría utilizar un algoritmo en tiempo polinomial que acepte a  $L_{sat}$  para aceptar a cualquier lenguaje de  $\mathcal{NP}$ . Sea  $L$  el lenguaje aceptado por alguna máquina de Turing no determinística con límite temporal  $p(n)$ ,  $M$  y sea  $A$  el transductor en espacio logarítmico (y, por tanto, en tiempo polinomial) que convierte a  $x$  en  $E_x$  en donde  $E_x$  es satisfaciente si y sólo si  $M$  acepta a  $x$ . Entonces  $A$ , combinada con el algoritmo para  $L_{sat}$ , como se muestra

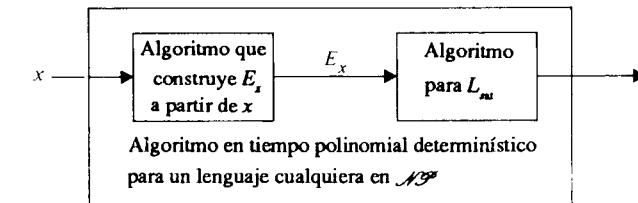


Fig 13.1 Algoritmo para un conjunto cualquiera  $L$  en  $\mathcal{NP}$  dado un algoritmo para  $L_{sat}$ .

en la Fig. 13.1, es un algoritmo en tiempo polinomial determinístico que acepta a  $L$ . Por consiguiente la existencia de un algoritmo en tiempo polinomial justamente para este problema, la satisfactoriedad de las expresiones booleanas, implicaría que  $\mathcal{P} = \mathcal{NP}$ .

### Problemas de satisfactoriedad restringida que son $NP$ -completos

Recuérdese que una fórmula booleana se encuentra en una forma normal conjuntiva (CNF) si es el operador lógico AND de las cláusulas que, a su vez, son los operadores OR lógicos de las literales. Decimos que la fórmula se encuentra en CNF- $k$  si cada cláusula tiene exactamente  $k$  literales. Por ejemplo,  $(x \vee y) \wedge (\bar{x} \vee z) \wedge (y \vee \bar{z})$  está en 2-CNF.

Consideraremos ahora dos lenguajes,  $L_{sat}$ , el conjunto de las fórmulas booleanas satisfacientes, y  $L_{3sat}$ , el conjunto de las fórmulas booleanas satisfacientes que se encuentran en 3-CNF. Daremos reducciones en espacio logarítmico de  $L_{sat}$  a  $L_{3sat}$  y de  $L_{3sat}$  a  $L_{sat}$ , demostrando que los últimos dos problemas son  $NP$ -completos, según el Lema

13.3. En cada caso, hacemos una transformación de una expresión a otra expresión que puede no ser equivalente, pero que es satisfaciente si y sólo si la expresión original lo es.

**Teorema 13.2**  $L_{\text{sat}}$ , el problema satisfaciente para las expresiones CNF, es NP-completo.

**Demostración** Es claro que  $L_{\text{sat}}$  se encuentra en  $\text{NP}$ , ya que  $L_{\text{sat}}$  lo está. Reducimos  $L_{\text{sat}}$  a  $L_{\text{cnf}}$  de la manera siguiente. Sea  $E$  una expresión booleana arbitraria de longitud  $n$ .<sup>†</sup> Ciertamente, el número de veces que aparece una variable en  $E$  no sobrepasa a  $n$ , ni tampoco el número de operadores  $\wedge$  y  $\vee$ . Utilizando las identidades

$$\begin{aligned}\neg(E_1 \wedge E_2) &= \neg(E_1) \vee \neg(E_2), \\ \neg(E_1 \vee E_2) &= \neg(E_1) \wedge \neg(E_2), \\ \neg\neg E_1 &= E_1,\end{aligned}\tag{13.1}$$

podemos transformar  $E$  a una expresión equivalente  $E'$ , en la cual los operadores —son aplicados solamente a las variables, nunca a expresiones más complejas. La validez de las Ecuaciones (13.1) puede verificarse considerando las cuatro asignaciones de valores 0 y 1 a  $E_1$  y  $E_2$ . De manera incidental, las dos primeras de estas ecuaciones se conocen como *Leyes de DeMorgan*.

La transformación puede considerarse como la composición de dos transformaciones en espacio logarítmico. Como resultado de la primera transformación, cada símbolo de negación que antecede inmediatamente a una variable es sustituido por una barra sobre la variable, y cada paréntesis que cierra, cuyo paréntesis correspondiente está precedido inmediatamente por una negación, se sustituye por  $)^{\neg}$ . El símbolo  $\neg$  indica el final del dominio de una negación. Esta primera transformación se lleva a cabo fácilmente en espacio logarítmico utilizando un contador para localizar los paréntesis correspondientes.

La segunda transformación se lleva a cabo mediante un autómata finito que barre la entrada de izquierda a derecha, manteniendo registro de la paridad (módulo 2 suma) de las negaciones *activas*, aquellas de las que inmediatamente sigue un paréntesis que se abre, pero cuyo paréntesis que se cierra aún no ha sido visto. Cuando la paridad de la negación es impar,  $x$  es sustituido por  $\bar{x}$ ,  $\bar{x}$  por  $x$ ,  $\vee$  por  $\wedge$  y  $\wedge$  por  $\vee$ . Los símbolos  $\neg$  y  $\neg$  se eliminan. Se debe demostrar que esta información es correcta utilizando (13.1) mediante una sencilla inducción sobre la longitud de una expresión. Tenemos, ahora, una expresión  $E'$  en la cual todas las negaciones son aplicadas directamente a las variables.

En seguida creamos  $E''$ , una expresión en CNF que es satisfaciente si y sólo si  $E'$  es satisfaciente. Sean  $V_1$  y  $V_2$  conjuntos de variables, con  $V_1 \subseteq V_2$ . Decimos que una asignación de valores a  $V_2$  es una *extensión* de una asignación de valores a  $V_1$  si las

<sup>†</sup> Recuérdese que la longitud de una expresión booleana es el número de caracteres, no la longitud de su código, y recuérdese también que esta diferencia no cuenta cuando se trata de reducciones en espacio logarítmico.

asignaciones concuerdan sobre las variables de  $V_1$ . Demostraremos por inducción en  $r$ , el número de  $\wedge$ s y  $\vee$ s de una expresión  $E'$ , cuyas negociaciones se aplican todas a variables, que si  $|E'| = n$ , entonces existe una lista de cuando más  $n$  cláusulas,  $F_1, F_2, \dots, F_n$ , sobre un conjunto de variables que incluye a las variables de  $E'$  y como mucho  $n$  variables más, tal que  $E'$  tiene valor 1 mediante una asignación a sus variables si y sólo si la extensión de tal asignación satisface  $F_1 \wedge F_2 \wedge \dots \wedge F_n$ .

**Base**  $r = 0$ . Entonces  $E$  es una literal y podemos tomar esa literal en una cláusula en la que ella misma satisface las condiciones.

**Inducción** Si  $E = E_1 \wedge E_2$ , hagamos que  $F_1, F_2, \dots, F_n$  y  $G_1, G_2, \dots, G_n$  sean las cláusulas para  $E_1$  y  $E_2$  que existen según la hipótesis inductiva. Supóngase, sin pérdida de generalidad, que ninguna variable que no se encuentre presente en  $E$  aparece tanto entre las  $F$ s como entre las  $G$ s. Entonces  $F_1, F_2, \dots, F_n, G_1, G_2, \dots, G_n$  satisface las condiciones para  $E'$ .

Si  $E = E_1 \vee E_2$ , tomemos las  $F$ s y  $G$ s de la misma manera que en el párrafo anterior, y sea  $y$  una nueva variable. Entonces  $y \vee F_1, y \vee F_2, \dots, y \vee F_n, \bar{y} \vee G_1, \bar{y} \vee G_2, \dots, \bar{y} \vee G_n$  satisface las condiciones. Para la demostración supóngase que una asignación de variables satisface a  $E'$ . Entonces debe satisfacer a  $E_1$  o a  $E_2$ . Si la asignación satisface a  $E_1$ , entonces alguna extensión de la asignación satisface a  $F_1, F_2, \dots, F_n$ . Cualquier extensión más de esta asignación que asigne  $y = 0$  satisfará todas las cláusulas para  $E'$ . Si la asignación satisface a  $E_2$ , un argumento parecido es suficiente. De manera inversa, supóngase que todas las cláusulas para  $E'$  son satisfechas por alguna asignación. Si esa asignación hace  $y = 1$ , entonces todas las  $G_1, G_2, \dots, G_n$  deben ser satisfechas, así pues  $E'$  se satisface. Un argumento similar se aplica si  $y = 0$ . La expresión deseada  $E'$  consiste en todas las cláusulas para  $E'$  conectadas mediante  $\wedge$ s.

Para ver que la transformación anterior puede llevarse a cabo en espacio logarítmico, considérese el árbol de análisis gramatical de  $E$ . Sea  $y_i$  la variable introducida por el  $i$ -ésimo  $\vee$ . La expresión final es el operador lógico AND de las cláusulas, en donde cada cláusula contiene una literal de la expresión original. Además, si la literal se encuentra en el subárbol izquierdo de la  $i$ -ésima  $\vee$ , entonces la cláusula también contiene a  $y_i$ . Si la literal está en el subárbol derecho de la  $i$ -ésima  $\vee$ , entonces la cláusula contiene a  $\bar{y}_i$ . La entrada es barrida de izquierda a derecha. Cada vez que se encuentra una literal, se emite una cláusula. Para determinar cuáles  $y$ s y  $\bar{y}$ s incluir en la cláusula, utilizamos un contador de longitud  $\log n$  para recordar nuestro lugar sobre la entrada. Entonces barreremos la entrada completa y, para cada símbolo  $\vee$ , digamos el  $i$ -ésimo a partir de la izquierda, determinamos sus operandos izquierdo y derecho, utilizando otro contador de longitud  $\log n$  para contar los paréntesis. Si la literal actual se encuentra en el operando izquierdo, se genera  $y_i$ ; si se encuentra en el operando derecho, se genera  $\bar{y}_i$ ; si no está en ningún operando, no se genera ni  $y_i$  ni  $\bar{y}_i$ .

Por consiguiente, hemos reducido cada expresión booleana  $E$  a una expresión CNF  $E''$  que está en  $L_{\text{sat}}$ , si y sólo si  $E$  se encuentra en  $L_{\text{sat}}$ . Puesto que la reducción se llevó a cabo en espacio logarítmico, la integridad NP de  $L_{\text{sat}}$  implica la integridad NP de  $L_{\text{sat}}$ .  $\square$

**Ejemplo 13.1** Sea

$$E = \neg(\neg(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)).$$

Aplicando las leyes de DeMorgan se produce

$$E' = (x_1 \vee x_2) \vee (x_1 \wedge \bar{x}_2).$$

La transformación a CNF introduce las variables  $y_1$  y  $y_2$  para producir

$$E'' = (x_1 \vee y_1 \vee y_2) \wedge (x_2 \vee \bar{y}_1 \vee y_2) \wedge (x_1 \vee \bar{y}_2) \wedge (\bar{x}_2 \vee \bar{y}_2)$$

**Teorema 13.3**  $L_{3sat}$ , el problema de la satisfactoriedad para las expresiones 3-CNF, es NP-completo.

**Demostración** Claramente,  $L_{3sat}$  se encuentra en  $\mathcal{NP}$ , ya que  $L_{sat}$  lo está. Sea  $E = F_1 \wedge F_2 \wedge \dots \wedge F_k$  una expresión CNF. Supóngase que alguna cláusula  $F_i$  tiene más de tres literales, digamos

$$F_i = \alpha_1 \vee \alpha_2 \vee \dots \vee \alpha_\ell, \quad \ell > 3.$$

Introduzcanse las nuevas variables  $y_1, y_2, \dots, y_{\ell-3}$ , y sustitúyase  $F_i$  por

$$(\alpha_1 \vee \alpha_2 \vee y_1) \wedge (\alpha_3 \vee \bar{y}_1 \vee y_2) \wedge (\alpha_4 \vee \bar{y}_2 \vee y_3) \wedge \dots$$

$$\wedge (\alpha_{\ell-2} \vee \bar{y}_{\ell-4} \vee y_{\ell-3}) \wedge (\alpha_{\ell-1} \vee \alpha_\ell \vee \bar{y}_{\ell-3}). \quad (13.2)$$

Entonces  $F_i$  es satisfecha por alguna asignación si y sólo si una extensión de dicha asignación satisface a (13.2). Una asignación que satisface a  $F_i$  debe tener  $\alpha_j = 1$ , para alguna  $j$ . Por consiguiente, supóngase que la asignación da a las literales  $\alpha_1, \alpha_2, \dots, \alpha_{j-1}$  el valor 0 y a  $\alpha_j$  el valor 1. Entonces  $y_m = 1$  para  $m \leq j-2$ , y  $y_m = 0$ , para  $m \geq j-1$ , es una extensión de la asignación que satisface (13.2).

De manera inversa, debemos demostrar que cualquier asignación que satisface a (13.2) debe tener  $\alpha_j = 1$ , para alguna  $j$ , y, por tanto, satisface a  $F_i$ . Supóngase, por el contrario, que la asignación da a todas las  $\alpha_m$ s el valor cero. Entonces, ya que la primer cláusula tiene valor 1, se concluye que  $y_1 = 1$ . Como la segunda cláusula tiene valor 1,  $y_2$  debe ser 1, y por inducción,  $y_m = 1$ , para toda  $m$ . Pero entonces la última cláusula tendría el valor 0, contradiciendo la suposición de que (13.2) se satisface. Por consiguiente cualquier asignación que satisface a (13.2) satisface también a  $F_i$ .

Las únicas otras alteraciones que son necesarias se dan cuando  $F_i$  consiste en una o dos literales. En el último caso, sustitúyase  $\alpha_1 \vee \alpha_2$  por  $(\alpha_1 \vee \alpha_2 \vee y) \wedge (\alpha_1 \vee \alpha_2 \vee \bar{y})$ , en donde  $y$  es una nueva variable; y en el primer caso la introducción de dos nuevas variables es suficiente. Por consiguiente  $E$  puede convertirse a una expresión 3-CNF que es satisfacible si y sólo si  $E$  es satisfacible. La transformación se lleva a cabo de manera sencilla en espacio logarítmico. Tenemos, por tanto, una reducción en espacio logarítmico de  $L_{sat}$  a  $L_{3sat}$ , y concluimos que  $L_{3sat}$  es NP-completo.  $\square$

**El problema de cobertura de vértice**

Resulta que la satisfactoriedad 3-CNF es un problema conveniente para otros problemas con el fin de mostrar que son NP-completos, justo como el problema de la correspondencia de Post es útil para demostrar que otros problemas son irresolubles. Otro problema que es NP-completo y que es fácil de reducir a otros problemas es el problema de la cobertura de vértice. Sea  $G = (V, E)$  un grafo (no dirigido) con un conjunto de vértices  $V$  y aristas  $E$ . Se dice que un subconjunto  $A \subseteq V$  es una *cobertura de vértice* de  $G$  si, para cada arista  $(v, w)$  de  $E$ , al menos  $v$  o  $w$  se encuentra en  $A$ . El *problema de cobertura de vértice* es: Dado un grafo  $G$  y un entero  $k$ , ¿tiene  $G$  una cobertura de vértice de tamaño  $k$  o menor?

Podemos representar este problema como un lenguaje  $L_{vc}$ , que consiste en cadenas de la forma:  $k$  en binario, seguido de un señalador, seguido por la lista de vértices, en donde  $v_i$  está representado por  $v$  seguido de  $i$  en binario y una lista de las aristas, en donde  $(v_i v_j)$  está representada por los códigos para  $v_i$  y  $v_j$  rodeados por paréntesis.  $L_{vc}$  consiste en todas las cadenas que representan a  $k$  y  $G$ , tales que  $G$  tiene una cobertura de vértice de tamaño  $k$  o menor.

**Teorema 13.4**  $L_{vc}$ , el problema de cobertura de vértice, es NP-completo.

**Demostración** Para mostrar que  $L_{vc}$  se encuentra en  $\mathcal{NP}$ , supóngase un subconjunto de  $k$  vértices y verifíquese que éste cubre todas las aristas. Esto puede hacerse en un tiempo proporcional al cuadrado de la longitud de la representación del problema. Se demuestra que  $L_{vc}$  es NP-completo reduciendo la satisfactoriedad 3-CNF a  $L_{vc}$ .

Sea  $F = F_1 \wedge F_2 \wedge \dots \wedge F_q$  una expresión en 3-CNF, en donde cada  $F_i$  es una cláusula de la forma  $(\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3})$ , cada  $\alpha_{ij}$  es una literal. Construimos un grafo no dirigido  $G = (V, E)$  cuyos vértices son pares de enteros  $(i, j)$ ,  $1 \leq i \leq q$ ,  $1 \leq j \leq 3$ . El vértice  $(i, j)$  representa a la  $j$ -ésima literal de la  $i$ -ésima cláusula. Las aristas del grafo son:

1.  $[(i, j), (i, k)]$  siempre y cuando  $j \neq k$ , y
2.  $[(i, j), (k, \ell)]$  si  $\alpha_{ij} = \neg \alpha_{k\ell}$

Cada par de vértices correspondiente a la misma cláusula son conectados por una arista en (1). Cada par de vértices correspondiente a una literal y su complemento son conectados por una arista en (2).

$G$  ha sido construido de modo que tenga una cobertura de vértice de tamaño  $2q$  si y sólo si  $F$  es satisfacible. Para ver esto, supóngase que  $F$  es satisfacible y fíjese una asignación que satisface a  $F$ . Cada cláusula debe tener una literal cuyo valor sea 1. Selecciónese una de tales literales para cada cláusula. Elimíñese de  $V$  los  $k$  vértices correspondientes a estas literales. Los vértices restantes forman una cobertura de vértice de tamaño  $2q$ . Es claro que para cada  $i$ , solamente un vértice de la forma  $(i, j)$  no se encuentra en la cobertura y, por tanto, cada vértice de (1) incide sobre† al menos un vértice de la cobertura. Puesto que las aristas en (2) inciden sobre dos vértices correspondientes a alguna literal y su complemento, y como no podríamos haber eliminado a ambas, la literal y su complemento, uno u otro de tales vértices se encuentra en la

† Una arista  $(v, w)$  incide sobre  $v$  y  $w$  y no sobre otros vértices.

cobertura. Por consiguiente, se tiene efectivamente una cobertura de tamaño  $2q$ .

De manera inversa, supóngase que tenemos una cobertura de vértice de tamaño  $2q$ . Para cada  $i$ , la cobertura debe contener a todos los vértices de la forma  $(i, j)$ , excepto a uno, porque si dos de tales vértices estuvieran ausentes, una arista de la forma  $[(i, j), (i, k)]$  no incidiría sobre ningún vértice de la cobertura. Para cada  $i$ , asígnese el valor de 1 a la literal  $\alpha_{ij}$  correspondiente al vértice  $(i, j)$  que no está en la cobertura. No puede haber conflicto, porque dos vértices que no se encuentran en la cobertura no pueden corresponder a una literal y su complemento, también habría una arista en el grupo (2) que no incidiría sobre ningún vértice de la cobertura. Para esta asignación  $F$  tiene valor 1. Por tanto  $F$  es satisfaciente. La reducción se lleva a cabo fácilmente en espacio logarítmico. Esencialmente, podemos utilizar los nombres de las variables en la fórmula  $F$  como los vértices de  $G$ , agregando dos bits para la componente  $j$  del vértice  $(i, j)$ . Las aristas del tipo (1) se generan directamente a partir de las cláusulas, mientras que las del tipo (2) requieren dos contadores para considerar a todos los pares de literales. Por consiguiente, concluimos que  $L_{vc}$  es  $NP$ -completo.  $\square$

**Ejemplo 13.2** Considérese la expresión

$$F = (x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee x_5) \wedge (\bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5).$$

La construcción del Teorema 13.4 produce el grafo de la Fig. 13.2.  $x_1 = 1$ ,  $x_2 = 1$ ,  $x_3 = 1$ ,  $x_4 = 0$  satisface a  $F$  y corresponde a la cobertura de vértice  $[1, 2], [1, 3], [2, 1], [2, 3], [3, 1], [3, 3], [4, 1], [4, 3]$ .

### El problema circuital de Hamilton

El *problema circuital de Hamilton* es: Dado un grafo, ¿tiene  $G$  una trayectoria que toca a cada vértice exactamente una vez y regresa a su punto inicial? El *problema circuital de Hamilton dirigido* es el problema análogo para los grafos dirigidos. Representamos estos problemas como lenguajes  $L_h$  y  $L_{dh}$  codificando los grafos de la misma forma que en el problema de cobertura de vértices.

**Teorema 13.5**  $L_{dh}$ , el problema circuital de Hamilton dirigido, es  $NP$ -completo.

*Demostración* Para demostrar que  $L_{dh}$  se encuentra en  $NP$ , supóngase una lista de arcos y verifíquese que éstos forman un ciclo simple† a través de todos los vértices. Para demostrar que  $L_{dh}$  es  $NP$ -completo, reducimos la satisfactoriedad 3-CNF a  $L_{dh}$ .

Sea  $F = F_1 \wedge F_2 \wedge \dots \wedge F_q$  una expresión en 3-CNF, en la que cada  $F_i$  es una cláusula de la forma  $(\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3})$ , cada  $\alpha_{ij}$  es una literal. Sean  $x_1, \dots, x_n$  las variables de  $F$ . Construimos un grafo dirigido  $G$  que está compuesto de dos tipos de subgrafos. Para cada variable  $x_i$  existe un subgrafo  $H_i$  de la forma en que se muestra en la Fig. 13.3(a), en donde  $m_i$  es el mayor de los números de veces que aparece  $x_i$  y  $\bar{x}_i$  en  $F$ . Las  $H_i$ s están conectadas en un ciclo, como se muestra en la Fig. 13.3(b). Esto es, existen arcos que van de  $d_i$  a  $a_{i+1}$ , para  $1 \leq i \leq n$ , y un arco que va de  $d_n$  a  $a_1$ .

† Un ciclo simple no tiene vértices repetidos.

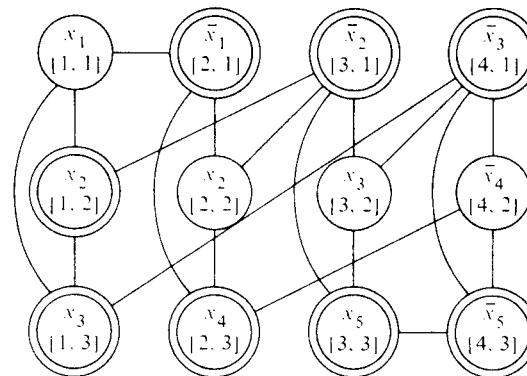


Fig. 13.2 Grafo construido por el Teorema 13.4.  
Los círculos dobles indican vértices que se encuentran en la cobertura de conjunción.

Supóngase que tenemos un circuito de Hamilton para el grafo de la Fig. 13.3(b). Podemos también suponer que empieza en  $a_1$ . Si enseguida va a  $b_{10}$ , exigimos que entonces debe ir hacia  $c_{10}$ , de otro modo  $c_{10}$  podría nunca aparecer en el ciclo. Para la demostración, nótese que ambos predecesores de  $c_{10}$  ya se encuentran en el ciclo, y para que el ciclo alcance más tarde a  $c_{10}$  tendría que repetir un vértice. (Este argumento con respecto a los ciclos de Hamilton se da con frecuencia en la demostración. Simplemente diremos que un vértice como  $c_{10}$  “se haría inaccesible”.) De manera similar, podemos argumentar que un circuito de Hamilton que comienza en  $a_1, b_{10}$  debe continuar con  $c_{10}, b_{11}, c_{11}, b_{12}, c_{12}, \dots$ . Si el circuito comienza en  $a_1, c_{10}$ , entonces desciende la escalera de la Fig. 13.3(a) en el lado opuesto, continuando con  $b_{10}, c_{11}, b_{11}, c_{12}, \dots$ . De manera similar podemos argumentar que cuando el circuito entra en cada  $H_i$ , a su vez debe ir de  $a_i$  a  $b_{10}$  o a  $c_{10}$ , pero entonces su trayectoria a través de  $H_i$  está fija; en el primer caso desciende por los arcos  $c_{ij} \rightarrow b_{i,j+1}$ , y en el segundo, por los arcos  $b_{ij} \rightarrow c_{i,j+1}$ . En lo que sigue, resulta de ayuda pensar en la alternativa que va de  $a_i$  a  $b_{10}$  como la que hace a  $x_i$  verdadera, mientras que la alternativa opuesta hace a  $x_i$  falsa. Teniendo esto en cuenta, obsérvese que el grafo de la Fig. 13.3(b) tiene exactamente  $2^n$  circuitos de Hamilton que corresponden, en una forma natural, a las asignaciones a las variables de  $F$ .

Para cada cláusula  $F_j$  introducimos un subgrafo  $I_j$ , el cual se muestra en la Fig. 13.3(c).  $I_j$  tiene las características que consisten en que si un circuito de Hamilton entra en él en  $r_j$ , debe dejarlo en  $u_j$ ; si entra en  $s_j$ , debe abandonarlo en  $v_j$ ; y si entra en  $t_j$ , debe dejarlo en  $w_j$ . Para la demostración, supóngase, por simetría, que el circuito entra a  $I_j$  en  $r_j$ .

**CASO 1** Los siguientes dos vértices del circuito son  $s_j$  y  $t_j$ . Entonces el circuito debe continuar con  $w_j$ , y si lo abandona en  $w_j$  o en  $v_j$ ,  $u_j$  se vuelve inaccesible. Así, en este caso abandona en  $u_j$ .

**CASO 2** Los siguientes dos vértices del circuito son  $s_j$  y  $v_j$ . Si el circuito no va

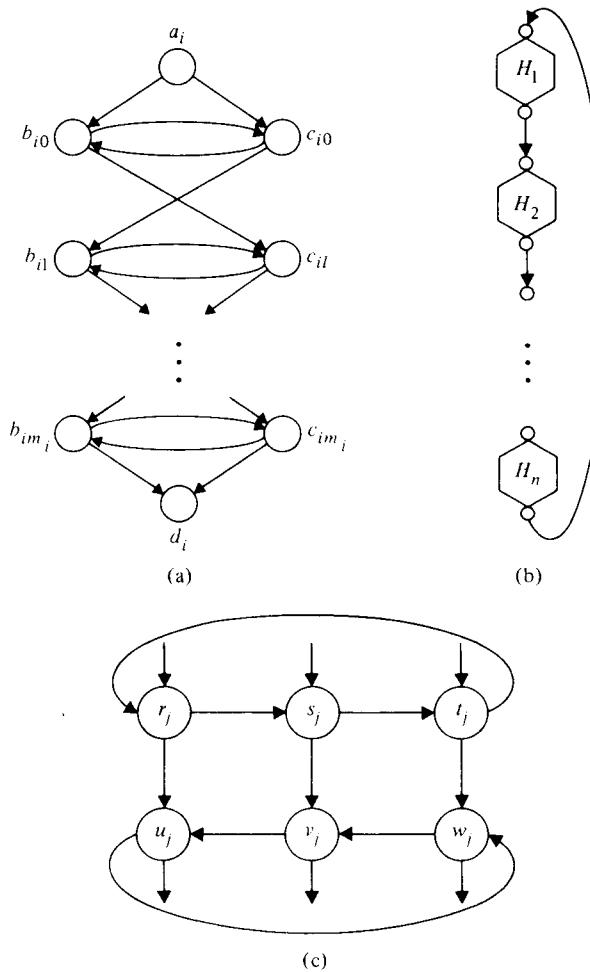


Fig. 13.3 Grafo concerniente a los circuitos de Hamilton dirigidos.

enseguida a  $u_j$ , entonces éste será inaccesible. Si después de  $u_j$  va a  $w_j$ , el vértice  $t_j$  no puede aparecer en el circuito porque sus sucesores ya se encuentran en el circuito. Por consiguiente en este caso el circuito también sale por  $u_j$ .

**CASO 3** El circuito va directamente a  $u_j$ . Si enseguida va a  $w_j$ , el circuito no puede incluir a  $t_j$ , debido a que sus sucesores ya se han utilizado. De modo que, de nuevo, debe dejar el subgrafo en  $u_j$ .

Obsérvese que el argumento dado más arriba es válido aun cuando el circuito pueda entrar a  $I_j$  más de una vez. Finalmente, el grafo  $I_j$  tiene la propiedad adicional de que al entrar en él en  $r_j$ ,  $s_j$  o en  $t_j$  puede cruzar los seis vértices antes de que existan. Para completar la construcción del grafo, conéctense los  $I_j$ s con los  $H_i$ s de la manera

siguiente. Supóngase que el primer término de  $F_j$  es  $x_i$ . Entonces escójase algún  $c_{ip}$  que aún no ha sido conectado con cualquier  $I_j$ , e introduzcase un arco de  $c_{ip}$  a  $r_j$  y de  $u_j$  a  $b_{i,p+1}$ . Si el primer término es  $\bar{x}_i$ , tómese un  $b_{ip}$  que no haya sido utilizado e introduzcanse arcos  $b_{ip} \rightarrow r_j$  y  $u_j \rightarrow c_{i,p+1}$ . Háganse conexiones parecidas con  $s_j$  y  $v_j$  para el segundo término de  $F_j$ ; y con  $t_j$  y  $w_j$  para el tercer término. Cada  $H_i$  fue escogida con la longitud suficiente para asegurarnos que suficientes pares de  $b_{ij}$ s y  $c_{ij}$ s están disponibles para hacer todas las conexiones.

Si la expresión  $F$  es satisfaciente, podemos encontrar un circuito de Hamilton para el grafo de la manera siguiente. Hagamos que el circuito vaya de  $a_i$  a  $b_{i0}$  si  $x_i$  es verdadera en la asignación que satisface, y de  $a_i$  a  $c_{i0}$  en cualquier otro caso. Entonces, ignorando los  $I_j$ s, tenemos un circuito de Hamilton único para el subgrafo de la Fig. 13.3(b). Ahora, siempre que el circuito construido utilice un arco  $b_{ik} \rightarrow c_{i,k+1}$  o  $c_{ik} \rightarrow b_{i,k+1}$ , y  $b_{ik}$  o  $c_{ik}$ , respectivamente, tiene un arco a un subgrafo  $I_j$ , que aún no ha sido tocado por el circuito, visítese a los seis vértices de  $I_j$  saliendo por  $c_{i,k+1}$  o por  $b_{i,k+1}$ , respectivamente. El hecho de que es satisfaciente implica que podemos cruzar  $I_j$  para toda  $j$ .

De manera inversa, debemos mostrar que la existencia de un circuito de Hamilton implica que  $F$  es satisfaciente. Recuérdese que en cualquier circuito de Hamilton si se entra en un  $I_j$  en  $r_j$ ,  $s_j$  o  $t_j$  se debe salir de él en un  $u_j$ ,  $v_j$  o  $w_j$ , respectivamente. Por consiguiente, en lo que a las trayectorias que pasan por  $H_i$  concierne, las conexiones a un  $I_j$  se ven como arcos en paralelo con un arco  $b_{ik} \rightarrow c_{i,k+1}$  o  $c_{ik} \rightarrow b_{i,k+1}$ . Si se ignoran las excursiones a los  $I_j$ s, se concluye que el circuito debe cruzar a los  $H_i$ s en una de las 2<sup>n</sup> formas que son posibles sin tomar en cuenta a los  $I_j$ s; esto es, se puede seguir el arco  $a_i \rightarrow b_{i0}$  o  $a_i \rightarrow c_{i0}$  para  $1 \leq i \leq n$ . Cada conjunto de alternativas determina una asignación de verdad para las  $x_i$ s. Si un conjunto de alternativas produce un circuito de Hamilton, incluyendo a los  $I_j$ s, entonces la asignación debe satisfacer a todas las cláusulas. Por ejemplo, si alcanzamos a  $I_j$  a partir de  $b_{ik}$  en el circuito, entonces  $\bar{x}_i$  es un término de  $F_j$ ; y puede suceder que el circuito vaya de  $a_i$  a  $c_{i0}$ , lo que corresponde a la alternativa  $x_i = 0$ . Nótese que si el circuito va de  $a_i$  a  $b_{i0}$ , entonces debe cruzar  $b_{i,k+1}$  antes que  $c_{i,k+1}$  y podríamos no cruzar  $I_j$  entre  $b_{ik}$  y  $c_{i,k+1}$ , como  $b_{i,k+1}$  podría no ser nunca incluido en el circuito.

Como una última anotación, debemos demostrar que tenemos una reducción en espacio logarítmico. Dada  $F$ , podemos listar los vértices y los arcos de  $H_i$  simplemente contando las veces que aparecen  $x_i$  y  $\bar{x}_i$ . Podemos hacer una lista de las conexiones entre las  $H_i$ s y las  $I_j$ s también de una manera sencilla. Dado un término como  $x_i$  en  $F_j$ , podemos encontrar un par de vértices libres en  $H_i$  para conectar a  $I_j$  contando las veces que aparece  $x_i$  en  $F_1, F_2, \dots, F_{j-1}$ . Como ninguna cuenta llega más allá del número de variables o de cláusulas, un espacio log  $n$  es suficiente, en donde  $n$  es la longitud de  $F$ .  $\square$

### Ejemplo 13.3 Sea $F$

$$(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3).$$

En la Fig. 13.4 se muestra el grafo construido a partir de  $F$  según el Teorema 13.5. Con líneas gruesas se dibuja el circuito de Hamilton correspondiente a la asignación  $x_1 = 1$ ,  $x_2 = 0$ ,  $x_3 = 0$ .

Finalmente mostraremos que el problema circuital de Hamilton es *NP*-completo mediante la reducción del problema circuital de Hamilton dirigido a éste.

**Teorema 13.6**  $L_h$ , el problema circuital de Hamilton para grafos no dirigidos, es *NP*-completo.

**Demostración** Para demostrar que  $L_h$  está en  $\mathcal{NP}$ , supóngase una lista de aristas y verifíquese que forman un circuito de Hamilton. Para mostrar que  $L_h$  es *NP*-completo reducimos  $L_{dh}$  a  $L_h$ . Sea  $G = (V, E)$  un grafo dirigido. Constrúyase un grafo no dirigido  $G'$  con vértices  $v_0, v_1$  y  $v_2$ , para cada  $v \in V$ , y aristas

1.  $(v_0, v_1)$  para cada  $v \in V$ ,
2.  $(v_1, v_2)$  para cada  $v \in V$ ,
3.  $(v_2, w_0)$  si y sólo si  $v \rightarrow w$  es un arco en  $E$ .

Cada vértice de  $V$  ha sido expandido a tres vértices. Los vértices con subíndice 1 tienen sólo dos aristas y, como un circuito de Hamilton debe visitar a todos los vértices, el subíndice de los vértices de cualquier circuito de Hamilton de  $G'$  deben estar en el orden  $0, 1, 2, 0, 1, \dots$  o a la inversa. Supóngase que el orden es  $0, 1, 2, \dots$ . Entonces las aristas cuyos subíndices van de 2 a 0 corresponden a un circuito de Hamilton en  $G$ . De manera inversa, un circuito de Hamilton de  $G$  puede convertirse a un circuito de Hamilton en  $G'$  mediante la sustitución de un arco  $v \rightarrow w$ , a través de la trayectoria de  $v_0$  a  $v_1$  a  $v_2$  a  $w_0$ . Por consiguiente  $G'$  tiene un circuito de Hamilton si y sólo si  $G$  tiene un circuito de Hamilton. La reducción de  $G$  a  $G'$  se lleva a cabo de manera sencilla en espacio logarítmico. Por tanto concluimos que  $L_h$  es *NP*-completo.  $\square$

### Programación lineal entera

Es fácilmente demostrable que la mayoría de los problemas *NP*-completos se encuentran en  $\mathcal{NP}$ , y solamente la reducción a partir de un problema *NP*-completo conocido es difícil. Daremos ahora un ejemplo de un problema en el que se da lo opuesto a lo dicho. Es fácil demostrar que la programación lineal entera es *NP*-difícil pero resulta difícil demostrar que se encuentra en  $\mathcal{NP}$ . El problema de la programación lineal entera es: Dada una matriz de enteros  $m \times n$ ,  $A$ , y un vector columna  $b$  de  $n$  enteros, ¿existe un vector columna de enteros  $x$  tal que  $Ax \geq b$ ? El lector puede formalizar este problema como un lenguaje de una forma obvia, en la que las palabras del lenguaje son los elementos de  $A$  y  $b$  escritos en binario.

**Lema 13.4** La programación lineal entera es *NP*-difícil.

**Demostración** Reducimos la satisfactoriedad 3-CNF a la programación lineal entera. Sea  $E = F_1 \wedge F_2 \wedge \dots \wedge F_q$  una expresión en 3-CNF y sean  $x_1, x_2, \dots, x_n$  las variables de  $E$ . La matriz  $A$  tendrá una columna para cada literal  $x_i$  o  $\bar{x}_i$ ,  $1 \leq i \leq n$ . Podemos, por

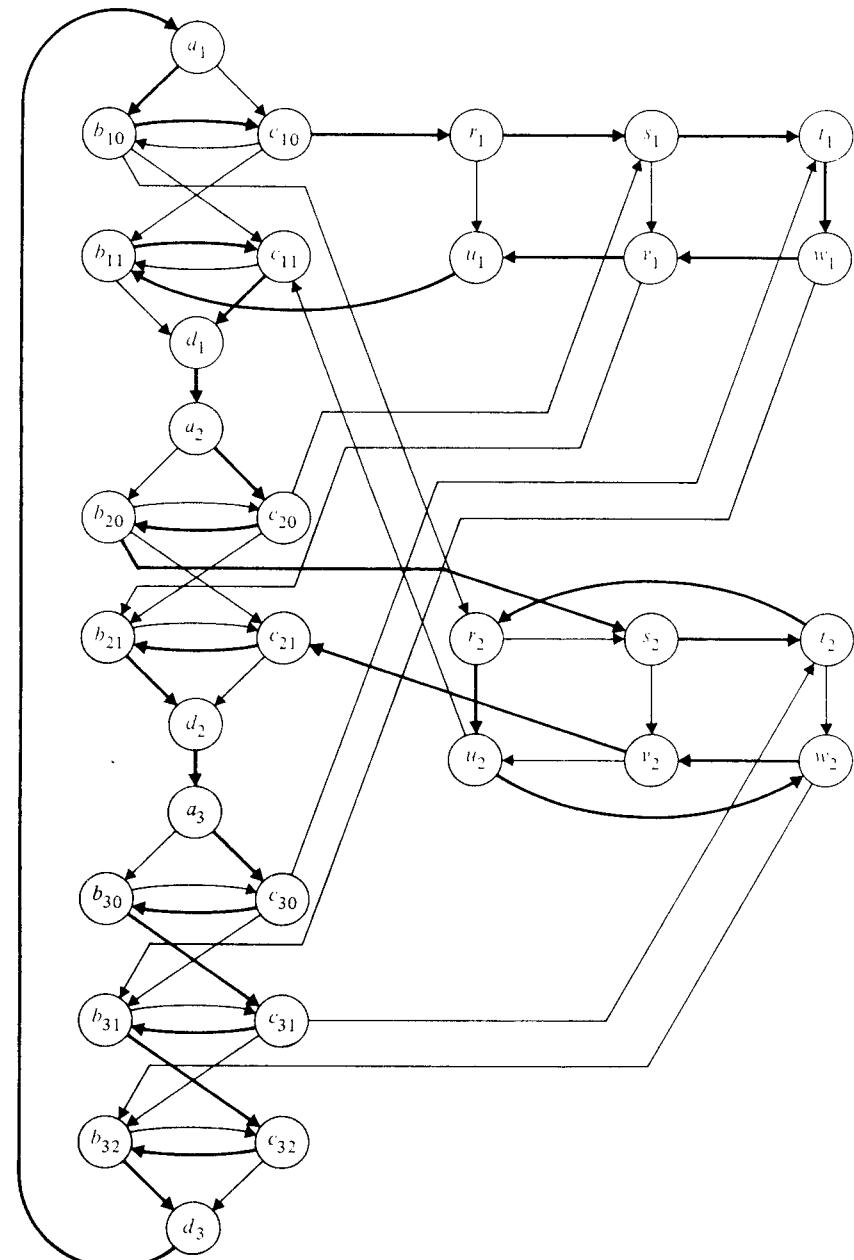


Fig. 13.4 Grafo construido para el Ejemplo 13.3.

tanto, ver la desigualdad  $Ax \geq b$  como un conjunto de desigualdades lineales entre las literales. Para cada  $i$ ,  $1 \leq i \leq n$ , tenemos las desigualdades

$$\begin{aligned} x_i + \bar{x}_i &\geq 1, & x_i \geq 0, \\ -x_i - \bar{x}_i &\geq -1, & \bar{x}_i \geq 0, \end{aligned}$$

que tienen el efecto de decir que  $x_i$  o  $\bar{x}_i$  es 0, y la otra es 1. Para cada cláusula  $\alpha_1 \vee \alpha_2 \vee \alpha_3$ , tenemos la desigualdad

$$\alpha_1 + \alpha_2 + \alpha_3 \geq 1,$$

que dice que al menos una literal en cada cláusula tiene valor 1. Resulta obvio que  $A$  y  $b$  pueden construirse en espacio logarítmico y las desigualdades se satisfacen todas si y sólo si  $E$  es satisfaciente. Por tanto la programación lineal es *NP*-difícil.  $\square$

Para mostrar que la programación lineal entera se encuentra en *NP*, podemos adivinar un vector  $x$  y verificar que  $Ax \geq b$ . Sin embargo, si la solución más pequeña tiene elementos que son demasiado grandes, podemos no ser capaces de escribir  $x$  en un tiempo polinomial. La dificultad consiste en mostrar que los elementos de  $x$  no necesitan ser muy grandes y, para llevar a cabo esto, necesitamos algunos conceptos del álgebra lineal, específicamente sobre determinantes de matrices cuadradas, el rango de una matriz, dependencia lineal de vectores y la regla de Cramer para resolver ecuaciones lineales simultáneas, conceptos todos éstos que, esperamos, le resulten familiares al lector.

En lo que sigue, suponemos que la matriz  $A$  y el vector  $b$  forman un ejemplo del problema de programación lineal entera y que  $A$  tiene  $m$  renglones y  $n$  columnas. Sea  $\alpha$  la magnitud del elemento más grande de  $A$  o  $b$ . Nótese que el número de bits que se necesitan para escribir  $A$  y  $b$  es de al menos  $mn + \log_2 \alpha$  y utilizaremos esta cantidad como un límite inferior del tamaño de la entrada; nuestro enunciador de solución no determinística trabajará en  $\text{TIEMPO}(p(mn + \log_2 \alpha))$  para algún polinomio  $p$ . Aún más, definimos  $a_i$ ,  $1 \leq i \leq m$ , como el vector de longitud  $n$  que consiste en el  $i$ -ésimo renglón de  $A$ . Hacemos que  $b_i$  sea el  $i$ -ésimo elemento de  $b$  y  $x = (x_1, x_2, \dots, x_n)$  un vector de incógnitas. Utilizamos  $|i|$  como la magnitud del entero  $i$  y  $\det B$  como el determinante de la matriz  $B$ . Necesitamos una serie de lemas técnicos.

**Lema 13.5** Si  $B$  es una submatriz cuadrada de  $A$ , entonces  $|\det B| \leq (\alpha q)^r$ , en donde  $q = (\max(m, n))$ .

**Demostración** Recuérdese que el determinante de una matriz  $k \times k$  es la suma o diferencia de  $k!$  términos, cada uno de los cuales es el producto de  $k$  elementos. Por consiguiente, si  $B$  es una submatriz  $k \times k$ ,  $k! \alpha^k$  es un límite superior de  $|\det B|$ . Como  $k! \leq k^k$  y  $k \leq q$ , tenemos la demostración de nuestro lema.  $\square$

**Lema 13.6** Sea  $A$  una matriz de rango  $r$ .<sup>†</sup> Si  $r < n$ , entonces existe un vector entero

<sup>†</sup> Recuérdese que el *rango* de  $A$  se define de manera equivalente como el número máximo de renglones linealmente independientes, el número máximo de columnas linealmente independientes o el tamaño de la submatriz cuadrada más grande que tenga un determinante diferente de cero.

$z = (z_1, z_2, \dots, z_n)$ , con  $z$  no idénticamente cero, tal que  $Az = 0$  ( $0$  es un vector cuyos elementos son todos 0) y ninguna de las  $z_i$  excede en magnitud a  $(\alpha q)^{2r}$ , en donde  $q = \max(m, n)$ .

**Demostración** Supóngase, sin pérdida de la generalidad, que  $B$ , la submatriz  $r \times r$  de  $A$  que se encuentra en la esquina superior izquierda, tiene un determinante diferente de cero. Sea  $C$  la matriz formada con los primeros  $r$  renglones de  $A$  y sea  $D$  la matriz formada con los últimos  $m - r$  renglones de  $A$ . Como cualesquiera  $r + 1$  renglones de  $A$  son linealmente dependientes y los renglones de  $C$  son linealmente independientes (debido a que  $B$  tiene un determinante diferente de cero), cada renglón de  $D$  puede expresarse como una combinación lineal de renglones de  $C$ . Esto es  $D = EC$ , para alguna matriz  $(m - r) \times r E$ . Entonces  $Az = 0$  si y sólo si  $Cz = 0$  y  $ECz = 0$ . Es suficiente, por tanto, demostrar que podemos hacer  $Cz = 0$ . Si escogemos  $z_n = -1$  y  $z_{r+1} = z_{r+2} = \dots = z_{n-1} = 0$ , entonces  $Cz = 0$  si y sólo si  $By = w$ , en la que  $y$  es el vector  $(z_1, z_2, \dots, z_r)$  y  $w$  es la  $n$ -ésima columna de  $C$ . Según la regla de Cramer,  $By = w$  se satisface si tomamos  $z_i = \det B_i / \det B$ , en donde  $B_i$  es la matriz  $B$  en la cual la  $i$ -ésima columna ha sido sustituida por  $w$ . Según el Lema 13.5, estos determinantes no exceden en magnitud a  $(\alpha q)^r$ . La  $z$  resultante puede no tener componentes enteros, pero si multiplicamos todos los componentes por  $\det B$ , éstos serán enteros y satisfarán  $Az = 0$ . Cuando hacemos lo anterior,  $z_n = -\det B$ ; las magnitudes de las primeras  $r$  componentes de  $z$  no exceden en magnitud a  $((\alpha q)^r)^2 = (\alpha q)^{2r}$ , y las componentes que van de  $r + 1$  hasta  $n - 1$  son cero.  $\square$

Se concluye que la solución  $z$  se puede escribir en un número de bits que es como mucho la segunda potencia de  $mn + \log_2 \alpha$ , el tamaño del planteamiento del problema.

**Lema 13.7** Sea  $A$  una matriz con al menos un elemento diferente de cero. Si existe una solución a  $Ax \geq b$ ,  $x \geq 0$ , entonces existe una solución en la cual, para alguna  $i$ ,  $b_i \leq a_i x_i < b_i + \alpha$ , en la que  $\alpha$  es la magnitud del elemento más grande de  $A$ .

**Demostración** Sea  $a_i x_0$  una solución a  $Ax \geq b$ . Supóngase que  $a_i x_0 \geq b_i + \alpha$ , para toda  $i$ . Sumando o restando 1 de alguna componente de  $x_0$  debemos reducir algún producto  $a_i x_0$ . Aún más, ningún producto puede disminuir en un factor mayor que  $\alpha$ . Por consiguiente  $x$  es también una solución. El proceso no puede repetirse de manera indefinida sin que se obtenga una solución  $x$  para la cual existe una  $i$  tal que  $b_i \leq a_i x_i < b_i + \alpha$ .  $\square$

**Teorema 13.7** La programación lineal entera es *NP*- completa.

**Demostración** Según el Lema 13.4 tenemos que demostrar solamente que el problema se encuentra en *NP*. Comenzaremos adivinando los signos de las  $x_i$ s de alguna solución hipotética y sumando  $n$  restricciones  $x_i \leq (\geq) 0$ . Dependiendo del signo adivinado. Después adivinéscse un renglón  $i$  y una constante  $c_i$  que estén en el intervalo  $b_i \leq c_i < b_i + \alpha$ , tal que en alguna solución  $x_0$ , tenemos  $a_i x_0 = c_i$ . Supóngase ahora que después de reordenar renglones, si esto fuera necesario, hemos adivinado correctamente las  $c_1, c_2, \dots, c_k$  tal que

$$1. b_i \leq c_i < b_i + (\alpha q)^{2r+1} \text{ y}$$

2.  $Ax \geq b$  tiene una solución no negativa si y sólo si

$$\begin{aligned} \mathbf{a}_i \mathbf{x} &= c_i, & 1 \leq i \leq k, \\ \mathbf{a}_i \mathbf{x} &\geq b_i, & k < i \leq m, \end{aligned}$$

tienen dicha solución.

Sea  $A_k$  la matriz compuesta por los primeros  $k$  renglones de  $A$ , y sea  $c$  el vector  $(c_1, c_2, \dots, c_k)$ .

**CASO 1** El rango de  $A_k$  es menor que  $n$ . Según el Lema 13.6 existe un vector entero  $\mathbf{z}, \mathbf{z} \neq 0$ , en el que ninguno de sus componentes tiene una magnitud mayor que  $(\alpha q)^{2q}$ , tal que  $A_k \mathbf{z} = 0$ . Por consiguiente, si  $A_k \mathbf{x}_0 = \mathbf{c}$ , se concluye que  $A_k(\mathbf{x}_0 + d\mathbf{z}) = \mathbf{c}$ , para cualquier entero  $d$ . Si también es verdad que  $\mathbf{a}_j \mathbf{x}_0 \geq b_j + (\alpha q)^{2q+1}$ , para toda  $j > k$ , entonces podemos, de manera repetida, agregar o restar 1 de  $d$  hasta que, para alguna  $j \geq k$ ,  $\mathbf{a}_j(\mathbf{x}_0 + d\mathbf{z})$  caiga por debajo de  $b_j + (\alpha q)^{2q+1}$ . Como  $\mathbf{z}$  tiene alguna componente diferente de cero, el renglón  $\mathbf{a}_j$  [correspondiente a la restricción  $x_j \leq (\geq) 0$ ] que tiene puros ceros, excepto en la componente  $x_j$ , debe tener  $\ell \geq k$ . Por consiguiente alguna  $\mathbf{a}_j(\mathbf{x}_0 + d\mathbf{z})$ , para  $j \geq k$ , deberá, en algún momento, caer por debajo de  $b_j + (\alpha q)^{2q+1}$ . Como cada componente de  $\mathbf{z}$  está acotada en magnitud por  $(\alpha q)^{2q}$ , cambiar  $d$  por 1 no puede cambiar a cualquier  $\mathbf{a}_j(\mathbf{x}_0 + d\mathbf{z})$  en más de  $\alpha n(\alpha q)^{2q}$ , que no es mayor que  $(\alpha q)^{2q+1}$ . En consecuencia,  $\mathbf{a}_j(\mathbf{x}_0 + d\mathbf{z}) \geq b_j$ . Reordenando renglones podemos suponer  $j = k + 1$  y repetir el proceso anterior para  $k + 1$  en lugar de  $k$ .

**CASO 2** El rango de  $A_k$  es  $n$ . En este caso, existe un único  $\mathbf{x}$  que satisface  $A_k \mathbf{x} = \mathbf{c}$ . Según la regla de Cramer, las componentes de  $\mathbf{x}$  son cocientes de dos determinantes cuyas magnitudes no exceden a  $q^q(\alpha + (\alpha q)^{2q+1})\alpha^{q-1}$ , que es menor que  $(2\alpha q)^{2q+1}$ . Podemos verificar si  $\mathbf{x}$  consiste solamente en enteros y satisface a  $\mathbf{a}_j \mathbf{x} \geq b_j$ , para  $j > k$ .

El proceso no determinístico de adivinar o suponer  $c_i$ 's se repite cuando más  $n$  veces y, para cualquier secuencia de alternativas, se requiere un cierto número de pasos aritméticos que es un polinomio en  $q$  [ya que la regla de Cramer puede aplicarse en  $O(r^4)$  pasos aritméticos a las matrices  $r \times r$ ] aplicado a los enteros cuya longitud en binario es un polinomio en  $\alpha q$ . Los pasos aritméticos que consisten en multiplicación o división de enteros pueden llevarse a cabo en un tiempo que es proporcional al cuadrado de la longitud de los enteros en binario† y la suma y la resta pueden efectuarse en un tiempo lineal. Por tanto, el proceso entero toma un tiempo que es un polinomio en la longitud de la entrada, ya que la longitud es de al menos  $mn + \log_2 \alpha$ .  $\square$

### Otros problemas NP-completos

Existe una amplia variedad de otros problemas NP-completos que se conocen. Enlistaremos algunos de ellos aquí.

1. *El problema de número cromático.* Dados un grafo  $G$  y un entero  $k$ , ¿puede  $G$  ser coloreado con  $k$  colores de modo que ninguno de los dos vértices adyacentes queden del mismo color?

† En realidad se puede efectuar en un tiempo considerablemente menor (véase Aho, Hopcroft y Ullman [1974]), aunque esto no tiene importancia en este caso.

2. *El problema del agente viajero.* Dado un grafo completo con pesos en sus aristas, ¿cuál es el circuito de Hamilton de menor peso? Para expresar este problema como un lenguaje necesitamos que los pesos sean enteros y preguntarnos si existe un circuito de Hamilton con peso  $k$  o menor. Este problema es *NP*-completo, aun si restringimos los pesos a 0 y 1, en cuyo caso se convierte exactamente en el problema circuital de Hamilton.

3. *El problema de la cobertura exacta.* Dada una colección de conjuntos  $S_1, S_2, \dots, S_k$ , en la cual todos los conjuntos son subconjuntos de algún conjunto  $U$ , ¿existe una subcolección, cuya unión es  $U$ , tal que cada par de conjuntos que se encuentren en la subcolección son disjuntos?

4. *El problema de la partición.* Dada una lista de enteros  $i_1, i_2, \dots, i_k$ , ¿existe un subconjunto cuya suma es exactamente  $\frac{1}{2}(i_1 + i_2 + \dots + i_k)$ ? Nótese que este problema parece estar en  $\text{P}$  hasta que recordamos que la longitud de una instancia no es  $i_1 + i_2 + \dots + i_k$ , sino la suma de la longitud de las  $i_j$ s escrita en binario o en alguna otra base fija.

Entre los problemas *NP*-completos hay muchos, incluidos los mencionados en esta sección, a los cuales se les ha dedicado un serio esfuerzo con el fin de encontrarlos algoritmos en tiempo polinomial. Como todos los problemas *NP*-completos se encuentran en  $\text{NP}$  o ninguno lo está, y como hasta ahora no se ha encontrado uno que se encuentre en  $\text{P}$ , resulta natural hacer la conjectura de que ninguno se encuentra en  $\text{P}$ . Lo que es más importante, si uno tiene que resolver un problema *NP*-completo, es cuestionable el hecho de si uno debería molestarse en buscar un algoritmo en tiempo polinomial. Tenemos la creencia de que es más acertado buscar la heurística que trabajar bien sobre las clases particulares de instancias que es posible que uno encuentre.

### Significancia extendida de la integridad *NP*

Sin darnos cuenta hemos implicado que el único problema con respecto a los problemas *NP*-completos es si éstos requieren un tiempo polinomial o exponencial. De hecho, la respuesta correcta podría estar entre estos extremos; por ejemplo, podrían necesitar un tiempo  $n^{\log n}$ . Si todos los lenguajes que se encuentran en  $\text{NP}$  son reducibles en espacio logarítmico o aun en tiempo polinomial a  $L$ , y  $L$  está en, digamos,  $\text{TIEMPOD}(n^{\log n})$ , entonces cada lenguaje en  $\text{NP}$  se encuentra en  $\text{TIEMPOD}(n^{\log n})$ , para alguna constante  $c$ . En general, si  $L$  fuera completo en espacio logarítmico o en tiempo polinomial para  $\text{NP}$ , y  $L$  estuviera en  $\text{TIEMPOD}(T(n))$ , entonces

$$\text{NP} \subseteq \bigcup_{c \geq 0} \text{TIEMPOD}(T(n^c))$$

### 13.3 LA CLASE CO-NP

No se sabe si la clase  $\text{NP}$  es cerrada con respecto a la complementación. De suceder que  $\text{NP}$  no es cerrado con respecto a la complementación, entonces  $\text{P} \neq \text{NP}$ , ya que  $\text{P}$  es cerrado con respecto a la complementación. No existe ningún problema *NP*-completo que sea complementario de otro problema *NP*-completo.

completo, cuyo complemento se sepa que esté en  $\text{NP}$ . Por ejemplo, para determinar la no satisfactoriedad para una fórmula booleana con  $n$  variables, parece necesario probar cada una de las  $2^n$  asignaciones posibles, aun si el algoritmo es no determinístico. De hecho, si se descubre que cualquier problema  $NP$ -completo tiene su complemento en  $\text{NP}$ , entonces  $\text{NP}$  sería cerrado con respecto a la complementación, como lo demostraremos en el siguiente teorema.

**Teorema 13.8**  $\text{NP}$  es cerrado con respecto a la complementación si y sólo si el complemento de algún problema  $NP$ -completo se encuentra en  $\text{NP}$ .

*Demostración* La parte “sólo si” es obvia. Para la parte “si” sea  $\bar{S}$  un problema  $NP$ -completo y supóngase que  $S$  estuviera en  $\text{NP}$ . Ya que cada  $L$  de  $\text{NP}$  es reducible en espacio logarítmico a  $\bar{S}$ , cada  $L$  es reducible en espacio logarítmico a  $\bar{S}$ . Por consiguiente  $L$  se encuentra en  $\text{NP}$ .  $\square$

Definiremos la clase  $\text{co-}\text{NP}$  como el conjunto de complementos de los lenguajes que se encuentran en  $\text{NP}$ . En la Fig. 13.5 se muestra la relación entre  $\text{P}$ ,  $\text{NP}$ ,  $\text{co-}\text{NP}$  y  $\text{ESPACIOP}$ , aunque se tiene la certeza de que todas las regiones, excepto la que tiene la etiqueta  $\text{P}$ , no se encuentran vacías.



Fig. 13.5 Relaciones entre algunas clases de lenguajes.

### El problema de la primalidad

Resulta interesante considerar un problema en  $\text{NP}$  como la “no excelencia”, para la cual no existe un algoritmo conocido en tiempo polinomial† y, aún más, no se sabe que sea  $NP$ -completo. †† Para probar si un entero no es primo, simplemente suponemos un divisor y lo verificamos. La observación interesante consiste en que el problema

† Aunque Miller [1976] presenta una fuerte evidencia de que sí existe.

†† Este es otro problema que parece que se encuentra en  $\text{P}$ , hasta que uno recuerda que el tamaño de la entrada  $p$  es  $\log p$ , no  $p$ .  $\square$

complementario se encuentra en  $\text{NP}$ , lo que no sugiere que pueden existir conjuntos en la intersección de  $\text{NP}$  y  $\text{co-}\text{NP}$  que no se encuentren en  $\text{P}$ .

Consideraremos, ahora, un algoritmo no determinístico en tiempo polinomial para probar si un entero es primo.

**Lema 13.8** Sean  $x$  y  $y$  enteros, con  $0 \leq x, y < p$ . Entonces

1.  $x + y \pmod p$  puede calcularse en un tiempo  $O(\log p)$ ;
2.  $xy \pmod p$  puede calcularse en tiempo  $O(\log^2 p)$ ;
3.  $x^y \pmod p$  puede calcularse en tiempo  $O(\log^3 p)$ .

*Demostración* (1) y (2) resultan obvios, ya que un entero módulo  $p$  requiere solamente  $\log p$  bits. Para (3) calcúlese  $x^y$  elevando  $x$  al cuadrado, de manera repetida, para obtener  $x^2, x^4, x^8, \dots, x^{2^i} \pmod p$ , en donde  $i = [\log_2 y]$ , entonces multiplíquense las potencias apropiadas de  $x$  para obtener  $x^y$ .  $\square$

En lo que sigue, haremos uso del teorema de Fermat;  $p > 2$  es un primo si y sólo si existe una  $x$  de orden  $p-1$ , esto es, para alguna  $x$ ,  $1 < x < p$ ,

1.  $x^{p-1} \equiv 1 \pmod p$ , y
2.  $x^i \not\equiv 1 \pmod p$ , para  $1 \leq i \leq p-1$ .

**Teorema 13.9** El conjunto de los números primos está en  $\text{NP}$ .

*Demostración* Si  $x = 2$ , entonces  $x$  es un primo. Si  $x = 1$  o  $x$  es un entero par mayor que 2, entonces  $x$  no es primo. Para determinar si  $p$  es primo, para  $p$  impar mayor que 2, supóngase una  $x$ ,  $0 < x < p$ , y verifíquese que

1.  $x^{p-1} \equiv 1 \pmod p$ , y
2.  $x^i \not\equiv 1 \pmod p$ , para toda  $i$ ,  $1 \leq i \leq p-1$ .

La condición (1) se verifica en  $O(\log^3 p)$  pasos. No podemos verificar la condición (2) para cada  $i$  de manera directa, ya que existen demasiadas  $i$ s. En lugar de esto, supóngase la factorización en números primos de  $p-1$ . Sea la factorización  $p-1 = p_1 p_2 \dots p_k$ . Verifíquese de manera recursiva si cada  $p_j$  es un primo. Verifíquese si  $p-1$  es el producto de las  $p_j$ s. Finalmente, verifíquese que  $x^{(p-1)/p_j} \not\equiv 1 \pmod p$ . Obsérvese que si  $x^{p-1} \equiv 1 \pmod p$ , entonces la menor  $i$  que satisface  $x^i \equiv 1 \pmod p$  debe dividir a  $p-1$ . Aún más, cualquier múltiplo de esta  $i$ , digamos  $ai$ , debe satisfacer también  $x^{ai} \equiv 1 \pmod p$ . Por consiguiente existe una  $i$  tal que  $x^i \equiv 1 \pmod p$ , entonces, para alguna  $p_j$ ,  $x^{(p-1)/p_j} \equiv 1 \pmod p$ .

Supóngase que el tiempo no determinístico que se lleva reconocer que  $p$  es primo está limitado por  $c \log^4 p$ . Entonces sólo necesitamos observar que

$$\sum_{j=1}^k c \log^4 p_j + \sum_{i=1}^k c_1 \log^3 p_i + c_1 \log^3 p \leq c \log^4 p$$

Para alguna constante  $c$  lo suficientemente grande.  $\square$

### 13.4 PROBLEMAS ESPACIOP-COMPLETOS

Mostraremos ahora que, varios problemas son completos para ESPACIOP con respecto al tiempo polinomial.

#### Fórmulas booleanas cuantificadas

Las fórmulas booleanas cuantificadas (QBF) se construyen a partir de variables, de los operadores  $\wedge$ ,  $\vee$  y  $\neg$ , paréntesis y los cuantificadores  $\exists$  (“existe”) y  $\forall$  (“para todo”). Cuando definimos a los QBFs de manera recursiva, encontramos útil definir de manera simultánea la aparición *libre* de las variables (apariciones a las que no se les aplica ningún cuantificador) la aparición en *unión* de éstas, y el campo de acción de un cuantificador (apariciones a las que el cuantificador se aplica).

1. Si  $x$  es una variable, entonces es un QBF. La aparición de  $x$  es libre.
2. Si  $E_1$  y  $E_2$  son QBFs también lo son  $\neg(E_1)$ ,  $(E_1) \wedge (E_2)$  y  $(E_1) \vee (E_2)$ . Una aparición de  $x$  es libre o acotada, dependiendo de si la aparición es libre o acotada en  $E_1$  o  $E_2$ . Se pueden omitir los paréntesis redundantes.
3. Si  $E$  es una QBF, entonces  $\exists x(E)$  y  $\forall x(E)$  son QBFs. Las coberturas de  $\exists x$  y  $\forall x$  son apariciones libres de  $x$  en  $E$ . (Nótese que también pueden haber apariciones acotadas de  $x$  en  $E$ ; éstas no forman parte de la cobertura.) Las apariciones libres de  $x$  en  $E$  están acotadas en  $\exists x(E)$  y  $\forall x(E)$ . Todas las demás apariciones de variables de  $E$  son libres o acotadas, dependiendo de si son libres o acotadas en  $E$ .

A una QBF que no tiene variables libres se le asigna un valor de **verdadero** o **falso**, lo cual denotamos con 1 o 0, respectivamente. El valor de una QBF con estas características se determina sustituyendo cada subexpresión de la forma  $\exists x(E)$  por  $E_0$ ,  $\vee E_1$ , y cada subexpresión de la forma  $\forall x(E)$  por  $E_0 \wedge E_1$ , en donde  $E_0$  y  $E_1$  son  $E$  en la que todas las apariciones de  $x$ , que se encuentren en la cobertura del cuantificador, son sustituidas por 0 y 1, respectivamente. El *problema QBF* consiste en determinar si una QBF que no tiene variables libres tiene valor **verdadero**.

**Ejemplo 13.4**  $\forall x[\forall x[\exists y(x \vee y)] \wedge \neg x]$  es una QBF. La cobertura de la  $\forall x$  interior es la primera de  $x$  (la primera  $x$  que aparece); la cobertura de la  $\forall x$  exterior es la segunda ocurrencia de  $x$ . Para probar la verdad de la fórmula anterior, debemos verificar que  $\forall x[\exists y(x \vee y)] \wedge \neg x$  es verdadero cuando las ocurrencias libres de  $x$  (es decir, la segunda aparición solamente) tienen asignado 0 y, también, 1. Se ve que la primera cláusula  $\forall x[\exists y(x \vee y)]$  es verdadera, como cuando esta  $x$  es 0 o 1, podemos escoger  $y = 1$  para hacer  $x \vee y$  verdadero. Sin embargo,  $\neg x$  no se hace verdadera cuando  $x = 1$ , de modo que la expresión completa es falsa.

Nótese que una expresión booleana  $E$ , con variables  $x_1, x_2, \dots, x_k$ , es satisfactoria si y sólo si la QBF  $\exists x_1 \exists x_2 \cdots \exists x_k (E_k)$  es verdadera. Por consiguiente el problema de la satisfactoriedad es un caso particular del problema que consiste en si una

QBF es verdadera, lo que inmediatamente nos dice que el problema QBF es *NP*-difícil. Sin embargo, no parece que la QBF se encuentre en *NP*.

#### Integridad ESPACIOP del problema QBF

**Lema 13.9** QBF se encuentra en ESPACIOP.

**Demostración** Un sencillo procedimiento recursivo, conocido como EVAL,<sup>†</sup> puede utilizarse para calcular el valor de un QBF que no tiene variables libres. De hecho, EVAL tratará un problema ligeramente más general, en el que las constantes booleanas 0 y 1 han sido sustituidas por algunas variables. Si la QBF consiste en una constante booleana, EVAL regresa dicha constante. Si la QBF consiste en un operador booleano aplicado a una subfórmula (o subfórmulas), entonces EVAL evalúa las subfórmulas de manera recursiva y después aplica el operador al resultado (o resultados). Si la QBF es de la forma  $\exists x(E)$  o  $\forall x(E)$ , entonces EVAL sustituye a todas las ocurrencias de  $x$  en  $E$ , que se encuentran dentro de la cobertura del operador, por 0, para obtener  $E_0$ , y después evalúa  $E_0$  de manera recursiva. En seguida EVAL sustituye las ocurrencias de  $x$  por 1, para obtener  $E_1$ , y evalúa  $E_1$  de manera recursiva. En el caso  $\exists x(E)$ , EVAL regresa el operador 0 de los dos resultados. En el caso  $\forall x(E)$ , EVAL regresa el operador AND.

Puesto que el número de operadores más los cuantificadores es cuando mucho de  $n$ , para una QBF de longitud  $n$ , la profundidad de la iteración es como máximo de  $n$ . Utilizando una cinta de Turing para la pila de los registros de activación (como se hizo en el Teorema 12.11), vemos que la cinta nunca necesita crecer más allá del cuadrado de la longitud de la QBF original. Por consiguiente el problema QBF se encuentra en ESPACIOP.  $\square$

**Teorema 13.10** El problema que consiste en decidir si una QBF es verdadera es ESPACIOP-completo.

**Demostración** Según el Lema 13.9, necesitamos demostrar solamente que el lenguaje  $L_{\text{qbf}}$  de las QBFs codificadas como verdaderas se encuentra en ESPACIOP-difícil. Esto es, debemos demostrar que cada lenguaje en ESPACIOP es reducible en tiempo polinomial a  $L_{\text{qbf}}$ .

Sea  $M$  una DTM con límite espacio polinomial de una sola cinta que acepta al lenguaje  $L$ . Entonces, para alguna constante  $c$  y algún polinomio  $p$ ,  $M$  hace más de  $c^{p(n)}$  movimientos sobre las entradas de longitud  $n$ . Podemos codificar las IDs de  $M$  como se hizo en el Teorema 13.1, utilizando las variables booleanas  $c_{ix}$ ,  $1 \leq i \leq p(n)$ , en donde  $X$  representa un símbolo de cinta o un símbolo compuesto que representa un símbolo y el estado de  $M$ . Puesto que  $M$  es determinística, no hay necesidad de codificar una alternativa de movimientos en el símbolo compuesto. Nuestro objetivo es construir, para cada  $j$ ,  $0 \leq j \leq p(n) \log c$ , una QBF  $F_j(I_1, I_2)$ , en donde

1.  $I_1$  e  $I_2$  son, cada una, conjuntos de variables, uno para cada  $i$ ,  $1 \leq i \leq p(n)$ , y cada símbolo de cinta o símbolo compuesto  $X$  es análogo a las  $c_{ix}$ s del Teorema 13.1. Digamos

<sup>†</sup>Por “evaluador”. N del T.

$$I_1 = \{c_{ix} \mid 1 \leq i \leq p(n) \text{ y } X \text{ es dicho símbolo}\},$$

y

$$I_2 = \{d_{iy} \mid 1 \leq i \leq p(n) \text{ y } Y \text{ es dicho símbolo}\}.$$

2.  $F_j(I_1, I_2)$  es verdadera si y sólo si  $I_1$  e  $I_2$  representan a las IDs  $\beta_1$  y  $\beta_2$  de  $M$ , esto es, para cada  $i$ , exactamente una  $c_{ix}$  y una  $d_{iy}$  son verdaderas, y  $\beta_1 \vdash^* \beta_2$  mediante una secuencia de cuando más  $2^i$  movimientos, en donde  $\beta_1 = X_1 X_2 \dots X_{p(n)}$ ,  $\beta_2 = Y_1 Y_2 \dots Y_{p(n)}$ , y  $X_i$  y  $Y_i$  son los símbolos tales que  $c_{ix}$  y  $d_{iy}$  son verdaderas.

Entonces, dada  $x$  de longitud  $n$ , podemos escribir a QBF como

$$Q_x = \exists I_0 \exists I_j [F_{p(n)\log c}(I_0, I_j) \wedge \text{INICIAL}(I_0) \wedge \text{FINAL}(I_j)],$$

en donde  $\exists I_0$  y  $\exists I_j$  representan una colección de variables existencialmente cuantificadas, una para cada símbolo  $X$  y entero  $i$ ,  $1 \leq i \leq p(n)$ , como se hizo más arriba. INICIAL( $I_0$ ) es una fórmula proposicional que nos dice que las variables del conjunto  $I_0$  representan a la ID inicial de  $M$ , con la entrada  $x$ , y FINAL( $I_j$ ) expresa el hecho de que  $I_j$  representa a la ID de aceptación de  $M$ . Entonces  $Q_x$  es verdadera si y sólo si  $x$  se encuentra en  $L(M)$ . INICIAL y FINAL pueden escribirse en un tiempo que es un polinomio en  $n$ , utilizando las técnicas de la demostración del Teorema 13.1.

Demostraremos ahora cómo construir, para cada  $j$ , la fórmula  $F_j(I_1, I_2)$ . La base  $j=0$  es fácil. Utilizando la técnica del Teorema 13.1, solamente tenemos que expresar como una fórmula booleana los hechos siguientes.

1.  $I_1$  e  $I_2$  representan IDs, digamos  $\beta_1$  y  $\beta_2$ ; esto es, exactamente una variable para cada posición de  $\beta_1$  y  $\beta_2$  es verdadera.
2. Se cumple cualquiera de las dos,  $\beta_1 = \beta_2$  o  $\beta_1 \vdash \beta_2$ .

Para el paso inductivo, nos vemos tentados a escribir

$$F_j(I_1, I_2) = (\exists I)[F_{j-1}(I_1, I) \wedge F_{j-1}(I, I_2)].$$

Sin embargo, si hacemos esto,  $F_j$  aproximadamente doble la longitud de  $F_{j-1}$ , y la longitud de  $F_{p(n)\log c}$  será de al menos  $c^{p(n)}$  y, por tanto, no puede escribirse en tiempo polinomial.

En lugar de esto utilizaremos un truco que nos permite hacer dos usos de una expresión como  $F_{j-1}$  en solamente una pequeña cantidad (polinomio en  $n$ ) más de espacio que el que se requiere para un solo uso. El truco consiste en expresar que existen  $J$  y  $K$  tales que si  $J = I_1$  y  $K = I_2$ , o  $J = I_1$  y  $K = I_2$ , entonces  $F_{j-1}(J, K)$  debe ser verdadera. La QBF para esto es

$$\begin{aligned} F_j(I_1, I_2) &= \exists J [\exists K [(\neg(J = I_1 \wedge K = I_2) \\ &\quad \wedge \neg(J = I_1 \wedge K = I_2)) \vee F_{j-1}(J, K)]]. \end{aligned} \quad (13.3)$$

Utilizamos expresiones como  $J = I_1$  para significar que, para cada par de variables correspondientes de los conjuntos  $J$  e  $I_1$  (aqueellas que representan la misma posición y símbolo), ambas son falsas o verdaderas. La Ecuación (13.3) establece que siempre que el par  $(J, K)$  sea  $(I_1, I_1)$  o  $(I_1, I_2)$ , entonces  $F_{j-1}(J, K)$  debe ser verdadero. Esto nos

permite afirmar que ambas  $F_{j-1}(I_1, I)$  y  $F_{j-1}(I, I_2)$ , son verdaderas utilizando solamente una copia de  $F_{j-1}$ . De manera intuitiva se utiliza  $F_{j-1}$  como una “subrutina” que se “llama” dos veces.

El número de símbolos de  $F_j$ , contando cualquier variable como uno, es  $O(p(n))$  más el número de símbolos de  $F_{j-1}$  como (13.3) introduce  $O(p(n))$  variables (en los conjuntos  $I, J$  y  $K$ ), el número de variables de  $F_j$  es  $O(jp(n))$ . Por consiguiente podemos codificar una variable con  $O(\log j + \log p(n))$  bits. Se concluye, por inducción en  $j$ , que  $F_j$  puede escribirse en un tiempo  $O(jp(n)(\log j + \log p(n)))$ . Si hacemos  $j=p(n)$  log  $c$  y observamos que, para cualquier polinomio  $p(n)$ ,  $\log p(n) = O(\log n)$ , vemos que  $Q_x$  puede escribirse en un tiempo  $O(p^2(n) \log n)$ . Por consiguiente existe una reducción en tiempo polinomial de  $L(M)$  a  $L_{\text{qbr}}$ . Ya que  $M$  es una TM cualquiera con límite en espacio polinomial, hemos demostrado que  $L_{\text{qbr}}$  es ESPACIOP-completo.  $\square$

### Reconocimiento sensible al contexto

Otro problema que es ESPACIOP-completo y que vale la pena tener en cuenta es: Dadas una CSG  $G$  y una cadena  $w$ , ¿se encuentra  $w$  en  $L(G)$ ? Este resultado es sorprendente, ya que los CSLs ocupan el “fondo” del ESPACIOP( $n^2$ ). Sin embargo, la técnica de “rellenado” que se utiliza en el lema de traslación (Lema 12.2) hace posible una demostración.

Para empezar, tómese un código binario sencillo para las gramáticas, como ya se hizo para las máquinas de Turing. Sea  $L_{ci}$  el lenguaje que consiste en todas las cadenas  $x\#w$ , en donde  $x$  es el código para una CSG  $G_x$  y  $w$  es la cadena codificada a partir del alfabeto de entrada de  $G_x$ . Supóngase que, para una gramática dada, todos los símbolos de gramática son codificados mediante cadenas de la misma longitud. Es fácil diseñar un LBA que, dada la entrada  $x\#w$ , adivine (o suponga) una derivación de  $G_x$  tal que ninguna forma oracional exceda la longitud de la cadena codificada por  $w$ . La forma oracional codificada puede almacenarse en un segundo registro bajo las celdas que contienen a  $w$ . Los movimientos se determinan consultando la parte  $x$  de la entrada (para ver cómo se puede hacer esto, resulta de ayuda suponer la existencia de una segunda cinta). Vemos que  $L_{ci}$  se encuentra en ESPACION( $n$ ) y, por tanto, en ESPACIOP.

**Teorema 13.11**  $L_{ci}$ , el problema de reconocimiento de CSL, es ESPACIOP-completo.

**Demostración** Ya se sabe que  $L_{ci}$  se encuentra en ESPACIOP. Sea  $L$  un miembro cualquiera de ESPACIOP digamos que  $L$  es aceptado por  $M$ , una DTM de complejidad espacial  $p(n)$ . Definase  $L'$  como  $\{y\$^{p(1)y}\mid y \text{ se encuentra en } L\}$ , en donde  $\$\text{ es un nuevo símbolo}$ . Resulta fácil verificar que  $L'$  se encuentra en ESPACIOD( $n$ ) y que, por consiguiente, es un CSL. Sea  $G$  una CSG para  $L'$ , y sea  $x$  el código binario para  $G$ . Entonces la transformación en tiempo polinomial que lleva  $y$  a  $x\#w$ , en donde  $w$  es el código para  $y\$^{p(1)y}$ , es una reducción de  $L$  a  $L_{ci}$ , demostrándose así que  $L_{ci}$  es ESPACIO-completo.  $\square$

### 13.5 PROBLEMAS COMPLETOS PARA $\mathcal{P}$ Y ESPACIO ( $\log n$ )

Es obvio que  $\text{ESPACIOD}(\log n) \subseteq \mathcal{P}$ , según el Teorema 12.10. ¿Podría ser que  $\mathcal{P} = \text{ESPACIOD}(\log n)$ , o tal vez  $\mathcal{P} \subseteq \text{ESPACIOD}(\log^k n)$ , para alguna  $k$ ? De manera similar, es obvio que  $\text{ESPACIOD}(\log n) \subseteq \text{ESPACION}(\log n)$ . ¿Podría ser que estas dos clases sean iguales? Si esto es cierto, entonces, mediante una traslación parecida a la que se hizo en el Lema 12.2, se concluye que  $\text{ESPACION}(n) \subseteq \text{ESPACIOD}(n)$ , esto es, los CSLs determinísticos y los no determinísticos son lo mismo.

Mostraremos un lenguaje  $L_1$  de  $\mathcal{P}$ , tal que cada lenguaje de  $\mathcal{P}$  es reducible en espacio logarítmico a  $L_1$ . De estar este lenguaje en  $\text{ESPACIOD}(\log^k n)$ , para alguna  $k$ , entonces  $\mathcal{P}$  está contenida en  $\text{ESPACIOD}(\log^k n)$ . De manera parecida, mostramos un lenguaje  $L_2$ , que se encuentra en  $\text{ESPACION}(\log n)$  tal que cada lenguaje de  $\text{ESPACION}(\log n)$  es reducible en espacio logarítmico a  $L_2$ . De estar  $L_2$  en  $\text{ESPACION}(\log n)$ , entonces  $\text{ESPACIOD}(\log n)$  sería igual a  $\text{ESPACION}(\log n)$ . No existe, por supuesto, una forma conocida para reconocer a  $L_1$  en espacio  $\log^k n$  ni una forma conocida para reconocer determinísticamente a  $L_2$  en espacio  $\log n$ .

Los lenguajes completos para  $\text{ESPACION}(\log n)$  o para  $\mathcal{P}$  no son necesariamente difíciles de reconocer y, de hecho, los lenguajes  $L_1$  y  $L_2$  son relativamente fáciles de reconocer. Los resultados de esta sección sirven meramente para reforzar la idea de que muchas clases de complejidad tienen problemas completos. No sugieren la no tratabilidad en la misma forma en que los resultados de la integridad  $NP$  o la integridad  $\text{ESPACIOP}$  lo hacen.

#### Vacio libre de contexto

Definase  $L_{\text{cfe}}$  como el lenguaje de las CFGs codificadas cuyos lenguajes están vacíos.  $L_{\text{cfe}}$  es el lenguaje  $L_1$  al que nos referimos más arriba. Demostraremos que  $\mathcal{P}$  es reducible en espacio logarítmico a  $L_{\text{cfe}}$ .

**Teorema 13.12**  $L_{\text{cfe}}$ , el problema del vacío para las CFGs, es completo para  $\mathcal{P}$  con respecto a las reducciones en espacio logarítmico.

**Demostración** Reduciremos un lenguaje  $L$  cualquiera de  $\mathcal{P}$  a  $L_{\text{cfe}}$  utilizando solamente un espacio  $\log n$ . Específicamente diseñaremos un transductor espacio logarítmico  $M_1$ . Dada la entrada  $x$  de longitud  $n$ ,  $M_1$  escribe una gramática  $G_x$  tal que  $L(G_x) = \emptyset$  si y sólo si  $x$  se encuentra en  $L$ . Sea  $\bar{M}$  una TM con límite temporal  $p(n)$  que acepte al complemento de  $L$ . Como  $\mathcal{P}$  es efectivamente cerrado con respecto al complemento, podemos encontrar a  $\bar{M}$ . Intuitivamente, una derivación de  $G_x$  corresponde a un cálculo válido de  $\bar{M}$  sobre  $x$ . Los símbolos no terminales de  $G_x$  son todos de la forma  $A_{xu}$ , en donde

1.  $X$  es un símbolo de cinta de  $\bar{M}$ , un par  $[qY]$ , en donde  $q$  es un estado y  $Y$  un símbolo de cinta, o el símbolo señalador  $\#$  que se utiliza para denotar los extremos finales de las IDs;
2.  $0 \leq i \leq p(n) + 1$ ;
3.  $0 \leq t \leq p(n)$ .

Se tiene la intención de que  $A_{xu} \xrightarrow{*} w$ , para una cadena  $w$ , si y sólo si  $X$  es el  $i$ -ésimo

símbolo de la ID de  $\bar{M}$  en el tiempo  $t$ . El símbolo  $S$  es también una no terminal de  $G_x$ ; es el símbolo inicial.

Las producciones de  $G_x$  son:

1.  $S \xrightarrow{} A_{i,q,Y,i,t}$ , para toda  $i, t$  y  $Y$ , en donde  $q_f$  es un estado final.
  2. Se  $f(X, Y, Z)$  el símbolo que se encuentra en la posición  $i$  de la  $i$ -ésima ID cuando  $X, Y, Z$  ocupan la posición  $i - 1, i$  e  $i + 1$  de la  $(i - 1)$ -ésima ID. Como  $\bar{M}$  es determinística,  $f(X, Y, Z)$  es un símbolo único y es independiente de  $i$  y  $t$ . Por consiguiente, para cada  $i$  y  $t$ ,  $1 \leq i, t, \leq p(n)$ , y para cada tripleta  $X, Y, Z$  como  $W = f(X, Y, Z)$ , tenemos la producción.
- $$A_{Wit} \rightarrow A_{X,i-1,t-1} A_{Y,i,t-1} A_{Z,i+1,t-1}$$
3.  $A_{\#,0} \rightarrow \epsilon$  y  $A_{\#,p(n)+1,t} \rightarrow \epsilon$ , para toda  $t$ .
  - 4).  $A_{x_{i0}} \rightarrow \epsilon$ , para  $1 \leq i \leq p(n)$ , si y sólo si el  $i$ -ésimo símbolo de la ID inicial, con entrada  $x$ , es  $X$ .

Una sencilla inducción sobre  $t$  muestra que, para  $1 \leq i \leq p(n)$ ,  $A_{w_{ii}} \xrightarrow{*} \epsilon$  si y sólo si  $W$  es el  $i$ -ésimo símbolo de la ID en el tiempo  $t$ . Por supuesto que ninguna cadena terminal, excepto  $\epsilon$ , nunca es derivada a partir de cualquier cadena no terminal.

**Base** La base,  $t = 0$ , se concluye de manera inmediata a partir de la regla (4).

**Inducción** Si  $A_{w_{ii}} \xrightarrow{*} \epsilon$ , entonces, según la regla (2), debe cumplirse que, para alguna  $X, Y$  y  $Z$ ,  $W \in f(X, Y, Z)$  y cada una de las  $A_{X,i-1,t-1}, A_{Y,i,t-1}$  y  $A_{Z,i+1,t-1}$  deriva a  $\epsilon$ . Según la hipótesis inductiva, los símbolos de la ID en el tiempo  $t - 1$  y las posiciones  $i - 1$ ,  $i$  e  $i + 1$  son  $X, Y$  y  $Z$ , de modo que  $W$  es el símbolo en la posición  $i$  y el tiempo  $t$ , según la definición de  $f$ .

De manera inversa, si  $W$  es el símbolo que se encuentra en la posición  $i$  y el tiempo  $t \geq 1$ , entonces  $W = f(X, Y, Z)$ , en donde  $X, Y$  y  $Z$  son los símbolos en el tiempo  $t - 1$  y las posiciones  $i - 1, i$  e  $i + 1$ . Por la hipótesis inductiva, o por la regla (3), si  $i = 0$  o  $i = p(n) + 1$ , entonces

$$A_{X,i-1,t-1} A_{Y,i,t-1} A_{Z,i+1,t-1} \xrightarrow{*} \epsilon$$

Por consiguiente, según la regla (2),  $A_{w_{ii}} \xrightarrow{*} \epsilon$ .

Entonces, según la regla (1),  $S \xrightarrow{*} \epsilon$  si y sólo si  $\bar{M}$  acepta a  $x$ .

Finalmente, necesitamos demostrar que las producciones de  $G_x$  pueden ser producidas por  $M_1$ , con entrada  $x$  de longitud  $n$ . Antes que nada, recuérdese que  $\log_2 p(n) \leq c \log_2 n$ , para alguna constante  $c$ , ya que  $p(n)$  es un polinomio. Por tanto  $M_1$  puede contar desde  $i=0$  hasta  $p(n)$  en un almacenamiento de tachado  $\log n$ . De manera similar  $M_1$  puede contar de  $t=0$  hasta  $p(n)$  en espacio  $\log n$ . Las producciones de  $G_x$  se generan de manera sencilla mediante un ciclo doble sobre  $i$  y  $t$ .

Ahora,  $G_x$  se encuentra en  $L_{\text{cfe}}$  si y sólo si  $\bar{M}$  no acepta a  $x$  y, en consecuencia, si y sólo si  $x$  se encuentra en  $L$ . Por tanto  $L_{\text{cfe}}$  es completo para  $\mathcal{P}$  con respecto a las reducciones en espacio logarítmico.  $\square$

#### El problema de la alcanzabilidad

Daremos, ahora, un problema que es completo para  $\text{ESPACION}(\log n)$  con respecto a las reducciones en espacio logarítmico. El problema de la alcanzabilidad en grafos

consiste en determinar, dado un grafo dirigido con vértices  $\{1, 2, \dots, n\}$ , si existe una trayectoria de 1 a  $n$ .

**Teorema 13.13** El problema de la alcanzabilidad en grafos es completo en espacio logarítmico para  $\text{ESPACION}(\log n)$  con respecto a las reducciones en espacio logarítmico.

**Demostración** La formalización del problema como un lenguaje se deja al lector. En primer lugar demostramos que el problema de la alcanzabilidad en grafos se encuentra en  $\text{ESPACION}(\log n)$ . Una TM no determinística,  $M$ , puede adivinar la trayectoria vértice por vértice.  $M$  no almacena la trayectoria, sino que en lugar de hacerlo la verifica, almacenando solamente los vértices que se alcanzan.

Ahora, dado un lenguaje  $L$  en  $\text{ESPACION}(\log n)$ , lo reducimos en espacio  $\log n$  de manera determinística al lenguaje de los digrafos codificados, para los cuales existe una trayectoria que va del primer vértice al último. Sea  $M$  una TM fuera de línea no determinística, con límite espacial  $\log n$ , que acepta a  $L$ . Una ID de  $M$  se puede representar mediante el contenido de la cinta de almacenamiento, lo que toma un espacio  $\log n$  en la representación, la posición de la cabeza de la cinta de almacenamiento y el estado, que puede ser codificado con el contenido del almacenamiento mediante un símbolo compuesto  $[qX]$ , y la posición de la cabeza de entrada, que requiere  $\log n$  bits.

Construimos un transductor espacio logarítmico  $M_1$  que toma la entrada  $x$  y produce un digrafo  $G_x$  que tiene una trayectoria que va del primer vértice hasta el último si y sólo si  $M$  acepta a  $x$ . Los vértices de  $G_x$  son las IDs de  $M$  con entrada  $x$  (pero con la posición de la cabeza de entrada, más que con la  $x$  misma) más un vértice especial, el último, que representa la aceptación. El primer vértice es la ID inicial con entrada  $x$ .  $M_1$  utiliza su almacenamiento  $\log n$  para formar un ciclo a través de todas las IDs de  $M$ . Para cada ID  $I$ ,  $M_1$  coloca su cabeza de entrada en la posición de entrada correcta, de modo que pueda ver el símbolo de entrada barrido por  $M$ .  $M_1$  entonces genera los arcos  $I \rightarrow J$ , para todas las  $J$ s, tales que  $I$  se puede convertir en  $J$  mediante un movimiento de  $M$ . Puesto que  $M_1$  tiene a  $I$  disponible en su cinta de almacenamiento, y  $J$  puede construirse fácilmente a partir de  $I$ , esta generación requiere no más de un espacio  $\log n$ . Si  $I$  es una ID de aceptación,  $M_1$  genera el arco  $I \rightarrow v$ , en el que  $v$  es el vértice especial.

Resulta sencillo verificar que existe una trayectoria en  $G_x$  que va de la ID inicial a  $v$  si y sólo si  $M$  acepta a  $x$ . Por consiguiente cada lenguaje que esté en  $\text{ESPACION}(\log n)$  es reducible en espacio logarítmico al problema de la alcanzabilidad. Concluimos que el problema de la alcanzabilidad es completo para  $\text{ESPACION}(\log n)$  con respecto a las reducciones en espacio logarítmico.  $\square$

### 13.6 ALGUNOS PROBLEMAS DEMOSTRABLEMENTE NO TRATABLES

Hasta este momento hemos insinuado fuertemente que ciertos problemas requieren un tiempo exponencial para demostrar que son *NP*-completos o *ESPACION*-completos. Demostraremos ahora que dos problemas requieren en realidad tiempo exponencial. En un caso, reducimos a nuestro problema un lenguaje que, según el teorema de la jerarquía espacial, se sabe que requiere espacio exponencial y, por tanto, tiempo exponencial. En el segundo caso, mostramos como reducir a nuestro

problema todos los lenguajes en un tiempo exponencial no determinístico y, entonces, argumentamos, mediante un teorema de jerarquía de tiempo no determinístico [Cook, 1973a], que entre estos problemas debe existir uno que realmente necesite un espacio de, digamos,  $2^n$ .

Consideraremos ahora un problema concerniente a las expresiones regulares que es un tanto artificial de manera que (a) al menos se requiere un espacio  $2^{\text{cu} \log n}$  para resolverlo y (b) este requisito puede probarse fácilmente. Después de esto, consideraremos un problema de lógica que no es artificial en el sentido de que fue considerado mucho antes de que se analizara su complejidad, y cuya demostración de exponencialidad está lejos de ser sencilla.

#### Expresiones regulares con exponentiación

Consideremos expresiones regulares sobre un alfabeto que, por conveniencia, se supone no contiene a los símbolos  $\uparrow$ , 0 o 1. Sea  $r^{\uparrow i}$  la expresión regular  $rr \cdots r$  ( $i$  veces), en donde  $i$  se escribe en binario. La expresión  $r$  puede incluir al operador  $\uparrow$  (operador exponentiación). Por ejemplo,  $(a \uparrow 11 + b \uparrow 11)^{\uparrow 10}$  significa

$$\{aaaaaa, aaabbb, bbbaaa, bbbbb\}.$$

Suponemos que  $\uparrow$  tiene mayor precedencia que los otros operadores. El problema del cual demostraremos que esencialmente requiere *espacio exponencial*, esto es, espacio  $2^{p(n)}$  para algún polinomio  $p(n)$ , consiste en si una expresión regular con exponentiación representa a todas las cadenas sobre su alfabeto (recuérdese que  $\uparrow$ , 0 y 1 se utilizan como operadores y no forman parte del alfabeto). Primero daremos un algoritmo en espacio exponencial para el problema.

**Teorema 13.14** El problema que consiste en si una expresión regular con exponentiación representa a todas las cadenas sobre su alfabeto, puede resolverse en espacio exponencial.

**Demostración** Dada una expresión regular de longitud  $n$ , expandiremos los operadores  $\uparrow$ s para obtener una expresión regular ordinaria y demostraremos que tiene una longitud de cuando mucho  $n2^n$ . Entonces convertiremos esta expresión a un NFA de cuando mucho  $n2^{n+2}$  estados y probaremos si ese NFA acepta a  $\Sigma^*$ . (Nótese que este último paso debe hacerse sin conversión a un DFA, ya que el DFA debe tener  $2^{n2^{n+2}}$  estados.) Para eliminar las  $\uparrow$ s trabajamos de adentro hacia afuera. Demostraremos por inducción en  $j$  que una expresión con  $\uparrow$ s, longitud  $m$  y  $j$  0s y 1s, tiene una expresión regular originaria equivalente cuya longitud es de cuando más  $m2^j$ .

**Base**  $j = 0$ . El resultado es inmediato.

**Inducción** Bárrase la expresión  $r$  de longitud  $m$  de izquierda a derecha hasta que se encuentre el primer  $\uparrow$ . Después bárrase de regreso hasta que se encuentre el argumento izquierdo  $r_1$  de dicho  $\uparrow$ . Suponemos que  $\uparrow$  tiene la mayor precedencia, de modo que su argumento debe ser un solo símbolo o estar rodeado de paréntesis; de aquí que esta extracción es fácil. Hagamos que la expresión sea  $r = r_2 r_1 \uparrow ir_3$ . Sustitúyase  $r$  por  $r' = r_2 r_1 r_1 \cdots r_1 r_3$ , en donde  $r_1$  se escribe  $i$  veces. Según la hipótesis inductiva,  $r'$  tiene una

expresión regular ordinaria equivalente cuya longitud es de cuando mucho  $(m + (i - 1)|r_1|)2^{j-\log_2 i} = 2^{j/i}$ , y como  $|r_1| \leq m$ , vemos que

$$(m + (i - 1)|r_1|)2^{j-\log_2 i} = \frac{m + (i - 1)|r_1|}{i} \times 2^j \leq m2^j.$$

Si  $r$  es de longitud  $n$ , entonces seguramente  $m = n$  y  $j \leq n$ , de modo que la expresión regular ordinaria equivalente tiene longitud de cuando mucho  $n2^n$ .

Ahora, utilizando el algoritmo del Teorema 2.3, podemos producir un NFA equivalente de cuando mucho  $4n2^n = n2^{n+2}$  estados. Adivínese de manera no determinística, símbolo por símbolo, una entrada  $a_1a_2\cdots$  que el NFA no acepte. Utilizando  $n2^{n+2}$  celdas podemos, después de cada adivinación, calcular el conjunto de estados accesados después que el NFA lee la secuencia de símbolos adivinados hasta entonces. No hay necesidad de escribir la entrada, ya que podemos calcular el conjunto de estados accesados a partir de este conjunto sobre cualquier símbolo de entrada. Si adivinamos una secuencia de entrada sobre la cual no se accesa ningún estado de aceptación del NFA, entonces aceptamos; la expresión original no representa a  $\Sigma^*$ . Según el teorema de Savitch podemos llevar a cabo este proceso de manera determinística utilizando un espacio  $n^2 2^n$ . Resulta sencillo diseñar una codificación del NFA que pueda ser almacenada en  $O(n^2 2^n)$  celdas, ya que cerca de  $n$  bits son suficientes para codificar un estado, y el alfabeto de entrada no es mayor que  $n$ . Como  $n^2 2^n > n^3 2^n$ , se concluye que  $n^2 2^n$  constituye un límite superior en el espacio requerido.  $\square$

Daremos ahora un límite inferior de  $2^{cn\log n}$ , para alguna constante  $c$ , para el espacio requerido en el problema anterior. Obsérvese que probar que se necesita una determinada cantidad de espacio también demuestra que se necesita la misma cantidad de tiempo (aunque el inverso no es verdadero).

**Teorema 13.15** Existe una constante  $c > 0$  tal que cada TM que acepta al lenguaje  $L_{\text{rex}}$  de las expresiones regulares con exponentiación, que denota a  $\Sigma^*$ , toma un espacio mayor a  $2^{cn\log n}$  (y por tanto un tiempo  $2^{cn\log n}$ ) infinitamente frecuente.

*Demotración* Considérese una TM,  $M$ , determinística de una sola cinta y con límite espacial  $2^n$ . Para cada entrada  $x$  de longitud  $n$ , construimos una expresión regular con exponentiación  $E_x$  que representa a  $\Sigma^*$ , en donde  $\Sigma$  es el alfabeto de  $E_x$ , si y sólo si  $M$  no acepta a  $x$ . Llevamos a cabo lo anterior haciendo que  $E_x$  represente a todos los cálculos no válidos de  $M$  sobre  $x$ . Hagamos que  $\Sigma$  consista en todos los símbolos de cinta de  $M$ , los símbolos compuestos  $[qX]$ , en donde  $q$  es un estado y  $X$  un símbolo de cinta, y el símbolo señalador  $\#$ . Supóngase que  $\uparrow$ ,  $0$  y  $1$  no son ninguno de tales símbolos.

Una cadena  $y$  de  $\Sigma^*$  no es un cálculo de aceptación de  $M$  sobre  $x$  si y sólo si una o más de las siguientes proposiciones es verdadera.

1. La ID inicial está equivocada.
2. No existe un estado de aceptación.
3. Una ID no se concluye de la anterior mediante un movimiento de  $M$ .

En lo que sigue, utilizamos conjuntos de símbolos para representar a la expresión regular que es la suma de dichos símbolos. Por consiguiente, si  $\Sigma = \{a_1, a_2, \dots, a_n\}$ , entonces utilizamos a  $\Sigma$  como una abreviatura de la expresión regular  $a_1 + a_2 + \dots + a_n$ .

De manera similar también utilizamos  $\Sigma - a$  para la expresión regular que es la suma de todos los símbolos de  $\Sigma$ , excepto  $a$ .

Una expresión regular que denota a todas las cadenas que no comienzan con la ID inicial está dada por

$$\begin{aligned} \text{COMIENZO} = & \epsilon + (\Sigma - \#)\Sigma^* + A_1 + A_2 + \dots + A_n \\ & + \Sigma \uparrow (n + 1)(\Sigma + \epsilon) \uparrow (2^n - n - 1)(\Sigma - B)\Sigma^* \\ & + \Sigma \uparrow (2^n + 1)(\Sigma - \#)\Sigma^*, \end{aligned}$$

En donde

$$A_1 = \Sigma \uparrow 1(\Sigma - [q_0 a_1])\Sigma^*,$$

y, para  $2 \leq i \leq n$ ,

$$A_i = \Sigma \uparrow i(\Sigma - a_i)\Sigma^*.$$

El término siguiente a último representa a  $\Sigma^{i+1}$  seguida de hasta  $2^n - n - 1$  símbolos seguidos por cualquier otro símbolo, excepto un espacio en blanco, y representa las cadenas tales que alguna posición entre  $n + 1$  y  $2^n$  de la primera ID no contiene un espacio en blanco. Como  $n$  y  $2^n - n - 1$  están escritos en binario, la longitud de este término es proporcional a  $n$ . El último término representa a las cadenas en las cuales el  $(2^n + 1)$ -ésimo símbolo no es  $\#$ . También tiene una longitud proporcional a  $n$ . Los términos restantes son de longitud proporcional a  $\log n$ , y existen  $n + 3$  de tales términos. Por consiguiente la longitud de la expresión es proporcional a  $n \log n$ . Curiosamente, la longitud de la expresión que representa a las IDs iniciales falsas domina a la longitud de los otros términos de  $E_x$ .

Una expresión regular que refuerza la condición de que no existe un estado de aceptación está dada por

$$\text{TERMINA} = (\Sigma - \{[qX] \mid q \text{ es un estado final}\})^*.$$

Esta expresión es de longitud constante, dependiendo solamente de  $M$ .

Finalmente, sea  $f(X, Y, Z)$  el símbolo  $Z$  tal que si  $W, X$  y  $Y$  se encuentran en las posiciones  $i - 1, i$  e  $i + 1$ , respectivamente, de una ID, entonces  $Z$  se encontrará en la posición  $i$  de la siguiente ID. Entonces sea

$$\text{MOVIMIENTO} = \sum_{(W, X, Y)} \Sigma^* W X Y \Sigma \uparrow (2^n - 1)(\Sigma - f(W, X, Y))\Sigma^*$$

Esto es, MOVIMIENTO es la suma, sobre el número finito de triplets  $(W, X, Y)$  de símbolos de  $\Sigma$ , de estas cadenas,  $W, X$  y  $Y$  ocupan posiciones consecutivas de una ID cuyo símbolo siguiente que está equivocado se encuentra a  $2^n$  posiciones a la derecha. Como la longitud de cada término es lineal en  $n$ , la longitud de MOVIMIENTO es lineal en  $n$ .

La expresión descada es  $E_x = \text{COMIENZO} + \text{TERMINAL} + \text{MOVIMIENTO}$ . Si  $M$  acepta a  $x$ , entonces el cálculo de aceptación no se encuentra en  $E_x$ . Si alguna cadena  $y$  no está en  $E_x$ , entonces debe comenzar con  $\#[q_0 a_1]a_2 \dots a_n B^{2^n - n} \#$ , cada ID debe seguir a la anterior mediante un movimiento de  $M$  y la aceptación debe darse a lo largo del proceso. Por consiguiente  $M$  acepta a  $x$ . En consecuencia  $E_x = \Sigma^*$  si y sólo si  $M$  no acepta a  $x$ .

Ahora, sea  $M$  una máquina de Turing que acepta al lenguaje  $L$ , que puede ser aceptado en espacio  $2^n$  pero no es espacio  $2^{n/2}$ . El teorema de la jerarquía para el espacio nos asegura la existencia de una  $M$  con tales características. Supóngase que existe una TM con límite espacial  $S(n)$  que acepta al conjunto  $L_{\text{reg}}$  de expresiones regulares con exponentiación que representa a  $\Sigma^*$ , adecuadamente codificado, de modo que  $L_{\text{reg}}$  tiene un alfabeto finito. Entonces podríamos reconocer a  $L$  de la manera siguiente.

1. A partir de  $x$ , de longitud  $n$ , constrúyase  $E_x$ , cuya longitud es proporcional a  $n \log n$ . Podemos construir  $E_x$  en un espacio proporcional a  $n \log n$  de una manera que es obvia.
2. Codifíquese  $E_x$  dentro del alfabeto  $L_{\text{reg}}$ . Como  $M$  tiene un número finito de símbolos, la longitud de la  $E_x$  codificada es  $cn \log n$ , para alguna constante  $c$ .
3. En un espacio  $S(cn \log n)$  determinése si  $E_x$  se encuentra en  $L_{\text{reg}}$ . Si sucede esto, rechácese  $x$ ; si no, acéptese  $x$ .

La cantidad total de espacio es el máximo de  $n \log n$  y  $S(cn \log n)$ . Como ninguna TM utiliza menos de un espacio  $2^{n/2}$ , y el conjunto  $L$  existe, se debe dar que

$$n \log n + S(cn \log n) > 2^{n/2} \quad \text{i.o.} \quad (13.4)$$

de otra manera  $L$  podría ser aceptado en espacio  $2^{n/2}$ , según el Lema 12.3. Existe una constante  $d > 0$  tal que si  $S(m)$  fuera menor a  $2^{dm \log m}$ , para toda  $m$ , excepto un conjunto finito de ellas, entonces (13.4) sería falsa. Se concluye que  $S(m) \geq 2^{dm \log m}$ , para alguna constante  $d$  y un número finito de  $m$ s.  $\square$

**Corolario**  $L_{\text{reg}}$  es completo para un espacio exponencial con respecto a la reducción en tiempo polinomial.

**Demostración** En la demostración del Teorema 13.15, dimos una reducción en tiempo polinomial a  $L_{\text{reg}}$  que funciona para cada lenguaje  $L$  en ESPACIOD( $2^n$ ). Podríamos haber generalizado este resultado para reducir cualquier lenguaje en ESPACIOD( $2^{p(n)}$ ), para el polinomio  $p$ , a  $L_{\text{reg}}$ .  $\square$

Observaremos que el límite  $n \log n$  sobre la longitud de  $E_x$  es un factor crítico para el Teorema 13.15, aunque, para su corolario, podríamos haber permitido la longitud de cualquier polinomio en  $|x|$ . Si, por ejemplo, pudiéramos demostrar solamente que  $|E_x| \leq |x|^2$ , entonces nuestro límite inferior en el espacio requerido por  $L_{\text{reg}}$ , sería  $2^{d\sqrt{n}}$ .

### Complejidad de las teorías de primer orden

Consideraremos ahora un problema que requiere, al menos, un tiempo  $2^{cn}$ , de manera no determinística, y que se sabe que es soluble en espacio exponencial y tiempo doble exponencial. Como también se puede demostrar que el problema es difícil no determinísticamente en tiempo exponencial con respecto a las reducciones en tiempo polinomial, se demuestra que un mejor límite inferior que toma en cuenta la cantidad de tiempo no determinístico mejoraría el teorema 12.10, lo cual parece lo más improbable.

Un lenguaje de primer orden consiste en un dominio (por ejemplo, los enteros no

negativos), un conjunto de operaciones (por ejemplo,  $+, *, =, <$ ), un conjunto de predicados (por ejemplo,  $=, <$ ), un conjunto de constantes escogidas del dominio y un conjunto de axiomas que definen el significado de los operadores y predicados. Para cada teoría podemos definir el lenguaje de las expresiones verdaderas sobre las constantes, operadores, predicados, variables, los conectores lógicos  $\wedge, \vee$  y  $\neg$ , y los cuantificadores  $\exists$  y  $\forall$ .

**Ejemplo 13.5** ( $\mathbb{N}, +, *, =, <, 0, 1$ ), en donde  $\mathbb{N}$  es el conjunto de los enteros no negativos, se conoce como teoría de números. El famoso teorema de la no integridad de Gödel establece que el lenguaje de las proposiciones verdaderas en la teoría de números es irresoluble. Mientras que el resultado de Gödel precedió a las máquinas de Turing, no resulta difícil demostrar este resultado. Si una TM  $M$  acepta cuando se le inicia sobre una cinta en blanco, lo hace mediante un cálculo en el cual ninguna ID es mayor que alguna constante  $m$ . Podemos tratar un entero  $i$ , en binario, como un cálculo de  $M$  con IDs de longitud  $m$ .

La proposición de que  $M$  acepta a  $\epsilon$ , que es irresoluble, puede expresarse como  $\exists i \exists m(E_m(i))$ , en donde  $E_m$  es un predicado que es verdadero si y sólo si  $i$  es el código binario de un cálculo que lleva a la aceptación de  $\epsilon$ , en el cual ninguna ID es mayor que  $m$ . Algunos de los detalles se encuentran en el Ejercicio 13.37. Por consiguiente, la teoría de números es una teoría irresoluble.

Existen varias teorías resolubles que son conocidas. Por ejemplo ( $[R, +, =, <, 0, 1]$ , la teoría de los reales con sumas, es resoluble y demostraremos que ésta requiere, de manera inherente, un tiempo exponencial. Si los reales se sustituyen por los racionales, obtenemos las mismas proposiciones verdaderas, ya que, sin la multiplicación, es imposible encontrar una proposición como  $\exists x (x * x = 2)$  que sea verdadera para los reales pero no para los racionales. La teoría de los enteros sin la adición ( $\mathbb{Z}, +, =, <, 0, 1$ ), conocida como aritmética de Presburger, es resoluble y, se sabe, requiere un tiempo determinístico doble exponencial. Esto es,  $2^{2n}$  es un límite inferior sobre la complejidad de tiempo no determinístico de la aritmética de Presburger.

**Ejemplo 13.6** Antes de seguir adelante, consideraremos un cierto número de ejemplos de la teoría de los reales con la adición.  $\forall x \exists y (y = x + 1)$  es verdadera: nos dice que  $x + 1$  es un número real siempre que  $x$  lo sea.

$$\forall x \forall y [x = y \vee \exists z (x < z \wedge z < y) \vee \exists z (y < z \wedge z < x)]$$

es también verdadera: establece que, entre dos reales diferentes, podemos encontrar un tercer número real; es decir, el conjunto de los números reales es denso. La proposición

$$\exists y \forall x (x < y \vee x = y)$$

es falsa, ya que, para cada número real  $y$ , existe un real mayor. Nótese que no hemos dicho cómo decidir si una proposición es verdadera; la decisión depende del conocimiento que se tenga de los números reales, con los cuales suponemos que el lector es familiar.

### Un procedimiento de decisión para los reales con adición

Comenzamos nuestro estudio de los reales con adición dando un procedimiento de decisión que requiere un espacio exponencial y tiempo doble exponencial. Para empezar, pongamos nuestras proposiciones dadas en la forma normal prenex en la que todos los cuantificadores se aplican a la expresión completa. Resulta sencillo obtener una expresión en esta forma si primero renombramos las variables cuantificadas de modo que sean únicas, y después aplicamos las identidades

$$\neg(\forall x(E)) = \exists x(\neg E) \quad \forall x(E_1) \vee E_2 = \forall x(E_1 \vee E_2)$$

y cuatro reglas parecidas que se obtienen a partir de estas identidades intercambiando  $\forall$  y  $\exists$  y/o sustituyendo  $\vee$  por  $\wedge$ . Este proceso no llega a doblar la longitud de la expresión; los únicos símbolos que podrían agregarse a la expresión son un par de paréntesis por cuantificador.<sup>†</sup> Tenemos ahora la fórmula

$$Q_1 x_1 Q_2 x_2 \cdots Q_m x_m F(x_1, x_2, \dots, x_m), \quad (13.5)$$

en donde las  $Q_i$ s son cuantificadores, y la fórmula  $F$  no tiene cuantificadores.  $F$  es, por tanto, una expresión booleana cuyos operandos son átomos, un *átomo* es una constante booleana o una expresión de la forma  $E_1 \text{ op } E_2$ , en donde  $\text{op}$  es  $=$ ,  $<$ ,  $>$  o  $\leq$  y  $E_1$  y  $E_2$  son sumas de variables y las constantes 0 y 1. Sabemos que  $F$  es de esta forma porque ninguna otra combinación de operadores tiene sentido. Esto es,  $+$  puede aplicarse solamente a variables y constantes,  $<$  e  $=$  relacionan solamente a expresiones aritméticas y los operadores booleanos pueden aplicarse sensiblemente sólo a expresiones que tienen falso o verdadero como valores posibles.

Para determinar la verdad o falsedad de (13.5) sustituimos de manera repetida, para el cuantificador más interior, una cuantificación limitada, que consiste en el operador lógico “or” (en lugar de  $\exists$ ) o “and” (en lugar de  $\forall$ ) de un número grande pero finito de términos. Supóngase que en (13.5) fijamos los valores de  $x_1, x_2, \dots, x_{m-1}$ . Cada átomo que involucra a  $x_m$  puede ponerse de la forma  $x_m \text{ op } t$ , en donde  $\text{op}$  es  $<$ ,  $=$ ,  $>$  o  $\leq$  y  $t$  es de la forma

$$c_0 + \sum_{i=1}^{m-1} c_i x_i,$$

en donde las  $c_i$ s son números racionales. Supóngase que todos estos átomos son  $x_m \text{ op } t_i$ ,  $1 \leq i \leq k$ , en donde  $t_1 \leq t_2 \leq \dots \leq t_k$ , para los valores dados de  $x_1, \dots, x_{m-1}$ . Para cualquier valor de  $x_m$  que se encuentre en el intervalo  $t_i < x_m < t_{i+1}$ , cada átomo tiene el mismo valor de verdad. Por consiguiente la verdad de (13.5) es independiente del valor real de  $x_m$ .

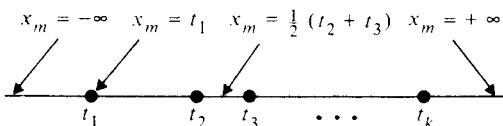


Fig. 13.6 Valores representativos de  $x_m$ .

<sup>†</sup> Técnicamente el renombramiento de variables puede aumentar la longitud de la fórmula en un factor de  $\log n$  cuando codificamos en un alfabeto fijo. Sin embargo, la complejidad depende del número de símbolos y no de la longitud de la cadena codificada.

en este intervalo. Esto nos conduce a la observación de que las  $t_i$ s parten la recta real en un número finito de segmentos, y la verdad de (13.5) depende solamente del segmento en el cual  $x_m$  se encuentra y no del valor de  $x_m$ . Por tanto, podemos probar (13.5) intentando un valor de  $x_m$  para cada región, a partir de un número finito de ellas, de la forma en que se sugiere en la Fig. 13.6.

Como los valores de  $x_1, \dots, x_{m-1}$ , variarán en realidad no sabemos el orden de las  $t_i$ s. Sin embargo, intentando  $x_m = t_i$  para cada  $i$ ,  $x_m = \frac{1}{2}(t_i + t_j)$ , para cada  $i \neq j$ , y  $x_m = \pm\infty$ ,<sup>†</sup> sabemos que no importa el orden de las  $t_i$ s, estamos seguros de tener una  $x_m$  representativa en cada intervalo de la Fig. 13.6 y también en las  $t_i$ s mismas, en donde los átomos que tienen el operador  $=$  pueden volverse verdaderos.

Se concluye que si  $Q_m = \exists$ , entonces  $\exists x_m F(x_1, \dots, x_m)$  puede sustituirse por

$$F'(x_1, \dots, x_{m-1}) = \bigvee_{\substack{x_m = t_i \text{ or} \\ x_m = (1/2)(t_i + t_j) \\ \text{or } x_m = \pm\infty}} F(x_1, \dots, x_m), \quad (13.6)$$

esto es, por el “or” lógico de  $k(k+1)/2 + 2$  términos, cada uno de los cuales es  $F$  con una sustitución por  $x_m$ . Si  $Q_m = \forall$ , se puede hacer una sustitución parecida, sustituyendo  $\wedge$  por  $\vee$ .

Si  $F$  tiene  $k$  átomos,  $F'$  tiene  $k[k(k+1)/2 + 2]$  átomos, que representa un máximo de  $k^3$  átomos para  $k \geq 3$ . También, si los coeficientes de los átomos de  $F$  son, para cada cociente de enteros, de cuando más  $r$  bits cada uno, entonces, después de agrupar términos, resolviendo para  $x_m$  y calculando el promedio de dos  $t_i$ s, encontramos que los coeficientes de los átomos de  $F'$  serán cocientes de enteros en los que se utilizan no más de  $4r + 1$  bits. Se concluye lo anterior debido a que si  $a, b, c$  y  $d$  son enteros de  $r$  bits,

$$\frac{a}{b} \frac{c}{d} = \frac{ac}{bd}$$

es el cociente de enteros con un máximo de  $2r$  bits y

$$\frac{a}{b} \pm \frac{c}{d} = \frac{ad \pm bc}{bd}$$

es el cociente de un entero de  $(2r+1)$  bits y uno de  $2r$  bits. Para  $r \geq 1$ , entonces los coeficientes de  $F'$  son de una longitud de no más de cinco veces la longitud de los coeficientes de  $F$ .

Si repetimos el procedimiento anterior para eliminar a todos los cuantificadores y variables, en algún momento produciremos una fórmula que contenga solamente a los operadores lógicos,  $=$ ,  $<$  y constantes. Las constantes son cocientes de enteros con cuando mucho  $5^m r$  bits. El número de átomos es de cuando mucho

$$(\underbrace{\dots ((k^3)^3) \dots}_m \text{ veces})^3 = k^{3^m}$$

Como cada átomo es una relación entre constantes de  $5^m r$  bits y  $k, m$  y  $r$  son menores

<sup>†</sup> Si  $x_m = +\infty$ , entonces  $x_m = t$  y  $x_m < t$  son falsas y  $x_m > t$  es verdadera, independientemente de  $t$ . Si  $x_m = -\infty$ , se da una simplificación parecida.

de  $n$ , la longitud de la expresión es de cuando mucho  $2^{2^n}$  para alguna constante  $c$  (nótese que  $k^{3^n} \leq 2^{2^n}$ ). Podemos evaluar un átomo de la forma  $a/b < c/d$  mediante el cálculo de  $ad - bc$  y comparándole con 0. Por consiguiente la expresión final completa puede evaluarse en el cuadrado de su longitud. En consecuencia nuestro procedimiento se lleva un tiempo de  $2^{2^d}$ , para alguna constante  $d$ .

El procedimiento, tal como lo hemos dado, se lleva también un espacio doble exponencial. Sin embargo, podemos reducir el espacio a un exponencial simple mediante la evaluación de  $F$  de manera recursiva. Ya hemos visto que necesitamos considerar sólo un conjunto finito de valores para cada  $x_i$ . Los valores de  $x_i$  están dados por una fórmula de la forma  $a_0 + \sum_{j=1}^{i-1} a_j x_j$ , en donde las  $a_j$ s son números racionales que, a su vez, son cocientes de enteros de  $5^{m-j+1}$  bits, en donde  $r$  es el número de bits de la constante más grande de la fórmula original,  $F$ ; nótese que  $r \leq \log n$ . Por consiguiente, los valores de  $x_i$  son números racionales que cuando mucho son cocientes de  $5^m r$ -bits, los valores para  $x_2$  son cocientes de enteros de cuando mucho  $5^{m+1} r$ -bits, etcétera. Así pues, necesitamos solamente hacer un ciclo a través de los valores de cada  $x_i$  que son de cuando mucho  $5^{2^m} r$ -bits. Utilizamos un procedimiento iterativo EVAL( $G$ ) que determina si  $G$  es verdadera cuando las variables toman los valores  $\pm\infty$  y cualquier cociente de enteros de  $5^{2^m} r$ -bits.

Si  $G$  no posee cuantificadores, entonces ésta consiste solamente en relaciones aritméticas y lógicas entre números racionales, de modo que su veracidad puede ser determinada directamente. Si  $G = \forall x(G')$ , EVAL( $G$ ) llama a EVAL( $G'$ ), para toda  $G'$  formada a partir de  $G$  mediante la sustitución de  $x$  por  $\pm\infty$  o un cociente de enteros de  $5^{2^m} r$  bits. EVAL( $G$ ) es **veradero** si EVAL( $G'$ ) se vuelve verdadera para todas estas expresiones  $G'$ . Si  $G = \exists x(G')$ , hacemos lo mismo, pero EVAL( $G$ ) se vuelve **verdadera** cuando alguna EVAL( $G'$ ) es verdadera.

Resulta fácil verificar que no más de  $m$  copias de EVAL se encuentran activas de manera simultánea. Los argumentos para las llamadas activas a EVAL pueden ponerse en una pila, y esta pila toma un espacio  $O(m5^{2^m}r)$ . Por consiguiente, si  $F$  es una expresión de longitud  $n$ , podemos evaluar  $F$  en espacio  $2^n$  y un tiempo de  $2^{2^d}$ , para algunas constantes  $c$  y  $d$ .

### Un límite inferior

Mostraremos, ahora, que la teoría de los números reales con la adición requiere, esencialmente, un tiempo exponencial no determinístico. Se necesitan una serie de lemas que muestran que la multiplicación y la exponentiación por enteros de tamaño limitado pueden expresarse mediante fórmulas cortas.

**Lema 13.10** Existe  $c > 0$  tal que, para cada  $n$ , existe una fórmula  $M_n(x, y, z)$  que es verdadero si y sólo  $x$  es un entero no negativo estrictamente menor que  $2^{2^n}$  y  $xy = z$ . Aún más,  $|M_n(x, y, z)| < c(n+1)$  y  $M_n(x, y, z)$  puede construirse a partir de  $n$  en un tiempo polinomial en  $n$ .

*Demostración* Para  $n = 0$ ,  $2^{2^0} = 2$ . Por consiguiente  $M_0(x, y, z)$  puede expresarse como  $(x = 0 \wedge z = 0) \vee (x = 1 \wedge z = y)$ .

*Paso inductivo:* (Construcción de  $M_{n+1}$  a partir de  $M_n$ ). Sea  $x$  un entero menor que  $2^{2^{n+1}}$ .

Existen enteros  $x_1, x_2, x_3, x_4 < 2^{2^n}$  tales que  $x = x_1 x_2 + x_3 + x_4$ . Para la prueba, sea  $x_1 = x_2 = \lfloor \sqrt{x} \rfloor$ . Ahora  $z = xy$  puede expresarse mediante  $z = x_1(x_2y) + x_3y + x_4y$ . Por consiguiente

$$\begin{aligned} M_{n+1}(x, y, z) &= \exists u_1 \cdots \exists u_5 \exists x_1 \cdots \exists x_4 [M_n(x_1, x_2, u_1) \\ &\quad \wedge x = u_1 + x_3 + x_4 \wedge M_n(x_2, y, u_2) \wedge M_n(x_1, u_2, u_3) \wedge M_n(x_3, y, u_4) \\ &\quad \wedge M_n(x_4, y, u_5) \wedge z = u_3 + u_4 + u_5] \end{aligned} \quad (13.7)$$

Esto es,

$$\begin{aligned} u_1 &= x_1 x_2, & x &= x_1 x_2 + x_3 + x_4, \\ u_2 &= x_2 y, & u_3 &= x_1 x_2 y, & u_4 &= x_3 y, & u_5 &= x_4 y, \\ y & & z &= x_1 x_2 y + x_3 y + x_4 y. \end{aligned}$$

La condición de que cada  $x_i$  sea un entero menor que  $2^{2^n}$  se ve reforzada por cada  $x_i$  que es el primer argumento de alguna  $M_k$ .

La fórmula (13.7) tiene cinco copias de  $M_n$ , de modo que parece que  $M_{n+1}$  debe ser, al menos, de longitud cinco veces mayor que  $M_n$ . Esto haría la longitud de  $M_n$  exponencial en  $n$ , no lineal como aseveramos. Sin embargo, podemos utilizar el “truco” del Teorema 13.10 para sustituir varias copias de un predicado por una sola copia. Esto es, podemos escribir

$$\begin{aligned} M_{n+1}(x, y, z) &= \exists_1 \cdots \exists u_5 \exists x_1 \cdots \exists x_4 \\ &\quad [x = u_1 + x_3 + x_4 \wedge z = u_3 + u_4 + u_5 \\ &\quad \wedge \forall r \forall s \forall t [\neg r = x_1 \wedge s = x_2 \wedge t = u_1) \\ &\quad \wedge \neg(r = x_2 \wedge s = y \wedge t = u_2) \\ &\quad \wedge \neg(r = x_1 \wedge s = u_2 \wedge t = u_3) \\ &\quad \wedge \neg(r = x_3 \wedge s = y \wedge t = u_4) \\ &\quad \wedge \neg(r = x_4 \wedge s = y \wedge t = u_5) \\ &\quad \vee M_n(r, s, t)]], \end{aligned}$$

la cual tiene un número constante de símbolos más que los que tiene  $M_n$ .

Un punto secundario consiste en que si introducimos nuevos nombres de variables para cada  $M_n$ , finalmente introduciremos un factor  $\log n$  en la longitud de  $M_n$ , ya que los nombres de variables deben codificarse en un alfabeto fijo en el lenguaje de las fórmulas verdaderas. Sin embargo, el dominio de las reglas para las fórmulas cuantificadas nos permite reutilizar variables sujetas a la restricción de que las doce nuevas variables introducidas en  $M_n$  no entran en conflictos con las variables libres  $x$ ,  $y$  y  $z$ . Por consiguiente  $M_n$  requiere solamente 15 variables diferentes y su longitud codificada es proporcional al número de símbolos.  $\square$

Obsérvese que  $M_n(x, 0, 0)$  establece que  $x$  es un entero menor que  $2^{2^n}$ . Por tanto,

podemos hacer proposiciones concernientes con enteros pequeños en la teoría de los reales con la adición mediante el uso de fórmulas muy cortas

**Lema 13.11** Existe una constante  $c > 0$  tal que, para cada  $n$ , existe una fórmula  $P_n(x, y, z)$  que es verdadera si y sólo si  $x$  y  $z$  son enteros que se encuentran en el intervalo  $0 \leq x, z < 2^n$  y  $y^x = z$ . Aún más,  $|P_n| \leq c(n+1)$  y  $P_n$  puede construirse a partir de  $n$  en tiempo polinomial en  $n$ .

*Demostración* Construimos por inducción sobre  $k$  una secuencia de fórmulas  $E_k(x, y, z, u, v, w)$  tal que  $E_k$  tiene a la exponenciación y la multiplicación construidas en ella. La razón para hacer esto consiste en que deseamos expresar  $E_k$  en términos de diferentes copias de  $E_{k-1}$  y después utilizar la cuantificación universal para expresar a  $E_k$  en términos de una copia de  $E_{k-1}$ . No podríamos hacer esto con  $P_k$ , ya que  $P_k$  involucra tanto a  $P_{k-1}$  como a  $M_{k-1}$ .

La fórmula  $E_k(x, y, z, u, v, w)$  será verdadera si y sólo si  $x, z$  y  $u$  son enteros tales que  $0 \leq x, z < 2^k$ ,  $z = y^x$ ,  $0 \leq u < 2^k$  y  $uv = w$ .

*Base* Para  $k = 0$ ,

$$\begin{aligned} E_0 = & (x = 0 \wedge z = 1) \vee (x = 1 \wedge y = 0 \wedge z = 0) \\ & \vee (x = 1 \wedge y = 1 \wedge z = 1) \wedge M_0(u, v, w). \end{aligned}$$

*Inducción* Para construir  $E_{k+1}(x, y, z, u, v, w)$  podemos utilizar el hecho de que

$$E_k(0, 0, 0, u, v, w) = M_n(u, v, w)$$

para expresar las condiciones sobre  $u, v$  y  $w$  como en el Lema 13.10. Utilizando varias copias de  $E_k$ , podemos afirmar que existen enteros  $x_1, x_2, x_3, x_4$ , que se encuentran en el intervalo  $0 \leq x_i < 2^k$  tales que

$$x = x_1 x_2 + x_3 + x_4 \quad y \quad y^x = (y^{x_1})^{x_2} y^{x_3} y^{x_4}.$$

Finalmente, utilizamos el “truco” que se usó en el Teorema 13.10 para expresar  $E_{k+1}$  en términos de una copia de  $E_k$  y un número constante de símbolos adicionales. Por último podemos escribir

$$P_n(x, y, z) = E_n(x, y, z, 0, 0, 0).$$

Esto nos asegura que  $z = y^x$  y que  $x$  y  $z$  son enteros que se encuentran en el intervalo  $0 \leq x, z < 2^n$ .  $\square$

Para mejorar la legibilidad de lo que sigue, utilizamos las abreviaturas  $2$  por  $1 + 1$ ,  $2x$  por  $x + x$ ,  $x \leq y$  por  $x < y \vee x = y$  y  $x \leq y < z$  por  $(x = y \vee x < y) \wedge y < z$ . Al expandirse una fórmula abreviada se tiene como resultado, cuando más, una multiplicación por un actor constante. Además de las abreviaturas anteriores usaremos constantes como  $2^n$  y multiplicaciones como  $ab$  en las fórmulas. Técnicamente, éstas deben ser sustituidas mediante la introducción de una variable existencialmente cuantificada,

digamos  $x$ , y aseguramos que  $x = 2^n$  o  $x = ab$  mediante  $p_n(n, 2, x)$  o  $M_n(a, b, x)$ . Esto puede también incrementar la longitud de la fórmula en un factor constante.

Intentaremos codificar los cálculos de una máquina de Turing como enteros. Sea  $M$  una NTM con límite temporal  $2^n$ . Si el número total de símbolos de cinta, símbolos compuestos y el señalador # es  $b$ , entonces un cálculo de  $M$  es un entero  $x$  en el intervalo  $0 \leq x < b^{(2^n+1)^2+1}$ . El afirmar que un entero es un cálculo se facilita mediante un predicho que interroga al  $i$ -ésimo dígito en la representación  $b$ -aria  $\dagger$  de  $x$ .

**Lema 13.12** Para cada  $n$  y  $b$  existe una constante  $c$ , que depende solamente de  $b$ , tal que existe una fórmula  $D_{n,b}(x, i, j)$  que es verdadera si y sólo si  $x$  e  $i$  son enteros,  $0 \leq x < b^{(2^n+1)^2+1}$ ,  $0 \leq i < 2^n$ , y  $x_i$ , el  $i$ -ésimo dígito de  $x$  contando desde el extremo de orden más bajo de la representación  $b$ -aria de  $x$ , es  $j$ . Aún más,  $|D_{n,b}| \leq c(n+1)$  y  $D_{n,b}$  puede construirse a partir de  $n$  y  $b$  en un tiempo polinomial en  $n$  y  $b$ .

*Demostración* Para cada  $b$  existe una constante  $s$  tal que  $b^{(2^n+1)^2+1} \leq 2^{2^n}$ , para toda  $n$ . Por consiguiente, el hecho de que  $x$  es un entero que se encuentra en el intervalo correcto puede expresarse mediante  $\exists m [P_{ns}((2^n+1)^2, b, m) \wedge 0 \leq x < m]$ . (Recuérdese nuestro señalamiento anterior que trata sobre las constantes como  $2^n$  y sus expansiones.) El hecho de que  $i$  es un entero que se encuentra en el intervalo  $0 \leq i < 2^n$  puede expresarse mediante  $M_n(i, 0, 0) \wedge (0 \leq i < 2^n)$ . Ahora,  $x$  en base  $b$  tiene ceros en las posiciones  $1, 2, \dots, i+1$  si y sólo si es divisible entre  $b^{i+1}$ . Por tanto,  $x_i = j$  si y sólo si existen enteros  $q$  y  $r$  tales que  $x = q^{b^{i+1}} + r$  y  $jb^i \leq r < (j+1)b^i$ . Este hecho se puede expresar de manera sencilla utilizando  $P_{ns}$  y  $M_{ns}$ .  $\square$

**Teorema 13.16** Cualquier algoritmo no determinístico para decidir si una fórmula en la teoría de primer orden de los números reales con la adición es verdadera debe, para alguna constante  $c \geq 0$ , tomar  $2^n$  pasos para un número infinito de  $ns$ .

*Demostración* La demostración es bastante parecida, en espíritu, a la del Teorema 13.1. Sea  $M$  una NTM cualquiera con límite temporal  $2^n$ . Aquí las IDs que se encuentran en un cálculo de  $M$  consisten en  $2^n$  símbolos, más que en  $p(n)$  como en el Teorema 13.1. Sea  $b$  el número total de símbolos de cinta, símbolos compuestos y #s. Entonces un cálculo de  $M$  sobre una entrada de longitud  $n$  consiste en  $[(2^n+1)^2+1]$  dígitos en  $b$ -aria. Podemos considerar este cálculo como un entero  $i$  en el intervalo  $0 \leq i < 2^{2^n}$ , para alguna constante  $s$ . Por conveniencia, tomamos los dígitos de orden inferior de  $i$  que se encuentran en el extremo izquierdo del cálculo.

Sea  $x$  una entrada de longitud  $n$  para  $M$ . Construimos una fórmula  $F_x$  que es verdadera si y sólo si  $M$  acepta a  $x$ .  $F_x$  es de la forma  $\exists i(\dots)$ , en donde la fórmula dentro del paréntesis asegura que  $i$  es un cálculo de aceptación de  $x$ . Esta fórmula es parecida a la del Teorema 13.1. Los primeros  $n+1$  símbolos del cálculo son

<sup>†</sup> Se refiere al sistema numérico en que se encuentra representado  $x$ ; si  $b = 2$ , la representación sería en binario (N. del T.).

$$\#[q_0, a_1, m]a_2 \cdots a_n.$$

suponiendo que  $x = a_1 a_2 \cdots a_n$ ,  $q_0$  sea el estado inicial y  $m$  sea cualquier alternativa de primer movimiento. Para decir que los primeros  $n + 1$  símbolos del cálculo son correctos, decimos que existen  $u$  y  $j$  tales que el valor de  $u$  representa a  $\#[q_0, a_1, m]a_2 \cdots a_n$ , para alguna  $m$ , e  $i = b^{n+1}j + u$ , para algún entero  $j$ .

Debemos escribir esta fórmula en espacio  $O(n)$  en un tiempo polinomial en  $n$ . Por inducción sobre  $k = 2, 3, \dots, n + 1$  podemos escribir una fórmula  $C_k(v)$  con variable libre  $v$ , que asegura que el valor de  $v$  es el valor numérico de los primeros  $k$  símbolos del cálculo. Para la base,  $k = 2$ , simplemente escribimos la fórmula

$$C_2(v) = (v = p_1 \vee v = p_2 \vee \cdots \vee v = p_n),$$

en donde las  $p_j$ s son los enteros representados por  $\#[q_0, a_1, m]$  para el conjunto finito de valores de  $m$ . Para la inducción,

$$C_k(v) = \exists w(C_{k-1}(w) \wedge v = bw + a_{k-1}),$$

en donde  $a_{k-1}$  se toma como el valor numérico del símbolo de cinta  $a_{k-1}$ . Para evitar el tener que utilizar  $n$  variables para expresar  $C_{n+1}$ , lo que haría su longitud de  $O(n \log n)$ , alternamos entre dos variables, como  $v$  y  $w$ , conforme construimos  $C_2, C_3, \dots, C_{n+1}$ .

La fórmula deseada asegura  $C_{n+1}(u)$  e  $i = b^{n+1}j + u$ , para el entero  $j$ . La última afirmación es parecida a lo que se hizo en el Lema 13.12, y la técnica no se repetirá aquí.

Para expresar que la ID inicial era correcta, en el Teorema 13.1, requerimos asegurar que “aproximadamente”  $p(n)$  celdas contenían el símbolo para el espacio en blanco. Esto se llevó a cabo mediante el operador lógico  $\vee$  de  $p(n)$  ítems. Ahora debemos asegurar que cerca de  $2^n$  celdas contienen el símbolo para el espacio en blanco y que por consiguiente no podemos utilizar el  $\vee$  lógico de  $2^n$  fórmulas; ésta sería una fórmula muy larga. En su lugar utilizamos el cuantificador  $\forall j$  y aseguramos que  $j$  no es un entero que se encuentra en el intervalo  $n + 2 < j \leq j \leq 2^n + 1$ , o el  $j$ -ésimo símbolo es el espacio en blanco, lo cual denotamos por 0. Así pues escribimos

$$\forall j[\neg M_{sn}(j, 0, 0) \vee \neg(n + 2 \leq j \leq 2^n + 1) \vee D_{n,b}(i, j, 0)].$$

Las fórmulas que fuerzan a que la última ID contenga un estado final, y que fuerzan a que cada ID se concluya de la ID anterior, debido a la alternativa de movimiento que se encuentra embebida en la ID anterior, se traducen de manera similar a partir de las técnicas del Teorema 13.1. Una vez que se ha hecho esto, tenemos la fórmula  $E_x$ , cuya longitud es proporcional a  $n$  y que es verdadera si y sólo si  $M$  acepta a  $x$ .

Supóngase que  $M$  acepta a un lenguaje  $L$  en un tiempo  $2^n$ , que a su vez no es aceptado por ninguna NTM con límite temporal  $2^{n/2}$ . (La existencia de un lenguaje tal se concluye de la jerarquía TIEMPON de Cook [1973a], que no hemos demostrado.) Podemos reconocer a  $L$  de la manera siguiente. Dada  $x$  de longitud  $n$ , produzcáse la fórmula  $E_x$  que es verdadera si y sólo si  $x$  se encuentra en  $L$ . Ahora, si un tiempo no determinístico  $T(n)$  es suficiente para aceptar al conjunto de fórmulas verdaderas en la teoría de primer orden de los reales con la adición, podemos determinar si  $x$  se encuentra en  $L$  en un tiempo

$p(n) + T(cn)$ . Entonces  $p(n) + T(cn) > 2^{n/2}$ , para cualquier número infinito de  $ns$ ; de otra manera, según el Lema 12.3, podemos reconocer a  $L$  en un tiempo de cuando más  $2^{n/2}$ , para toda  $n$ . Se concluye que  $T(n) \geq 2^{dn}$  i.o., para alguna  $d > 0$ .  $\square$

**Corolario** La teoría de los números reales con la adición es difícil en tiempo no determinístico exponencial con respecto a las reducciones en tiempo polinomial.

**Demostración** La demostración es una sencilla generalización de la anterior reducción de una TM de tiempo no determinístico  $2^n$ .  $\square$

### 13.7 LA CUESTION $\mathcal{P} = \mathcal{NP}$ PARA LAS MAQUINAS DE TURING CON ORACULOS: LIMITES EN NUESTRA CAPACIDAD PARA DETERMINAR SI $\mathcal{P} = \mathcal{NP}$

El lector recordará, de la Sección 8.9, nuestra discusión sobre las máquinas de Turing con oráculo. Estas TMs tienen asociados lenguajes, conocidos como oráculos, y tienen estados especiales en los que se puede probar, en un solo paso, si la membresía de una cadena escrita hacia la izquierda de la cabeza tiene membresía en el oráculo. Cualquier TM con oráculo puede tener a cualquier oráculo “reconectado”, aunque su comportamiento, naturalmente, variará dependiendo del oráculo escogido. Si  $A$  es un oráculo, utilizamos  $M^A$  para representar a  $M$  con el oráculo  $A$ . El tiempo que se toma una TM con oráculo de un paso en cada interrogación al oráculo y un paso en cada movimiento ordinario de la TM.

Definimos  $\mathcal{P}^A$  como el conjunto de lenguajes aceptados en tiempo polinomial por las DTMs con oráculos  $A$ . Definimos también  $\mathcal{NP}^A$  como el conjunto de lenguajes aceptados por las NTMs con oráculo  $A$  en un tiempo polinomial. Demostraremos que existen oráculos  $A$  y  $B$  para los cuales  $\mathcal{P}^A = \mathcal{NP}^A$  y  $\mathcal{P}^B \neq \mathcal{NP}^B$ . Este resultado tiene implicaciones que tienen que ver con nuestra habilidad para resolver la cuestión  $\mathcal{P} = \mathcal{NP}$  para las TMs sin oráculos. De manera intuitiva se sabe que todos los métodos conocidos para resolver la cuestión de un modo u otro funcionarán cuando se les asignan oráculos arbitrarios. Pero la existencia de  $A$  y  $B$  nos dice que ninguno de estos métodos puede funcionar para oráculos arbitrarios. Por consiguiente los métodos que ya existen probablemente no son suficientes para establecer si  $\mathcal{P} = \mathcal{NP}$ . Proporcionaremos algunos detalles en este mismo sentido después de ver la construcción de  $A$  y  $B$ .

#### Un oráculo para el cual $\mathcal{P} = \mathcal{NP}$

**Teorema 13.17**  $\mathcal{P}^A = \mathcal{NP}^A$ , en donde  $A = L_{\text{qbr}}$ , el conjunto de todas las fórmulas booleanas cuantificadas como verdaderas (o cualquier otro problema, ESPACIOP-completo).

**Demostración** Sea  $M^A$  una máquina con límite en tiempo polinomial no determinístico, y sea  $L(M^A)$ . Entonces  $M^A$  interroga a su oráculo un número polinomial de veces sobre las cadenas cuyas longitudes están limitadas por un polinomio de la longitud de la entrada de  $M^A$ . Por consiguiente podemos simular el cálculo del oráculo en un espacio polinomial. Se concluye que  $\mathcal{NP}^A \subseteq \text{ESPACIOP}$ . Sin embargo, cualquier lenguaje  $L$  en ESPACIOP es aceptado por alguna DTM  $M^A$  que reduce  $L$  a  $A$  en un tiempo polinomial y después interroga a su oráculo. Por consiguiente,  $\text{ESPACIOP} \subseteq \mathcal{P}^A$ . Pero es claro que  $\mathcal{P}^A \subseteq \mathcal{NP}^A$ , de modo que  $\mathcal{P}^A = \mathcal{NP}^A$ .  $\square$

### Un oráculo para el cual $\mathcal{P} \neq \mathcal{NP}$

Mostraremos ahora cómo construir un oráculo  $B \subseteq (0+1)^*$  para el cual  $\mathcal{P}^B \neq \mathcal{NP}^B$ .  $B$  tendrá cuando más una palabra de cualquier longitud; qué palabra exactamente es algo que será discutido más adelante. Nos centraremos en el lenguaje

$$L = \{0^i \mid B \text{ tiene una palabra de longitud } i\}.$$

Podemos construir, de manera sencilla, una NTM con oráculo  $B$  que, dada la entrada  $0^i$ , adivina una cadena de longitud  $i$  en  $(0+1)^*$  e interroga a su oráculo acerca de su cadena supuesta, aceptando si el oráculo dice "sí". Por tanto,  $L$  estará en  $\mathcal{NP}^B$ . Sin embargo, podemos construir  $B$  de tal manera que la cadena de cualquier longitud, si existe, está tan bien escondida que una DTM con oráculo  $B$  no puede encontrarla en tiempo polinomial.

**Teorema 13.18** Existe un oráculo  $B$  para el cual  $\mathcal{P}^B \neq \mathcal{NP}^B$ .

**Demostración** Daremos un procedimiento para enumerar el conjunto  $B$ . El conjunto  $B$  tendrá cuando más una palabra de cualquier longitud. Conforme generamos a  $B$ , mantenemos una lista de las palabras que no están permitidas; estas palabras no se toman en consideración debido a que posiblemente pertenezcan a  $B$ . Supóngase una enumeración de DTMs con oráculo y alfabeto de entrada  $\{0, 1\}$ , en donde cada TM aparece con una frecuencia infinita. Consideramos a cada  $M_i$ ,  $i = 1, 2, \dots$ , por turno. Una vez que se ha considerado a  $M_i$ , habremos generado algunas palabras no permitidas y un conjunto  $B_i$  de palabras que hasta ahora están en  $B$ . Cuando mucho habrá una palabra en  $B_i$  de longitud  $0, 1, \dots, i-1$ , y ninguna palabra de longitud mayor. Aún más, ninguna otra palabra de longitud menor que  $i$  será colocada en  $B$  posteriormente. Simuлимamos  $M_i^{Bi}$  sobre la entrada  $0^i$ . Si  $M_i$  interroga a una palabra de longitud menor que  $i$ , consultamos a  $B_i$ , que está constituido por todas las palabras que se encuentran en  $B$  hasta ahora, para ver si el oráculo responde "sí" o "no". Si  $M_i$  interroga a una palabra de longitud  $i$  o mayor, suponemos que y no se encuentra en  $B$  (es decir, la respuesta es "no") y para asegurarnos que y no será colocada en  $B$  más adelante, añadimos y a la lista de palabras no permitidas.

La simulación de  $M_i^{Bi}$  sobre la entrada  $0^i$  continúa en  $i^{\log i}$  pasos. Después, se haya detenido  $M_i$  o no, tomamos una decisión acerca de si poner una palabra en  $B$ . Si dentro de  $i^{\log i}$  pasos,  $M_i^{Bi}$  se detiene y rechaza a  $0^i$ , entonces colocamos una palabra de longitud  $i$  que no se encuentra en la lista de palabras prohibidas en  $B$ , siempre y cuando exista dicha palabra. La palabra puede tomarse de manera arbitraria, digamos la primera palabra que lexicográficamente no está prohibida. Si  $M_i^{Bi}$  no rechaza a  $0^i$  dentro de  $i^{\log i}$  pasos, entonces no se coloca ninguna palabra de longitud  $i$  en  $B$ .

Tampoco existirá ninguna palabra de longitud  $i$  en  $B$  si todas las palabras de longitud  $i$  están prohibidas al momento de terminar la simulación de  $M_i^{Bi}$ . Sin embargo, el número de pasos simulados por  $M_i^{Bi}$  es  $j^{\log j}$ , de modo que el número total de palabras de todas las longitudes que no son permitidas por  $M_1, M_2, \dots, M_i$  es, cuando mucho

$$\sum_{j=1}^i j^{\log j} \leq i(i^{\log i}) \leq i^{1+\log i}.$$

Como existen  $2^i$  palabras de longitud  $i$ , sabemos que no todas las palabras de longitud  $i$  están prohibidas si  $2^i > i^{1+\log i}$ , esto es, si  $i > (1 + \log i) \log i$ . Pero la última relación vale para  $i \geq 32$ , de modo que es solamente para un número finito de  $i$ s pequeñas que todas las palabras de longitud  $i$  podrían estar prohibidas.

Una vez que hemos terminado la simulación de  $M_i^{Bi}$  sobre la entrada  $0^i$  para  $i^{\log i}$  pasos, generamos la palabra seleccionada, si existe alguna, obteniendo un nuevo conjunto  $B_{i+1}$  de palabras generadas. Estamos, ahora, listos para repetir el proceso para  $M_{i+1}^{Bi+1}$  sobre  $0^{i+1}$ .

En seguida definiremos un lenguaje  $L$  que está en  $\mathcal{NP}^B - \mathcal{P}^B$ . Sea

$$L = \{0^i \mid B \text{ tiene una palabra de longitud } i\}.$$

Podemos fácilmente construir una NTM de tiempo lineal con oráculo  $B$  que, dada la entrada  $0^i$ , adivina no determinísticamente la cadena  $w$  de longitud  $i$  en  $(0+1)^*$  e interroga a su oráculo acerca de  $w$ , aceptándola si el oráculo dice "sí". Por consiguiente  $L$  se encuentra en  $\mathcal{NP}^B$ .

Supóngase que  $L$  se encuentra en  $\mathcal{P}^B$ . Hagamos que  $M_k^B$  acepte a  $L$ , en donde  $M_k^B$  es una TM con límite temporal determinístico que es un polinomio  $p(n)$  con oráculo  $B$ . Como cada TM tiene códigos arbitrariamente largos, podemos escoger  $k$  tal que  $k \geq 32$  y  $k^{\log k} \geq p(k)$ . Si  $M_k^B$  acepta a  $0^k$ , entonces  $0^k$  se encuentra en  $L$ , de modo que  $B$  tiene una palabra de longitud  $k$ . Esto significa que  $M_k^{Bk}$  rechaza a  $0^k$ . Pero  $M_k^{Bk}$  y  $M_k^B$  deben comportarse de manera idéntica sobre la entrada  $0^k$ , ya que  $B$  y  $B_k$  coinciden en las palabras de longitud menor a  $k$ , y  $B$  no tiene palabras de longitud  $k$  o mayores que son interrogadas por  $M_k^{Bk}$  sobre  $0^k$ . Por consiguiente  $M_k^{Bk}$  rechaza a  $0^k$ , lo que constituye una contradicción.

Si  $M_k^B$  rechaza a  $0^k$ , de manera que  $0^k$  no se encuentra en  $L$ , entonces  $M_k^{Bk}$  no puede rechazar a  $0^k$  dentro de  $K^{\log k}$  pasos. Esto se concluye porque  $k \geq 32$  y en caso de que  $M_k^{Bk}$  hubiera rechazado a  $0^k$  dentro de  $K^{\log k}$  pasos, aún existiría una palabra de longitud  $k$  que no se encuentra en la lista de palabras prohibidas, y esta palabra estaría en  $B$ . Por consiguiente  $0^k$  estaría en  $L$ . En consecuencia,  $M_k^B$  no rechaza a  $0^k$  dentro de  $K^{\log k}$  pasos. Pero como  $K^{\log k} \geq p(k)$ ,  $M_k^B$  no rechaza a  $0^k$  de ninguna manera, lo que constituye otra contradicción. Concluimos que  $L$  se encuentra en  $\mathcal{NP}^B - \mathcal{P}^B$ .  $\square$

### Significancia de los resultados del oráculo

Consideraremos las formas que hemos utilizado en este libro para demostrar que dos clases de lenguajes son iguales o diferentes, y veamos por qué los Teoremas 13.17 y 13.18 sugieren que estos métodos no tendrán éxito al resolver la cuestión  $\mathcal{P} = \mathcal{NP}$ . Demostramos que ciertas clases son iguales mediante simulación. Por ejemplo, el Capítulo 7 contiene muchas simulaciones de un tipo de TM por otro. El Capítulo 5 contiene simulaciones de PDAs mediante CFGs y viceversa.

Supóngase que podemos simular NTMs cualesquiera con límite temporal polinomial, mediante DTMs con límite temporal polinomial. ( Nótese que dar un algoritmo en tiempo polinomial para cualquier problema  $NP$ -completo es, en efecto, una simulación en tiempo polinomial de todas las NTMs.) Es probable que la

simulación todavía sea válida si asignamos el mismo oráculo a cada TM. Por ejemplo, todas las simulaciones del Capítulo 7 son aún válidas si utilizamos TMs con oráculos. Pero entonces tendríamos  $\mathcal{P}^B = \mathcal{NP}^B$ , lo que acabamos de demostrar que es falso.

Se demostró que otras clases de lenguajes son distintas mediante diagonalización. Los teoremas de jerarquía, Teoremas 12.8 y 12.9, y la demostración de que  $L_u$  es un conjunto r.e., pero no recursivo, constituyen importantes ejemplos. La diagonalización también tiende a funcionar cuando los oráculos se asignan a las TMs, por lo menos en los tres ejemplos que se citaron más arriba. Si pudiéramos diagonalizar sobre  $\mathcal{P}$  para mostrar que un lenguaje se encuentra en  $\mathcal{NP} - \mathcal{P}$ , entonces la misma demostración funcionaría también para demostrar que  $\mathcal{NP}^A - \mathcal{P}^A \neq \emptyset$ . Esto violaría el Teorema 13.17.

También utilizamos los lemas de traslación para refinar las jerarquías temporales y espaciales en el Capítulo 12. ¿Podría esto ayudarnos a demostrar que  $\mathcal{P} = \mathcal{NP}$ ? Probablemente no, porque los lemas de traslación también valen cuando se tienen oráculos.

Por último, podemos utilizar las propiedades de cerradura para mostrar una diferencia entre dos clases de lenguajes. Por ejemplo los DCFLs están contenidos en los CFLs, pero los DCFLs son cerrados con respecto a la complementación y los CFLs no. Esto demuestra que existe un CFL que no está en un DCFL. ¿Nos sería posible hallar una propiedad de cerradura de  $\mathcal{P}$  que no sea compartida por  $\mathcal{NP}$ ? A primera vista, esto parece el planteamiento más prometedor. Mientras que es probable que las demostraciones de que  $\mathcal{P}$  es cerrado con respecto a una operación muestren que  $\mathcal{P}^A$  es cerrado con respecto a dicha operación, un resultado de no cerradura para  $\mathcal{NP}$  podría no trasladarse a  $\mathcal{NP}^A$ . Por otro lado, mostrar que  $\mathcal{NP}$  no es cerrado con respecto a una operación involucra demostrar que un lenguaje particular no se encuentra en  $\mathcal{NP}$ . Esta demostración podría llevarse a cabo mediante diagonalización; pero entonces sería probable trasladarla a  $\mathcal{NP}^A$ . Podría hacerse desarrollando un lema de bombeo para  $\mathcal{NP}$ , pero parece estar mucho más allá de la capacidad presente. Por último, podríamos desarrollar algunos argumentos *ad hoc* de nuevo, ninguno de tales argumentos se ha podido encontrar, siendo éstos muy difíciles.

## EJERCICIOS

**13.1** Supóngase que existe una reducción con límite temporal  $2^n$  de  $L_1$  a  $L_2$  y que  $L_2$  se encuentra en TIEMPOD( $2^n$ ). ¿Qué podemos concluir acerca de  $L_1$ ?

**13.2** ¿Cuáles de las siguientes formas booleanas son satisfactores?

a)  $\bar{x}_1 \wedge x_3 \wedge (\bar{x}_2 \vee \bar{x}_3)$

\*b)  $\bigwedge_{i_1, i_2, i_3} (x_{i_1} \vee x_{i_2} \vee x_{i_3}) \wedge \bigwedge_{i_1, i_2, i_3} (\bar{x}_{i_1} \vee \bar{x}_{i_2} \vee \bar{x}_{i_3})$

en donde  $(i_1, i_2, i_3)$  varía sobre todos los triples de tres enteros distintos que se encuentran entre 1 y 5.

**13.3** Un grupúsculo de una gráfica  $G$  es una subgráfica de  $G$  que es completa; es decir, cada par de vértices está conectado por una arista. El problema del grupúsculo consiste en determinar si una gráfica dada contiene un grupúsculo de un tamaño  $k$  dado.

- a) Formule el problema del grupúsculo como un problema de reconocimiento de lenguaje.  
 b) Demuestre que el problema del grupúsculo es  $NP$ -completo mediante la reducción del problema de cobertura de vértice al problema del grupúsculo.

[*Sugerencia:* Considérese un grafo  $G$  y su complemento  $\bar{G}$ , en donde  $\bar{G}$  tiene una arista si y sólo si  $G$  no tiene dicha arista.]

**13.4** Dados un grafo  $G$  y un entero  $k$ , el *problema de cobertura de grupúsculo* consiste en determinar si existen  $k$  grupúsculos en  $G$  tales que cada vértice de  $G$  se encuentre en al menos uno de los  $k$  grupúsculos. Demuestre que el problema de cobertura de grupúsculo es  $NP$ -completo mediante la reducción del problema de la cobertura de vértice al problema de la cobertura de vértice para gráficas sin triángulos y, por consiguiente, al problema de cobertura de grupúsculo [*Sugerencia:* Considérense los grafos  $G = (V, E)$  y

$$G' = (E, \{(e_1, e_2) \mid e_1 \text{ y } e_2 \text{ inciden sobre el mismo vértice de } G\})$$

**13.5** El grafo de la Fig. 13.7

- a) ¿tiene un circuito de Hamilton?  
 b) ¿tiene una cobertura de vértice de tamaño 10?  
 c) ¿tiene un coloreado de vértice con dos colores tales que ninguno de los dos vértices adyacentes son del mismo color?

**13.6** Demuestre que el problema del número cromático es  $NP$ -completo mediante la reducción del problema de la satisfactoriedad 3-CNF al problema del número cromático. [*Sugerencia:* El grafo de la Fig. 13.8 puede utilizarse como una subgráfica en la construcción. Nótese que cada  $v_i$  debe colorearse con un color distinto, digamos el color  $i$ . El grafo completo puede colorearse con  $n + 1$  colores si y sólo si, para cada  $i$ ,  $1 \leq i \leq n$ , uno de los  $x_i$  y  $\bar{x}_i$  está coloreado con el color  $i$  y el otro con el color  $n + 1$ .]

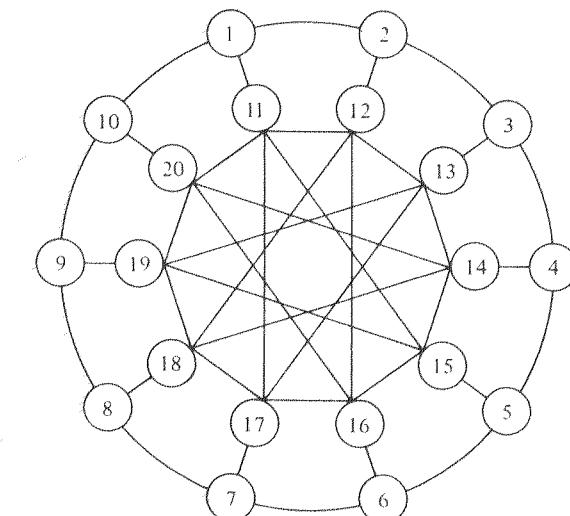
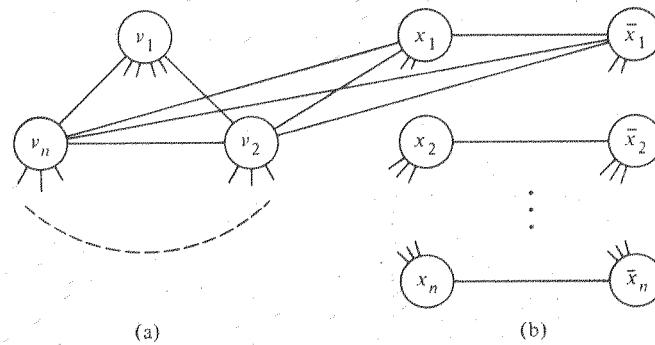


Fig. 13.7 Un grafo no dirigido.



$$V = \{v_i, x_i, \bar{x}_i \mid 1 \leq i \leq n\}$$

$$E = \{(v_i, v_j) \mid i \neq j\} \cup \{(x_i, \bar{x}_j) \mid 1 \leq i, j \leq n\} \cup \{(v_i, x_j), (v_i, \bar{x}_j) \mid i \neq j\}$$

Fig. 13.8 Grafo utilizado para mostrar que el problema del número cromático es *NP*-completo: (a) grafo completo con  $n$  vértices; (b)  $x_i$  y  $\bar{x}_i$  están conectados a todo  $v_j$  para los que  $i \neq j$ .

### 13.7 Muestre que los siguientes problemas son *NP*-completos.

- a) Dados un grafo  $G$ , con distancias enteras en las aristas, y dos enteros  $f$  y  $d$ , ¿existe una forma de seleccionar  $f$  vértices de  $G$  en los cuales localizar "estaciones de bomberos", de modo que ningún vértice se encuentre a una distancia mayor que  $d$  de una estación de bomberos?
- \*\*b) *El problema de la generación de código de un registro.* Supóngase que tenemos una computadora con un registro de instrucciones.
- LOAD  $m$  trae el valor a la localidad  $m$  de memoria al registro
  - STORE  $m$  almacena el valor del registro en la localidad  $m$  de memoria
  - OP  $m$  aplica OP, que puede ser cualquier operador binario, con el registro como argumento izquierdo y la localidad  $m$  como argumento derecho; deja el resultado en el registro.
- Dadas una expresión aritmética, cada uno de cuyos operandos representa una localidad de memoria, y una constante  $k$ , ¿existe un programa que evalúe la expresión en  $k$  instrucciones o menos?
- \*\*c) *El problema de la programación en un tiempo de ejecución unitaria.* Dados un conjunto de tareas  $T_1, \dots, T_k$ , un número de procesadores  $p$ , un límite temporal  $t$  y un conjunto de restricciones de la forma  $T_i < T_j$ , que significa que la tarea  $T_i$  debe realizarse antes que la  $T_j$ , ¿existe un *programa*, es decir, una asignación de cuando mucho una tarea a cualquier procesador en cualquier unidad de tiempo, de modo que si  $T_i \leq T_j$  es una restricción, entonces  $T_i$  es asignada una unidad de tiempo antes que  $T_j$ , y dentro de  $t$  unidades de tiempo cada tarea ha sido asignada a un procesador por unidad de tiempo?
- \*\*d) *El problema de la cobertura exacta.* Dados un conjunto  $S$  y un conjunto de subconjuntos  $S_1, S_2, \dots, S_k$  de  $S$ , ¿existe un subconjunto  $T \subseteq \{S_1, S_2, \dots, S_k\}$  de manera que cada  $x$  que esté en  $S$  está en exactamente una  $S_i$  en  $T$ ?

13.8 *El problema del árbol de extensión.* Determinar si un árbol  $T$  es isomórfico con respecto a algún árbol de extensión de  $G$ .

a) Dé una reducción en espacio logarítmico del problema del circuito de Hamilton al problema del árbol de extensión.

\*b) Dé una reducción directa en espacio logarítmico de la satisfactoriedad 3-CNF al problema del árbol de extensión

### 13.9

a) Una *red n dimensional* es un grafo  $G = (V; E)$  en donde

$$V = \{(i_1, i_2, \dots, i_n) \mid 1 \leq i_j \leq m_j, 1 \leq j \leq n\}$$

y  $E = \{v_i, v_j\} \mid v_i, v_j$  difieren en solamente una coordenada, y la diferencia de  $v_i$  y  $v_j$  en esa coordenada es 1]. ¿Para qué valores de  $m_j$  y  $n$  tiene  $G$  un circuito de Hamilton?

\*b) Sea  $G$  un grafo cuyos vértices son los cuadrados de un tablero de ajedrez de  $8 \times 8$  y cuyas aristas son los movimientos legales del caballo. Encuentre un circuito de Hamilton en  $G$ .

\*13.10 Demuestre que el problema circuital de Hamilton es *NP*-completo aun cuando se le restrinja a grafos planos. [Sugerencia: Primero muéstrese que el problema circuital de Hamilton es *NP*-completo para gráficas planas con "restricciones", mediante la reducción de  $L_{3 \text{ sat}}$  a éste. En particular, considérese la clase de los grafos planos con flechas de restricción que conectan ciertos pares de aristas. A las flechas de restricción se les permite cruzarse entre sí, pero no pueden cruzar las aristas del grafo. Muestre que la existencia de circuitos de Hamilton que utilizan exactamente una arista de cada par de aristas restringidas en *NP*-completa. Entonces sustitúyase las flechas de restricción, una por una, por aristas del grafo mediante la sustitución de la Fig. 13.9(a). Durante el proceso, una flecha de restricción puede cruzar una arista, pero solamente si la arista debe encontrarse presente en cualquier circuito de Hamilton. Estos cruzamientos pueden eliminarse mediante la sustitución de la Fig. 13.9(b). La gráfica de la Fig. 13.10 puede ser útil en el primer paso de la sugerencia para representar una cláusula  $x + y + z$ .]

\*13.11 Un grafo es *4-conectado* si al eliminar cualesquiera tres vértices y las aristas incidentes el grafo se queda conectado. Demuestre que el problema circuital de Hamilton es *NP*-completo aun para los grafos 4-conectados. [Sugerencia: Constrúyase un subgrafo con cuatro vértices distinguidos que pueden sustituir a un vértice en un grafo cualquiera  $G$ , de manera que si se le agregan aristas adicionales que van de los cuatro vértices distinguidos a otros vértices de  $G$ , el grafo resultante tendrá un circuito de Hamilton si y sólo si  $G$  lo tiene.]

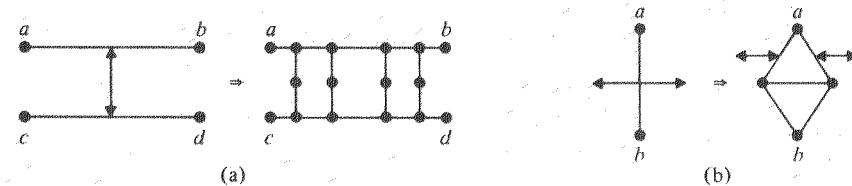


Fig. 13.9 Transformaciones para el Ejercicio 13.10.

\*13.12 Demuestre que el problema que consiste en determinar si un conjunto de ecuaciones lineales  $Ax = b$  tiene una solución con  $k$  componentes de  $x$  igual a cero de *NP*-completo. [Sugerencia: Si los  $x_i$  se restringen a 0 y 1, entonces una desigualdad de la forma  $x_1 + x_2 + x_3 \geq 1$  puede ser sustituida mediante una ecuación de la forma  $y + x_1 + x_2 + x_3 = 4$ , siempre y cuando

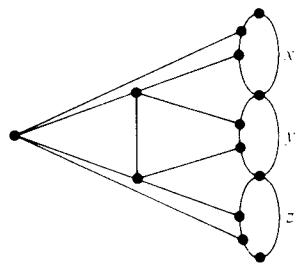


Fig. 13.10 Grafo utilizado en la construcción del Ejercicio 13.10.

y esté restringida a ser 1, 2 o 3. El sistema de ecuaciones  $y + z_1 + z_2 = 3$ ,  $y = z_3 + z_4$  y  $z_i + z_i = 1$ ,  $1 \leq i \leq 4$ , no tiene solución con más de cuatro variables cero y tiene una solución con exactamente cuatro variables cero si y sólo si  $y = 1, 2$ , o 3.]

\*13.13 Un *kernel*<sup>†</sup> de un grafo dirigido es un conjunto de vértices tales que

1. no existe un arco que va de un vértice del kernel a otro vértice del kernel y
2. cada vértice se encuentra en el kernel o tiene un arco que va de éste al kernel.

Demuestre la determinación de si un grafo dirigido tiene un kernel en *NP*-completo. [Sugerencia: Obsérvese que un ciclo de longitud dos o tres puede tener solamente un vértice en un kernel.]

13.14 Demuestre que el problema del agente viajero es *NP*-completo.

\*\*13.15 Considere aproximaciones al problema del agente viajero. Muestre que la existencia de un algoritmo en tiempo polinomial que produce un recorrido con el doble del costo del recorrido óptimo implicaría que  $\mathcal{P} = \mathcal{NP}$ .

\*13.16 Considere el problema del agente viajero en el que las distancias satisfacen la desigualdad del triángulo, esto es

$$d(v_1, v_3) \leq d(v_1, v_2) + d(v_2, v_3).$$

Dé un algoritmo de tiempo polinomial para encontrar un recorrido que se halle dentro del doble del costo del recorrido óptimo.

\*13.17 Supóngase que existe un algoritmo en tiempo polinomial para encontrar un grupúsculo en un grafo que, al menos, tenga la mitad del tamaño del grupúsculo máximo.

- a) Demuestre que existirá un algoritmo en tiempo polinomial para encontrar un grupúsculo que sea, de al menos,  $1/\sqrt{2}$  veces el tamaño del grupúsculo máximo. [Sugerencia: Considérese la posibilidad de sustituir cada vértice de un grafo por una copia del grafo.]
- b) Demuestre que, para cualquier  $k < 1$ , existirá un algoritmo en tiempo polinomial para encontrar un grupúsculo cuyo tamaño es de, al menos,  $k$  veces el tamaño del grupúsculo máximo.

\*13.18 Demuestre que es *NP*-completo determinar si un número cromático de una gráfica es

<sup>†</sup> Kernel: núcleo, en alemán (N. del T.)

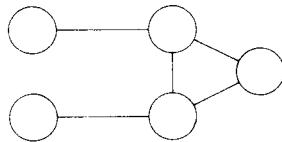


Fig. 13.11 Grafo utilizado en el Ejercicio 13.18.

menor o igual a 3. [Sugerencia: La gráfica de la Fig. 13.11 puede ser utilizada como una forma débil de una compuerta OR cuando sólo se tienen disponibles tres colores, en el mismo sentido en que la salida puede colorearse como "verdadera" si y sólo si, al menos, una entrada tiene color "verdadero".]

\*\*13.19 Para  $n \geq 6$ , sea  $G_n = (V_n, E_n)$  el grafo en donde

$$\begin{aligned} V_n &= \{(i, j, k) \mid i, j, k \text{ son elementos diferentes de } \{1, 2, \dots, n\}\}, \\ E_n &= \{(u, v) \mid u \text{ y } v \text{ son tripletes disjuntos}\}. \end{aligned}$$

a) Sea  $X_m(G)$  el número mínimo de colores que se necesitan para asignar  $m$  colores distintos a cada vértice de  $G$ , de manera que no hay dos vértices adyacentes con un mismo color. Demuestre, para  $n \geq 6$ , que  $X_3(G_n) = n$  y  $X_4(G_n) = 2n - 4$ .

b) Supóngase que existe un algoritmo en tiempo polinomial para colorear un grafo  $G$  con, cuando más, dos veces el mínimo del número de colores que se necesita. Entonces demuestre que  $\mathcal{P} = \mathcal{NP}$ . [Sugerencia: Combínese la parte (a) con el Ejercicio 13.18.]

\*\*13.20 Construya un algoritmo para encontrar un circuito de Hamilton en un grafo que, bajo la suposición de que  $\mathcal{P} = \mathcal{NP}$ , encontrará un circuito de Hamilton en un tiempo polinomial siempre y cuando dicho circuito exista. Si no existe circuito de Hamilton, el algoritmo no necesita correr en tiempo polinomial. Nótese que no es suficiente diseñar un algoritmo no determinístico y después utilizar la hipótesis  $\mathcal{P} = \mathcal{NP}$  para exigir que existe un algoritmo determinístico en tiempo polinomial. En realidad se debe mostrar el algoritmo que potencialmente está en tiempo polinomial.

\*\*13.21 Si  $\mathcal{P} \neq \mathcal{NP}$ , demuestre que es irresoluble para  $L$  en  $\mathcal{NP}$  decidir si  $L$  está en  $\mathcal{P}$ .

\*\*13.22 Demuestre que la existencia de un subconjunto *NP*-completo de  $0^*$  implica que  $\mathcal{P} = \mathcal{NP}$ .

\*13.23 Un entero  $n$  es compuesto si y sólo si existe una  $a$ ,  $1 \leq a \leq n$ , tal que se cumpla una de las condiciones siguientes:

1.  $a^{n-1} \neq 1 \pmod{n}$ , o
2. existen enteros  $b$  e  $i$ , con  $n-1 = 2^i b$ , y  $a^b$  y  $n$  tienen un divisor común.

Si  $n$  es compuesto, al menos una mitad de los enteros que se encuentran entre 1 y  $n$  satisfacen (1) o (2). Dé un algoritmo aleatorizado que, con una probabilidad alta, determine, en tiempo polinomial, si un número es primo.

\* 13.24 Supóngase que existe una función  $f$  que transforma a los enteros de longitud  $k$  en los enteros de longitud  $k$ , tal que;

1.  $f$  es calculable en tiempo polinomial;  
 2.  $f^{-1}$  no es calculable en tiempo polinomial.

Demuestre que lo anterior implicaría que

$$A = \{(x, y) \mid f^{-1}(x) < y\} \text{ es en } (\mathcal{NP} \cap \text{Co-}\mathcal{NP}) - \mathcal{P}$$

### 13.25 Muestre que los problemas siguientes son ESPACIOP-completos.

- a) ¿Una expresión regular dada (que sólo tiene los operadores usuales  $\cdot$ ,  $+$ ,  $\ast$ ) define a todas las cadenas sobre su alfabeto? [Sugerencia: La demostración sigue pasos paralelos a la del Teorema 13.14.]

\*\*Sb) *El juego de comutación de Shannon.* Dado un grafo  $G$  con dos vértices distinguidos  $s$  y  $t$ , supóngase que existen dos jugadores PEQUEÑO y CORTO. De manera alternada, con PEQUEÑO primero, los jugadores seleccionan vértices de  $G$  diferentes de  $s$  y  $t$ . PEQUEÑO gana si selecciona vértices que, junto con  $s$  y  $t$ , forman una trayectoria que va de  $s$  a  $t$ . CORTO gana si PEQUEÑO no puede hacer dicha trayectoria. ¿Puede PEQUEÑO forzar su victoria sobre  $G$  sin importar lo que CORTO haga?

\*\*13.26 Muestre que si  $\text{ESPACIOP} \neq \mathcal{P}$ , entonces existe una demostración mediante diagonalización. Es decir, existe una enumeración  $L_1, L_2, \dots$  de  $\mathcal{P}$ , y una función calculable  $f$  de los enteros a las cadenas y un conjunto  $L$  que está en  $\text{ESPACIOP}$  tal que, para cada  $i$ ,  $f(i)$  se encuentra en  $L$  si y sólo si  $f(i)$  no se encuentra en  $L_i$ .

13.27 Dé un algoritmo en tiempo polinomial para convertir una fórmula booleana cuantificada a la forma normal prenex  $Q_1 X_1 Q_2 X_2 \cdots Q_k X_k (E)$ , en la que  $E$  es una expresión booleana en 3-CNF.

\*13.28 ¿Se puede convertir a cualquier QBF en tiempo polinomial a una fórmula equivalente con, cuando mucho, diez variables diferentes?

13.29 Muestre que los problemas siguientes son completos para  $\mathcal{P}$  con respecto a las reducciones en espacio logarítmico.

- a) ¿Está  $x$  en  $L(G)$ , para la cadena  $x$  y la CFG  $G$ ?

\*\*b) *El problema del valor de circuito.* Codifique un circuito como una sucesión  $C_1, C_2, \dots, C_n$ , en la que cada  $C_i$  es una variable  $x_1, x_2, \dots, 0 \wedge (j, k) o \neg(j)$ , con  $j$  y  $k$  menores que  $i$ . Dada una codificación de un circuito y una asignación de verdadero o falso a las variables, ¿es la salida del circuito verdadera?

\*\*13.30 Muestre que los problemas siguientes son completos para  $\text{ESPACION}(\log n)$  con respecto a las reducciones en espacio logarítmico.

- a) ¿Es una expresión booleana en 2-CNF no satisfaciente?  
 b) ¿Es un grafo dirigido fuertemente conectado?  
 c) ¿Es  $L(G)$  infinito para la CFG  $G$  sin producciones  $\epsilon$ o que no tenga terminales inútiles?

\*13.31 Dadas las CFGs,  $G_1$ , y  $G_2$ , y el entero  $k$ , muestre que el problema que consiste en determinar si existen palabras  $w_1$  en  $L(G_1)$ , y  $w_2$  en  $L(G_2)$  que coincidan en los primeros símbolos  $k$ , es completo para un tiempo determinístico exponencial con respecto a las reducciones en tiempo polinomial.

\*\*13.32 Muestre que el problema que consiste en determinar si una expresión regular, en la

cual está permitido el operador intersección, representa a todas las cadenas en su alfabeto, requiere un tiempo  $2^{c\sqrt{n}}$  i.o., para alguna  $c > 0$ , y que puede resolverse en un tiempo  $2^d n$ .

### 13.33

- a) Escriba una fórmula de la teoría de los enteros bajo la adición, que exprese que cada entero mayor que 5 es la suma de tres enteros positivos diferentes.  
 b) Escriba una fórmula en la teoría de los números que exprese que  $d$  es el máximo común divisor de  $a$  y  $b$ .  
 \*\*c) Escriba una fórmula de la teoría de los números que exprese que  $z = x^y$ .

13.34 Aplique el procedimiento de decisión de la Sección 13.6 para la teoría de los números reales, para decidir si la fórmula

$$\exists y \exists x [(x + y = 14) \wedge (3x + y = 5)] \text{ es verdadera}$$

### \*\*13.35

- a) Muestre que la teoría de la aritmética de Presburger (los enteros con  $+, =, <$ ) requiere un tiempo no determinístico  $2^{2^{\Theta(n)}}$  i.o., para alguna  $c > 0$ . [Sugerencia: Desarrollense las siguientes fórmulas de tamaño proporcional a  $n$ :

1.  $R_n(x, y, z) : 0 \leq y < 2^n$ , y  $z$  es el residuo de  $x$  mod  $y$ .
2.  $P_n(x) : 0 \leq x < 2^n$ , y  $x$  es un número primo.
3.  $G_n(x) : x$  es el menor entero divisible entre todos los primos menores que  $2^n$ .
4.  $M_n(x, y, z) : x, y$  y  $z$  son enteros que se encuentran en el intervalo comprendido entre  $0$  y  $2^{2^n} - 1$ , y  $xy = z$ .]

- b) Muestre que la aritmética de Presburger puede ser decidida en espacio  $2^{2^{\Theta(n)}}$  y tiempo  $2^{2^{\Theta(n)}}$ .  
 c) Utilice el algoritmo de la parte (b) para decidir

$$\exists y \exists x [(x + y = 14) \wedge (3x + y = 5)].$$

\*\*13.36 Extienda la aritmética de Presburger para permitir la cuantificación sobre arreglos de enteros. Por tanto podemos escribir fórmulas tales como

$$\forall A \forall n \exists B \forall i [\neg(1 \leq i \leq n) \vee [\exists j (1 \leq j \leq n) \wedge A(i) = B(j)]].$$

Demuestre que la teoría de la aritmética de Presburger con arreglos es irresoluble.

\*\*13.37 Para demostrar que la teoría de los números es irresoluble, resulta conveniente codificar una sucesión de longitud  $n + 1$ ,  $x_0, x_1, \dots, x_n$ , en un entero  $x$  tal que cada  $x_i$  puede obtenerse a partir de  $x$  mediante una fórmula.

- a) Sea  $m = \max \{n, x_0, x_1, \dots, x_n\}$ . Demuestre que el conjunto de las  $u_i = 1 + (i + 1)m!$ ,  $0 \leq i \leq n$ , son primos relativos por parejas y que  $u_i > x_i$ . Esto implica que existe un entero  $b < u_0 u_1 \cdots u_n$  tal que  $b = x_i \text{ mod } u_i$ ,  $0 \leq i \leq n$ .  
 b) Exprese la función  $\beta$  de Gödel

$$\beta(b, c, i) = b \text{ mod } [1 + (i + 1)c]$$

como un predicado.

- c) Demuestre que la teoría de los números es irresoluble.

**\*\*13.38** Muestre que existen oráculos  $C, D$  y  $E$ , los cuales

- a)  $\mathcal{P}^C = \mathcal{N}\mathcal{P}^C$  y  $\text{co-}\mathcal{N}\mathcal{P}^C$  son todos distintos.
- b)  $\mathcal{P}^D \neq \mathcal{N}\mathcal{P}^D$  pero  $\mathcal{N}\mathcal{P}^D = \text{Co-}\mathcal{N}\mathcal{P}^D$ .
- c)  $\mathcal{P}^E = \mathcal{N}\mathcal{P}^E$  es independiente de los axiomas de la teoría de los números.

**\*13.39** Muestre que  $\mathcal{P} = \mathcal{N}\mathcal{P}$  si y sólo si  $\mathcal{P}$  es un AFL.

### Soluciones a los ejercicios seleccionados

**13.16** Constrúyase un árbol de extensión de costo mínimo mediante el ordenamiento de las aristas según su costo creciente, seleccionando las aristas de modo que se comience con la de menor costo, y descártense cualquier arista que forme un ciclo. Sea  $T_{\text{opt}}$  el costo mínimo de un circuito de Hamilton, y sea  $T_s$  el costo del árbol de extensión de mínimo costo. Es claro que  $T_s \leq T_{\text{opt}}$ , ya que un árbol de extensión puede obtenerse a partir de un circuito de Hamilton mediante la eliminación de una arista. Constrúyase una trayectoria que pase por todos los vértices del grafo y que atraviesa todo el árbol de extensión. La trayectoria no constituye un circuito de Hamilton, ya que cada arista del árbol de extensión es atravesada dos veces. El costo de esta trayectoria es de cuando mucho  $2T_s \leq 2T_{\text{opt}}$ . Recórrase la trayectoria hasta encontrar alguna arista  $e_1$  que lleve a un vértice por segunda ocasión. Sea  $e_2$  la arista que sigue inmediatamente a  $e_1$  sobre la trayectoria. Sustitúyase la parte de la trayectoria que consiste en  $e_1$  y  $e_2$  mediante una sola arista. Según la desigualdad del triángulo, esto puede aumentar el costo. Repítase el proceso de sustitución de parejas de aristas por una sola hasta que se obtengan circuitos de Hamilton.

**13.25b** En primer lugar mostraremos que el juego de conmutación de Shannon está en ESPACIOP. Considérese un árbol de juegos. La raíz indica la posición inicial del juego. Supóngase que PEQUEÑO es el primero en mover. Los hijos de la raíz corresponden a cada posición del juego posible después de un movimiento de PEQUEÑO. En general, un vértice del árbol corresponde a los movimientos que se han hecho hasta entonces (los cuales determinan la posición del juego) y los hijos del vértice corresponden a la posición del tablero después de cada movimiento adicional posible.

Una posición es ganadora si PEQUEÑO llega a la victoria desde esa posición. Por consiguiente una hoja es una posición ganadora si PEQUEÑO tiene una trayectoria de  $s$  a  $t$ . De manera recursiva podemos definir posiciones ganadoras de la forma siguiente. Si el vértice  $v$  no es una hoja y corresponde a una posición en la cual se lleva a cabo el movimiento de PEQUEÑO, entonces  $v$  es una posición ganadora si existe un hijo que, a su vez, sea una posición ganadora. Si le toca mover a CORTO, entonces  $v$  es una posición ganadora sólo si cada hijo es una posición ganadora. Puesto que la profundidad del árbol es de cuando mucho  $n$ , el número de vértices de  $G$ , un algoritmo recursivo para determinar si la raíz es una posición ganadora requiere un espacio de cuando mucho  $n$ . Por consiguiente el problema se encuentra en ESPACIOP.

Para mostrar que el juego de conmutación de Shannon es ESPACIOP-completo, reducimos el problema de la fórmula booleana cuantificada a éste. Considérese una fórmula booleana cuantificada y, sin pérdida de generalidad, supóngase que los cuantificadores se alternan (de otra manera añádase cuantificadores y variables mudos).

$$\exists x_1 \forall x_2 \exists x_3 \cdots \forall x_{n-1} \exists x_n F(x_1 \cdots x_n).$$

Considérese el grafo conocido como escalera, que se muestra en la Fig. 13.12, en la que  $n = 3$ . Habrá aristas adicionales (veáñse las líneas punteadas) pero no son de importancia para la primera observación. PEQUEÑO juega primero. En algún momento debe escoger entre  $\bar{x}_2(1)$

y  $x_1(1)$ . Esto corresponde a que PEQUEÑO seleccione un valor para la variable existencialmente cuantificada  $\bar{x}_1$ . Los siguientes cuatro movimientos son forzados, y terminan con que PEQUEÑO ha seleccionado  $x_1(1), x_1(2)$  y  $\exists x_1$ , y CORTO ha seleccionado a  $\bar{x}_1(1)$  y  $\bar{x}_1(2)$  o PEQUEÑO ha seleccionado  $x_1(1), x_1(2)$  y  $\exists x_1$ , y CORTO ha escogido  $x_1(1)$  y  $x_2(2)$ . Si PEQUEÑO no escoge alguna de las  $x_1(1), \bar{x}_1(1), x_1(2), \bar{x}_1(2)$  o  $\exists x_1$ , entonces CORTO gana. Si PEQUEÑO escoge  $\exists x_1$ , entonces se le ha dado a CORTO la ventaja al escoger  $x_1(1)$  o  $\bar{x}_1(1)$ . El propósito del vértice  $\exists x_1$  es consumir un movimiento adicional de PEQUEÑO, permitiendo así a CORTO la primera selección del conjunto  $\{x_2(1), \bar{x}_2(1), x_2(2), \bar{x}_2(2)\}$ . Lo anterior significa que CORTO selecciona el valor de la variable universalmente cuantificada  $x_2$ , y así sucesivamente.

Una vez que se han escogido los valores para  $x_1, x_2, \dots, x_n$ , la parte punteada del grafo, que corresponde a la parte que no tiene cuantificadores de las fórmulas, entra en el juego. Sin que se pierda generalidad, podemos suponer que  $F(x_1, \dots, x_n)$  se encuentra en la forma normal conjuntiva. Sea  $F = F_1 \wedge F_2 \wedge \cdots \wedge F_n$ , en donde cada  $F_i$  es una cláusula. Constrúyase el árbol de la Fig. 13.13. Identifíquese la raíz 1 con el vértice  $\exists x_1$  de la Fig. 13.12. A partir del vértice  $F_i$  añádase una arista hacia el vértice  $x_j(1)$  o al  $\bar{x}_j(1)$  en caso de que  $x_j$  o  $\bar{x}_j$ , respectivamente, aparezca en  $F_i$ . Obsérvese ahora que PEQUEÑO selecciona el vértice 1. CORTO puede escoger  $F_1$  o 2, y PEQUEÑO escoge el otro. Resulta claro que PEQUEÑO puede construir una trayectoria que vaya, al menos, a un  $F_i$ , y CORTO puede forzar a PEQUEÑO a alcanzar solamente un  $F_i$ , y puede determinar cuál  $F_i$ . Ahora PEQUEÑO tiene una trayectoria de  $s$  a  $t$  si  $F_i$  está conectado con algún  $x_j(1)$  o  $\bar{x}_j(1)$  que “tiene valor 1”; es decir, PEQUEÑO ha escogido  $x_j(1)$  o  $\bar{x}_j(1)$ .

Obsérvese que si la fórmula cuantificada es verdadera, entonces PEQUEÑO puede

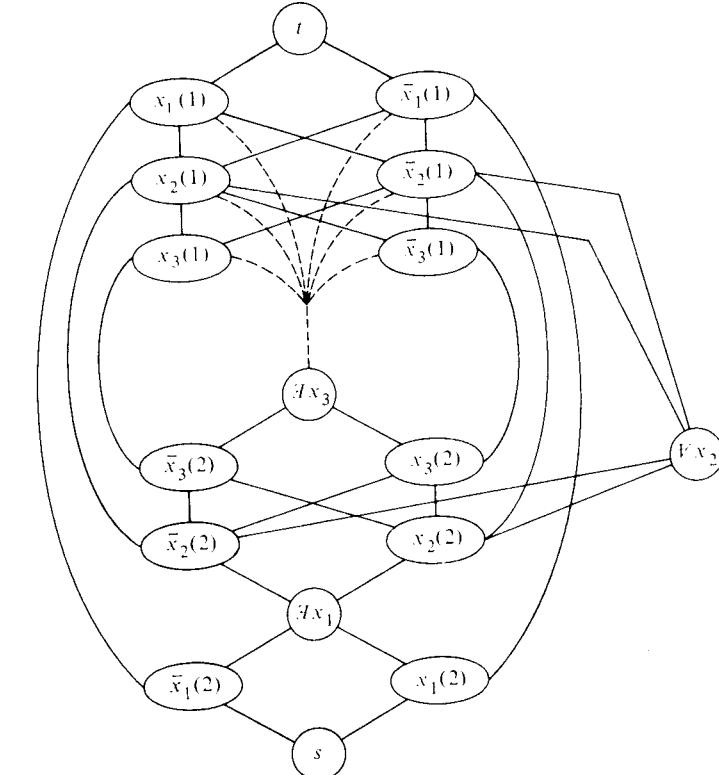


Fig. 13.12 Una escalera para el juego de conmutación de Shannon.

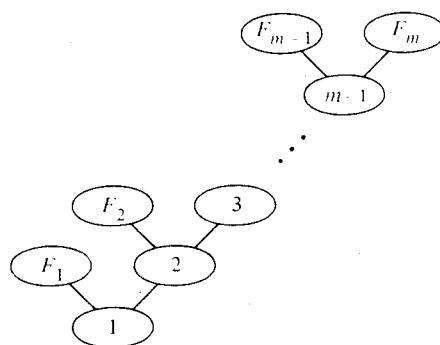


Fig. 13.13 El árbol para la fórmula  $F = F_1 F_2 \cdots F_m$ .

especificar las variables existencialmente cuantificadas, de modo que independientemente de las selecciones de CORTO de las variables universalmente cuantificadas,  $F$  es verdadera. Por consiguiente, sin importar qué  $F_i$  se vio forzado a escoger PEQUEÑO, ese  $F_i$  es verdadero y por tanto está conectado a un  $x_i$  o  $\bar{x}_i$  seleccionado. En consecuencia PEQUEÑO puede ganar.

Por otro lado, si la fórmula booleana cuantificada es falsa, CORTO puede escoger las variables universalmente cuantificadas de modo que, para la asignación a las  $x$ s,  $F$  es falsa. Entonces CORTO fuerza a PEQUEÑO a alcanzar solamente un  $F_i$ , y en particular ese  $F_i$  es falso para la asignación. Por consiguiente PEQUEÑO no completa la trayectoria y CORTO gana. En consecuencia podemos garantizar que PEQUEÑO ganará si y sólo si la fórmula booleana cuantificada es verdadera y, por tanto, el juego de conmutación de Shannon es completo para el ESPACIOP.

#### NOTAS BIBLIOGRAFICAS

Cobham [1964] fue el primero que prestó atención a la clase  $\mathcal{P}$ . El primer problema  $NP$ -completo, incluyendo las versiones de los problemas de satisfactoriedad de los Teoremas 13.1, 13.2 y 13.3, fueron introducidos por Cook [1971b]. Karp [1972] dio una amplia variedad de problemas  $NP$ -completos y demostró claramente la importancia de esta idea. Algunos de estos problemas incluyen el de cobertura de vértice (Teorema 13.4), el de cobertura de grupúsculo (Ejercicio 13.4), el de la cobertura exacta (Ejercicio 13.7d), el de número cromático (Ejercicio 13.6), el problema circuital de Hamilton (Teorema 13.6) y los problemas del agente viajero y de partición que se mencionaron en la Sección 13.2. El problema del grupúsculo cúmulo (Ejercicio 13.3) se tomó de Cook [1971]. El Teorema 13.7, la integridad  $NP$  de la programación lineal entera, fue desarrollado independientemente por Gathen y Sieveking [1976] y Borosh y Treybig [1976]. La demostración que dimos tomada de Kannan y Monma [1978].

Se ha demostrado que un gran número de problemas son  $NP$ -completos, y dichos problemas vienen de una amplia variedad de áreas. Garey y Johnson [1978] hacen un intento por catalogarlos; mencionaremos aquí sólo una muestra del trabajo que se ha hecho y las áreas que han sido cubiertas. Sethi [1975], y Bruno y Sethi [1976] cubren los problemas relacionados con

la generación de códigos de cobertura (el Ejercicio 13.7b aparece en el último trabajo). Los problemas sobre programas e itinerarios se consideran en Coffman [1976] y Ullman [1975]; la solución al Ejercicio 13.7(c) se encuentra en ambos. Garey, Johnson y Stockmeyer [1976] y Garey, Graham y Johnson [1976] proporcionan una gran variedad de resultados poderosos, principalmente para los problemas de grafos. Papadimitriou [1976] y Papadimitriou y Steiglitz [1977] estudiaron los problemas sobre las trayectorias en grafos. El ejercicio 13.18 se tomó de Stockmeyer [1973], el Ejercicio 13.10 de Garey, Johnson y Tarjan [1976], y el Ejercicio 13.12 se debe a J. E. Hopcroft.

Un cierto número de resultados que muestran clases grandes de problemas  $NP$ -completos aparecen en Hunt y Szymanski [1976], Hunt y Rosenkrantz [1977], Kirkpatrick y Hell [1978], Lewis [1978], Schaefer [1978] y Yannakakis [1978].

Entre los planteamientos prometedores para tratar problemas  $NP$ -completos encontramos la idea de tomar en consideración *algoritmos aproximados* para las versiones de optimización de los problemas. Tales algoritmos corren en tiempo polinomial, pero se tiene la garantía de que terminarán solamente dentro de algún intervalo específico de lo óptimo. Johnson [1974] consideró algoritmos de aproximación para algunos de los problemas  $NP$ -completos que aparecen en Karp [1972]. Sahni y Gonzalez [1976] fueron los primeros en demostrar que la aproximación a un problema  $NP$ -completo es ella misma  $NP$ -completa (Ejercicio 13.15), mientras que Garey y Johnson [1976] mostraron que terminar con un factor menor de dos del número cromático de un grafo (número de "colores" que se necesitan para asegurar que cada vértice estará coloreado con un color diferente al de los vértices adyacentes) en  $NP$ -completo (Ejercicio 13.19). El Ejercicio 13.17, sobre el mejoramiento de una aproximación a un grupúsculo máximo, se tomó también de Garey y Johnson [1976]. Rosenkrantz, Stearns y Lewis [1977] estudiaron aproximaciones al problema del agente viajero (Ejercicio 13.16). Christofides [1976] ha mejorado sus resultados.

Un cierto número de artículos han intentado explorar la estructura de  $\mathcal{NP}$  sobre la hipótesis de que  $\mathcal{P} \neq \mathcal{NP}$ . Ladner [1975a] muestra, por ejemplo, que si  $\mathcal{P} \neq \mathcal{NP}$ , entonces existen problemas que no están en  $\mathcal{P}$  ni son  $NP$ -completos. Adleman y Manders [1977] muestran que ciertos problemas tienen la propiedad de que se encuentran en  $\mathcal{P}$  si y sólo si  $\mathcal{NP} = \text{co-}\mathcal{NP}$ . Book [1974, 1976] muestra la desigualdad entre ciertas clases de complejidad, como TIEMPO( $n^k$ ) o ESPACIOD( $\log^k n$ ). El Ejercicio 13.39, que relaciona  $\mathcal{P} = \mathcal{NP}$  con la teoría de los AFLs, se tomó de Book [1970]. Berman y Hartmanis [1977] enfocan su atención a las reducciones, que conservan la densidad, de un problema a otro. El Ejercicio 13.22 se tomó de Berman [1978] y el Ejercicio 13.20 de Levin [1973].

Se le ha dado una particular atención a la complejidad de reconocimiento de números primos. Es fácil demostrar que los no primos (escritos en binario) se encuentran en  $\mathcal{NP}$ , pero no se sabía que los números primos se encontraban en  $\mathcal{NP}$  hasta Pratt [1975]. Por consiguiente, si el reconocimiento de primos es en  $NP$ -completo, entonces, según el Teorema 13.8,  $\text{co-}\mathcal{NP} = \mathcal{NP}$ . Miller [1976] da una fuerte evidencia de que el reconocimiento de números primos escritos en binario se encuentran en  $\mathcal{P}$ . El Ejercicio 13.23, que muestra una prueba eficiente para determinar la primalidad con una alta probabilidad se tomó de Rabin [1977]. Un resultado similar se encuentra en Solovay y Strassen [1977]. El Ejercicio 13.24 es de Brassard, Fortune y Hopcroft [1978].

El primer problema ESPACIOP-completo fue introducido por Karp [1972], incluyendo el reconocimiento de CSL (Teorema 13.11) y la " $=\Sigma^*$ " para expresiones regulares (Ejercicio 13.25a). La integridad ESPACIOP de las fórmulas booleanas cuantificadas fue demostrada por Stockmeyer [1974]. El Ejercicio 13.25(b), integridad ESPACIOP para el juego de conmutación de Shannon, se debe a Even y Tarjan [1976]. Stockmeyer [1978] da una jerarquía de los problemas entre  $\mathcal{NP}$  y ESPACIOP, bajo la suposición de que  $\mathcal{NP} \neq \text{ESPACIOP}$ .

Los problemas completos para  $\mathcal{P}$  con respecto a las reducciones en espacio logarítmico fueron considerados por Cook [1973b], Cook y Sethi [1976], Jones [1975], Jones y Laaser [1976] (incluyendo el Teorema 13.12) y Ladner [1975b] (Ejercicio 13.29b). Los problemas completos para ESPACIOP ( $\log n$ ) con respecto a las reducciones en espacio logarítmico se consideran en Savitch [1970], incluyendo el Teorema 13.13 (sobre la alcanzabilidad), Sudborough [1975a, b], Springsteel [1976] y Jones, Lien y Laaser [1976]. El Ejercicio 13.30 se tomó de Jones, Lien y Laaser [1976].

El primer problema que se demostró requería tiempo exponencial (de hecho, espacio exponencial) fue presentado por Meyer y Stockmeyer [1973]. En esencia, el problema es parecido al del Teorema 13.15. Los límites inferiores de la complejidad de la teoría de los números reales consuma (Teorema 13.16) y de la aritmética de Presburger (Ejercicio 13.35) se tomaron de Fischer y Rabin [1974]. Los límites superiores de estos problemas son de Cooper [1972], Ferrante y Rackoff [1975] y Oppen [1973]. Berman [1977] y Bruss y Meyer [1978] establecen lo que, en un cierto sentido, constituyen límites más precisos (fuera de las jerarquías espacio-tiempo comunes) para estos problemas. La irresolubilidad de la aritmética de Presburger con arreglos se tomó de Suzuki y Jefferson [1977].

La literatura contiene un cierto número de artículos que tratan sobre la complejidad de una variedad de problemas y sus casos especiales, dividiendo los problemas en grupos, principalmente: polinomiales,  $NP$ -completos, ESPACIOP-completos y, demostrablemente exponenciales. Una muestra de las áreas cubiertas incluyen las ecuaciones de Diophantine en el trabajo de Adleman y Manders [1976], computación asíncrona en Cardoza, Lipton y Meyer [1976], los problemas acerca de las expresiones regulares en Hunt [1975] (incluyendo el Ejercicio 13.32), Hunt, Rosenkrantz y Szymanski [1976] y Stockmeyer y Meyer [1973], los problemas acerca de las gramáticas libres de contexto en Hunt y Rosenkrantz [1974, 1977], Hunt y Szymanski [1975, 1976] y Hunt, Szymanski y Ullman [1975] (incluyendo el Ejercicio 13.31), y la teoría de juegos; en Schaefer [1976].

Los resultados de la Sección 13.7 y el Ejercicio 13.38, sobre la cuestión  $\mathcal{P} = NP$  cuando se tienen oráculos, se tomaron de Baker, Gill y Solovay [1975]. Sin embargo, Kozen [1978] presenta otro punto de vista en la materia; el Ejercicio 13.26 se tomó de ese trabajo. Ladner, Lynch y Selman [1974] estudiaron las diferentes clases de reducibilidad limitada, como la de muchos-a-uno, la de Turing y las tablas verdad. Otro tratamiento de la cuestión  $\mathcal{P} = NP$  ha sido el desarrollo de modelos cuyas versiones con límite temporal determinísticas no determinísticas son equivalentes. Las máquinas vectoriales (Pratt y Stockmeyer [1976]) son las primeras, y otros modelos han sido propuestos por Chandra y Stockmeyer [1976] y Kozen [1976]. El lector deberá notar también la equivalencia para las versiones con límite espacial de los "PDAs auxiliares" que se discutirán en la Sección 14.1.

## CAPITULO

## 14

CARACTERISTICAS PRINCIPALES  
DE OTRAS CLASES  
DE LENGUAJES

En la literatura se ha introducido un gran número de modelos y clases de lenguajes. Este capítulo presenta algunos de éstos que parecen ser del mayor interés.

En la Sección 14.1 se discuten los autómatas de pila auxiliares, que son PDAs con entrada de dos direcciones y un almacenamiento adicional para objetivos generales, el cual se encuentra en forma de una cinta de Turing con límite espacial. La propiedad destacada de los autómatas de presión determinísticos consiste en que, para una cantidad fija de almacenamiento extra, las versiones determinística y no determinística son equivalentes en poder de reconocimiento de lenguajes, y la clase de lenguajes aceptadas por los PDAs auxiliares con un límite espacial dado es equivalente a la clase de lenguajes aceptados por las máquinas de Turing de complejidad temporal exponencial en el mismo límite espacial.

La Sección 14.2 trata sobre los autómatas de pila que son PDAs que poseen el privilegio de poder barrer la pila por debajo del símbolo del tope pero solamente en el modo de sólo lectura. Los lenguajes aceptados por las variantes del autómata de pila de dos direcciones resultan ser clases de complejidad temporal o espacial.

La Sección 14.3 se dedica a los lenguajes indexados, ya que éstos surgen en un cierto número de contextos y parecen ser una generalización natural de los CFLs. Por último, la Sección 14.4 introduce los sistemas de desarrollo, que intentan hacer un modelo de ciertos patrones biológicos de crecimiento.

## 14.1 AUTOMATAS DE PRESION AUXILIARES

En la Fig. 14.1 se esquematiza un *autómata de presión auxiliar*  $S(n)$  (APDA). Este consiste en:

1. una cinta de entrada de sólo lectura, precedida y seguida por los señaladores de extremos  $\phi$  y  $\$$ ,
2. un control finito de estados,

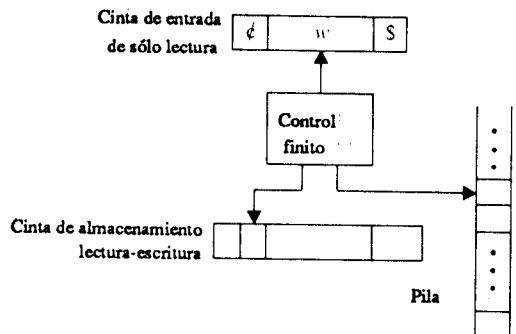


Fig. 14.1 PDA Auxiliar.

3. una cinta de almacenamiento de lectura–escritura de longitud  $S(n)$ , en la que  $n$  es la longitud de la cadena de entrada  $w$ , y
4. una pila.

Un movimiento del APDA está determinado por el estado del control finito, junto con los símbolos barridos por las cabezas de entrada, almacenamiento y pila. En un movimiento, el APDA puede hacer cualquiera de las acciones siguientes, o todas ellas:

1. cambio de estado,
2. mueve su cabeza de entrada una posición hacia la izquierda o la derecha, pero no fuera de la entrada,
3. imprime un símbolo sobre la celda barrida por la cabeza de almacenamiento y mueve dicha cabeza una posición hacia la izquierda o la derecha,
4. coloca un símbolo sobre la pila o elimina el símbolo del tope de la pila.

Si el dispositivo es no determinístico, tiene un número finito de alternativas del tipo descrito más arriba. Al principio, las cabezas de cinta se encuentran en el extremo izquierdo de las cintas de entrada y almacenamiento, con el control finito en un estado inicial designado, y la pila consiste en un símbolo inicial designado. La aceptación se da por agotamiento de pila.

### Equivalencia de los APDAs determinísticos y no determinísticos

El interés que se tiene en los APDAs se originó a partir del descubrimiento de que los APDAs determinísticos y no determinísticos con el mismo límite espacial son equivalentes, y de que un espacio  $S(n)$  sobre un APDA es equivalente a un tiempo  $c^{S(n)}$  sobre una máquina de Turing. Esto es, las siguientes tres proposiciones son equivalentes.

1.  $L$  es aceptado por un APDA determinístico con límite espacial  $S(n)$ .
2.  $L$  es aceptado por un APDA no determinístico con límite espacial  $S(n)$ .
3.  $L$  se encuentran en TIEMPOD( $c^{S(n)}$ ), para alguna constante  $c$ .

Estos hechos se satisfacen en la siguiente serie de lemas.

**Lema 14.1** Si  $L$  es aceptado por un APDA no determinístico con límite espacial  $S(n)$ ,  $A$ , en el que  $S(n) \geq \log n$ , entonces  $L$  se encuentra en TIEMPOD( $c^{S(n)}$ ). para alguna constante  $c$ .

**Demostración** Hagamos que  $A$  tenga  $s$  estados,  $t$  símbolos de almacenamiento y  $p$  símbolos de pila. Dada una entrada de longitud  $n$ , existen  $n + 2$  posiciones de la cabeza de entrada,  $s$  estados,  $S(n)$  posiciones de la cabeza de almacenamiento y  $t^{S(n)}$  contenidos de cinta de almacenamiento posibles, para hacer un total de

$$\hat{s}(n) = (n + 2)sS(n)t^{S(n)}$$

configuraciones posibles.<sup>†</sup> Como  $S(n) \geq \log n$ , existe una constante  $d$  tal que  $\hat{s}(n) \leq d^{S(n)}$ , para toda  $n \geq 1$ .

Constrúyase una TM  $M$ , que efectúe las operaciones siguientes sobre la entrada  $w$  de longitud  $n$

1.  $M$  construye un PDA  $P_w$  que, sobre la entrada  $\epsilon$ , simula todos los movimientos de  $A$  sobre la entrada  $w$ .
2.  $M$  convierte a  $P_w$  a un CFG  $G_w$ , mediante el algoritmo del Teorema 5.4.

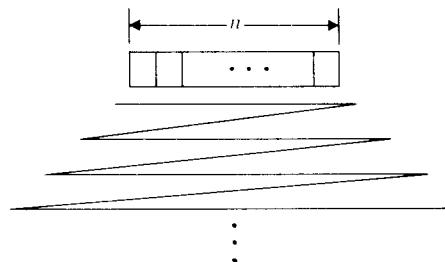
Para  $A$  fijo,  $P_w$  es un PDA para cada  $w$ , en el cual el estado y contenido de las cintas de entrada y almacenamiento de  $A$  están codificados en el estado de  $P_w$ .  $N(P_w)$  es  $\{\epsilon\}$  o  $\emptyset$ , dependiendo de si  $A$  acepta a  $w$  o no.

$P_w$  tiene cuando más  $\hat{s}(n) \leq d^{S(n)}$  estados y  $p$  símbolos de cinta. Por tanto  $G_w$  tiene un máximo de  $pd^{2S(n)} + 1$  variables. Como  $A$  puede presionar sólo un símbolo, ningún lado derecho de una producción de  $G_w$  tiene más de dos variables, así que existen cuando mucho  $rd^{S(n)}$  producciones para cualquier variable no terminal de  $G_w$ , en donde  $r$  es el número máximo de alternativas que  $A$  tiene en cualquier situación. Por consiguiente la prueba del Teorema 6.6, para decir si  $L(G_w)$  está vacía, toma un tiempo proporcional a  $rp^2d^{5S(n)}$ , cuando mucho. Puesto que  $r, p$  y  $d$  son constantes, existe una constante  $c$  tal que  $M$  puede determinar, en un tiempo máximo de  $c^{S(n)}$ , si  $L(G_w)$  está no vacío, es decir, si  $w$  es aceptado por  $A$ .  $\square$

**Lema 14.2** Si  $L$  se encuentra en TIEMPOD( $T(n)$ ), entonces  $L$  es aceptado en un tiempo  $T^*(n)$  por una TM de una sola cinta,  $M$ , que recorre su cinta haciendo un barrido completo en una dirección, y que alcanza la primera celda que no había sido barrida hasta entonces, invierte su dirección y repite el proceso. Véase la Fig. 14.2

**Demostración** Según los Teoremas 12.3 y 12.5,  $L$  es aceptado por una TM de una

<sup>†</sup> Nótese que una “configuración” en el sentido utilizado en el texto no incluye a los contenidos de pila.

Fig. 14.2 Patrón transversal de la TM,  $M$ .

cinta y límite temporal  $\frac{1}{2} T^2(n), M_1$ . Simula a  $M_1$  señalando, en un segundo registro, la posición de la cabeza de  $M_1$  y las celdas que  $M_1$  ya ha barrido. Mientras que la cabeza de  $M_1$  se desplace en la misma dirección que la de  $M$ ,  $M$  puede simular un movimiento de  $M_1$  con cada uno de sus propios movimientos. Cuando  $M_1$  se mueve en la dirección opuesta,  $M$  abandona al señalador de cabeza, completa su barrido y simula ese movimiento en el recorrido de regreso. Por consiguiente  $M$  simula al menos un movimiento de  $M_1$  por recorrido, tomando un máximo de  $\sum_{i=0}^{(1/2)T^2(n)} n + i \leq T^4(n)$  movimientos para completar la simulación de  $M_1$ .  $\square$

**Lema 14.3** Si  $L$  se encuentra en  $\text{TIEMPOD}(c^{s(n)})$ , para cualquier constante  $c$ , entonces  $L$  es aceptado por un APDA determinístico con límite espacial  $S(n)$ .

*Demuestra* Según el Lema 14.2,  $L$  es aceptado por una TM de una sola cinta y límite temporal  $c^{4S(n)}, M$ , con el patrón de recorrido que se muestra en la Fig. 14.2. Definase  $d = c^4$  de modo que  $M$  tenga límite temporal  $d^{s(n)}$ . Hagamos que el conjunto de tres parámetros  $(q, Z, t)$  sea la proposición consistente en que, al tiempo  $t$ ,  $\dagger M$  esté en el estado  $q$  barriendo al símbolo  $Z$ , en donde  $t \leq d^{s(n)}$ . Nótese que como el movimiento de la cabeza de  $M$  es independiente de los datos, la celda barrida al tiempo  $t$  se puede calcular de manera sencilla a partir de  $t$ .

La parte medular de la construcción de un APDA determinístico con límite  $S(n)$ ,  $A$ , que acepte a  $L$  consiste en el procedimiento recursivo PRUEBA de la Fig. 14.3, que asigna el valor de verdadero a  $(q, Z, t)$  si y sólo si.

1.  $t = 0$ ,  $q$  es el estado inicial y  $Z$  el símbolo de la primera celda de  $M$ ,
2. al tiempo  $t$ ,  $M$  barre por primera vez una celda,  $Z$  es el símbolo original en esa celda de cinta, y existe un triplete  $(p, X, t-1)$  que es verdadero e implica que  $M$  accesa el estado  $q$  después de un movimiento, o
3.  $M$  ha barrido, previamente, la celda visitada en el tiempo  $t$  y existen tripletes verdaderos  $(p_1, X_1, t-1)$  y  $(p_2, X_2, t')$  tales que el primer triplete implica que  $q$  es accesado después de un movimiento, y el segundo implica que  $Z$  se quedó en la celda de cinta la última vez que ésta fue barrida. Recuérdese que el movimiento

de la cabeza de  $M$  es uniforme y, por tanto, el tiempo  $t'$ , al cual la celda fue visitada por última vez, se puede calcular fácilmente a partir de  $t$ .

Dado que PRUEBA solamente se llama a sí misma con argumentos menores, finalmente termina. El APDA en espacio  $S(n), A$ , evalúa a PRUEBA manteniendo los argumentos en la cinta de almacenamiento.

Cuando PRUEBA se llama a sí misma,  $A$  incluye a los viejos argumentos en la pila y cuando PRUEBA regresa,  $A$  los elimina de la pila y los coloca en la cinta de almacenamiento.

El algoritmo completo que  $A$  ejecuta es

```
for cada triplete  $(q, Z, t)$  tal que  $q$  es un estado de aceptación y  $0 \leq t \leq d^{s(n)}$  do
    if PRUEBA  $(q, Z, t)$  then accept
     $\square$ 
```

**Teorema 14.1** Para  $s(n) \geq \log n$ , las siguientes proposiciones son equivalentes.

1.  $L$  es aceptado por APDA determinístico con límite  $S(n)$ .

---

```
procedure PRUEBA  $(q, Z, t)$ ;
begin
    if  $t = 0$ ,  $q$  es el estado inicial de  $M$  y  $Z$  es el primer símbolo de entrada then return true;
    if  $1 \leq t < n$  y  $Z$  es el  $t$ -ésimo símbolo de entrada, o  $t = in + i(i-1)/2$ , para algún entero  $i \geq 1$ ,
        y  $Z = B$ , then
            for cada estado  $p$  y símbolo  $X$  do
                if  $M$  entra el estado  $q$  cuando está barriendo a  $X$  en el estado  $p$ , y PRUEBA  $(p, X, t-1)$ 
                    then return true;
    /* los tiempos  $in + i(i-1)/2$  son exactamente los tiempos en que  $M$  barre una nueva celda */
    if  $t \geq n$  y  $t \neq in + i(i-1)/2$ , para cualquier entero  $i \geq 1$  then
        begin
            sea  $t'$  la ocasión anterior en que  $M$  barrió la misma celda que en el tiempo  $t$ ;
            for todos los estados  $p_1$  y  $p_2$  y los símbolos  $X_1$  y  $X_2$  do
                if  $M$  entra un estado  $q$  cuando está barriendo a  $X_1$  en el estado  $p_1$  y  $M$  escribe  $Z$  cuando
                    barre a  $X_2$  en el estado  $p_2$  y PRUEBA  $(p_1, X_1, t-1)$  y PRUEBA  $(p_2, X_2, t')$ 
                then return true
        end;
        return false
    end
```

Fig. 14.3 El procedimiento PRUEBA.

2.  $L$  es aceptado por un APDA no determinístico con límite  $S(n)$ .
3.  $L$  se encuentra en  $\text{TIEMPOD}(C^s(n))$ , para alguna constante  $c$ .

*Demuestra* Que (1) implica (2) es obvio. El Lema 14.1 establece que (2) implica (3) y el Lema 14.3 establece que (3) implica (1).  $\square$

**Corolario**  $L$  se encuentra en  $\mathcal{P}$  si y sólo si  $L$  es aceptado por un APDA con límite espacial  $\log n$ .

$\dagger$  Al tiempo "significa" después de que  $t$  movimientos han transcurrido", de manera que, inicialmente,  $t = 0$ .

## 14.2 AUTOMATAS DE PILA

El *autómata de pila* (SA, del inglés *Stack Automata*) es un PDA con las dos características siguientes.

1. La entrada es de dos direcciones, se encuentra en el modo de sólo lectura con señaladores de extremo.
2. La cabeza de pila, además de colocar y eliminar movimientos en el tope de la pila, puede accesar a la pila en el modo de sólo lectura, trasladándose hacia arriba y abajo de la pila sin reescribir ningún símbolo.

En la Fig. 14.4 se muestra un autómata de pila en el modo de sólo lectura.

Un movimiento de un SA queda determinado por el estado, la entrada y los símbolos de pila barridos, ya sea que el tope de la pila esté siendo barrido por la cabeza de pila o no lo esté. En cualquier caso, en un solo movimiento el estado puede cambiar y la cabeza de entrada moverse una posición hacia la izquierda o la derecha. Si la cabeza de pila no se encuentra en el tope de ésta, un movimiento también puede incluir un desplazamiento de la cabeza de pila, en una posición arriba o abajo de la pila. Si la cabeza se encuentra en el tope las acciones de pila que se permiten son:

1. introducir un símbolo en la pila,

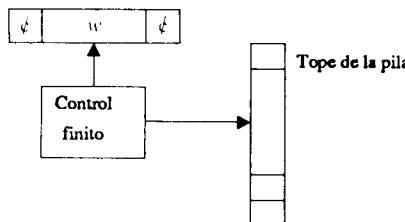


Fig. 14.4 Un autómata de pila.

2. retirar el símbolo del tope, o
3. moverse una posición más abajo en la pila sin añadir o retirar símbolos.

Durante las acciones (1) y (2) la cabeza de la pila permanece en el tope; durante la acción (3) abandona el tope y entra en el modo sólo de lectura, el cual puede abandonar sólo si regresa al tope de la pila.

Inicialmente, la cabeza de entrada se encuentra en el extremo izquierdo y el control finito en un estado inicial designado, y la pila consiste en un solo símbolo inicial designado. La aceptación se produce al entrar en un estado final.

Si nunca se efectúa más de un movimiento en cualquier actuación, el dispositivo es determinístico (un DSA); si existe un número finito de alternativas para el movimiento en cualquier situación, el autómata es no determinístico (un NSA). Si el dispositivo nunca retira un símbolo, entonces es *no borrador* (NEDSA o NENSA; NE,

del inglés *non erasing*). Si la cabeza de entrada nunca se mueve hacia la izquierda, el autómata de pila es de *una sola dirección* (un 1DSA, 1NENSA, etcétera). Cuando no haya ninguna aseveración que indique lo contrario, supondremos que un SA es de dos direcciones, determinístico, y que permite borrar.

**Ejemplo 14.1** Sea  $L = \{0^n 1^n 2^n \mid n \geq 1\}$ . Diseñamos un SA para aceptar a  $L$  de la manera siguiente. La cabeza de entrada se desplaza hacia la derecha en cada movimiento. Mientras encuentre sólo ceros, éstos son colocados en la pila por encima del señalador del fondo (símbolo inicial)  $Z_0$ . La cabeza de la pila permanece en el tope de ésta en el modo lectura-escritura. La Fig. 14.5(a) muestra la situación después de haber leído los ceros. Cuando lee el primer uno, la cabeza de la pila se mueve una posición hacia abajo y entra en el modo de sólo lectura. Conforme se van leyendo unos sucesivos, la cabeza de pila se desplaza una posición hacia abajo por cada uno (pero si no se ve el primer dos al mismo tiempo que la cabeza de pila alcanza el señalador del fondo, entonces no habrá un siguiente movimiento y el SA no aceptará). La situación en la que se encuentra entonces el SA se muestra en la Fig. 14.5(b). Conforme se van leyendo los 2s en la entrada, la cabeza de cinta se desplaza hacia arriba, una posición por cada 2. Se puede permitir un movimiento a un estado de aceptación solamente cuando la cabeza de la pila se encuentran en el tope, y se barre el señalador  $\$$  sobre la entrada, como se muestra en la Fig. 14.5(c). Por supuesto que el estado a partir del cual se puede hacer este movimiento solamente es accesado después de haber visto 2s, de modo que no podemos aceptar entradas como  $\$0\$$  o  $\$00\$$ .

Nótese que el SA que hemos descrito es de una sola dirección, determinístico y no borrador.

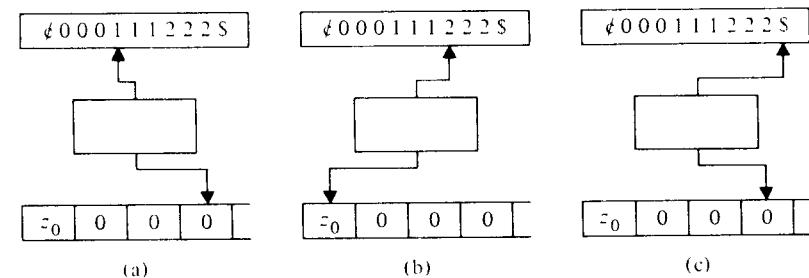


Fig. 14.5 IDs de un autómata de pila.

### Tablas de transición

En lo que resta de la sección daremos bosquejos de demostración para un cierto número de los resultados fundamentales que caracterizan a los lenguajes aceptados por los diferentes autómatas de pila. Una de las ideas principales consiste en la simulación de autómatas de pila por otros dispositivos mediante una *tabla de transición*, que es una representación concisa de una pila (en realidad es la pila sin el símbolo del tope).

Supóngase que un autómata de pila determinístico se encuentra en el estado  $q$  con la cabeza de entrada en la posición  $i$  y la cabeza de la pila en el símbolo siguiente al del tope. Entonces el SA se encuentra en el modo de sólo lectura y la pila no puede cambiar hasta que la cabeza de pila alcanza el tope. Para una secuencia particular de símbolos de pila, puede ser que la cabeza de pila nunca alcance el tope o llegará a ésta por primera vez en algún estado  $p$  con la cabeza de entrada en la posición  $j$ . Para cada  $q$  e  $i$ , la tabla de transición nos dice si la cabeza de pila se mueve hasta el tope y ya así da el estado  $p$  y la posición  $j$  en los cuales se llega a la cima. Por consiguiente, la tabla de transición caracteriza completamente el efecto de la secuencia de símbolos de pila que se encuentra debajo del símbolo del tope siempre y cuando no ocurra la aceptación cuando la cabeza se encuentre dentro de la pila.

El número de tablas de transición distintas para un SA con entrada de longitud  $n$  (excluyendo los señaladores de extremo) y con  $s$  estados es, por tanto,  $[s(n+2)+1]^{s(n+2)}$ . Con las posiciones de entrada codificadas en binario, una tabla de transición sólo requiere  $cn \log n$  bits para alguna constante  $c$  que depende del número de estados del SA dado.

Si el autómata de pila es no determinístico, entonces, para cada  $q$  e  $i$ , la tabla de transición debe dar el conjunto de pares  $(p, j)$  tales que si se comienza en el estado  $q$ , con posición de entrada  $i$  y la cabeza de pila en el símbolo anterior al del tope puede alcanzarse el tope de la pila en el estado  $p$  y la posición  $j$ . El número de tablas de transición posibles de un NSA de estado  $s$  con entrada de longitud  $n$  es  $[2^{s(n+2)+1}]^{s(n+2)} \leq 2^{cn^2}$ , de modo que una tabla de transición con estas características puede ser codificada en  $cn^2$  bits, en donde  $c$  depende solamente del número de estados del NSA.

### Caracterización de los lenguajes de pila mediante clases de complejidad temporal y espacial

Mostraremos ahora que un SA determinístico puede ser simulado por un APDA con límite  $n \log n$  y, de manera inversa, que un APDA con límite  $n \log n$  puede ser simulado por un DSA, estableciéndose la equivalencia de los DSA y los APDA con límite  $n \log n$ . De manera similar, estableceremos la equivalencia de los NSA y los APDA con límite  $n^2$ . Para el SA no borrador estableceremos la equivalencia de NEDSA y ESPACIOD( $n \log n$ ) y la equivalencia de NENSA y ESPACION( $n^2$ ). Utilizaremos una serie de lemas.

**Lema 14.4** Cada tipo de autómata de pila es equivalente a uno del mismo tipo que acepta solamente en el tope de la pila.

*Demostración* Modificamos un SA dado de manera que, en cada estado de aceptación, mueva su cabeza de pila hacia arriba de la pila hasta que alcance el tope.  $\square$

**Lema 14.5** Si  $L$  es aceptado por un NEDSA, entonces  $L$  se encuentra en ESPACIOD( $n \log n$ ).

*Demostración* Dado un NEDSA  $A$  que acepta solamente en el tope de la pila, construimos una máquina de Turing  $M$  que simula a  $A$  manteniendo registro del estado

de  $A$ , la posición de la cabeza de entrada, del símbolo que se encuentra en la cima de la pila y de la tabla de transición para la parte de la pila que está debajo del símbolo del tope. La tabla de transición inicial es la tabla asociada con la cadena vacía ("no definida" para ninguna  $q$  e  $i$ ). Sólo necesitamos explicar cómo construir una tabla de transición  $T'$  asociada con la cadena de pila  $X_1 X_2 \cdots X_m$ , dada la tabla  $T$  para  $X_1 X_2 \cdots X_{m-1}$ .

Para cada estrado  $q$  y posición de entrada  $i$ , ejecútese el algoritmo de la Fig. 14.6. El algoritmo mantiene un registro de la secuencia de pares estado-posición de entrada  $(p, j)$  en el cual  $X_m$  está siendo barrido. Cada vez que la cabeza de pila se mueve a  $X_{m-1}$ ,  $T$  es consultado para determinar el siguiente par estado-posición en el cual  $X_m$  será barrido, si esto sucede la variable CUENTA verifica que la longitud de la secuencia de  $(p, j)$ s no excede el producto de  $s$ , el número de estados, y  $n+2$ , el número de posiciones de entrada. Si lo anterior sucede,  $A$  se encuentra seguramente en un ciclo, de modo que dicho valor de  $T'(q, i)$  está "indefinido".

```

begin
    CUENTA: = 0;
    (p, j): = (q, i);
    while CUENTA < s(n+2) do
        begin
            CUENTA: = CUENTA + 1
            supóngase que  $A$  está en el estado  $p$ , y cuando está barriendo el símbolo de pila  $X_m$ , en
            la posición de entrada  $j$ , entra el estado  $r$ , mueve la cabeza de entrada a la posición
             $k$  y la cabeza de pila en la dirección D;
            if  $D$  = "arriba" the return  $(r, k)$ ;
            if  $D$  = "estacionario" the  $(p, j): = (r, k)$ ;
            if  $D$  = "abajo" then
                if  $T(r, k) = "indefinido"$  the return "indefinido"
                else  $(p, j): = T(r, k)$ 
            end
            return "indefinido"
        end
    end

```

Fig. 14.6 Algoritmo para calcular la tabla de transición para NEDSA.

Nótese que, para un  $(q, i)$  dado, el algoritmo de las Fig. 14.6 sólo requiere espacio  $O(\log n)$  para mantener el valor de CUENTA. Por consiguiente  $T'$  puede ser calculado a partir de  $T$  y  $X_m$  en el espacio que se lleva en almacenar a  $T$  y  $T'$ , que es  $n \log n$ . La TM  $M$ , sólo tiene que simular a  $A$  directamente cuando la cabeza de la pila abandona el tope consultar la tabla de transición actual cuando la cabeza de la pila abandona el tope calcular una nueva tabla de transición (eliminando la anterior) cuando se coloca un nuevo símbolo de pila. Como los símbolos de pila nunca se borran, no necesitamos conservar la pila.  $\square$

**Lema 14.6** Si  $L$  es aceptado por un NENSA, entonces  $L$  se encuentra en ESPACION( $n^2$ ).

*Demostración* La demostración es parecida a la del lema anterior, excepto que se necesita un espacio  $n^2$  para almacenar la matriz de transición, y la simulación debe ser no determinística.  $\square$

**Lema 14.7** Si  $L$  es aceptado por un DSA, entonces es aceptado por un APDA con límite  $n \log n$ .

*Demostración* De nuevo la demostración es parecida a la del Lema 14.5. El APDA utiliza su pila (la cual no puede ser introducida en el modo sólo lectura, por supuesto) para contener la pila del DSA. Entre cada símbolo de pila del DSA el APDA almacena una tabla de transición. La tabla de transición con respecto a un símbolo de pila particular corresponde a la pila completa, hasta incluir a dicho símbolo. El símbolo de pila que se encuentra en el tope y la tabla de transición para la pila debajo de éste se colocan en la cinta de almacenamiento. Cuando el DSA coloca un símbolo en la pila, el APDA coloca la tabla que se encuentra en su cinta de almacenamiento junto con el anterior símbolo del tope en su propia pila, y calcula la nueva tabla como se hizo en el Lema 14.5. Cuando el DSA elimina un símbolo, el APDA descarta el símbolo que se encuentra en el tope de la pila y después traslada la tabla del tope a su cinta de almacenamiento.  $\square$

**Lema 14.8** Si  $L$  es aceptado por un NSA, entonces  $L$  es aceptado por un APDA con límite  $n^2$ .

*Demostración* La demostración es una combinación de las ideas que se introdujeron en los Lemas 14.6 y 14.7. Nótese que, según el Teorema 14.1, el APDA puede hacerse determinístico.  $\square$

Pondremos nuestra atención ahora en la simulación de dispositivos limitados en el espacio mediante autómatas de pila. La idea clave aquí consiste en que el SA puede utilizar su entrada de longitud  $n$  para contar  $n$  símbolos o “bloques” de símbolos, de arriba hacia abajo, en su pila. Se construye en la pila una secuencia de IDs que representan un cálculo en un dispositivo limitado en el espacio, mediante el copiado sucesivo de la ID del tope en la pila, haciendo que los cambios sean representados por un movimiento del dispositivo limitado en el espacio. La capacidad para contar  $n$  símbolos o “bloques de símbolos” le permite al SA copiar el ID actual en la cima, símbolo por símbolo.

Con una introducción sencilla considérese la simulación de un autómata determinístico,  $M$ , limitado linealmente, mediante un NEDSA  $A$ . Dada la entrada  $w = a_1 \dots a_n$ ,  $A$  coloca a  $[q_0 a_1] a_2 \dots a_n \#$  en su pila, en donde  $q_0$  es el símbolo inicial y  $\#$  es un símbolo especial que separa IDs. El estado se combina con los símbolos barridos, de modo que una ID es siempre exactamente de  $n$  símbolos de longitud. Supóngase que  $A$  ha construido una pila que es una sucesión de IDs, incluyendo los primeros  $i$  símbolos de la siguiente ID:

$$\dots \# X_1 X_2 \dots X_n \# X_1 X_2 \dots X_i.$$

(En realidad uno o dos de los  $X$ s de la ID que se está construyendo pueden ser diferentes de los correspondientes símbolos de la ID que se encuentra debajo, debido al movimiento hecho por  $M$ ). Comenzando por el extremo izquierdo de la entrada,  $A$ , repetidamente, se mueve una posición a la derecha sobre la entrada y una posición hacia abajo en la pila, hasta que se alcanza, sobre la entrada, el señalador de extremo de la derecha. En este punto, la cabeza de pila de  $A$  se encontrará a  $n + 1$  símbolos del to-

pe de la pila, barriendo a  $X_{i+1}$  de la última ID completa. A mira un símbolo encima y debajo de  $X_{i+1}$  para observar si  $X_{i+1}$  cambia en la siguiente ID debido al movimiento que hace  $M$ . Entonces  $A$  se mueve al tope de la pila y coloca en ésta a  $X_{i+1}$  o el símbolo que sustituya a  $X_{i+1}$  en la siguiente ID debido al movimiento de  $M$ .  $A$  acepta si y sólo si  $M$  entra un estado de aceptación.

En realidad un autómata de pila puede simular dispositivos con IDs de longitud mayor que  $n$  mediante un uso más inteligente de la entrada. En particular, un DSA puede manipular IDs de longitud  $n \log n$ , y un NSA puede manejar IDs de longitud  $n^2$ . El caso no determinístico resulta más sencillo, de modo que lo presentaremos primero.

**Lema 14.9** Si  $L$  se encuentra en  $\text{ESPACION}(n^2)$ , entonces  $L$  es aceptado por un NENSA.

*Demostración* Puesto que  $n^2$  es mayor que  $n$ , podemos suponer que  $L$  es aceptado por una TM de una sola cinta más que por una TM fuera de línea. Una ID de longitud  $n^2$  está representada mediante un listado de los símbolos de cinta, combinando el estado con el símbolo barrido por la cabeza de cinta. Se inserta un señalador \* después de cada  $n$  símbolos. Los  $n$  símbolos que se encuentran entre \*s constituyen un *bloque*. Las IDs sucesivas se colocan en la pila como se hizo en la descripción del LBA más arriba. Supóngase que se tienen  $j$  bloques e  $i$  símbolos del  $(j + 1)$ -ésimo bloque copiados. La cinta de entrada se utiliza para medir  $n$  \*s hacia abajo en la pila hasta el  $(j + 1)$ -ésimo bloque de la ID anterior. Se adivina una posición  $k$ ,  $1 \leq k \leq n$ , en el  $(j + 1)$ -ésimo bloque. Verificando un símbolo arriba y uno abajo se determina si el símbolo se ve afectado por un movimiento de TM. Si esto es así, se adivina un movimiento, siempre y cuando no se haya adivinado un movimiento para esta ID con anterioridad; de otra forma el símbolo es registrado en el estado del SA. La cinta de entrada entonces se utiliza para registrar a  $k$  moviendo alternativamente la cabeza de entrada un símbolo hacia la izquierda (comenzando en el extremo derecho) y la cabeza de pila un símbolo hacia abajo hasta encontrar un \*. En seguida la cabeza de pila se mueve hasta el tope y compara a  $k$  con  $i$ , el número de símbolos del  $j$ -ésimo bloque que ya ha sido copiado. Si  $k \neq i + 1$ , esta sucesión de alternativas “se muere”. Si  $k = i + 1$ , entonces el siguiente símbolo de la nueva ID se coloca en el tope de la pila. La entrada entonces se utiliza para determinar si  $i + 1 = n$ . Si esto es cierto se imprime un \*, y después se verifica si  $j + 1 = n$ . En el caso en que  $j + 1 = n$ , se coloca un # en la pila para señalar el final de una ID. La aceptación se da si el símbolo copiado incluye un estado final. De otra manera se copia el siguiente símbolo.

Un punto pequeño pero importante consiste en que una vez que se ha adivinado un movimiento, al copiar una ID, la adivinación no puede ser combinada al copiar símbolos posteriores de esa ID. De lo contrario puede construirse una ID sucesora que no es válida.  $\square$

**Teorema 14.2** La familia de lenguajes aceptados por los autómatas de pila no determinísticos y no borradores se encuentra exactamente en  $\text{ESPACION}(n^2)$ .

*Demostración* La demostración es inmediata si se consideran los Lemas 14.6 y 14.9.  $\square$

**Teorema 14.3** La familia de los lenguajes aceptados por los autómatas de pila no determinísticos se encuentran exactamente en  $\bigcup_{c>0} \text{TIEMPOD}(c^n)$ .

**Demostración** Según el Teorema 14.1,  $L$  se encuentra en  $\bigcup_{c>0} \text{TIEMPOD}(c^n)$  si y sólo si  $L$  es aceptado por un APDA con límite  $n^2$ . Según el Lema 14.8, si  $L$  es aceptado por un NSA, entonces  $L$  es aceptado por un APDA determinístico con límite  $n^2$ . Esto es suficiente para mostrar que un APDA determinístico con límite  $n^2$ ,  $A$ , puede ser simulado por un NSA,  $S$ .

Suponemos que la entrada de  $A$  se mantiene en la cinta de almacenamiento de  $A$  más que en la entrada de sólo lectura, ya que  $n^2$  es mayor que  $n$ . La pila de  $S$  contendrá a la pila de  $A$  así como a una sucesión de IDs que representan a la cinta de almacenamiento de  $A$ . Supóngase que  $S$  tiene el contenido actual de la cinta de almacenamiento de  $A$  en el tope de su pila, y que  $A$  coloca un símbolo.  $S$  supone el contenido de la cinta de  $A$  cuando dicho símbolo es retirado y coloca su adivinanza en el tope de la pila. Entonces  $S$  coloca el símbolo colocado por  $A$  y crea el nuevo contenido actual de la cinta de  $A$  a partir del contenido viejo, como se hizo en la demostración del Lema 14.9. La ID adivinada que interviene se ignora mientras se recorre la pila de arriba abajo; sus símbolos pueden escogerse de alfabetos separados, de modo que  $S$  pueda saltar sobre ella.

Si  $A$  elimina un símbolo,  $S$  verifica que la ID adivinada que se encuentra debajo de ese símbolo sea correcta; esto es, que la ID supuesta sea la cinta de almacenamiento de  $A$  después del movimiento de eliminación. La ID actual de  $A$  que se mantiene en el tope de la pila de  $S$  es eliminada, un símbolo a la vez, y cada símbolo eliminado es comparado con el símbolo correspondiente de la ID adivinada mediante un método parecido al del Lema 14.9. Si la adivinación es correcta, la ID adivinada se convierte en el contenido actual de la cinta de almacenamiento de  $A$ , y la simulación de  $A$  continúa; si no, esta secuencia de alternativas de  $S$  “muere”  $S$  acepta si y sólo si  $A$  vacía su pila.  $\square$

**Corolario**  $L$  es aceptado por una NSA si y sólo si  $L$  es aceptado por  $n$  APDA con límite  $n^2$ .

**Demostración** La parte “sólo si” se estableció en el Lema 14.8. La parte “si” se concluye de manera inmediata de los Teoremas 14.1 y 14.3.  $\square$

En el caso determinístico la función  $n^2$  se sustituye por  $n \log n$  en los teoremas que son los análogos del 14.2 y 14.3. La razón de esto es que durante la construcción del Lema 14.9, el NSA hizo un uso esencial de su no determinismo para copiar IDs de longitud  $n^2$ . Un DSA sólo es capaz de copiar IDs de longitud  $n \log n$ .

**Lema 14.10** Si  $L$  se encuentra en  $\text{ESPACIOD}(n \log n)$ , entonces  $L$  es aceptado por un NEDSA.

**Demostración** Sea  $L$  un lenguaje aceptado por alguna TM  $M$ , de una sola cinta que utiliza exactamente  $n \log n$  celdas. Sea  $t$  el número de símbolos de la forma  $X$  o  $[qX]$ , en donde  $x$  es un símbolo de cinta y  $q$  un estado. Estos símbolos se identifican con los

dígitos  $1, 2, \dots, t$  en base  $t+1$ . Las cadenas formadas con  $\lfloor n \log_{t+1}(n) \rfloor$  de tales símbolos de  $M$  se codifican como bloques de ceros cuya cantidad se encuentra entre 0 y  $(n-1)$ . Existe un entero  $c$ , que depende solamente de  $M$ , tal que una ID de  $M$  puede estar representada por  $cn$  bloques de ceros, cada uno de los cuales codifica  $\lfloor n \log_{t+1}(n) \rfloor$  símbolos, siempre que  $n > t$ .

Diséñese un autómata de pila  $S$  para construir una sucesión de IDs de  $M$ , y que cada ID sea una sucesión de  $cn$  bloques de entre 0 y  $(n-1)$  ceros separados por señaladores \*. Los bloques son copiados en el tope de la pila mediante el uso de la entrada para contar  $cn$  \*s hacia abajo en la pila, midiendo la longitud del bloque que va ha ser copiado en la entrada; más adelante se mueve al tope de la pila y coloca un número igual de ceros en la pila.

Antes de colocar un nuevo bloque en la pila es necesario determinar qué símbolo cambia, si realmente hay alguno. Para hacer esto decodifíquese  $0^k$  dividendo repetidamente entre  $t+1$ , y tomando a los residuos posteriores como los símbolos posteriores de la ID. La división se lleva a cabo midiendo  $k$  sobre la entrada y después moviendo la entrada de regreso al señalador de extremo, colocando  $Xs$  en la pila por cada  $t+1$  posiciones que la cabeza de entrada se mueva. Las  $Xs$  no forman parte de la ID. El control finito calcula a  $k \bmod (t+1)$ , y el dígito resultante se coloca por encima de las  $Xs$ . Entonces el bloque de  $Xs$  se mide sobre la entrada y el proceso se repite hasta que el bloque tenga longitud cero. Los dígitos escritos en la pila entre los bloques de  $X$  son el bloque deseado de la ID.

$S$  verifica si la cabeza está barriendo un símbolo en el bloque y también anota si la cabeza se mueve hacia un bloque contiguo. Los bloques se recodifican en cadena de 0 a  $(n+1)$  ceros, efectuando los cambios necesarios para reflejar el movimiento de  $M$ . El proceso de recodificación es el inverso de lo que se acaba de describir. Nótese que como  $S$  es no borrador, nunca se deshace de las  $Xs$  o de los dígitos de su pila; simplemente son ignorados en el cálculo posterior. También, antes de copiar un bloque,  $S$  debe decodificar el bloque que se encuentra arriba, para ver si la cabeza de  $M$  se mueve hacia la izquierda en el presente bloque.

$S$  inicia su pila mediante una codificación de su propia entrada como una ID de  $M$ . Los detalles de este proceso se omiten.  $S$  acepta si descubre que  $M$  entra un estado de aceptación.  $\square$

**Teorema 14.4**  $L$  es aceptado por un autómata de pila determinístico no borrador si y sólo si  $L$  se encuentra en  $\text{ESPACIOD}(n \log n)$

**Demostración** Esta se construye a partir de los Lemas 14.5 y 14.10.  $\square$

**Teorema 14.5**  $L$  es aceptado por un autómata de pila determinístico si y sólo si  $L$  se encuentra en  $\bigcup_{c>0} \text{TIEMPOD}(n^c)$ .

**Demostración** Nótese que  $n^c = 2^{cn \log n}$ . Según el Teorema 14.1,  $L$  se encuentra en  $\bigcup_{c>0} \text{TIEMPOD}(n^c)$  si y sólo si  $L$  es aceptado por un APDA con límite  $cn \log n$ , según el Lema 14.7, si  $L$  es aceptado por un DSA, entonces  $L$  es aceptado por un APDA con límite  $n \log n$  determinístico. Por consiguiente es suficiente mostrar que si  $L$  es

aceptado por un APDA con límite  $n \log n$  determinístico,  $A$ , entonces  $L$  es aceptado por un DSA,  $S$ . Suponemos de nuevo que la cinta de entrada de  $A$  se combina con la cinta de almacenamiento. La demostración se efectúa de manera paralela a la del Teorema 14.3, utilizando las técnicas del Lema 14.10 para representar las cintas de almacenamiento de  $A$ , y simular los movimientos de  $A$ . Sin embargo, cuando  $A$  coloca un símbolo  $X$  en su pila,  $S$ , como es determinístico, no puede adivinar el contenido de la cinta de almacenamiento de  $A$  cuando  $A$ , en algún momento, elimina a ese símbolo  $X$ , en lugar de esto,  $S$  efectúa ciclos a través de todas las IDs posibles de manera sistemática. Si ha hecho la elección equivocada, genera a la siguiente ID posible y recomienza la simulación de  $A$  a partir del momento en que  $X$  fue colocado por  $A$ . El hecho de que  $A$  vacíe su pila para aceptar, nos asegura que si  $A$  acepta,  $S$  en algún momento tendrá la oportunidad de generar la alternativa correcta.  $\square$

**Corolario**  $L$  es aceptado por un DSA si y sólo si  $L$  es aceptado por un APDA con límite  $n \log n$ .

**Demostración** La parte “sólo si” se establece con el Lema 14.7. La parte “si” se concluye de manera inmediata a partir de los Teoremas 14.1 y 14.5.  $\square$

Un autómata de pila de una sola dirección no es lo suficientemente poderoso para simular dispositivos con límites en la cinta. Sin embargo, existe una importante relación de contención que plantearemos sin dar una demostración.

**Teorema 14.6** Si  $L$  es aceptado por un 1NSA, entonces  $L$  se encuentra en ESPACIOD( $n$ ).

### 14.3 LENGUAJES INDEXADOS

De las diversas generalizaciones de las gramáticas libres de contexto que hemos propuesto, una clase, conocida como “indexada”, parece ser la más natural, en el sentido de que aparece en una amplia variedad de contextos. Aquí daremos una definición de gramática. Otras definiciones de los lenguajes indexados se citan en las notas bibliográficas.

Una gramática indexada es un conjunto de cinco parámetros  $(V, T, I, P, S)$ , en el que  $V$  es el conjunto de variables,  $T$  el conjunto de terminales,  $I$  el conjunto de índices,  $S$  en  $V$  el símbolo inicial y  $P$  es un conjunto finito de producciones de las formas.

1.  $A \rightarrow \alpha$ ,
2.  $A \rightarrow Bf$       o      3.  $Af \rightarrow \alpha$ ,

en las que  $A$  y  $B$  se encuentran en  $V$ ,  $f$  en  $I$  y  $\alpha$  en  $(V \cup T)^*$ .

Las derivaciones de una gramática indexada son parecidas a las de una CFG, excepto que las variables pueden estar seguidas de cadenas de índices. (Las terminales pueden no estar seguidas por índices). Cuando se aplica una producción como  $A \rightarrow BC$ , la cadena de índices para  $A$  se asigna a  $B$  y  $C$ . Esta característica permite que muchas partes de una forma oracional sean relacionadas entre sí mediante el compartimiento de una cadena de índices común.

Formalmente, definimos la relación  $\Rightarrow$  sobre las *formas oracionales*, que son

cadenas de  $(VI^* \cup T)^*$ , de la manera siguiente. Sean  $\beta$  y  $\gamma$  en  $(VI^* \cup T)^*$ ,  $\delta$  en  $I^*$  y  $X_i$  en  $V \cup T$ .

1. Si  $A \rightarrow X_1 X_2 \cdots X_k$  es una producción del tipo (1), entonces

$$\beta A \delta \gamma \Rightarrow \beta X_1 \delta_1 X_2 \delta_2 \cdots X_k \delta_k \gamma,$$

en donde  $\delta_i = \delta$  si  $X_i$  se encuentra en  $V$  y  $\delta_i = \epsilon$  si  $X_i$  se encuentra en  $T$ . Cuando se aplica una producción del tipo (1), la cadena de índices  $\delta$  se distribuye sobre las variables que se encuentran en el lado derecho.

2. Si  $A \rightarrow Bf$  es una producción del tipo (2), entonces  $\beta A \delta \gamma \Rightarrow \beta B f \delta \gamma$ . Aquí  $f$  se convierte en el primer índice de la variable  $B$  que sigue después de la cadena, que sustituye a  $A$ .

3. Si  $Af \rightarrow X_1 X_2 \cdots X_k$  es una producción del tipo (3), entonces

$$\beta A f \delta \gamma \Rightarrow \beta X_1 \delta_1 X_2 \delta_2 \cdots X_k \delta_k \gamma,$$

en donde  $\delta_i = \delta$  si  $X_i$  se encuentra en  $V$  y  $\delta_i = \epsilon$  si  $X_i$  se encuentra en  $T$ . El primer índice de la lista de  $A$  es consumido y los restantes se distribuyen sobre las variables como en (1).

Hagamos que  $\overset{*}{\Rightarrow}$  sea la cerradura reflexiva y transitiva de  $\Rightarrow$  como acostumbramos, y definimos  $L(G)$  como  $\{w \mid S \overset{*}{\Rightarrow} w \text{ y } w \text{ se encuentra en } T^*\}$

**Ejemplo 14.2** Sea  $G = (\{S, T, A, B, C\}, \{a, b, c\}, \{f, g\}, P, S)$ , en donde  $P$  consiste en

$$\begin{array}{lll} S \rightarrow Tg, & Af \rightarrow aA, & Ag \rightarrow a, \\ T \rightarrow Tf, & Bf \rightarrow bB, & Bg \rightarrow b, \\ T \rightarrow ABC, & Cf \rightarrow cC, & Cg \rightarrow c. \end{array}$$

Un ejemplo de una derivación en esta gramática indexada es

$$\begin{aligned} S &\Rightarrow Tg \Rightarrow Tf g \Rightarrow AfgBfgCfg \\ &\Rightarrow aAgBfgCfg \Rightarrow aaBfgCfg \Rightarrow aabBfgCfg \\ &\Rightarrow aabbCfg \Rightarrow aabbcCg \Rightarrow aabbcc. \end{aligned}$$

En general tenemos,

$$S \overset{*}{\Rightarrow} Tf^i g \Rightarrow Af^i g Bf^i g Cf^i g \overset{*}{\Rightarrow} a^{i+1} b^{i+1} c^{i+1}.$$

Como la única libertad de derivación de  $G$  consiste en variaciones triviales en orden de sustitución y la alternativa de cuantas veces aplicar  $T \rightarrow Tf$ , deberá resultar claro que

$$L(G) = \{a^n b^n c^n \mid n \geq 1\}.$$

Por supuesto, este lenguaje no es libre de contexto.

Plantearemos, sin demostración, dos de los resultados principales con respecto a los lenguajes indexados.

**Teorema 14.7** (a) Si  $L$  es aceptado por un autómata de pila no determinístico de una sola dirección, entonces  $L$  es un lenguaje indexado, (b) Si  $L$  es un lenguaje indexado, entonces  $L$  es un lenguaje sensible al contexto.

De hecho (a) puede hacerse más fuerte si definimos una generalización de un SA, conocida como “autómata anidado”, cuya variedad no determinística de una sola dirección caracteriza exactamente a los lenguajes indexados. El SA anidado tiene la capacidad, cuando la cabeza de la pila se encuentra dentro de ésta en el modo de sólo lectura, de crear una nueva pila. Sin embargo esta pila debe ser destruida antes de que la cabeza de cinta pueda moverse hacia arriba en la pila original. El proceso de crear nuevas pilas es recursivo y permite la creación de nuevas pilas a una profundidad arbitraria.

#### 14.4 SISTEMAS DE DESARROLLO

La aplicación de las gramáticas al estudio del crecimiento de organismos celulares introdujo nuevas familias de gramáticas conocidas como sistemas  $L$ . Estas familias de gramáticas se diferencian de las gramáticas de Chomsky en que

1. no se hace ninguna distinción entre terminales y no terminales y
2. en cada paso de una derivación se aplica una producción a cada símbolo de una forma oracional, más que a sólo un símbolo o a una subcadena corta.

La modelación de organismos mediante sistemas  $L$  permite la prueba de hipótesis concernientes a los mecanismos que se encuentran detrás de ciertos fenómenos biológicos observables. Aquí nos contentaremos con definir solamente la familia más básica de estas gramáticas, conocidas como sistemas  $L0$  (el cero representa a cero símbolos del contexto; la  $L$  se ha dado en reconocimiento a Arvid Lindenmeyer, quien primero utilizó estas gramáticas para estudiar el crecimiento en organismos).

Una gramática  $L0$  es un conjunto de tres parámetros  $G = (\Sigma, P, \alpha)$ , en donde  $\Sigma$  es un alfabeto finito conocido como *vocabulario*,  $\alpha$  es una cadena de  $\Sigma^*$  conocida como la *cadena inicial* y  $P$  es un conjunto de producciones de la forma  $a \rightarrow \beta$ , en donde  $a$  se encuentra en  $\Sigma$  y  $\beta$  en  $\Sigma^*$ . La relación  $\Rightarrow$  se define mediante

$$a_1 a_2 \cdots a_n \Rightarrow \alpha_1 \alpha_2 \cdots \alpha_n$$

si  $a_i \rightarrow \alpha_i$  se encuentra en  $P$  para  $1 \leq i \leq n$ . Nótese que  $a_i \rightarrow a_i$  debe ser una producción, permitiéndonos no sustituir a  $a_i$ . De otra forma, debe hacerse una sustitución para cada símbolo. La sustitución para diferentes apariciones del mismo símbolo no necesita ser la misma. La relación  $\stackrel{*}{\Rightarrow}$  es la cerradura reflexiva y transitiva de  $\Rightarrow$ , y  $L(G)$  se define como  $(\beta \mid \alpha \stackrel{*}{\Rightarrow} \beta)$ .

**Ejemplo 14.3** Sea  $G = (\{a, b\}, P, a)$ , en la que  $P$  consiste en  $a \rightarrow b$  y  $b \rightarrow ab$ . En este caso, existe solamente una producción para cada símbolo, de modo que en realidad sólo existe una derivación (de longitud infinita), y cada palabra del lenguaje aparece en dicha derivación. Esta es

$$a \Rightarrow b \Rightarrow ab \Rightarrow bab \Rightarrow abbab \Rightarrow bababbab \Rightarrow \dots$$

Nótese que la longitud de las palabras de  $L(G)$  es exactamente el número de Fibonacci. Los números de Fibonacci se definen como  $f_1 = f_2 = 1$  y  $f_i = f_{i-1} + f_{i-2}$ , para  $i \geq 3$ . Se puede demostrar por inducción sobre  $i \geq 3$ , que la  $i$ -ésima palabra de la derivación tiene  $f_{i-1}$   $b$ s y  $f_{i-2}$   $a$ s, y hacen un total de  $f_i$  símbolos

**Ejemplo 14.4** El lenguaje  $\{a, aa\}$  no es un lenguaje  $L0$ . Supóngase que  $L(G) = \{a, aa\}$ , en donde  $G = (\{a\}, P, \alpha)$ . Entonces  $\alpha$  debe encontrarse en  $a$  o en  $aa$ . Ahora, todas las producciones son de la forma  $a \rightarrow a^i$ , para alguna  $i \geq 0$ . Supóngase que  $\alpha = aa$ . Seguramente no puede existir una producción  $a \rightarrow a^i$  para  $i \geq 3$ . Entonces debe existir una producción  $a \rightarrow aa$ , de otra manera  $aa$  nunca sería generada. Pero entonces  $a \Rightarrow aa \Rightarrow aaaa$ , lo que significa una contradicción. En seguida supóngase que  $\alpha = a$ . Debe existir una producción  $a \rightarrow \epsilon$ ; de otra manera todas las cadenas de  $L(G)$  son de longitud dos o mayores. Pero entonces  $aa \Rightarrow \epsilon$ , de modo que  $L(G) \neq \{a, aa\}$  de nuevo.

Un resultado básico concerniente a los lenguajes  $L0$  es el siguiente.

**Teorema 14.8** Si  $L$  es un lenguaje  $L0$ , entonces  $L$  es un lenguaje indexado.

**Demostración** Sea  $G_1 = (\Sigma, P_1, \alpha)$  una gramática  $L0$ . Defínase la gramática indexada  $G_2 = (V, \Sigma, \{f, g\}, P_2, S)$ , en donde

$$V = \{S, T\} \cup \{A_a \mid a \text{ se encuentra en } \Sigma\},$$

y  $P_2$  contiene a

$$S \rightarrow Tg,$$

$$T \rightarrow Tf,$$

$$T \rightarrow A_{a_1} A_{a_2} \cdots A_{a_k} \quad \text{si } \alpha = a_1 a_2 \cdots a_k,$$

$$A_{a_j} \rightarrow A_{b_1} A_{b_2} \cdots A_{b_l} \quad \text{para cada producción } a \rightarrow b_1 b_2 \cdots b_l \text{ in } P_1,$$

$$A_a g \rightarrow a \quad \text{para cada } a \text{ en } \Sigma.$$

De manera informal, la cadena de  $f$ s cuenta el número de pasos de una derivación de  $G_1$ , y el índice  $g$  señala el final de una cadena de índices, permitiendo que una variable sea sustituida por la terminal que representa. Una inducción sencilla sobre la longitud de una derivación nos muestra que

$$S \xrightarrow{*} Tf^i g \xrightarrow{*} A_{b_1} f^{i-j} g A_{b_2} f^{i-j} g \cdots A_{b_k} f^{i-j} g$$

en  $G_2$  si y sólo si  $\alpha :: b_1 b_2 \dots b_k$  mediante una derivación de  $j$  pasos en  $G_1$ . Por consiguiente

$$S \xrightarrow{*} A_{b_1} g A_{b_2} g \cdots A_{b_k} g \xrightarrow{*} b_1 b_2 \cdots b_k$$

si y sólo si  $\alpha \Rightarrow: b_1 b_2 \cdots b_k$ . □

## 14.5 RESUMEN

La Fig. 14.7 muestra las diferentes equivalencias y contenciones demostradas o planteadas en el presente capítulo, más algunas otras que se concluyen de manera inmediata a partir de las definiciones. La contención se indica mediante las aristas verticales.

## EJERCICIOS

### 14.1

- a) Diseñe un DSA de una sola dirección para reconocer al lenguaje  $\{0^n 1^n \mid n \geq 1\}$
- b) Diseñe un NSA de una sola dirección para reconocer al lenguaje  $\{ww \mid w \text{ se encuentra en } (0+1)^*\}$ .

\*14.2 Diseñe un DSA de dos direcciones que acepte al conjunto de cadenas binarias cuyo valor, tratado como un entero, es una potencia de 3.

\*\*14.3 Puesto que cada CFL puede ser reconocido en tiempo polinomial mediante el algoritmo CYK, el corolario al Teorema 14.1 implica que cada CFL es reconocido por algún APDA determinístico con límite  $\log n$ . Dé una construcción directa de dicho DPDA a partir de una CFG.

14.4 Muestre que la familia de los lenguajes 1NSA y la familia de los lenguajes 1NENSA forma AFLs completos.

14.5 Muestre que las familias de lenguajes 1DSA y 1NEDSA son cerradas con respecto a:

- a) la intersección con un conjunto regular,
- b) las transformaciones GSM inversas,
- \*\*c) la complementación
- d) el cociente con un conjunto regular.

14.6 Dé gramáticas indexadas que generen a los lenguajes siguientes.

- |   |   |
|---|---|
| a) $\{0^n \mid n \text{ es un cuadrado perfecto}\}$ | b) $\{0^n \mid n \text{ es una potencia de } 2\}$   |
| c) $\{0^n \mid n \text{ no es un número primo}\}$   | d) $\{ww \mid w \text{ se encuentra en } (0+1)^*\}$ |

14.7 Dé gramáticas  $L_0$  que generen a los lenguajes siguientes

- a)  $\{a^n \mid n \text{ es una potencia de } 2\}$
- b)  $\{wcw^R \mid w \text{ se encuentra en } (0+1)^*\}$

S\*14.8 Dé una gramática  $L_0$  que tenga la propiedad de que cada cadena generada tenga como longitud un cuadrado perfecto y, aún más, para cada cuadrado perfecto, existe al menos una cadena de esa longitud generada.

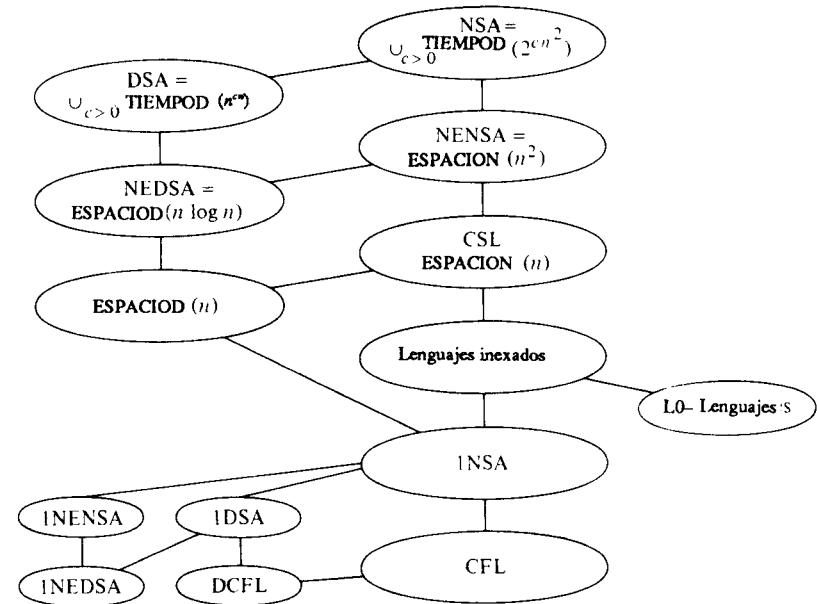


Fig 14.7 Contención entre las clases de lenguajes.

\*14.9 De los ocho subconjuntos de  $(\epsilon, a, aa)$ , ¿cuántos son lenguajes  $L_0$ ?

\*\*14.10 Muestre que la familia de los lenguajes  $L_0$  no es cerrada con respecto a cualquiera de las operaciones de los AFL.

14.11 Muestre que es soluble la cuestión de si el lenguaje generado por una gramática indexada está vacío.

\*14.12 Demuestre que el teorema de Greibach (Teorema 8.14) se aplica a los lenguajes 1NEDSA, y que " $= \Sigma^*$ " es no soluble para esta clase.

\*\*14.13 Muestre que es no soluble la cuestión de si dos lenguajes  $L_0$  son equivalentes.

## Soluciones a los ejercicios seleccionados

14.6 a) Hacemos uso del hecho de que el  $n$ -ésimo cuadrado perfecto es la suma de los primeros  $n$  enteros impares. La gramática indexada con producciones

$$\begin{aligned} S &\rightarrow Ag \\ A &\rightarrow Af \\ A &\rightarrow B \\ A &\rightarrow CD \\ B &\rightarrow CD \\ Df &\rightarrow B \\ Dg &\rightarrow \epsilon \end{aligned}$$

$$Cf \rightarrow 00C$$

$$Cg \rightarrow 0$$

genera a  $\{0^n \mid n \text{ es un cuadrado perfecto}\}$ . Las derivaciones son variaciones triviales de la siguiente derivación.

$$\begin{aligned} S &\Rightarrow Ag \xrightarrow{*} Af^{n-1}g \Rightarrow Bf^{n-1}g \\ &\Rightarrow Cf^{n-1}gDf^{n-1}g \Rightarrow Cf^{n-1}gBf^{n-2}g \\ &\Rightarrow Cf^{n-1}gCf^{n-2}gDf^{n-2}g \\ &\Rightarrow Cf^{n-1}gCf^{n-2}gBf^{n-3}g \Rightarrow \dots \\ &\Rightarrow Cf^{n-1}gCf^{n-2}g \dots CfgCgDg \\ &\Rightarrow Cf^{n-1}gCf^{n-1}g \dots CfgCg \\ &\xrightarrow{*} 0^{2n-1}Cf^{n-1}g \dots Cf_gC_g \xrightarrow{*} \dots \\ &\xrightarrow{*} 0^{2n-1}0^{2n-3} \dots 0^30^1 = 0^{n^2} \end{aligned}$$

**14.8** De nuevo utilizamos el hecho de que el  $n$ -ésimo cuadrado perfecto es la suma de los primeros  $n$  enteros impares. Considérese la gramática  $L0 (\{a, b, c\}, \{a \rightarrow abbc, b \rightarrow bc, c \rightarrow c\} a)$ . Una simple inducción muestra que la  $n$ -ésima cadena generada tiene una  $a$ ,  $2(n-1)$   $bs$  y  $(n-1)^2 cs$ . Por consiguiente la longitud de la  $n$ -ésima cadena es  $1 + 2(n-1) + (n-1)^2$  o  $n^2$ .

## NOTAS BIBLIOGRAFICAS:

El autómata de presión auxiliar y el Teorema 14.1 se tomaron del trabajo de Cook [1971a]. Antes, Mager [1969] ya había considerado "aceptadores de presión de escritura", que son APDAs con límite  $n$ . Los autómatas de pila fueron considerados por primera vez por Ginsburg, Greibach y Harrison [1967a,b]. Los Teoremas 14.2 y 14.4, que relaciona a los autómatas de pila no borradores con las clases de complejidad espacial, se tomaron de Hopcroft y Ullman [1967a], aunque el hecho de que los CSLs están contenidos en los lenguajes NEDSA se tomó del trabajo de Ginsburg, Greibach y Harrison [1967a]. Los Teoremas 14.3 y 14.5, que relaciona los lenguajes de pila con los APDAs y las clases de complejidad temporal, se tomaron de Cook [1971a].

Las propiedades básicas de cerradura y decisión de los lenguajes de pila de una sola dirección se trataron en Ginsburg, Greibach y Harrison [1967b]. El ejercicio 14.5(d), la cerradura de los lenguajes 1DSA con respecto al cociente con un conjunto regular, es de Hopcroft y Ullman [1968b]. El Teorema 14.6, la contención de los lenguajes 1NSA en ESPACIOD( $n$ ) se debe a Hopcroft y Ullman [1968c]. Ogden [1969] da un "lema de sondeo" para los lenguajes de pila de una sola dirección. Beeri [1975] muestra que los SAs de dos direcciones son equivalentes a los autómatas de pila anidados de dos direcciones.

Las gramáticas indexadas fueron estudiadas por primera vez por Aho [1968]. El Teorema 14.7(b), la contención dentro de los CSLs, se tomó de la misma referencia, como también el Ejercicio 14.11, la resolubilidad del vacío. Se conoce una variedad de otras caracterizaciones de los lenguajes indexados. Ahora [1969] discute los autómatas de pila anidados de una sola dirección, una caracterización de un autómata. Fischer [1968] discute las macromáticas, Greibach [1970] proporciona otra caracterización de un autómata: un dispositivo con una pila

de pilas y Maibaum [1974] presenta una caracterización algebraica. Estas formulaciones alternativas le dan validez a la idea de que los lenguajes indexados conforman una clase "natural". Hayashi [1975] da un "lema de sondeo" para lenguajes indexados.

Los sistemas  $L$  se originaron con Lindenmayer [1968], y los sistemas  $L0$ , sobre los cuales centramos nuestra atención, fueron considerados por Lindenmayer [1971]. El Ejercicio 14.10, sobre las propiedades de no cerradura de estos lenguajes, se tomó de Herman [1974]. El Ejercicio 14.13, la irresolubilidad de la equivalencia de los lenguajes  $L0$ , se deduce de un resultado más consistente que dio Blattner [1973], y que consiste en el hecho de que es irresoluble la cuestión de si los conjuntos de formas oracionales generadas por dos CFGs son lo mismo. Se ha escrito mucho sobre la materia, y el lector que se interese puede acudir a Salomaa [1973] y Herman y Rozenberg [1975].

Sólo hemos abarcado una pequeña parte de la multitud de especies de autómatas y gramáticas que se han estudiado. Rosenkrantz [1969] es representativo de otro paso anterior en esta dirección, y Salomaa [1973] cubre una variedad de clases que no han sido tocadas aquí.

## BIBLIOGRAFIA

- Aanderaa, S. O. [1974]. "On  $k$ -tape versus  $(k - 1)$ -tape real time computation," *Complexity of Computation* (R. M. Karp, ed.). Proceedings of SIAM-AMS Symposium in Applied Mathematics.
- Adleman, L., and K. Manders [1976]. "Diophantine complexity," *Proc. Seventeenth Annual IEEE Symposium on Foundations of Computer Science*, pp. 81-88.
- Adleman, L., and K. Manders [1977]. "Reducibility, randomness and intractability," *Proc. Ninth Annual ACM Symposium on the Theory of Computing*, pp. 151-163.
- Aho, A. V. [1968]. "Indexed grammars—an extension of context-free grammars," *J. ACM* **15**: 4, 647-671.
- Aho, A. V. [1969]. "Nested stack automata," *J. ACM* **16**: 3, 383-406.
- Aho, A. V., and M. J. Corasick [1975]. "Efficient string matching: an aid to bibliographic search," *Comm. ACM* **18**: 6, 333-340.
- Aho, A. V., J. E. Hopcroft, and J. D. Ullman [1968]. "Time and tape complexity of push-down automaton languages," *Information and Control* **13**: 3, 186-206.
- Aho, A. V., J. E. Hopcroft, and J. D. Ullman [1974]. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass.
- Aho, A. V., and S. C. Johnson [1974]. "LR parsing," *Computing Surveys*, **6**: 2, 99-124.
- Aho, A. V., and J. D. Ullman [1970]. "A characterization of two-way deterministic classes of languages," *J. Computer and Systems Sciences* **4**: 6, 523-538.
- Aho, A. V., and J. D. Ullman [1972]. *The Theory of Parsing, Translation and Compiling*, Vol. I: *Parsing*, Prentice Hall, Englewood Cliffs, N.J.
- Aho, A. V., and J. D. Ullman [1973]. *The Theory of Parsing, Translation and Compiling*, Vol. II: *Compiling*, Prentice Hall, Englewood Cliffs, N.J.
- Aho, A. V., and J. D. Ullman [1977]. *Principles of Compiler Design*, Addison-Wesley, Reading, Mass.
- Arbib, M. A. [1970]. *Theories of Abstract Automata*, Prentice Hall, Englewood Cliffs, N.J.

- Arden, D. N. [1960]. "Delayed logic and finite state machines," *Theory of Computing Machine Design*, pp. 1-35, Univ. of Michigan Press, Ann Arbor, Mich.
- Axt, P. [1959]. "On a subrecursive hierarchy and primitive recursive degrees," *Trans. AMS* **92**, 85-105.
- Backus, J. W. [1959]. "The syntax and semantics of the proposed international algebraic language of the Zürich ACM-GAMM conference," *Proc. Intl. Conf. on Information Processing*, pp. 125-132, UNESCO.
- Baker, B. S., and R. V. Book [1974]. "Reversal bounded multipushdown machines," *J. Computer and Systems Sciences* **8**: 3, 315-332.
- Baker, T., J. Gill, and R. Solovay [1975]. "Relativizations of the  $P = ?NP$  question," *SIAM J. Computing* **4**: 4, 431-442.
- Bar-Hillel, Y., M. Perles, and E. Shamir [1961]. "On formal properties of simple phrase structure grammars," *Z. Phonetik. Sprachwiss. Kommunikationsforsch.* **14**, 143-172.
- Bauer, M., D. Brand, M. J. Fischer, A. R. Meyer, and M. S. Paterson [1973]. "A note on disjunctive form tautologies," *SIGACT News* **5**: 2, 17-20.
- Beeri, C. [1975]. "Two-way nested stack automata are equivalent to two-way stack automata," *J. Computer and Systems Sciences* **10**: 3, 317-339.
- Beeri, C. [1976]. "An improvement on Valiant's decision procedure for equivalence of deterministic finite-turn pushdown automata," *Theoretical Computer Science* **3**: 3, 305-320.
- Berman, L. [1977]. "Precise bounds for Presburger arithmetic and the reals with addition," *Proc. Eighteenth Annual IEEE Symposium on Foundations of Computer Science*, pp. 95-99.
- Berman, L., and J. Hartmanis [1977]. "On isomorphisms and density of NP and other complete sets," *SIAM J. Computing* **6**: 2, 305-322.
- Berman, P. [1978]. "Relationship between density and deterministic complexity of NP-complete languages," *Fifth International Symposium on Automata, Languages, and Programming*, Udine, Italy.
- Bird, M. [1973]. "The equivalence problem for deterministic two-tape automata," *J. Computer and Systems Sciences* **7**: 2, 218-236.
- Blattner, M. [1973]. "The unsolvability of the equality problem for sentential forms of context-free grammars," *J. Computer and Systems Sciences* **7**: 5, 463-468.
- Blum, M. [1967]. "A machine-independent theory of the complexity of recursive functions," *J. ACM* **14**: 2, 322-336.
- Blum, M. [1971]. "On effective procedures for speeding up algorithms," *J. ACM* **18**: 2, 290-305.
- Boasson, L. [1973]. "Two iteration theorems for some families of languages," *J. Computer and Systems Sciences* **7**: 6, 583-596.
- Book, R. V. [1972]. "On languages accepted in polynomial time," *SIAM J. Computing* **1**: 4, 281-287.
- Book, R. V. [1974]. "Comparing complexity classes," *J. Computer and Systems Sciences* **9**: 2, 213-229.
- Book, R. V. [1976]. "Translational lemmas, polynomial time, and  $(\log n)^3$  space," *Theoretical Computer Science* **1**: 3, 215-226.
- Book, R. V., and S. A. Greibach [1970]. "Quasi-realtime languages," *Math. Systems Theory* **4**: 2, 97-111.
- Book, R. V., S. A. Greibach, and B. Wegbreit [1970]. "Time- and tape-bounded Turing acceptors and AFL's," *J. Computer and Systems Sciences* **4**: 6, 606-621.

- Borodin, A. [1972]. "Computational complexity and the existence of complexity gaps," *J. ACM* **19**: 1, 158-174.
- Borosh, I., and L. B. Treybig [1976]. "Bounds on positive integral solutions of linear Diophantine equations," *Proc. AMS* **55**, 299-304.
- Brainerd, W. S., and L. H. Landweber [1974]. *Theory of Computation*, John Wiley and Sons, New York.
- Bruno, J. L., and R. Sethi [1976]. "Code generation for a one-register machine," *J. ACM* **23**: 3, 502-510.
- Bruss, A. R., and A. R. Meyer [1978]. "On time-space classes and their relation to the theory of real addition," *Proc. Tenth Annual ACM Symposium on the Theory of Computing*, pp. 233-239.
- Brzozowski, J. A. [1962]. "A survey of regular expressions and their applications," *IEEE Trans. on Electronic Computers* **11**: 3, 324-335.
- Brzozowski, J. A. [1964]. "Derivatives of regular expressions," *J. ACM* **11**: 4, 481-494.
- Bullen, R. H., Jr., and J. K. Millen [1972]. "Microtext—the design of a microprogrammed finite-state search machine for full text retrieval," *Proc. 1972 Fall Joint Computer Conference*, pp. 479-488, AFIPS Press, Montvale, N.J.
- Cantor, D. C. [1962]. "On the ambiguity problem of Backus systems," *J. ACM* **9**: 4, 477-479.
- Cardoza, E., R. J. Lipton, and A. R. Meyer [1976]. "Exponential space complete problems for Petri nets and commutative semi-groups: preliminary report," *Proc. Eighth Annual ACM Symposium on the Theory of Computing*, pp. 50-54.
- Chandler, W. J. [1969]. "Abstract families of deterministic languages," *Proc. First Annual ACM Symposium on the Theory of Computing*, pp. 21-30.
- Chandra, A. K., and L. J. Stockmeyer [1976]. "Alternation," *Proc. Seventeenth Annual IEEE Symposium on Foundations of Computer Science*, pp. 98-108.
- Chomsky, N. [1956]. "Three models for the description of language," *IRE Trans. on Information Theory* **2**: 3, 113-124.
- Chomsky, N. [1959]. "On certain formal properties of grammars," *Information and Control* **2**: 2, 137-167.
- Chomsky, N. [1962]. "Context-free grammars and pushdown storage," *Quarterly Prog. Rept. No. 65*, pp. 187-194, MIT Res. Lab. Elect., Cambridge, Mass.
- Chomsky, N. [1963]. "Formal properties of grammars," *Handbook of Math. Psych.*, Vol. 2, pp. 323-418, John Wiley and Sons, New York.
- Chomsky, N., and G. A. Miller [1958]. "Finite state languages," *Information and Control* **1**: 2, 91-112.
- Chomsky, N., and M. P. Schützenberger [1963]. "The algebraic theory of context free languages," *Computer Programming and Formal Systems*, pp. 118-161, North Holland, Amsterdam.
- Christofides, N. [1976]. "Worst case analysis of a new heuristic for the traveling salesman problem," *Algorithms and Complexity: New Directions and Recent Results* (J. Traub, ed.), p. 341, Academic Press, New York.
- Church, A. [1936]. "An unsolvable problem of elementary number theory," *Amer. J. Math.* **59**, 345-363.
- Church, A. [1941]. "The Calculi of Lambda-Conversion," *Annals of Mathematics Studies* **6**, Princeton Univ. Press, Princeton, N.J.

- Cobham, A. [1964]. "The intrinsic computational difficulty of functions," *Proc. 1964 Congress for Logic, Mathematics, and Philosophy of Science*, pp. 24–30, North Holland, Amsterdam.
- Coffman, E. G., Jr. (ed.) [1976]. *Computer and Job Shop Scheduling Theory*, John Wiley and Sons, New York.
- Cole, S. N. [1969]. "Pushdown store machines and real-time computation," *Proc. First Annual ACM Symposium on the Theory of Computing*, pp. 233–246.
- Constable, R. L. [1972]. "The operator gap," *J. ACM* 19: 1, 175–183.
- Conway, J. H. [1971]. *Regular Algebra and Finite Machines*, Chapman and Hall, London.
- Cook, S. A. [1971a]. "Characterizations of pushdown machines in terms of time-bounded computers," *J. ACM* 18: 1, 4–18.
- Cook, S. A. [1971b]. "The complexity of theorem proving procedures," *Proc. Third Annual ACM Symposium on the Theory of Computing*, pp. 151–158.
- Cook, S. A. [1971c]. "Linear time simulation of deterministic two-way pushdown automata," *Proc. 1971 IFIP Congress*, pp. 75–80, North Holland, Amsterdam.
- Cook, S. A. [1973a]. "A hierarchy for nondeterministic time complexity," *J. Computer and Systems Sciences* 7: 4, 343–353.
- Cook, S. A. [1973b]. "An observation on time-storage trade off," *Proc. Fifth Annual ACM Symposium on the Theory of Computing*, pp. 29–33.
- Cook, S. A., and R. A. Reckhow [1973]. "Time bounded random access machines," *J. Computer and Systems Sciences* 7: 4, 354–375.
- Cook, S. A., and R. Sethi [1976]. "Storage requirements for deterministic polynomial time recognizable languages," *J. Computer and Systems Sciences* 13: 1, 25–37.
- Cooper, C. D. [1972]. "Theorem proving in arithmetic without multiplication," *Machine Intelligence* 7 (Melzer and Mitchie, eds.), pp. 91–99, John Wiley and Sons, New York.
- Cremers, A., and S. Ginsburg [1975]. "Context-free grammar forms," *J. Computer and Systems Sciences* 11: 1, 86–117.
- Cudia, D. F. [1970]. "General problems of formal grammars," *J. ACM* 17: 1, 31–43.
- Cudia, D. F., and W. E. Singletary [1968]. "Degrees of unsolvability in formal grammars," *J. ACM* 15: 4, 680–692.
- Davis, M. [1958]. *Computability and Unsolvability*, McGraw-Hill, New York.
- Davis, M. (ed.) [1965]. *The Undecidable*, Raven Press, New York.
- De Remer, F. L. [1969]. "Generating parsers for BNF grammars," *Proc. 1969 Spring Joint Computer Conference*, pp. 793–799, AFIPS Press, Montvale, N.J.
- De Remer, F. L. [1971]. "Simple LR( $k$ ) grammars," *Comm. ACM* 14: 7, 453–460.
- Earley, J. [1970]. "An efficient context-free parsing algorithm," *Comm. ACM* 13: 2, 94–102.
- Eilenberg, S., and C. C. Elgot [1970]. *Recursiveness*, Academic Press, New York.
- Even, S., and R. E. Tarjan [1976]. "A combinatorial problem which is complete in polynomial space," *J. ACM* 23: 4, 710–719.
- Evey, J. [1963]. "Application of pushdown store machines," *Proc. 1963 Fall Joint Computer Conference*, pp. 215–227, AFIPS Press, Montvale, N.J.
- Ferrante, J., and C. Rackoff [1975]. "A decision procedure for the first order theory of real addition with order," *SIAM J. Computing* 4: 1, 69–76.
- Fischer, M. J. [1968]. "Grammars with macro-like productions," *Proc. Ninth Annual IEEE Symposium on Switching and Automata Theory*, pp. 131–142.
- Fischer, M. J. [1969]. "Two characterizations of the context-sensitive languages," *Proc. Tenth Annual IEEE Symposium on Switching and Automata Theory*, pp. 157–165.
- Fischer, M. J., and M. O. Rabin [1974]. "Super-exponential complexity of Presburger arithmetic," *Complexity of Computation* (R. M. Karp, ed.). Proceedings of SIAM-AMS Symposium in Applied Mathematics.
- Fischer, P. C. [1963]. "On computability by certain classes of restricted Turing machines," *Proc. Fourth Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, pp. 23–32.
- Fischer, P. C. [1965]. "On formalisms for Turing machines," *J. ACM* 12: 4, 570–588.
- Fischer, P. C. [1966]. "Turing machines with restricted memory access," *Information and Control* 9: 4, 364–379.
- Fischer, P. C., A. R. Meyer, and A. L. Rosenberg [1968]. "Counter machines and counter languages," *Math. Systems Theory* 2: 3, 265–283.
- Fischer, P. C., A. R. Meyer, and A. L. Rosenberg [1972]. "Real-time simulation of multihead tape units," *J. ACM* 19: 4, 590–607.
- Floyd, R. W. [1962a]. "On ambiguity in phrase structure languages," *Comm. ACM* 5: 10, 526–534.
- Floyd, R. W. [1962b]. "On the nonexistence of a phrase structure grammar for ALGOL 60," *Comm. ACM* 5: 9, 483–484.
- Floyd, R. W. [1964]. "New proofs and old theorems in logic and formal linguistics," Computer Associates Inc., Wakefield, Mass.
- Floyd, R. W. [1967]. "Nondeterministic algorithms," *J. ACM* 14: 4, 636–644.
- Freedman, A. R., and R. E. Ladner [1975]. "Space bounds for processing contentless inputs," *J. Computer and Systems Sciences* 11: 1, 118–128.
- Friedman, A. [1975]. *Logical Design of Digital Systems*, Computer Science Press, Potomac, Md.
- Friedman, E. P. [1976]. "The inclusion problem for simple languages," *Theoretical Computer Science* 1: 4, 297–316.
- Friedman, E. P. [1977]. "The equivalence problem for deterministic context-free languages and monadic recursion schemes," *J. Computer and Systems Sciences* 14: 3, 344–359.
- Gabriellian, A., and S. Ginsburg [1974]. "Grammar schemata," *J. ACM* 21: 2, 312–326.
- Garey, M. R., R. L. Graham, and D. S. Johnson [1976]. "Some NP-complete geometric problems," *Proc. Eighth Annual ACM Symposium on the Theory of Computing*, pp. 10–22.
- Garey, M. R., and D. S. Johnson [1976]. "The complexity of near-optimal graph coloring," *J. ACM* 23: 1, 43–49.
- Garey, M. R., and D. S. Johnson [1978]. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, H. Freeman, San Francisco.
- Garey, M. R., D. S. Johnson, and L. J. Stockmeyer [1976]. "Some simplified NP-complete problems," *Theoretical Computer Science* 1: 3, 237–267.
- Garey, M. R., D. S. Johnson, and R. E. Tarjan [1976]. "The planar Hamilton circuit problem is NP-complete," *SIAM J. Computing* 5: 4, 704–714.
- Gathen, J., and M. Sieveking [1976]. "A bound on the solutions of linear integer programs," Unpublished notes.
- Ginsburg, S. [1962]. "Examples of abstract machines," *IEEE Trans. on Electronic Computers* 11: 2, 132–135.

- Ginsburg, S. [1966]. *The Mathematical Theory of Context-Free Languages*, McGraw Hill, New York.
- Ginsburg, S. [1975]. *Algebraic and Automata-Theoretic Properties of Formal Languages*, North Holland, Amsterdam.
- Ginsburg, S., and S. A. Greibach [1966a]. "Deterministic context-free languages," *Information and Control* 9: 6, 563–582.
- Ginsburg, S., and S. A. Greibach [1966b]. "Mappings which preserve context-sensitive languages," *Information and Control* 9: 6, 563–582.
- Ginsburg, S., and S. A. Greibach [1969]. "Abstract families of languages," *Studies in Abstract Families of Languages*, pp. 1–32, Memoir No. 87, American Mathematical Society, Providence, R.I.
- Ginsburg, S., and S. A. Greibach [1970]. "Principal AFL," *J. Computer and Systems Sciences* 4: 3, 308–338.
- Ginsburg, S., S. A. Greibach, and M. A. Harrison [1967a]. "Stack automata and compiling," *J. ACM* 14: 1, 172–201.
- Ginsburg, S., S. A. Greibach, and M. A. Harrison [1967b]. "One-way stack automata," *J. ACM* 14: 2, 389–418.
- Ginsburg, S., and J. E. Hopcroft [1970]. "Two-way balloon automata and AFL," *J. ACM* 17: 1, 3–13.
- Ginsburg, S., and H. G. Rice [1962]. "Two families of languages related to ALGOL," *J. ACM* 9: 3, 350–371.
- Ginsburg, S., and G. F. Rose [1963a]. "Some recursively unsolvable problems in ALGOL-like languages," *J. ACM* 10: 1, 29–47.
- Ginsburg, S., and G. F. Rose [1963b]. "Operations which preserve definability in languages," *J. ACM* 10: 2, 175–195.
- Ginsburg, S., and G. F. Rose [1966]. "Preservation of languages by transducers," *Information and Control* 9: 2, 153–176.
- Ginsburg, S., and E. H. Spanier [1963]. "Quotients of context free languages," *J. ACM* 10: 4, 487–492.
- Ginsburg, S., and E. H. Spanier [1966]. "Finite turn pushdown automata," *SIAM J. Control*, 4: 3, 429–453.
- Ginsburg, S., and J. S. Ullian [1966a]. "Ambiguity in context-free languages," *J. ACM* 13: 1, 62–88.
- Ginsburg, S., and J. S. Ullian [1966b]. "Preservation of unambiguity and inherent ambiguity in context free languages," *J. ACM* 13: 3, 364–368.
- Graham, S. L. [1970]. "Extended precedence languages, bounded right context languages and deterministic languages," *Proc. Eleventh Annual IEEE Symposium on Switching and Automata Theory*, pp. 175–180.
- Graham, S. L., M. A. Harrison, and W. L. Ruzzo [1976]. "On-line context-free language recognition in less than cubic time," *Proc. Eighth Annual ACM Symposium on the Theory of Computing*, pp. 112–120.
- Gray, J. N., M. A. Harrison, and O. Ibarra [1967]. "Two-way pushdown automata," *Information and Control* 11: 1–2, 30–70.
- Greibach, S. A. [1963]. "The undecidability of the ambiguity problem for minimal linear grammars," *Information and Control* 6: 2, 117–125.
- Greibach, S. A. [1965]. "A new normal form theorem for context-free phrase structure grammars," *J. ACM* 12: 1, 42–52.
- Greibach, S. A. [1966]. "The unsolvability of the recognition of linear context-free languages," *J. ACM* 13: 4, 582–587.
- Greibach, S. A. [1968]. "A note on undecidable properties of formal languages," *Math Systems Theory* 2: 1, 1–6.
- Greibach, S. A. [1970]. "Full AFL's and nested iterated substitution," *Information and Control* 16: 1, 7–35.
- Greibach, S. A. [1973]. "The hardest context-free language," *SIAM J. Computing* 2: 4, 304–310.
- Greibach, S. A., and J. E. Hopcroft [1969]. "Independence of AFL operations," *Studies in Abstract Families of Languages*, pp. 33–40, Memoir No. 87, American Mathematical Society, Providence, R.I.
- Greibach, S. A., and J. E. Hopcroft [1969]. "Scattered context grammars," *J. Computer and Systems Sciences* 3: 3, 233–247.
- Griffiths, T. V. [1968]. "The unsolvability of the equivalence problem for  $\Lambda$ -free nondeterministic generalized machines," *J. ACM* 15: 3, 409–413.
- Gross, M. [1964]. "Inherent ambiguity of minimal linear grammars," *Information and Control* 7: 3, 366–368.
- Grzegorczyk, A. [1953]. "Some classes of recursive functions," *Rosprawy Matematyczne* 4, Instytut Matematyczny Polskiej Akademie Nauk, Warsaw, Poland.
- Haines, L. [1965]. "Generation and recognition of formal languages," Ph.D. thesis, MIT, Cambridge, Mass.
- Hardy, G. H., and E. M. Wright [1938]. *An Introduction to the Theory of Numbers*, Oxford Univ. Press, London.
- Hartmanis, J. [1967]. "Context-free languages and Turing machine computations," *Proc. Symposia in Applied Math.* 19, American Mathematical Society, Providence, R.I.
- Hartmanis, J. [1968]. "Computational complexity of one-tape Turing machine computations," *J. ACM* 15: 2, 325–339.
- Hartmanis, J. [1969]. "On the complexity of undecidable problems in automata theory," *J. ACM* 16: 1, 160–167.
- Hartmanis, J., and J. E. Hopcroft [1968]. "Structure of undecidable problems in automata theory," *Proc. Ninth Annual IEEE Symposium on Switching and Automata Theory*, pp. 327–333.
- Hartmanis, J., and J. E. Hopcroft [1971]. "An overview of the theory of computational complexity," *J. ACM* 18: 3, 444–475.
- Hartmanis, J., and Hopcroft, J. E. [1976]. "Independence results in computer science," *SIGACT News* 8: 4, 13–23.
- Hartmanis, J., P. M. Lewis II, and R. E. Stearns [1965]. "Hierarchies of memory limited computations," *Proc. Sixth Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, pp. 179–190.
- Hartmanis, J., and H. Shank [1968]. "On the recognition of primes by automata," *J. ACM* 15: 3, 382–389.
- Hartmanis, J., and R. E. Stearns [1965]. "On the computational complexity of algorithms," *Trans. AMS* 117, 285–306.
- Hayashi, T. [1973]. "On derivation trees of indexed grammars—an extension of the  $uvwx$  theorem," *Publications of the Research Institute for Mathematical Sciences* 9: 1, pp. 61–92.

- Hennie, F. C. [1964]. "Fault detecting experiments for sequential circuits," *Proc. Fourth Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, pp. 95-110.
- Hennie, F. C. [1965]. "One-tape off-line Turing machine computations," *Information and Control* 8: 6, 553-578.
- Hennie, F. C. [1977]. *Introduction to Computability*, Addison-Wesley, Reading, Mass.
- Hennie, F. C., and R. E. Stearns [1966]. "Two-tape simulation of multitape Turing machines," *J. ACM* 13: 4, 533-546.
- Herman, G. T. [1974]. "Closure properties of some families of languages associated with biological systems," *Information and Control* 24: 2, 101-121.
- Herman, G. T., and G. Rozenberg [1975]. *Developmental Systems and Languages*, North Holland, Amsterdam.
- Hibbard, T. N. [1974]. "Context-limited grammars," *J. ACM* 21: 3, 446-453.
- Hogben, L. [1955]. *The Wonderful World of Mathematics*, Garden City Books, Garden City, N.Y.
- Hopcroft, J. E. [1971]. "An  $n \log n$  algorithm for minimizing the states in a finite automaton," *The Theory of Machines and Computations* (Z. Kohavi, ed.), pp. 189-196, Academic Press, New York.
- Hopcroft, J. E., W. J. Paul, and L. G. Valiant [1975]. "On time versus space and related problems," *Proc. Sixteenth Annual IEEE Symposium on Foundations of Computer Science*, pp. 57-64.
- Hopcroft, J. E., and J. D. Ullman [1967a]. "Nonerasing stack automata," *J. Computer and Systems Sciences* 1: 2, 166-186.
- Hopcroft, J. E., and J. D. Ullman [1967b]. "An approach to a unified theory of automata," *Bell System Technical J.* 46: 8, 1763-1829.
- Hopcroft, J. E., and J. D. Ullman [1968a]. "Decidable and undecidable questions about automata," *J. ACM* 15: 2, 317-324.
- Hopcroft, J. E., and J. D. Ullman [1968b]. "Deterministic stack automata and the quotient operator," *J. Computer and Systems Sciences* 2: 1, 1-12.
- Hopcroft, J. E., and J. D. Ullman [1968c]. "Sets accepted by one-way stack automata are context sensitive," *Information and Control* 13: 2, 114-133.
- Hopcroft, J. E., and J. D. Ullman [1969a]. "Some results on tape-bounded Turing machines," *J. ACM* 16: 1, 168-177.
- Hopcroft, J. E., and J. D. Ullman [1969b]. *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, Mass.
- Huffman, D. A. [1954]. "The synthesis of sequential switching circuits," *J. Franklin Institute* 257: 3-4, 161-190, 275-303.
- Hunt, H. B., III [1973]. "On the time and tape complexity of languages," *Proc. Fifth Annual ACM Symposium on the Theory of Computing*, pp. 10-19.
- Hunt, H. B., III, and D. J. Rosenkrantz [1974]. "Computational parallels between the regular and context-free languages," *Proc. Sixth Annual ACM Symposium on the Theory of Computing*, pp. 64-74.
- Hunt, H. B., III, and D. J. Rosenkrantz [1977]. "On equivalence and containment problems for formal languages," *J. ACM* 24: 3, 387-396.
- Hunt, H. B., III, D. J. Rosenkrantz, and T. G. Szymanski [1976]. "On the equivalence, containment and covering problems for regular expressions," *J. Computer and Systems Sciences* 12: 2, 222-268.
- Hunt, H. B., III, and T. G. Szymanski [1975]. "On the complexity of grammar and related problems," *Proc. Seventh Annual ACM Symposium on the Theory of Computing*, pp. 54-65.
- Hunt, H. B., III, and T. G. Szymanski [1976]. "Complexity metatheorems for context-free grammar problems," *J. Computer and Systems Sciences* 13: 3, 318-334.
- Hunt, H. B., III, T. G. Szymanski, and J. D. Ullman [1975]. "On the complexity of LR( $k$ ) testing," *Comm. ACM* 18: 12, 707-715.
- Ibarra, O. H. [1972]. "A note concerning nondeterministic tape complexities," *J. ACM* 19: 4, 608-612.
- Ibarra, O. H. [1977]. "The unsolvability of the equivalence problem for free GSM's with unary input (output) alphabet," *Proc. Eighteenth Annual IEEE Symposium on Foundations of Computer Science*, pp. 74-81.
- Johnson, D. S. [1974]. "Approximation algorithms for combinatorial problems," *J. Computer and Systems Sciences* 9: 3, 256-278.
- Johnson, S. C. [1974]. "YACC—yet another compiler compiler," CSTR 32, Bell Laboratories, Murray Hill, N.J.
- Johnson, W. L., J. H. Porter, S. I. Ackley, and D. T. Ross [1968]. "Automatic generation of efficient lexical analyzers using finite state techniques," *Comm. ACM* 11: 12, 805-813.
- Jones, N. D. [1975]. "Space-bounded reducibility among combinatorial problems," *J. Computer and Systems Sciences* 11: 1, 68-85.
- Jones, N. D. [1973]. *Computability Theory: an Introduction*, Academic Press, New York.
- Jones, N. D., and W. T. Laaser [1976]. "Complete problems for deterministic polynomial time," *Theoretical Computer Science* 3: 1, 105-118.
- Jones, N. D., E. Lien, and W. T. Lasser [1976]. "New problems complete for non-deterministic log space," *Math. Systems Theory* 10: 1, 1-17.
- Jones, N. D., and S. S. Muchnick [1977]. "Even simple programs are hard to analyze," *J. ACM* 24: 2, 338-350.
- Kannan, R., and C. L. Monma [1977]. "On the computational complexity of integer programming problems," Report 7780-OR, Inst. für Operations Research, Univ. Bonn, Bonn, West Germany.
- Karp, R. M. [1972]. "Reducibility among combinatorial problems," *Complexity of Computer Computations*, pp. 85-104, Plenum Press, N.Y.
- Karp, R. M. [1977]. "The probabilistic analysis of some combinatorial search algorithms," *Algorithms and Complexity: New Directions and Recent Results* (J. Traub, ed.), pp. 1-20, Academic Press, New York.
- Kasami, T. [1965]. "An efficient recognition and syntax algorithm for context-free languages," *Scientific Report AFCRL-65-758*, Air Force Cambridge Research Lab., Bedford, Mass.
- Kasami, T., and K. Torii [1969]. "A syntax analysis procedure for unambiguous context-free grammars," *J. ACM* 16: 3, 423-431.
- Kirkpatrick, D. G., and P. Hell [1978]. "On the completeness of a generalized matching problem," *Proc. Tenth Annual ACM Symposium on the Theory of Computing*, pp. 240-245.
- Kleene, S. C. [1936]. "General recursive functions of natural numbers," *Mathematische Annalen* 112, 727-742.

- Kleene, S. C. [1952]. *Introduction to Metamathematics*, D. Van Nostrand, Princeton, N.J.
- Kleene, S. C. [1956]. "Representation of events in nerve nets and finite automata," *Automata Studies*, pp. 3-42, Princeton Univ. Press, Princeton, N.J.
- Knuth, D. E. [1965]. "On the translation of languages from left to right," *Information and Control* 8: 6, 607-639.
- Knuth, D. E., J. H. Morris, Jr., and V. R. Pratt [1977]. "Fast pattern matching in strings," *SIAM J. Computing* 6: 2, 323-350.
- Kohavi, Z. [1970]. *Switching and Finite Automata Theory*, McGraw-Hill, New York.
- Korenjak, A. J. [1969]. "A practical method for constructing LR( $k$ ) processors," *Comm. ACM* 12: 11, 613-623.
- Korenjak, A. J., and J. E. Hopcroft [1966]. "Simple deterministic languages," *Proc. Seventh Annual IEEE Symposium on Switching and Automata Theory*, pp. 36-46.
- Kosaraju, S. R. [1974]. "Regularity preserving functions," *SIGACT News* 6: 2, 16-17.
- Kosaraju, S. R. [1975]. "Context free preserving functions," *Math. Systems Theory* 9: 3, 193-197.
- Kozen, D. [1976]. "On parallelism in Turing machines," *Proc. Seventeenth Annual IEEE Symposium on Foundations of Computer Science*, pp. 89-97.
- Kozen, D. [1978]. "Indexing of subrecursive classes," *Proc. Tenth Annual ACM Symposium on the Theory of Computing*, pp. 287-295.
- Kuroda, S. Y. [1964]. "Classes of languages and linear bounded automata," *Information and Control* 7: 2, 207-223.
- Ladner, R. E. [1975a]. "On the structure of polynomial time reducibility," *J. ACM* 22: 1, 155-171.
- Ladner, R. E. [1975b]. "The circuit value problem is log-space complete for  $P$ ," *SIGACT News* 7: 1, 18-20.
- Ladner, R. E., N. Lynch, and A. Selman [1974]. "Comparison of polynomial time reducibilities," *Proc. Sixth Annual ACM Symposium on the Theory of Computing*, pp. 110-121.
- Landweber, P. S. [1963]. "Three theorems on phrase structure grammars of type 1," *Information and Control* 6: 2, 131-136.
- Landweber, P. S. [1964]. "Decision problems of phrase structure grammars," *IEEE Trans. on Electronic Computers* 13, 354-362.
- Leong, B., and J. Seiferas [1977]. "New real-time simulations of multihead tape units," *Proc. Ninth Annual ACM Symposium on the Theory of Computing*, pp. 239-240.
- Lesk, M. E. [1975]. "LEX—a lexical analyzer generator," CSTR 39, Bell Laboratories, Murray Hill, N.J.
- Levin, L. A. [1973]. "Universal sorting problems," *Problemi Peredachi Informatsii* 9: 3, 265-266.
- Lewis, J. M. [1978]. "On the complexity of the maximum subgraph problem," *Proc. Tenth Annual ACM Symposium on the Theory of Computing*, pp. 265-274.
- Lewis, P. M., II, D. J. Rosenkrantz, and R. E. Stearns [1976]. *Compiler Design Theory*, Addison-Wesley, Reading, Mass.
- Lewis, P. M., II, and R. E. Stearns [1968]. "Syntax directed transduction," *J. ACM* 15: 3, 465-488.
- Lewis, P. M., II, R. E. Stearns, and J. Hartmanis [1965]. "Memory bounds for recognition of context-free and context-sensitive languages," *Proc. Sixth Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, pp. 191-202.

- Lindenmayer, A. [1968]. "Mathematical models for cellular interactions in development, parts I and II," *J. Theor. Biol.* 18, 280-315.
- Lindenmayer, A. [1971]. "Developmental systems without cellular interaction, their languages and grammars," *J. Theor. Biol.* 30, 455-484.
- Machtey, M., and P. R. Young [1978]. *An Introduction to the General Theory of Algorithms*, North Holland, New York.
- Mager, G. [1969]. "Writing pushdown acceptors," *J. Computer and Systems Sciences* 3: 3, 276-319.
- Maibaum, T. S. E. [1974]. "A generalized approach to formal languages," *J. Computer and Systems Sciences* 8: 3, 409-439.
- McCreight, E. M., and A. R. Meyer [1969]. "Classes of computable functions defined by bounds on computation," *Proc. First Annual ACM Symposium on the Theory of Computing*, pp. 79-88.
- McCulloch, W. S., and W. Pitts [1943]. "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophysics* 5, 115-133.
- McNaughton, R., and H. Yamada [1960]. "Regular expressions and state graphs for automata," *IEEE Trans. on Electronic Computers* 9: 1, 39-47.
- Mealy, G. H. [1955]. "A method for synthesizing sequential circuits," *Bell System Technical J.* 34: 5, 1045-1079.
- Meyer, A. R., and R. Ritchie [1967]. "The complexity of loop programs," *Proc. ACM Natl. Conf.*, pp. 465-469.
- Meyer, A. R., and L. J. Stockmeyer [1973]. "The equivalence problem for regular expressions with squaring requires exponential space," *Proc. Thirteenth Annual IEEE Symposium on Switching and Automata Theory*, pp. 125-129.
- Miller, G. L. [1976]. "Riemann's hypothesis and tests for primality," *J. Computer and Systems Sciences* 13: 3, 300-317.
- Minsky, M. L. [1961]. "Recursive unsolvability of Post's problem of 'tag' and other topics in the theory of Turing machines," *Annals of Math.*, 74: 3, 437-455.
- Minsky, M. L. [1967]. *Computation: Finite and Infinite Machines*, Prentice Hall, Englewood Cliffs, N.J.
- Minsky, M. L., and S. Papert [1966]. "Unrecognizable sets of numbers," *J. ACM* 13: 2, 281-286.
- Moore, E. F. [1956]. "Gedanken experiments on sequential machines," *Automata Studies*, pp. 129-153, Princeton Univ. Press, Princeton, N.J.
- Moore, E. F. (ed.) [1964]. *Sequential Machines: Selected Papers*, Addison-Wesley, Reading, Mass.
- Myhill, J. [1957]. "Finite automata and the representation of events," WADD TR-57-624, pp. 112-137, Wright Patterson AFB, Ohio.
- Myhill, J. [1960]. "Linear bounded automata," WADD TR-60-165, pp. 60-165, Wright Patterson AFB, Ohio.
- Naur, P. et al. [1960]. "Report on the algorithmic language ALGOL 60," *Comm. ACM* 3: 5, 299-314, revised in *Comm. ACM* 6: 1, 1-17.
- Nerode, A. [1958]. "Linear automaton transformations," *Proc. AMS*, 9, pp. 541-544.

- Oettinger, A. G. [1961]. "Automatic syntactic analysis and the pushdown store," *Proc. Symposia in Applied Math.* **12**, American Mathematical Society, Providence, R.I.
- Ogden, W. [1968]. "A helpful result for proving inherent ambiguity," *Math. Systems Theory* **2**: 3, 191–194.
- Ogden, W. [1969]. "Intercalation theorems for stack languages," *Proc. First Annual ACM Symposium on the Theory of Computing*, pp. 31–42.
- Oppen, D. C. [1973]. "Elementary bounds for Presburger arithmetic," *Proc. Fifth Annual ACM Symposium on the Theory of Computing*, pp. 34–37.
- Papadimitriou, C. H. [1976]. "On the complexity of edge traversing," *J. ACM* **23**: 3, 544–554.
- Papadimitriou, C. H., and K. Steiglitz [1977]. "On the complexity of local search for the traveling salesman problem," *SIAM J. Computing* **6**: 1, 76–83.
- Parikh, R. J. [1966]. "On context-free languages," *J. ACM* **13**: 4, 570–581.
- Paull, M. C., and S. H. Unger [1968]. "Structural equivalence of context-free grammars," *J. Computer and Systems Sciences* **2**: 4, 427–468.
- Paul, W. J. [1977]. "On time hierarchies," *Proc. Ninth Annual ACM Symposium on the Theory of Computing*, pp. 218–222.
- Paul, W. J., R. E. Tarjan, and J. R. Celoni [1976]. "Space bounds for a game on graphs," *Proc. Eighth Annual ACM Symposium on the Theory of Computing*, pp. 149–160.
- Plaisted, D. A. [1977]. "Sparse complex polynomials and polynomial reducibility," *J. Computer and Systems Sciences* **14**: 2, 210–221.
- Post, E. [1936]. "Finite combinatory processes-formulation, I," *J. Symbolic Logic*, **1**, 103–105.
- Post, E. [1943]. "Formal reductions of the general combinatorial decision problem," *Amer. J. Math.* **65**, 197–215.
- Post, E. [1946]. "A variant of a recursively unsolvable problem," *Bull. AMS*, **52**, 264–268.
- Pratt, V. R. [1975]. "Every prime has a succinct certificate," *SIAM J. Computing* **4**: 3, 214–220.
- Pratt, V. R., and L. J. Stockmeyer [1976]. "A characterization of the power of vector machines," *J. Computer and Systems Sciences* **12**: 2, 198–221.
- Rabin, M. O. [1963]. "Real-time computation," *Israel J. Math.* **1**: 4, 203–211.
- Rabin, M. O. [1976]. "Probabilistic algorithms," *Algorithms and Complexity: New Directions and Recent Results* (J. Traub, ed.), pp. 21–40, Academic Press, New York.
- Rabin, M. O., and D. Scott [1959]. "Finite automata and their decision problems," *IBM J. Res.* **3**: 2, 115–125.
- Rackoff, C. [1978]. "Relativized questions involving probabilistic algorithms," *Proc. Tenth Annual ACM Symposium on the Theory of Computing*, pp. 338–342.
- Reedy, A., and W. J. Savitch [1975]. "The Turing degree of the inherent ambiguity problem for context-free languages," *Theoretical Computer Science* **1**: 1, 77–91.
- Rice, H. G. [1953]. "Classes of recursively enumerable sets and their decision problems," *Trans. AMS* **89**, 25–59.
- Rice, H. G. [1956]. "On completely recursively enumerable classes and their key arrays," *J. Symbolic Logic* **21**, 304–341.
- Ritchie, R. W. [1963]. "Classes of predictably computable functions," *Trans. AMS* **106**, 139–173.
- Rogers, H., Jr. [1967]. *The Theory of Recursive Functions and Effective Computability*, McGraw-Hill, New York.
- Rosenkrantz, D. J. [1967]. "Matrix equations and normal forms for context-free grammars," *J. ACM* **14**: 3, 501–507.
- Rosenkrantz, D. J. [1969]. "Programmed grammars and classes of formal languages," *J. ACM* **16**: 1, 107–131.
- Rosenkrantz, D. J., and R. E. Stearns [1970]. "Properties of deterministic top-down grammars," *Information and Control* **17**: 3, 226–256.
- Rosenkrantz, D. J., R. E. Stearns, and P. M. Lewis, II [1977]. "An analysis of several heuristics for the traveling salesman problem," *SIAM J. Computing* **6**: 3, 563–581.
- Rounds, W. C. [1970]. "Mappings and grammars on trees," *Math. Systems Theory* **4**: 3, 257–287.
- Ruby, S., and P. C. Fischer [1965]. "Translational methods and computational complexity," *Proc. Sixth Annual IEEE Symp. on Switching Circuit Theory and Logical Design*, pp. 173–178.
- Sahni, S. [1974]. "Computationally related problems," *SIAM J. Computing* **3**: 4, 262–279.
- Sahni, S., and T. Gonzalez [1976]. "P-complete approximation problems," *J. ACM* **23**: 3, 555–565.
- Sakoda, W. J., and Sipser, M. [1978]. "Nondeterminism and the size of two-way finite automata," *Proc. Tenth Annual ACM Symposium on the Theory of Computing*, pp. 275–286.
- Salomaa, A. [1966]. "Two complete axiom systems for the algebra of regular events," *J. ACM* **13**: 1, 158–169.
- Salomaa, A. [1973]. *Formal Languages*, Academic Press, N.Y.
- Savitch, W. J. [1970]. "Relationships between nondeterministic and deterministic tape complexities," *J. Computer and Systems Sciences* **4**: 2, 177–192.
- Savitch, W. J. [1972]. "Maze recognizing automata," *Proc. Fourth Annual ACM Symposium on the Theory of Computing*, pp. 151–156.
- Schaefer, T. J. [1976]. "Complexity of decision problems based on finite two-person perfect information games," *Proc. Eighth Annual ACM Symposium on the Theory of Computing*, pp. 41–49.
- Schaefer, T. J. [1978]. "The complexity of satisfiability problems," *Proc. Tenth Annual ACM Symposium on the Theory of Computing*, pp. 216–226.
- Scheinberg, S. [1960]. "Note on the Boolean properties of context-free languages," *Information and Control* **3**: 4, 372–375.
- Schutzenberger, M. P. [1963]. "On context-free languages and pushdown automata," *Information and Control* **6**: 3, 246–264.
- Seiferas, J. I. [1974]. "A note on prefixes of regular languages," *SIGACT News* **6**: 1, 25–29.
- Seiferas, J. I. [1977a]. "Techniques for separating space complexity classes," *J. Computer and Systems Sciences* **14**: 1, 73–99.
- Seiferas, J. I. [1977b]. "Relating refined complexity classes," *J. Computer and Systems Sciences* **14**: 1, 100–129.
- Seiferas, J. I., M. J. Fischer, and A. R. Meyer [1973]. "Refinements of nondeterministic time and space hierarchies," *Proc. Fourteenth Annual IEEE Symposium on Switching and Automata Theory*, pp. 130–137.

- Seiferas, J. I., and R. McNaughton [1976]. "Regularity preserving relations," *Theoretical Computer Science* 2: 2, 147–154.
- Sethi, R. [1975]. "Complete register allocation problems," *SIAM J. Computing* 4: 3, 226–248.
- Shannon, C. E. [1956]. "A universal Turing machine with two internal states," *Automata Studies*, pp. 129–153, Princeton Univ. Press, Princeton, N.J.
- Shannon, C. E., and J. McCarthy (eds.) [1956]. *Automata Studies*, Princeton Univ. Press, Princeton, N.J.
- Sheperdson, J. C. [1959]. "The reduction of two-way automata to one-way automata," *IBM J. Res.* 3: 2, 198–200.
- Solovay, R., and V. Strassen [1977]. "A fast Monte Carlo test for primality," *SIAM J. Computing* 6: 1, 84–85. A correction *ibid.* 7: 1, p. 118.
- Springsteel, F. N. [1976]. "On the pre-AFL of  $[\log n]$  space and related families of languages," *Theoretical Computer Science* 2: 3, 295–304.
- Stanley, R. J. [1965]. "Finite state representations of context-free languages," *Quarterly Prog. Rept. No. 76*, 276–279, MIT Res. Lab. Elect., Cambridge, Mass.
- Stearns, R. E. [1967]. "A regularity test for pushdown machines," *Information and Control* 11: 3, 323–340.
- Stearns, R. E., and J. Hartmanis [1963]. "Regularity preserving modifications of regular expressions," *Information and Control* 6: 1, 55–69.
- Stockmeyer, L. J. [1973]. "Planar 3-colorability is polynomial complete," *SIGACT News* 5: 3, 19–25.
- Stockmeyer, L. J. [1974]. "The complexity of decision problems in automata theory and logic," MAC TR-133, Project MAC, MIT, Cambridge, Mass.
- Stockmeyer, L. J. [1976]. "The polynomial time hierarchy," *Theoretical Computer Science* 3: 1, 1–22.
- Stockmeyer, L. J., and A. R. Meyer [1973]. "Word problems requiring exponential space," *Proc. Fifth Annual ACM Symposium on the Theory of Computing*, pp. 1–9.
- Sudborough, I. H. [1975a]. "A note on tape-bounded complexity classes and linear context-free languages," *J. ACM* 22: 4, 499–500.
- Sudborough, I. H. [1975b]. "On tape-bounded complexity classes and multihead finite automata," *J. Computer and Systems Sciences* 10: 1, 62–76.
- Suzuki, N., and D. Jefferson [1977]. "Verification decidability of Presburger array programs," *A Conference on Theoretical Computer Science*, pp. 202–212, Univ. of Waterloo, Waterloo, Ont., Canada.
- Taniguchi, K., and T. Kasami [1976]. "A result on the equivalence problem for deterministic pushdown automata," *J. Computer and Systems Sciences* 13: 1, 38–50.
- Thompson, K. [1968]. "Regular expression search algorithm," *Comm. ACM* 11: 6, 419–422.
- Trakhtenbrot, B. A. [1964]. "Turing computations with logarithmic delay," *Algebra i Logika*, 3: 4, 33–48.
- Turing, A. M. [1936]. "On computable numbers with an application to the Entscheidungsproblem," *Proc. London Math. Soc.*, 2: 42, 230–265. A correction, *ibid.*, 43, pp. 544–546.
- Ullman, J. D. [1975]. "NP-complete scheduling problems," *J. Computer and Systems Sciences* 10: 3, 384–393.
- Valiant, L. G. [1973]. "Decision procedures for families of deterministic pushdown automata," *Theory of computation—Report No. 1*, Univ. of Warwick, Coventry, Great Britain.
- Valiant, L. G. [1974]. "The equivalence problem for deterministic finite-turn pushdown automata," *Information and Control* 25: 2, 123–133.
- Valiant, L. G. [1975a]. "General context-free recognition in less than cubic time," *J. Computer and Systems Sciences* 10: 2, 308–315.
- Valiant, L. G. [1975b]. "Regularity and related problems for deterministic pushdown automata," *J. ACM* 22: 1, 1–10.
- Valiant, L. G., and M. S. Paterson [1975]. "Deterministic one-counter automata," *J. Computer and Systems Sciences* 10: 3, 340–350.
- Wang, H. [1957]. "A variant to Turing's theory of computing machines," *J. ACM* 4: 1, 63–92.
- Wegbreit, B. [1969]. "A generator of context-sensitive languages," *J. Computer and Systems Sciences* 3: 3, 456–461.
- Wise, D. S. [1976]. "A strong pumping lemma for context-free languages," *Theoretical Computer Science* 3: 3, 359–370.
- Yamada, H. [1962]. "Real-time computation and recursive functions not real-time computable," *IEEE Trans. on Electronic Computers* 11: 6, 753–760.
- Yannakakis, M. [1978]. "Node and edge deletion NP-complete problems," *Proc. Tenth Annual ACM Symposium on the Theory of Computing*, pp. 253–264.
- Yasuhara, A. [1971]. *Recursive Function Theory and Logic*, Academic Press, New York.
- Younger, D. H. [1967]. "Recognition and parsing of context-free languages in time  $n^3$ ," *Information and Control* 10: 2, 189–208.

## INDICE

### A

- Aanderaa, S.O., 341  
Aceleración lineal, 310-312  
Aceleración, (*Véase también* Blum, teorema de la aceleración de.)  
Aceptación, estado de, (*Véase* Estado final)  
Ackermann, función de, 188-189  
Ackley, S. I., 188-189  
Adelman, L., 400-402  
AFL (*Véase Familia de lenguajes abstracta*)  
AFL completo, 299  
(*Véase también* Familia de lenguajes abstracta)  
AFL principal, 301  
Agente Viajero, problema del, 349-394  
Aregar, 251, 408  
Aho, A. V., 57-58, 79, 115, 134-143, 286, 302-303, 422-423  
Ajuste (o equilibrio) derecho, 42-43  
Ajuste (o equilibrio) izquierdo, 42-43  
alcanzabilidad, problema de la, 373-374  
Alfabeto, 2  
Alfabeto de cinta, 185  
Alfabeto de pila, 120  
ALGOL, 286  
Algoritmo, 147-149  
Algoritmo CYK, 164-165  
Almacenamiento en control finito, 164-165  
Ambigüedad inherente, 106-111  
Analizador léxico, 9, 49  
Ancestro, 4  
Arbib, M.A., 58  
Arbol, 3, 88  
(*Véase también* Derivación, árbol de.)  
Arbol de análisis gramatical (*Véase* Derivación, árbol de.)  
Arbol de extensión, problema de, 393  
Arco, 2  
Arden, D. N., 57  
Arista, 2  
Asimetría, 7  
Atomo, 380  
Autómata de pila, 408-409, 422-423  
Autómata de pila de una sola dirección, 408, 418, 422-423  
Autómata de pila no borrador, 408, 410, 422-423  
Autómata de presión, 9, 117, 134  
(*Véase también* Autómata de presión determinístico)  
Autómata de presión auxiliar, 400-402, 403, 422  
Autómata de presión de dos direcciones, 134  
Autómata de presión determinístico, 124-132, 131, 249  
Autómata finito de dos cintas, 78  
Autómata finito de dos direcciones, 39-41  
Autómata finito determinístico, 21

(Véase también Autómata finito)  
 Autómata finito no determinístico, 21-24  
 Autómata finito no determinístico de dos direcciones, 55  
 Autómata finito, minimización de, 15-58, 72-76  
 Autómata lineal acotado, 241  
 Axt, P., 341-342

**B**  
 Backus, J. W., 115  
 Backus-Naur, forma de, 84  
 Baker, B. S., 231-232  
 Baker, T., 400-402  
 Bar-Hillel, Y., 80-81, 155, 231-232  
 Beeri, C., 2285-286, 222-223  
 Berman, L. 400-402  
 Bird, M., 76, 80-81, 302-303  
 Blanco, espacio en, 159  
 Blattner, M., 422-423  
 Blum, axioma de, 335  
 Blum, M., 341-342  
 Blum, teorema de la aceleración de, 329-331  
 Boasson, L., 155  
 Book, R. V., 231-232, 341-342  
 Booleana, expresión, 324-235  
 Borodin, A., 319  
 Borodin, teorema de espacio de, 328-329  
 Borosh, I., 400-402  
 Borrado con límite  $k$  (Véase Borrado limitado)  
 Borrado limitado, 292-294  
 Brainerd, W. S., 188-189  
 Bruno, J. L., 400-402  
 Bruss, A. R., 400-402  
 Brzozowski, J. A., 57-58  
 Bullen, R. H., hijo., 557-58

**C**  
 Cabeza de cinta, 39  
 Cadena, 1  
 Cadena vacía, 1-2, 30  
 Cálculo de una máquina de Turing, 216, 226-227  
 Cálculo no válido (Véase Cálculo de una máquina de Turing)  
 Cálculo válido (Véase Cálculo de una máquina de Turing)  
 Cantor, D. C., 115, 231-232  
 Cardoza, E., 400-402  
 Cartesiano, producto, 5  
 Categoría sintáctica (Véase Variable)  
 Celoni, J. R., 341-342  
 Cerradura (Véase Kleene, cerradura de.)  
 Cerradura transitiva (Véase Cerradura reflexiva y transitiva)  
 Cerradura de una relación, 8

Cerradura efectiva, 63, 219  
 Cerradura-épsilon, 27  
 Cerradura positiva, 30, 247, 296  
 (Véase también Kleene, cerradura de.)  
 Cerradura reflexiva y transitiva, 8  
 Cerradura, propiedades de, 62, 141, 146, 251, 254, 259-262, 291, 294, 390, 420  
 CFG/CFL (Véase Gramática/lenguaje libre de contexto)  
 Chandler, W. J., 302-303  
 Chandra, A. K., 400-402  
 Chomsky, forma normal de, 100-101  
 Chomsky, jerarquía de, 233  
 Chomsky, N., 115, 134, 231-232, 248  
 Christofides, N., 400-402  
 Church, A., 188-189  
 Church, hipótesis de, 178-179  
 CICLO, 76, 152, 154, 298  
 Cinta, 19, 39, 159, 404  
 Cinta infinita de dos direcciones 171-173  
 Circuito de interrupción, 15  
 Circuito, problema del valor de, 396  
 Cobertura de vértice, problema de la, 355-356  
 Cobertura exacta, problema de, 365-392  
 Cobham, A., 400  
 Cociente, 66, 152, 260, 297, 420,  
 Cocke, J., 155  
 (Véase también Algoritmo CYK)  
 Codificación de una máquina de Turing, 195-197  
 Código, problema de generación de, 367  
 Coffman, R. g., 400-402  
 Cole, S. N., 286  
 Compilador, 132-133  
 Complejidad axiomática, 334  
 Complejidad, clase de, 308, 327-329  
 Complejidad computacional, 305-342  
 Complejidad espacial, 305-307, 316-317, 368-371, 410, 412, 414-415  
 Complejidad espacial no determinística, 308, 321-323  
 Complejidad temporal, 306-307, 320-321, 329, 335, 343-347, 404-405  
 Complejidad temporal no determinística, 308, 323, 378-379  
 Complementación, 63, 194, 251-256, 298, 366, 420  
 Compresión de cinta, 308-316  
 Concatenación, 1-2, 30, 63, 141, 246, 262  
 Conectividad fuerte, problema de, 396  
 Configuración, 405  
 (Véase también Descripción instantánea)  
 Congruencia, relación de, 78  
 Conjuntiva, forma normal, 348, 351  
 Conjunto, 5-6  
 Conjunto contable, 6

Conjunto de adelanto, 278  
 Conjunto infinito, 6  
 Conjunto no contable, 6  
 Conjunto potencia, 5  
 Conjunto regular, 20, 59-81, 203-204, 217, 219, 234, 244, 262, 287-288, 295, 296, 297  
 Conjunto regular libre de épsilon, 288  
 Conjunto semilineal, 76  
 Conjunto vacío, 31  
 Co-~~A~~, 365-366  
 Constable, R. L., 341-342  
 Constructibilidad completa tiempo/espacio, 317-320  
 Constructibilidad espacial, 317  
 Constructibilidad temporal, 320  
 Contención de conjuntos, 5  
 Contención, problema de, 218  
 Contención, propiedad de, 204  
 Contradominio, 6  
 Control finito, 19  
 (Véase también Estado)  
 Conway, J. H., 57-58  
 Cook, S. A., 134, 188-189, 341-342, 375, 400-402, 422  
 Corasick, M. J., 57-58  
 Corrimiento de símbolos, 168-169  
 Cremers, A., 302-303  
 CSG/CSL (Véase Gramáticas / lenguajes sensibles al contexto)  
 CSL determinístico, 246  
 Cuadrados perfectos, 61  
 Cudia, D. F., 232  
 Cúmulo, problema de, 390, 395  
 Cúmulo, problema de cobertura de, 390

**D**  
 Davis, M., 188-189  
 Decisión, propiedades de, 67-69, 147-149, 262-263, 298, 422-423  
 De Remer, F. L., 285-286  
 Derivación, 88, 91-94, 237, 416  
 Derivación, árbol de, 88-91  
 Derivación derecha, 94  
 Derivación izquierda, 94  
 Descendiente, 4  
 Descripción instantánea, 39, 121-122, 160-161  
 DFA (Véase Autómata finito determinístico)  
 Diagonalización, 197-198, 390  
 Diagrama de transiciones, 18  
 Digraf (Véase Gráfica dirigida)  
 Dominio, 6

**E**  
 Early, J., 155  
 Editor de textos, suprimir, 49, 251, 409  
 Entrada, alfabeto de, 19, 120, 289

Entrada, cinta de, 403, 416  
 Entrada, símbolo de, 159  
 Enumeración, 180-183, 204, 244  
 Equivalencia, clase de, 7, 11-12, 71  
 Equivalencia, estados de, 72  
 Equivalencia, problema de, 69-70, 298  
 Equivalencia, relación de, 7, 69-70  
 ESPACIOD (Véase Complejidad espacial)  
 ESPACION (Véase Complejidad espacial no determinística)  
 ESPACIOP, 344  
 ESPACIOP-completo, problema 347, 364-371  
 ESPACIOP-difícil, problema, 347  
 Estado, 17, 120, 159, 289, 404  
 Estado final, 19, 120, 159, 289  
 Estado inicial, 19, 120, 159, 289  
 Estado muerto, 252  
 Estados distinguibles, 72-73  
 Euclides, 8  
 Even, S., 400-402  
 Evey, J., 134  
 Expresión regular, 9, 30, 32, 375-378, 396

**F**  
 FA (Véase Autómata finito)  
 Familia de lenguajes, 287  
 Familia de lenguajes abstracta, 295-296  
 (Véase también Familia de lenguajes abstracta, Trío)  
 Fermat, conjetura de, 193  
 Ferrante, J., 400-402  
 FIN, 299  
 Finitud, problema de, 68, 147-148, 204, 395  
 Fischer, M. J., 248, 341-342, 400, 402  
 Fischer, P. C., 134, 188-189, 285-286, 341, 342  
 Floyd, R. W., 115, 155, 231-232  
 Forma oracional, 87, 416, 422-423  
 Forma oracional derecha, 265  
 Formador de conjuntos, 5  
 Fórmula booleana cuantificada, 341-342  
 Freedman, A. R., 341-342  
 Friedman, A. D., 57-58  
 Friedman, E. P., 285-286  
 Función calculable, 9-10  
 Función de movimiento siguiente, 159  
 Función de transición, 19, 52  
 Función parcialmente recursiva, 162  
 Función primitiva recursiva, 188  
 Función recursiva, 187, 221-222  
 (Véase también Función parcialmente recursiva)

**G**

Gabriellian, A., 302  
 Garey, M. R., 400-402  
 Gathen, J., 400-402  
 Generador de pares, 182  
 Gill, J., 400-402  
 Ginsburg, S., 80-81, 115, 134, 231-232, 248, 285-286, 302-303, 422-423  
 GMS (*Véase Máquina secuencial generalizada*)

Godel, K., 158, 379, 397

González, T., 400-402

Grafo, 2

Grafo dirigido, 2  
 (*Véase también Diagrama de transición*)

Graham, R. L., 400-402

Graham, S. L., 155

Gramática (*Véase Gramática libre de contexto*, Gramática sensible al contexto, Gramática regular, Gramática tipo 0)

Gramática ambigua, 94, 215, 272

(*Véase también Ambigüedad inherente*)

Gramática autofijable, 245

Gramática de estructura de frase (*Véase Gramática tipo 0*)

Gramática de operador, 114

Gramática lineal derecha (*Véase Gramática regular*)

Gramática lineal izquierda (*Véase Gramática regular*)

Gramática *LR* (0), 268-277, 285-286

Gramática *LR* (*k*), 277-281

Gramática no restringida (*Véase Gramática tipo 0*)

Gramática regular, 233-236

Gramática simple, 245

Gramática tipo 0, 237-240

Gramática lenguaje de contexto disperso, 300-301

Gramática lenguaje lineal, 114, 153, 229, 301-302

Gramáticas lenguajes libres de contexto, 9, 83-115, 124-130, 203-204, 218-219, 244-245, 287, 422-423

Gramáticas lenguajes sensibles al contexto, 240-243, 287, 371-373, 418

Gray, J. N., 134

Greibach, forma normal de, 101-106

Greibach, S. A., 115, 134, 202-203, 231-232, 248, 285-286, 341-342, 422-423

Greibach, teorema de, 219-222

Griffiths, T. V., 302-303

Gross, M., 231-232

Grzegorczyk, A., 341-342

GSM libre de épsilon, 272

**H**

Haines, L., 285-286

Hamilton, problema circuital de, 356-360

**I**

Hardy, G. H., 62  
 Harrison, M. A., 134, 155, 422-423  
 Hartmanis, J., 80-81, 134, 155, 188-189, 231-232, 341-342, 400-402  
 Hayashi, T., 422-423  
 Hell, P., 400-401  
 Hennie, F. C., 80-81, 188-189, 241-242,  
 Herman, G. T., 422-423  
 Hibbard, T., N., 428  
 Hijo, 4  
 Hilbert, D., 158  
 Hipótesis inductiva, 88  
 Hogben, L., 8  
 Homomorfismo, 64-66, 141, 246, 262, 283, 387, 295, 297, 301  
 Homomorfismo inverso, 64-66, 141-142, 246, 262, 283, 295, 297, 301  
 Homomorfismo libre de épsilon, 270, 278, 280  
 Hopcroft, J. E., 80-81, 134, 231-232, 243, 285-286, 302-303, 341-342, 400-403, 422-423  
 Huffman, D. A., 57-58, 80-81  
 Hunt, H. B., III, 231-232, 400-402

**I**

Ibarra, O. H., 134, 341-342  
 ID (*Véase Descripción instantánea*)  
 Independencia de operadores, 296-298  
 INIT, 76, 152, 297, 299  
 Instancia de un problema, 191  
 Intercalación, 300  
 Intersección, 5, 63, 144, 301  
 Intersección con un conjunto regular, 145-146, 262, 287, 296, 297, 298, 420  
 Inverso, 65, 152, 298  
 Irreflexividad, 7  
 Item, 264, 278  
 Item *LR* (*Véase Item*)  
 Item válido, 266

**J**

Jefferson, D., 400-402  
 Jerarquía espacial, 317-320  
 Jerarquía espacial no determinística, 323-327  
 Jerarquía temporal, 320-323  
 Jerarquía temporal no determinística, 327  
 Johnson, D. S., 400-402  
 Johnson, S. C., 285-286  
 Johnson, W. L., 49, 57-58  
 Jones, N. D., 188-189, 400-402

**K**

Kannan, R., 400-402  
 Karp, R. M., 400-402  
 Kasami, T., 155, 285-286  
 (*Véase también Algoritmo CYK*)  
 Kernel, T., 394

**L**

Kernel, T., 394  
 Kirkpatrick D. G., 400-402  
 Kleene, cerradura de, 30, 62, 141, 262, 282, 295  
 Kleene, S. C., 57-58, 188-189, 231-232  
 Knuth, D. E., 57-58, 285-286  
 Kohavi, Z., 57-58  
 Korenjak, A. J., 285-286  
 Kosaraju, S. R., 80-81  
 Kozen, D., 400-402  
 Kuroda, S. Y., 248

**L**

Laaser, W. T., 400-402  
 Ladner, R. E., 341-342, 400-402  
 Landweber, L. H., 188-189, 248  
 LBA (*Véase Autómata lineal limitado*)  
 Lema de sondeo, 59-62, 135-138, 153, 422  
 Lenguaje, 2, 18, 86-88, 106, 122, 162  
 Lenguaje determinístico (*Véase Autómata de presión determinístico*)  
 Lenguaje Dyck, 153

Lenguaje enumerable de manera recursiva, 162, 180-181, 193-195, 225, 244, 246, 287

Lenguaje formal (*Véase Lenguaje*)  
 Lenguaje indexado, 416-418, 422  
 Lenguaje metalineal, 153  
 Lenguaje recursivo, 162, 182, 193-195, 243-244, 287-288

Leong, B., 188-189  
 Lesk, M. E., 46, 49, 57-58  
 Levin, L. A., 400-402  
 Lewis, P. M., II, 115, 134, 188-189, 285-286, 341-342, 400-402

Lien, E., 400-402  
 Lindenmayer, A., 422-423  
 Lipton, R. J., 400-402  
 Literal, 348  
 Longitud de una cadena, 1  
 Lynch, N., 400-402

**M**

Machtey, M., 188-189  
 Mager, G., 421  
 Maibaum, T. S. E., 422-423  
 Manders, K., 400-402  
 Máquina contadora, 134, 183-185  
 Máquina de acceso aleatorio, 179-180  
 Máquina de dos pilas, 183  
 Máquina de predicción, 256-259  
 Máquina de Turing con límite espacial, 305  
 Máquina de Turing que se detiene, 160, 230  
 Máquina secuencial generalizada, 289-291  
 (*Véase también Transformación GSM, Transformación GSM inversa*)  
 MAX, 76, 152, 261, 262

**M**

McCarthy, J., 57-58  
 McGreight E. M., 341-342  
 McCulloch, W. S., 57, 58, 80, 81  
 McNaughton, R., 80-81  
 Mealy, G. H., 57-58  
 Mealy, máquina de, 46  
 Membresía, problema de, 149-152, 298  
 Meyer, A. R., 134, 188-189, 341-342, 400-402  
 Mezcla, 152  
 Millen, J. K., 57-58  
 Miller, G. L., 248, 400-402  
 MIN, 72, 152, 261-262, 299-300  
 Minsky, M. L., 57-58, 188-189, 231-232  
 Monna, C. L., 400-402  
 Moore, E. F., 57-58, 302-303  
 Moore, máquina de, 46  
 Morris, J. H., Hijo, 57-58  
 Movimiento, 120  
 Movimiento épsilon, 27, 255, 281  
 Myhill, J., 80-81, 248  
 Myhill-Nerode, teorema de, 69

**N**

Naur, P., 115  
 Nerode, A., 80-81  
 (*Véase también Myhill-Nerode, teorema de*)  
 NFA (*Véase Autómata finito no determinístico*)  
 No terminal (*Véase Variable*)  
 Nodo (*Véase Vértice*)  
 NP, 343-349, 387-390  
 NP-completo, problema, 348-354,  
 NP-difícil, problema, 360  
 Números, teoría de, 379, 398

**O**

Oettinger, A. G., 134  
 Ogden, lema de, 139-140  
 Ogden, W., 145, 394  
 Oppen, D. C., 400-402  
 Oráculo, 224-227, 387-390  
 Orden canónico, 181

**P**

*P*, 343-349, 387-390  
*P*-completo, problema, 372-375  
 Padre, 4  
 Palabra, 1  
 Palíndromo, 2, 11-12, 114  
 Papadimitriou, C. H., 400-402  
 Papert, S., 80-81  
 Parikh, R. J., 155  
 Parser (Analizador gramatical), 9, 285-286  
 Partición, problema de, 400  
 Paul, W. J., 341-342  
 Paull, M. C., 115  
 PCP (*Véase Post, problema de correspondencia de*)

PCP, solución parcial al, 212  
 PDA (*Véase* Autómata de presión)  
 PDA de vueltas finitas, 153  
 PDA en forma normal, 250  
 Perles, M., 80-81, 155, 231-232  
 Pilá vacía, aceptación, 122, 125, 271  
 Pilá, 117, 404  
 Pippenger, N., 341-342  
 Pitts, W., 57-58  
 Porter, J. H., 57-58  
 Post, problema de correspondencia de, 208-209  
 Post, E., 188-189, 231-232  
 Pratt, V. R., 57-58, 401-402  
 Predecesor, 2  
 Prefijo, propiedad del, 131, 264  
 Prefijo/sufijo propio, 1  
 Prenex, forma normal, 380  
 Presberger, aritmética de, 379, 397  
 Primos, números, 61-62, 367  
 Problema, 190  
 Problema completo, 347  
 Problema completo en espacio logarítmico, 364, 391  
 Problema del número cromático, 341, 369  
 Problema irresoluble, 193  
 Problema resoluble, 193  
 Problemas no tratables, 343, 402  
 Procedimiento efectivo (*Véase* Algoritmo)  
 Producción, 97-99  
 Producción épsilon, 97-99  
 Producción unitaria, 99  
 Programa, problema de, 392  
 Programación lineal (*Véase* Programación lineal entera)  
 Programación lineal entera, 360-364  
 Propiedad de lenguajes, 193-195

**R**

Rabín, M. O., 57-58, 341-342  
 Rackoff, C., 400-402  
 Raíz, 3  
 RAM (*Véase* Máquina de acceso aleatorio)  
 Reales con adición, 380-387  
 Reckhow, R. A., 188-189  
 Recursión, teorema de, 222-224  
 Red neuronal, 51  
 Reducción, 345-346  
   (*Véase también* 227-228)  
 Reducción de cinta, 309, 312-316  
 Reducción en espacio logarítmico, 345-346  
 Reducción de tabla verdadera, 229  
 Reducción en tiempo polinomial 345  
 Reducción muchos a uno, 227  
 Reedy, A., 231-232  
 Refinamiento de un relación de equivalencia, 70

Reflexividad, 7  
 Registro de una cinta, 179  
 Relación, 7-8  
 Rice, H. G., 115, 155, 231-232  
 Rice, teorema de, 199-207  
 Ritchie, R. W., 341-342  
 Rogers, H., Hijo, 188-189  
 Rose, G. F., 80-81, 134, 155, 231-232, 302-303  
 Rosenberg, A. L., 134, 188-189  
 Rosenkrantz, D. J., 115, 231-232, 400-402, 422-423  
 Ross, D. T., 57-58  
 Rozenberg, G., 422-423  
 Ruby, S., 341-342  
 Ruzzo, W. L., 155

**S**

SA (*Véase* Autómata de pila)  
 Sahni, S., 400-402  
 Salida, alfabeto de, 46-47, 398  
 Salida, cinta de, 180  
 Salomaa, A., 115, 422-423  
 Satisfactoriedad de árbol, 354-355  
 Satisfactoriedad, problema de, 348-354, 400  
 Savitch, teorema de, 344  
 Savitch, W. J., 231-232, 341-342, 400-402  
 Schaefer, T. J., 400-402  
 Scheinberg, S., 155  
 Schutzenberger, M. P., 115, 134, 231-232, 285-286  
 Scott, D., 57-58  
 Secuencia cruzada, 41, 336, 340  
 Seiferas, J. I., 80-81, 188-189, 341-342  
 Selman, A., 400-402  
 Señalador de extremo, 55, 178, 403, 408  
 Señalamiento de símbolo, 166-168  
 Sethi, R., 400-402  
 Shamir, E., 80-81, 155, 231-232  
 Shank, H., 155  
 Shannon, C. E., 57-58, 188-189  
 Shannon, juego de conmutación de, 396, 398-400  
 Shepherdson, J. C., 57-58  
 Sieveking, M., 400-402  
 Símbolo, 1  
 Símbolo de cinta, 159  
 Símbolo inicial, 120  
 Símbolo inútil, 95  
 Simétrica, 7  
 Simulación, 389  
 Singletary, W. E., 231-232  
 Singletón, 207  
 Sistema de estado finito, 15  
   (*Véase también* Autómata finito)  
 Sistema de estampilla, 228  
 Sistema L, 418-420  
 Sistema semi Thue (*Véase* Gramática tipo 0)

Sistema semi Thue (*Véase* Gramática tipo 0)  
 Solovay, R., 400-402  
 Spanier, E. H., 80-81, 155  
 Springsteel, F. N., 400-402  
 Stanley, R. J., 155  
 Sterns, R. E., 80-81, 115, 134, 188-189, 285-286,  
   341-342, 400-402  
 Steiglitz, K., 400-402  
 Stockmeyer, L. J., 400-402  
 Strassen, V., 400-402  
 SUB, 300  
 Subdorough, I. H., 400-402  
 Subrutina, 169-170  
 Sucesor, 2  
 Sufijo, 1  
 Sustitución, 65-67, 141-142, 246  
 Sustitución inversa, 152  
 Sustracción propia, 163  
 Suzuki, N., 400-402  
 Szymanski, T. G., 400-402

**T**

Tabla de transición, 409  
 Taniguchi, K., 285-286  
 Tarjan, R. E., 341-342, 400-402  
 Teorema de honestidad, 339  
 Teorema de la Aceleración, 329-331  
 Teorema del espacio (*Véase* Borodin, teorema del espacio de,)  
 Teorema Smn, 222  
 Terminal de una gramática, 83, 85, 416  
 Thompson, K., 50, 57-58  
 TIEMPOD (*Véase* Complejidad temporal)  
 TIEMPON (*Véase* Complejidad temporal no determinística)  
 Torii, K., 155  
 Trakhtenbrot, B. A., 341-342  
 Transductor A, 300  
 Transductor de presión, 134  
 Transductor en espacio logarítmico, 345  
 Transformación GSM, 290-291  
 Transformación GSM inversa, 290, 297, 420  
 Transformación GSM libre de épsilon, 290, 291, 297  
 Transitividad, 7

**U**

Ullian, J. S., 115, 231-132  
 Ullman, J. D., 57-58, 79, 115, 134, 231-232, 285-286, 302-303, 341-342, 400-402, 421  
 Unión, 5, 30, 62, 141, 194, 246, 262, 263  
 Unión, teorema de, 327-329

**V**

Vacio, problema del, 147, 298, 372-373  
 Valiant, L. G., 155, 285-286, 341-342  
 Variable de una gramática, 83, 86, 416  
 Vértice, 2  
 Veertice interior, 4

**W**

Wang, H., 188-189  
 Wegbreit, B., 248  
 Wise, D. S., 155  
 Wright, E. M., 61

**Y**

Yamada, H., 57-58, 341-342  
 Yannakakis, M., 400-402  
 Yasuhara, A., 188-189  
 Young, P. R., 188-189  
 Younger, D. H., 155  
   (*Véase también* Algoritmo CYK)