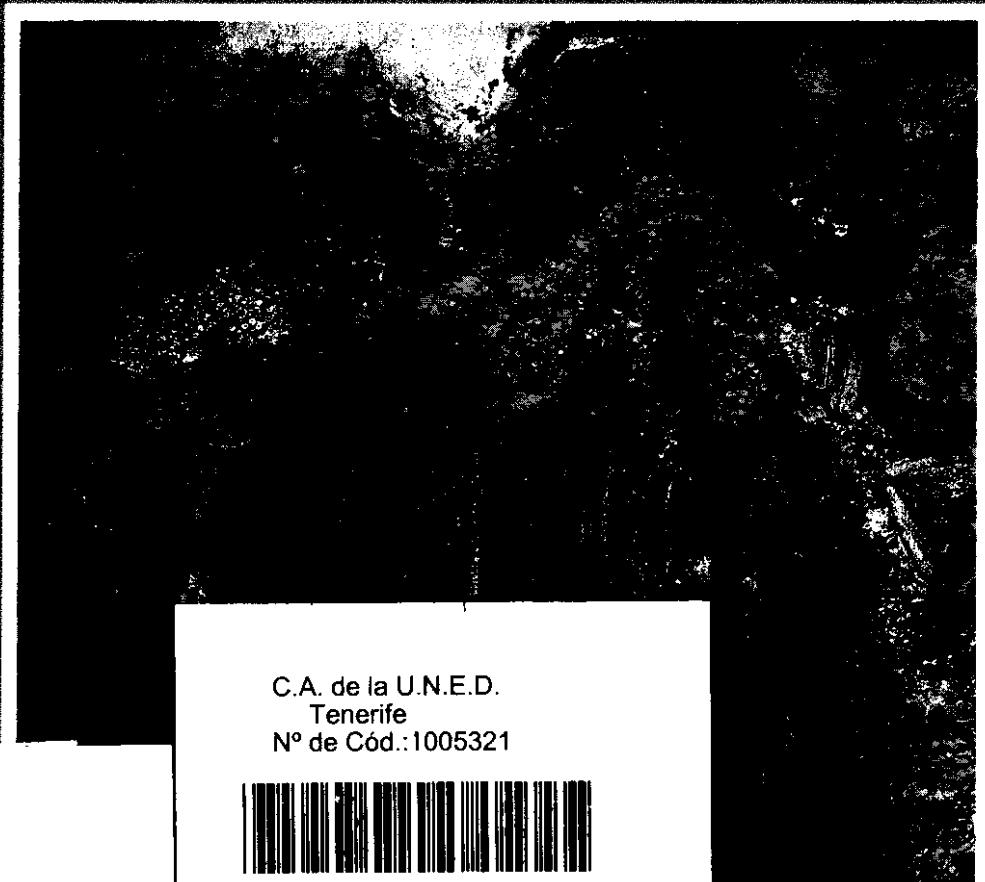


J. Glenn Brookshear

# TEORÍA DE LA COMPUTACIÓN

Lenguajes formales,  
autómatas y complejidad



C.A. de la U.N.E.D.  
Tenerife  
Nº de Cód.:1005321



ADDISON-WESLEY IBEROAMERICANA

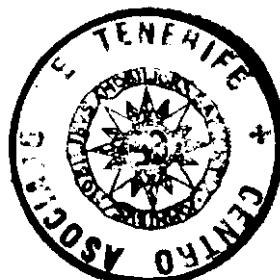
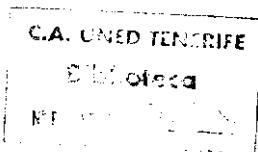
# TEORÍA DE LA COMPUTACIÓN

## Lenguajes formales, autómatas y complejidad

J. Glenn Brookshear  
*Marquette University*

Versión en español de  
**Ernesto Morales Peake**  
México, D.F.

Con la colaboración de  
**Jesús Luis Díaz Labrador**  
*Universidad de Deusto*  
Bilbao, España  
y  
**Guillermo Levine**  
*Universidad de Guadalajara*  
Jalisco, México



ADDISON-WESLEY IBEROAMERICANA  
Argentina • Brasil • Chile • Ecuador • España  
Estados Unidos • México • Perú • Puerto Rico • Venezuela

# ÍNDICE GENERAL

<b>Capítulo 0 Preliminares .....</b>	<b>1</b>
0.1 Repaso de la teoría de conjuntos .....	2
0.2 Base gramatical de la traducción de lenguajes .....	12
0.3 Antecedentes históricos .....	14
0.4 Esbozo del resto del texto .....	16
Problemas de repaso del capítulo .....	17
<b>Capítulo 1 Autómatas finitos y lenguajes regulares .....</b>	<b>21</b>
1.1 Análisis léxico .....	22
1.2 Autómatas finitos deterministas .....	28
1.3 Límites de los autómatas finitos deterministas .....	36
1.4 Autómatas finitos no deterministas .....	42
1.5 Gramáticas regulares .....	50
1.6 Expresiones regulares .....	57
1.7 Comentarios finales .....	68
Problemas de repaso del capítulo .....	69
<b>Capítulo 2 Autómatas de pila y lenguajes independientes del contexto .....</b>	<b>75</b>
2.1 Autómatas de pila .....	76
2.2 Gramáticas independientes del contexto .....	82
2.3 Límites de los autómatas de pila .....	102
2.4 Analizadores sintácticos $LL(k)$ .....	114
2.5 Analizadores sintácticos $LR(k)$ .....	121
2.6 Comentarios finales .....	134
Problemas de repaso del capítulo .....	135
<b>Capítulo 3 Máquinas de Turing y lenguajes estructurados por frases .....</b>	<b>141</b>
3.1 Máquinas de Turing .....	142

3.2	Construcción modular de máquinas de Turing .....	148
3.3	Máquinas de Turing como aceptadores de lenguajes .....	158
3.4	Lenguajes aceptados por máquinas de Turing .....	171
3.5	Más allá de los lenguajes estructurados por frases .....	181
3.6	Comentarios finales .....	195
	Problemas de repaso del capítulo .....	196
<b>Capítulo 4 Computabilidad .....</b>	<b>203</b>	
4.1	Fundamentos de la teoría de funciones recursivas .....	204
4.2	Alcance de las funciones recursivas primitivas .....	211
4.3	Funciones recursivas parciales .....	219
4.4	Poder de los lenguajes de programación .....	233
4.5	Comentarios finales .....	240
	Problemas de repaso del capítulo .....	241
<b>Capítulo 5 Complejidad .....</b>	<b>247</b>	
5.1	Complejidad de los cálculos .....	248
5.2	Complejidad de los algoritmos .....	251
5.3	Complejidad de los problemas .....	258
5.4	Complejidad temporal de los problemas de reconocimiento de lenguajes .....	267
5.5	Complejidad temporal de máquinas no deterministas .....	278
5.6	Comentarios finales .....	291
	Problemas de repaso del capítulo .....	292
<b>Apéndices .....</b>	<b>297</b>	
A.	Más acerca de la construcción de tablas de análisis sintáctico $LR(1)$ .....	297
B.	Más acerca de la función de Ackerman .....	305
C.	Algunos problemas importantes sin solución .....	313
D.	Acerca de la complejidad del problema de comparación de cadenas .....	319
E.	Muestras de problemas $NP$ .....	323
<b>Lecturas adicionales .....</b>	<b>327</b>	
<b>Índice de materias .....</b>	<b>329</b>	
<b>Vocabulario técnico bilingüe .....</b>	<b>333</b>	

## CAPÍTULO 0

# Preliminares

---

- 0.1 Repaso de la teoría de conjuntos
  - Definiciones básicas
  - Demostración por inducción
  - Conjuntos contables e incontables
  - ¿Y qué?
- 0.2 Base gramatical de la traducción de lenguajes
- 0.3 Antecedentes históricos
- 0.4 Esbozo del resto del texto

El concepto de proceso computacional, también llamado proceso algorítmico o algoritmo, es fundamental para la ciencia de la computación. Al ejecutar estos procesos, los computadores son capaces de resolver problemas; por el contrario, un computador no puede resolver un problema que no tenga una solución algorítmica. Así, cualquier limitación de las capacidades de los procesos computacionales es también una limitación de las capacidades de una máquina computadora.

Este texto es una introducción al estudio de los procesos computacionales. Nuestro objetivo es explorar el alcance de estos procesos utilizando métodos matemáticos rigurosos, y al mismo tiempo relacionar los resultados teóricos con aspectos de interés práctico. Comenzaremos con un estudio teórico de las técnicas de reconocimiento de patrones que asociamos en lo general con los aspectos prácticos del procesamiento de lenguajes y en lo particular con la construcción de compiladores. Esto nos llevará a ciertos límites aparentes con respecto al poder de los procesos computacionales y luego a un estudio teórico de la computabilidad. Una vez más, no sólo veremos los aspectos teóricos; encontraremos que existen muchos problemas que teóricamente tienen una solución algorítmica, aun cuando no cuenten con soluciones algorítmicas prácticas. Para ser más precisos, muchos de los problemas que en teoría pueden resolverse con algoritmos requieren tal cantidad de recursos (en lo que se refiere a tiempo o espacio de almacenamiento) que, desde el punto de vista práctico, se debe considerar que aún no tienen solución. Por lo tanto, nuestro

estudio concluirá con una exploración de los trabajos de investigación actuales encaminados a identificar aquellos problemas que tienen soluciones prácticas.

Para motivar aún más nuestro estudio de la computabilidad, comparemos a un físico que resuelve problemas utilizando las leyes físicas descubiertas por Isaac Newton con un programador que resuelve problemas expresando sus soluciones en un lenguaje de programación. Existen varios problemas que el físico puede resolver dentro del marco newtoniano; sin embargo, también existen problemas para los cuales las leyes físicas de Newton son demasiado restrictivas y, para darles solución, el físico debe emplear las leyes más generales de la teoría de la relatividad de Einstein. La ciencia de la física incorpora la búsqueda de leyes más poderosas con las cuales explicar fenómenos físicos cada vez más complejos. No obstante, sin importar los principios fundamentales sobre los cuales se apoya su estudio, existirá un límite para las preguntas que pueden responder los físicos. Por ejemplo, si suponemos que la teoría de la gran explosión es verdadera, es poco probable que los físicos puedan explorar alguna vez la porción del universo que yace más allá de la distancia que ha viajado la luz desde la explosión.

Como punto de comparación, existen varios problemas que el programador de computadores puede resolver utilizando un lenguaje de programación determinado, pero este lenguaje quizás imponga limitaciones a los problemas que el programador puede resolver. ¿Es posible superar estas limitaciones desarrollando lenguajes de programación más poderosos? Además, como sucede con la teoría de la gran explosión, ¿existe algún punto en el cual la adición de características a un lenguaje de programación, o el cambio a otro lenguaje, no incremente el poder de solución de problemas del programador? Estas preguntas son parte medular de nuestro estudio.

## 0.1 REPASO DE LA TEORÍA DE CONJUNTOS

Gran parte de nuestro estudio comprenderá los conceptos básicos de la teoría de conjuntos, por lo que repasaremos estas ideas antes de proseguir.

### Definiciones básicas

Intuitivamente, un **conjunto** es una colección de objetos que por lo general se denominan **elementos** (o **miembros**) del conjunto. Con frecuencia se emplea una notación con corchetes para describir un conjunto; por ejemplo, podemos escribir  $A = \{a, b, c\}$  para indicar que  $A$  es el conjunto cuyos elementos son  $a$ ,  $b$ , y  $c$ . Otro ejemplo es  $W = \{x: x \text{ es un entero positivo}\}$ , que se lee “ $W$  equivale al conjunto de todo  $x$  tal que  $x$  es un entero positivo”, lo cual indica que  $W$  es el conjunto de todos los enteros positivos. Un caso algo especial es el **conjunto vacío**, representado por  $\{\}$  y también por  $\emptyset$ , que no contiene elementos.

A un conjunto también se le puede llamar colección o familia, y podemos referirnos a un conjunto de números, a una colección de números o a una familia de números. Otro término que se emplea con frecuencia es clase, aunque técnicamente se trata de un término más general que un conjunto. De hecho, el concepto de clase se desarrolló para evitar paradojas como “el conjunto de todos los conjuntos que no pertenecen a sí mismos” (¿se contendría a sí mismo un conjunto de este tipo?). Sin embargo, no tenemos que preocuparnos por estos detalles; sencillamente, usaremos el término “clase” en aquellos casos donde se acostumbra hacerlo.

Un conjunto que aparece con frecuencia en los capítulos siguientes es el conjunto de los **números naturales**, también conocido como conjunto de los enteros no negativos  $\{0, 1, 2, 3, \dots\}$ , al cual representamos con el símbolo  $\mathbb{N}$ . Por desgracia, la definición de los números naturales varía un poco. Algunas personas no consideran que el cero se encuentre en el conjunto de números naturales, y definen a estos números como el conjunto de los enteros positivos  $\{1, 2, 3, \dots\}$ . Este conjunto se representa con  $\mathbb{N}^+$ . Tal distinción no constituye más que un pequeño inconveniente. La tendencia a incluir el cero como número natural es un reflejo de que resulta más conveniente para los científicos de la computación comenzar a contar en cero que en uno.

El símbolo  $\in$  se lee como “es un elemento de”. Así,  $x \in X$  significa que  $x$  es un elemento de  $X$ . Por otra parte,  $\notin$  se lee como “no es un elemento de”.

La **unión** de dos conjuntos  $A$  y  $B$ , representada por  $A \cup B$ , es la colección de todos los elementos que se encuentran en  $A$  o en  $B$ . Así,  $A \cup B = \{x: x \in A \text{ o } x \in B\}$ . La **intersección** de dos conjuntos  $A$  y  $B$ , representada por  $A \cap B$ , es la colección de objetos que son elementos tanto de  $A$  como de  $B$ ; por consiguiente,  $A \cap B = \{x: x \in A \text{ y } x \in B\}$ . Por ejemplo, si  $A = \{a, b, c\}$  y  $B = \{b, c, d\}$ , entonces  $A \cup B$  sería  $\{a, b, c, d\}$  y  $A \cap B$  sería  $\{b, c\}$ .

Utilizamos el signo menos para representar la resta de conjuntos. Es decir,  $\{a, b, c\} - \{a, c\} = \{b\}$ , y  $\{a, b\} - \{d, e\} = \{a, b\}$ . El conjunto  $A - B$  se llama **complemento** de  $B$  con respecto a  $A$ . Algunas veces se da por sentado que los elementos de todos los conjuntos considerados pertenecen a un conjunto universal de gran tamaño. En estos casos, nos referimos al complemento de un conjunto  $X$  con relación a este conjunto universal sencillamente como el complemento de  $X$ . De esta manera, al hablar de números naturales, el complemento del conjunto  $\{1, 5\}$  se entendería como el complemento de  $\{1, 5\}$  con relación a  $\mathbb{N}$ ; si habláramos de símbolos alfábéticos, el complemento de  $\{a, b, c\}$  sería el conjunto de símbolos distintos de  $a, b$  y  $c$ .

Decimos que  $A$  es un **subconjunto** de  $B$ , representado por  $A \subseteq B$ , si todos los elementos de  $A$  son también elementos de  $B$  (así, el conjunto vacío es un subconjunto de cualquier conjunto). El conjunto  $A$  es un **subconjunto propio** de  $B$  si  $A \subseteq B$  y  $B - A \neq \emptyset$ . Los conjuntos  $A$  y  $B$  son iguales si se cumple que  $A \subseteq B$  y  $B \subseteq A$ .

El **producto cartesiano** de dos conjuntos  $A$  y  $B$ , representado por  $A \times B$ , es el conjunto de todos los pares ordenados de la forma  $(a, b)$ , donde  $a \in A$  y  $b \in B$  (observe que, por lo general,  $A \times B \neq B \times A$  ya que la disposición de los

componentes en un par ordenado es significativa). Es posible generalizar el concepto del producto de conjuntos para obtener el producto de más de dos conjuntos. Por ejemplo,  $A \times B \times C = \{(a, b, c) : a \in A, b \in B \text{ y } c \in C\}$ . Es común utilizar la notación abreviada  $A^2$  para representar  $A \times A$ ,  $A^3$  para representar  $A \times A \times A$  y, de manera general,  $A^n$  para representar la colección de todas las  $n$ -tuplas de elementos de  $A$ .

Dados dos conjuntos  $A$  y  $B$ , una función de  $A$  a  $B$  es un subconjunto de  $A \times B$  tal que cada elemento de  $A$  aparece como el primer elemento de uno y sólo uno de los pares ordenados de la función. Decimos que una función establece una **correspondencia** entre  $A$  y  $B$ . Si  $f$  es una función de  $A$  a  $B$ , entonces  $A$  es el **dominio** de  $f$  y se dice que  $f$  es una función de  $A$ . El conjunto de elementos que aparece como el segundo componente de los pares ordenados de una función se denomina **contradominio** de la función. Si cada elemento de  $B$  se halla en el contradominio de  $f$ , decimos que  $f$  es una función de  $A$  sobre  $B$  (o que  $f$  establece una correspondencia de  $A$  sobre  $B$ ). Si cada miembro del contradominio de  $f$  está asociado a sólo un elemento del dominio, decimos que  $f$  es una función **uno a uno**, o sencillamente que  $f$  es uno a uno.

Por lo general, los elementos del dominio de una función se consideran como entradas para la función y los elementos del contradominio como los valores de salida. Este concepto de entrada y salida origina la notación  $f: A \rightarrow B$ , lo cual representa una función  $f$  de  $A$  a  $B$ . Dada una función  $f: A \rightarrow B$ , representamos como  $f(x)$  al valor de salida asociado a la entrada  $x$ .

Por ejemplo, suponga que  $f: \mathbb{N} \rightarrow \mathbb{N}$  es la función que consiste en los pares ordenados de la forma  $(n, n^2)$ , es decir,  $f(n) = n^2$  para cada  $n \in \mathbb{N}$ . Esta función es uno a uno, pero no se encuentra sobre  $\mathbb{N}$  ya que el contradominio de  $f$  sólo contiene cuadrados perfectos.

## Demostración por inducción

Algunas veces queremos demostrar que una proposición es verdadera para una serie de enteros. Por ejemplo, si  $\mathcal{P}(X)$  representa el **conjunto potencia** de  $X$  (es decir, la colección de todos los subconjuntos de  $X$ ) y  $|X|$  representa el número de elementos del conjunto  $X$ , quizás querriámos demostrar que

$$|\mathcal{P}(X)| = 2^{|X|}$$

para todos los conjuntos finitos  $X$  con tamaño mayor o igual que uno.

En la figura 0.1 se confirma esta fórmula para todos los conjuntos de tamaño uno, dos y tres, presentando exhaustivamente todos los subconjuntos de cada caso (observe que los nombres asignados a los elementos de los conjuntos no tienen importancia; aquí empleamos la notación  $\{1, 2, \dots, n\}$  para representar un conjunto genérico de tamaño  $n$ ). No obstante, esta estrategia explícita de caso por caso nunca podrá demostrar que la fórmula es verdadera para todos los casos posibles, ya que existe un número

El conjunto $X$	Todos los subconjuntos de $X$	
{1}	{ }	{1}
{1, 2}	{ } {1}	{2} {1, 2}
{1, 2, 3}	{ } {1} {2} {1, 2} {1} {2, 3} {1, 2, 3}	{3} {1, 3} {2, 3} {1, 2, 3}

Figura 0.1 Conjuntos de cardinalidad 1, 2 y 3 con todos sus subconjuntos

infinito de casos que habría que evaluar. Lo que necesitamos es un argumento general que cubra al mismo tiempo todos los casos.

En nuestro ejemplo, podemos obtener este argumento identificando un método genérico para construir la colección de subconjuntos del conjunto  $\{1, 2, \dots, n + 1\}$  a partir de la colección de subconjuntos del conjunto  $\{1, 2, \dots, n\}$ . De manera específica, una vez generados todos los subconjuntos del conjunto  $\{1, 2, \dots, n\}$ , podemos generar todos los subconjuntos de  $\{1, 2, \dots, n, n + 1\}$  presentando una lista de todos los subconjuntos que se encuentran en el caso anterior (en una columna) y luego repitiendo esta lista (en una segunda columna) e insertando el elemento adicional,  $n + 1$ , en cada subconjunto (ésta fue la técnica empleada para construir la figura 0.1). A partir de esta observación general, llegamos a la conclusión de que existen dos veces más subconjuntos para un conjunto de tamaño  $n + 1$  que para un conjunto de tamaño  $n$ . Es decir, para cualquier  $n \in \mathbb{N}^*$ ,

$$|\mathcal{P}(\{1, 2, \dots, n + 1\})| = 2 |\mathcal{P}(\{1, 2, \dots, n\})|$$

Aquí lo importante es que se trata de una conclusión genérica, ya que no se basa en un valor específico de  $n$ : tiene la misma validez para el valor  $n = 3$  que para  $n = 153$  o para cualquier otro  $n \in \mathbb{N}^*$ . Por consiguiente, si podemos demostrar con un argumento específico que  $|\mathcal{P}(X)| = 2^{|X|}$  para un conjunto  $X$ , esta proposición genérica indicaría que la ecuación puede ampliarse para cada conjunto finito mayor sin tener que validar cada caso con un argumento aparte. De hecho, si siempre existen dos veces más subconjuntos para un conjunto de tamaño  $n + 1$  que para un conjunto de tamaño  $n$  y si  $|\mathcal{P}(X)| = 2^{|X|}$  para un conjunto finito  $X$ , entonces, para cualquier conjunto  $Y$  donde  $|Y| = |X| + 1$ , tenemos

$$|\mathcal{P}(Y)| = 2 |\mathcal{P}(X)| = 2 (2^{|X|}) = 2^{|Y|}$$

Entonces, si confirmamos la ecuación  $|\mathcal{P}(X)| = 2^{|X|}$  para el caso de un conjunto de tamaño uno (como se hizo en el primer renglón de la figura 0.1), la ecuación debe ser válida para cualquier conjunto de tamaño  $n \in \mathbb{N}^+$ .

Esta técnica se conoce como demostración por **inducción**; resulta útil cuando se desea demostrar un enunciado que tiene que ver con una serie de enteros. En vez de tratar de usar un argumento caso por caso, lo que nunca sería suficiente si existiera un número infinito de casos, se proporciona un argumento específico que contempla el menor valor considerado (**caso básico**) y luego se proporciona un argumento genérico para demostrar que, si el enunciado es verdadero en un caso, también deberá ser verdadero para el siguiente caso mayor.

Hemos empleado el ejemplo del conjunto potencia ya que la fórmula obtenida será de utilidad en análisis posteriores. Como un ejemplo más vinculado al ambiente de programación, se aplicará la inducción para demostrar que el procedimiento recursivo **clasificación**, que se esboza en la figura 0.2, obtiene la versión clasificada de su lista de entrada. Utilizaremos la inducción en el número de entradas de la lista.

Primero se confirma que **clasificación** opera correctamente para el caso básico, donde la lista de entrada no contiene dato alguno (lista vacía), rastreando explícitamente la ejecución del procedimiento para ese caso. Luego se supone que el procedimiento da como resultado la versión clasificada de listas con  $n$  elementos, donde  $n$  es un entero positivo genérico, y se demuestra que, entonces, debe producir la versión clasificada de cualquier lista que contenga  $n + 1$  elementos.

Para este fin observamos que una llamada de la forma **clasificación** (*ListaEntrada*), donde *ListaEntrada* es una lista que contiene  $n + 1$  elementos, ocasionaría la ejecución de la porción *then* del procedimiento. Allí, otra activación de **clasificación** produce la versión clasificada de la sublista que consiste en los elementos posteriores al primero de la lista original *ListaEntrada*. Puesto que esta sublista sólo contiene  $n$  elementos, la activación de **clasificación** se ejecutará correctamente (por nuestra suposición respecto a la lista con sólo  $n$  elementos) y, por lo tanto, la lista de entrada original aparecerá como el primer elemento original seguido por la versión clasificada del resto de la lista. Por ende, la siguiente instrucción del procedimiento, que mueve el primer elemento de la lista a su lugar correcto entre los demás, dará fin a la tarea de clasificar toda la lista. Concluimos que si el procedimiento **clasificación** clasifica correctamente listas de  $n$  elementos, entonces también debe clasificar de manera correcta listas con  $n + 1$  elementos. Al combinar esto con la primera conclusión anterior (que **clasificación** opera en forma correcta si se le proporciona la lista vacía), demostramos que el procedimiento clasificará listas finitas de cualquier tamaño.

## Conjuntos contables e incontables

Consideremos ahora el concepto intuitivo del número de elementos en un conjunto. A primera vista, no parece haber nada complicado con respecto a

```

procedure clasificación (lista);
begin
    if (lista no está vacía) then
        begin
            clasificación (porción de la lista que sigue al primer elemento);
            mover el primer elemento a su lugar correcto
            dentro de la parte restante de la lista
        end
    end;

```

**Figura 0.2** Esbozo del procedimiento *clasificación*

este concepto. Es obvio que en el conjunto  $\{a, b, c\}$  el número de elementos es tres, y que en  $\{x, y\}$  es dos. Sin embargo, ¿cuántos elementos hay en el conjunto  $\mathbb{N}^+$ ? Una persona común diría que una “infinidad”, pero esto, para nuestros propósitos, es demasiado simplista. El infinito no es un número de nuestro sistema contable normal, sino que el término refleja el concepto de un conjunto con más elementos que los que pueden representarse con un número tradicional. Además, en seguida veremos que los conjuntos con un número infinito de elementos, llamados **conjuntos infinitos**, no contienen la misma cantidad de elementos. El tamaño de dos conjuntos infinitos puede variar, al igual que el tamaño de dos conjuntos finitos. Por esto, existen varios niveles de infinitud, donde algunos conjuntos infinitos son mayores que otros.

Ya que en nuestro sistema contable tradicional no existen números que representen el tamaño de un conjunto infinito, es necesario ampliar nuestro sistema para tener en cuenta estos conjuntos. En este sistema ampliado nos referimos al tamaño de un conjunto como su **cardinalidad**. La cardinalidad de un conjunto finito corresponde al concepto tradicional del número de elementos en un conjunto, pero la cardinalidad de un conjunto infinito no es un número en el sentido habitual. Más bien, la cardinalidad de un conjunto infinito es un número, llamado **número cardinal**, que existe únicamente en nuestro sistema de numeración ampliado. La cardinalidad de un conjunto  $A$  se representa con  $|A|$ .

Para completar nuestro sistema de cardinalidad, es necesario establecer una forma de comparar las cardinalidades de dos conjuntos diferentes, es decir, se requiere definir qué significa el hecho de que una cardinalidad sea menor que otra. Para esto, se establece que la cardinalidad de un conjunto  $X$  es menor o igual que la cardinalidad del conjunto  $Y$ , lo cual se representa con  $|X| \leq |Y|$ , si y sólo si puede establecerse una correspondencia de cada elemento de  $X$  con un elemento único de  $Y$ . En otras palabras,  $|X| \leq |Y|$  si y sólo si hay una función uno a uno de  $X$  a  $Y$ . Observe que si  $X$  y  $Y$  son conjuntos finitos, esta definición de menor o igual equivale a la definición tradicional:  $|\{x, y\}| \leq |\{r, s, t\}|$  ya que puede establecerse una correspondencia entre  $x$  y  $r$  y entre  $y$  y  $s$ .

A continuación, se dice que la cardinalidad del conjunto  $X$  es igual a la cardinalidad del conjunto  $Y$ , representado por  $|X| = |Y|$ , si y sólo si existe una función uno a uno que establezca la correspondencia de  $X$  sobre  $Y$  (el teorema de Schröder-Bernstein, cuyos detalles no son importantes ahora, confirma que estas definiciones están de acuerdo con nuestra intuición; establece que si  $|X| \leq |Y|$  y  $|Y| \leq |X|$ , entonces  $|X| = |Y|$ ). Observe que  $|\mathbb{N}| = |\mathbb{N}^+|$ , donde una función uno a uno adecuada es aquella que asocia cada  $x$  de  $\mathbb{N}$  con el número  $x + 1$  de  $\mathbb{N}^+$ . Por último, definimos  $|X| < |Y|$  para representar  $|X| \leq |Y|$  y  $|X| \neq |Y|$ .

En nuestro análisis, es muy importante la relación entre un conjunto y su conjunto potencia. En el caso de un conjunto finito  $X$ , ya se ha mostrado que hay más elementos en  $\mathcal{P}(X)$  que en  $X$  (de hecho, si  $X$  contiene  $n$  elementos, entonces  $\mathcal{P}(X)$  contendrá  $2^n$  elementos). El teorema siguiente extiende a los conjuntos infinitos esta relación entre un conjunto y su conjunto potencia.

### TEOREMA 0.1

Si  $X$  es un conjunto cualquiera, entonces  $|X| < |\mathcal{P}(X)|$ .

#### DEMOSTRACIÓN

Obviamente,  $|X| \leq |\mathcal{P}(X)|$  pues la función que establece la correspondencia entre cada elemento  $x$  de  $X$  y el conjunto  $\{x\}$  proporciona la función uno a uno que se requiere (si  $X$  está vacío, entonces cada elemento de  $X$  puede corresponderse con un elemento de  $\mathcal{P}(X)$  ya que no hay elementos en  $X$ ).

Nuestra tarea, entonces, es mostrar que  $|X| \neq |\mathcal{P}(X)|$ . Esto se puede lograr demostrando que no existe función alguna de  $X$  sobre  $\mathcal{P}(X)$ . Suponga que  $f$  es cualquier función de  $X$  a  $\mathcal{P}(X)$  y considere el conjunto  $Y = \{x: x \in X \text{ y } x \notin f(x)\}$ . Observe que  $Y$  es un subconjunto de  $X$  y, por lo tanto, un elemento de  $\mathcal{P}(X)$ . Luego, si  $f$  es sobre  $\mathcal{P}(X)$ , entonces en  $X$  debe existir una  $y$  tal que  $f(y) = Y$ .

Sin embargo, si existe tal  $y$ , entonces debe cumplirse  $y \in f(y)$  o  $y \notin f(y)$ , las cuales conducen a contradicciones. Si  $y \in f(y)$ , entonces la definición de  $Y$  significaría que  $y \notin Y$ , lo que contradice la afirmación de que  $f(y) = Y$ . De manera parecida, si  $y \notin f(y)$ , entonces la definición de  $Y$  implicaría que  $y \in Y$ , lo cual también contradice la suposición de que  $f(y) = Y$ .

Concluimos que no existe un  $y \in X$  tal que  $f(y) = Y$ . Por consiguiente  $f$  no puede ser sobre  $\mathcal{P}(X)$  y, por lo tanto,  $|X| < |\mathcal{P}(X)|$ .



Ahora nos encontramos en un punto donde podemos apoyar nuestra afirmación de que los conjuntos infinitos pueden tener cardinalidades distin-

tas. Basta con seleccionar un conjunto infinito  $X$  y comparar su cardinalidad con la de su conjunto potencia  $\mathcal{P}(X)$ . Por el teorema 0.1, la cardinalidad del segundo conjunto debe ser mayor que la del primero. De hecho, el proceso de formación de conjuntos potencia nos lleva a una jerarquía de conjuntos infinitos ya que la cardinalidad de  $\mathcal{P}(\mathcal{P}(X))$  debe ser mayor que la cardinalidad de  $\mathcal{P}(X)$ , etcétera. Es decir,

$$|X| < |\mathcal{P}(X)| < |\mathcal{P}(\mathcal{P}(X))| < \dots$$

Una consecuencia de esta jerarquía es que no existe ningún cardinal infinito mayor. Dado un conjunto infinito, su potencia será siempre mayor.

Existe, no obstante, un cardinal infinito menor: la cardinalidad de  $\mathbb{N}$ . Esta afirmación va de acuerdo con nuestra percepción intuitiva de que si  $X$  es un conjunto infinito, entonces es posible extraer elemento por elemento sin agotar el conjunto; es decir, se puede extraer un elemento “cero”, luego un “primer” elemento seguido por un “segundo”, etc.; por lo tanto,  $X$  debe contener por lo menos  $|\mathbb{N}|$  elementos. Sin embargo, este argumento intuitivo se basa en un enfoque de caso por caso que, como ya establecimos en nuestro análisis de la inducción, no proporciona una demostración válida. Por lo tanto, presentaremos el siguiente teorema formal y su demostración; se basa en el axioma de elección, el cual establece que, dada una colección cualquiera de conjuntos no vacíos, existe una función  $g$  cuyo dominio es la colección de conjuntos, y que para cada conjunto  $X$  de la colección,  $g(X) \in X$ . Es decir, si se toma como entrada un conjunto  $X$  de la colección, la función dará como resultado un elemento de  $X$ .

### TEOREMA 0.2

La cardinalidad del conjunto de los números naturales es menor o igual que la cardinalidad de cualquier conjunto infinito.

### DEMOSTRACIÓN

Debemos mostrar que, dado un conjunto infinito  $X$ , existe una función  $f$  uno a uno de  $\mathbb{N}$  a  $X$ . Para esto, hagamos que  $g$  represente una función cuyo dominio es la colección de los subconjuntos no vacíos de  $X$ , y que para cada subconjunto no vacío  $Y$  de  $X$ ,  $g(Y)$  es un elemento de  $Y$  (la existencia de esta función se basa en el axioma de elección). Mediante esta asociación entre subconjuntos no vacíos de  $X$  y elementos, definimos ahora la función  $f$  por inducción. Definimos que  $f(0)$  es  $g(X)$ . Entonces, si  $n$  es un número natural para el cual se han definido  $f(n), f(n-1), \dots, f(0)$ , definimos  $f(n+1)$  como el elemento  $g(X - \{f(0), f(1), \dots, f(n)\})$ . Observe que, de acuerdo con nuestra definición de  $g$ ,  $f(n+1)$  se toma del conjunto  $X - \{f(0), f(1), \dots, f(n)\}$ , el cual es no vacío ya que  $X$  es infinito y  $\{f(0), f(1), \dots, f(n)\}$  es finito.

Por lo tanto,  $f(n + 1)$  debe ser distinto de  $f(0), f(1), \dots, f(n)$ . Entonces,  $f$  es una función uno a uno de  $\mathbb{N}$  en  $X$ . Concluimos que la cardinalidad de los números naturales no es mayor que la de  $X$ .



El teorema 0.2 muestra que el conjunto  $\mathbb{N}$  es uno de los conjuntos infinitos más pequeños. Decimos que es *uno* de los más pequeños ya que existen varios conjuntos del mismo tamaño (de la misma manera que existen varios conjuntos de tamaño tres). Por ejemplo, ya se vio que  $\mathbb{N}^+$  es del mismo tamaño que  $\mathbb{N}$ .

Finalmente, presentaremos los conceptos de conjuntos contables e incontables, como antes prometimos. Aquí resulta útil recordar el proceso al que normalmente nos referimos como contar. Contar los elementos de un conjunto significa asignar los valores  $1, 2, 3, \dots$ , a los elementos del conjunto. Incluso podemos pronunciar las palabras “uno, dos, tres, …” conforme señalamos los objetos durante el recuento. En resumen, el proceso de recuento asigna enteros positivos a los elementos de un conjunto. Sin embargo, hemos visto que existen conjuntos infinitos con más elementos que la cantidad de enteros positivos. Por consiguiente, no es posible contar estos conjuntos ya que no hay enteros positivos suficientes. Se dice que estos conjuntos son *incontables*. Por otra parte, se dice que un conjunto es **contable** (o *enumerable*) si su cardinalidad es menor o igual que la de los enteros positivos. El conjunto de los números naturales es contable; no así su conjunto potencia. De hecho, la combinación de los teoremas 0.1 y 0.2 indica que *el conjunto potencia de cualquier conjunto infinito es incontable*.

Una forma de mostrar que un conjunto infinito es contable consiste en demostrar que existe un proceso para presentar una lista en secuencia de sus elementos de manera que cada elemento del conjunto aparezca en la lista. La elaboración de esta lista es una forma de contar los elementos: a la primera entrada de la lista se le asigna uno; a la segunda se le asigna dos; a la tercera, tres, etcétera.

Como ejemplo, se mostrará que el conjunto  $\mathbb{N}^+ \times \mathbb{N}^+$  es contable. Esto se logra describiendo la elaboración de una lista de todos los elementos de  $\mathbb{N}^+ \times \mathbb{N}^+$ . Esta lista consiste en todos los pares cuyos componentes suman dos, a continuación todos los pares cuyos componentes suman tres, en seguida todos los pares cuyos componentes suman cuatro, etcétera. Los pares de cada una de estas agrupaciones se colocan en orden, de acuerdo con sus primeros componentes. Así, la lista tiene la forma  $(1, 1), (1, 2), (2, 1), (1, 3), (2, 2), (3, 1), \dots$  (véase Fig. 0.3). Observe que, a final de cuentas, esta lista contendrá cada uno de los elementos de  $\mathbb{N}^+ \times \mathbb{N}^+$  (el par  $(m, n)$  aparecerá como el elemento  $n$  la porción de la lista que contiene los pares cuyos componentes suman  $m + n$ ). Por último, observamos que la elaboración de esta lista es, en realidad, un recuento de los elementos de  $\mathbb{N}^+ \times \mathbb{N}^+$ : hemos asignado uno a  $(1, 1)$ , dos a  $(1, 2)$ , tres a  $(2, 1)$ , etcétera. Por consiguiente, podemos concluir que  $\mathbb{N}^+ \times \mathbb{N}^+$  es contable.

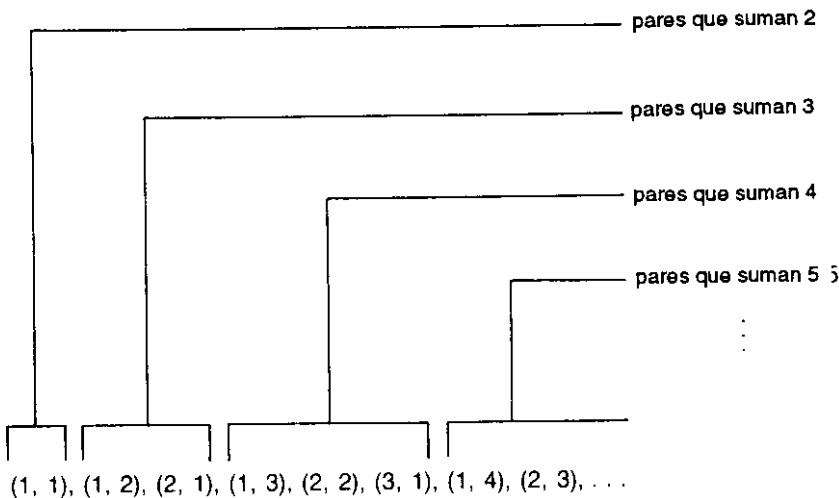


Figura 0.3 Elaboración de la lista de elementos de  $\mathbb{N}^+ \times \mathbb{N}^+$  en secuencia

### ¿Y qué?

Concluyamos esta sección relacionando nuestro análisis con un ambiente de computación típico. Suponga que tenemos un programa listo para ejecutarse en un computador. El programa está diseñado para aceptar cierta entrada (quizás una lista de nombres que debe clasificarse) y producir determinada salida (como sería la lista ordenada). Sin importar cuáles sean la entrada y la salida, desde el punto de vista de la máquina no son más que una cadena de bits. Por lo tanto, podemos considerar a la entrada y a la salida como números naturales representados en notación binaria (podemos imaginar un 1 adicional colocado en el extremo izquierdo de estas cadenas de bits, de modo que los 0 iniciales de las cadenas originales sean significativos). Desde esta perspectiva, el programa acepta un número de entrada y, con base en el valor de ese número, produce un número de salida. Es decir, la acción del programa es calcular una función de  $\mathbb{N}$  a  $\mathbb{N}$ . A la vez, cada función de  $\mathbb{N}$  a  $\mathbb{N}$  sugiere un programa que en alguna ocasión podríamos escribir.

¿Cuántas funciones de éstas existen? Obviamente, no hay más funciones de  $\mathbb{N}$  a  $\{0, 1\}$  que de  $\mathbb{N}$  a  $\mathbb{N}$ , ya que éstas proporcionan más valores de salida potenciales. Sin embargo, hay tantas funciones de  $\mathbb{N}$  a  $\{0, 1\}$  como elementos en el conjunto potencia de  $\mathbb{N}$ . Incluso, para cada subconjunto  $X$  de  $\mathbb{N}$  hay una función  $f$ , llamada **función característica** de  $X$ , de  $\mathbb{N}$  a  $\{0, 1\}$ , definida por

$$f(n) = \begin{cases} 1 & \text{si } n \in X \\ 0 & \text{si } n \notin X \end{cases}$$

Concluimos que existen por lo menos tantas funciones de  $\mathbb{N}$  a  $\mathbb{N}$  como elementos de  $\mathcal{P}(\mathbb{N})$ . Por lo tanto, existe una cantidad incontable de funciones para las cuales podríamos escribir programas.

Ahora, considere su lenguaje de programación preferido. Existe un número finito de símbolos a partir de los cuales se construye cualquier programa en dicho lenguaje. Establezcamos un orden “alfabético” para estos símbolos. Luego podríamos generar una lista de todos los programas en el lenguaje que tienen longitud de un símbolo (si acaso existe alguno) en orden alfabetico; después podríamos presentar en orden alfabetico los programas con longitudes; a continuación, los programas de longitud tres en orden alfabetico, etcétera. Esta lista contendría cualquier programa que pudiera escribirse en este lenguaje. Por consiguiente, este procedimiento de generación de la lista sugiere que el conjunto de los programas que pueden escribirse en el lenguaje es contable.

¿Qué hemos mostrado con esto? Sólo existe una cantidad contable de programas que pueden escribirse en su lenguaje de programación favorito, pero hay una cantidad incontable de funciones que podrían calcularse con un programa. Puesto que cada programa calcula sólo una función, se quedará corto. Por ende, existen problemas que se quisieran resolver utilizando un computador, ¡pero para los cuales no puede escribirse un programa en su lenguaje de programación preferido!

¿Cuáles son algunos de estos problemas? ¿Es posible resolver alguno de ellos cambiando a otro lenguaje de programación? ¿Podrían resolverse estos problemas si construimos un computador más grande? Incluso si usted puede producir un programa para resolver un problema, ¿sería capaz una máquina actual de ejecutarlo con la rapidez suficiente para producir una respuesta mientras usted esté vivo? ¿Será capaz el computador del mañana de producir una respuesta durante la vida de sus hijos? Estas son algunas de las preguntas que trataremos directa o indirectamente en los capítulos siguientes.

## 0.2 BASE GRAMATICAL DE LA TRADUCCIÓN DE LENGUAJES

Gran parte de las primeras etapas de nuestro estudio se basará en problemas que tienen que ver con el procesamiento del lenguaje. Por lo anterior, tomemos un momento para repasar la terminología y algunos de los conceptos asociados al proceso de traducción de lenguajes.

En primer lugar, debemos esclarecer los términos **lenguaje formal** y **lenguaje natural**. Es posible distinguir a ambos por medio de la pregunta “¿qué surgió primero, el lenguaje o sus reglas gramaticales?” En general, un lenguaje natural es aquel que ha evolucionado con el paso del tiempo para fines de la comunicación humana, por ejemplo el español, el inglés o el alemán. Estos lenguajes continúan su evolución sin tomar en cuenta reglas gramaticales formales; cualquier regla se desarrolla después de que sucede el hecho, en

un intento por explicar, y no determinar, la estructura del lenguaje. Como resultado de esto, pocas veces los lenguajes naturales se ajustan a reglas gramaticales sencillas u obvias. Esta situación es el pelo en la sopa de los académicos de la lengua, la mayoría de los cuales trata de convencernos de que nuestra forma de hablar es incorrecta en vez de aceptar la posibilidad de que no han encontrado el conjunto de reglas adecuado, que sus reglas ya no son válidas o que no existen dichas reglas.

En contraste con los lenguajes naturales, los formales están definidos por reglas preestablecidas y, por lo tanto, se ajustan con todo rigor a ellas. Como ejemplo tenemos los lenguajes de programación de computadores y los lenguajes matemáticos como el álgebra y la lógica de proposiciones. Gracias a esta adhesión a las reglas, es posible construir traductores computadorizados eficientes para los lenguajes de programación, a la vez que la falta de observancia de reglas establecidas dificulta la construcción de un traductor para un lenguaje natural.

Para realzar el papel de la gramática en el proceso de traducción, revisemos los pasos que sigue un típico traductor de lenguajes, como un compilador para determinado lenguaje de programación. Una de las tareas fundamentales de este traductor es reconocer los bloques de construcción individuales del lenguaje que se traduce. En su nivel más bajo, esto implica reconocer que ciertas cadenas de símbolos del programa que se traduce (llamado **programa fuente**) deben considerarse como la representación de un solo objeto. Por ejemplo, hay que reconocer la variable LISTA como un identificador, en vez de cinco letras por separado; la cadena de dígitos 524 debe reconocerse como un solo número; la cadena: = (que se utiliza en varios lenguajes de programación) debe reconocerse como la operación de asignación.

El proceso de identificación de estas secuencias multisímbolos dentro del programa fuente es manejada por una porción del compilador llamada **analizador léxico**. En esencia, se trata de un submódulo que recibe el programa fuente como una cadena de símbolos y produce una cadena de **componentes léxicos (tokens)**, cada uno de los cuales representa un solo objeto que pudo haberse representado con más de un símbolo en la cadena de entrada (un componente léxico puede representarse como un valor entero que es posible almacenar en una sola celda de la memoria de la máquina).

Después de reducir a un componente léxico cada secuencia multisímbolos que representa a un objeto, la siguiente tarea del traductor es analizar el patrón de componentes léxicos. Al llegar a este punto, un compilador típico tendría que reconocer que los componentes léxicos de if, then y else representan conjuntamente una estructura if-then-else, o que la cadena que se encuentra entre el componente léxico repeat y el componente léxico until correspondiente representa el cuerpo de una estructura de ciclo. Este análisis se conoce como **análisis sintáctico** y es realizado por un módulo del compilador que se llama **analizador sintáctico**.

Conforme el analizador sintáctico reconoce las distintas estructuras del programa, solicita los servicios de otro módulo, llamado **generador de código**,

para producir la versión traducida de esa parte específica del programa. Al formar estas porciones traducidas del programa, empieza a constituirse la versión traducida de éste (llamada **programa objeto**).

En resumen, el proceso de traducción consta de tres componentes básicos: análisis léxico, análisis sintáctico y generación de código, como se muestra en la figura 0.4. Los primeros dos componentes dependen en gran medida de la capacidad para reconocer patrones. Si éstos se basan en reglas gramaticales, entonces el proceso de reconocimiento también puede apoyarse en estas reglas. Lo anterior sucede al procesar un lenguaje formal. Sin embargo, si el lenguaje que se traduce no se adhiere estrictamente a reglas gramaticales bien definidas, puede ser muy difícil construir un segmento de programa para el análisis sintáctico del lenguaje.

### 0.3 ANTECEDENTES HISTÓRICOS

La mayor parte del conocimiento científico es el resultado de muchos años de investigación, con frecuencia sobre temas aparentemente no relacionados. Esto sucede incluso en un campo tan joven como el de la ciencia de la computación. Una gran parte de los resultados que se presentan en los capítulos siguientes fue descubierta a principios del siglo XX por matemáticos que trabajaban en áreas como la lógica y los sistemas axiomáticos. Por lo tanto, es relevante para nuestro estudio hacer una breve introducción a este periodo matemático.

Al iniciar el siglo XX, las matemáticas estaban a punto de efectuar grandes descubrimientos. Los logros de los siguientes 40 años estaban destinados a sacudir las bases de las matemáticas y tuvieron consecuencias que se extendieron al campo de la ciencia de la computación, la cual se hallaba por nacer. Para poder comprender este periodo de transición, consideremos el enfoque axiomático para las matemáticas, un tema que recibió mucha atención al comenzar el siglo.

Poniéndolo de manera sencilla, el método axiomático requiere una colección de enunciados básicos, llamados **axiomas**, que describen las propiedades fundamentales del sistema que se estudia. Apartir de estos **axiomas**, se derivan

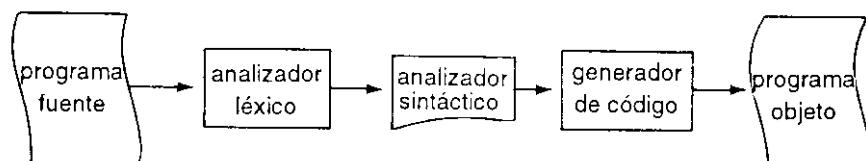


Figura 0.4 Proceso básico de traducción

enunciados adicionales, llamados **teoremas**, aplicando secuencias finitas de **reglas de inferencia**. Estas reglas están diseñadas para que cualquier teorema que se derive sea un enunciado que constituya una consecuencia lógica de los axiomas. Por ejemplo, una regla de inferencia bastante conocida es *modus ponens*, que establece que a partir de dos enunciados de la forma

si  $X$  entonces  $Y$

$y$

$X$

podemos derivar el enunciado  $Y$ . Por medio del *modus ponens*, podemos concluir "sopla el viento" a partir de los enunciados "los árboles se mueven" y "si los árboles se mueven, sopla el viento".

Una ventaja del método axiomático es que ofrece un modelo de razonamiento deductivo en el cual todas las suposiciones están aisladas en los axiomas iniciales y las reglas de inferencia. Se asegura que cualquier enunciado que se derive posteriormente sea una consecuencia de estas suposiciones, y sólo de estas suposiciones. Así, el método axiomático proporciona una importante herramienta que puede emplearse para pesquisas científicas.

A principios de este siglo, muchos matemáticos creían que todas las matemáticas podían basarse en un solo sistema axiomático, y su meta era encontrar el conjunto de axiomas y reglas de inferencias correctos. Así, cualquier teorema matemático podría derivarse de los axiomas con sólo aplicar una serie finita de reglas de inferencia. De esta manera, las matemáticas se verían reducidas a un sistema computacional en el cual podría determinarse algorítmicamente la veracidad o falsedad de cualquier enunciado matemático.

En 1931, surgió un descubrimiento sorprendente con la publicación del teorema de incompletitud de Kurt Gödel. En esencia, este teorema establece que para cualquier sistema axiomático válido con la riqueza suficiente para la parte de las matemáticas que consiste en los números naturales y las operaciones de suma y multiplicación, deben existir enunciados matemáticos cuya veracidad o falsedad no pueda probarse a partir del sistema. Es decir, la axiomatización de las matemáticas estaba condenada al fracaso.

El teorema de Gödel generó un gran interés por el poder de los métodos axiomáticos y los procesos computacionales. Si no existe ningún algoritmo general para demostrar teoremas matemáticos, entonces, ¿qué puede lograrse con los medios axiomáticos? Para responder esta pregunta, los matemáticos comenzaron a desarrollar máquinas computacionales teóricas (esto sucedió antes de que la tecnología fuera capaz de producir máquinas reales) y a estudiar las capacidades y limitaciones de estas máquinas. Los resultados de esta labor forman el núcleo del tema conocido como teoría de la computación, que hoy en día abarca subtemas como la teoría de lenguajes formales, la teoría de la computabilidad y la teoría de la complejidad, objetos de nuestro estudio.

Por lo tanto, no es ninguna sorpresa que la ciencia computacional teórica tenga un fuerte sabor matemático. Las preguntas de computabilidad a las

que se enfrentan los científicos teóricos de la computación son las mismas preguntas de demostrabilidad a las que se enfrentan los matemáticos, y viceversa. Básicamente, la diferencia se debe a la perspectiva.

## 0.4 ESBOZO DEL RESTO DEL TEXTO

Damos fin a este capítulo preliminar con un breve resumen de los capítulos restantes. En los capítulos 1 a 3 estudiamos tres clases de máquinas teóricas: autómatas finitos, autómatas de pila y máquinas de Turing, cada una de las cuales es más poderosa que la anterior. Consideraremos el poder de estas máquinas en el contexto de la resolución de problemas de reconocimiento de patrones. Es decir, nuestro interés se centrará en la identificación de los tipos de patrones de símbolos que pueden reconocer las distintas clases de máquinas.

Como ya hemos visto, los problemas de reconocimiento de patrones son parte medular de los sistemas de procesamiento de lenguajes, como los compiladores de los lenguajes de programación de alto nivel que existen en la actualidad. Por lo tanto, no nos sorprende que el material que se analiza en los tres capítulos siguientes proporcione la base teórica para la construcción de compiladores modernos, una relación entre la teoría y la práctica que examinaremos.

El material que se cubre en los capítulos 1 a 3 también se aplica en el campo del procesamiento de lenguajes naturales. Aprenderemos que la jerarquía de máquinas que se estudiará en los capítulos siguientes corresponde a la jerarquía de lenguajes descubierta en la década de los cincuenta por el lingüista N. Chomsky, quien se interesó por establecer una base formal para los estudios lingüísticos. Por esto, nuestro estudio de las máquinas de reconocimiento de patrones está estrechamente relacionado con la lingüística.

Los tres capítulos siguientes también nos llevarán a establecer un límite aparente para las capacidades computacionales de las máquinas. Nadie ha sido capaz de diseñar un tipo de máquinas (reales o teóricas) con mayor poder computacional que la clase de máquinas de Turing. Es más, una conjeta generalmente aceptada, conocida como tesis de Turing, dice que las máquinas de Turing poseen la capacidad suficiente para resolver cualquier problema para el cual existe una solución por medios computacionales. La tesis de Turing ofrece el contexto para formular nuestro estudio de la computabilidad.

En el capítulo 4 analizaremos evidencias que apoyan la tesis de Turing. Teniendo presente este objetivo, consideraremos otras áreas de investigación que han expuesto límites aparentes para los procesos computacionales y que muestran cómo estos límites son equivalentes a los de las máquinas de Turing. En las primeras secciones del capítulo 4 presentaremos el tema de la teoría de funciones recursivas, en el cual se identifica el límite aparente del poder computacional en una conjeta conocida como tesis de Church. A continuación mostraremos que la tesis de Church equivale a la tesis de Turing. Como

evidencia adicional, consideraremos los límites de los procesos computacionales en el contexto del diseño de lenguajes de programación, donde una vez más encontraremos un límite que está de acuerdo con la tesis de Turing.

Por último, en el capítulo 5 abordaremos la cuestión de la computabilidad desde un punto de vista práctico en vez de teórico. Aquí analizaremos algunos de los retos a los que se enfrentan los investigadores que intentan clasificar los problemas en función de los recursos requeridos para resolverlos por medios algorítmicos. Encontraremos que las soluciones algorítmicas de muchos problemas requieren tales cantidades de tiempo o espacio de almacenamiento que, desde un punto de vista práctico, los problemas permanecen sin solución, aunque existe una solución en las capacidades teóricas de los procesos computacionales. También veremos que los investigadores todavía no han podido clasificar gran cantidad de problemas. Por lo tanto, aún no existe respuesta a la pregunta de si estos problemas tienen o no solución.

En resumen, los capítulos siguientes presentan una amplia gama de temas. Estudiaremos temas tan abstractos como el poder computacional de las máquinas teóricas y las capacidades de los procesos computacionales. También veremos la relación entre estas ideas abstractas y situaciones tan reales como las técnicas de construcción de compiladores, el poder computacional de los lenguajes de programación, y preguntas con respecto a qué tan práctico es resolver ciertos problemas utilizando computadores.

## Problemas de repaso del capítulo

---

- Muestre que las siguientes igualdades (conocidas como leyes de DeMorgan) son verdaderas para tres conjuntos  $X$ ,  $Y$  y  $Z$  cualesquiera. (Para demostrar que  $A = B$ , demuestre que  $A \subseteq B$  y  $B \subseteq A$ ).

$$(X - Y) \cap (X - Z) = X - (Y \cup Z)$$

$$(X - Y) \cup (X - Z) = X - (Y \cap Z)$$

- Demuestre que para tres conjuntos  $X$ ,  $Y$  y  $Z$  cualesquiera:
  - $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$
  - $X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$
- Suponga que  $X = \{x: x \in \mathbb{N} \text{ y } x \text{ es impar}\}$ ,  $Y = \{y: y \in \mathbb{N} \text{ y } y \text{ es primo}\}$ , y  $Z = \{z: z \in \mathbb{N} \text{ y } z \text{ es múltiplo de tres}\}$ . Describa cada uno de los conjuntos siguientes.
  - $X \cap Y$
  - $Z - Y$
  - $(Y \cap Z) - X$
  - $X \cup (Y \cap Z)$
  - $X \cap Z$
  - $X - Y$
  - $(X \cap Y) \cap Z$
  - $Y \cap Z$
  - $X - (Y \cap Z)$
  - $X \cup Y$



4. Presente una lista con todos los elementos de cada uno de los conjuntos siguientes.
- $\{x, y\} \times \{a, b, c\}$
  - $\{a, b, c\} \times \{x, y\}$
  - $\{x, y\} \times \{k\} \times \{a, b\}$
  - $\{x, y\} \times \{\}$
5. Encuentre ejemplos de conjuntos no vacíos para los cuales sea verdadero cada uno de los enunciados siguientes.
- $X - Y = Y$
  - $A \times B = B \times A$
  - $P - Q = Q - P$
6. a. Defina una función  $f: \mathbb{N} \rightarrow \mathbb{N}$  que sea uno a uno pero no sobre.  
 b. Defina una función  $f: \mathbb{N} \rightarrow \mathbb{N}$  que sea sobre pero no uno a uno.
7. Demuestre que  $0 + 1 + 2 + \cdots + n = \frac{n(n + 1)}{2}$  para todos los valores de  $n$  que pertenezcan a  $\mathbb{N}$ .
8. Demuestre que  $1^2 + 2^2 + \cdots + n^2 = \frac{n(n + 1)(2n + 1)}{6}$  para todos los valores de  $n$  que pertenezcan a  $\mathbb{N}^+$ .
9. Demuestre que  $1^3 + 2^3 + \cdots + n^3 = \frac{n^2(n + 1)^2}{4}$  para todos los valores de  $n$  que pertenezcan a  $\mathbb{N}^+$ .
10. Demuestre que  $2^2 + 4^2 + \cdots + (2n)^2 = \frac{2n(n + 1)(2n + 1)}{3}$  para todos los valores de  $n$  que pertenezcan a  $\mathbb{N}^+$ .
11. Demuestre que  $1 + 2 + 2^2 + \cdots + 2^n = 2^{n+1} - 1$  para todos los valores de  $n$  que pertenezcan a  $\mathbb{N}$ .
12. Muestre que  $x^n - y^n$  es divisible por  $x - y$  para todo  $n$  en  $\mathbb{N}^+$ .
13. Muestre que si  $x$  es un número no negativo, entonces  $(1 + X)^n - 1 \geq nx$  para todos los valores de  $n$  en  $\mathbb{N}$ .
14. Demuestre que  $\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{n(n + 1)} = \frac{n}{n + 1}$  para todos los valores de  $n$  que pertenezcan a  $\mathbb{N}^+$ .
15. Presente una lista de todos los elementos de  $\mathcal{P}(\{x, y\})$ . Luego, genere una lista de todos los elementos de  $\mathcal{P}(\mathcal{P}(\{x, y\}))$ . Muestre que  $|\{x, y\}| \leq |\mathcal{P}(\{x, y\})| \leq |\mathcal{P}(\mathcal{P}(\{x, y\}))|$  identificando funciones uno a uno de  $\{x, y\}$  a  $\mathcal{P}(\{x, y\})$  y  $\mathcal{P}(\{x, y\})$  a  $\mathcal{P}(\mathcal{P}(\{x, y\}))$ .
16. Muestre que la cardinalidad del conjunto de personas registradas en el sistema de seguridad social de Estados Unidos no es mayor que el conjunto de enteros de 1 000 000 000 a 1 999 999 999.

17. Muestre que la cardinalidad de  $\mathcal{P}(X)$  es igual a la cardinalidad del conjunto de todas las funciones de  $X$  a  $\{0, 1\}$ .
18. Suponga que  $f_1, f_2, \dots$  es una lista infinita contable de funciones de  $\mathbb{N}^+$  a  $\mathbb{N}^+$ . Defina la función  $f: \mathbb{N} \rightarrow \mathbb{N}$  de manera que  $f(n) = f_n(n) + 1$  para cada  $n \in \mathbb{N}^+$ . Demuestre que la función  $f$  no es una de las funciones de la lista original.
19.
  - Proporcione un ejemplo de dos conjuntos disjuntos, no vacíos,  $A$  y  $B$  tales que  $|A| < |B| < |A \cup B|$ .
  - Proporcione un ejemplo de dos conjuntos disjuntos, no vacíos,  $A$  y  $B$  tales que  $|A| < |B| = |A \cup B|$ .
  - Proporcione un ejemplo de dos conjuntos disjuntos, no vacíos,  $A$  y  $B$  tales que  $|A| = |B| = |A \cup B|$ .
20. Tome su lenguaje de programación favorito e identifique los primeros tres programas que aparecerían en la lista descrita en los últimos párrafos de la sección 0.1.
21. Muestre que existe una cantidad por lo menos contable de frases en el idioma español.
22. Muestre que  $|\mathbb{N}| = |\mathbb{N} \times \mathbb{N} \times \mathbb{N}|$ .
23. Muestre que el conjunto de los números naturales pares tiene la misma cardinalidad que el conjunto de números naturales impares.
24. Muestre que la cardinalidad del conjunto de números primos es igual que la cardinalidad de  $\mathbb{N}^+$ .
25. Proporcione ejemplos para mostrar que la intersección de dos conjuntos incontables puede ser:
  - finita
  - infinita contable
  - incontable
26. Muestre que si  $X$  es un conjunto incontable y  $Y$  es un conjunto contable, entonces  $X - Y$  debe ser incontable.
27. Muestre que la unión de dos conjuntos contables es contable.
28. Muestre que un conjunto puede tener la misma cardinalidad que un subconjunto propio de sí mismo.
29. Muestre que el conjunto de todos los subconjuntos finitos de  $\mathbb{N}$  es contable.

30. Muestre que el conjunto de todos los números reales es incontable.
31. Muestre que cualquier conjunto infinito contable tiene una cantidad de subconjuntos infinitos de los que dos cualesquiera tienen sólo un número finito de elementos en común.

## CAPITULO 1

# Autómatas finitos y lenguajes regulares

---

### 1.1 Análisis léxico

Diagramas de transiciones

Tablas de transiciones

### 1.2 Autómatas finitos deterministas

Definiciones básicas

Diagramas de transiciones deterministas

Ejemplos de autómatas finitos deterministas

### 1.3 Límites de los autómatas finitos deterministas

Autómatas finitos deterministas como aceptadores  
de lenguajes

Un lenguaje no regular

### 1.4 Autómatas finitos no deterministas

### 1.5 Gramáticas regulares

### 1.6 Expresiones regulares

### 1.7 Comentarios finales

En este capítulo definimos y estudiamos la clase de máquinas teóricas conocida como autómatas finitos. Aunque su poder es limitado, encontraremos que estas máquinas son capaces de reconocer numerosos patrones de símbolos, los cuales identificamos con la clase de los lenguajes regulares.

Los autómatas finitos y los lenguajes regulares se encuentran en el nivel más bajo de la jerarquía de máquinas y lenguajes que estudiaremos en los tres capítulos siguientes. Esta base establece la importancia de tales máquinas y lenguajes desde una perspectiva teórica, pero su importancia no se limita a la teoría. Los conceptos que presentan los autómatas finitos y los lenguajes regulares son de interés fundamental para la mayoría de las aplicaciones que requieren técnicas de reconocimiento de patrones.

Una de estas aplicaciones es la construcción de compiladores. Por ejemplo,

un compilador debe ser capaz de reconocer cuáles son las cadenas de símbolos del programa fuente que deben considerarse como representaciones de objetos individuales, por ejemplo nombres de variables, constantes numéricas y palabras reservadas. Esta tarea de reconocimiento de patrones es manejada por el analizador léxico del compilador. En este capítulo se considerarán los principios básicos que rigen la construcción de analizadores léxicos; estos principios serán la base para una gran parte de nuestro estudio posterior.

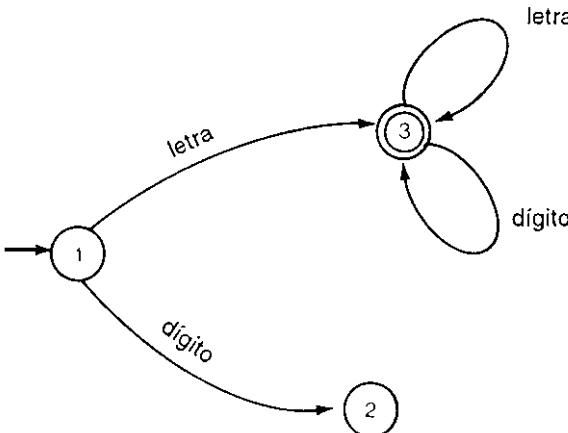
## 1.1 ANÁLISIS LÉXICO

Abordemos ahora un problema al que se enfrenta un compilador: detectar si una cadena del programa fuente representa o no un nombre de variable aceptable. En un lenguaje de programación típico, estos nombres comienzan con una letra, seguida por una combinación arbitraria (pero finita) de letras y dígitos. Así, X25, PepeRosas y x2y3z serían nombres aceptables, mientras que 25, x.h y Pepe-Rosas no lo serían. Además, cualquier estructura léxica en un lenguaje de programación termina con un conjunto de símbolos reconocido como fin de la estructura. A estos símbolos los llamaremos **marcas de fin de cadena**. En el caso de nombres de variables, estas marcas podrían ser espacios, puntos y comas, y retorno de carro.

### Diagramas de transiciones

Para desarrollar una unidad de programa que reconozca las ocurrencias de nombres de variables, nuestro primer paso podría ser representar de una manera concisa, no ambigua, la estructura de un nombre aceptable. Para este fin podemos utilizar la forma gráfica de un **diagrama de transiciones** (también llamado **diagrama de estado** o, en el campo del procesamiento de lenguajes naturales, **red de transiciones**), como se muestra en la figura 1.1. Un diagrama de transiciones es una colección finita de círculos, los cuales se pueden rotular para fines de referencia, conectados por flechas que reciben el nombre de **arcos**. Cada uno de estos arcos se etiqueta con un símbolo o categoría de símbolos (p. ej., "dígito" o "letra") que podría presentarse en la cadena de entrada que se analiza. Uno de los círculos se designa con un apuntador, y representa una posición inicial. Además, por lo menos uno de los círculos se representa como un círculo doble; estos círculos dobles designan posiciones del diagrama en las cuales se ha reconocido una cadena válida.

Decimos que una cadena de símbolos es aceptada por un diagrama de transiciones si los símbolos que aparecen en la cadena (de izquierda a derecha) corresponden a una secuencia de arcos rotulados que conducen del círculo designado por el apuntador a un círculo doble. Así, al analizar la cadena X25 utilizando la figura 1.1, partimos del círculo inicial siguiendo el arco con



**Figura 1.1** Diagrama de transiciones que representa la sintaxis de un nombre de variable

etiqueta “letra” (ya que la cadena comienza con X). A partir del círculo 3, recorremos dos veces (una para el 2 y otra para el 5) el arco llamado “dígito” y nos encontramos en el doble círculo 3. Por el contrario, si utilizamos la cadena 25, comenzamos en el mismo círculo inicial, pero esta vez seguimos el arco rotulado “dígito” y nos encontramos en un callejón sin salida. Concluimos entonces que X25 es un nombre de variable aceptable, no así 25.

Observe que los círculos en un diagrama de transiciones representan posiciones, o estados, donde nos podemos encontrar al evaluar una cadena de símbolos. Haciendo referencia a la figura 1.1, vemos que el círculo 1 se puede indentificar como el estado inicial del análisis de una cadena que el círculo 2 representa el estado de haber localizado un dígito como primer símbolo de la cadena, y que el círculo 3 representa el estado de haber encontrado una cadena que comienza con una letra. Por esto, es común llamar estados a los círculos de un diagrama de transiciones. El círculo de partida se llama **estado inicial** y los círculos dobles, **estados de aceptación**.

Una vez que hemos desarrollado un diagrama de transiciones que acepta únicamente las estructuras sintácticas que constituyen un nombre de variable válido, no es difícil escribir un programa que reconozca estas estructuras. Por ejemplo, el diagrama de la figura 1.1 sugiere el segmento de programa que se muestra en la figura 1.2. Observe que la estructura básica del algoritmo producido es la de un solo enunciado case que dirige las actividades de acuerdo con el estado actual. Las opciones posibles para cada estado se manejan utilizando estructuras condicionales adicionales, como los enunciados if-then-else anidados. En el caso de una cadena inaceptable, el segmento de programa sale a una rutina de error que, por medio de un parámetro, puede emplearse para imprimir un mensaje adecuado al respecto, como “El nombre de la variable debe comenzar con una letra” o “Identificador no válido”.

```

Estado := 1;
Leer el siguiente símbolo de la entrada;
while no es fin-de-cadena do
    case Estado of
        1: if símbolo actual es una letra then Estado:= 3,
            else if símbolo actual es un dígito then Estado:= 2,
            else salir a la rutina de error;
        2: Salir a la rutina de error;
        3: if símbolo actual es una letra then Estado:= 3,
            else if símbolo actual es un dígito then Estado:= 3,
            else salir a la rutina de error;
    Leer el siguiente símbolo de la entrada;
end while;
if Estado no es 3 then salir a la rutina de error;

```

**Figura 1.2** Serie de instrucciones sugerida por el diagrama de transiciones de la figura 1.1

Vemos entonces que los diagramas de transiciones se pueden emplear como herramientas de diseño para producir rutinas de análisis léxico, de manera similar a como los ingenieros de software utilizan los diagramas de flujo y de estructura. No obstante, el código generado directamente a partir de un diagrama de transiciones no siempre representa la mejor solución al problema; se obtiene una solución más elegante si se emplean tablas de transiciones.

## Tablas de transiciones

Una **tabla de transiciones** es un arreglo (o matriz) bidimensional cuyos elementos proporcionan el resumen de un diagrama de transiciones correspondiente. Para elaborar una tabla de este tipo construimos primero un arreglo, con una fila para cada estado del diagrama de transiciones y una columna para cada símbolo o categoría de símbolos que podría ocurrir en la cadena de entrada. El elemento que se encuentra en la fila  $m$  y la columna  $n$  es el estado que se alcanzaría en el diagrama de transiciones al dejar el estado  $m$  a través de un arco con etiqueta  $n$ . Si no existe arco n alguno que salga del estado  $m$ , entonces la casilla correspondiente de la tabla se marca como un estado de error. Con la finalidad de completar la tabla de transiciones, agregamos una columna rotulada "FDC" para el fin de cadena. La casilla en la columna FDC contiene el valor "aceptar" si la fila corresponde a un estado de aceptación del diagrama, o contiene el valor "error", en caso contrario. La figura 1.3 representa una tabla de transiciones obtenida a partir del diagrama de transiciones de la figura 1.1.

Es bastante sencillo diseñar un analizador léxico basándose en una tabla de transiciones. Lo único que tenemos que hacer es asignar a una variable un valor inicial, correspondiente al estado inicial, y luego actualizar repetidamente esta variable con base en la tabla, conforme se lean los símbolos de la cadena de entrada, hasta llegar al fin de la cadena. Por ejemplo, en la figura 1.4 se presenta un analizador léxico basado en la figura 1.3.

Como un ejemplo más, las figuras 1.5, 1.6 y 1.7 muestran un diagrama de transiciones, una tabla de transiciones y un analizador léxico, respectivamente, para reconocer cadenas que representan números reales positivos en notación decimal o exponencial, como 35.7, 2.56E10 o 34.0E-7. Observe que la estructura de la figura 1.7 es esencialmente igual que la figura 1.4; la única diferencia está en la rutina que clasifica el símbolo de entrada. De hecho, el algoritmo fundamental para el análisis de una cadena utilizando una tabla de transiciones es el mismo para cualquier estructura léxica.

En resumen, hemos presentado algunas técnicas bastante sencillas para desarrollar analizadores léxicos. Ahora, lo que necesitamos descubrir es el grado en el cual pueden aplicarse estas técnicas. ¿Es posible utilizar segmentos de programa basados en diagramas de transiciones para analizar cualquier estructura sintáctica, o existen estructuras que no pueden reconocerse con estos diagramas? Si existen límites para estas técnicas, ¿qué se requiere para manejar los casos más complejos? Atenderemos estas preguntas en las secciones restantes de este capítulo, y continuaremos nuestra búsqueda por los capítulos 2 y 3.

	letra	dígito	FDC
1	3	2	error
2	error	error	error
3	3	3	aceptar

---

Figura 1.3 Tabla de transiciones construida a partir del diagrama de transiciones de la figura 1.1

Estado := 1;

repeat

Leer el siguiente símbolo del flujo de entrada;

case símbolo of

letra: Entrada := "letra";

dígito: Entrada := "dígito";

marca de fin de cadena: Entrada := "FDC";

ninguno de los anteriores: salir a la rutina de error;

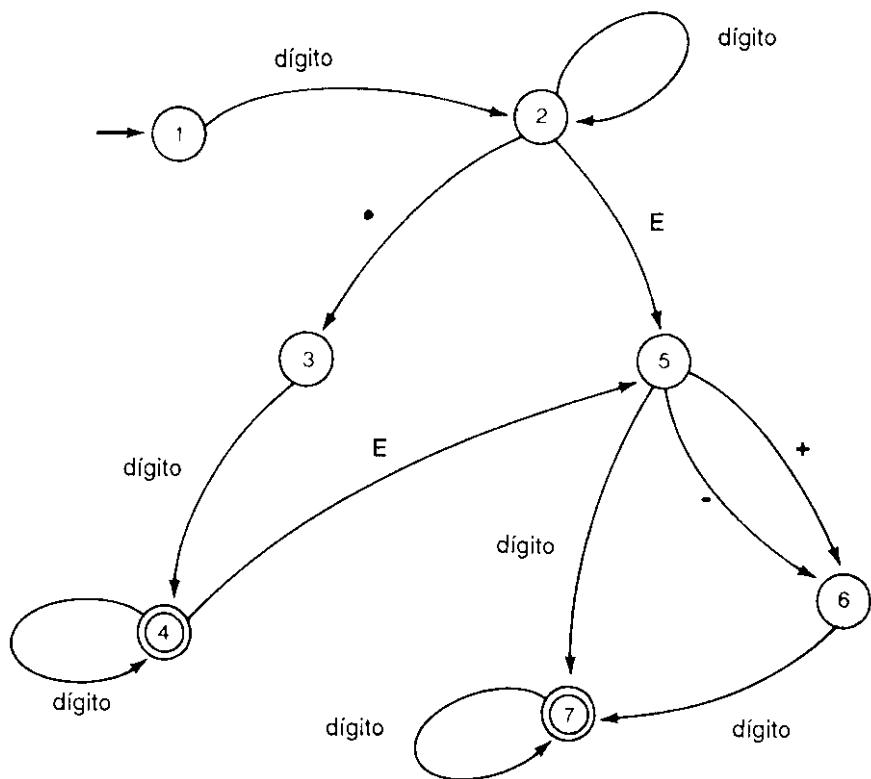
Estado := Tabla [Estado, Entrada];

if Estado = "error" then salir a la rutina de error;

until Estado = "aceptar"

---

Figura 1.4 Analizador léxico basado en la tabla de transiciones de la figura 1.3



**Figura 1.5** Diagrama de transiciones que representa la sintaxis de un número real

	dígito	.	E	+	-	FDC
1	2	error	error	error	error	error
2	2	3	5	error	error	error
3	4	error	error	error	error	error
4	4	error	5	error	error	aceptar
5	7	error	error	6	6	error
6	7	error	error	error	error	error
7	7	error	error	error	error	aceptar

**Figura 1.6** Tabla de transiciones construida a partir del diagrama de transiciones de la figura 1.5

Estado: = 1;

**repeat**

Leer el siguiente símbolo del flujo de entrada;

**case** símbolo **of**

0 a 9: Entrada := "dígito";

Marca de fin de cadena: Entrada := "FDC";

•, E, +, - Entrada := símbolo;

Ninguno de los anteriores: salir a la rutina de error;

Estado := Tabla [Estado, Entrada];

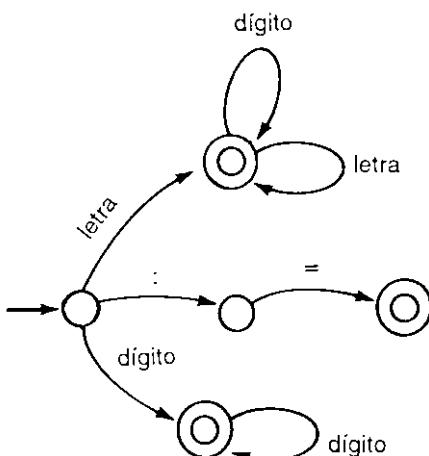
**if** Estado = "error" **then** salir a la rutina de error;

**until** Estado = "aceptar"

**Figura 1.7** Analizador léxico basado en la tabla de transiciones de la figura 1.6

## Ejercicios

1. Diseñe un diagrama de transiciones para reconocer expresiones aritméticas de longitud arbitraria que comprenden enteros positivos separados por signos de suma, resta, multiplicación o división.
2. Escriba un analizador léxico directamente a partir del siguiente diagrama de transiciones.



3. Construya una tabla de transiciones a partir del diagrama del ejercicio 2 y escriba un analizador léxico basado en esa tabla.
4. Muestre que se puede modificar un diagrama de transiciones que contiene un arco rotulado por una cadena de símbolos de longitud dos o más (lo cual significa que para recorrer el arco se requiere la cadena completa y no un solo símbolo) para que contenga sólo arcos rotulados con símbolos sencillos, pero de manera que siga aceptando las mismas cadenas que antes.

## 1.2 AUTÓMATAS FINITOS DETERMINISTAS

Concluimos la sección anterior preguntando si los diagramas de transiciones proporcionan una herramienta con el poder suficiente para desarrollar programas que reconozcan estructuras sintácticas de complejidad arbitraria. En esta sección atendemos esta pregunta, formalizando con mayor precisión el proceso de reconocimiento de estructuras por medio de diagramas de transiciones. Nuestra meta es identificar las características pertinentes de un sistema de reconocimiento de patrones construido sobre el principio de los diagramas de transiciones, para que podamos estudiar de manera genérica el potencial de un sistema de reconocimiento de patrones en vez de presentar cada ejemplo como un caso aislado.

### Definiciones básicas

Para iniciar esta tarea de formalización, reconocemos que las cadenas que deben analizarse en una aplicación están construidas a partir de un conjunto de símbolos. En el caso de un analizador léxico en un compilador, estas cadenas generalmente están formadas por los símbolos disponibles en el teclado de una terminal de computador. Sin embargo, en un computador digital moderno todos estos símbolos están representados por patrones de ceros y unos. A partir de esta perspectiva más elemental, cualquier cadena consiste en una combinación de sólo dos símbolos. Por lo tanto, dependiendo del punto de vista, varía la colección de símbolos utilizada para construir cadenas. No obstante, en cualquier situación encontramos que el conjunto de símbolos es finito, por lo que nuestro primer paso hacia la formalización del proceso de reconocimiento es asumir la hipótesis de la existencia de un conjunto finito, no vacío, de símbolos a partir del cual se construyen las cadenas que se analizarán. A este conjunto lo llamamos **alfabeto**.

A continuación vemos que cada cadena que se recibe se analizará como una secuencia de símbolos, uno a la vez. Nos referimos a la fuente de esta secuencia como el **flujo de entrada** (Fig. 1.8). Conforme llega cada símbolo del flujo de entrada, nuestro proceso de reconocimiento implica cambiar de un estado, tomado de entre una cantidad finita de ellos, a otro, o bien permanecer en el estado actual. El nuevo estado dependerá únicamente del estado actual y del símbolo que se recibe. Para subrayar este punto, observe que, una vez que el proceso de reconocimiento ha llegado al estado 5 de la figura 1.5, su estado siguiente no dependerá de si ha llegado al estado 5 proveniente del estado 2 o del estado 4; en cambio, el estado siguiente estará determinado sólo por el siguiente símbolo que se reciba y por estar en el estado 5. En análisis subsecuentes se verá la importancia de que los sistemas de reconocimiento basados en estos diagramas de transiciones no puedan recordar cómo llegaron al lugar donde se encuentran.

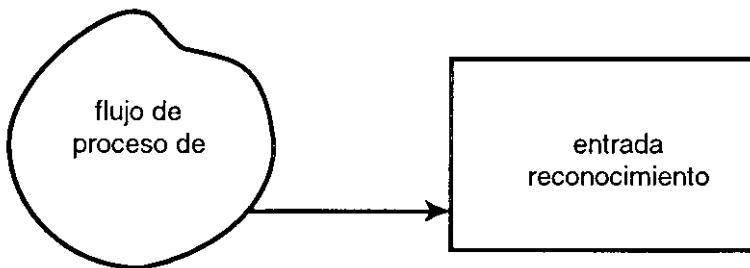
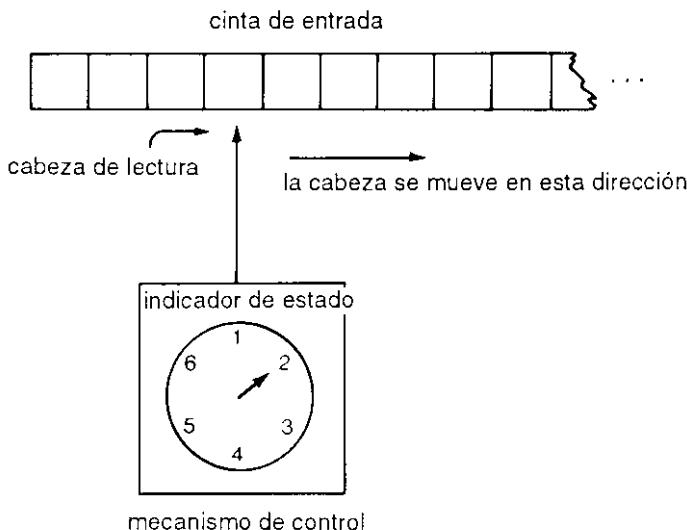


Figura 1.8 Flujo de entrada como fuente de símbolos.

Los conceptos que tenemos hasta ahora se fortalecen definiendo una máquina conceptual conocida como **autómata finito determinista**. Esta máquina consiste en un dispositivo que puede estar en cualquiera de un número finito de estados, uno de los cuales es el estado inicial y por lo menos uno es un estado de aceptación. A este dispositivo está unido un flujo de entrada por medio del cual llegan en secuencia los símbolos de un alfabeto determinado. La máquina tiene la capacidad para detectar los símbolos conforme llegan y, basándose en el estado actual y el símbolo recibido, ejecutar una transición (de estado) que consiste en un cambio a otro estado o la permanencia en el estado actual. La determinación de cuál será precisamente la transición que ocurrirá al recibir un símbolo depende de un **mechanismo de control** de la máquina, programado para conocer cuál debe ser el nuevo estado dependiendo de la combinación del estado actual y el símbolo de entrada.

Debemos hacer hincapié en que el programa de un autómata finito determinista no debe contener ambigüedades, lo mismo que sucede con cualquier programa para computador. Éste es un punto importante que analizaremos pronto con mayor detalle. Por el momento, observamos que ésta es la razón detrás de la palabra "determinista" en "autómata finito determinista". La palabra "finito" se refiere a que la máquina sólo tiene un número finito de estados. En ocasiones se hace mención de estas máquinas como autómatas deterministas de estado finito.

Tradicionalmente, un autómata finito determinista se visualiza de la manera presentada en la figura 1.9. El mecanismo de control de la máquina está representado por un rectángulo que contiene una especie de carátula de reloj. Los estados posibles están representados en la circunferencia de este reloj y un apuntador indica el estado actual. El flujo de entrada a la máquina aparece como una cinta de papel dividida en celdas, cada una de las cuales es capaz de almacenar un solo símbolo. Consideraremos que la cinta tiene un extremo izquierdo, pero que se extiende infinitamente hacia la derecha. La máquina es capaz de leer los símbolos de esta cinta, de izquierda a derecha, por medio de una cabeza de lectura cuya posición en la figura 1.9 está indicada por una flecha. Cada vez que se lee un símbolo, la cabeza de lectura se mueve a la



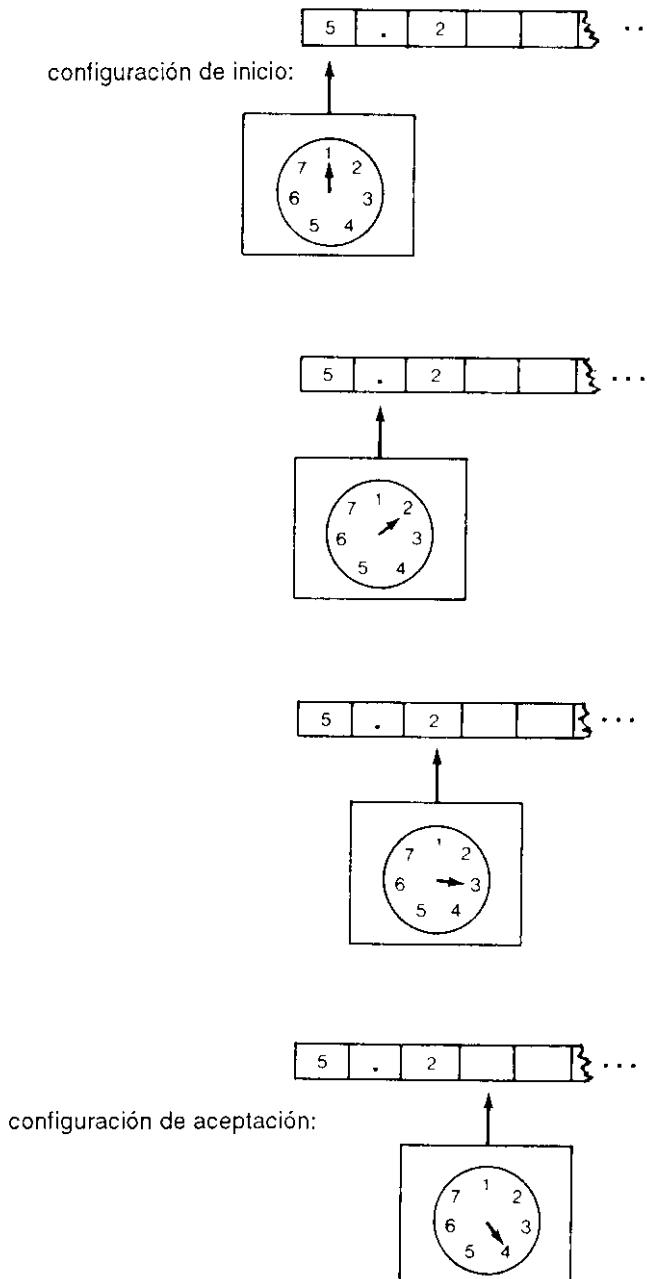
**Figura 1.9** Representación de un autómata finito determinista

siguiente posición de la cinta. Así, la posición donde descansa la cabeza de lectura corresponde a la siguiente celda que se leerá.

Para representar un programa en el mecanismo de control, utilizamos un diagrama de transiciones cuyos estados representan los estados de la máquina y cuyos arcos representan una posible transición de la máquina. En este contexto, los estados de inicio y aceptación del diagrama corresponden a los estados de inicio y de aceptación del autómata.

Decimos que un autómata finito determinista acepta su cadena de entrada si, después de comenzar sus cálculos en el estado inicial con la cabeza de lectura sobre el primer símbolo de la entrada, la máquina cambia a un estado de aceptación después de leer el último símbolo de la cadena (véase Fig. 1.10). Si después de leer el último símbolo de la cadena la máquina no queda en un estado de aceptación, decimos que la cadena ha sido rechazada. Por ende, un autómata finito determinista es, en esencia, una máquina analizadora de cadenas que acepta aquellas cadenas aceptadas por su diagrama de transiciones interno y rechaza todas las demás.

Surge un caso ligeramente distinto si la máquina llega al final de su entrada antes de leer algún símbolo (es decir, si la entrada comienza con una marca de fin de cadena). En este caso decimos que la entrada es una cadena vacía (una cadena que no contiene símbolos). Por desgracia, una cadena vacía no aparece muy bien en una página impresa, por lo que la representamos con el símbolo  $\lambda$ . Recalcamos que la cadena vacía no contiene símbolos; por lo tanto, un autómata finito determinista aceptará  $\lambda$  si y sólo si su estado inicial es también un estado de aceptación.



**Figura 1.10** Pasos efectuados por un autómata finito determinista (programado por el diagrama de la Fig. 1.5) al procesar la cadena "5.2"

Por último, debemos hacer notar que la tecnología de cintas, cabezas de lectura y carátulas de reloj con la cual describimos los autómatas finitos deterministas no es un factor crítico en nuestra definición. En cambio, esta implantación es tan sólo un modelo informal que nos ayuda a recordar las propiedades de los autómatas finitos deterministas. Pueden construirse máquinas con las mismas propiedades de computación utilizando diversas tecnologías, como veremos al final de esta sección, y cada una de estas máquinas debe reconocerse como un autómata finito determinista. Así, reanudaremos ahora nuestro análisis con una definición formal y precisa de un autómata finito determinista, que identifica las características pertinentes de estas máquinas y que puede servir como referencia decisiva.

Un autómata finito determinista consiste en una quíntupla  $(S, \Sigma, \delta, i, F)$  donde:

- $S$  es un conjunto finito de estados.
- $\Sigma$  es el alfabeto de la máquina.
- $\delta$  es una función (llamada **función de transición**) de  $S \times \Sigma$  a  $S$ .
- $i$  (un elemento de  $S$ ) es el estado inicial.
- $F$  un subconjunto de  $S$  es el conjunto de estados de aceptación.

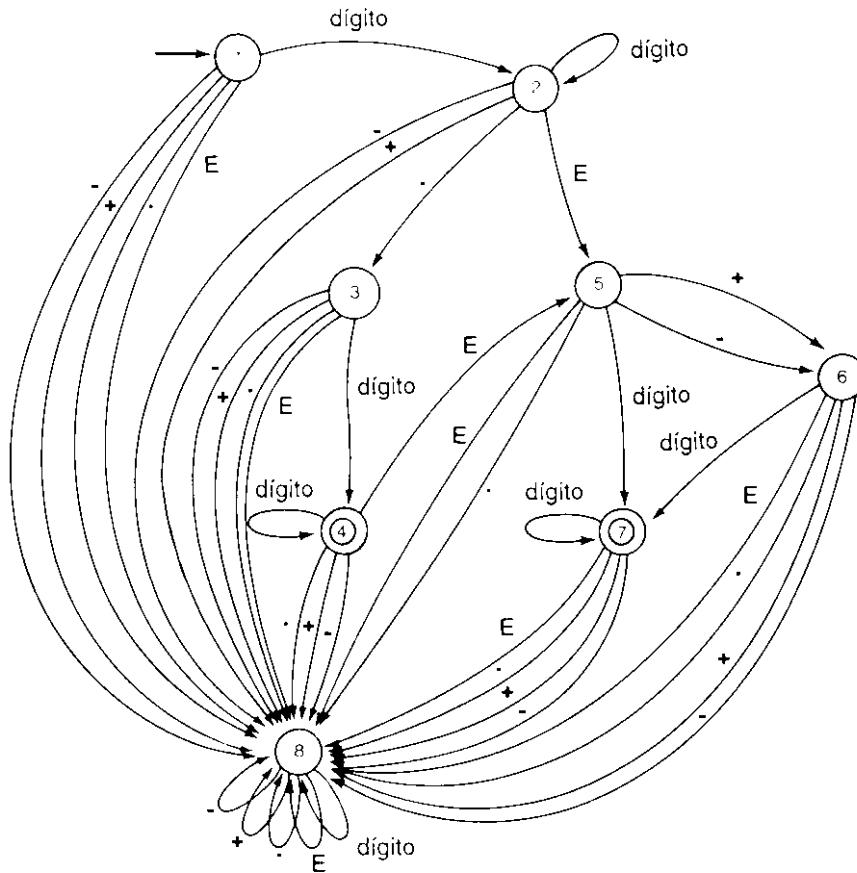
La interpretación de la función de transición  $\delta$  de un autómata finito determinista es que  $\delta(p, x) = q$  si y sólo si la máquina puede pasar de un estado  $p$  a un estado  $q$  al leer el símbolo  $x$ . Así, un diagrama de transiciones para un autómata finito determinista no es más que una representación gráfica de la función de transición de la máquina. De aquí se desprende que el autómata  $(S, \Sigma, \delta, i, F)$  acepta la cadena no vacía  $x_1 x_2 \cdots x_n$  si y sólo si existe una serie de estados  $s_0, s_1, \dots, s_n$  tal que  $s_0 = i, s_n \in F$ , y para cada entero  $j$  de 1 a  $n$ ,  $\delta(s_{j-1}, x_j) = s_j$ .

## Diagramas de transiciones deterministas

El requisito del determinismo impone ciertas restricciones sobre los diagramas de transiciones que pueden aparecer en los programas para un autómata finito determinista. En particular, cada estado de estos diagramas sólo debe tener un arco que sale para cada símbolo del alfabeto; de lo contrario, una máquina que llega a ese estado se enfrentará a una elección de cuál debe ser el arco a seguir. Además, dicho diagrama deberá estar completamente definido, es decir, debe existir por lo menos un arco para cada símbolo del alfabeto; de lo contrario, una máquina que llega a ese estado puede enfrentarse a una situación donde no puede aplicarse ninguna transición. Observe que estos requisitos están reflejados en nuestra definición formal, primero porque  $\delta$  es una función y, segundo porque su dominio es  $S \times \Sigma$ .

Se dice que un diagrama de transiciones es **determinista** si cumple estas dos condiciones. Entonces, desde un punto de vista técnico, el diagrama de la

figura 1.1 no es determinista ya que no está completamente definido: no representa cuál será la acción que debe ocurrir si se recibe una letra o un dígito mientras se encuentre en el estado 2. El diagrama de la figura 1.5 tiene problemas similares ya que, entre otras cosas, no describe lo que deberá suceder si recibe un punto mientras se encuentra en el estado 1. No obstante, los dos diagramas no tienen más de un arco de salida de un estado para cada símbolo y, por consiguiente, pueden modificarse para ajustarse a los requisitos del determinismo. Primero añadimos un estado adicional que representará un papel de captación global. Luego, para cada símbolo del alfabeto, dibujamos un arco, rotulado con dicho símbolo, que empieza y termina en este nuevo estado. Por último, agregamos arcos de los otros estados a este nuevo, hasta que cada uno de los estados sea el origen de un arco para cada símbolo del alfabeto.



**Figura 1.11** Diagrama “completo” de la figura 1.5

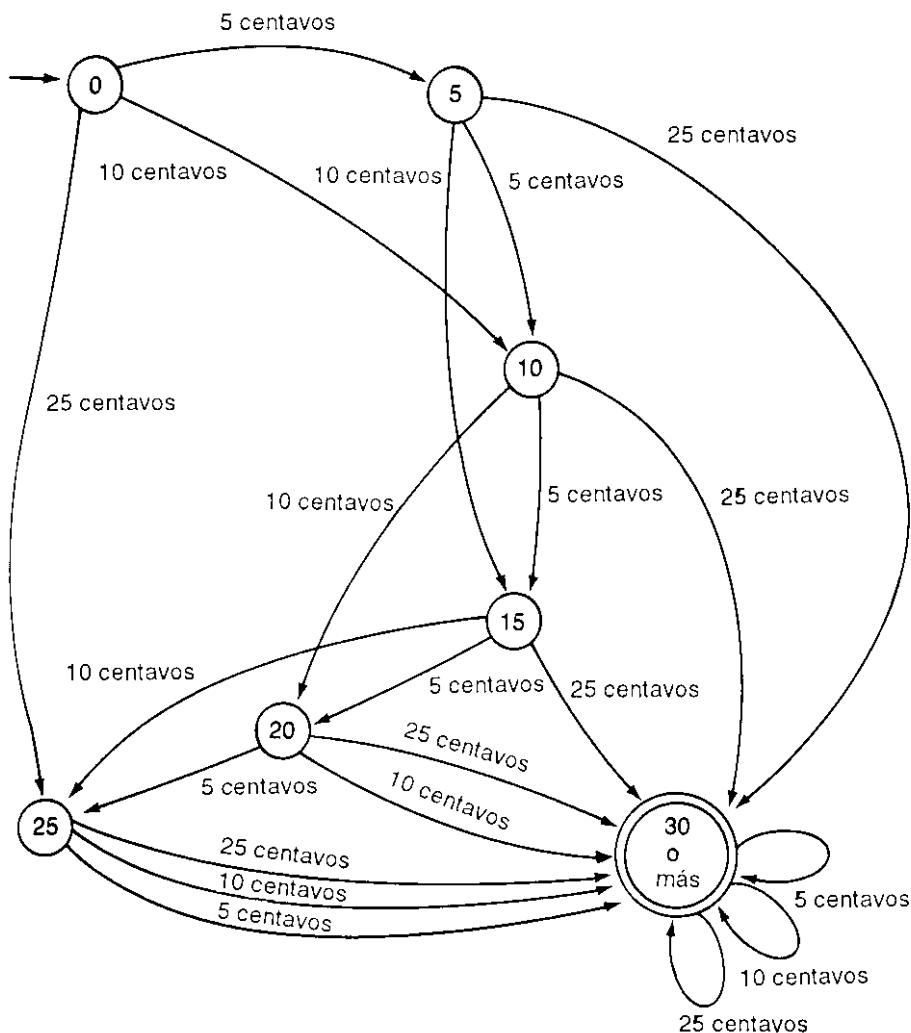
En la figura 1.11 se muestra el resultado de aplicar esta técnica al diagrama de transiciones de la figura 1.5. El nuevo estado es el número 8. Observe que en el diagrama original la ocurrencia de una cadena inaceptable ocasionaba un error al solicitar el recorrido de un arco inexistente. En el diagrama modificado, una cadena inaceptable ocasiona que la máquina recorra un arco al estado 8, donde permanece hasta alcanzar el final de la cadena de entrada. Al llegar a este punto se rechazará la cadena, ya que el estado 8 no es de aceptación. Por esto, los dos diagramas son equivalentes en lo que se refiere a que aceptan las mismas cadenas; difieren tan sólo en la manera en que llegan a sus conclusiones. La figura 1.11 también demuestra por qué, al dibujar diagramas de transiciones deterministas, nos vemos tentados a no representar todas las transiciones. De hecho, si se incluyen todas las transiciones, fácilmente saturamos los diagramas, lo que a su vez oculta la estructura significativa del autómata que se representa. Por consiguiente, con frecuencia se dibujan versiones parciales, o esqueletos, de los diagramas de transiciones deterministas, en los cuales se omiten el estado de captación global y los arcos correspondientes. Este es el caso de las figuras 1.1 y 1.5.

## Ejemplos de autómatas finitos deterministas

Hemos visto los autómatas finitos deterministas en el contexto de los computadores digitales modernos, por lo que no es ninguna sorpresa que nuestros modelos se orienten en esa dirección. Sin embargo, el concepto de autómata finito determinista no se restringe al ambiente de computadores tradicional: estos autómatas están en todas partes. Considere una máquina vendedora que entrega a una persona el caramelo elegido después de recibir un total de 30 centavos en monedas de 5, 10 y 25 centavos. En este caso, el alfabeto del autómata consiste en tres tamaños de monedas distintos, un estado de la máquina es la cantidad total de dinero que ha recibido desde que se entregó el último caramelo, el estado inicial es no haber recibido ninguna moneda desde la entrega del último caramelo y un estado de aceptación es recibir al menos 30 centavos. Suponiendo que la máquina no está diseñada para entregar cambio, con lo que aceptaría cualquier sobrepago, su diagrama de transiciones se parecería al de la figura 1.12. Se acepta una cadena de entrada si conduce al estado de aceptación (haber recibido por lo menos 30 centavos). En este estado, la máquina entregará un caramelo cuando el operador oprima el botón que indica su elección.

Encontramos otro ejemplo de un autómata finito determinista al efectuar una llamada telefónica. El teléfono es un dispositivo que recibe cadenas de dígitos introducidas por medio de un disco o un sistema de botones. Los estados de esta máquina incluyen estar en modo de larga distancia (cuando la clave adecuada se encuentra al inicio de la cadena), una solicitud de ayuda a la operadora o la recepción de un número telefónico válido.

Hemos considerado estos ejemplos de autómatas finitos deterministas distintos a los computadores para tratar de ampliar nuestra perspectiva más allá de

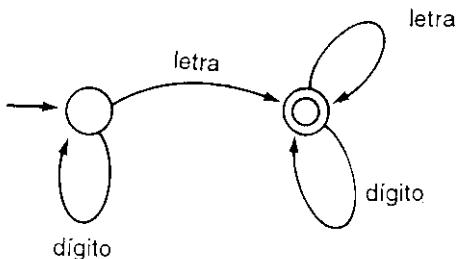


**Figura 1.12** Diagrama de transiciones para una máquina vendedora

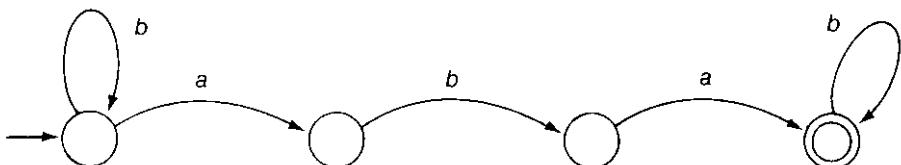
los analizadores léxicos. Sin embargo, al reconocer que estos dispositivos son autómatas finitos deterministas nos percatamos de que su acción es, en esencia, la de un análisis léxico. La máquina vendedora analiza cadenas de monedas; el teléfono analiza cadenas de dígitos. Así, si abstraemos los ingredientes esenciales del análisis léxico que se encuentran en un compilador, en otros ambientes observamos detalles que de otra manera pasarían desapercibidos. Aunque este ejemplo específico no tiene una trascendencia significativa, sí constituye una muestra del poder de las teorías abstractas.

## Ejercicios

1. Describa las cadenas que acepta el autómata finito determinista representado en el siguiente diagrama de transiciones.



2. Modifique el siguiente esqueleto de diagrama de transiciones de manera que esté completamente definido y acepte las mismas cadenas que antes.



3. Identifique otro autómata finito determinista de la vida diaria y dibuje su diagrama de transiciones.

## 1.3 LÍMITES DE LOS AUTÓMATAS FINITOS DETERMINISTAS

Hasta ahora hemos presentado la clase de máquinas conocidas como autómatas finitos deterministas y hemos analizado su relación con los analizadores léxicos. Ahora, nuestro problema es determinar el grado hasta el cual pueden construirse algoritmos de reconocimiento de patrones a partir de estos principios. De manera específica, lo que nos preguntamos es si el empleo de tablas de transiciones, como se presentaron en la sección 1.1, ofrece la flexibilidad suficiente para el procesamiento general de cadenas.

## Autómatas finitos deterministas como aceptadores de lenguajes

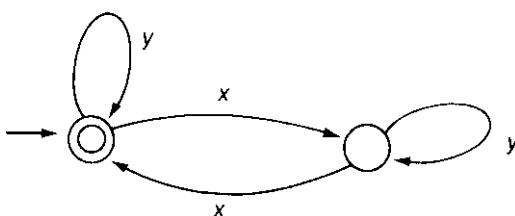
Reconsideremos la tarea que queremos que realicen los autómatas finitos deterministas: aceptar sólo aquellas cadenas que se adhieran a ciertas reglas de composición. En otras palabras, cada autómata finito determinista se puede considerar como un agrupador de todas las cadenas de entrada potenciales en dos categorías: las cadenas que son aceptables y las que no lo son.

Para caracterizar esta tarea en términos más formales, primero definimos la longitud de una cadena  $w$  como el número de símbolos que contiene y a este valor lo representamos con  $|w|$ . Así,  $|xy|$  es dos y  $|xyz|$  es tres. Luego, consideraremos la colección de todas las cadenas de longitud finita (incluyendo la cadena vacía) que pueden construirse a partir del alfabeto que se utiliza. Suponiendo que el alfabeto es denotado con  $\Sigma$ , este conjunto de cadenas está representado por  $\Sigma^*$ . Por lo tanto, si  $\Sigma$  fuera  $\{a,b\}$ , entonces  $\Sigma^*$  sería  $\{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$ .

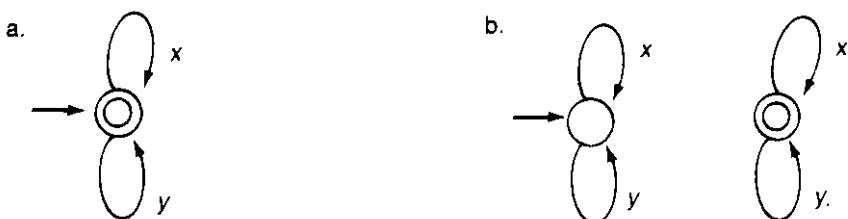
Un subconjunto de  $\Sigma^*$  se llama lenguaje (del alfabeto  $\Sigma$ ), de acuerdo con nuestro empleo tradicional del término. De hecho, si  $\Sigma$  contiene todas las letras, signos de puntuación y dígitos, así como el espacio en blanco, entonces todas las frases en español serían un subconjunto de  $\Sigma^*$ , mientras que otro subconjunto consistiría en las frases en latín. Por supuesto, existirían también subconjuntos de  $\Sigma^*$  que no satisfarían nuestra definición intuitiva de un lenguaje, pero para nuestros fines se seguirán considerando como lenguajes. Por lo tanto, el conjunto de cadenas  $\{a, ab, b\}$  se considerará un lenguaje del alfabeto  $\{a, b\}$ .

Si  $M$  es un autómata finito determinista  $(S, \Sigma, \delta, s_0, F)$ , la colección de cadenas que acepta constituye un lenguaje con respecto al alfabeto  $\Sigma$ . Este lenguaje se representa con  $L(M)$ , que se lee "el lenguaje que acepta  $M$ ". Subrayamos que  $L(M)$  no es cualquier colección de cadenas que acepta  $M$ , sino la colección de todas las cadenas que acepta  $M$ , ni una más ni una menos. Un lenguaje de la forma  $L(M)$  para un autómata finito  $M$  se denomina **lenguaje regular**.

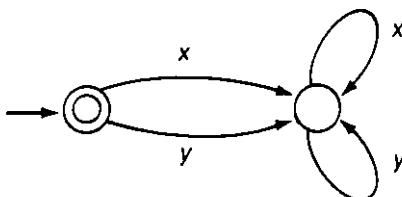
Abundan los ejemplos de lenguajes regulares. Para obtener uno nos basta con dibujar un diagrama de transiciones para un autómata finito determinista



**Figura 1.13** Diagrama de transiciones para un autómata finito determinista que acepta exactamente aquellas cadenas de  $x$  y  $y$  que contienen un número par de  $x$



**Figura 1.14** Diagramas de transiciones para autómatas finitos deterministas que aceptan a) el lenguaje  $\{x, y\}^*$  y b) el lenguaje  $\emptyset$



**Figura 1.15** Diagrama de transiciones para un autómata finito determinista que acepta el lenguaje  $\{\lambda\}$

y luego identificar el lenguaje que acepta. Como muestra, la figura 1.13 presenta un diagrama de transiciones que acepta exactamente aquellas cadenas de  $x$  y  $y$  que contienen un número par de  $x$  y cualquier número de  $y$ . Así, este lenguaje (del alfabeto  $\{x, y\}$ ) es regular. Un par de lenguajes regulares algo especiales es la colección de todas las cadenas de longitud finita de un alfabeto  $\Sigma$ , que ya representamos con  $\Sigma^*$ , y la colección que no contiene cadenas, conocida como lenguaje vacío, que representamos con  $\emptyset$ . En la figura 1.14 se presentan diagramas de transiciones para estos lenguajes con respecto al alfabeto  $\{x, y\}$ .

Otro ejemplo de un lenguaje regular es el que consiste en una cadena vacía, el cual denotamos con  $\{\lambda\}$ . Éste es el lenguaje que acepta el autómata finito determinista cuyo diagrama de transiciones se muestra en el figura 1.15. Observe la distinción que se hace entre  $\{\lambda\}$  y  $\emptyset$ : el primero contiene una cadena, mientras que el segundo no contiene ninguna.

Nótese que los lenguajes regulares son aquellos para los cuales son aplicables las técnicas de análisis léxico de la sección 1.1. Por esto, el teorema siguiente, que establece la existencia de lenguajes no regulares, tiene impor-

tancia con respecto a nuestra pregunta acerca del poder de esta técnicas. En resumen, las técnicas de análisis léxico de la sección 1.1 no tienen la flexibilidad suficiente para el procesamiento general de cadenas.

### TEOREMA 1.1

Para cualquier alfabeto  $\Sigma$ , existe un lenguaje que no es igual a  $L(M)$  para cualquier autómata finito determinista  $M$ .

### DEMOSTRACION

Puesto que cualquier arco de un autómata finito determinista que esté rotulado con un símbolo fuera de  $\Sigma$  no tendrá efecto alguno sobre el procesamiento de una cadena en  $\Sigma^*$ , podemos considerar sólo las máquinas con alfabeto  $\Sigma$ . Sin embargo, la colección de autómatas finitos deterministas con alfabeto  $\Sigma$  es contable ya que podemos elaborar en forma sistemática una lista de todas las máquinas posibles con un estado, seguidas por todas las máquinas con dos estados, luego por aquéllas con tres estados, etcétera. Por otra parte, el número de lenguajes con respecto al alfabeto  $\Sigma$  es incontable, ya que el conjunto infinito  $\Sigma^*$  tiene un número incontable de subconjuntos. Por lo tanto, hay más lenguajes que autómatas finitos deterministas. Por consiguiente, ya que cada autómata finito determinista acepta sólo un lenguaje, deben existir lenguajes que no son aceptados por una máquina de este tipo.



El teorema 1.1 nos dice que existen conjuntos de cadenas que no pueden ser identificados por los autómatas finitos deterministas, y por lo tanto nuestras técnicas de análisis léxico no pueden reconocerlos. Sin embargo, el hecho de que los autómatas finitos deterministas no puedan resolver todos estos problemas de reconocimientos no es más que la punta del iceberg. Más adelante veremos que cualquier problema algorítmico tiene limitaciones similares.

### Un lenguaje no regular

Mostremos ahora un ejemplo específico de un lenguaje que no es regular. Este tipo de ejemplos indicará cómo pueden mejorarse las técnicas de análisis léxico de la sección 1.1 para permitirnos una mayor gama de estructuras de cadenas.

Teniendo en cuenta este objetivo, consideremos el problema de analizar expresiones matemáticas donde se usan paréntesis para hacer más claro el orden de ejecución o para alterar la precedencia normal de los operadores, como sucede en las expresiones  $(a + b) + c$  y  $a + (b + c)$ . En estas expresiones debe existir el mismo número de paréntesis izquierdos y derechos. Además, al recorrer una expresión de izquierda a derecha, el número de paréntesis

derechos detectados no debe exceder el número de paréntesis izquierdos encontrados hasta ese punto. Con estas observaciones, nuestra intuición nos dice que el análisis de estas expresiones aritméticas requiere la habilidad para recordar cuántos paréntesis izquierdos se han encontrado sin que exista un paréntesis derecho correspondiente. Puesto que un autómata finito determinista no tiene manera de almacenar este recuento para una referencia posterior, podemos adivinar que el análisis de expresiones aritméticas que contienen paréntesis es una tarea que rebasa las capacidades de los autómatas finitos. Sin embargo, para demostrarlo requerimos ciertos antecedentes.

En primer lugar, presentamos la notación  $w^n$ , donde  $w$  es una cadena de  $\Sigma^*$  y  $n$  es un entero no negativo. Se trata de una forma abreviada para representar una cadena de  $n$  copias del patrón  $w$ . Así, si  $\Sigma = \{x, y\}$ , entonces  $x^4 = xxxx$ ,  $x^2y^2 = xxyy$ ,  $(xy)^3 = xyxyxy$  y  $y^0 = \lambda$  (la cadena vacía).

Ahora necesitamos el siguiente teorema.

### TEOREMA 1.2

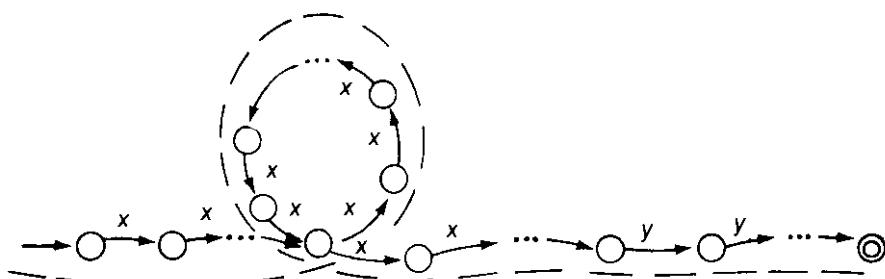
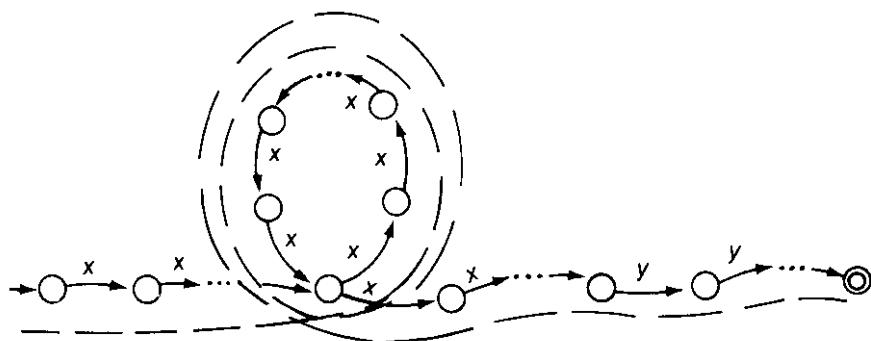
Si un lenguaje regular contiene cadenas de la forma  $x^n y^n$  para enteros arbitrariamente grandes,  $n$ , entonces debe contener cadenas de la forma  $x^m y^n$ , donde  $m$  y  $n$  no son iguales.

#### DEMOSTRACIÓN

Suponga que  $M$  es un autómata finito determinista tal que  $L(M)$  contiene  $x^n y^n$  para una  $n$  arbitrariamente grande. Entonces, debe existir un entero positivo  $k$  mayor que el número de estados en  $M$  y tal que  $x^k y^k$  se encuentre en  $L(M)$ . Puesto que existen más símbolos en  $x^k$  que estados en  $M$ , el proceso de aceptación de  $x^k y^k$  dará como resultado que se recorra más de una vez alguno de los estados de  $M$  antes de llegar a alguna de las  $y$  de la cadena. Es decir, durante la lectura de algunas de las  $x$  se recorrerá una ruta circular del diagrama de transiciones de la máquina. Si  $j$  es el número de  $x$  leídas al recorrer esta ruta, entonces la máquina puede aceptar la cadena  $x^{k+j} y^k$  recorriendo esta ruta una vez más (véase Fig 1.16). Por lo tanto, existe un entero positivo  $m$  (específicamente  $k + j$ ) que no es igual a  $k$ , tal que  $x^m y^k$  se encuentra en  $L(M)$ .



Una consecuencia inmediata del teorema 1.2 es que el lenguaje  $\{x^n y^n : n \in \mathbb{N}\}$  no es regular. Una consecuencia un poco más sutil es que los autómatas finitos deterministas carecen de poder suficiente para analizar expresiones aritméticas que contienen paréntesis. Si un autómata finito determinista aceptara tales expresiones, entonces tendría que aceptar expresiones de la forma  $(^n)^n$  para enteros  $n$  arbitrariamente grandes. Sin embargo, el teorema 1.2 nos dice que un autómata de este tipo también

a. Ruta que se recorre al aceptar  $x^k y^k$ b. Ruta que se recorre al aceptar  $x^{k+i} y^k$ 

**Figura 1.16** Porción de un diagrama de transiciones que acepta tanto  $x^k y^k$  como  $x^{k+i} y^k$

debe aceptar expresiones en donde el número de paréntesis izquierdos no sea igual al número de paréntesis derechos. Así, tal máquina aceptaría expresiones tanto incorrectas como correctas.

Para concluir, debemos notar que los autómatas finitos deterministas, aunque tengan un poder limitado, no son inútiles. Si usted repasa la sintaxis de su lenguaje de programación de alto nivel preferido, quizás encuentre que un autómata finito no puede reconocer todos los detalles de la estructura de su programa ya que, entre otras cosas, es probable que el lenguaje permita emplear paréntesis en expresiones aritméticas. Por otra parte, también

encontrará que los autómatas finitos pueden procesar la estructura de construcciones como palabras reservadas, nombres de variables y símbolos de operaciones. La simplicidad de este nivel del lenguaje no es fortuita: permite construir el analizador léxico del compilador utilizando técnicas sencillas como las presentadas en este capítulo.

## Ejercicios

1. ¿Cómo puede alterarse el autómata finito determinista  $M = (S, \Sigma, \delta, s_0, F)$  para obtener una máquina que acepte el lenguaje  $\Sigma^* - L(M)$ ? (Entonces, el complemento de un lenguaje regular con respecto a  $\Sigma^*$  es regular.)
2. Elabore una lista de los autómatas finitos deterministas basados en el alfabeto  $\{x, y\}$  que tengan un estado y luego los que tengan dos estados.
3. Muestre que si un autómata finito determinista es capaz de aceptar un número de cadenas infinito, entonces debe aceptar una cadena que consista en la concatenación de tres segmentos tales que cualquier repetición del segmento central (que es no vacío) dé como resultado otra cadena aceptable. (Esto se conoce como el **lema de bombeo pumping lemma**, pues indica que pueden generarse otras cadenas aceptables “bombeando” o “ampliando” una cadena aceptable.) Sugerencia: Considere de nuevo la demostración del teorema 1.2.
4. Muestre que no existe un autómata finito determinista  $M$  tal que  $L(M) = \{x^n y^n z^n : n \in \mathbb{N}\}$ .
5. Muestre que puede utilizarse un autómata finito determinista para reconocer una cadena de paréntesis anidados y equilibrados si se asegura que la profundidad del anidamiento no excederá un nivel determinado.

## 1.4 AUTÓMATAS FINITOS NO DETERMINISTAS

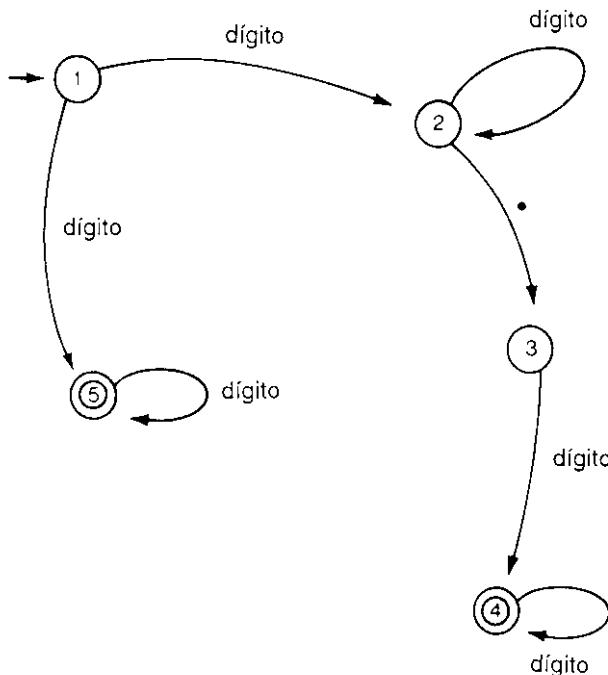
Una vez que nos hemos percatado de las limitantes de los autómatas finitos deterministas, examinar ahora algunas modificaciones que pueden incrementar su potencial. Aquí y en los capítulos subsecuentes, nuestro enfoque será considerar una reducción de las restricciones que hemos impuesto sobre las máquinas. Después de todo, la intuición nos indica que una máquina más flexible debería ser capaz de desempeñar tareas más variadas y, por lo tanto, aceptar un lenguaje que no podrían aceptar las versiones más restringidas. Sin embargo, estamos a punto de ver que nuestra intuición no siempre es correcta.

En lo que llevamos de nuestro estudio, hemos insistido en que los diagramas de transiciones de los autómatas que consideramos deben ser

deterministas. A partir de un estado sólo puede existir un solo arco rotulado con un símbolo determinado. La razón de esto fue que pensamos desarrollar programas a partir de estas máquinas, y sería de poca utilidad un programa que se comportara de manera no determinista. Ahora que nos enfrentamos al problema de ampliar el poder de nuestras técnicas, parece razonable volver a considerar esta restricción; si el no determinismo resulta benéfico, podríamos diseñar un programa que manejara varias opciones aplicando técnicas de recorrido con retrocesos.

Tomando esto en cuenta, consideremos otro tipo de máquinas conceptuales, conocidas como **autómatas finitos no deterministas**. Esta máquina se parece mucho a un autómata finito determinista pues también analiza cadenas construidas a partir de un alfabeto finito y sólo puede tener un número finito de estados, algunos de los cuales son de aceptación y uno es el estado inicial. Sin embargo, a diferencia de los autómatas finitos deterministas, la transición que se ejecuta en una etapa dada de un autómata finito no determinista puede ser incierta. Esto quizás se debiera a que es posible aplicar más de una transición, o a que ninguna es aplicable, como sucede con una máquina que no está completamente definida.

Como ejemplo, considere el diagrama de transiciones de la figura 1.17, el cual es intrínsecamente diferente de los diagramas que dibujamos antes.



**Figura 1.17** Diagrama de transiciones que acepta cadenas que representan enteros o cadenas que representan números reales en notación decimal

Para ser más precisos, aunque una cadena comience con un dígito, esto no nos indica con exactitud cuál es el arco que debe seguirse a partir del estado inicial, pues existen dos posibilidades: una que conduce a la sección del diagrama que describe la estructura de un número en notación decimal y otra que lleva a la descripción de un entero. Por lo tanto, un autómata finito programado con este diagrama sería no determinista.

Recuerde que en el caso de un autómata finito determinista, decimos que una cadena se acepta si al analizarla la máquina queda en un estado de aceptación. Sin embargo, en el caso de un autómata finito no determinista, el intento de análisis de una cadena puede llevar un error simplemente porque se tomaron las decisiones incorrectas en los puntos donde existían varias opciones (si tomamos una decisión incorrecta en el primer paso del análisis de la cadena 352 utilizando la figura 1.17, no llegaríamos a un estado de aceptación, a pesar de que la cadena en cuestión es compatible con otras rutas del diagrama). Por esto, decimos que un autómata finito no determinista acepta una cadena si es *possible* que su análisis deje a la máquina en un estado de aceptación.

Como sucede en el caso de los autómatas finitos deterministas, el conjunto de todas las cadenas aceptadas por un autómata finito no determinista  $M$  es un lenguaje que representamos con  $L(M)$ , y nos referimos a él como el lenguaje aceptado por  $M$ .

En resumen, definimos de manera formal un autómata finito no determinista como sigue:

Un autómata finito no determinista consiste en una quíntupla  $(S, \Sigma, \rho, i, F)$ , donde

- $S$  es un conjunto finito de estados.
- $\Sigma$  es el alfabeto de la máquina.
- $\rho$  es un subconjunto de  $S \times \Sigma \times S$ .
- $i$  (un elemento de  $S$ ) es el estado inicial
- $F$  (un subconjunto de  $S$ ) es la colección de estados de aceptación.

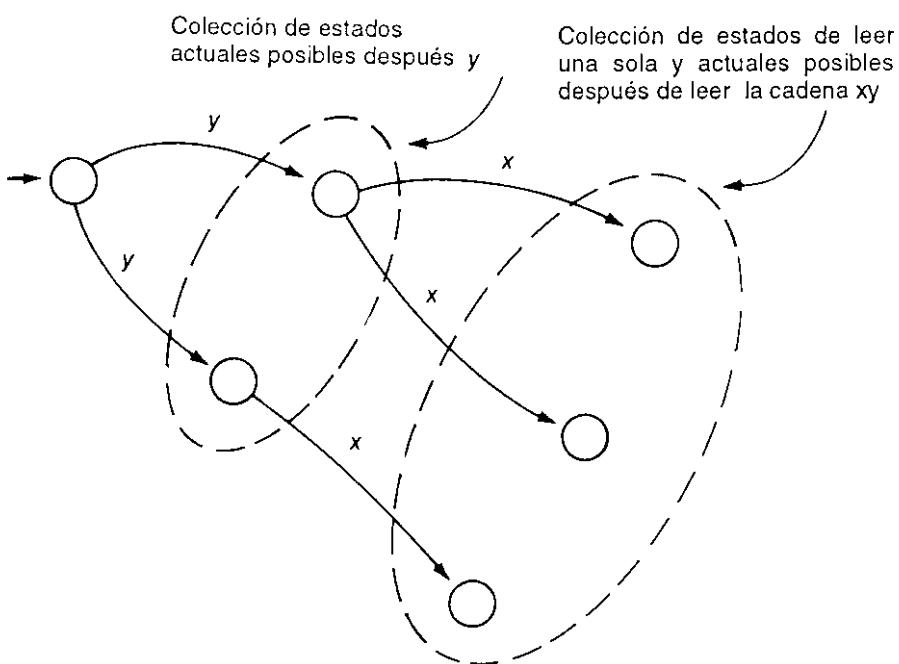
En esta definición, el subconjunto  $\rho$  representa las posibles transiciones de la máquina. Es decir, la tupla  $(p, x, q)$  está en  $\rho$  si y sólo si el autómata puede pasar del estado  $p$  al estado  $q$  al leer el símbolo  $x$  de la cadena de entrada. Así, un autómata finito no determinista  $(S, \Sigma, \rho, i, F)$ , acepta la cadena no vacía  $x_1 x_2 \cdots x_n \in \Sigma^*$  si y sólo si existe una secuencia de estados  $s_0, s_1, \dots, s_n$  tal que  $s_0 = i$ ,  $s_n \in F$  y para cada entero  $j$  de 1 a  $n$ ,  $(s_{j-1}, x_j, s_j)$  está en  $\rho$ .

Observe también que tanto  $(p, x, q_1)$  como  $(p, x, q_2)$  pueden estar en  $\rho$  incluso cuando  $q_1$  y  $q_2$  no son iguales. Si esto llega a suceder, se presentará una opción si la máquina se encuentra en el estado  $p$ : al leer el símbolo  $x$  de la cadena de entrada: puede pasar a  $q_1$  o a  $q_2$ . A su vez, la traducción directa de un autómata finito no determinista a formato de programa puede producir un analizador léxico no determinista. Para corregir esta situación, tendríamos que modificar el analizador a fin de que no rechace una cadena de entrada hasta que haya

probado todas las rutas posibles del diagrama de transiciones y encontrado que todas fallan. Este sistema de retrocesos haría más complejo al analizador y sólo valdría la pena si obtuviéramos más poder de procesamiento. Así, el teorema 1.3 explica por qué rehusamos utilizar un autómata finito no determinista como base para un analizador léxico.

Para motivar este teorema, imaginemos la estrategia que seguiríamos si utilizáramos un diagrama de transiciones no determinista para analizar una cadena. En esta situación, podríamos evitar ajustarnos a una sola posibilidad y tratar de seguir todas las opciones en paralelo; es decir, atravesaríamos al mismo tiempo más de una ruta a través del diagrama de transiciones. Al obtener cada símbolo de la entrada, cada una de estas rutas avanzaría de manera independiente. En este contexto, aceptaríamos una cadena si alguna de las rutas terminara en un estado de aceptación al llegar al final de la entrada. Las otras rutas representarían únicamente opciones que, de haberlas seguido, habrían llevado al fracaso y a la necesidad de retroceder para seguir otras opciones.

Cuando seguimos este enfoque de procesamiento en paralelo, nuestro estado en un momento dado ya no se hallará determinado por un solo



**Figura 1.18** Análisis de una cadena con un autómata finito no determinista, siguiendo las opciones en paralelo

estado del diagrama de transiciones, sino por la colección de los estados actuales de todas las rutas posibles que se consideran (véase Fig. 1.18). Además, si  $K$  es esta colección de estados actuales, entonces, al leer un símbolo  $x$  del flujo de entrada obtendríamos un nuevo estado actual, representado por la colección de los estados a los que es posible llegar desde un estado  $K$  siguiendo un arco con etiqueta  $x$ .

Observe que este enfoque en paralelo, que pasa de colecciones de estados a colecciones de estados, elimina la necesidad de tomar decisiones cuando se presentan varias opciones: basta con seguir en paralelo todas las opciones. Así, parecería que con este enfoque se podría superar el no determinismo en un autómata finito. Esta observación es la que da origen a nuestra demostración del siguiente teorema.

### TEOREMA 1.3

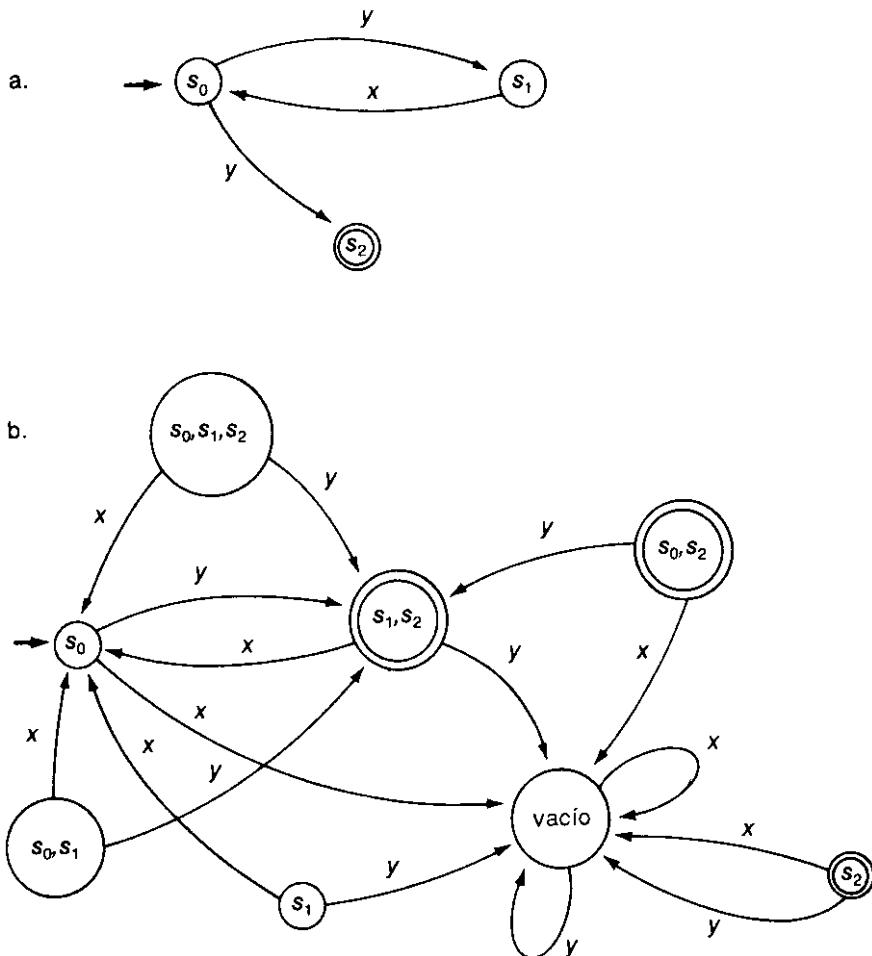
Para cada autómata finito no determinista, existe un autómata finito determinista que acepta exactamente el mismo lenguaje.

#### DEMOSTRACIÓN

Suponga que  $M$  es el autómata finito no determinista definido por la quíntupla  $(S, \Sigma, p, t, F)$ . Nuestra tarea es demostrar la existencia de un autómata finito determinista que acepta exactamente las mismas cadenas que  $M$ . Para esto, definimos otro autómata,  $M'$ , con el quíntuplo  $(S', \Sigma, \delta, t', F')$ , donde  $S' = \mathcal{P}(S)$ ,  $t' = \{t\}$ ,  $F'$  es la colección de subconjuntos de  $S$  que contienen por lo menos un estado de  $F$ , y  $\delta$  es la función de  $S' \times \Sigma$  a  $S'$  tal que para cada  $x$  en  $\Sigma$  y  $s'$  en  $S'$ ,  $\delta(s', x)$  es el conjunto de todo  $s$  en  $S$  tal que  $(u, x, s)$  está en  $p$  para algún  $u$  en  $s'$  (es decir,  $\delta(s', x)$  es el conjunto de todos los estados de  $S$  a los que es posible llegar desde un estado en  $s'$  siguiendo un arco con etiqueta  $x$ ). Observe que como  $\delta$  es una función,  $M'$  es un autómata finito determinista (para obtener un ejemplo de esta construcción, véase Fig. 1.19).

Lo que falta es mostrar que  $M$  y  $M'$  aceptan exactamente las mismas cadenas. Para esto, aplicamos un argumento de inducción que muestre que para cada  $n \in \mathbb{N}$  es cierto el siguiente enunciado.

*Para cada ruta en  $M$  del estado  $t$  al estado  $s_n$ , que recorre arcos rotulados  $w_1, w_2, \dots, w_n$ , existe una ruta en  $M'$  del estado  $t'$  al estado  $s'_n$ , que recorre los arcos rotulados  $w_1, w_2, \dots, w_n$ , de modo que  $s_n \in s'_n$ ; y a la inversa, para cada ruta en  $M'$  de  $t'$  a  $s'_n$  recorriendo arcos rotulados  $w_1, w_2, \dots, w_n$  y cada  $s_n \in s'_n$ , existe una ruta en  $M$  de  $t$  a  $s_n$  que recorre arcos rotulados  $w_1, w_2, \dots, w_n$ .*



**Figura 1.19** a. Diagrama de transiciones para un autómata finito no determinista  
 b. Diagrama de transición para el autómata finito determinista equivalente, construido de acuerdo con la demostración del teorema 1.3

A partir de esto, debe desprenderse que para cualquier ruta en  $M$  que vaya de  $i$  a un estado de aceptación y recorra arcos rotulados  $w_1, w_2, \dots, w_n$ , debe existir una ruta en  $M'$ , de  $i'$  a un estado de aceptación, que siga los arcos rotulados  $w_1, w_2, \dots, w_n$  y viceversa. Por lo tanto,  $M$  y  $M'$  deben aceptar el mismo lenguaje.

Comenzamos nuestra inducción considerando el caso  $n = 0$ . Aquí  $s_n$  debe ser  $i$  y  $s'_n$  debe ser  $i' = \{i\}$ , por lo que, trivialmente, el enunciado es verdadero.

A continuación, suponemos que el enunciado es verdadero para un  $n \in \mathbb{N}$  y consideramos una ruta en  $M$ , de  $i$  a algún  $s_{n+1}$ , que recorre los arcos rotulados  $w_1, w_2, \dots, w_{n+1}$ . Sea  $s_n$  el penúltimo estado de  $M$  por esta ruta. Así,  $(s_n, w_{n+1}, s_{n+1}) \in \rho$ . Por nuestra hipótesis de inducción, existe una ruta en  $M'$ , de  $i'$  a algún  $s'_n$ , que recorre arcos rotulados  $w_1, w_2, \dots, w_{n+1}$  de modo que  $s_n \in s'_n$ . Puesto que  $(s_n, w_{n+1}, s_{n+1}) \in \rho$ , existe un arco en  $M'$  rotulado  $w_{n+1}$  a un estado que contiene a  $s_{n+1}$ . Sea  $s'_{n+1}$  tal estado. Entonces existe una ruta en  $M'$ , de  $i'$  a  $s'_{n+1}$ , que recorre los arcos rotulados  $w_1, w_2, \dots, w_{n+1}$ , de modo que  $s_{n+1} \in s'_{n+1}$ .

A la inversa, considere una ruta en  $M'$ , de  $i'$  a algún  $s'_{n+1}$ , que recorre los arcos rotulados  $w_1, w_2, \dots, w_{n+1}$ , y sea  $s'_n$  el penúltimo estado de esta ruta. Luego, por inducción, para cada  $s_n \in s'_n$  debe existir una ruta en  $M$ , de  $i$  a  $s_n$ , que recorre los arcos rotulados  $w_1, w_2, \dots, w_n$ . Pero  $\delta(s'_n, w_{n+1}) = s'_{n+1}$ , por lo que  $s'_{n+1}$  es el conjunto de estados  $s$  en  $M$  tal que  $(s_n, w_{n+1}, s) \in \rho$ . Por lo tanto, para cada  $s$  en  $s'_{n+1}$ , existe una ruta en  $M$ , de  $i$  a  $s$ , que recorre arcos rotulados  $w_1, w_2, \dots, w_{n+1}$ , como se requiere.



El teorema 1.3 representa tanto buenas como malas noticias. Las buenas noticias son que no se necesitan las ramificaciones y los retrocesos que se requerirían para convertir un autómata finito no determinista en un segmento de programa, ya que siempre existirá un autómata finito determinista que realice la misma tarea. Las malas noticias son que, al permitir el no determinismo, no podemos mejorar el poder de nuestras técnicas basadas en autómatas finitos deterministas.

Para concluir esta sección, presentamos un teorema que resume de manera explícita la relación entre los lenguajes aceptados por los autómatas finitos deterministas y no deterministas.

#### TEOREMA 1.4

Para cualquier alfabeto  $\Sigma$ ,  $\{L(M) : M \text{ es un autómata finito determinista con alfabeto } \Sigma\} = \{L(M) : M \text{ es un autómata finito no determinista con alfabeto } \Sigma\}$ .

#### DEMOSTRACION

El hecho de que los lenguajes aceptados por los autómatas finitos no deterministas sean también aceptados por los autómatas finitos deterministas no es más que una repetición del teorema 1.3. A la

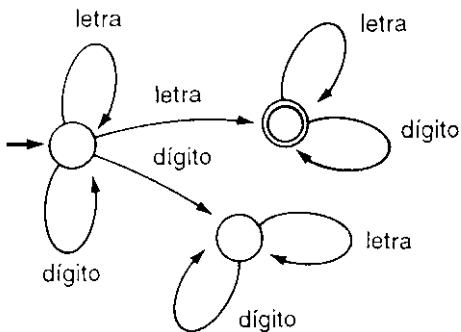
inversa, si  $M = (S, \Sigma, \delta, i, F)$  es un autómata finito determinista, entonces el autómata finito no determinista  $(S, \Sigma, \rho, i, F)$ , donde  $(p, x, q) \in \rho$  si y sólo si  $\delta(p, x) = q$ , acepta el mismo lenguaje.

■

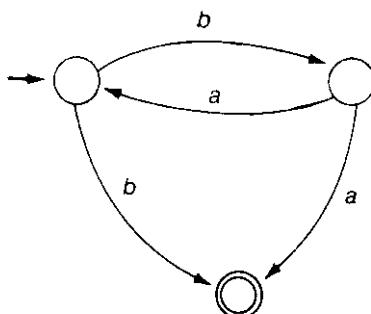
Por el teorema 1.4, muchos autores no distinguen entre autómatas finitos deterministas y los no deterministas cuando analizan la aceptación de lenguajes. Por el contrario, únicamente hacen referencia a autómatas finitos; con frecuencia haremos lo mismo.

## Ejercicios

- Identifique los puntos donde existe no determinismo en el siguiente diagrama de transiciones.



- Utilizando métodos intuitivos, modifique el diagrama de transiciones de la figura 1.17 para que acepte las mismas cadenas que antes, pero ahora sin ramificaciones no deterministas.
- Muestre que cada autómata finito determinista  $(S, \Sigma, \delta, i, F)$  es un autómata finito no determinista, demostrando que la función de transición  $\delta$  puede representarse como un subconjunto de  $S \times \Sigma \times S$ .
- Empleando las técnicas presentadas en la demostración del teorema 1.3, dibuje un diagrama de transiciones para un autómata finito determinista que acepte las mismas cadenas que el autómata finito no determinista representado en el siguiente diagrama de transiciones.



## 1.5 GRAMÁTICAS REGULARES

Nuestro objetivo para lo que resta del capítulo es aprender más acerca de los lenguajes regulares. En esta sección analizaremos su estructura gramatical y en la siguiente su composición algebraica. Comenzaremos con el concepto general de gramática.

Resulta útil motivar nuestro análisis con ejemplos de los lenguajes naturales, donde se emplean reglas gramaticales para distinguir aquellas cadenas de símbolos que constituyen enunciados aceptables de las que no. Por ejemplo, una frase sencilla en español puede describirse como una estructura con un sujeto seguido por un predicado. El sujeto podría ser un nombre sencillo, mientras que el predicado podría ser un verbo intransitivo o quizás un verbo transitivo seguido por un objeto. Esta descomposición descendente podría continuar hasta llegar a un nivel de detalle donde describiríamos los símbolos literales que aparecen en una frase en español. Por ejemplo, podríamos describir un verbo transitivo como "golpear" o la palabra "quiere". En la figura 1.20 se muestra esta descomposición.

Hay que señalar varios puntos con respecto a la figura 1.20. En primer lugar, usted observará que algunos de los términos se encuentran encerrados entre corchetes agudos, mientras que otros no lo están. Los términos que no aparecen entre corchetes se llaman **terminales**, o símbolos que pueden aparecer en una frase. Los términos que se hallan encerrados entre corchetes se denominan **no terminales**. Uno de estos no terminales (en la Fig. 1.20 es `<frase>`) se considera como **símbolo de inicio**. Cada una de las líneas de la figura 1.20 se llama **regla de reescritura** consiste en una parte izquierda y una derecha, conectadas por una flecha. La parte derecha de estas reglas representa una descripción más detallada de la parte izquierda. Por ejemplo, la primera regla de reescritura de la figura 1.20 indica que una `<frase>` tiene la estructura de un `<sujeto>` seguido por un `<predicado>` seguido por un `<punto>`. Más adelante en la misma figura, aparece una descripción más detallada de estos no terminales. Así, la figura 1.20 describe de manera jerárquica la estructura de una frase, comenzando con el símbolo de inicio `<frase>`.

```

<frase> → <sujeto> <predicado> <punto>
<sujeto> → <sustantivo>
<sustantivo> → María
<sustantivo> → Juan
<predicado> → <verbo intransitivo>
<predicado> → <verbo transitivo> <objeto>
<verbo intransitivo> → patinar
<verbo transitivo> → golpear
<verbo transitivo> → <quiere>
<objeto> → a <sustantivo>
<punto> → .

```

**Figura 1.20** Gramática que describe un pequeño subconjunto del lenguaje español

Esta colección de no terminales y terminales, junto con un símbolo de inicio y un conjunto finito de reglas de reescritura, se denomina gramática o, más precisamente, **gramática estructurada por frases**, ya se está basa en la composición de cadenas en términos de "frases", donde cada frase está representada por un no terminal. De manera más formal, una gramática se define como una cuádrupla ( $V, T, S, R$ ), donde  $V$  es un conjunto finito de no terminales,  $T$  es un conjunto finito de terminales,  $S$  (un elemento de  $N$ ) es el símbolo inicial y  $R$  es un conjunto finito de reglas de reescritura. En general, los lados derecho e izquierdo de las reglas de reescritura de una gramática pueden ser cualquier combinación de terminales y no terminales, siempre y cuando el lado izquierdo contenga por lo menos un no terminal. Además, el lado derecho de algunas reglas de reescritura puede consistir en una cadena vacía (en este caso, la regla significa que el patrón representado por el lado derecho de la regla puede ser la cadena vacía). Esta regla se llama **regla  $\lambda$** .

A fin de evitar confusiones, se emplean corchetes para distinguir los terminales de los no terminales. Por ejemplo, en una gramática que describiera la estructura de las frases en español, el término *frase* podría aparecer como un no terminal, representando el símbolo de inicio, y como un terminal, representando una palabra en una frase. En este caso, los corchetes nos permitirían diferenciar usos duales del término. En nuestro estudio, sin embargo, distinguiremos entre terminales y no terminales con una notación menos engorrosa. Salvo que se especifique lo contrario, representaremos a

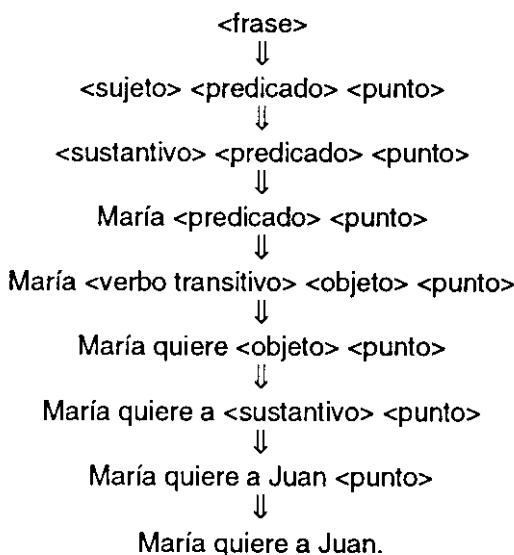
los no terminales con letras mayúsculas y a los terminales con letras minúsculas. De esta manera, una regla de la forma  $S \rightarrow xN$  significará el no terminal  $S$  se puede refinar como el terminal  $x$  seguido por el no terminal  $N$ .

Se dice que una gramática genera una cadena de terminales si, al comenzar con el símbolo de inicio, se puede producir esa cadena sustituyendo sucesivamente los patrones que se encuentran en el lado izquierdo de las reglas de reescritura de la gramática con las expresiones correspondientes de la derecha, hasta que sólo queden terminales. La secuencia de pasos de este proceso se le conoce como derivación de la cadena. Por ejemplo, la cadena (la frase).

María quiere a Juan.

puede generarse a partir de la gramática de la figura 1.20 usando la derivación de la figura 1.21. Como un ejemplo, más la figura 1.22 muestra otra gramática, con el símbolo inicial  $S$ , y una derivación que muestra cómo puede generarse la cadena  $xyz$ .

Si los terminales de una gramática  $G$  son símbolos del alfabeto  $\Sigma$ , decimos que  $G$  es una gramática del alfabeto  $\Sigma$ . En este caso, las cadenas generadas por  $G$  son en realidad cadenas de  $\Sigma^*$ . Por lo tanto, una gramática  $G$  de un alfabeto  $\Sigma$  especifica un lenguaje de  $\Sigma$  que consiste en las cadenas generadas por  $G$ ; a este lenguaje se le representa con  $L(G)$ . Si lo desea, puede confirmar que el lenguaje generado por la gramática de la figura 1.22 es  $\{x^m y^m z^m : m, n \in \mathbb{N}\}$ .




---

**Figura 1.21** Derivación de la frase "María quiere a Juan."

$$\begin{aligned} S &\rightarrow XSZ \\ S &\rightarrow Y \\ Y &\rightarrow yY \\ Y &\rightarrow \lambda \\ X &\rightarrow x \\ Z &\rightarrow z \end{aligned}$$

$$S \Rightarrow XSZ \Rightarrow xSZ \Rightarrow xyYZ \Rightarrow xyZ \Rightarrow xyz$$

**Figura 1.22** Gramática y una derivación asociada de la cadena xyz

$$\begin{array}{llll} S \rightarrow 0T & T \rightarrow 0T & U \rightarrow 0V & V \rightarrow 0V \\ S \rightarrow 1T & T \rightarrow 1T & U \rightarrow 1V & V \rightarrow 1V \\ S \rightarrow 2T & T \rightarrow 2T & U \rightarrow 2V & V \rightarrow 2V \\ S \rightarrow 3T & T \rightarrow 3T & U \rightarrow 3V & V \rightarrow 3V \\ S \rightarrow 4T & T \rightarrow 4T & U \rightarrow 4V & V \rightarrow 4V \\ S \rightarrow 5T & T \rightarrow 5T & U \rightarrow 5V & V \rightarrow 5V \\ S \rightarrow 6T & T \rightarrow 6T & U \rightarrow 6V & V \rightarrow 6V \\ S \rightarrow 7T & T \rightarrow 7T & U \rightarrow 7V & V \rightarrow 7V \\ S \rightarrow 8T & T \rightarrow 8T & U \rightarrow 8V & V \rightarrow 8V \\ S \rightarrow 9T & T \rightarrow 9T & U \rightarrow 9V & V \rightarrow 9V \\ & T \rightarrow .U & & V \rightarrow \lambda \end{array}$$

**Figura 1.23** Gramática regular que genera cadenas que representan números racionales en notación decimal

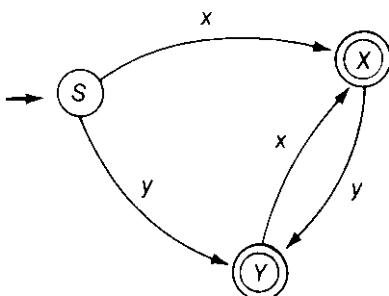
Ahora definimos una **gramática regular** como aquella gramática cuyas reglas de reescritura se adhieren a las siguientes restricciones. El lado izquierdo de cualquier regla de reescritura de una gramática regular debe consistir en un solo no terminal, mientras que el lado derecho debe ser un terminal seguido por un no terminal, o un solo terminal, o la cadena vacía. Así, las reglas de reescritura de la forma

$$\begin{aligned} Z &\rightarrow yX \\ Z &\rightarrow x \\ W &\rightarrow l \end{aligned}$$

$$\begin{aligned} S &\rightarrow xX \\ X &\rightarrow yY \\ Y &\rightarrow xX \\ Y &\rightarrow \lambda \end{aligned}$$

**Figura 1.24** Gramática regular que genera cadenas consistentes en una o más copias del patrón  $xy$

$$\begin{aligned} S &\rightarrow xX \\ S &\rightarrow yY \\ X &\rightarrow yY \\ X &\rightarrow \lambda \\ Y &\rightarrow xX \\ Y &\rightarrow \lambda \end{aligned}$$



**Figura 1.25** Gramática regular  $G$  y diagrama de transiciones para un autómata finito  $M$  tal que  $L(G) = L(M)$

estarían permitidas en una gramática regular, mientras que las reglas de la forma

$$\begin{aligned} yW &\rightarrow X \\ X &\rightarrow xZy \\ YX &\rightarrow WvZ \end{aligned}$$

no lo estarían.

La figura 1.23 presenta una gramática regular que genera cadenas que representan números racionales en notación decimal, y la figura 1.24 presenta una gramática regular que genera cadenas consistentes en una o más copias del patrón  $xy$ . En los dos ejemplos, el símbolo inicial es  $S$ . Usted observará que ninguna de estas gramáticas utiliza una regla cuyo lado derecho consista en un solo terminal. Aunque lo permite la definición de una gramática regular, en realidad no se requieren reglas de este tipo. De hecho, en una gramática regular cualquier regla de la forma

$$N \rightarrow x$$

podría reemplazarse con el par de reglas

$$\begin{aligned} N &\rightarrow xX \\ X &\rightarrow \lambda \end{aligned}$$

donde  $X$  es un no terminal que no aparece en ningún otro lugar de la gramática, sin alterar el conjunto de cadenas que podría generar la gramática.

La importancia de las gramáticas regulares reside en que los lenguajes generados por ellas son exactamente aquellos que reconocen los autómatas finitos, como lo muestra el teorema 1.5

### TEOREMA 1.5

Para cada alfabeto  $\Sigma$   $\{L(G): G$  es una gramática regular de  $\Sigma\} = \{L(M): M$  es un autómata finito de  $\Sigma\}$ .

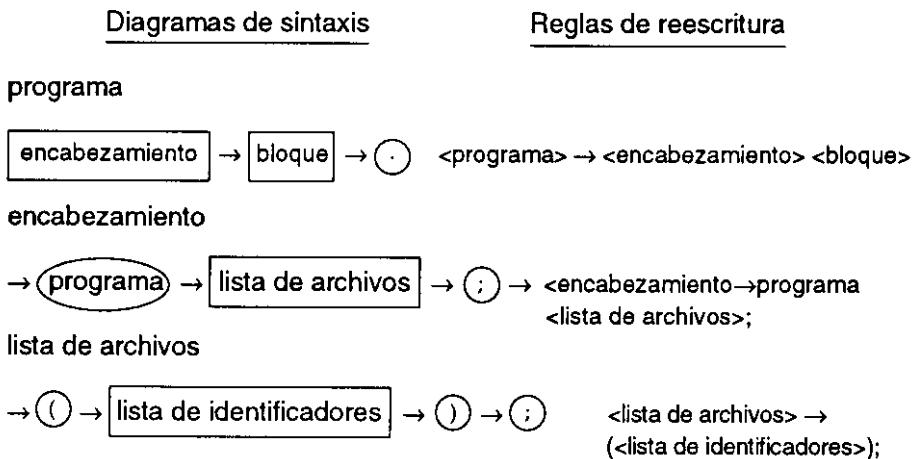
#### DEMOSTRACIÓN

Si  $G$  es una gramática regular de  $\Sigma$ , podemos convertirla en una gramática regular  $G'$  que genera el mismo lenguaje pero que no contiene reglas de reescritura cuyo lado derecho consiste en un solo terminal (véanse los comentarios que preceden al Teorema 1.5). Entonces podemos definir  $M$  como el autómata finito no determinista  $(S, \Sigma, \rho, \iota, F)$ , donde  $S$  es la colección de no terminales de  $G'$ ,  $\iota$  es el símbolo inicial de  $G'$ ,  $F$  es la colección de no terminales de  $G'$  que aparecen en el lado izquierdo de alguna regla  $\lambda$ , y  $\rho$  consiste en la tripleta  $(P, x, Q)$  para el cual  $G'$  contiene una regla de reescritura de la forma  $P \rightarrow xQ$ . A la inversa, si  $M$  es el autómata finito no determinista  $(S, \Sigma, \rho, \iota, F)$ , entonces podemos definir  $G'$  como la gramática regular de  $\Sigma$  para la cual los no terminales son los estados de  $S$ , el símbolo inicial es  $\iota$  y las reglas de reescritura son de la forma  $P \rightarrow xQ$  si  $(P, x, Q) \in \rho$  y  $Q \rightarrow \lambda$  si  $Q \in F$ .

En ambos casos,  $L(M) = L(G')$  ya que la derivación de una cadena de la gramática  $G'$  corresponde directamente a una ruta en el diagrama de transiciones de  $M$  que conduce del estado inicial a un estado de aceptación y viceversa. Como ejemplo, véase la figura 1.25.



La importancia del enfoque gramatical de la especificación de lenguajes se hace aparente en los dos capítulos siguientes, donde veremos que el empleo de gramáticas lleva a un esquema de clasificación para diversos tipos de lenguajes. Por el momento, la composición de gramáticas se puede comparar con la de los diagramas de sintaxis que se utilizan con frecuencia en la actualidad al describir la sintaxis de lenguajes de programación como Pascal, Modula-2 y Ada. Encontrará que estos diagramas de sintaxis son, en esencia, otra forma de representación de las reglas de reescritura de una gramática (véase Fig. 1.26).



**Figura 1.26** Algunos diagramas de sintaxis que describen partes del lenguaje de programación Pascal, comparados con sus formas gramaticales equivalentes

## Ejercicios

1. Elabore una lista con todas las frases generadas por la gramática de la figura 1.20.
2. Muestre que es posible modificar la gramática que se presenta a continuación (con símbolo inicial  $S$ ) para formar una gramática regular, sin cambiar el lenguaje que genera. ¿Cuál de los ejercicios de la sección anterior es en esencia el mismo problema, aunque en un contexto diferente?

$$\begin{aligned} S &\rightarrow yX \\ X &\rightarrow xxX \\ X &\rightarrow yY \\ Y &\rightarrow \lambda \end{aligned}$$

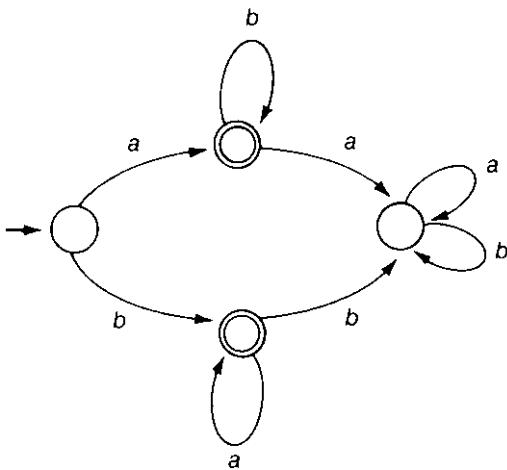
3. Convierta la siguiente gramática regular en otra gramática regular que no contenga reglas de reescritura cuyos lados derechos consistan en un solo terminal, pero que a la vez genere el mismo lenguaje. Describa con sus propias palabras el lenguaje generado.

$$\begin{aligned} S &\rightarrow xS \\ S &\rightarrow y \\ S &\rightarrow z \end{aligned}$$

4. Dibuje un diagrama de transiciones para un autómata finito que acepte el lenguaje generado por la gramática regular (con símbolo inicial S) que se presenta a continuación. Describa el lenguaje con sus propias palabras.

$$\begin{aligned}S &\rightarrow \lambda \\S &\rightarrow xX \\S &\rightarrow yY \\Y &\rightarrow yY \\Y &\rightarrow \lambda \\X &\rightarrow xX \\X &\rightarrow \lambda\end{aligned}$$

5. Presente una gramática regular que genere el lenguaje aceptado por el autómata finito cuyo diagrama de transiciones se presenta a continuación.



## 1.6 EXPRESIONES REGULARES

Hemos definido los lenguajes regulares como aquellos que son reconocidos por autómatas finitos, y hemos mostrado que son también aquellos lenguajes generados por gramáticas regulares. En esta sección desarrollaremos otra caracterización de los lenguajes regulares, la cual proporciona mayores detalles acerca de la composición de dichos lenguajes.

Aquí nuestro enfoque será mostrar cómo se pueden construir los lenguajes regulares a partir de pequeños bloques de construcción. Comenzamos por considerar los lenguajes más sencillos que pueden formarse con el

alfabeto  $\Sigma$ . Es decir, nos interesan los subconjuntos más simples de  $\Sigma^*$ . Quizás usted conjeturará que serían los subconjuntos que consisten en cadenas simples de longitud uno, y de hecho éstos son los lenguajes que utilizaremos como algunos de nuestros bloques de construcción. Después de todo, si  $\Sigma = \{x, y\}$ , entonces los lenguajes  $\{x\}$  y  $\{y\}$  parecerían ser los bloques de construcción naturales para la construcción de otros lenguajes de  $\Sigma$ .

Existen, sin embargo dos lenguajes que no podríamos construir utilizando estos bloques. Uno de ellos es el lenguaje  $\{\lambda\}$ ; el otro es  $\emptyset$ . Por razones que serán obvias en unos momentos, no empleamos  $\{\lambda\}$  como uno de nuestros bloques de construcción. En vez de esto, nuestra colección de bloques contiene el lenguaje vacío y todos los lenguajes de cadenas simples con longitud uno.

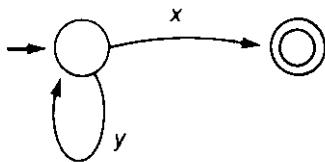
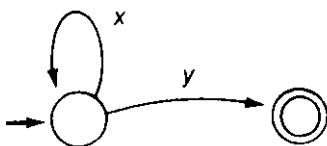
Una vez establecida esta base, nos centramos en las maneras como se pueden construir lenguajes más complicados a partir de los más básicos. Quizás la técnica más directa sea combinar dos lenguajes, utilizando la operación de unión de la teoría de conjuntos. Esta operación entre dos lenguajes se representan con el símbolo tradicional para la unión de conjuntos,  $\cup$ . Así, si  $L_1$  fuese el lenguaje  $\{x, xy\}$  y  $L_2$  fuera  $\{yz, yy\}$ , entonces el lenguaje  $L_1 \cup L_2$  sería  $\{x, xy, yx, yy\}$ .

¿Es la unión de dos lenguajes regulares otro lenguaje regular? Para responder esta pregunta, consideremos los diagramas de la figura 1.27a. Uno de ellos acepta el lenguaje que consiste en cero o más  $x$  seguidas por una sola  $y$ , mientras que el otro acepta cero o más  $y$  seguidas por una sola  $x$ . Para construir un diagrama que acepte la unión de estos lenguajes, uno tendería a identificar los estados iniciales de los dos diagramas, como se muestra en la figura 1.27b. Sin embargo, el diagrama resultante acepta la cadena  $xyyx$ , que no está en la unión de los dos lenguajes originales.

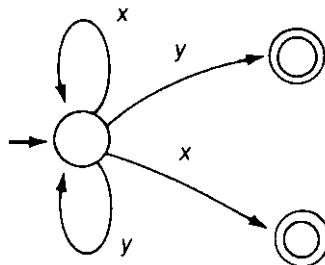
Aquí el problema es que este sencillo método de combinación de los diagramas originales permite mayor interacción que la que podemos permitir. Lo que necesitamos es una técnica de combinación que produzca un diagrama que nos restrinja a una u otra de las estructuras originales, sin permitir que alternemos entre ambas. La respuesta se encuentra en la introducción de un nuevo estado inicial a partir del cual podamos entrar a uno de los diagramas originales sin poder regresar, como se muestra en la figura 1.27c. La construcción general es la siguiente: dibuje un nuevo estado inicial; déclarélo como un estado de aceptación si y sólo si uno de los estados iniciales era de aceptación; de los estados iniciales originales dibuje un arco, con la misma etiqueta, a partir del nuevo estado inicial; para cada estado que sea el punto de destino de un arco de alguno elimine la característica de inicio de los estados iniciales originales.

Este proceso, cuando se aplica a cualquier par de diagramas de transiciones para autómatas finitos, producirá otro diagrama de transiciones para el cual cualquier cálculo es una copia del cálculo que habría efectuado una de las máquinas originales, y viceversa. Por lo tanto, *la unión de dos lenguajes regulares cualesquiera también es regular*.

a.



b.



c.

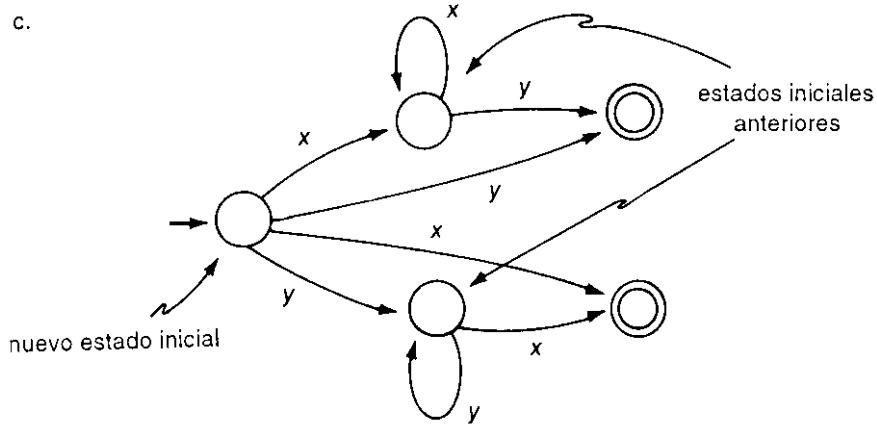


Figura 1.27 Formación de un diagrama de transiciones para un autómata finito que acepta la unión de dos lenguajes regulares

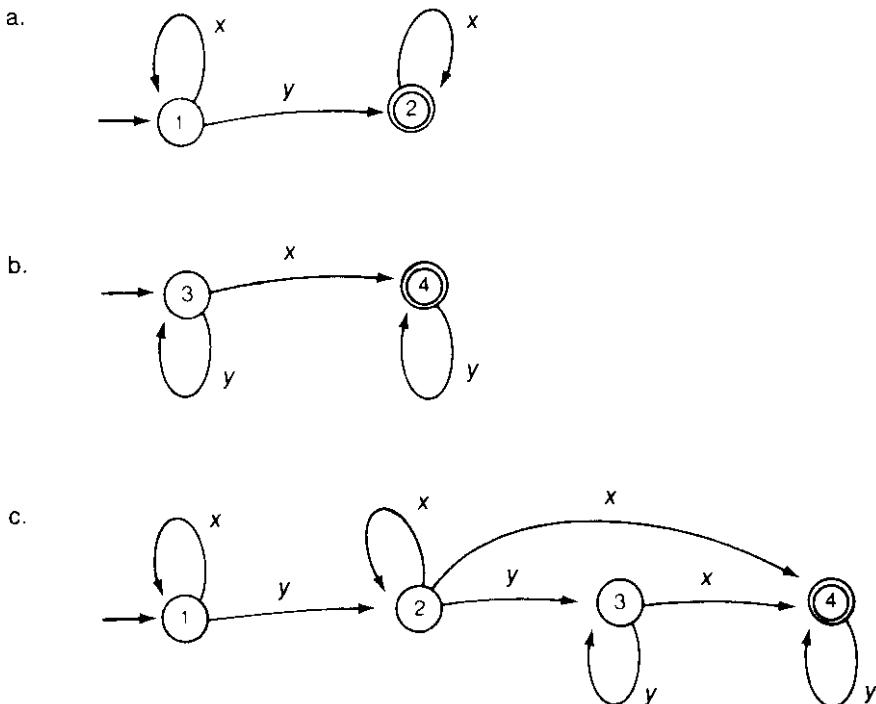
Otra técnica para combinar dos lenguajes es recopilar todas las cadenas formadas al concatenar una cadena del primer lenguaje y una cadena del segundo. La colección de las cadenas formadas de esta manera se denomina **concatenación** de dos lenguajes. La operación de concatenación se representa con un pequeño círculo,  $\circ$ . Así, si  $L_1$  fuera, una vez más, el lenguaje  $\{x, xy\}$  y  $L_2$  el lenguaje  $\{yx, yy\}$ , entonces el lenguaje  $L_1 \circ L_2$  sería  $\{xyx, xyy, xyyx, xyyy\}$ . Observe que  $L_1 \circ L_2$  no es igual que  $L_2 \circ L_1 = \{yxx, yxxy, yyx, yyxy\}$ . En otras palabras, la concatenación de lenguajes no es conmutativa.

Suponga que tenemos los diagramas de transiciones  $T_1$  y  $T_2$ , que aceptan los lenguajes  $L_1$  y  $L_2$  respectivamente, y deseamos construir un diagrama de transiciones que acepte  $L_1 \circ L_2$ . Procederíamos de la siguiente forma: a partir de cada estado de aceptación de  $T_1$ , dibuje un arco hacia cada estado de  $T_2$  que sea el destino de un arco del estado inicial de  $T_2$ . Rotule cada uno de estos arcos con la etiqueta del arco correspondiente en  $T_2$ . Deje que los estados de aceptación de  $T_1$  sigan siendo estados de aceptación si y sólo si el estado inicial  $T_2$  esté también un estado de aceptación (como ejemplo, véase Fig. 1.28). Observe que este diagrama combinado aceptará una cadena si y sólo si se trata de la concatenación de dos subcadenas, la primera que define una ruta del estado inicial de  $T_1$  a un antiguo estado de aceptación de  $T_1$ , y la segunda que define una ruta de este antiguo estado de aceptación a un estado de aceptación de  $T_2$ . Por lo tanto, el lenguaje aceptado por el diagrama combinado será  $L_1 \circ L_2$ . Así mismo, la concatenación de dos lenguajes regulares también es regular.

La última operación que consideraremos se conoce como **estrella de Kleene** (llamada así en honor de S. C. Kleene), y difiere de las anteriores en que amplía un solo lenguaje en vez de combinar dos. Esto se logra formando todas las concatenaciones de cero o más cadenas del lenguaje que se amplía (puesto que incluimos la concatenación de cero cadenas, la cadena vacía será un miembro del lenguaje ampliado). Esta operación se representa por medio de un asterisco supraíndice,  $^*$  (observe que ya hemos empleado la estrella de Kleene al representar con  $\Sigma^*$  el conjunto de todas las cadenas finitas que pueden formarse a partir del alfabeto  $\Sigma$ ).

Por ejemplo, si  $L_1$  es el lenguaje  $\{y\}$ , entonces  $L_1^*$  sería el lenguaje que consiste en todas las cadenas finitas de varias  $y$ , incluyendo la cadena vacía, o si  $L_2$  es el lenguaje  $\{yy\}$ , entonces  $L_2^*$  serían todas las cadenas que consisten en un número par de  $y$ , incluyendo la cadena vacía. Además, puesto que  $\lambda$  pertenece a la estrella de Kleene de cualquier lenguaje, debe pertenecer a  $\emptyset^*$ . Por ende,  $\emptyset^{**} = \{\delta\}$  (la razón por la cual no incluimos  $\{\delta\}$  como uno de nuestros bloques de construcción es que  $\{\delta\}$  se genera a partir de  $\emptyset$  por medio de la estrella de Kleene)

La intuición nos dice que la construcción de un diagrama de transiciones que acepte la estrella de Kleene de un lenguaje regular implica concatenar el diagrama de transiciones original hacia atrás consigo mismo. En este caso, nuestra intuición es correcta salvo por un pequeño detalle: el nuevo diagrama de transiciones debe aceptar la cadena vacía. Para lograr esto, nuestra intuición nos podría indicar que basta con designar el estado inicial original como un nuevo estado de aceptación, pero el problema no

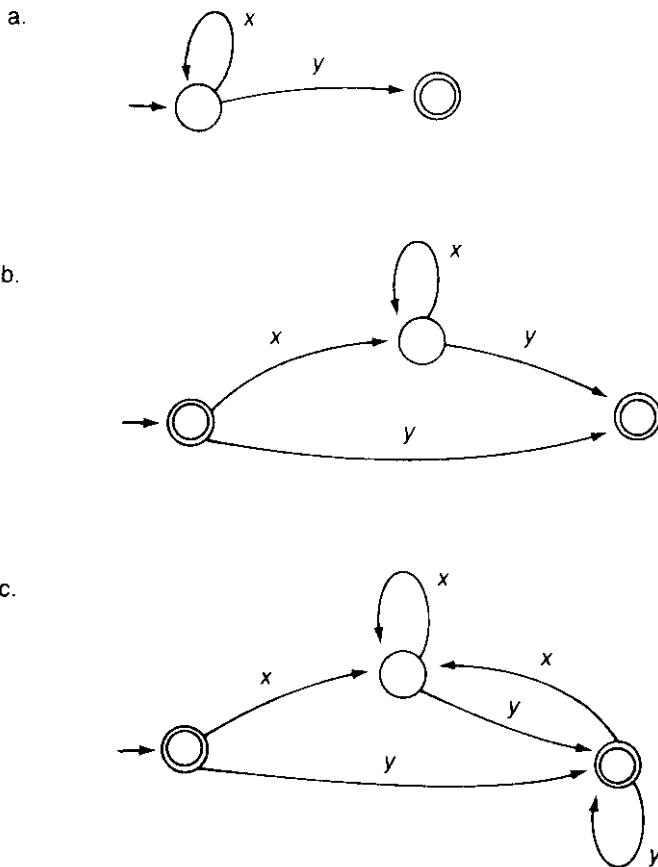


**Figura 1.28** Formación de un diagrama de transiciones para un autómata finito que acepta la concatenación de dos lenguajes regulares

se soluciona tan fácilmente. Por ejemplo, designar el estado inicial de la figura 1.29a como estado de aceptación no sólo permitiría la aceptación de la cadena vacía, sino además cadenas como xxxx.

Por consiguiente, nuestro primer paso para modificar un diagrama de transiciones a fin de que acepte la estrella de Kleene del lenguaje originalmente aceptado es dibujar un nuevo estado inicial y conectarlo al diagrama original, de manera muy parecida a lo que se hace en la operación de unión: por cada estado que sea dibujamos un arco con la misma etiqueta del nuevo estado inicial; hacia el destino de un arco del estado inicial del diagrama original luego designamos a éste como estado de aceptación (véase Fig. 1.29b).

Una vez que se ha agregado el nuevo estado inicial, nuestra siguiente tarea es modificar el diagrama para que podamos establecer un ciclo de los estados de aceptación al “inicio” del diagrama. Esto se logra añadiendo un arco de cada estado de aceptación a cada estado que es el destino de un arco del estado inicial. Cada uno de estos nuevos arcos se rotula con la etiqueta que corresponda al arco del estado inicial. El resultado es un diagrama de transiciones que acepta una cadena no vacía si y sólo si esa cadena es la concatenación de cadenas aceptadas por el diagrama original (como ejemplo, véase Fig. 1.29c). Concluimos que *la estrella de Kleene de cualquier lenguaje regular es regular*.



**Figura 1.29** Formación de un diagrama de transiciones para un autómata finito que acepta la estrella de Kleene de un lenguaje regular

Dado un alfabeto particular y usando las operaciones de unión, concatenación y estrella de Kleene, podemos construir muchos lenguajes a partir de nuestros bloques de construcción. Para presentar este proceso con mayor precisión, establecemos la siguiente definición.

Una expresión regular (para un alfabeto  $\Sigma$ ) se define como sigue:

- $\emptyset$  es una expresión regular.
- Cada miembro de  $\Sigma$  es una expresión regular.
- Si  $p$  y  $q$  son expresiones regulares, entonces también lo es  $(p \cup q)$ .
- Si  $p$  y  $q$  son expresiones regulares, entonces también lo es  $(p \circ q)$ .
- Si  $p$  es una expresión regular, entonces también lo es  $p^*$ .

Como ejemplo, si el alfabeto  $\Sigma$  fuera  $\{x, y, z\}$ , entonces  $(x \cup (z \circ y))$  sería una expresión regular ya que  $(z \circ y)$ , por la regla d, es una expresión regular y en consecuencia, por la regla c,  $(x \cup (z \circ y))$  sería una expresión regular.

Cada expresión regular  $r$  de un alfabeto  $\Sigma$  representa un lenguaje, denotado por  $L(r)$ , que se construye a partir de nuestros bloques de construcción. Para ser más precisos,  $L(\emptyset)$  es el lenguaje  $\emptyset$ , y para cada  $x \in \Sigma$ ,  $L(x)$  es el lenguaje  $\{x\}$ . Además, si  $p$  y  $q$  son expresiones regulares, entonces  $L((p \cup q)) = L(p) \cup L(q)$ ,  $L((p \circ q)) = L(p) \circ L(q)$ , y  $L(p^*) = L(p)^*$ . Por ejemplo, la expresión  $(x \cup (z \circ y))$  representa el lenguaje  $\{x, zy\}$ . Es decir, representa el lenguaje generado al unir  $\{x\}$  con la concatenación de  $\{z\}$  y  $\{y\}$ . De forma similar, la expresión  $((x \circ y)^* \cup z^*)$  (obtenida al aplicar la regla d a  $x$  y  $y$ , luego la regla e a  $(x \circ y)$  y  $z$ , y por último la regla c a  $(x \circ y)^*$  y  $z^*$ ) representa el lenguaje que consiste en cadenas de cero o más copias del patrón  $xy$  además de las cadenas de cero o más  $z$ .

Así, las expresiones regulares ofrecen otro medio, además de los diagramas de transiciones, los autómatas y las gramáticas, para especificar lenguajes. El teorema 1.6 muestra por qué nos interesan los lenguajes representados por lenguajes regulares.

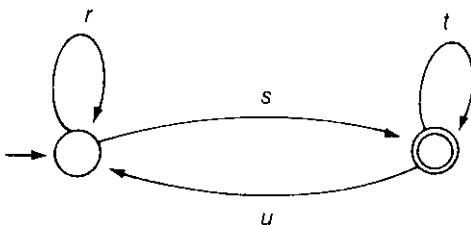
### TEOREMA 1.6

Dado un alfabeto  $\Sigma$ , los lenguajes regulares de  $\Sigma$  son exactamente los lenguajes representados por las expresiones regulares de  $\Sigma$ .

#### DEMOSTRACIÓN

Primero debemos mostrar que un lenguaje representado por una expresión regular es regular. Ya hemos visto que el lenguaje  $\emptyset$  es regular, así como cualquier lenguaje que contenga sólo una cadena de longitud uno. Además, hemos visto que la unión y la concatenación de dos lenguajes son regulares siempre y que la estrella de Kleene de cualquier lenguaje regular también es regular. Así, el primer paso de nuestra demostración está completo.

A continuación mostramos que cualquier lenguaje aceptado por un autómata finito puede representarse con una expresión regular. Suponemos que  $T$  es el diagrama de transiciones para un autómata finito y mostramos, por inducción para el número de estados en  $T$  que no son el inicial o los de aceptación, que existe una expresión regular que representa los lenguajes aceptados por  $T$ . Los diagramas que consideraremos son un poco más generales que los diagramas de transiciones tradicionales, ya que sus arcos pueden estar rotulados con expresiones regulares (para recorrer uno de estos arcos se requiere que la máquina lea un patrón compatible con la expresión). Si generalizamos los lenguajes aceptados por estos



**Figura 1.30** Arcos posibles en un diagrama de transiciones con dos estados, uno inicial y otro de aceptación

diagramas pueden representarse con expresiones regulares, entonces también puede representarse con ellas cualquier lenguaje aceptado por un diagrama de transiciones tradicional para un autómata finito.

Nuestro proceso de inducción supone que  $T$  sólo tiene un estado de aceptación. De lo contrario, para cada estado de aceptación podríamos hacer una copia aparte de  $T$  donde sólo ese estado sería de aceptación, encontrar expresiones regulares relacionadas con estos diagramas y luego formar la unión de estas expresiones a fin de obtener la expresión deseada para  $T$ .

Para comenzar nuestro proceso de inducción, supongamos entonces que  $T$  es un diagrama de transiciones generalizado con sólo un estado de aceptación, y que cada estado de  $T$  es un estado inicial o de aceptación. Entonces existen dos posibilidades:  $T$  contiene sólo un estado que es tanto el inicial como el de aceptación, o  $T$  contiene dos estados, de los cuales uno es el inicial y el otro es el de aceptación. En el primer caso, la expresión deseada no es más que la estrella de Kleene de la unión de las etiquetas que aparecen en los arcos del diagrama.

El segundo caso es un poco más complejo. Si existen varios arcos que conectan los mismos estados en la misma dirección, los sustituimos con un solo arco rotulado con la expresión regular que representa la unión de las etiquetas originales. Al llegar a este punto,  $T$  puede contener, a lo sumo, cuatro arcos: uno del estado inicial al estado inicial, uno del estado inicial al de aceptación, uno del estado de aceptación a sí mismo y uno del estado de aceptación al inicial. Representaremos estos arcos con las etiquetas  $r$ ,  $s$ ,  $t$  y  $u$  respectivamente (véase Fig. 1.30).

Si el arco  $s$  no está presente en el diagrama, entonces la expresión regular relacionada es  $\emptyset$  ya que no habría manera de llegar al estado de aceptación desde el inicial. De lo contrario, la estructura de la expresión regular relacionada dependerá de la ausencia o existencia del arco con etiqueta  $u$ . Si no existe este arco, entonces la expresión regular deseada es

$$((r^* \circ s) \circ t^*)$$

donde  $r$  y  $t$  son sustituidos por  $\emptyset$  si no existe en el diagrama el arco correspondiente. Si existe un arco del estado de aceptación al estado inicial, entonces la expresión deseada es

$$(((r^* \circ s) \circ t^*) \circ (u \circ ((r^* \circ s) \circ t^*))^*)$$

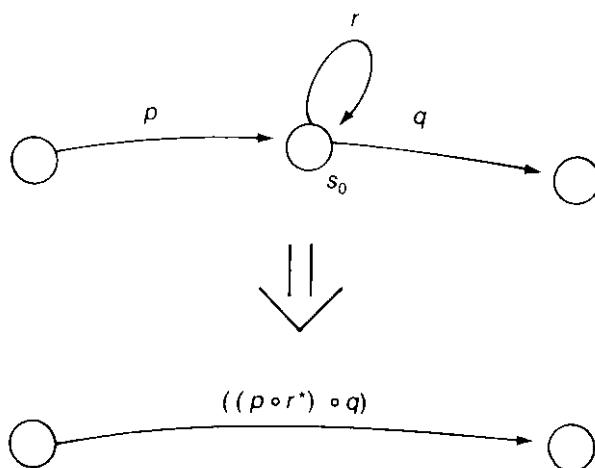
donde, una vez más,  $r$  y  $t$  son sustituidos por  $\emptyset$  si no existe en  $T$  el arco correspondiente.

Supongamos ahora que  $n \in \mathbb{N}$  y que pueda representarse con una expresión regular el lenguaje aceptado por cualquier diagrama de transiciones generalizado que no tenga más de  $n$  estados que no sean el inicial o los de aceptación. Para cualquier diagrama de transiciones generalizado con  $n + 1$  estados que no sean el inicial o los de aceptación, proseguimos de la siguiente manera (véase Fig. 1.31): seleccione un estado  $s_0$  de  $T$  que no sea inicial o de aceptación; elimine de  $T$  ese estado y todos sus arcos incidentes; para cada par de arcos eliminados (uno que conduce de otro estado a  $s_0$  y otro que sale de  $s_0$  a otro estado, con etiquetas  $p$  y  $q$  respectivamente), dibuje un arco desde el origen del arco con etiqueta  $p$  al destino del arco con etiqueta  $q$ , y rotule este nuevo arco como  $((p \circ r^*) \circ q)$ , donde  $r$  es la unión de las etiquetas de los arcos que originalmente iban de  $s_0$  a  $s_0$ , o bien  $\emptyset$  si no existían tales arcos. El diagrama resultante es otro diagrama de transiciones generalizado que aceptará el mismo lenguaje que  $T$  y que sólo tiene  $n$  estados que no sean iniciales o de aceptación. Entonces, por nuestra hipótesis de inducción, es posible representar el lenguaje en cuestión con una expresión regular.



El teorema 1.6 proporciona una forma más de caracterizar lenguajes regulares: son los lenguajes aceptados por autómatas finitos deterministas, los lenguajes aceptados por autómatas finitos no deterministas, los lenguajes generados por gramáticas regulares y los lenguajes representados por expresiones regulares.

De lo anterior podemos concluir que el poder de las expresiones regulares es un tanto limitado, pero no significa que estas sean inútiles. Por el contrario, dichas expresiones proporcionan detalles de la estructura de los lenguajes regulares que quizás no sean aparentes en otras caracterizaciones. Asimismo, las expresiones regulares proporcionan una manera relativamente concisa para expresar y comunicar lenguajes regulares. Por ejemplo, para describir un lenguaje regular utilizando un autómata finito, se requiere una definición de la función de transición, quizás en forma de diagrama o de tabla de transiciones; para describir un lenguaje por medio de una gramática, se



**Figura 1.31** Eliminación de un estado  $s_0$  de un diagrama de transiciones generalizado para un autómata finito

requiere una lista de reglas de reescritura; pero, utilizando una expresión regular, se puede describir un lenguaje en una sola línea.

Uno de los principales ejemplos de las ventajas de esta notación concisa se puede encontrar en muchos de los editores de texto que se emplean actualmente, los cuales permiten buscar un patrón específico en un archivo, para eliminarlo o reemplazarlo por otra cadena. En esta situación se requiere un sistema de notación para comunicar la forma del patrón a encontrar. Aunque la sintaxis que se emplee puede ser distinta de la que aquí se ha presentado, en estas aplicaciones es fundamental el tema de la expresión del patrón por medio de las operaciones de unión, concatenación y estrella de Kleene. Como ejemplo específico, al utilizar el editor *ed* de UNIX, el mandato

$$s/xx^*/z/$$

indica al sistema que sustituya la primera ocurrencia de una o más  $x$  por una sola  $z$ . En este caso, la expresión  $xx^*$  representa una sola  $x$  concatenada con la estrella de Kleene de  $x$ .

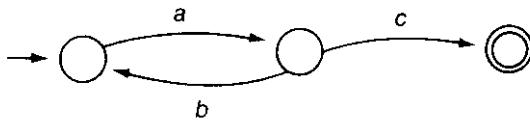
Por último, una consecuencia de los teoremas 1.3 y 1.6 es que los sistemas de comparación de patrones que se basan exclusivamente en la unión, la concatenación y la estrella de Kleene para la representación de patrones, son un tanto restrictivos en cuanto a los patrones que se pueden solicitar. Por otra parte, como testimonio del alcance de los lenguajes regulares, se han diseñado muchos sistemas de gran utilidad a pesar de estas restricciones.

## Ejercicios

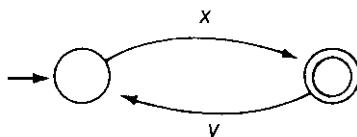
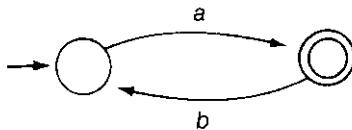
1. ¿Cuáles de los lenguajes descritos por las siguientes expresiones regulares para el alfabeto  $\{x, y, z\}$  son infinitos? Describa, en una sola frase, el contenido de cada uno de estos lenguajes infinitos y elabore una lista exhaustiva de las cadenas de los lenguajes que sean finitos.

$$\begin{array}{ll} \text{a. } (x \circ (y \circ z^*)) & \text{e. } (y \circ y)^* \\ \text{b. } (x^* \circ (y \circ z)) & \text{f. } (x^* \cup y^*) \\ \text{c. } ((z \cup y) \circ x) & \text{g. } ((x \circ x) \cup z) \\ \text{d. } (z \cup y)^* & \text{h. } ((z \cup y) \cup x) \end{array}$$

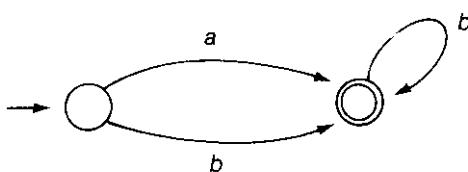
2. Dibuje un diagrama de transiciones que acepte la estrella de Kleene del lenguaje aceptado por el siguiente diagrama.



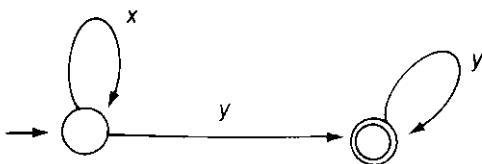
3. Dibuje un diagrama de transiciones que acepte la unión de los lenguajes aceptados por los siguientes diagramas.



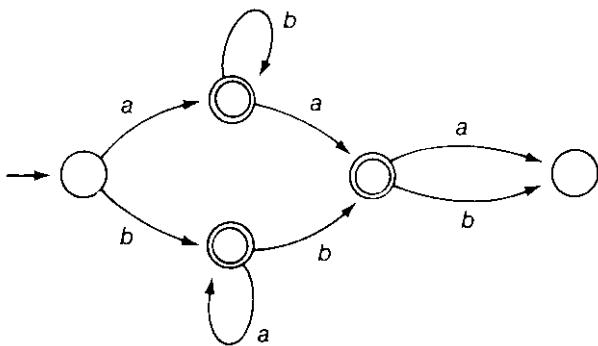
4. Dibuje un diagrama de transiciones que acepte la concatenación del lenguaje aceptado por



seguido por el lenguaje aceptado por



5. Construya una expresión regular que describa el lenguaje aceptado por el siguiente diagrama de transiciones.



## 1.7 COMENTARIOS FINALES

Este capítulo ha presentado la clase de los lenguajes regulares y ha mostrado cómo pueden caracterizarse usando autómatas finitos, gramáticas regulares y expresiones regulares. Además de ser importantes en sí, muchos de los conceptos que se presentan en este capítulo también sirven como introducción de los conceptos de los capítulos restantes. Por ejemplo, hemos presentado las máquinas deterministas y las no deterministas, una distinción que no afecta el poder computacional de los autómatas finitos. No obstante, en el siguiente capítulo veremos que, en otros contextos, esta diferencia puede tener ramificaciones importantes. También hemos visto cómo la tarea de análisis léxico de un compilador puede reducirse a un sencillo proceso de consulta de una tabla. Aunque las tablas tienen una utilidad potencial en este contexto, su empleo es más común en las rutinas de análisis sintáctico de los compiladores, algo que veremos en el siguiente capítulo.

El concepto de gramática presentado en este capítulo es importante ya que la sintaxis de la mayoría de los lenguajes de programación está descrita hoy en día por gramáticas, y en esta estructura gramatical se basa el proceso de compilación. Gran parte del estudio de los siguientes capítulos implicará la cuestión de cuáles son los lenguajes que pueden generar las gramáticas

cuyas reglas de reescritura se adhieren a distintas restricciones. Esto tiene relevancia en la medida en que, al hacerse más complejas las reglas de reescritura, también el algoritmo de reconocimiento del lenguaje asociado se vuelve más complicado. Así, si nos asignan la tarea de diseñar un compilador para un lenguaje de programación (o algún tipo de procesador para un lenguaje natural), nos gustaría basar nuestro diseño en la gramática más sencilla posible.

Sin embargo, nuestro objetivo final es descubrir hasta dónde pueden emplearse las máquinas para resolver problemas; hasta ahora, el problema que hemos elegido es el del análisis sintáctico de lenguajes. Como veremos en los capítulos siguientes, este problema tiene la complejidad suficiente para que la búsqueda de su solución nos lleve a los límites aparentes de los procesos computacionales.

## Problemas de repaso del capítulo

---

1. Muestre que la colección de todas las cadenas de  $x, y$  y  $z$ , que contienen un número impar de  $x$ , un número impar de  $y$  y un número par de  $z$  es un lenguaje regular del alfabeto  $\{x, y, z\}$ .
2. Muestre que si  $L_1$  y  $L_2$  son lenguajes regulares, entonces  $L_1 \cap L_2$  es regular.
3. Muestre que si  $L_1$  y  $L_2$  son lenguajes regulares, entonces  $L_1 - L_2$  es regular.
4.
  - a. Si  $\Sigma$  es un alfabeto, ¿es la colección de palíndromos de  $\Sigma^*$  un lenguaje regular? ¿Por qué?
  - b. Si  $L$  es un lenguaje regular, ¿es la colección de palíndromos de  $L$  un lenguaje regular? ¿Por qué?
5. Muestre que si  $L$  es un lenguaje regular del alfabeto  $\Sigma$ , entonces también es regular el lenguaje que consiste en todas las cadenas de la forma  $wv$ , donde  $w \in L$  y  $v \in \Sigma^* - L$ .
6. Muestre que si  $L$  es un lenguaje regular, entonces también es regular el lenguaje que se obtiene al escribir en forma inversa las cadenas de  $L$ .
7. Muestre que si  $L$  es un lenguaje regular, entonces también es regular la colección de cadenas cuyas inversas se encuentran así mismo en  $L$ .

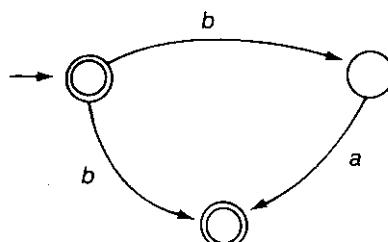
8. La cadena 527943001 es una lista de los dígitos que aparecen en el problema de adición presentado a continuación, si leyéramos las columnas en forma descendente y de derecha a izquierda (colocamos los ceros iniciales que se supone que están en la columna extrema izquierda). Diseñe un diagrama de transiciones para un autómata finito que acepte únicamente aquellas cadenas de dígitos que pueden obtenerse de un problema de adición de este tipo.

$$\begin{array}{r} 95 \\ + 42 \\ \hline 137 \end{array}$$

9. Se dice que una regla de reescritura de una gramática, que sustituye un no terminal por un par terminal-no terminal, es **lineal por la izquierda** o **lineal por la derecha**, dependiendo de si el no terminal está a la izquierda o a la derecha (respectivamente) del terminal (la regla  $N \rightarrow Mx$  sería lineal por la izquierda, mientras que la regla  $N \rightarrow xM$  sería lineal por la derecha). Observe que nuestra definición de una gramática regular permitía reglas lineales por la derecha pero no por la izquierda. Muestre que los lenguajes regulares se generan de nuevo si cambiamos la definición para que permita reglas lineales por la izquierda, a la vez que se excluyen las lineales por la derecha.

Explique cómo se relaciona este problema con el problema 6.

10. Muestre que no es regular el lenguaje de  $\{x, y\}$  que consiste en aquellas cadenas con el mismo número de  $x$  y  $y$ .
11. Muestre que es regular el lenguaje de  $\{x, y\}$  que consiste en aquellas cadenas que no contienen tres  $x$  consecutivas.
12. Muestre que cualquier lenguaje finito es regular.
13. Diseñe un diagrama de transiciones para un autómata finito determinista que acepte las mismas cadenas que el autómata finito no determinista representado a continuación:



14. Suponga que permitimos que los autómatas finitos cambien de un estado a otro sin leer un símbolo de sus cintas de entrada. En un diagrama de transiciones, este tipo de transiciones normalmente se representa como un arco con etiqueta  $\lambda$  en vez de un símbolo del alfabeto, y se le conoce como **transición  $\lambda$** . Muestre que cualquier diagrama de transiciones que tenga transiciones  $\lambda$  se puede modificar para que ya no contenga este tipo de transiciones pero que a la vez acepte el mismo lenguaje (el diagrama modificado puede ser no determinista).
15. Describa el lenguaje representado por cada una de las siguientes expresiones regulares.
- a.  $(z \cup y)^* \circ x$       c.  $((x \circ x^*) \cup (y \circ y^*))$   
b.  $((x \circ x^*) \circ y \circ y^*)$       d.  $((x^* \circ y^*) \circ z^*)$
16. Escriba expresiones regulares que describan los siguientes lenguajes.
- a. Todas las cadenas que consisten en un número impar de  $x$ .  
b. Todas las cadenas que consisten en un número impar de  $x$  y un número par de  $y$ .  
c. Todas las cadenas de  $x$  y  $y$  tales que cada  $y$  esté inmediatamente precedida por una  $x$  e inmediatamente seguida por una  $x$ .
17. Desarrolle una gramática para generar las cadenas que constituyen literales de tipo real en el lenguaje de programación Pascal.
18. Suponga que  $L$  es un lenguaje regular del alfabeto  $\Sigma = \{x, y\}$ . Muestre que  $\{w: w \in \Sigma^* \text{ y } xw \in L\}$  y  $\{w: w \in \Sigma^* \text{ y } wx \in L\}$  son también lenguajes regulares de  $\Sigma$ .
19. Encuentre un lenguaje regular (que no contenga  $\lambda$ ) que no pueda ser el lenguaje aceptado por un autómata finito determinista con un solo estado de aceptación.
20. Muestre que si un autómata finito determinista  $M$  acepta una cadena que contiene más símbolos que estados tiene  $M$ , entonces  $M$  debe aceptar un número infinito de cadenas.
21. Diseñe un algoritmo que convierta diagramas de transiciones de autómatas finitos a expresiones regulares equivalentes.
22. Muestre que el lenguaje  $\{x^n: n \text{ es primo}\}$  no es regular.
23. Muestre que el lenguaje  $\{x^n: n = m^2 \text{ para algún } m \in \mathbb{N}\}$  no es regular.
24. Encuentre una expresión regular que represente la intersección de los lenguajes representados por cada uno de los siguientes pares de expresiones regulares.

- a.  $(x \cup y^*)y(x \cup y)^*$   
 b.  $(x \circ (x \cup y)^*)y((x \cup y)^* \circ y)$   
 c.  $((x \cup y) \circ y) \circ (x \cup y)^*y(y \circ (x \cup y)^*)$   
 d.  $((x \cup y) \circ (x \cup \emptyset))y((x \cup y) \circ (x \circ y)^*)$
25. Muestre que si  $L$  es un lenguaje regular, entonces el conjunto de cadenas en  $L$  de longitud impar es también un lenguaje regular.  
 ¿Se aplica lo mismo al conjunto de cadenas de longitud par? Justifique su respuesta.
26. Muestre que si  $L$  es un lenguaje regular que no contiene  $\lambda$ , entonces  $L$  puede ser aceptado por un autómata finito no determinista con un solo estado de aceptación.
27. Hemos mostrado que la unión de dos lenguajes regulares es regular. ¿Es siempre regular la unión de una colección de lenguajes regulares? Justifique su respuesta.
28. Encuentre un diagrama de transiciones que acepte el lenguaje generado por la gramática regular que se presenta a continuación (el símbolo inicial es  $S$ ). Luego, encuentre una expresión regular que represente el mismo lenguaje.

$$\begin{array}{l} S \rightarrow xN \\ S \rightarrow x \\ N \rightarrow yM \\ N \rightarrow y \\ M \rightarrow zN \\ M \rightarrow z \end{array}$$

29. Muestre que un lenguaje regular que no contiene  $\lambda$  puede ser generada una gramática por regular que no contenga ninguna regla  $\lambda$ .
30. Muestre que si el lenguaje aceptado por un autómata finito contiene una cadena no vacía, entonces debe contener una cadena no vacía cuya longitud no sea mayor que el número de estados en el autómata.
31. Diseñe un diagrama de transiciones para un autómata finito que acepte el lenguaje consistente en las cadenas del alfabeto  $\{w, x, y, z\}$  en donde el patrón  $xy$  siempre esté seguido por una  $w$  y donde al patrón  $yx$  siempre siga una  $z$ .
32. ¿Es cada lenguaje regular el lenguaje aceptado por un autómata finito determinista cuyo diagrama de transiciones pueda dibujarse sobre una superficie plana sin que alguno de sus arcos se crucen?

## Problemas de programación

---

1. Escriba un programa que evalúe cadenas de entrada para ver si se adhieren a los patrones descritos por la figura 1.17.
2. Diseñe e implante un algoritmo para evaluar gramáticas regulares y determinar si generan por lo menos una cadena no vacía (puede comenzar por traducir el problema 30 de repaso del capítulo a un enunciado acerca de gramáticas).
3. Desarrolle un paquete de software que genere automáticamente analizadores sintácticos para lenguajes regulares. En primer lugar, escriba un programa que acepte gramáticas regulares como entrada y genere como salida las tablas de transiciones correspondientes. Luego, escriba un programa que analice su cadena de entrada de acuerdo con la tabla de transiciones generada por el programa anterior.
4. Desarrolle un paquete similar al del problema de programación 3, pero que acepte expresiones regulares en vez de gramáticas regulares.

## CAPÍTULO 2

# Autómatas de pila y lenguajes independientes del contexto

---

### 2.1 Autómatas de pila

Definición de los autómatas de pila

Autómatas de pila como aceptadores de lenguajes

### 2.2 Gramáticas independientes del contexto

Definición de las gramáticas independientes del contexto

Gramáticas independientes del contexto y autómatas de pila

Forma normal de Chomsky

### 2.3 Límites de los autómatas de pila

Alcance de los lenguajes independientes del contexto

Autómatas de pila deterministas

Principio de preanálisis

### 2.4 Analizadores sintácticos $LL(k)$

Proceso de análisis sintáctico  $LL$

Aplicación del principio de preanálisis

Tablas de análisis sintáctico  $LL$

### 2.5 Analizadores sintácticos $LR(k)$

Proceso de análisis sintáctico  $LR$

Implantación de analizadores sintácticos  $LR(k)$

Tablas de análisis sintáctico  $LR$

Comparación entre los analizadores sintácticos  $LR(k)$  y  $LL(k)$

### 2.6 Comentarios finales

En el capítulo anterior vimos cómo se pueden construir analizadores léxicos utilizando los principios de los autómatas finitos, e investigamos los lenguajes regulares que estos autómatas son capaces de reconocer. También vimos que las técnicas sencillas relacionadas con los autómatas finitos son limitadas; específicamente, encontramos que los sistemas de

análisis sintáctico basados en estos autómatas no eran capaces de manejar gran parte de las estructuras sintácticas que se presentan en los lenguajes de programación generales.

En este capítulo generalizamos los conceptos de autómatas finitos y gramáticas regulares a fin de obtener técnicas para el análisis sintáctico de una mayor gama de lenguajes, conocidos como lenguajes independientes del contexto. Para esto, agregamos al autómata un sistema de memoria interna (en forma de pila). Esta adición incrementa de manera considerable el potencial de procesamiento de lenguaje del autómata y proporciona un marco en el cual se formulan varios algoritmos eficientes para el análisis sintáctico; el capítulo concluye con un estudio de las técnicas de análisis sintáctico que se encuentran en los compiladores modernos.

Para clasificar los lenguajes reconocidos por los autómatas mejorados, utilizaremos de nuevo el concepto de gramática. Al permitir mayor complejidad en la estructura de las reglas de reescritura de la gramática, seremos capaces de identificar las gramáticas que generan los lenguajes reconocidos por nuestras máquinas mejoradas. Esta caracterización gramatical será de utilidad en varias situaciones; con estas gramáticas se describe la sintaxis de la mayoría de los lenguajes de programación modernos.

## 2.1 AUTÓMATAS DE PILA

Como vimos en el capítulo 1, no existe ningún autómata finito que pueda reconocer el lenguaje  $\{x^n y^n : n \in \mathbb{N}\}$ . De hecho, este lenguaje (donde  $x$  era un paréntesis izquierdo y  $y$  un paréntesis derecho) fue nuestro principal ejemplo de las limitaciones de los autómatas finitos. Presentamos la hipótesis de que este problema ocurre porque los autómatas finitos no tienen forma de recordar cuántas  $x$  se detectaron en la primera parte de la cadena, por lo que eran incapaces de verificar si existía el mismo número de  $y$ . Por consiguiente, ahora especulamos que el problema podría resolverse añadiendo algún tipo de memoria a la máquina conceptual en consideración.

### Definición de los autómatas de pila

Después de esta introducción, presentamos la clase de máquinas conocidas como **autómatas de pila**; una máquina de este tipo se representa en la figura 2.1. Al igual que un autómata finito, un autómata de pila cuenta con un flujo de entrada y un mecanismo de control que puede encontrarse en uno de entre un número finito de estados. Uno de estos estados se designa como el inicial y por lo menos un estado se designa como estado de aceptación. La principal diferencia entre los autómatas de pila y los finitos es que los primeros cuentan con una pila en donde pueden almacenar información para recuperarla más tarde.

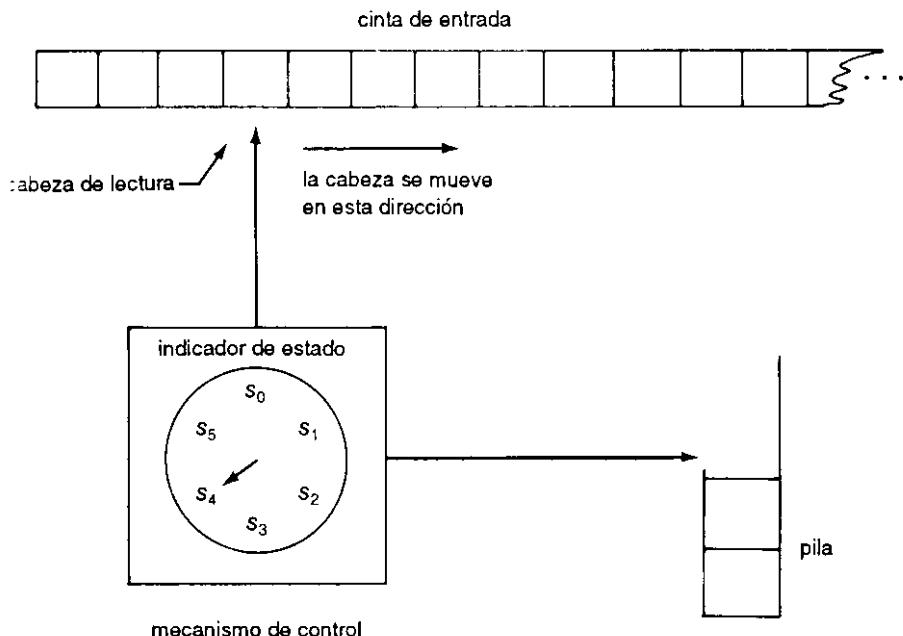


Figura 2.1 Autómata de pila

Los símbolos que pueden almacenarse en esta pila (conocidos como **símbolos de pila** de la máquina) constituyen un conjunto finito que puede incluir algunos o todos los símbolos del alfabeto de la máquina y quizás algunos símbolos adicionales que la máquina utiliza como marcas internas. Por ejemplo, una máquina podrá almacenar símbolos especiales en su pila para separar secciones que tengan interpretaciones distintas. Para ser más precisos, si una máquina inserta un símbolo especial en la pila antes de efectuar algún otro cálculo, entonces la presencia de ese símbolo en la cima de la pila puede usarse como indicador de "pila vacía" para cálculos posteriores. En nuestros ejemplos adoptamos el símbolo # para este fin.

Las transiciones que ejecutan los autómatas de pila deben ser variantes de la siguiente secuencia básica: leer un símbolo de la entrada, extraer un símbolo de la pila, insertar un símbolo en la pila y pasar a un nuevo estado. Este proceso se representa con la notación  $(p, x, s; q, y)$ , donde  $p$ ,  $x$ ,  $s$ ,  $q$  y  $y$  son, respectivamente el estado actual, el símbolo del alfabeto que se lee de la entrada, el símbolo que se extrae de la pila, el nuevo estado y el símbolo que se inserta en la pila. Esta notación está diseñada para indicar que, el estado actual, el símbolo de entrada y el símbolo en la cima de la pila ayudan a determinar conjuntamente el nuevo estado y el símbolo que deberá insertarse en la pila.

Se obtienen variantes de este proceso básico de transición permitiendo que las transiciones lean, extraigan o inserten la cadena vacía. Por ejemplo,

una transición posible sería  $(p, \lambda, \lambda; q, \lambda)$ . Es decir, al encontrarse en el estado  $p$ , la máquina podría no avanzar su cabeza de lectura (lo que consideramos como la lectura de la cadena vacía), no extraer un símbolo de su pila (extraer la cadena vacía), no insertar un símbolo en su pila (insertar la cadena vacía), y pasar al estado  $q$ . Otro ejemplo es la transición que sólo pasa del estado  $p$  al estado  $q$  extrayendo el símbolo  $s$  de la pila, lo cual se representa con  $(p, \lambda, s; q, \lambda)$ . Otros ejemplos incluyen transiciones como  $(p, x, \lambda; q, z)$ ,  $(p, \lambda, \lambda; q, z)$ , etcétera.

Para representar la colección de transiciones disponibles para un autómata de pila, es conveniente utilizar un diagrama de transiciones que semeje el de un autómata finito, donde los estados se representan con pequeños círculos y las transiciones por medio de arcos entre los círculos. Sin embargo, en el caso de los autómatas de pila, la rotulación de los arcos es más elaborada ya que hay que presentar más información. Un arco de  $p$  a  $q$  que representa la transición  $(p, x, y; q, z)$  tendría una etiqueta  $x, y; z$ . Por ejemplo, la figura 2.2 muestra un diagrama de transiciones para un autómata de pila en donde el estado inicial es el estado 1 (indicado por el apuntador) y los estados 1 y 4 son de aceptación (indicados por los círculos dobles). A partir de este diagrama podemos observar que si la máquina lee una  $x$  de la entrada cuando se encuentra en el estado 2, insertará una  $x$  en la pila y regresará al estado 2; si la máquina lee una  $y$  de la entrada y puede extraer una  $x$  de la pila cuando se encuentra en el estado 3, regresará al estado 3; o si el símbolo  $\#$  se encuentra en la cima de la pila cuando la máquina se halla en el estado 3, la máquina puede extraer este símbolo y pasar al estado 4.

Como sucede con los autómatas finitos, es posible implantar los autómatas de pila con varias tecnologías, por lo que aislamos las propiedades definitivas de los autómatas de pila en la siguiente definición formal.

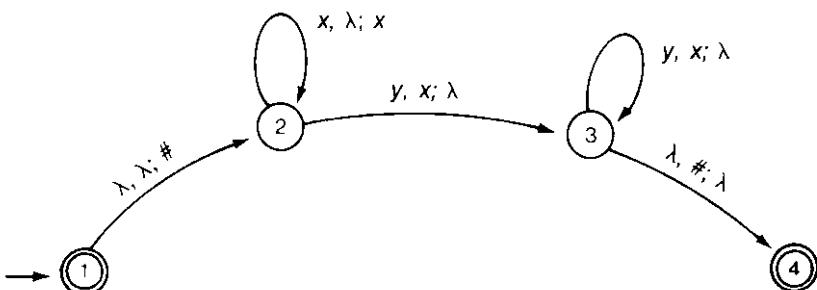


Figura 2.2 Diagrama de transiciones para un autómata de pila

Un autómata de pila es una séxtupla de la forma  $(S, \Sigma, \Gamma, T, i, F)$ , donde:

- $S$  es una colección finita de estados.
- $\Sigma$  es el alfabeto de la máquina.
- $\Gamma$  es la colección finita de símbolos de pila.
- $T$  es una colección finita de transiciones.
- $i$  (un elemento de  $S$ ) es el estado inicial.
- $F$  (un subconjunto de  $S$ ) es la colección de estados de aceptación.

### Autómatas de pila como aceptadores de lenguajes

Los autómatas de pila se pueden utilizar para analizar cadenas, en forma similar a como se usan los autómatas finitos. En la cinta de entrada de la máquina colocamos la cadena que se analizará, con la cabeza de lectura sobre la celda del extremo izquierdo de la cinta. Luego ponemos en marcha la máquina desde su estado inicial, con la pila vacía, y declaramos que la cadena se aceptará si es posible que la máquina llegue a un estado de aceptación después de leer toda la cadena. Esto no quiere decir que la máquina deba encontrarse en un estado de aceptación inmediatamente después de leer el último símbolo de la cadena de entrada, sino que significa que ya no tiene que leer más de la cinta. Por ejemplo, después de leer el último símbolo de la entrada, un autómata de pila puede ejecutar varias transiciones de la forma  $(p, \lambda, x; q, y)$  antes de aceptar la cadena.

Usamos la palabra "posible" al definir la aceptación de cadenas en un autómata de pila ya que los autómatas de este tipo que aquí se consideran son no deterministas; no hemos establecido restricción alguna con respecto al número de transiciones que son aplicables en un instante determinado. Por lo tanto, como sucede con los autómatas finitos no deterministas, pueden existir varias secuencias de transiciones que se recorran a partir de la configuración inicial de la máquina, de las cuales sólo una debe conducir a un estado de aceptación para que declaremos la aceptación de la cadena (es lamentable que la terminología común no haga hincapié en la naturaleza no determinista de los autómatas de pila. Técnicamente, deberían llamarse autómatas de pila no deterministas).

Siguiendo los caminos establecidos para los autómatas finitos, nos referimos a la colección de todas las cadenas aceptadas por un autómata de pila  $M$  como el lenguaje aceptado por la máquina, representado por  $L(M)$ . Subrayamos de nuevo que el lenguaje  $L(M)$  no es cualquier colección de cadenas aceptadas por  $M$ , sino la colección de *todas* las cadenas que acepta  $M$ .

Se puede obtener una importante clase de máquinas restringiendo las transiciones disponibles para un autómata de pila a las de la forma  $(p, x, \lambda; q, \lambda)$ . Los pasos de esta forma ignoran que la máquina tiene una pila y, por consiguiente, las actividades de la máquina dependen exclusivamente del estado y el símbolo de entrada actuales. Por ende, la clase de máquinas construidas de esta manera es la clase de los autómatas finitos. Por lo tanto, los lenguajes aceptados por los autómatas de pila incluyen los lenguajes regulares.

Los autómatas de pila también pueden aceptar lenguajes que no pueden aceptar los autómatas finitos, por ejemplo el lenguaje  $\{x^n y^n : n \in \mathbb{N}\}$ . De hecho, la figura 2.2 es un diagrama de transiciones de dicha máquina. El primer paso es marcar la parte inferior de la pila con el símbolo # y luego insertar en la pila las  $x$  conforme se lean de la entrada. Luego la máquina extrae una  $x$  de la pila cada vez que se lee una  $y$ . De esta manera, cuando el símbolo # reaparece en la parte superior de la pila, se ha leído el mismo número de  $y$  y  $x$ . Observe que, como el estado inicial es también un estado de aceptación, se permite que la máquina acepte la cadena  $x^0 y^0$ , que es  $\lambda$ .

Antes de concluir esta sección, se necesitan algunos comentarios adicionales. Recuerde que el criterio de aceptación que se proporcionó antes permite que un autómata de pila declare la aceptación de una cadena sin que tenga que vaciar antes su pila. Por ejemplo, un autómata de pila basado en el diagrama de la figura 2.3 aceptará el lenguaje  $\{x^m y^n : m, n \in \mathbb{N}^+ \text{ y } m \geq n\}$ , pues se aceptarán aquellas cadenas con más  $x$  que  $y$  aunque queden  $x$  en la pila (observe que este autómata no aceptaría cadenas con más  $y$  que  $x$ , pues no podría leer todos los símbolos de dicha cadena).

Se podría conjeturar que la aplicación de una teoría basada en estos autómatas nos llevaría a módulos de programa que devolvieran el control a otros módulos dejando en la pila residuos de sus cálculos que podrían ocasionar confusiones en cálculos posteriores. Por esto, es frecuente que se prefiera considerar únicamente los autómatas de pila que vacían sus pilas antes de llegar a un estado de aceptación. En el teorema 2.1 podemos ver que esta restricción no reduce el poder de estas máquinas.

### TEOREMA 2.1

Para cada autómata de pila que acepte cadenas sin vaciar su pila, existe un autómata que acepta el mismo lenguaje pero que vacía su pila antes de llegar a un estado de aceptación.

### DEMOSTRACIÓN

Suponga que  $M = (S, \Sigma, \Gamma, T, \iota, F)$  es un autómata de pila que acepta cadenas sin tener que vaciar necesariamente su pila. Podemos modificar  $M$  de la manera siguiente:

1. Elimine la designación “inicial” del estado inicial de  $M$ . Luego, añada un nuevo estado inicial y una transición que permita a  $M$  pasar del nuevo estado inicial al anterior a la vez que inserta en la pila un símbolo especial # (que no se encontraba anteriormente en  $\Gamma$  ).

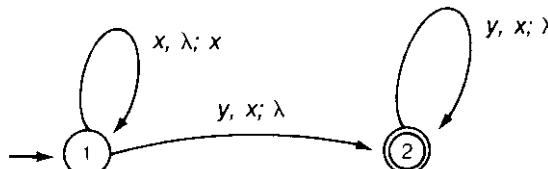


Figura 2.3 Autómata de pila que puede aceptar cadenas sin una pila vacía

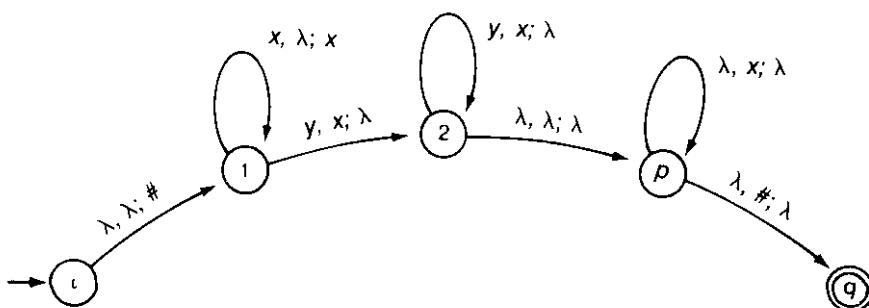
2. Elimine la característica de aceptación de cada estado de aceptación de  $M$ . Luego, añada un estado  $p$  junto con las transiciones que permiten a la máquina pasar de cada uno de los antiguos estados de aceptación a  $p$  sin leer, extraer o insertar un símbolo.
3. Para cada  $x$  en  $\Gamma$  (sin incluir  $\#$ ), introduzca la transición  $(p, \lambda, x; p, \lambda)$ .
4. Añada un nuevo estado de aceptación  $q$  y la transición  $(p, \lambda, \#; q, \lambda)$ .

Observe que la versión modificada de  $M$  sólo marca el fondo de su pila antes de efectuar algún cálculo y luego simula los cálculos de la máquina original hasta el punto donde la máquina original habría declarado la aceptación de la entrada. Aquí la máquina modificada pasa al estado  $p$ , vacía su pila y luego pasa a su estado de aceptación  $q$  quitando la marca de fin de pila. Así, tanto la máquina original como la modificada aceptan las mismas cadenas, aunque la versión modificada llega a su estado de aceptación únicamente cuando su pila está vacía.



La figura 2.4 muestra el resultado de aplicar la técnica de la demostración anterior al diagrama de la figura 2.3. Un autómata de pila basado en este nuevo diagrama aceptará exactamente las mismas cadenas que el original, pero no puede aceptar una cadena a menos que su pila se encuentre vacía.

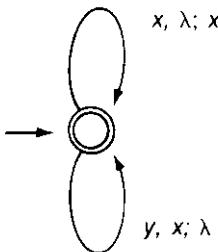
Por último, es importante recordar que los autómatas que aquí se consideran son no deterministas. El proceso de modificación descrito en la demostración del teorema 2.1 puede introducir varios puntos de no determinismo por medio de las transiciones que conducen de los antiguos estados de aceptación al nuevo estado  $p$ .



**Figura 2.4** El diagrama de la figura 2.3 después de modificarlo para que vacíe su pila antes de aceptar una cadena

## Ejercicios

- Diseñe un autómata de pila  $M$  tal que  $L(M) = \{x^n y^m x^n : m, n \in \mathbb{N}\}$
- ¿Cuál es el lenguaje que acepta el autómata de pila cuyo diagrama de transiciones se presenta a continuación?



- Modifique el diagrama de transiciones del ejercicio 2 para que el autómata de pila acepte el mismo conjunto de cadenas pero con su pila vacía.
- Muestre cómo pueden combinarse dos autómatas de pila  $M_1$  y  $M_2$  para formar un solo autómata de pila que acepte el lenguaje  $L(M_1) \cup L(M_2)$ .

## 2.2 GRAMÁTICAS INDEPENDIENTES DEL CONTEXTO

Ahora que hemos extendido las máquinas en consideración de autómatas finitos a autómatas de pila, nuestra investigación se centra en cuáles son los lenguajes que estos autómatas extendidos pueden reconocer. Comenzamos la búsqueda de la respuesta regresando al concepto de las gramáticas.

### Definición de las gramáticas independientes del contexto

Para caracterizar los lenguajes que reconocen los autómatas de pila, presentamos el concepto de **gramática independiente del contexto**. A diferencia de las gramáticas regulares, estas gramáticas no tienen restricciones con respecto a la forma del lado derecho de sus reglas de reescritura, aunque aún se requiere que el lado izquierdo de cada regla sea un solo no terminal. La gramática de la figura 2.5 es una gramática independiente del contexto, pero no es regular.

El término “independiente del contexto” refleja que, como el lado izquierdo de cada regla de reescritura únicamente puede contener un solo no terminal, la regla puede aplicarse sin importar el contexto donde se encuentre dicho no terminal. Por el contrario, considere una regla de reescritura cuyo lado izquierdo contenga más de un no terminal, como  $xNy \rightarrow xzy$ . Esta regla

dice que el no terminal  $N$  puede sustituirse con el terminal  $z$  sólo cuando esté rodeado por los terminales  $x$  y  $y$ . Por lo tanto, la capacidad de eliminar  $N$  aplicando la regla dependerá del contexto en vez de ser independiente.

Al igual que las gramáticas regulares, las gramáticas independientes del contexto generan cadenas por medio de derivaciones. No obstante, en el caso de las gramáticas independientes del contexto pueden surgir dudas con respecto a cuál será el no terminal que deberá reemplazarse en un paso específico de la derivación. Por ejemplo, al generar una cadena con la gramática de la figura 2.5, el primer paso produce la cadena  $zM Nz$ , que presenta la opción de reemplazar el no terminal  $M$  o el  $N$  en el siguiente paso. Por consiguiente, para generar la cadena  $za z a^nb^mzb^nz$ , se podría producir la derivación

$$S \Rightarrow zMNz \Rightarrow zaMaNz \Rightarrow zazaNz \Rightarrow zazabNbz \Rightarrow zazabzbz$$

siguiendo la regla rutinaria de aplicar siempre una regla de reescritura al no terminal situado más a la izquierda en la cadena actual (esto se llama **derivación por la izquierda**). También podría producirse la derivación

$$S \Rightarrow zMNz \Rightarrow zMbNbz \Rightarrow zMbzbz \Rightarrow zaMabzbz \Rightarrow zazabzbz$$

aplicando siempre la regla de reescritura al no terminal situado más a la derecha, lo cual daría como resultado una **derivación por la derecha**. Incluso se podrían seguir otros patrones y obtener otras derivaciones de la misma cadena.

Lo cierto es que el orden en que se apliquen las reglas de reescritura no afecta la determinación de si una cadena puede generarse a partir de cierta gramática independiente del contexto. Esto resulta obvio cuando reconocemos que *si una cadena puede generarse a partir de alguna derivación, entonces puede ser generada por una derivación por la izquierda*. Para ver esto, primero consideramos el árbol de análisis sintáctico asociado a una derivación.

Un **árbol de análisis sintáctico** no es más que un árbol cuyos nodos representan terminales y no terminales de la gramática, donde el nodo raíz es el símbolo de inicio de la gramática y los hijos de cada nodo no terminal son los símbolos que reemplazan a ese no terminal en la derivación (ningún símbolo terminal puede ser un nodo interior del árbol ni ningún símbolo no terminal puede ser una hoja). En la figura 2.6 se presenta un árbol de análisis

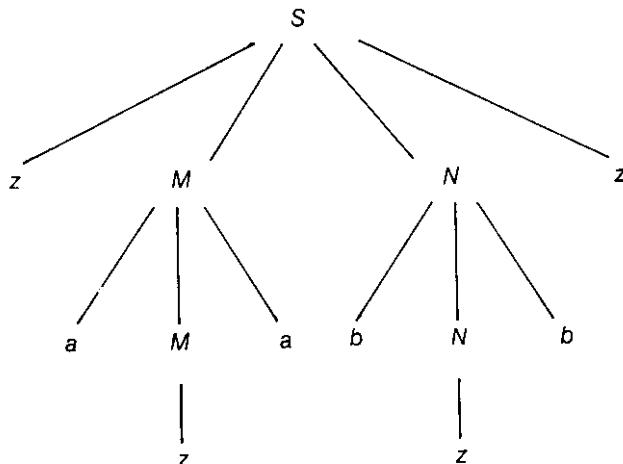
$$\begin{aligned} S &\rightarrow zMNz \\ M &\rightarrow aMa \\ M &\rightarrow z \\ N &\rightarrow bNb \\ N &\rightarrow z \end{aligned}$$

---

**Figura 2.5** Gramática independiente del contexto que genera cadenas de la forma  $za^nza^nb^mzb^nz$ , donde  $m, n \in \mathbb{N}$

sintáctico para la cadena  $zazabzbz$  usando la gramática de la figura 2.5 y cualquiera de las derivaciones anteriores.

Ahora, para ver que cualquier cadena generada por una gramática independiente del contexto se puede generar con una derivación por la izquierda, observamos que las derivaciones que corresponden al mismo árbol de análisis sintáctico únicamente difieren en cuanto al orden en que se aplican las reglas de reescritura. El orden de aplicación de las reglas sólo refleja el orden de construcción de las ramas del árbol. Una derivación por la izquierda corresponde a la construcción del árbol sintáctico comenzando por la rama izquierda, mientras que una derivación por la derecha corresponde a una construcción que se inicia por la rama derecha. Sin embargo, el orden de construcción de las ramas no afecta la estructura final del árbol, ya que cada rama es independiente de las demás. Por lo tanto, dada una derivación que no es por la izquierda, se puede construir el árbol de análisis sintáctico asociado y luego elaborar un derivación por la izquierda de la misma cadena, aplicando una "evaluación por la izquierda" sistemática del árbol.



**Figura 2.6** Árbol de análisis sintáctico para la cadena  $zazabzbz$  utilizando la gramática de la figura 2.5

$$\begin{aligned} S &\rightarrow xSy \\ S &\rightarrow \lambda \end{aligned}$$

**Figura 2.7** Gramática independiente del contexto que genera cadenas de la forma  $x^n y^n$  donde  $n \in \mathbb{N}$

Finalmente, observamos que la flexibilidad que ofrecen las gramáticas independientes del contexto permite la construcción de una gramática que genera el lenguaje  $\{x^n y^n : n \in \mathbb{N}\}$ , que se presenta en la figura 2.7. Este ejemplo, combinado con el hecho de que cualquier gramática regular es una gramática independiente del contexto, nos permite llegar a la conclusión de que las gramáticas independientes del contexto generan una mayor colección de lenguajes que las gramáticas regulares. Los lenguajes generados por gramáticas independientes del contexto se denominan **lenguajes independientes del contexto**.

### Gramáticas independientes del contexto y autómatas de pila

Antes de considerar la relación entre las gramáticas independientes del contexto y los autómatas de pila, debemos esclarecer un aspecto de la notación. En los análisis que se presentan a continuación, será conveniente considerar transiciones únicas que insertan más de un símbolo en la pila, como  $(p, a, s; q, xyz)$ . En este caso, se insertan en la pila los símbolos  $z$ ,  $y$  y  $x$  (en ese orden). Así, después de efectuar la transición,  $x$  se hallará en la cima de la pila (con  $y$  debajo y  $z$  en el fondo). Observe que las transiciones de este tipo sólo representan una forma conveniente de notación y no añaden capacidades adicionales a la máquina. De hecho, la transición de inserción múltiple  $(p, a, s; q, xyz)$  podría simularse con la secuencia de transiciones tradicionales  $(p, a, s; q_1, z), (q_1, \lambda, \lambda; q_2, y)$  y  $(q_2, \lambda, \lambda; q, x)$ , donde  $q_1$  y  $q_2$  son estados adicionales a los que no puede llegar ninguna otra secuencia de transiciones.

Ahora debemos mostrar que los *lenguajes generados por gramáticas independientes del contexto son exactamente los mismos lenguajes que aceptan los autómatas de pila*. Esto se hará en dos etapas. Primero mostramos que para cualquier gramática  $G$  independiente del contexto existe un autómata de pila  $M$  tal que  $L(M) = L(G)$  (Teorema 2.2). Luego mostramos que para cualquier autómata de pila  $M$  existe una gramática  $G$  independiente del contexto tal que  $L(G) = L(M)$  (Teorema 2.3).

#### TEOREMA 2.2

Para cada gramática  $G$  independiente del contexto, existe un autómata de pila  $M$  tal que  $L(G) = L(M)$ .

#### DEMOSTRACIÓN

Dada una gramática  $G$  independiente del contexto, construimos un autómata de pila  $M$  de la manera siguiente:

1. Designe el alfabeto de  $M$  como los símbolos terminales de  $G$ , y los símbolos de pila de  $M$  como los símbolos terminales y no terminales de  $G$ , junto con el símbolo especial  $\#$  (podemos suponer que  $\#$  no es un símbolo terminal o no terminal de  $G$ ).

2. Designe los estados de  $M$  como  $i, p, q$  y  $f$ , donde  $i$  es el estado inicial y  $f$  es el único estado de aceptación.
3. Introduzca la transición  $(i, \lambda, \lambda; p, \#)$ .
4. Introduzca una transición  $(p, \lambda, \lambda; q, S)$  donde  $S$  es el símbolo inicial de  $G$ .
5. Introduzca una transición de la forma  $(q, \lambda, N; q, w)$  para cada regla de reescritura  $N \rightarrow w$  en  $G$  (aquí empleamos nuestra nueva convención que permite que una sola transición inserte más de un símbolo de pila. Específicamente,  $w$  puede ser una cadena de cero o más símbolos, incluyendo terminales y no terminales).
6. Introduzca una transición de la forma  $(q, x, x; q, \lambda)$  para cada terminal  $x$  de  $G$  (es decir, para cada símbolo del alfabeto de  $M$ ).
7. Introduzca la transición  $(q, \lambda, \#; f, \lambda)$ .

Un autómata de pila construido de esta manera analizará una cadena de entrada marcando primero el fondo de la pila con el símbolo  $\#$ , luego insertando en la pila el símbolo inicial de la gramática y después entrando al estado  $q$ . De ahí y hasta que el símbolo  $\#$  vuelve a aparecer en la cima de la pila, el autómata extraerá un no terminal de la pila y lo reemplazará con el lado derecho de una regla de reescritura aplicable, o extraerá un terminal de la pila a la vez que lee el mismo terminal en la entrada. Una vez que el símbolo  $\#$  regresa a la cima de la pila, el autómata cambiará a su estado de aceptación  $f$ , indicando que la entrada recibida hasta ese punto es aceptable.

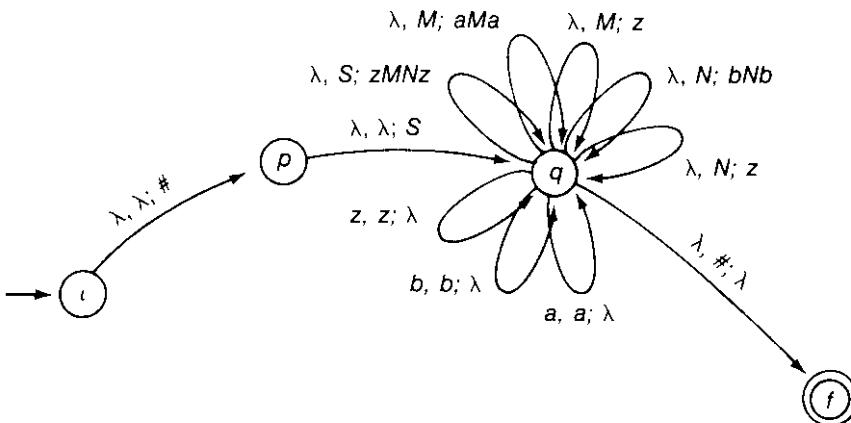
Observe que la cadena de símbolos que integran la parte derecha de una regla de reescritura se inserta en la pila de derecha a izquierda. Así, el no terminal situado mas a la izquierda será el primero en surgir en la cima de la pila; por tanto, también será el primer no terminal de la pila que se reemplazará. Por consiguiente, el autómata analiza su entrada efectuando una derivación por la izquierda de acuerdo con las reglas de la gramática en la cual se basa. Sin embargo, como ya vimos, las cadenas generadas por una gramática independiente del contexto son exactamente aquellas que tienen una derivación por la izquierda. Entonces, el autómata acepta exactamente el mismo lenguaje que genera la gramática.



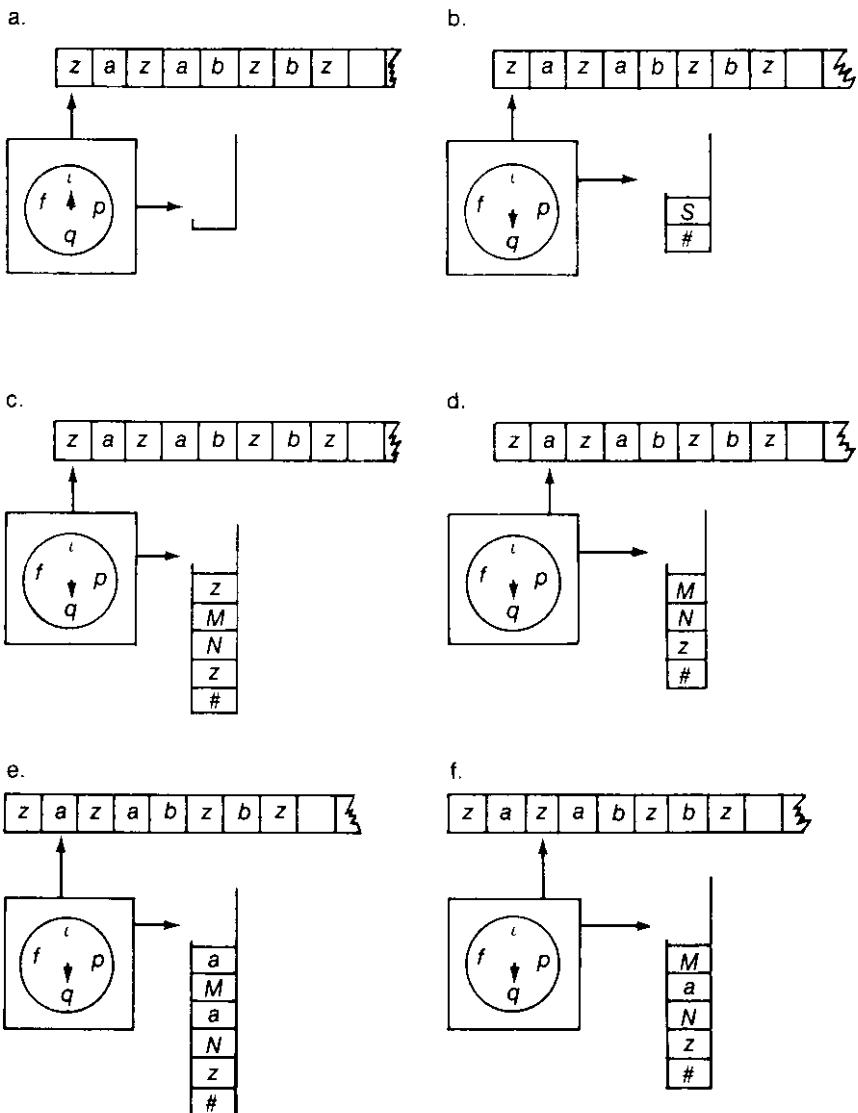
Quizás con un ejemplo pueda comprenderse mejor la función de las distintas transiciones construidas en la demostración del teorema 2.2. Consideremos los pasos que ejecuta el autómata de pila descrito en el diagrama de transiciones de la figura 2.8, construido a partir de la gramática independiente del contexto de la figura 2.5. Para evaluar la cadena  $zazabzbz$ , iniciamos la máquina con la configuración representada en la figura 2.9a. A partir de esta

configuración, la máquina marca el fondo de la pila con el símbolo # y cambia al estado  $q$  a la vez que inserta el no terminal  $S$  en la pila, para llegar a la configuración representada en la figura 2.9 b. A partir de este punto, la pila se emplea para contener una descripción de la estructura que la máquina espera encontrar en la parte restante de su flujo de entrada. Así, el símbolo  $S$  que se encuentra actualmente en la parte superior de la pila indica que la máquina espera que los símbolos restantes de su entrada constituyan una estructura que pueda generarse a partir del no terminal  $S$ .

Sin embargo, puesto que  $S$  no es un terminal, la máquina no puede esperar que el contenido de su pila aparezca explícitamente en la entrada, por lo que hay que reemplazar el no terminal antes de que la máquina intente comparar su pila directamente con el flujo de entrada. De hecho, se trata de una regla general que sigue la máquina: en cualquier instante, la cima de la pila contiene un no terminal que debe reemplazarse con una descripción equivalente, aunque más detallada, de la estructura. Este es el propósito de las transiciones incluidas en la regla 5 de la demostración del teorema 2.2. Con su ejecución se reemplaza un no terminal de la cima de la pila por una descripción más detallada de la estructura, de acuerdo con alguna regla de reescritura de la gramática original. Así, la máquina de nuestro ejemplo continúa con la ejecución de la transición  $(q, \lambda, S; q, zMNz)$  para llegar a la configuración representada en la figura 2.9c.



**Figura 2.8** Diagrama de transiciones de un autómata de pila construido a partir de la gramática de la figura 2.5 utilizando las técnicas presentadas en la demostración del teorema 2.2



**Figura 2.9** Autómata de pila en funcionamiento

Ahora la cima de la pila de la máquina contiene el terminal  $z$  y, por tanto, puede compararse con la cadena de entrada a través de la transición  $(q, z, z; q, \lambda)$ . Después de ejecutar esta transición, la pila de la máquina contiene  $MNz$  y los símbolos restantes de la cadena de entrada son  $azabzbz$ , como se muestra en la figura 2.9d.

De nuevo, la cima de la pila contiene un no terminal que la máquina debe reemplazar. Sin embargo, a diferencia de la sustitución anterior del no terminal  $S$ , existen dos reglas de reescritura que pueden aplicarse para sustituir el no terminal  $M$ . El autómata de pila podría ejecutar la transición  $(q, \lambda, M; q, z)$ , lo que ocasionaría que la máquina fallara en su intento por aceptar la cadena, o podría ejecutar  $(q, \lambda, M; q, aMa)$ , que en este caso sería la opción correcta (este autómata es no determinista). Supongamos que se elige la opción correcta ya que nuestro objetivo es observar cómo la máquina podría aceptar la cadena de entrada (recuerde que una cadena se encuentra en el lenguaje de una máquina no determinista si es *possible* que la máquina acepte la cadena). Después de ejecutar  $(q, \lambda, M; q, aMa)$  la máquina aparece como se muestra en la figura 2.9e, con el terminal  $a$  en la cima de la pila. Entonces, este terminal se compara con la cadena de entrada a través de la transición  $(q, a, a; q, \lambda)$ , lo que deja la pila con  $MaNz$  y los símbolos  $zabzbz$  por leerse de la cadena de entrada, como se ilustra en la figura 2.9f.

En la figura 2.10 se resumen las actividades restantes de la máquina. Observe que la lectura del último símbolo de la entrada descubre la marca  $\#$  en la pila. Entonces, la transición  $(q, \lambda, \#; f, \lambda)$  transfiere la máquina al estado de aceptación  $f$ , donde se declara la aceptación de la entrada.

Preparémonos ahora para el teorema 2.3. Suponga que se nos da un autómata de pila que acepta cadenas sólo cuando su pila está vacía. Entonces, desde el punto de vista del autómata, su tarea es tratar de pasar de su estado inicial a uno de aceptación de manera que su pila siempre se encuentre en la misma condición que al comenzar los cálculos. La forma cómo se logre este objetivo dependerá de las transiciones disponibles. Si, por ejemplo,  $i$  es el estado inicial de la máquina y  $(i, \lambda, \lambda; p, \#)$  es una de las transiciones disponibles, entonces el autómata puede tratar de lograr su objetivo ejecutando primero esta transición. Esto daría como resultado que la máquina se encontrara en el estado  $p$  con el símbolo  $\#$  en su pila, y entonces la meta de la máquina sería pasar del estado  $p$  a un estado de aceptación a la vez que eliminaría de la pila el símbolo  $\#$ .

Por lo general, este objetivo se representa con  $\langle p, x, q \rangle$ , donde  $p$  y  $q$  son estados y  $x$  es un símbolo de pila de la máquina. Es decir,  $\langle p, x, q \rangle$  representa el deseo de pasar del estado  $p$  al estado  $q$  de manera que el símbolo  $x$  se elimine de la parte superior de la pila. Se especifica un caso algo especial por medio de  $\langle p, \lambda, q \rangle$ , que representa el objetivo de pasar del estado  $p$  al estado  $q$  de modo que la pila no se altere. En otras palabras, al llegar al estado  $q$ , la pila será la misma que existía cuando se encontraba en el estado  $p$ . Un ejemplo de esta situación es el objetivo original, antes mencionado, de pasar del estado inicial a uno de aceptación. De hecho, para cada estado de aceptación  $f$ , el autómata

Contenido de la pila	Resto de la entrada	Transición ejecutada
$\lambda$	$zazabzbz$	$(i, \lambda, \lambda; p, \#)$
#	$zazabzbz$	$(p, \lambda, \lambda; q, S)$
$S\#$	$zazabzbz$	$(q, \lambda, S; q, zMNz)$
$zMNz\#$	$zazabzbz$	$(q, z, z; q, \lambda)$
$MNz\#$	$azabzbz$	$(q, \lambda, M; q, aMa)$
$aMaNz\#$	$azabzbz$	$(q, a, a; q, \lambda)$
$MaNz\#$	$zabzbz$	$(q, \lambda, M; q, z)$
$zaNz\#$	$zabzbz$	$(q, z, z; q, \lambda)$
$aNz\#$	$abzbz$	$(q, a, a; q, \lambda)$
$Nz\#$	$bzbz$	$(q, \lambda, N; q, bNb)$
$bNbz\#$	$bzbz$	$(q, b, b; q, \lambda)$
$Nbz\#$	$z bz$	$(q, \lambda, N; q, z)$
$zbz\#$	$z bz$	$(q, z, z; q, \lambda)$
$bz\#$	$bz$	$(q, b, b; q, \lambda)$
$z\#$	$z$	$(q, z, z; p, \lambda)$
#	$\lambda$	$(q, \lambda, \#; f, \lambda)$

**Figura 2.10** Análisis completo de la cadena  $zazabzbz$  que efectúa el autómata de pila descrito en la figura 2.8

tiene un objetivo principal representado por  $\langle i, \lambda, f \rangle$ , donde  $i$  es el estado inicial de la máquina.

Ahora estamos listos para mostrar que los lenguajes aceptados por un autómata de pila son independientes del contexto.

### TEOREMA 2.3

Para cada autómata de pila  $M$ , existe una gramática  $G$  independiente del contexto tal que  $L(M) = L(G)$ .

#### DEMOSTRACIÓN

Dado un autómata de pila  $M$ , nuestra tarea es producir una gramática  $G$  independiente del contexto que genere el lenguaje  $L(M)$ . Como resultado del teorema 2.1, podemos suponer que el autómata de pila  $M$  acepta cadenas únicamente cuando su pila está vacía. Una vez hecha esta observación, construimos  $G$  de tal manera que sus no terminales representen los objetivos de  $M$ , como se mencionó antes, y que sus reglas de reescritura representen refinamientos de objetivos mayores, presentados en términos de objetivos más pequeños.

Para ser más precisos, especificamos que los no terminales de  $G$  consisten en el símbolo de inicio  $S$  más todos los objetivos de  $M$ , es decir, todas las estructuras sintácticas de la forma  $\langle p, x, q \rangle$  donde

$p$  y  $q$  son estados de  $M$  y  $x$  es  $\lambda$  o un símbolo de pila de  $M$  (así,  $G$  puede contener un gran número de no terminales incluso si  $M$  es un autómata pequeño). Los terminales de  $G$  son los símbolos del alfabeto de  $M$ .

Ahora estamos listos para introducir la primera colección de reglas de reescritura de  $G$ . Estas reglas se obtienen de la siguiente manera:

1. Para cada estado de aceptación  $f$  de  $M$ , forme la regla de reescritura  $S \rightarrow <1, \lambda, f>$ , donde  $1$  es el estado inicial de  $M$ .

Las reglas de reescritura obtenidas en el paso 1 aseguran que cualquier derivación que utilice esta gramática comenzará sustituyendo el símbolo inicial de la gramática por un objetivo principal del autómata.

Se obtiene otra colección de reglas de reescritura con

2. Para cada estado  $p$  en  $M$ , forme la regla de reescritura  $<p, \lambda, p> \rightarrow \lambda$ .

Las reglas obtenidas en el paso 2 son reflejo de que puede eliminarse el objetivo de pasar de un estado a sí mismo sin modificar la pila.

Cada una de las demás reglas de reescritura de  $G$  se construye a partir de una transición en  $M$ , siguiendo el paso 3 o el paso 4 descritos ahora.

3. Para cada transición  $(p, x, y; q, z)$  de  $M$  (donde  $y$  no es  $\lambda$ ), genere una regla de reescritura  $<p, y, r> \rightarrow x <q, z, r>$  para cada estado  $r$  de  $M$ .

Las reglas generadas por el paso 3 indican que el objetivo de pasar de un estado  $p$  a un estado  $r$  eliminando  $y$  de la pila puede lograrse si se pasa primero a un estado  $q$  mientras que se lee  $x$  de la entrada y se intercambia  $z$  por  $y$  en la pila (usando la transición  $(p, x, y; q, z)$ ) y luego intentando pasar del estado  $q$  al estado  $r$  a la vez que se elimina  $z$  de la pila.

4. Para cada transición de la forma  $(p, x, \lambda; q, z)$ , genere todas las reglas de reescritura de la forma  $<p, w, r> \rightarrow x <q, z, k> <k, w, r>$ , donde  $w$  es un símbolo de pila o  $\lambda$ , mientras que  $k$  y  $r$  (que pueden ser iguales) son estados de  $M$ .

Las reglas de reescritura construidas en el paso 4 reflejan que el objetivo de pasar de un estado  $p$  a un estado  $r$  a la vez que se elimina  $w$  de la pila puede lograrse si primero se pasa al estado  $q$  mientras se lee  $x$  de la entrada y se inserta  $z$  en la pila (por medio de la transición  $(p, x, \lambda; q, z)$ ) y luego se intenta pasar del estado  $q$  al estado  $r$  a través de un estado  $k$ , a la vez que se eliminan  $z$  y  $w$  de la pila.

Observe que las reglas de reescritura construidas en los pasos 1 a 4 forman una gramática independiente del contexto. Sólo resta mostrar que esta gramática genera el mismo lenguaje que acepta el autómata: debemos mostrar que este último puede aceptar cualquier cadena generada por la gramática y que la gramática puede generar cualquier cadena que acepte el autómata. Ambos enunciados serán verdaderos si demostramos la siguiente afirmación.

*Aplicando reglas de la gramática, se puede reescribir un no terminal de la forma  $\langle p, \alpha, q \rangle$  (donde  $\alpha$  es  $\lambda$  o un símbolo de pila) como una cadena de terminales  $w$ , si y sólo si el autómata puede pasar de un estado  $p$  a un estado  $q$  siguiendo una ruta cuyo recorrido da como resultado que se elimine  $\alpha$  de la pila y que se lea la cadena  $w$  de la cinta de la máquina.*

Comencemos por demostrar la parte “sólo si” de esta afirmación, aplicando la inducción en el número de pasos requeridos para reescribir  $\langle p, \alpha, q \rangle$  como  $w$ . Si se requiere sólo un paso, entonces el no terminal  $\langle p, \alpha, q \rangle$  debe ser en realidad  $\langle p, \lambda, q \rangle$  ya que ésta es la única forma de un no terminal que puede reescribirse como cadena terminal en un solo paso. A su vez, esto significa que  $w$  debe ser  $\lambda$ . Así, nuestra afirmación es verdadera para el caso básico, ya que el autómata siempre puede pasar de un estado  $p$  a  $p$  sin eliminar nada de la pila ni leer nada de la cinta.

Ahora, supongamos que la ruta deseada en el autómata existe para cualquier caso donde, en  $n$  o menos pasos, el no terminal que representa un objetivo puede reescribirse como una cadena de terminales, y suponga que en  $n+1$  pasos puede reescribirse  $\langle p, \alpha, q \rangle$  como la cadena de terminales  $w$ . Observe que el primer paso de este proceso debe ser la aplicación de una regla de reescritura del paso 3 o del paso 4 anteriores. Supongamos que se obtiene del paso 3, con la transición  $(p, w_1, \alpha; r, \beta)$ . (El caso del paso 4 no es más que una leve generalización de este caso.) El primer paso del proceso de reescritura aparece como  $\langle p, \alpha, q \rangle \Rightarrow w_1 \langle r, \beta, q \rangle$ . Esto significa que el resto del proceso de reescritura convierte  $\langle r, \beta, q \rangle$  en una cadena  $w_2$ , tal que  $w = w_1 w_2$ , en sólo  $n$  pasos. Entonces, por nuestra hipótesis de inducción, el autómata puede pasar del estado  $r$  al estado  $q$  leyendo  $w_2$  a la vez que elimina  $\beta$  de su pila. Si hacemos que la transición  $(p, w_1, \alpha; r, \beta)$  anteceda a este cálculo, obtenemos un proceso que comienza en el estado  $p$  y pasa al estado  $q$  a la vez que lee de la cinta la cadena  $w$  y elimina  $\alpha$  de la pila. Por lo tanto, podemos concluir que la parte “sólo si” de nuestra afirmación es verdadera.

Ahora, consideraremos la parte “si” de nuestra afirmación. Una vez más aplicamos la inducción, pero esta vez sobre la longitud de la ruta de  $p$  a  $q$ . Si la longitud de esta ruta es cero, la ruta no

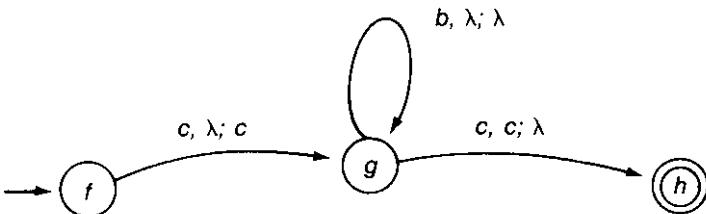
requiere transiciones y  $q$  debe ser igual a  $p$ . Por lo tanto, basta con la regla de reescritura  $\langle p, \alpha, p \rangle \rightarrow \lambda$ .

Luego suponemos que si el autómata puede pasar de cualquier estado  $r$  a un estado  $s$ , siguiendo una ruta que consiste en no más de  $n$  transiciones y que da como resultado la lectura de la cadena  $v$  de la cinta y la eliminación de  $\alpha$  de la pila (donde  $\alpha$  es  $\lambda$  o un símbolo de pila), entonces hay en la gramática reglas de reescritura que permiten reescribir  $\langle p, \alpha, q \rangle$  como  $w$ . Consideré una ruta de  $n + 1$  pasos del estado  $p$  a  $q$  cuyo recorrido da como resultado la lectura de cinta de la cadena  $w$  y la eliminación de  $\alpha$  de la pila (donde  $\alpha$  es  $\lambda$  o un símbolo de pila). Supongamos que el primer paso de esta ruta es la ejecución de la transición  $(p, w_1, \alpha; t, z)$ , donde  $\alpha \neq \lambda$  (los otros primeros pasos posibles se manejan de manera semejante). Entonces, la porción restante de la ruta debe pasar del estado  $t$  al estado  $q$  en sólo  $n$  transiciones, sin eliminar nada de la pila al leer de la cinta la cadena  $w_2$ , donde  $w = w_1w_2$ . Sin embargo, por nuestra hipótesis de inducción, esto significa que deben existir en la gramática reglas de reescritura que permitan reescribir  $\langle t, \lambda, q \rangle$  como  $w_2$ . Además, la existencia de la transición  $(p, w_1, \alpha; t, z)$  implica la existencia de la regla  $\langle p, \alpha, q \rangle \rightarrow w_1 \langle t, \lambda, q \rangle$ . Por lo tanto, el no terminal  $\langle p, \alpha, q \rangle$  puede reescribirse como  $w$  aplicando primero esta regla y luego reescribiendo  $\langle t, \lambda, q \rangle$  como  $w_2$ .

Concluimos que las partes "si" y "sólo si" de nuestra afirmación deben ser verdaderas, y por lo tanto la gramática independiente del contexto construida a partir de los pasos 1, 2, 3 y 4 debe generar el mismo lenguaje que acepta el autómata de pila  $M$ .



De nuevo, un ejemplo debe ayudar a esclarecer varios de los puntos de la demostración anterior. Consideré la construcción de una gramática independiente del contexto que acepte el lenguaje  $\{cb^n c : n \in \mathbb{N}^*\}$  del autómata de pila de la figura 2.11. Nuestra primera observación es que en la gramática resultante existen numerosos no terminales, incluyendo el símbolo de inicio  $S$  más un no terminal de la forma  $\langle p, x, q \rangle$  para cada tripleta  $(p, x, q)$ , donde  $x$  es  $\lambda$  o  $c$  (puesto que  $c$  es el único símbolo de pila), y  $p$  y  $q$  (que pueden ser iguales) son estados de la máquina. Además como se muestra en la figura 2.12 se forma un gran número de reglas de reescritura, que se presentan, junto con sus transiciones asociadas. Por último, en la figura 2.13 se muestra una derivación de la cadena  $cbbc$ . Aquí vemos que las reglas de reescritura asociadas a las transiciones que leen símbolos de la entrada de la máquina introducen estos mismos símbolos en la cadena que se deriva. Así, una vez que el proceso de derivación ha eliminado todos los no terminales, los símbolos en la cadena restante son exactamente los mismos que habría leído el cálculo correspondiente del autómata de pila.



**Figura 2.11** Diagrama de transiciones para un autómata de pila que acepta  $\{cb^n c : n \in \mathbb{N}^*\}$

En resumen, contamos ahora con dos caracterizaciones para los lenguajes independientes del contexto: son los lenguajes aceptados por autómatas de pila, así como los lenguajes generados por gramáticas independientes del contexto.

### Forma normal de Chomsky

Una de las ventajas de clasificar los lenguajes en función de su gramática es que estas clasificaciones proporcionan detalles con respecto a las estructuras de cadenas que pueden aparecer en los lenguajes correspondientes. Sin embargo, a primera vista parece que la flexibilidad que permiten las gramáticas independientes del contexto impone pocas restricciones a las posibles estructuras de cadenas que pueden encontrarse en los lenguajes independientes del contexto. Por esto, quizás le sorprenda saber que los lenguajes independientes del contexto sí tienen gramáticas cuyas reglas de reescritura se adhieren a formatos extremadamente rígidos. Por lo tanto, investigaremos con mayor detenimiento la estructura de las reglas de reescritura que se encuentran en las gramáticas independientes del contexto.

Comenzamos nuestra investigación considerando la necesidad de tener reglas  $\lambda$  en una gramática independiente del contexto. Si el lenguaje generado por la gramática contiene la cadena vacía, entonces debe aparecer cuando menos una regla  $\lambda$  en la gramática; de lo contrario, no habría manera de derivar la cadena vacía del símbolo inicial de la gramática. Sin embargo, podríamos preguntar hasta qué punto se requieren las reglas  $\lambda$  si el lenguaje que se genera no contiene la cadena vacía.

Para responder esta pregunta, démonos cuenta primero que la existencia de una regla  $\lambda$  puede permitir que más no terminales que el que aparece en la regla puedan reescribirse como la cadena vacía. Por ejemplo, si una gramática contiene las reglas  $N \rightarrow \lambda$  y  $M \rightarrow N$ , entonces tanto  $M$  como  $N$  se podrían reescribir como  $\lambda$ , aunque la regla  $M \rightarrow \lambda$  no se encuentre en la gramática. Para identificar el efecto de las reglas  $\lambda$  en una gramática independiente del

contexto, debemos aislar todos los no terminales que pueden reescribirse como la cadena vacía. Para esto, definimos un encuadernamiento  $\lambda$  de longitud  $n$  como la secuencia de reglas de la forma  $N_n \rightarrow N_{n-1}, N_{n-1} \rightarrow N_{n-2}, \dots, N_0 \rightarrow \lambda$ , y definimos el no terminal  $N_n$  como el origen del encuadernamiento.

### Del paso 1

$S \rightarrow \langle f, \lambda, h \rangle$

### Del paso 2

$\langle f, \lambda, f \rangle \rightarrow \lambda$   
 $\langle g, \lambda, g \rangle \rightarrow \lambda$   
 $\langle h, \lambda, h \rangle \rightarrow \lambda$

### Del paso 3, transición ( $g, c, c; h, \lambda$ )

$\langle g, c, f \rangle \rightarrow c \langle h, \lambda, f \rangle$   
 $\langle g, c, g \rangle \rightarrow c \langle h, \lambda, g \rangle$   
 $\langle g, c, h \rangle \rightarrow c \langle h, \lambda, h \rangle$

### Del paso 4, transición ( $f, c, \lambda; g, c$ )

$\langle f, \lambda, f \rangle \rightarrow c \langle g, c, f \rangle$	$\langle f, c, f \rangle \rightarrow c \langle g, c, f \rangle$	$\langle f, c, f \rangle$
$\langle f, \lambda, f \rangle \rightarrow c \langle g, c, g \rangle$	$\langle f, c, f \rangle \rightarrow c \langle g, c, g \rangle$	$\langle g, c, f \rangle$
$\langle f, \lambda, f \rangle \rightarrow c \langle g, c, h \rangle$	$\langle f, c, f \rangle \rightarrow c \langle g, c, h \rangle$	$\langle h, c, f \rangle$
$\langle f, \lambda, g \rangle \rightarrow c \langle g, c, f \rangle$	$\langle f, c, g \rangle \rightarrow c \langle g, c, f \rangle$	$\langle f, c, g \rangle$
$\langle f, \lambda, g \rangle \rightarrow c \langle g, c, g \rangle$	$\langle f, c, g \rangle \rightarrow c \langle g, c, g \rangle$	$\langle g, c, g \rangle$
$\langle f, \lambda, g \rangle \rightarrow c \langle g, c, h \rangle$	$\langle f, c, g \rangle \rightarrow c \langle g, c, h \rangle$	$\langle h, c, g \rangle$
$\langle f, \lambda, h \rangle \rightarrow c \langle g, c, f \rangle$	$\langle f, c, h \rangle \rightarrow c \langle g, c, f \rangle$	$\langle f, c, h \rangle$
$\langle f, \lambda, h \rangle \rightarrow c \langle g, c, g \rangle$	$\langle f, c, h \rangle \rightarrow c \langle g, c, g \rangle$	$\langle g, c, h \rangle$
$\langle f, \lambda, h \rangle \rightarrow c \langle g, c, h \rangle$	$\langle f, c, h \rangle \rightarrow c \langle g, c, h \rangle$	$\langle h, c, h \rangle$

\*

### Del paso 4, transición ( $g, b, l; g, \lambda$ )

$\langle g, \lambda, f \rangle \rightarrow b \langle g, \lambda, f \rangle$	$\langle g, c, f \rangle \rightarrow b \langle g, \lambda, f \rangle$	$\langle f, c, f \rangle$
$\langle g, \lambda, f \rangle \rightarrow b \langle g, \lambda, g \rangle$	$\langle g, c, f \rangle \rightarrow b \langle g, \lambda, g \rangle$	$\langle g, c, f \rangle$
$\langle g, \lambda, f \rangle \rightarrow b \langle g, \lambda, h \rangle$	$\langle g, c, f \rangle \rightarrow b \langle g, \lambda, h \rangle$	$\langle h, c, f \rangle$
$\langle g, \lambda, g \rangle \rightarrow b \langle g, \lambda, f \rangle$	$\langle g, c, g \rangle \rightarrow b \langle g, \lambda, f \rangle$	$\langle f, c, g \rangle$
$\langle g, \lambda, g \rangle \rightarrow b \langle g, \lambda, g \rangle$	$\langle g, c, g \rangle \rightarrow b \langle g, \lambda, g \rangle$	$\langle g, c, g \rangle$
$\langle g, \lambda, g \rangle \rightarrow b \langle g, \lambda, h \rangle$	$\langle g, c, g \rangle \rightarrow b \langle g, \lambda, h \rangle$	$\langle h, c, g \rangle$
$\langle g, \lambda, h \rangle \rightarrow b \langle g, \lambda, f \rangle$	$\langle g, c, h \rangle \rightarrow b \langle g, \lambda, f \rangle$	$\langle f, c, h \rangle$
$\langle g, \lambda, h \rangle \rightarrow b \langle g, \lambda, g \rangle$	$\langle g, c, h \rangle \rightarrow b \langle g, \lambda, g \rangle$	$\langle g, c, h \rangle$
$\langle g, \lambda, h \rangle \rightarrow b \langle g, \lambda, h \rangle$	$\langle g, c, h \rangle \rightarrow b \langle g, \lambda, h \rangle$	$\langle h, c, h \rangle$

**Figura 2.12** Reglas de reescritura obtenidas a partir del autómata de pila de la figura 2.11

$$\begin{aligned}
 S &\Rightarrow \langle f, \lambda, h \rangle \\
 &\Rightarrow c \langle g, c, h \rangle \langle h, \lambda, h \rangle \\
 &\Rightarrow cb \langle g, \lambda, g \rangle \langle g, c, h \rangle \langle h, \lambda, h \rangle \\
 &\Rightarrow cb \langle g, c, h \rangle \langle h, \lambda, h \rangle \\
 &\Rightarrow cbb \langle g, \lambda, g \rangle \langle g, c, h \rangle \langle h, \lambda, h \rangle \\
 &\Rightarrow cbb \langle g, c, h \rangle \langle h, \lambda, h \rangle \\
 &\Rightarrow cbcc \langle h, \lambda, h \rangle \langle h, \lambda, h \rangle \\
 &\Rightarrow cbcc \langle h, \lambda, h \rangle \\
 &\Rightarrow cbcc
 \end{aligned}$$

**Figura 2.13** Derivación de  $cbcc$  utilizando las reglas de reescritura de la figura 2.12

Ahora, si  $G$  es una gramática independiente del contexto que no genera la cadena vacía, definimos  $U_0$  como el conjunto de los no terminales que aparecen como origen de los encadenamientos  $\lambda$  de longitud cero, o sea, los no terminales que aparecen del lado izquierdo de las reglas  $\lambda$ . A este conjunto le añadimos los orígenes de todos los encadenamientos  $\lambda$  de longitud uno para formar otro conjunto,  $U_1$ . Luego, a  $U_1$  le agregamos los orígenes de todos los encadenamientos  $\lambda$  de longitud dos para obtener un conjunto llamado  $U_2$ , etcétera (si  $G$  fuera la gramática en la figura 2.14a, entonces  $U_0$  sería  $\{Q\}$  y  $U_1$  sería  $\{P, Q\}$ ).

Puesto que en una gramática sólo hay un número finito de no terminales, debe existir un punto en el cual este proceso deje de introducir no terminales adicionales. Al llegar a este punto hemos recopilado todos los no terminales de  $G$  que pueden reescribirse como la cadena vacía; este conjunto se representa con  $U$  (observe que, como  $G$  no genera la cadena vacía,  $U$  no puede contener el símbolo inicial de  $G$ ).

Ahora, para cada regla de reescritura de la forma  $N \rightarrow w$ , donde  $w$  es una cadena de terminales y no terminales, agregamos a  $G$  todas las reglas de la forma  $N \rightarrow w'$ , donde  $w'$  es cualquier cadena *no vacía* obtenida al eliminar de  $w$  una o más ocurrencias de no terminales en  $U$ . (Una vez más, haciendo referencia a la figura 2.14a, el conjunto  $U$  sería  $\{P, Q\}$  por lo que agregamos a la gramática las reglas siguientes:

$$\left. \begin{array}{l} S \rightarrow zPzz \\ S \rightarrow zzQz \\ S \rightarrow zzz \end{array} \right\} \text{a partir de } S \rightarrow zPzQz$$

$$\begin{array}{ll} P \rightarrow xx & \text{a partir de } P \rightarrow xPx \\ Q \rightarrow yy & \text{a partir de } Q \rightarrow yPy \end{array}$$

a.	$S \rightarrow zPzQz$	b.	$S \rightarrow zPzQz$
	$P \rightarrow xPx$		$S \rightarrow zzQz$
	$P \rightarrow Q$		$S \rightarrow zPzz$
	$Q \rightarrow yPy$		$S \rightarrow zzz$
	$Q \rightarrow$		$P \rightarrow zPx$
			$P \rightarrow xx$
			$P \rightarrow Q$
			$Q \rightarrow yPz$
			$Q \rightarrow yy$

**Figura 2.14** Una gramática independiente del contexto (que no genera la cadena vacía) y otra gramática que genera el mismo lenguaje sin utilizar reglas  $\lambda$

Observe que no agregamos la regla  $P \rightarrow \lambda$  que se obtendría de la regla  $P \rightarrow Q$ . (Véase Fig. 2.14b.)

Una vez que agregamos estas nuevas reglas a la gramática, ya no necesitamos las reglas  $\lambda$ . Suponga que la derivación de una cadena, empleando la gramática original, requiere la aplicación de una regla  $\lambda$ , y sea  $N$  el origen del encadenamientos  $\lambda$  más largo que aparece en la derivación que termina con esta regla  $\lambda$ . Entonces, la ocurrencia de  $N$  debe introducirse en la derivación aplicando alguna regla de la forma  $M \rightarrow w_1Nw_2$ , donde  $w_1$  y  $w_2$  son cadenas de terminales y no terminales, una de las cuales debe ser no vacía (se escogió  $N$  como el origen del encadenamiento  $\lambda$  que da fin a la regla  $\lambda$ ; además,  $N$  no es el símbolo inicial, ya que  $N \in U$  y  $S \in U$ ). Esto significa que la regla  $M \rightarrow w_1w_2$  es una de las reglas que hemos agregado a la gramática. Así, podemos eliminar de la derivación el empleo de la regla  $\lambda$  si aplicamos la regla  $M \rightarrow w_1w_2$  en vez de  $M \rightarrow w_1Nw_2$ . De esta manera podemos eliminar cualquier utilización de reglas  $\lambda$  en una derivación. Además, podemos deshacernos de todas las reglas  $\lambda$  de la gramática sin reducir sus poderes generativos.

Como ejemplo, considere la derivación de la figura 2.15a, basada en la gramática de la figura 2.14a. El último paso de la derivación utiliza la regla  $\lambda$ ,  $Q \rightarrow \lambda$ . El origen del encadenamientos  $\lambda$  que condujo al uso de esta regla es el no terminal  $Q$  introducido en el primer paso de la derivación. Por lo tanto, podemos cambiar este paso utilizando la nueva regla  $S \rightarrow zPzz$  para obtener la derivación de la figura 2.15b. Esta derivación usa la regla  $\lambda$ ,  $Q \rightarrow \lambda$ , en su último paso. El no terminal  $P$  introducido en el segundo paso es el origen del encadenamiento  $\lambda$  que lleva a esta regla. Por ello, alteramos este paso para aprovechar la nueva regla  $P \rightarrow xx$ , obteniendo así la derivación de la figura 2.15c que sólo usa reglas de la gramática modificada de la figura 2.14b.

Por último, observamos que la gramática que se desprende de este proceso de eliminación de reglas  $\lambda$  no puede generar cadenas que no generaba la gramática original. Después de todo, es posible simular cualquiera de las reglas añadidas por medio de cortas secuencias de reglas de la gramática original. Por

- a.  $S \Rightarrow zPzQz$   
 $\Rightarrow zxPxzQz$   
 $\Rightarrow zxQxzQz$   
 $\Rightarrow zxxzQz$   
 $\Rightarrow zxxxz$
- b.  $S \Rightarrow zPzz$   
 $\Rightarrow zxPxzz$   
 $\Rightarrow zxQxzz$   
 $\Rightarrow zxzzz$
- c.  $S \Rightarrow zPzz$   
 $\Rightarrow zxzzz$

**Figura 2.15** Modificación de una derivación basada en la gramática de la figura 2.14a para obtener una derivación basada en la gramática de la figura 2.14b

lo tanto, concluimos que *cualquier lenguaje independiente del contexto que no contiene la cadena vacía se puede generar por medio de una gramática independiente del contexto que no tenga reglas λ*.

Tomando como punto de partida esta conclusión, podemos demostrar el siguiente teorema.

#### TEOREMA 2.4

Si  $L$  es un lenguaje independiente del contexto que no contiene la cadena vacía, entonces existe una gramática  $G$  independiente del contexto tal que  $L(G) = L$  y el lado derecho de cualquier regla de reescritura en  $G$  consiste en un solo terminal o exactamente dos no terminales.

#### DEMOSTRACIÓN

Sea  $L$  un lenguaje independiente del contexto que no contiene la cadena vacía. Ya sabemos que una gramática  $G$  independiente del contexto que no contiene reglas  $λ$  puede generar  $L$ . Nuestro enfoque es modificar esta gramática para que se adhiera a las restricciones del teorema.

Para cada terminal  $x$  en  $G$ , introducimos un nuevo no terminal único  $X$  y la regla de reescritura  $X \rightarrow x$ , y luego reemplazamos las ocurrencias del terminal  $x$  en todas las demás reglas de  $G$  con  $X$ . Esto

produce una gramática  $G'$  independiente del contexto en la cual el lado derecho de cada regla de reescritura es un solo terminal o una cadena de no terminales. Además,  $L(G') = L(G)$ .

Ahora reemplazamos cada regla de  $G'$  de la forma

$$N \rightarrow N_1 N_2 \cdots N_n$$

donde  $n > 2$ , con la colección de reglas

$$\begin{aligned} N &\rightarrow N_1 R_1 \\ R_1 &\rightarrow N_2 R_2 \end{aligned}$$

$$R_{n-1} \rightarrow N_{n-1} N_n$$

donde cada  $R_k$  es un no terminal único que no aparece en ningún otro lugar de la gramática. Obviamente, esta modificación no cambia el lenguaje generado por la gramática.

Al llegar a esta etapa tenemos una gramática  $G'$  independiente del contexto que genera  $L$ , para la cual el lado derecho de cada regla de reescritura es un solo terminal, dos no terminales o un solo no terminal. Entonces, lo que queda por hacer es eliminar las reglas de la última forma. Para esto, consideramos cada secuencia de reglas de reescritura de la forma  $N_n \rightarrow N_{n-1}, N_{n-1} \rightarrow N_{n-2}, \dots, N_2 \rightarrow N_1$ , introducimos la regla  $N_n \rightarrow x$  si  $N_1 \rightarrow x$  es una regla de  $G'$ , e introducimos la regla  $N_n \rightarrow AB$  si  $N_1 \rightarrow AB$  está en  $G'$ . Una vez que se han agregado estas reglas, podemos eliminar aquellas donde el lado derecho contiene un solo no terminal, sin reducir los poderes generativos de la gramática. Después de todo, cualquier derivación que usa las reglas de la forma  $M \rightarrow N$  puede modificarse para que utilice en cambio las reglas que acabamos de introducir. Así, la gramática resultante genera  $L$  y satisface las restricciones del teorema.



Para hacer más claros los pasos de la demostración anterior, considere cómo afectarían a la gramática de la figura 2.16a, con símbolo inicial  $S$ . El primer paso sería introducir los nuevos no terminales  $X$ ,  $Y$  y  $Z$  y convertir la gramática en la gramática  $G'$  que se presenta en la figura 2.16 b. A continuación, la regla  $S \rightarrow ZMZ$  se reemplazaría por el par de reglas  $S \rightarrow ZR_1$  y  $R_1 \rightarrow MZ$ , mientras que  $M \rightarrow YMY$  se reemplazaría por  $M \rightarrow YP_1$  y  $P_1 \rightarrow MY$ , para obtener la gramática de la figura 2.16 c. Finalmente, la secuencia  $N \rightarrow X$  y la secuencia  $M \rightarrow N$  y  $N \rightarrow X$  darían origen a las reglas  $N \rightarrow x$  y  $M \rightarrow x$ , produciendo así la gramática de la figura 2.16 d.

a.	$S \rightarrow zMz$	b.	$S \rightarrow ZMZ$
	$M \rightarrow N$		$M \rightarrow N$
	$M \rightarrow yMy$		$M \rightarrow YMY$
	$N \rightarrow x$		$N \rightarrow X$
			$X \rightarrow x$
			$Y \rightarrow y$
			$Z \rightarrow z$
c.	$S \rightarrow ZR_1$	d.	$S \rightarrow ZR_1$
	$R_1 \rightarrow MZ$		$R_1 \rightarrow MZ$
	$M \rightarrow N$		$M \rightarrow x$
	$M \rightarrow YP_1$		$M \rightarrow YP_1$
	$P_1 \rightarrow MY$		$P_1 \rightarrow MY$
	$N \rightarrow X$		$N \rightarrow x$
	$X \rightarrow x$		$X \rightarrow x$
	$Y \rightarrow y$		$Y \rightarrow y$
	$Z \rightarrow z$		$Z \rightarrow z$

---

**Figura 2.16** Aplicación del proceso descrito en la demostración del teorema 2.4

Se dice que una gramática cuyas reglas de reescritura se adhieren a las restricciones del teorema 2.4 tiene la **forma normal de Chomsky** (llamada así en honor de N Chomsky). Por ejemplo, la gramática

$$\begin{aligned} S &\rightarrow XM \\ M &\rightarrow SY \\ X &\rightarrow x \\ Y &\rightarrow y \end{aligned}$$

cuyo símbolo inicial es  $S$  tiene la forma normal de Chomsky, mientras que

$$\begin{aligned} S &\rightarrow xSy \\ S &\rightarrow xy \end{aligned}$$

que genera el mismo lenguaje, no la tiene.

En resumen, el teorema 2.4 indica que cualquier lenguaje independiente del contexto que no contenga la cadena vacía puede ser generado por una gramática independiente del contexto que tenga la forma normal de Chomsky. Aunque se limita a un subconjunto de los lenguajes independientes del contexto, esta caracterización sigue siendo bastante general. Por ejemplo, la mayoría de los lenguajes en ambientes de aplicación, como los lenguajes de programación, no contienen la cadena vacía; incluso para aquellos lenguajes que sí contienen la cadena vacía, la caracterización tiene su mérito. De hecho,

Gramática para $L - \{\lambda\}$ con símbolo inicial $S$	Gramática para $L$ con símbolo inicial $S'$
$S \rightarrow MN$	$S' \rightarrow \lambda$
$N \rightarrow MS$	$S' \rightarrow MN$
$N \rightarrow x$	$S \rightarrow MN$
$M \rightarrow x$	$N \rightarrow MS$
	$N \rightarrow x$
	$M \rightarrow x$

$S' \rightarrow \lambda$   
 $S' \rightarrow MN$

} Nuevas reglas

**Figura 2.17** Modificación de una gramática con forma normal de Chomsky que genera  $L - \{\lambda\}$ , para obtener una nueva gramática que genere  $L$

si un lenguaje  $L$  independiente del contexto contiene  $\lambda$ , podemos encontrar una gramática independiente del contexto que “casi” tiene la forma normal de Chomsky pero que aún genera  $L$ . Para lograr esto, primero encontramos una gramática  $G$  independiente del contexto con forma normal de Chomsky que genere  $L - \{\lambda\}$ . Luego, modificamos  $G$  agregando un nuevo no terminal  $S'$  que se convierte en el símbolo inicial de la nueva gramática; después añadimos la regla  $S' \rightarrow \lambda$  (para que la nueva gramática genere  $\lambda$ ); y, por último, para cada regla de  $G$  cuyo lado izquierdo consiste en el antiguo símbolo inicial de  $G$ , agregamos una nueva regla en donde éste se sustituye con el nuevo no terminal  $S'$  en ese lado izquierdo (véase Fig. 2.17). Es evidente que esta gramática modificada generará las cadenas del lenguaje  $L$ , y sólo esas cadenas (al introducir el nuevo símbolo inicial  $S'$  se elimina la posibilidad de que la regla  $\lambda$  interactúe con otras reglas de la gramática). Por consiguiente, incluso los lenguajes independientes del contexto que contienen la cadena vacía pueden ser generados por gramáticas que casi tienen la forma normal de Chomsky.

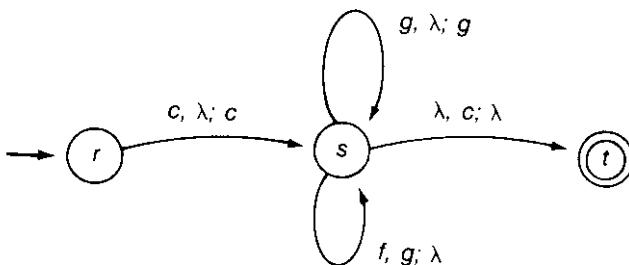
## Ejercicios

1. Muestre que toda cadena derivada por la izquierda de una gramática independiente del contexto puede derivarse también por la derecha.
2. Se dice que una gramática es ambigua cuando permite más de un árbol de análisis sintáctico para una sola cadena (véase Ap. C, Sec. C.1). Demuestre que la gramática que se presenta a continuación es ambigua, mostrando que la cadena *ictictses* tiene derivaciones que producen distintos árboles de análisis sintáctico.

$$\begin{aligned} S &\rightarrow ictS \\ S &\rightarrow ictSeS \\ S &\rightarrow s \end{aligned}$$

(Quizás ya haya observado antes este problema si asocia las siguientes palabras con los símbolos terminales de la gramática anterior:  $i$  = if,  $c$  = condición,  $t$  = then,  $e$  = else y  $s$  = enunciado.)

- Utilice el procedimiento descrito en el teorema 2.3 para construir una gramática independiente del contexto que genere el lenguaje aceptado por el autómata de pila cuyo diagrama de transiciones es el siguiente.



- Demuestre que la unión de dos lenguajes independientes del contexto también es independiente del contexto mostrando cómo pueden combinarse dos gramáticas independientes del contexto de los lenguajes originales para formar una gramática independiente del contexto que genere la unión.
- Convierta la siguiente gramática, con símbolo inicial  $S$ , en una gramática con forma normal de Chomsky que genere el mismo lenguaje.

$$\begin{aligned} S &\rightarrow xSy \\ S &\rightarrow wNz \\ N &\rightarrow S \\ N &\rightarrow \lambda \end{aligned}$$

## 2.3 LÍMITES DE LOS AUTÓMATAS DE PILA

Hasta ahora hemos caracterizado los lenguajes independientes del contexto como aquellos generados por gramáticas independientes del contexto y como aquellos aceptados por los autómatas de pila. Sin embargo, no hemos considerado el alcance de estos lenguajes: no nos hemos preguntado si existen lenguajes que no son independientes del contexto. Además, los autómatas de pila que hemos considerado hasta ahora son no deterministas y, ya que nuestro plan es desarrollar herramientas de diseño de compiladores basadas en las propiedades de los autómatas de pila, es necesario comprender el papel del determinismo en los autómatas de pila. Éstos serán los aspectos que trataremos en esta sección.

## Alcance de los lenguajes independientes del contexto

Primero presentaremos un lenguaje que no es independiente del contexto. Para esto utilizaremos el teorema siguiente, conocido como **lema de bombeo** ya que muestra cómo en algunos lenguajes independientes del contexto pueden producirse cadenas "bombeando" ("ampliando") porciones de otras cadenas.

### TEOREMA 2.5

Si  $L$  es un lenguaje independiente del contexto que contiene un número infinito de cadenas, entonces debe existir en  $L$  una cadena que tenga la forma  $svuwt$ , donde  $s, v, u, w$  y  $t$  son subcadenas, por lo menos una de  $v$  y  $w$  es no vacía, y  $sv^nuw^nt$  está en  $L$  para cada  $n \in \mathbb{N}^+$ .

### DEMOSTRACIÓN

Sea  $L$  un lenguaje independiente del contexto que contiene un número infinito de cadenas, y sea  $G$  una gramática independiente del contexto tal que  $L(G) = L$ . Sea  $m$  el número máximo de símbolos (terminales y no terminales) que se encuentran en el lado derecho de cualquier regla de reescritura en  $G$ ; es decir,  $m$  es la longitud del lado derecho más largo de las reglas de reescritura de  $G$ . Entonces, cada nodo de un árbol de análisis sintáctico basado en  $G$  puede tener cuando mucho  $m$  hijos. A su vez, cualquier árbol de análisis sintáctico de profundidad  $d$  puede producir una cadena de longitud máxima  $m^d$  (donde la profundidad del árbol es el número de aristas en la ruta más larga de la raíz a una hoja).

Sea ahora  $j$  el número de símbolos no terminales de  $G$ , y elija una cadena de  $L$  con longitud mayor que  $m^j$ . Entonces, el árbol de análisis sintáctico  $T$  para dicha cadena debe tener una profundidad mayor que  $j$ . Esto indica que existe una ruta en  $T$ , que va de la raíz a una hoja, la cual contiene más de  $j$  no terminales. Por consiguiente, algún no terminal  $N$  debe aparecer por lo menos dos veces en la ruta. Consideremos el subárbol de  $T$  cuyo nodo raíz es la ocurrencia más alta de  $N$  en esta ruta, y en el cual la siguiente ocurrencia de  $N$  es una hoja (como lo indica la región sombreada de la Fig. 2.18). En otras palabras, consideramos el subárbol de  $T$  cuya raíz es la ocurrencia más alta de  $N$  y luego descartamos todo lo que queda debajo de la siguiente ocurrencia de  $N$ .

Este subárbol indica que el patrón  $vNw$  se deriva de la primera ocurrencia de  $N$ , donde  $v$  y  $w$  son concatenaciones de las hojas del subárbol a la izquierda y a la derecha de la segunda  $N$ , respectivamente (véase de nuevo la Fig. 2.18). Podemos suponer que  $v$  o  $w$  debe ser no vacía, pues de lo contrario podríamos eliminar la región sombreada de la figura 2.18 para obtener el árbol de la figura 2.19a, recortando así la ruta elegida. Sin embargo si pudieran recortarse así todas las rutas con

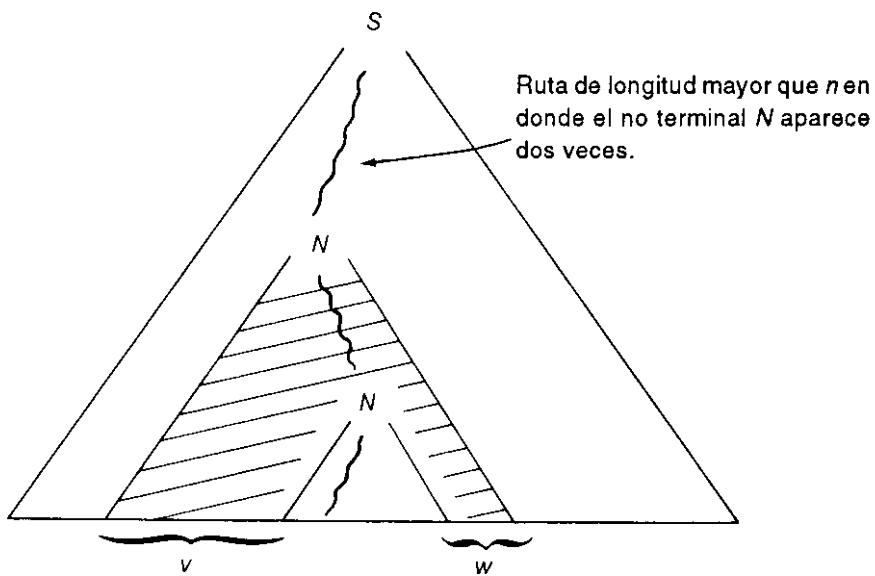


Figura 2.18 Representación gráfica del árbol de derivación  $T$

longitud mayor que  $j$ , produciríamos un árbol de análisis sintáctico para la cadena elegida que tuviera una profundidad menor que  $j$ , lo cual sería una contradicción.

Observe que pueden construirse otros árboles de análisis sintáctico repitiendo un número arbitrario de veces las copias del subárbol seleccionado, como se muestra en la figura 2.19b. Cada uno de estos árboles de análisis sintáctico representa una cadena que la gramática  $G$  puede generar. Por lo tanto,  $G$  genera cadenas que contienen estructuras de la forma  $vNw^i$  para cada entero positivo  $i$ . A su vez, debe existir una cadena en  $L$  de la forma  $svuwt$  donde  $sv^nuw^nt$  se encuentre en  $L$  para cada  $n \in \mathbb{N}^+$ , como se afirma en el teorema.

■

Una consecuencia del teorema 2.5 es que el lenguaje  $\{x^n y^n z^n : n \in \mathbb{N}^+\}$  no es independiente del contexto. De hecho, este lenguaje contiene una cantidad infinita de cadenas, pero no existe en el lenguaje cadena alguna que tenga un segmento con posibilidades de repetirse según lo establecido en el teorema y aún así producir cadenas en el lenguaje (si cada uno de los dos segmentos repetidos consiste solo en las  $x$ , sólo en las  $y$  o sólo en las  $z$ , entonces el resultado no contendrá el mismo número para cada símbolo. Además, si repetimos un segmento con más de un tipo de símbolo, entonces el resultado tendrá las  $y$  antes de las  $x$  o  $z$  antes de las  $y$ ).

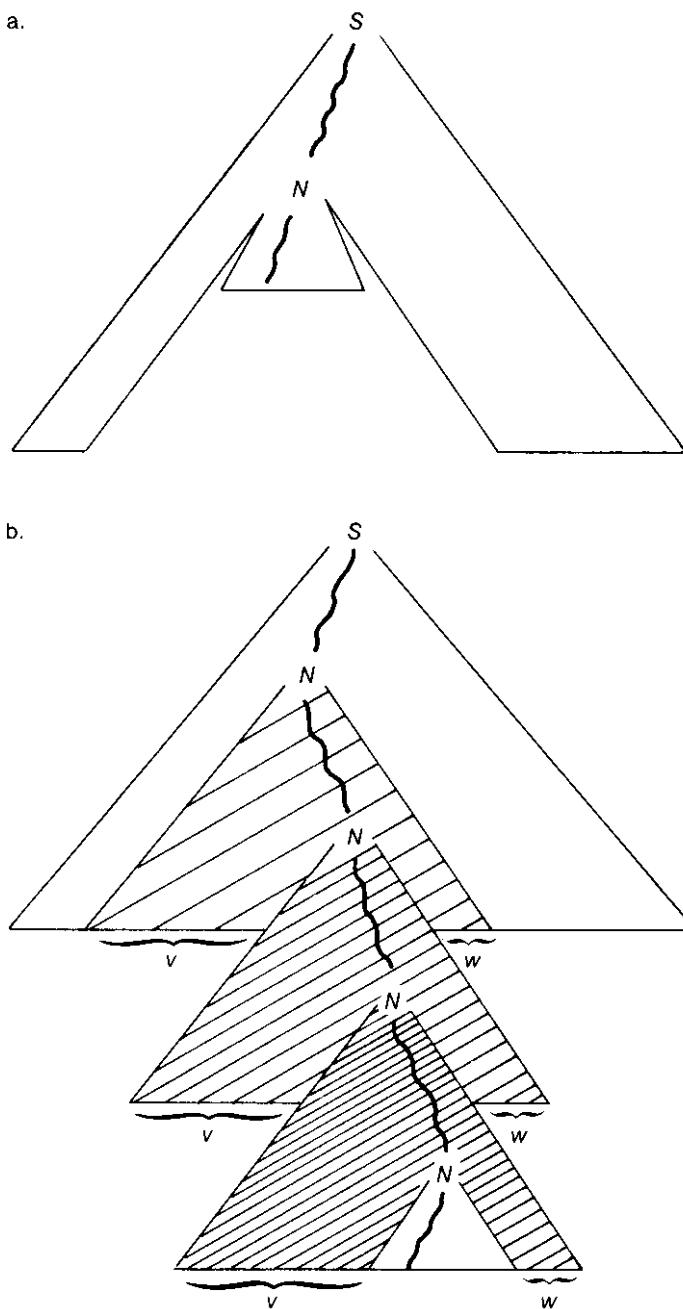


Figura 2.19 Árboles sintácticos que pueden construirse modificando el árbol de la figura 2.17

El hecho de que el lenguaje  $\{x^n y^n z^n : n \in \mathbb{N}^*\}$  no sea independiente del contexto puede parecer insignificante; consideremos entonces un ejemplo en el cual ocurren estos patrones. En algunos procesadores de palabras, las palabras que se subrayarán durante la impresión se almacenan como una cadena de símbolos (la palabra) seguida por el mismo número de retrocesos, seguidos por el mismo número de caracteres de subrayado. Así, estas palabras subrayadas constituyen cadenas que se ajustan al patrón  $x^n y^n z^n$ , donde las  $x$  son las letras de la palabra, las  $y$  los retrocesos y las  $z$  los símbolos de subrayado. Por consiguiente, el teorema 2.5 nos dice que el poder de los autómatas de pila sería insuficiente para construir una rutina de análisis sintáctico que pueda reconocer estas palabras subrayadas.

Si el ejemplo de las palabras subrayadas parece un tanto artificial, se debe a que la colección de los lenguajes independientes del contexto casi abarca las estructuras que se encuentran en los lenguajes de programación actuales. De hecho, los diagramas de sintaxis de uso común, utilizados para expresar la sintaxis de los lenguajes de programación son, esencialmente, reglas de reescritura independientes del contexto. Sin embargo, existen algunas características de estos lenguajes que tales diagramas no pueden representar. Los diagramas de sintaxis son incapaces de expresar la restricción de que diferentes variables no pueden tener el mismo nombre, que el número de parámetros formales de un subprograma debe ser igual al número de parámetros actuales cuando se llama al subprograma, y que las referencias a identificadores no declarados son ilegales.

No obstante lo anterior, el poder de las gramáticas independientes del contexto permite incluir un considerable número de reglas de sintaxis de los lenguajes de programación actuales, y por lo tanto vale la pena invertir tiempo en el desarrollo de técnicas de análisis sintáctico basadas en las propiedades de los autómatas de pila. De hecho, es con tales técnicas con las que muchos compiladores se constituyen en la actualidad. En estos casos, las características del lenguaje que se salen del alcance de las gramáticas independientes del contexto se manejan como casos especiales o se evalúan como parte del análisis semántico, en vez de hacerlo en las rutinas de análisis sintáctico.

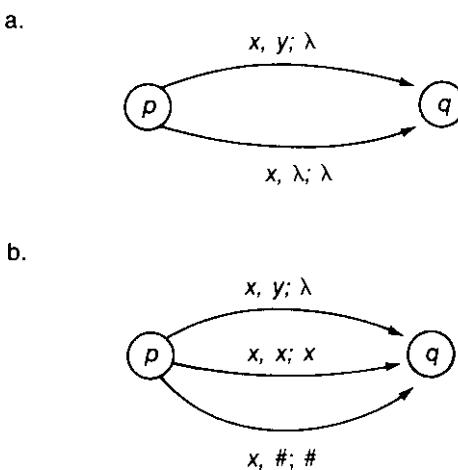
## Autómatas de pila deterministas

Persiste un problema que tenemos que resolver antes de centrarnos en la producción de rutinas de análisis sintáctico para autómatas de pila. Los autómatas de pila que hasta ahora hemos analizado son no deterministas, y nuestras rutinas de compilación deben ser deterministas. Si vamos a utilizar autómatas de pila como herramientas de diseño para el desarrollo de rutinas deterministas de análisis sintáctico, debemos saber cuáles son las limitaciones, si existen, que pueden surgir al requerir un comportamiento determinista. Nuestro primer paso en esa dirección es presentar el concepto de autómatas de pila deterministas.

De manera general, un **autómata de pila determinista** es un autómata de pila en el cual es aplicable una, y sólo una, transición en cualquier instante. Esto implica que si  $(p, x, y; q, z)$  y  $(p, x, y; r, w)$  son transiciones, entonces  $q$  debe ser igual a  $r$  y  $z$  debe ser igual a  $w$ . Sin embargo, esta condición no basta para excluir la posibilidad del no determinismo. Por ejemplo, si el siguiente símbolo de entrada fuera  $x$ , la presencia de las transiciones  $(p, \lambda, y; q, z)$  y  $(p, x, y; q, z)$  nos ofrecería la opción de pasar del estado  $p$  a otro ignorando la entrada o salir del estado  $p$  leyéndola. Se presentaría un problema similar si una máquina con  $y$  en la parte superior de la pila pudiera elegir entre  $(p, \lambda, \lambda; q, z)$  y  $(p, x, y; q, z)$ . ¿Debe salir la máquina del estado  $p$  ignorando la pila o extrayendo de ella un símbolo?

Con base en estas observaciones, un autómata de pila determinista se define como el autómata de pila  $(S, \Sigma, \Gamma, T, \iota, F)$  tal que para cada tripleta  $(p, x, y)$  en  $S \times \Sigma \times \Gamma$ , existe una y sólo una transición en  $T$  de la forma  $(p, u, v; q, z)$ , donde  $(u, v)$  está en  $\{(x, y), (\lambda, y), (y, \lambda)\}$ ,  $q$  está en  $S$  y  $z$  está en  $\Gamma$ . Por ejemplo, esta restricción eliminaría la presencia de  $(p, \lambda, y; q, z)$  y  $(p, x, \lambda; r, s)$ , ya que tanto  $(x, \lambda)$  como  $(\lambda, y)$  se hallan en  $\{(x, y), (\lambda, y), (\lambda, \lambda), (y, \lambda)\}$ .

Para subrayar las sutilezas del desarrollo de un autómata de pila determinista, supongamos que queremos diseñar la porción de una de estas máquinas de manera que cuando se encuentre en el estado  $p$  ejecute la transición  $(p, x, y; q, \lambda)$  si hay una  $y$  en la cima de la pila, o de lo contrario ejecuta la transición  $(p, x, \lambda; q, \lambda)$ . Aquí no se presenta incertidumbre alguna, pero sí la hay en el sencillo enfoque que se presenta en la figura 2.20a. Específicamente, el diagrama permitiría que la máquina tuviera una opción si en la cima de la pila existiera una  $y$ , y serían aplicables tanto  $(p, x, y; q, \lambda)$  como  $(p, x, \lambda; q, \lambda)$ . Para resolver este problema pasamos a la estrategia que se ilustra en la figura 2.20b, donde suponemos que los símbolos de pila de la máquina son  $x$ ,  $y$  y  $\#$ .



**Figura 2.20** Porción del diagrama de transiciones para un autómata de pila que presenta no determinismo y su contrapartida determinista

Allí hemos introducido opciones explícitas para la máquina en caso de que el símbolo en la cima de la pila no fuera  $y$ . En estos casos se indica a la máquina que extraiga un símbolo de la pila y luego lo inserte de nuevo. Por supuesto, este enfoque requiere que la pila no se encuentre vacía cuando la máquina se halle en el estado  $p$ , pues de lo contrario ninguna de las transiciones sería aplicable. Es por esto que hemos introducido el símbolo  $\#$  que se supone está en el fondo de la pila al iniciar cualquier cálculo, y que se conserva allí hasta el final.

Al igual que en el capítulo anterior, nuestro uso del término determinista implica que el sistema está completamente definido. Sin embargo, las afirmaciones que estamos a punto de efectuar con respecto a los autómatas de pila deterministas serán válidas, sin importar si los autómatas en cuestión se encuentran o no completamente definidos. Además, si se obliga a definir por completo a los autómatas, se obtienen diagramas más bien saturados. Como vimos en el caso de los autómatas finitos, la inclusión de los arcos para todos los casos es una cuestión más de paciencia que de contenido (compare la Fig. 1.5 con la 1.11). Entonces, al dibujar diagramas para autómatas de pila deterministas, con frecuencia representaremos sólo aquellas transiciones pertinentes para la tarea en cuestión, entendiéndose que si es necesario puede completarse el diagrama parcial.

Nuestra siguiente tarea es decidir si el requisito del determinismo reduce o no el poder de un autómata de pila. Por desgracia para nosotros, así es, como se muestra en el teorema siguiente.

### TEOREMA 2.6

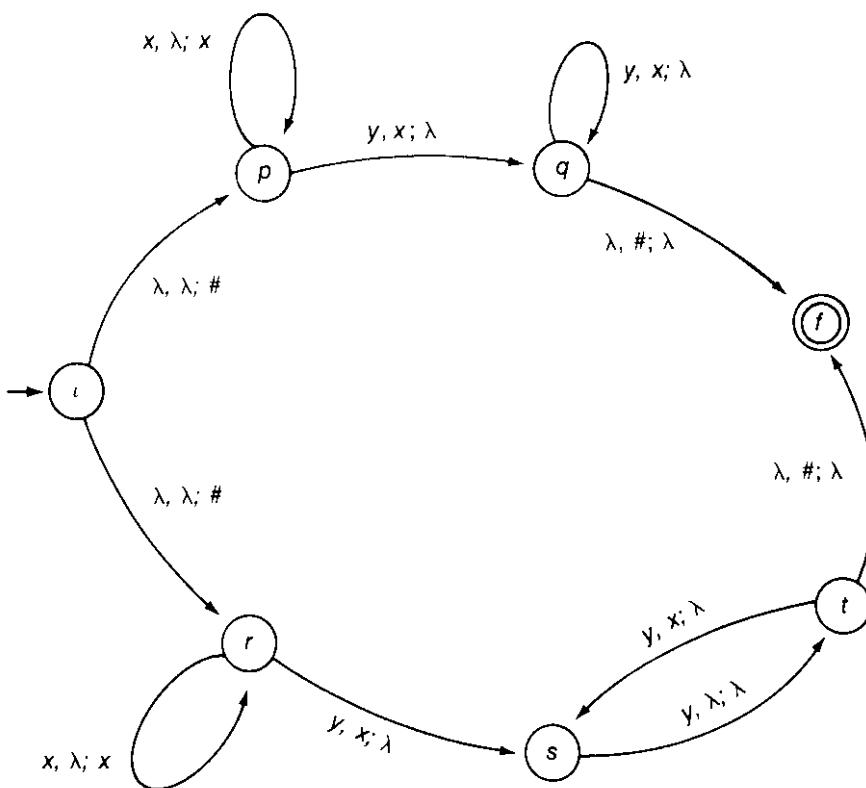
Existe un lenguaje independiente del contexto que no es el lenguaje aceptado por ningún autómata de pila determinista.

#### DEMOSTRACIÓN

Demostramos este teorema mostrando que el lenguaje  $L = \{x^n y^n : n \in \mathbb{N}^+\} \cup \{x^n y^{2n} : n \in \mathbb{N}^+\}$  es independiente del contexto pero que no puede ser aceptado por ningún autómata de pila determinista. En primer lugar, observe que la figura 2.21 muestra un diagrama de transiciones para un autómata de pila que acepta  $L$ ; por lo tanto,  $L$  es independiente del contexto.

Para probar que  $L$  no puede ser aceptado por ningún autómata de pila determinista, mostramos que se llega a una contradicción si se supone lo contrario. Suponga entonces que  $M$  es un autómata de pila determinista tal que  $L(M) = L$ . Utilizando  $M$ , construimos otro autómata de pila de la manera siguiente:

1. Construya dos copias de  $M$ , llamadas  $M_1$  y  $M_2$ . Los estados de  $M_1$  y  $M_2$  se llamarán primos si son copias del mismo estado en  $M$ .
2. Elimine la característica de aceptación de los estados de aceptación de  $M_1$  y la característica de inicio del estado inicial de  $M_2$ .



**Figura 2.21** Diagrama de transiciones para un autómata de pila  $M$  para el cual  $L(M) = \{x^n y^n : n \in \mathbb{N}^+\} \cup \{x^n y^{2n} : n \in \mathbb{N}^+\}$

3. Cambie el estado destino  $p$  de cada una de las transiciones que se originan en un antiguo estado de aceptación de  $M_1$  al primo de  $p$  en  $M_2$ . (Estas transiciones alteradas forman enlaces entre  $M_1$  y  $M_2$ , de manera que las dos máquinas se convierten en un solo autómata de pila).
4. Modifique todas aquellas transiciones que lean una  $y$  de la entrada  $y$  cuyos estados destino se encuentren en  $M_2$  (incluyendo quizás, aquellas transiciones alteradas en el paso 3), para que lean el símbolo  $z$  en su lugar.

Afirmamos que el lenguaje aceptado por el autómata de pila construido así a partir de  $M_1$  y  $M_2$  será  $\{x^n y^n z^n : n \in \mathbb{N}^+\}$ . De hecho, si a esta máquina se le proporcionara una cadena de entrada de la forma  $x^n y^n z^n$ , tendría que llegar a uno de los antiguos estados de aceptación de  $M_1$  después de leer las  $x$  y las  $y$  pero antes de leer alguna  $z$  (la  $M_1$

original habría aceptado  $x^n y^n$  y, puesto que es determinista, debe llegar al mismo estado después de leer  $x^n y^n$  sin importar qué símbolos vengan después). Al llegar a este punto, la ejecución cambiaría a  $M_2$  (por el paso 3 anterior), donde todas las  $z$  de la cadena de entrada llevarían a un estado de aceptación. En efecto,  $M_2$  proseguiría como si estuviera procesando la segunda porción de una cadena de la forma  $x^n y^{2n}$ , pero como se han alterado sus instrucciones de lectura, en realidad leerá las  $z$  en vez de las  $y$ .

Para ver que sólo se aceptarían las cadenas de la forma  $x^n y^n z^n$ , observe que para llegar a un estado de aceptación se requiere que  $M_1$  procese una cadena de la forma  $x^n y^n$  para que pueda transferir el control a  $M_2$ , y luego que  $M_2$  encuentre  $n z$ , ya que  $M_2$  prosigue como si estuviera procesando la segunda parte de  $x^n y^{2n}$ .

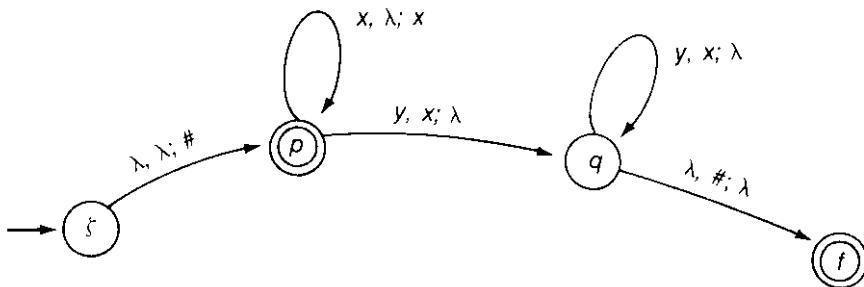
Vemos entonces que nuestra suposición de que  $L$  puede ser aceptado por un autómata de pila determinista nos lleva a la conclusión de que el lenguaje  $\{x^n y^n z^n : n \in \mathbb{N}\}$  es independiente del contexto; pero sabemos que esto es falso. Por consiguiente, nuestra suposición debe ser errónea:  $L$  no puede ser aceptado por un autómata de pila determinista.



Tomando en cuenta el teorema 2.6, nos referimos a los lenguajes aceptados por un autómata de pila determinista como **lenguajes independientes del contexto determinista**. El hecho de que esta clase de lenguajes no incluya todos los lenguajes independientes del contexto nos indica que nuestro objetivo de construir rutinas deterministas de análisis sintáctico y basadas en autómatas de pila no se alcanzará para todos los lenguajes independientes del contexto, sino únicamente en el caso, más reducido, de los lenguajes independientes del contexto determinista.

De hecho, el panorama es aún más desalentador. Los autómatas de pila deterministas que consideramos aceptan cadenas sin que necesariamente tengan que vaciar sus pilas y, como se mencionó antes, es probable que este modelo dé origen a segmentos de programa que saturen la memoria de un computador con los residuos de cálculos anteriores. Por desgracia, este problema es más serio para los autómatas de pila deterministas que para los autómatas de pila ordinarios, ya que no podemos modificar cada autómata de pila determinista para que vacíe su pila antes de llegar al estado de aceptación.

Para convencernos de esta situación, considere el lenguaje  $L$  obtenido de la unión de  $\{x^n : n \in \mathbb{N}\}$  y  $\{x^n y^n : n \in \mathbb{N}\}$ , que es aceptado por el autómata de pila determinista representado en el diagrama parcial de la figura 2.22, y que por lo tanto es un lenguaje independiente del contexto determinista. Afirmamos que  $L$  no puede ser aceptado por un autómata de pila determinista al cual se le requiera que vacíe su pila antes de llegar a un estado de aceptación. De hecho, si  $M$  fuera dicha máquina y recibiera una entrada de la forma  $x^n y^n$ , después de leer cada  $x$  tendría que llegar a un estado de aceptación con la pila



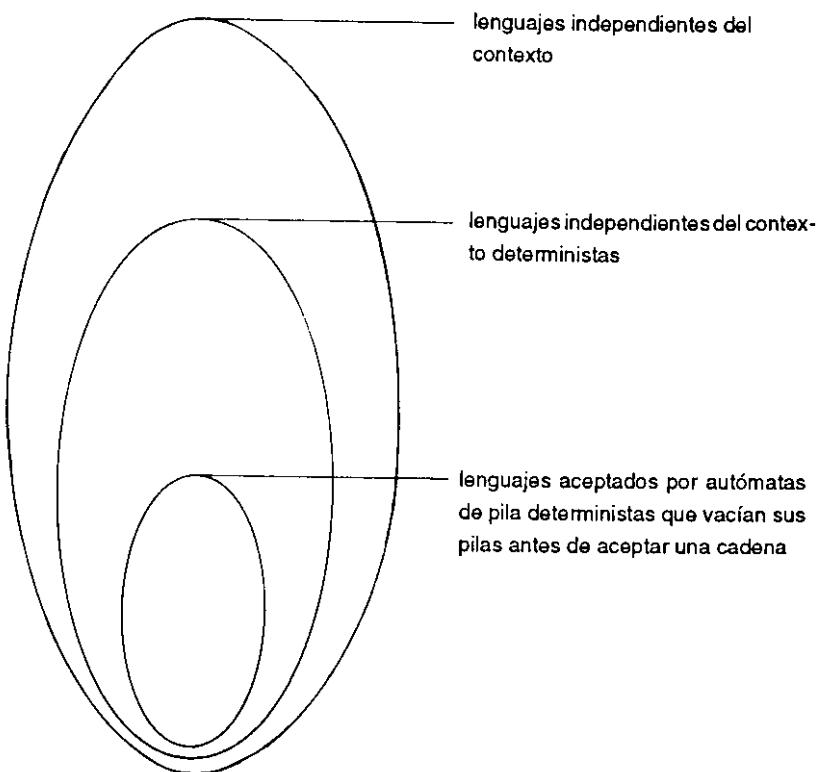
**Figura 2.22** Diagrama parcial para un autómata de pila determinista  $M$  donde  $L(M) = \{x^n: n \in \mathbb{N}^+\} \cup \{x^n y^n: n \in \mathbb{N}^+\}$

vacía. (Puesto que la máquina es determinista, debe seguir la misma ruta de ejecución al procesar las  $x$  en  $x^n y^n$  que al procesar cualquier cadena de  $x$ , y puesto que cualquier cadena de  $x$  es en sí una cadena aceptable, la máquina debe vaciar su pila y pasar a un estado de aceptación después de leer cada  $x$ ). Ahora, si escogemos que  $n$  sea mayor que el número de estados en  $M$ , podemos concluir que en algún punto del procesamiento de las  $x$  en  $x^n y^n$  la máquina debe estar por lo menos dos veces en un estado  $p$  con la pila vacía. Si  $m$  es el número de  $x$  leídas entre estas visitas a  $p$ ,  $M$  debe aceptar la cadena  $x^{n+m} y^n$ , lo que contradice la definición de  $M$ .

En resumen, hemos encontrado una jerarquía de lenguajes asociados a los autómatas de pila que se presenta en la figura 2.23. La mayor de las clases es la colección de lenguajes independientes del contexto que los autómatas de pila generales aceptan. Dentro de esta clase se encuentran propiamente incluidos los lenguajes independientes del contexto deterministas aceptados por los autómatas de pila deterministas (aquellos que no tienen el requisito de vaciar sus pilas). Luego, propiamente contenidos en la clase de los lenguajes independientes del contexto deterministas, se hallan los lenguajes que pueden ser aceptados por los autómatas de pila deterministas que vacían sus pilas antes de aceptar una cadena de entrada.

### Principio de preanálisis

Al llegar a este punto es probable que comience a desvanecerse nuestra esperanza de desarrollar segmentos de programa utilizables para el análisis sintáctico de una amplia clase de lenguajes. Parece que cualquier técnica que se desarrolle estará restringida a la menor clase de lenguajes de la figura 2.23. Por fortuna, esta situación mejora si consideramos de nuevo la forma en que los autómatas aceptan sus cadenas de entrada. Recuerde que en la sección 2.1 vimos que, para aceptar una cadena, un autómata de pila debe cumplir con su criterio de aceptación después de leer la cadena pero sin desplazar más allá su cabeza de lectura por la cinta. Esto significa que un autómata de pila debe



**Figura 2.23** Clasificación de los lenguajes independientes del contexto

llegar a un estado de aceptación sin realmente leer la marca de fin de cadena. El resultado de esto es lo que podríamos considerar como un proceso de aceptación pasivo en el cual la máquina debe “caer” en un estado de aceptación sin que se lo indique una marca de fin de cadena. Por el contrario, si imaginamos un sistema mediante el cual la máquina pudiera observar el siguiente símbolo de la cinta (sin tener que avanzar su cabeza de lectura), podría detectar la marca de fin de cadena próxima y ejecutar actividades especiales “de cierre” que de otra manera no ejecutaría.

Nos referiremos a la técnica de observar los símbolos siguientes sin leerlos como **principio de preanálisis**. Su aplicación se exemplifica con la estructura del ciclo while del segmento de programa de la figura 2.24. En esencia, este segmento observa el siguiente símbolo y emplea la información obtenida para decidir si procesa o no el símbolo dentro de la estructura while. En las distintas opciones del enunciado case, la rutina finalmente decide si consume el símbolo y se prepara para procesar otro. Así, la variable símbolo es en realidad un área de almacenamiento temporal, o *buffer*, para el símbo-

```

leer(símbolo);
while (símbolo no es la marca de fin de cadena)
    case símbolo of
        x: insertar (y) y leer (símbolo)
        y: salir a la rutina de error
        z: insertar (#) y leer (símbolo)
    end case
end while

```

---

**Figura 2.24 Ciclo *while* típico**

lo siguiente. Para tomar una decisión, se puede considerar un símbolo allí almacenado, sin que tenga que consumirse (o leerse oficialmente) hasta tomar la decisión de procesarlo. De modo específico, esta rutina no consumirá la marca de fin de cadena sino que permanecerá en el buffer tras terminar la estructura *while*, donde estará disponible como entrada para la siguiente rutina. Después de todo, en un compilador real es posible que la marca de fin de cadena sea el primer símbolo de la siguiente estructura que haya que analizar.

Vemos entonces que la aplicación del principio de preanálisis es una técnica de programación común. Estamos a punto de ver también que al aplicar este principio durante la conversión de autómatas de pila a segmentos de programa podemos construir programas que superen el no determinismo de algunos autómatas de pila. Por lo tanto, la teoría de autómatas de pila proporciona la base para construir rutinas de análisis sintáctico destinadas a una amplia gama de lenguajes independientes del contexto. Cómo logra esto es el tema de las dos secciones siguientes.

## Ejercicios

1. Aplique el teorema 2.5 para demostrar que el lenguaje  $\{x^m y^n x^m y^n x^m y^n : m, n \in \mathbb{N}^*\}$  no es independiente del contexto.
2. Proporcione ejemplos de los siguientes lenguajes:
  - a. Un lenguaje que no es independiente del contexto.
  - b. Un lenguaje independiente del contexto pero no determinista.
  - c. Un lenguaje que es independiente del contexto determinista pero que no es aceptado por un autómata de pila determinista que tiene que vaciar su pila.
  - d. Un lenguaje que es aceptado por un autómata de pila determinista que tiene que vaciar su pila pero que no es un lenguaje regular.

3. Dibuje la porción pertinente de un diagrama de transiciones para un autómata de pila determinista que pasará del estado  $p$  al  $q$  leyendo una  $x$ , extrayendo una  $y$  e insertando una  $z$ , si la cima de la pila contiene una  $y$ , o pasará al estado  $q$  leyendo una  $x$ , sin extraer nada e insertando una  $z$  si la cima de la pila no contiene una  $y$ .
4. Identifique las situaciones que darían origen a incertidumbres en la ejecución de un autómata de pila que contuviera las transiciones  $(p, \lambda, y; q, z)$  y  $(p, x, \lambda; r, w)$ .

## 2.4 ANALIZADORES SINTÁCTICOS $LL(k)$

Ahora es el momento de considerar cómo pueden desarrollarse las rutinas de análisis sintáctico a partir de los autómatas de pila. Tradicionalmente, este problema surge cuando se describe un lenguaje en función de reglas de reescritura gramaticales. Después de ello, se desarrollan las rutinas de análisis sintáctico para el lenguaje empleando la teoría de los autómatas de pila como herramienta de desarrollo. Éste será el contexto para nuestro análisis.

### Proceso de análisis sintáctico $LL$

Una técnica para traducir gramáticas independientes del contexto a autómatas de pila es seguir el proceso descrito en la demostración del teorema 2.2. Esta construcción produce un autómata de pila que analiza su cadena de entrada marcando antes el fondo de la pila e insertando en la pila el símbolo inicial de la gramática. Luego repite los tres pasos siguientes, según resulte aplicable.

1. Si la cima de la pila contiene un no terminal de la gramática, reemplace ese no terminal de acuerdo con una de las reglas de reescritura de la gramática.
2. Si la cima de la pila contiene un terminal, elimínelo de la pila a la vez que lee de la entrada el mismo terminal. Si el símbolo en la entrada no equivale al símbolo de la pila, se declara que la entrada es una cadena ilegal.
3. Si aparece en la superficie de la pila la marca de fondo de pila, elimínela y declare que es aceptable la porción de la cadena de entrada procesada hasta el momento.

Recuerde que este proceso analiza la sintaxis de la cadena de entrada produciendo una derivación por la izquierda conforme lee la cadena de izquierda a derecha. Por consiguiente, actuará de la misma manera un segmento de programa que se obtenga traduciendo directamente el autómata a enunciados de programa. Los analizadores sintácticos desarrollados de esta manera se conocen como **analizadores sintácticos  $LL$** . La primera  $L$  denota que

el analizador lee su entrada de izquierda a derecha (*Left to right*, en inglés); la segunda  $L$  indica que el objetivo del analizador sintáctico es producir una derivación por la izquierda (*Leftmost derivation*, en inglés).

La figura 2.25b muestra un diagrama de transiciones construido a partir de la gramática de la figura 2.25a usando el proceso descrito en la demostración del teorema 2.2 (se trata de la gramática independiente del contexto que vimos en la figura 2.7, la cual genera cadenas de la forma  $x^n y^n$  para enteros no negativos  $n$ ). Para producir una rutina de análisis sintáctico a partir de esta gramática, podemos convertir las transiciones de la máquina directamente a enunciados de programa, obteniendo así la rutina de la figura 2.26, donde hemos empleado la estructura *while* tradicional para simular las actividades disponibles para la máquina cuando se encuentra en el estado  $q$  (mientras el símbolo en la cima de la pila no sea la marca de fondo de pila, la máquina permanecerá en el estado  $q$ ).

Obviamente, el segmento de la figura 2.26 no es un producto terminado. En primer lugar, no hemos considerado los errores ocasionados por entradas inválidas. Por ejemplo, si aparece una  $x$  en la superficie de la pila durante el ciclo *while*, nuestra rutina supone que el siguiente símbolo de la entrada es una  $x$  y ejecuta la instrucción “leer una  $x$  de la cadena de entrada”. En realidad, el siguiente símbolo puede ser distinto de  $x$ , por lo que nuestra rutina debe tomar en cuenta esta posibilidad. Debido a ello debemos ampliar la instrucción “leer una  $x$  de la cadena de entrada” para obtener el par de instrucciones que se muestra a continuación:

```
leer(símbolo);
if símbolo no es x then salir a la rutina de error;
```

Otro problema de menor magnitud que existe en la rutina de la figura 2.26 es que puede llegar al estado  $f$  con una pila vacía sin haber leído toda la cadena de entrada. Por ejemplo, la cadena  $xyx$  no existe en el lenguaje descrito por la gramática original, pero nuestra rutina nunca se percatará de esto. En cambio, leerá la entrada hasta  $xy$ , donde se detendrá suponiendo que la cadena de entrada es válida. Este problema se puede corregir agregando las instrucciones

```
leer(símbolo);
if símbolo no es la marca de fin de cadena then salir a la rutina de error
```

al final de la rutina.

Sin embargo, en la rutina existe un problema que es más severo que los anteriores: en algunos casos las órdenes presentan opciones sin resolver. De hecho, si el estado actual es  $q$  y el símbolo en la cima de la pila es  $S$ , la rutina ofrece la opción de reemplazar la  $S$  con  $xSy$  o simplemente eliminar la  $S$  de la pila. Este problema es fundamentalmente distinto de los temas ya mencionados, ya que implica la selección de instrucciones en vez de una mera clarificación o refinamiento de los detalles de las instrucciones.

a.  $S \rightarrow xSy$   
 $S \rightarrow \lambda$

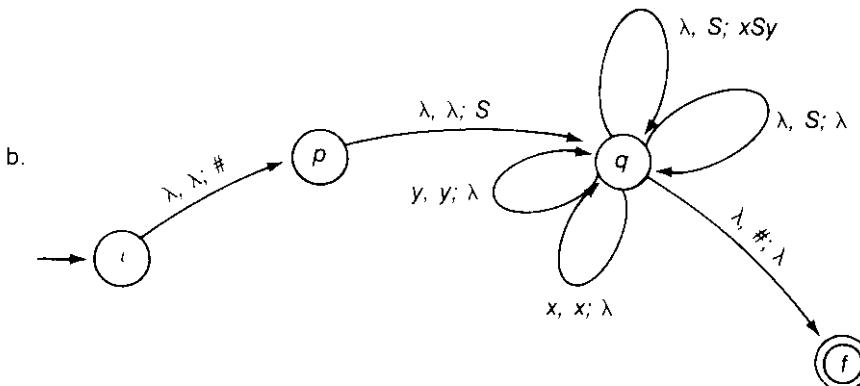


Figura 2.25 Gramática independiente del contexto y diagrama de transiciones asociado para un autómata de pila

```

Estado := i;
insertar (#);
Estado := p;
insertar (S);
Estado := q ;
while cima-de-la-pila ≠ # do
    case cima-de-la-pila of
        S: extraer (S) e insertar (xSy), o extraer (S);
        x: extraer (x), leer una x de la entrada;
        y: extraer (y), leer una y de la entrada;
    end case
end while;
extraer (#);
Estado := f

```

Figura 2.26 Segmento de “programa” obtenido al traducir a enunciados el diagrama de la figura 2.25

### Aplicación del principio de preanálisis

Por fortuna, el no determinismo de nuestra rutina se puede resolver utilizando el principio de preanálisis presentado en la sección anterior. Si

```

Estado := t;
insertar (#);
Estado := p;
insertar (S);
Estado := q;
leer (Símbolo);
while cima-de-la-pila ≠ # do
    case cima-de-la-pila of
        S : if Símbolo ≠ x then extraer (S)
              else extraer (S), insertar (xSy);
        x : if Símbolo no es x then salir a la rutina de error
              else extraer (x), leer (Símbolo);
        y : if Símbolo no es y then salir a la rutina de error
              else extraer (y), leer (Símbolo);
    end case
end while;
extraer (#)
if Símbolo no es la marca de fin de cadena then salir a la rutina de error;
Estado := f

```

**Figura 2.27** Rutina de análisis sintáctico basada en la gramática de la figura 2.25

encontramos una  $x$  al observar el siguiente símbolo de la entrada, entonces debemos reemplazar la  $S$  por la cadena  $xSy$ ; de lo contrario, debemos sustituirla por la cadena vacía. (Si insertamos  $xSy$  en la pila sabiendo que el siguiente símbolo de la entrada no es una  $x$ , estamos condenados al fracaso. Una vez que insertamos un símbolo terminal en la pila, para poder extraerlo debe ser igual a un símbolo de la entrada. Si colocamos  $xSy$  en la pila cuando observamos en la entrada un símbolo distinto de  $x$ , el símbolo de la entrada no será igual a la  $x$  en la cima de la pila, y nunca podríamos vaciar la pila y pasar al estado de aceptación.)

Con base en lo anterior, podemos convertir el diagrama no determinista de la figura 2.25 en el segmento de programa determinista que se muestra en la figura 2.27. Aquí hemos utilizado la variable Símbolo como un almacenamiento temporal para el siguiente símbolo de la entrada. A partir de este almacenamiento temporal es posible interrogar cuál es el símbolo cada vez que se tengan que tomar decisiones, pero sin procesarlo antes de que sea necesario. Sobre todo, observe que la rutina no consume la marca de fin de cadena, aunque la detecte; permanece en el almacenamiento temporal, donde puede usarse como el primer símbolo de la siguiente estructura que será analizada por el sistema de análisis sintáctico.

El problema que surge en el ejemplo anterior es un fenómeno común en los analizadores sintácticos  $LL$ , pues se origina cuando la gramática propone más de una forma de reescribir el mismo no terminal. Estas opciones múltiples son esenciales para las gramáticas que deben generar lenguajes que contienen más de una cadena (una gramática independiente del contexto que sólo ofrece una manera de reescribir cada no terminal sólo puede generar una cadena). Por

$$\begin{aligned} S &\rightarrow xSz \\ S &\rightarrow xyTyz \\ T &\rightarrow \lambda \end{aligned}$$

**Figura 2.28** Gramática independiente del contexto que requiere una analizador sintáctico  $LL(2)$

esto, la actividad básica de los analizadores sintácticos  $LL$  es predecir cuál de las distintas reglas de reescritura es la que debe usarse para procesar los símbolos de entrada restantes. Por consiguiente, a estos analizadores se les llama **analizadores sintácticos predictivos**.

Si se aplica el principio de preanálisis, pueden resolverse muchas de las incertidumbres que se presentan en los analizadores sintácticos predictivos. Sin embargo, incluso en aquellos casos donde el principio de preanálisis es la técnica indicada, es posible que su aplicación no sea tan directa como en nuestro ejemplo. Si fuéramos a construir un analizador sintáctico a partir de la gramática de la figura 2.28, encontraríamos que la decisión con respecto a la reescritura de  $S$  no puede resolverse con sólo observar el siguiente símbolo de entrada (el conocimiento de que el siguiente símbolo es  $x$  no nos indica que debemos aplicar  $S \rightarrow xSy$  en vez de  $s \rightarrow xyTyz$ ). En cambio, la decisión depende de los dos símbolos siguientes. Entonces, para desarrollar una rutina determinista de análisis sintáctico debemos contar con espacio de almacenamiento temporal para dos símbolos de entrada.

Como resultado, existe una jerarquía de analizadores sintácticos  $LL$  cuya característica distintiva es el número de símbolos de entrada que comprende su sistema de preanálisis. Estos analizadores se llaman **analizadores sintácticos  $LL(k)$** , donde  $k$  es un entero que indica el número de símbolos preanalizados por el analizador sintáctico. El ejemplo de la figura 2.27 es un analizador sintáctico  $LL(1)$ , mientras que un analizador sintáctico basado en la gramática de la figura 2.28 sería un analizador sintáctico  $LL(2)$ .

Usted quizás piense (correctamente) que la carga de predicciones que se coloca sobre los analizadores sintácticos  $LL(k)$  restringe a fin de cuentas, los lenguajes que pueden analizarse. En efecto, existen lenguajes dentro de los límites de los analizadores sintácticos de pila que no puede reconocer ningún analizador sintáctico  $LL(k)$ , sin importar la magnitud de  $k$ . Un ejemplo es el lenguaje  $\{x^n : n \in \mathbb{N}\} \cup \{x^n y^n : n \in \mathbb{N}\}$ , que ya sabemos es determinista independiente del contexto. Por intuición, cualquier gramática independiente del contexto que genere este lenguaje debe permitir que se reescriba un no terminal con una cadena que contiene sólo  $x$  o una cadena que contiene una combinación equilibrada de  $x$  y  $y$ . Esto quiere decir que existirán por lo menos dos reglas para reescribir este no terminal. A su vez, cualquier analizador sintáctico  $LL(k)$  se enfrentará al problema de decidir cuál de estas reglas será la que se aplique cuando surja el no terminal en la superficie de la pila. Por desgracia, independientemente de la magnitud de  $k$ , existen cadenas en el lenguaje en las cuales no es posible detectar la existencia de o la ausencia de  $y$  posteriores sin antes observar más de  $k$   $x$  de

preanálisis. Entonces, cualquier analizador sintáctico  $LL(k)$  será incapaz de manejar las decisiones necesarias para analizar la sintaxis de este lenguaje.

La existencia de un lenguaje determinista independiente del contexto que no puede ser analizado por un analizador sintáctico  $LL(k)$  sugiere que pueden existir analizadores sintácticos basados en la teoría de los autómatas de pila más poderosos que estos analizadores sintácticos predictivos, hipótesis que nos conduce a la siguiente sección. No obstante, al incrementar el poder aumenta también la complejidad, por lo que la sencillez de los analizadores sintácticos  $LL(k)$  hace de ellos una opción popular cuando son capaces de manejar el lenguaje que se considera. Por ahora, dejamos a un lado nuestra búsqueda de potencia y consideraremos cómo pueden simplificarse los analizadores sintácticos  $LL(k)$  utilizando tablas de análisis sintáctico.

### Tablas de análisis sintáctico $LL$

Una tabla de análisis sintáctico para un analizador sintáctico  $LL(1)$  es una matriz bidimensional. Las filas se etiquetan con los no terminales de la gramática sobre la cual se basa el analizador sintáctico. Las columnas se etiquetan con los terminales de la gramática más una columna adicional FDC (que representa la marca de fin de cadena). El elemento  $(m, n)$  de la tabla indica la acción que debe seguirse cuando el no terminal  $m$  aparece en la cima de la pila y el símbolo de preanálisis es  $n$ . Si en esta situación debiera reemplazarse el no terminal  $m$  siguiendo una regla de reescritura, el lado derecho de la regla aparecería en la posición  $(m, n)$ . De lo contrario, la casilla contiene un indicador de error. Por ejemplo, en la figura 2.29 se presenta una tabla de análisis sintáctico para la gramática de la figura 2.5.

Una vez que se ha construido la tabla de análisis sintáctico, la tarea de escribir un segmento de programa que efectúe el análisis sintáctico del lenguaje es bastante sencilla. Lo único que tiene que hacer el segmento es insertar en la pila el símbolo inicial de la gramática y luego, mientras no se vacíe la pila, igualar los terminales de la cima de la pila con los de la entrada o reemplazar los no terminales de la cima de la pila siguiendo las directrices de la tabla de análisis sintáctico. Por ejemplo, la rutina de la figura 2.30 es un analizador sintáctico  $LL(1)$  que utiliza la tabla de la figura 2.29.

	<i>a</i>	<i>b</i>	<i>z</i>	FDC
<i>S</i>	error	error	<i>zM Nz</i>	error
<i>m</i>	<i>a Ma</i>	error	<i>z</i>	error
<i>N</i>	error	<i>b Nb</i>	<i>z</i>	error

Figura 2.29 Tabla de análisis sintáctico  $LL(1)$  para la gramática de la figura 2.5

```

insertar (S);
leer (Símbolo);
while pila no está vacía do
  case cima-de-la-pila of
    terminal: if cima-de-la-pila = Símbolo
      then extraer de la pila y leer (Símbolo)
      else salir a la rutina de error;
    no terminal: if tabla [cima-de-la-pila, Símbolo] ≠ error
      then reemplazar cima-de-la-pila por tabla
      [cima-de-la-pila, Símbolo]
      else salir a la rutina de error;
  end case
end while
if Símbolo no es la marca de fin de cadena then salir a la rutina de error

```

Figura 2.30 Rutina genérica de análisis sintáctico  $LL(1)$ 

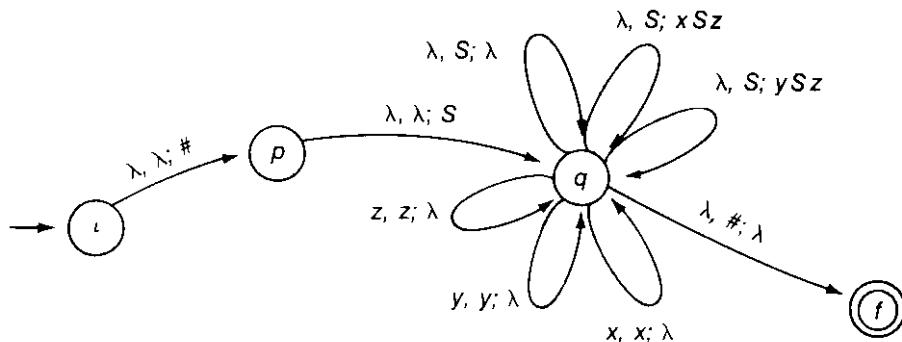
<i>S</i>	<i>x</i>	<i>y</i>	<i>FDC</i>
	<i>xSy</i>	$\lambda$	$\lambda$

Figura 2.31 Tabla de análisis sintáctico  $LL(1)$  para la gramática de la figura 2.7

Otra ventaja de utilizar tablas de análisis sintáctico es que permiten normalizar el algoritmo de análisis. Cualquier analizador sintáctico  $LL(1)$  puede emplear el mismo algoritmo; si se desea obtener un analizador sintáctico para otro lenguaje, basta con sustituir la tabla de análisis sintáctico por una nueva. Para subrayar este punto, concluimos observando que la combinación del segmento de programa de la figura 2.30 con la tabla de análisis sintáctico de la figura 2.31 produce un analizador sintáctico para el lenguaje generado por la gramática de la figura 2.7.

## Ejercicios

1. Reescriba el segmento de programa de la figura 2.30 utilizando una estructura `repeat ... until` en vez de una estructura `while`.
2. Traduzca el diagrama de transiciones que se muestra a continuación directamente a un segmento de programa que analice el lenguaje en cuestión. ¿Cuáles son las incertidumbres que deben resolverse para obtener una rutina determinista?



3. Diseñe una tabla de análisis sintáctico  $LL(1)$  para la gramática siguiente.

$$\begin{aligned} S &\rightarrow xSz \\ S &\rightarrow ySz \\ S &\rightarrow \lambda \end{aligned}$$

4. ¿Cuántos símbolos de preanálisis requeriría un analizador sintáctico  $LL$  al analizar la sintaxis de cadenas basadas en la gramática siguiente? Diseñe una tabla de análisis sintáctico correspondiente.

$$\begin{aligned} S &\rightarrow xSy \\ S &\rightarrow xy \end{aligned}$$

## 2.5 ANALIZADORES SINTÁCTICOS $LR(k)$

En la sección anterior mencionamos que la naturaleza predictiva de los analizadores sintácticos  $LL(k)$  restringe la clase de lenguajes que pueden manejar estos analizadores. En esta sección presentamos una clase de analizadores sintácticos que evitan muchos de los problemas relacionados con sus homólogos predictivos. Estos analizadores se conocen como **analizadores sintácticos  $LR(k)$** , ya que leen su entrada de izquierda a derecha (*Left to right*, en inglés) mientras construyen una derivación por la derecha (*Right derivation*, en inglés) de sus cadenas de entrada utilizando un sistema de preanálisis que comprende  $k$  símbolos.

### Proceso de análisis sintáctico $LR$

En términos generales, un analizador sintáctico  $LR$  transfiere símbolos de su entrada a la pila hasta que los símbolos superiores de la pila sean iguales al lado

derecho de alguna regla de reescritura de la gramática en que se basa el analizador. Al llegar a este punto, el analizador sintáctico puede reemplazar estos símbolos con el no terminal que se encuentra en el lado izquierdo de la regla de reescritura antes de transferir otros símbolos de la entrada a la pila. De esta manera, la pila acumula cadenas de terminales y no terminales, que a su vez son reemplazadas por no terminales “más altos” de la gramática. Por último, todo el contenido de la pila se reduce al símbolo inicial de la gramática, indicando que los símbolos leídos hasta ese punto forman una cadena que puede derivarse con la gramática.

Con base en este esquema general, los analizadores sintácticos  $LR(k)$  se clasifican como **analizadores sintácticos ascendentes**, ya que sus actividades corresponden a la construcción de ocurrencias de no terminales a partir de sus componentes, hasta generar el símbolo inicial de la gramática. En comparación, los analizadores sintácticos  $LL(k)$  se conocen como **analizadores sintácticos descendentes** pues comienzan con el símbolo inicial en la pila y repetidamente dividen los no terminales de la pila en sus componentes, hasta generar una cadena de símbolos equivalente a la cadena de entrada.

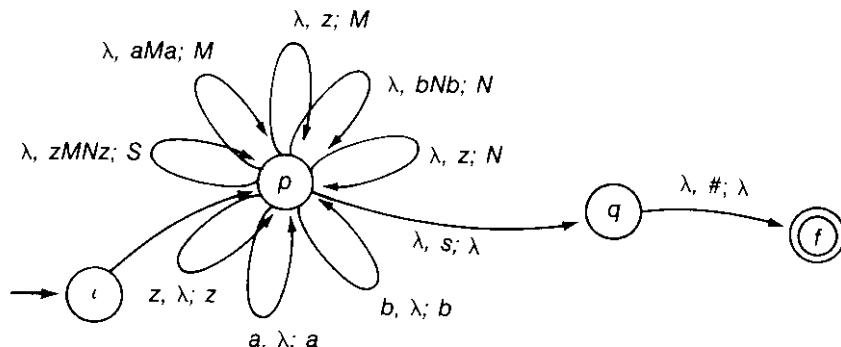
Demos marcha atrás por un momento y desarrollemos los detalles específicos de los analizadores sintácticos  $LR(k)$ . Recuerde que un analizador  $LL(k)$  se basa en un autómata de pila construido a partir de una gramática independiente del contexto; esta construcción se basa en el proceso descrito en la demostración del teorema 2.2. De manera similar, un analizador sintáctico  $LR(k)$  se basa en un autómata de pila construido a partir de una gramática independiente del contexto, con la excepción de que el autómata se construye de la manera descrita en los cinco pasos siguientes.

1. Establezca cuatro estados: uno inicial llamado  $i$ , un estado de aceptación llamado  $f$  y otros dos estados llamados  $p$  y  $q$ .
2. Introduzca las transiciones  $(i, \lambda, \lambda; p, \#)$  y  $(q, \lambda, \#; f, \lambda)$ , donde suponemos que  $\#$  es un símbolo que no se presenta en esta gramática.
3. Para cada símbolo terminal  $x$  en la gramática, introduzca la transición  $(p, x, \lambda; p, x)$ . Estas transiciones permiten que el autómata transfiera a la pila los símbolos de entrada mientras se encuentra en el estado  $p$ . La ejecución de esta transición se llama **operación de desplazamiento** ya que su efecto es desplazar un símbolo de la cadena de entrada a la pila.
4. Para cada regla de reescritura  $N \rightarrow w$  (donde  $w$  representa una cadena de uno o más símbolos) que exista en la gramática, introduzca la transición  $(p, \lambda, w; p, N)$ . (Aquí permitimos que una transición elimine más de un símbolo de la pila. Para ser más precisos, la transición  $(p, i, xy; p, z)$  es una forma abreviada de transición  $(p, \lambda, y; p, \lambda)$  seguida por  $(p, \lambda, x; p, z)$ , donde  $p$  es un estado al que no puede llegar ninguna otra transición. Así, para ejecutar la transición  $(p, \lambda, xy; p, z)$ , un autómata debe tener una  $y$  en la cima de la pila con una  $x$  inmediatamente debajo). La presencia de estas transiciones significa que si los símbo-

los de la porción superior de la pila concuerden con los del lado derecho de la regla de reescritura, entonces es posible reemplazar esos símbolos con el no terminal de la parte izquierda de esa regla. La ejecución de esta transición se llama **operación de reducción** ya que su efecto es reducir el contenido de la pila a una forma más simple.

- Introduzca la transición  $(p, \lambda, S; q, \lambda)$ , donde  $S$  es el símbolo inicial de la gramática. La presencia de esta transición significa que si se han reducido a  $S$  los símbolos de la pila, el autómata puede pasar al estado  $q$  a la vez que extrae  $S$  de la pila.

Como ejemplo de esta construcción, considere el diagrama de la figura 2.32, el cual se construyó a partir de la gramática de la figura 2.5. Un autómata de pila basado en este diagrama de transiciones analizaría la cadena *zazabzbz* tal como se muestra en el resumen de la figura 2.33. Comenzaría por marcar el fondo de la pila con el símbolo  $\#$  y pasar al estado  $p$ , teniendo la cadena *zazabzbz* en la entrada, como lo indica la primera fila de la figura. A partir de esta configuración, el autómata desplazaría la primera *z* de la entrada a la pila a través de la transición  $(p, z, \lambda; p, z)$ , para llegar a la configuración que se presenta en la segunda fila de la figura 2.33. En esta etapa, el autómata tendría la opción de ejecutar  $(p, \lambda, z; p, M)$ ,  $(p, \lambda, z; p, N)$  o  $(p, a, \lambda; p, a)$ . (Este autómata es no determinista.) En nuestro ejemplo, las primeras dos opciones representan decisiones incorrectas; deben tomarse si hay que analizar la *z* en la cima de la pila como un refinamiento del no terminal *M* o *N*, respectivamente. La tercera opción es la correcta en nuestro caso, pues hay que leer más de la cadena antes de ejecutar la operación de reducción.



**Figura 2.32** Otro diagrama de transiciones para un autómata de pila basado en la gramática de la figura 2.5

(Números de fila)	Estado actual	Contenido de la pila	Resto de la entrada
1	$p$	#	<i>zazabzbz</i>
2	$p$	# <i>z</i>	<i>azabzbz</i>
3	$p$	# <i>za</i>	<i>abzbz</i>
4	$p$	# <i>zaz</i>	
5	$p$	# <i>zaM</i>	<i>abzbz</i>
6	$p$	# <i>zaMa</i>	<i>bzbz</i>
7	$p$	# <i>zM</i>	<i>bzbz</i>
8	$p$	# <i>zMb</i>	<i>bz</i>
9	$p$	# <i>zMbz</i>	<i>bz</i>
10	$p$	# <i>zMbN</i>	<i>bz</i>
11	$p$	# <i>zMbNb</i>	<i>z</i>
12	$p$	# <i>zMN</i>	<i>z</i>
13	$p$	# <i>zMNz</i>	
14	$p$	# <i>S</i>	
15	$q$	#	
16	$f$	vacío	

**Figura 2.33** Análisis completo de la cadena *zazabzbz* efectuado por el autómata de la figura 2.32

Puesto que el objetivo es mostrar cómo puede aceptar el autómata la cadena *zazabzbz*, seguimos esta última opción. De hecho, el autómata debe continuar con la ejecución de operaciones de desplazamiento hasta que llegue a la configuración representada por la cuarta fila de la figura 2.33. En esta etapa, la acción correcta es ejecutar la transición  $(p, \lambda, z; p, M)$ , reconociendo que la *z* en la cima de la pila se deriva de una ocurrencia del no terminal *M*. Después de esta transición, el autómata debe ejecutar la operación de desplazamiento  $(p, a, \lambda; p, a)$  y llegar a la configuración representada por la fila seis de nuestra figura. Aquí el autómata reconoce los símbolos *aMa* como una cadena que puede derivarse del no terminal *M*, y por lo tanto reduce su pila por medio de la transición  $(p, \lambda, aMa; p, M)$  antes de proseguir con otras operaciones de desplazamiento.

Continuando así, el autómata finalmente desplazará la última *z* de su entrada a la pila (fila 13 de la Fig. 2.33); reconocerá que el contenido de la pila, *zMNz*, es una cadena que se puede derivar del símbolo de inicio *S*, reducirá su pila por medio de la transición  $(p, \lambda, zMNz; p, S)$  (fila 14); y por fin extraerá el no terminal *S* al pasar al estado *q* (fila 15). A partir de aquí, la máquina extrae el símbolo # de la pila y pasa al estado de aceptación *f*.

Como un asunto secundario, ahora podemos justificar la *R* del término analizador sintáctico *LR(k)*, la cual, según dijimos, indica que en estos analizadores del análisis de las entradas se lleva a cabo construyendo derivaciones por la derecha. Recuerde que una derivación es una

secuencia de reglas de reescritura que transforma el símbolo inicial en la cadena derivada. Sin embargo, el proceso ascendente, cuyo resumen se presenta en la figura 2.33, lo hace a la inversa: genera el símbolo inicial a partir de la cadena derivada. Por consiguiente, la derivación implicada aparece en orden inverso. Para encontrarla, leemos en forma ascendente la columna "contenido de la pila" de la figura 2.33 a la vez que registramos las reglas de reescritura que aplicó el autómata. En nuestro ejemplo, esto revela la derivación

$$S \Rightarrow zMNz \Rightarrow zMbNbz \Rightarrow zMbzbz \Rightarrow zaMabzbz \Rightarrow zazabzbz$$

la cual, como se dijo, es una derivación por la derecha.

### Implantación de analizadores sintácticos $LR(k)$

Se presentan dos problemas principales al tratar de convertir los autómatas de pila, como el que se muestra en la figura 2.32, a un formato de programa más tradicional. El primero tiene que ver con el no determinismo, de manera similar a lo que sucede con los analizadores sintácticos  $LL$ : si se presenta una opción, ¿cómo saber si debemos desplazar o reducir? Además, si nuestra opción es reducir, puede existir más de una reducción posible (si  $z$  está en la cima de la pila, ¿reducimos con  $(p, \lambda, z; p, M)$  o con  $(p, \lambda, z; p, N)$ ? Como podrá suponer, estos asuntos se resuelven con la aplicación del principio de preanálisis.

El segundo problema tiene que ver con los aspectos técnicos de la interrogación de la pila. Por ejemplo, antes de que podamos decidir si ejecutamos la transición  $(p, \lambda, aMa; p, M)$ , debemos ser capaces de deducir que los tres símbolos superiores de la pila son  $a$ ,  $M$  y  $a$ . Sin embargo, en un momento determinado sólo está disponible para observación el símbolo que se encuentra en la cima de la pila. Éste parece ser un problema que puede solucionarse si implantamos la pila como una estructura híbrida que permite observar los símbolos ubicados debajo de la cima de la pila, pero esta modificación soslaya el problema real: la necesidad de realizar búsquedas repetidas en la pila. De hecho, parece que cualquier implantación de una rutina de análisis sintáctico  $LR$  incluiría un importante componente de reconocimiento de patrones para comparar el contenido de la pila con los lados derechos de las reglas de reescritura.

Quizás el resultado más fascinante en el campo de la construcción de compiladores resida en que este problema de interrogación de la pila se pueda resolver sin necesidad de llevar a cabo costosas búsquedas o siquiera recurrir a estructuras híbridas para la pila. De hecho, en el caso de los lenguajes deterministas independientes del contexto, puede integrarse en una sola tabla de análisis sintáctico toda la información necesaria para resolver este problema de interrogación de la pila, así como el problema de elección de opciones.

En la figura 2.34 se muestra el ejemplo de una tabla para un analizador sintáctico  $LR(1)$  basado en la gramática de la figura 2.5. Las columnas de esta tabla están rotuladas con los símbolos de la gramática (incluyendo tanto terminales como no terminales) junto con una marca de fin de cadena (representada por FDC). Las filas se etiquetan con números que representan símbolos (componentes léxicos) especiales (veremos que estos símbolos especiales se utilizan para representar patrones que pueden aparecer en la pila).

Para describir los elementos de la tabla, consideraremos el proceso donde se usa la tabla. El análisis sintáctico de cualquier cadena comienza asignando el valor uno a una variable de símbolo especial e insertando este valor en la pila vacía. (Desde este momento, a la inserción de un símbolo terminal o no terminal en la pila le seguirá la inserción sobre él valor actual de la variable de símbolo especial. Esto quiere decir que el contenido de la pila alternará entre símbolos terminales o no terminales y símbolos especiales, donde cada uno de estos últimos representa el patrón de lo que yace debajo de él. Por lo tanto, podemos investi-

	a	b	z	FDC	S	M	N
1			desplazar 2		14		
2	desplazar 3		desplazar 7			4	
3	desplazar 3		desplazar 7			8	
4		desplazar 5	desplazar 9				6
5		desplazar 5	desplazar 9				10
6			desplazar 11				
7	$M \rightarrow z$	$M \rightarrow z$	$M \rightarrow z$				
8	desplazar 12						
9		$N \rightarrow z$	$N \rightarrow z$				
10		desplazar 13					
11				$S \rightarrow zMNz$			
12	$M \rightarrow aMa$	$M \rightarrow aMa$	$M \rightarrow aMa$				
13		$N \rightarrow bNb$	$N \rightarrow bNb$				
14				aceptar			

Figura 2.34 Tabla de análisis sintáctico  $LR(1)$  basada en la gramática de la figura 2.5

gar la estructura interna de la pila observando el símbolo especial de la cima.)

Una vez que el símbolo especial se ha establecido y se ha almacenado en la pila, hacemos referencia a la tabla de análisis sintáctico. La fila que nos interesa está determinada por el símbolo especial actual y la columna por el símbolo de preanálisis. Los casos más sencillos se presentan cuando la casilla correspondiente de la tabla está vacía o contiene la palabra aceptar. En el primer caso se considera que la cadena es inválida y que debe ejecutarse la rutina de error adecuada. En el segundo caso se indica que la cadena leída de la entrada es aceptable y que el proceso de análisis sintáctico debe concluir.

Otra posibilidad es que la casilla de la tabla contenga desplazar, lo cual indica que debe ejecutarse la operación de desplazamiento. En este caso el siguiente símbolo debe leerse (el símbolo de preanálisis) de la entrada y colocarse en la pila; a la variable de símbolo especial se le debe asignar el valor que existe en la casilla de la tabla junto con la operación de desplazamiento y este nuevo valor de símbolo especial debe insertarse en la pila; por último, debe actualizarse el símbolo de preanálisis.

La última posibilidad es que la entrada de la tabla contenga una regla de reescritura de la gramática, lo cual indica que se trata una operación de reducción. El proceso que aquí se requiere implica la sustitución de una cadena de símbolos de la pila (el lado derecho de la regla de reescritura) con un solo no terminal (el lado izquierdo de la regla). Sin embargo, se requiere un poco más de detalle para manejar los valores de los símbolos especiales que también se encuentran almacenados en la pila. En primer lugar hay que eliminar dos símbolos de la pila por cada símbolo del lado derecho de la regla de reescritura. Esto elimina cada uno de los símbolos del lado derecho de la regla, así como el valor de símbolo especial que se encuentra almacenado encima del símbolo. En este punto, la cima de la pila contendrá el valor del símbolo especial que se colocó después de crear la porción inferior (la que queda) de la pila. Hay que recordar este valor como "símbolo especial temporal" e insertar encima el no terminal del lado izquierdo de la regla de reescritura. Luego, este no terminal se emplea para identificar una columna de la tabla de análisis sintáctico, a la vez que el símbolo especial temporal determina una fila. El valor que se encuentra en este lugar en la tabla de análisis sintáctico debe asignarse a la variable de símbolo especial y además debe insertarse en la pila.

Así, al emplear una tabla de análisis sintáctico, el analizador *LR* sencillamente hace referencia de manera cíclica a la tabla hasta encontrar una entrada en blanco o de aceptación. En la figura 2.35 se presenta un algoritmo de análisis sintáctico *LR(1)* que utiliza la tabla de la figura 2.34. La figura 2.36 resume las actividades del analizador durante el procesamiento de la cadena *zazabzbz*. Observe que esta figura es esencialmente igual que la figura 2.33, excepto por los valores adicionales de símbolos especiales en la pila.

```

Símbolo Especial := 1;
insertar (Símbolo Especial);
leer (Símbolo);
Valor Tabla := Tabla [Símbolo Especial, Símbolo];
while Valor Tabla no es "aceptar" do
    if Valor Tabla es un desplazamiento
        then begin
            insertar (Símbolo); Símbolo Especial := Valor Tabla. Estado;
            insertar (Símbolo Especial); leer (Símbolo)
            end
    else if Valor Tabla es una reducción
        then begin
            extraer (lado derecho de Valor Tabla.ReglaReescritura);
            Símbolo Especial := cima-de-la-pila; (* Esto no es una
            extracción *)
            insertar (lado izquierdo de Valor Tabla.ReglaReescritura);
            Símbolo Especial := Tabla [Símbolo Especial
            lado izquierdo de Valor Tabla.ReglaReescritura];
            insertar (Símbolo Especial)
            end
    else if Valor Tabla está en blanco then salir a la rutina de error;
    Valor Tabla := Tabla [Símbolo Especial, Símbolo];
end while;
if Símbolo no es FDC then salir a la rutina de error;
vaciar pila

```

Figura 2.35 Algoritmo de análisis sintáctico  $LR(1)$ 

### Tablas de análisis sintáctico $LR$

Aunque la construcción de una tabla de análisis sintáctico  $LR$  corresponde más a la construcción de compiladores que a la teoría de los lenguajes formales, no debemos ignorar por completo este tema. De hecho, la construcción de estas tablas es una importante aplicación de la teoría de autómatas finitos. La tabla de un analizador sintáctico  $LR(1)$  se basa en la existencia de un autómata finito que acepta exactamente las cadenas de símbolos de la gramática (terminales y no terminales) que conducen a operaciones de reducción. Aquí presentamos un breve ejemplo de esto; en el apéndice A se ofrecen mayores detalles.

El diagrama de la figura 2.37 es el diagrama de transiciones del autómata finito a partir del cual se construyó la tabla de análisis sintáctico de la figura 2.34. Observe que acepta cadenas como  $zaMa$ ,  $zMbNb$  y  $zMNz$ , las cuales, cuando se encuentran en la pila del analizador, deben reducirse a  $zM$ ,  $zMN$  y  $S$ , respectivamente. En términos de este diagrama, el objetivo del analizador sintáctico  $LR$  es llegar al estado 14 recorriendo el arco con etiqueta  $S$ .

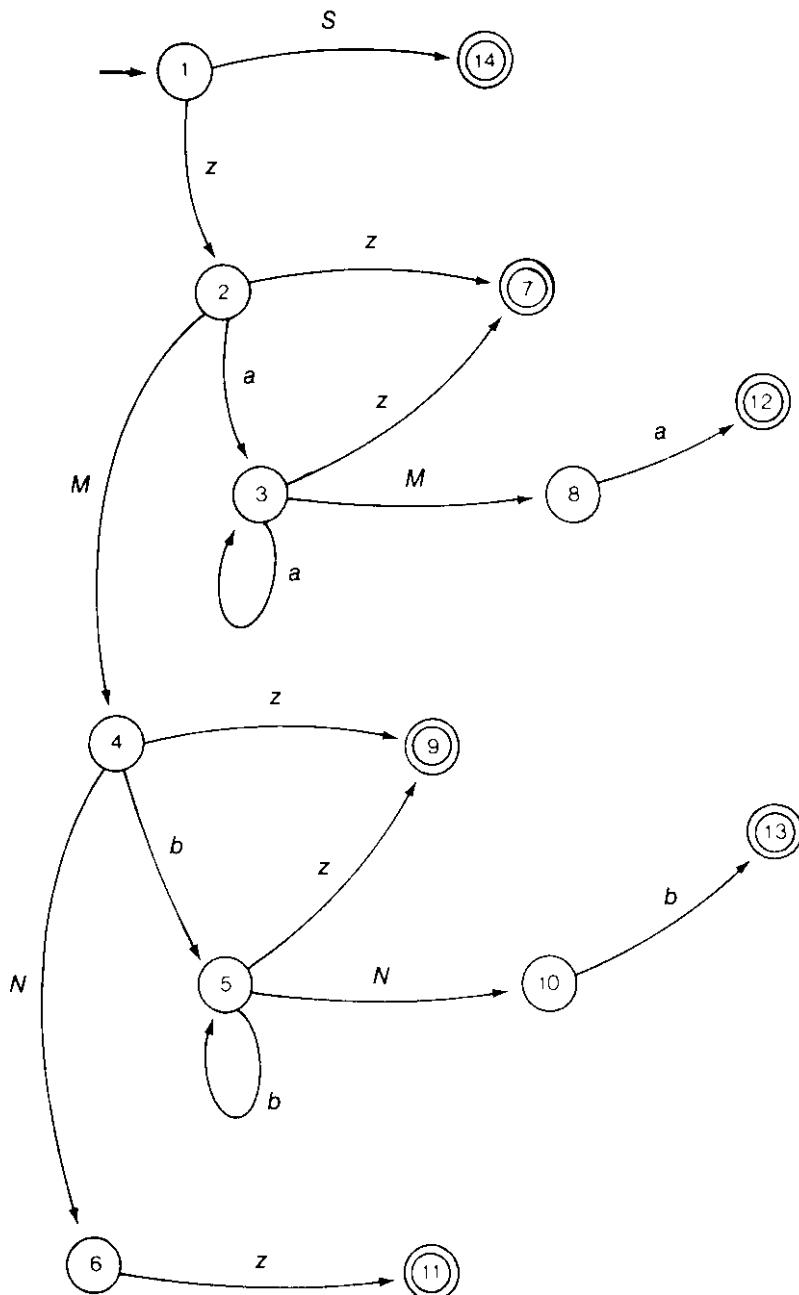
<u>Contenido de la pila</u>	<u>Resto de la entrada</u>
vacía	
①	<i>zazabzbz</i>
① z ②	<i>zazabzbz</i>
① z ② a ③	<i>azabzbz</i>
① z ② a ③ z ⑦	<i>zabzbz</i>
① z ② a ③ M ⑧	<i>abzbz</i>
① z ② a ③ M ⑧ a ⑫	<i>bzbz</i>
① z ② M ④	<i>bzbz</i>
① z ② M ④ b ⑤	<i>zbz</i>
① z ② M ④ b ⑤ z ⑨	<i>bz</i>
① z ② M ④ b ⑤ N ⑩	<i>z</i>
① z ② M ④ b ⑤ N ⑩ b ⑬	<i>z</i>
① z ② M ④ N ⑥	
① z ② M ④ N ⑥ z ⑪	
① S ⑭	
vacía	

**Figura 2.36** Análisis sintáctico de la cadena *zazabzbz* con el algoritmo de la figura 2.35 y la tabla de la figura 2.34

Para lograrlo, ejecuta lo que podría parecer un proceso de prueba y error donde el analizador sigue repetidamente una ruta hacia un estado de aceptación, retrocede a un estado anterior siguiendo la misma ruta en sentido inverso y a partir de allí se encamina hacia una nueva dirección.

No obstante, este proceso no es de prueba y error, sino una secuencia bien definida de eventos guiados por los símbolos que se detectan en la cadena analizada. La idea es comenzar el proceso de análisis sintáctico siguiendo la ruta determinada por la cadena de entrada hasta encontrar un estado de aceptación. Al llegar a este punto, la ruta recorrida por el autómata finito corresponde al patrón de símbolos que el analizador ha desplazado a la pila. Entonces, el proceso de análisis retrocede por esta ruta recorriendo los símbolos que deben eliminarse de la pila del analizador durante la operación de reducción. A partir de este punto, el analizador sintáctico recorre el arco del diagrama de transiciones del autómata finito que tenga etiqueta equivalente al no terminal colocado en la pila por la operación de reducción correspondiente. Así, una vez que se ha completado la operación de reducción, los símbolos en la pila del analizador corresponderán una vez más a los símbolos de la ruta que recorre el autómata finito.

Para ver cómo funciona este proceso, consideremos de nuevo la tarea de análisis sintáctico de la cadena *zazabzbz*. Conforme el analizador lee los



**Figura 2.37** Autómata finito a partir del cual se construyó la tabla de la figura 2.34

símbolos  $zaz$ , el autómata finito se mueve por la secuencia de estados 1, 2, 3 y 7. Al llegar a este punto, se requiere una operación de reducción basada en la regla  $M \rightarrow z$ . Entonces, el autómata retrocede al estado 3 (siguiendo el arco  $z$ ) y del estado 3 pasa al estado 8 por el arco con etiqueta  $M$ . Al mismo tiempo, el analizador sintáctico  $LR$  extrae la  $z$  superior de la pila y la reemplaza con el no terminal  $M$ . Así, la nueva ruta que recorre el autómata finito (a través de los estados 1, 2, 3 y 8) corresponde de nuevo al patrón de símbolos en la pila del analizador. La lectura del siguiente símbolo de la entrada,  $a$ , lleva al autómata finito al estado 12, donde el analizador retrocederá al estado 2 y seguirá el arco con etiqueta  $M$  hasta el estado 4. De esta manera, el proceso de análisis sintáctico finalmente llegará al estado 11 a través de los arcos  $z, M, N$  y  $z$ . De aquí, el proceso de análisis retrocederá al estado 1 y pasará al estado 14 ya que la cadena  $zMNz$  se reduce al símbolo inicial  $S$ .

Tomando en cuenta esta situación, no es difícil comprender la construcción de un analizador sintáctico  $LR(1)$ . Los valores de los símbolos especiales representan los estados del autómata finito. Las casillas de desplazamiento de la tabla corresponden a los arcos rotulados con terminales, mientras que el símbolo especial que se encuentra en esa casilla representa el estado al final del arco. Las casillas de reducción indican que el analizador ha llegado a un estado de aceptación en el autómata finito, y también proporcionan la información necesaria para realizar el proceso de retrocesos. (Observe que el estado de un autómata finito donde deberá detenerse el retroceso está representado por el símbolo especial que aparece en la cima de la pila después de extraer el lado derecho de la regla de reescritura.) Las casillas de las columnas etiquetadas con no terminales permiten al analizador sintáctico establecer una nueva dirección en el diagrama después del retroceso.

Hay que hacer un comentario final antes de abandonar nuestro tratamiento de los analizadores sintácticos  $LR$ . Para completar un compilador es necesario combinar un analizador con un generador de código que produzca instrucciones en el lenguaje objeto que reflejen las estructuras detectadas por el analizador. Al emplear un analizador sintáctico  $LR$ , este enlace entre el analizador y el generador de código se presenta en relación con cada operación de reducción. De hecho, es en esta etapa del proceso de análisis sintáctico donde se ha reconocido una estructura, por lo que es el momento indicado para solicitar al generador de código que construya el código para esa estructura. En resumen, los cálculos efectuados por el analizador sintáctico y el generador de código siguen el patrón básico.

**repeat**

analizar la sintaxis hasta ejecutar una reducción

generar código

**until** termina el análisis sintáctico

## Comparación entre los analizadores sintácticos $LR(k)$ y $LL(k)$

Para concluir, debemos confirmar que la colección de analizadores sintácticos  $LR(k)$  es más poderosa que la de los analizadores  $LL(k)$ . Ya hemos planteado que ningún analizador sintáctico  $LL(k)$  puede manejar el lenguaje  $\{x^n: n \in \mathbb{N}\} \cup \{x^n y^n: n \in \mathbb{N}\}$ . Sin embargo, en la figura 2.38 se presenta una gramática para este lenguaje y su tabla de análisis sintáctico  $LR(1)$  correspondiente. Si se combina esta tabla con el algoritmo de la figura 2.35, se obtiene un analizador  $LR(1)$  que reconoce el lenguaje (esto se logra desplazando las  $x$  de la entrada a la pila hasta alcanzar una  $y$  o una marca de fin de cadena. En ese momento, el analizador es capaz de aplicar las reglas de reescritura apropiadas para analizar correctamente la cadena).

Existen, no obstante, lenguajes independientes del contexto que ningún analizador sintáctico  $LR(k)$  puede reconocer. De hecho, la clase de lenguajes

	$x$	$y$	FDC	$S$	$X$	$Y$
1	desplazar 2		$S \rightarrow \lambda$	9	8	7
2	desplazar 2	desplazar 6	$X \rightarrow x$		3	4
3			$X \rightarrow xX$			
4		desplazar 5				
5		$Y \rightarrow xXy$				
6		$Y \rightarrow xy$				
7						
8						
9			aceptar			

Figura 2.38 Gramática independiente del contexto y su tabla de análisis sintáctico  $LR(1)$

que pueden ser analizados por los analizadores  $LR(k)$  es precisamente la clase de los lenguajes independientes del contexto deterministas. Aunque no lo demostraremos, por lo menos debemos señalar que este límite del poder de los analizadores  $LR(k)$  está de acuerdo con nuestra intuición: un analizador sintáctico  $LR(k)$  debe ser determinista y, puesto que su estructura se basa en un autómata de pila, debe deducirse que los analizadores sintácticos  $LR(k)$  sólo pueden analizar aquellos lenguajes que aceptan los autómatas de pila deterministas.

Un ejemplo de lenguaje independiente del contexto que un analizador sintáctico  $LR(k)$  no puede analizar el lenguaje

$$\{x^n y^n : n \in \mathbb{N}^+\} \cup \{x^n y^{2n} : n \in \mathbb{N}^+\}$$

Intuitivamente, el problema aquí es que una vez que se llega a la primera  $y$  de la entrada, el analizador sintáctico debe decidir cuál es la regla de reescritura que debe aplicarse conociendo sólo los siguientes  $k$  símbolos de la cadena. Si  $n$  es mayor que  $k$ , conocer sólo los siguientes  $k$  símbolos no basta para que el analizador detecte si la entrada contendrá  $n$  o  $2n$   $y$ , y por lo tanto el analizador es incapaz de seleccionar la regla de reescritura correcta.

## Ejercicios

1. Con el proceso alternativo descrito en esta sección para construir un autómata de pila a partir de una gramática independiente del contexto, construya un diagrama de transiciones para un autómata de pila utilizando la gramática que se presenta a continuación.

$$S \rightarrow xSz$$

$$S \rightarrow ySz$$

$$S \rightarrow \lambda$$

2. Identifique la derivación obtenida por el autómata de pila construido en el ejercicio 1 al analizar la cadena  $yxyzzz$ .
3. Construya una tabla de análisis  $LR(1)$  para la gramática siguiente (existen procesos algorítmicos que hacen esto y que no hemos analizado [para obtener más información consulte el apéndice A], pero esta gramática es lo suficientemente sencilla para que pueda desarrollarse una tabla con sólo saber cómo se usan las tablas de análisis sintáctico  $LR(1)$ ).

$$S \rightarrow xSy$$

$$S \rightarrow \lambda$$

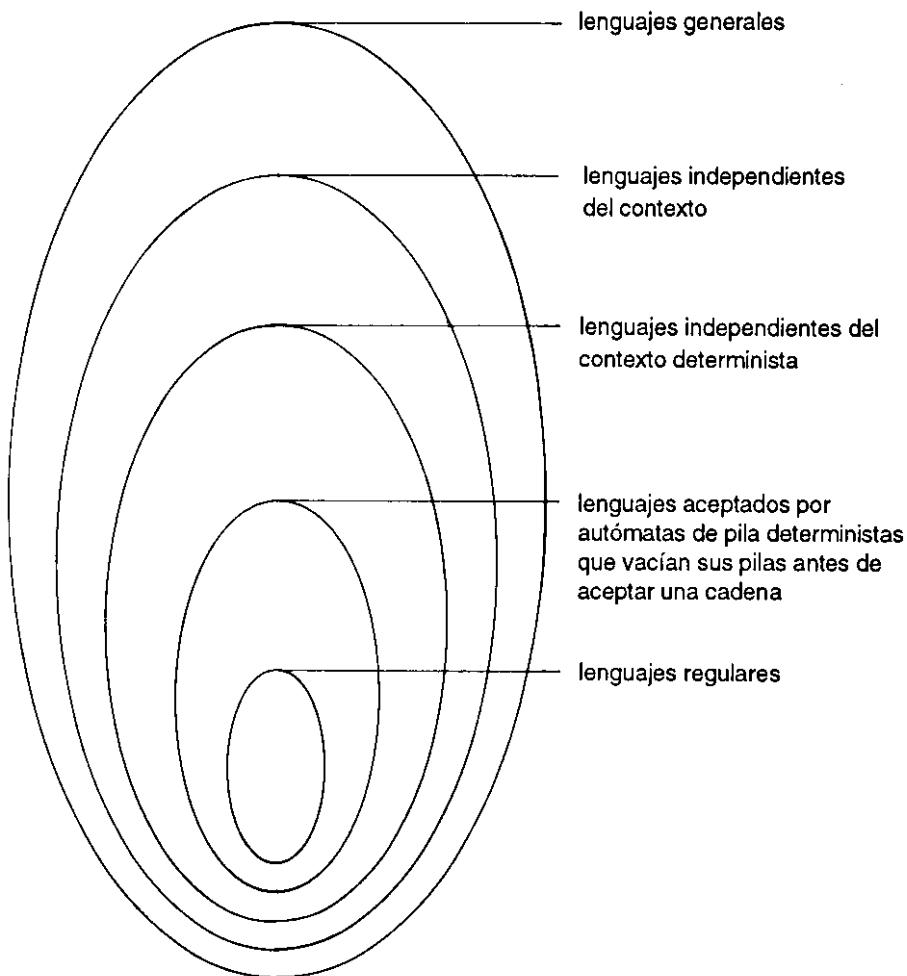
## 2.6 COMENTARIOS FINALES

Primero debemos aclarar la relación entre la jerarquía asociada con los lenguajes independientes del contexto y la clase de lenguajes regulares presentados en el capítulo anterior. Ya hemos visto que los lenguajes regulares son independientes del contexto, pero la clasificación puede ser más precisa. Los lenguajes regulares están propiamente contenidos en la clase de los lenguajes aceptados por los autómatas de pila deterministas que vacían sus pilas antes de aceptar una cadena. Esto es evidente si se observa que cualquier lenguaje regular puede ser aceptado por un autómata finito determinista, que es en esencia un autómata de pila determinista cuyas transiciones nunca utilizan la pila, y como la pila nunca se emplea, debe estar vacía cuando se llega a un estado de aceptación. Además, la inclusión es propia puesto que el lenguaje  $\{x^n y^n : n \in \mathbb{N}\}$  no es regular pero sí aceptado por un autómata de pila determinista que vacía su pila antes de aceptar una cadena (Fig. 2.2).

Así, en la figura 2.39, una ampliación de la figura 2.23, podemos resumir lo que hasta ahora ha cubierto nuestro estudio de los lenguajes.

Por último, como primer paso para introducir algunos de los temas de los capítulos restantes, observe que los lenguajes independientes del contexto no son cerrados para la intersección. Por ejemplo, los lenguajes  $\{x^m y^n z^m : m, n \in \mathbb{N}\}$  y  $\{x^m y^n z^n : m, n \in \mathbb{N}\}$  son independientes del contexto (el primero es generado por  $S \rightarrow TZ, T \rightarrow \lambda, T \rightarrow xTy, Z \rightarrow \lambda, Z \rightarrow zZ$ , y el segundo es similar), pero su intersección es el lenguaje  $\{x^n y^n z^n : m, n \in \mathbb{N}\}$ , el cual, como hemos visto, no es independiente del contexto. Al combinar esto con el hecho de que los lenguajes independientes del contexto son cerrados para la unión finita, y aplicar las leyes de DeMorgan, resulta que los lenguajes independientes del contexto no son cerrados para la complementación. Es decir, existen subconjuntos de  $\Sigma^*$  que son independientes del contexto pero cuyos complementos en  $\Sigma^*$  no lo son.

Esto significa que la capacidad de un autómata de pila para aceptar cadenas de un lenguaje no es simétrica con la capacidad para rechazar las cadenas que no se encuentran en el lenguaje (la capacidad para rechazar las cadenas que no están en el lenguaje equivaldría a la capacidad para aceptar el complemento del lenguaje). Así, existe una importante diferencia entre la capacidad para responder sí cuando la cadena está en el lenguaje y la capacidad para responder sí o no cuando la cadena está o no está en el lenguaje (en el primer caso, si la entrada no se halla en el lenguaje, la máquina puede quedar atrapada en un ciclo y nunca responder. En el segundo caso, la máquina debe, a fin de cuentas, responder de manera correcta sin importar si la respuesta es sí o no). La carencia de simetría será un factor importante en análisis subsecuentes.

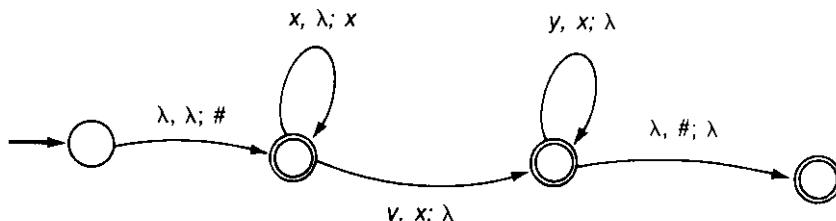


**Figura 2.39** La jerarquía de los lenguajes que hemos presentado hasta ahora

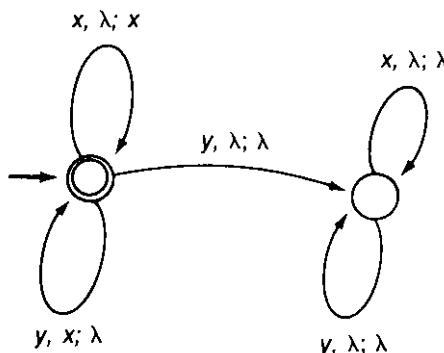
## Problemas de repaso del capítulo

1. Para cada nivel en la jerarquía de la figura 2.39, proporcione un ejemplo de un lenguaje que exista en ese nivel pero no en el nivel inmediato inferior.
2. Muestre que el lenguaje  $\{x^r y^s z^t : s = r + t\}$  es independiente del contexto.
3. Muestre que:
  - a. El lenguaje  $\{x^m y^n : m, n \in \mathbb{N}^*\}$  es regular.

- b. El lenguaje  $\{x^m y^n x^m : m, n \in \mathbb{N}^*\}$  es independiente del contexto pero no regular.
  - c. El lenguaje  $\{x^m y^n x^m y^n : m, n \in \mathbb{N}^*\}$  no es independiente del contexto.
4. Muestre que el lenguaje del alfabeto  $\{x, y\}$  que consiste en aquellas cadenas con el mismo número de  $x$  y  $y$  es independiente del contexto determinista.
5. Muestre que si  $L$  es un lenguaje independiente del contexto, entonces el lenguaje que consiste en las cadenas de  $L$  escritas a la inversa también es independiente del contexto.
6. Con base en la gramática de la figura 2.5, dibuje el árbol de análisis sintáctico para la cadena  $zzbbzbbz$ . ¿Cuántas derivaciones distintas son posibles para esta cadena? Escriba las derivaciones por la izquierda y por la derecha.
7. Construya un autómata de pila  $M$  para el cual  $L(M) = \{w^r x^t y^s z^u : r, s, t$  y  $u$  son enteros no negativos tales que  $r + t = s + u\}$ .
8. Describa el lenguaje aceptado por el autómata de pila cuyo diagrama de transiciones se muestra a continuación.



9. Describa el lenguaje que acepta el autómata de pila cuyo diagrama de transiciones se muestra a continuación. Describa las cadenas que se aceptan cuando la pila de la máquina no está vacía. Modifique el diagrama para que acepte las mismas cadenas que antes pero sólo después de vaciar la pila.



10. Encuentre una gramática independiente del contexto para el lenguaje  $\{x^m y^n : m \text{ y } n \text{ son enteros positivos tales que } m = n \text{ o } m = 2n\}$ .
11. Desarrolle una gramática independiente del contexto que describa la estructura de un lenguaje de programación semejante a Pascal que sólo permite variables de tipo entero o real, que no contiene procedimientos o funciones y cuyas únicas instrucciones son asignaciones y enunciados while.
12. Diseñe una tabla de análisis sintáctico  $LL(1)$  para la gramática

$$\begin{aligned} S &\rightarrow xSz \\ S &\rightarrow y \end{aligned}$$

13. Diseñe una tabla de análisis sintáctico  $LL(1)$  para la gramática

$$\begin{aligned} S &\rightarrow xSy \\ S &\rightarrow y \end{aligned}$$

14. Diseñe una tabla de análisis sintáctico  $ID(1)$  para la gramática

$$\begin{aligned} S &\rightarrow xSz \\ S &\rightarrow y \end{aligned}$$

15. Diseñe una tabla de análisis sintáctico  $LR(1)$  para la gramática

$$\begin{aligned} S &\rightarrow xSy \\ S &\rightarrow y \end{aligned}$$

16. Diseñe un autómata de pila determinista  $M$  para el cual  $L(M)$  sea el lenguaje generado por la gramática

$$\begin{aligned} S &\rightarrow xSx \\ S &\rightarrow y \end{aligned}$$

17. Diseñe un autómata de pila determinista  $M$  para el cual  $L(M)$  sea el lenguaje generado por la gramática

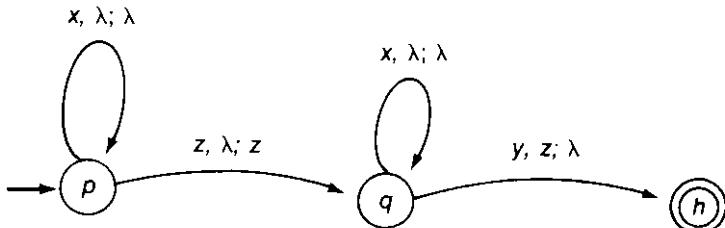
$$\begin{aligned} S &\rightarrow xSy \\ S &\rightarrow y \end{aligned}$$

18. Diseñe un autómata de pila determinista  $M$  para el cual  $L(M)$  sea el lenguaje generado por la gramática siguiente, con símbolo inicial  $S$ .

$$\begin{aligned}S &\rightarrow xyN \\N &\rightarrow zS \\N &\rightarrow z \\N &\rightarrow \lambda\end{aligned}$$

19. Sea  $\Sigma = \{x, y, \dots, \alpha, \beta\}, (\cdot, \Delta)$ . Diseñe un autómata de pila  $M$  tal que  $L(M)$  sea el conjunto de todas las cadenas que contituyan expresiones regulares del alfabeto  $\{x, y\}$ .
20. Construya una gramática  $G$  independiente del contexto tal que  $L(G)$  sea el conjunto de todas las cadenas formadas por la concatenación de una cadena en  $\{x, y\}^*$  con la misma cadena escrita a la inversa.
21. Muestre que un lenguaje que consiste en los palíndromos de  $\{x, y\}^*$  es independiente del contexto (un palíndromo es una cadena que es idéntica a sí misma escrita a la inversa).
22. a. Diseñe una gramática independiente del contexto que genere el lenguaje  $\{x^m y^n : m \text{ y } n \text{ son enteros no negativos donde } n < m\}$ .  
b. Diseñe una gramática independiente del contexto que genere el lenguaje  $\{x^m y^n : m \text{ y } n \text{ son enteros no negativos donde } n > m\}$ .
23. a. Diseñe un autómata de pila  $M$  tal que  $L(M) = \{x^m y^n : m \text{ y } n \text{ son enteros no negativos y } m \leq n \leq 2m\}$ .  
b. Diseñe un autómata de pila  $M$  tal que  $L(M) = \{x^m y^n : m \text{ y } n \text{ son enteros no negativos y } n < m \text{ o } 2m < n\}$ .
24. Muestre que el lenguaje  $\{x^n : n \text{ es un entero positivo primo}\}$  no es independiente del contexto.
25. Muestre que el lenguaje  $\{x^n : n = m^2 \text{ para una } m \in \mathbb{N}\}$  no es independiente del contexto.
26. Muestre que el lenguaje  $\{x^n y^n z x^n y^n : n \in \mathbb{N}\}$  no es independiente del contexto.
27. ¿Es el lenguaje  $\{x^m y^n : m, n \in \mathbb{N} \text{ y } m \leq n \leq 2m\}$  independiente del contexto? Justifique su respuesta.
28. ¿Es siempre independiente del contexto la unión de una colección de lenguajes independientes del contexto? Justifique su respuesta.
29. Utilizando un argumento de cardinalidad, muestre que deben existir lenguajes que no son independientes del contexto.

30. Aplique la técnica utilizada en la demostración del teorema 2.3 para construir una gramática independiente del contexto que genere el lenguaje aceptado por el autómata de pila descrito a continuación.



31. Aplique la técnica utilizada en la demostración del teorema 2.3 para construir una gramática independiente del contexto que genere el lenguaje aceptado por el autómata de pila descrito en la figura 2.2.
32. Convierta la gramática siguiente, con símbolo inicial  $S$ , a la forma normal de Chomsky.

$$\begin{aligned} S &\rightarrow MzN \\ M &\rightarrow xM \\ M &\rightarrow \lambda \\ N &\rightarrow yN \\ N &\rightarrow \lambda \end{aligned}$$

33. Encuentre una gramática independiente del contexto en forma normal de Chomsky que genere el mismo lenguaje que la gramática de la figura 2.5.
34. ¿Por qué se preferiría utilizar la colección de reglas de reescritura

$$\begin{aligned} X &\rightarrow xY \\ Y &\rightarrow yZ \\ Z &\rightarrow z \\ Z &\rightarrow \lambda \end{aligned}$$

en vez de

$$\begin{aligned} X &\rightarrow xyz \\ X &\rightarrow x \end{aligned}$$

al construir un analizador sintáctico?

35. Muestre que si  $L$  es un lenguaje independiente del contexto que no contiene la cadena vacía, entonces existe una gramática  $G$  independiente

del contexto en la cual el lado derecho de cada regla de reescritura consiste en un terminal seguido por cero o más no terminales (se dice que estas gramáticas tienen la forma normal de Chomsky).

36. Muestre que la intersección de un lenguaje regular y un lenguaje independiente del contexto siempre es independiente del contexto.
37. Encuentre una gramática independiente del contexto para el lenguaje de  $\{x, y\}$  que consiste en las cadenas en las cuales la relación entre el número de  $x$  y el número de  $y$  es de tres a dos.

## Problemas de programación

---

1. Realice el algoritmo de análisis sintáctico  $LL(1)$  que se muestra en la figura 2.30. Aplique los resultados a las tablas de análisis sintáctico de las figuras 2.29 y 2.31, así como a su respuesta al problema 13 de repaso del capítulo.
2. Realice el algoritmo de análisis sintáctico  $LR(1)$  que se muestra en la figura 2.35. Aplique los resultados a las tablas de análisis sintáctico de las figuras 2.34 y 2.38 así como a su respuesta al problema 15 de repaso del capítulo.
3. Escriba un programa para convertir gramáticas independientes del contexto que no generan la cadena vacía en gramáticas con la forma normal de Chomsky.

## CAPÍTULO 3

# Máquinas de Turing y lenguajes estructurados por frases

---

### 3.1 Máquinas de Turing

Propiedades básicas de las máquinas de Turing

Los orígenes de las máquinas de Turing

### 3.2 Construcción modular de máquinas de Turing

Combinación de máquinas de Turing

Bloques de construcción básicos

### 3.3 Máquinas de Turing como aceptadores de lenguajes

Procedimientos de evaluación de cadenas

Máquinas de Turing de varias cintas

Máquinas de Turing no deterministas

### 3.4 Lenguajes aceptados por máquinas de Turing

Comparación entre lenguajes aceptados por máquinas de Turing y lenguajes estructurados por frases

Alcance de los lenguajes estructurados por frases

### 3.5 Más allá de los lenguajes estructurados por frases

Sistema de codificación de máquinas de Turing

Un lenguaje no estructurado por frases

Máquinas de Turing universales

Comparación entre lenguajes aceptables y decidibles

El problema de la parada

### 3.6 Comentarios finales

Hasta ahora hemos presentado las clases de autómatas finitos y de pila. En ambos casos nuestro objetivo ha sido comprender el potencial de reconocimiento de lenguajes de estas máquinas teóricas. En este capítulo presentaremos otra clase de autómatas, todavía más generales, conocidos como máquinas de Turing, y estudiaremos el poder computacional de

estas máquinas en el contexto de la solución de problemas de reconocimiento de lenguajes.

Uno de los principales resultados será que las máquinas de Turing son capaces de aceptar exactamente los mismos lenguajes que pueden generar las gramáticas menos restrictivas: las gramáticas estructuradas por frases y así, el poder de procesamiento de lenguajes de esta clase de máquinas está limitado por las mismas fronteras que delimitan los poderes generativos de las gramáticas. La importancia de esta equivalencia aumenta por el hecho de que nadie ha podido definir una clase de máquinas computacionales más poderosas que las máquinas de Turing. De hecho, aprenderemos que en la actualidad los científicos de la computación aceptan de manera general la conjetura de que la clase de máquinas de Turing encierra el poder de cualquier proceso computacional (esta conjetura se conoce como tesis de Turing). Por esto, se cree que la correspondencia entre el potencial de reconocimiento de lenguajes de las máquinas de Turing y el poder generativo de las gramáticas que veremos en este capítulo define los límites máximos de cualquier sistema computacional de reconocimiento de lenguajes.

En resumen, la presentación de las máquinas de Turing en este capítulo servirá como culminación de nuestro estudio de las gramáticas, los lenguajes y los algoritmos de análisis sintáctico. Empero, esto no quiere decir que habrá concluido todo nuestro estudio; más bien, las limitaciones aparentes de los poderes computacionales que descubriremos en este capítulo serán el punto de partida para el estudio de la computabilidad.

### 3.1 MÁQUINAS DE TURING

La clase de autómatas que ahora se conoce como **máquinas de Turing** fue propuesta por Alan M. Turing en 1936. (La idea básica de Turing fue estudiar los procesos algorítmicos utilizando un modelo computacional. En ese mismo año, Emil L. Post presentó un enfoque similar, y más tarde se mostró que ambas estrategias son equivalentes en cuanto a poder computacional.) Para nuestros fines es conveniente considerar a las máquinas de Turing como una versión generalizada de los autómatas que hemos visto en capítulos anteriores. Las máquinas de Turing se asemejan a los autómatas finitos en que constan de un mecanismo de control y un flujo de entrada que concebimos como una cinta; la diferencia es que las máquinas de Turing pueden mover sus cabezas de lectura hacia adelante y hacia atrás y pueden leer o escribir en la cinta. Veremos que estas características aumentan en gran medida la capacidad de las máquinas.

#### Propiedades básicas de las máquinas de Turing

Al igual que las demás máquinas que hemos estudiado, la máquina de Turing contiene un mecanismo de control que en cualquier momento puede encontrarse en uno de entre un número finito de estados. Uno de estos estados

se denomina **estado inicial** y representa el estado en el cual la máquina comienza los cálculos. Otro de los estados se conoce como **estado de parada**; una vez que la máquina llega a ese estado, terminan todos los cálculos. De esta manera, el estado de parada de una máquina de Turing difiere de los estados de aceptación de los autómatas finitos y de pila en que éstos pueden continuar sus cálculos después de llegar a un estado de aceptación, mientras que una máquina de Turing debe detenerse en el momento en que llegue a su estado de parada. (nota: con base en la definición anterior, el estado inicial de una máquina de Turing no puede ser a la vez el estado de parada; por lo tanto, toda máquina de Turing debe tener cuando menos dos estados).

Una diferencia más importante entre una máquina de Turing y los autómatas de los capítulos anteriores es que la máquina de Turing puede leer y escribir en su medio de entrada. Para ser más precisos, la máquina de Turing está equipada con una cabeza que puede emplearse para leer y escribir símbolos en la cinta de la máquina (como sucede con los otros autómatas, esta cinta tiene un extremo izquierdo pero se extiende indefinidamente hacia la derecha). Así, una máquina de Turing puede emplear su cinta como almacenamiento auxiliar, de la misma forma en que otra clase de autómatas utiliza su pila. Sin embargo, con este almacenamiento, una máquina de Turing no se limita a las operaciones de inserción y extracción, sino que puede rastrear los datos de la cinta y modificar las celdas que deseé sin alterar las demás.

Al utilizar la cinta para fines de almacenamiento auxiliar, es conveniente que una máquina de Turing emplee marcas especiales para distinguir porciones de la cinta. Para esto, permitimos que una máquina de Turing lea y escriba símbolos que no aparecen en los datos de entrada; es decir, hacemos una distinción entre el conjunto (finito) de símbolos, llamado alfabeto de la máquina, en el que deben estar codificados los datos de entrada iniciales, y un conjunto, posiblemente mayor (también finito), de **símbolos de cinta**, que la máquina puede leer y escribir (esta distinción es similar a la que se establece entre el alfabeto de un autómata y sus símbolos de pila). De esta manera, los símbolos de cinta de una máquina de Turing pueden incluir marcas especiales que no sean símbolos del alfabeto de la máquina.

El espacio en blanco es un símbolo que cae en esta categoría; se trata de un símbolo que se supone está en cualquier celda de la cinta que no esté ocupada. Por ejemplo, si una máquina de Turing leyera más allá de los símbolos de entrada de su cinta, encontraría y leería las celdas en blanco que allí se encuentran. Además, puede ser necesario que una máquina de Turing tenga que borrar una celda, escribiendo en ella un espacio en blanco. Por esto, con frecuencia se considera que el espacio en blanco pertenece al conjunto de símbolos de cinta de la máquina de Turing, pero no forma parte del alfabeto de la máquina.

Es fácil que el símbolo de espacio en blanco ocasione confusiones y malas interpretaciones en una página impresa. Por cuestiones de comunicación, adoptamos el símbolo  $\Delta$  para representar el espacio en blanco. Esta convención elimina la ambigüedad entre las cadenas  $x \Delta y$  y  $x y \Delta$  que podemos expresar la primera como  $x \Delta y$ , y la segunda, como  $x \Delta \Delta y$ .

Las acciones específicas que puede realizar una máquina de Turing consisten en operaciones de escritura y de movimiento. La operación de escritura consiste en reemplazar un símbolo en la cinta con otro símbolo y luego cambiar a un nuevo estado (el cual puede ser el mismo donde se encontraba antes). La operación de movimiento comprende mover la cabeza una celda a la derecha o a la izquierda y luego pasar a un nuevo estado (que, una vez más, puede ser igual al de partida). La acción que se ejecutará en un momento determinado dependerá del símbolo (el símbolo actual) que está en la celda visible en ese momento para la cabeza (la celda actual), así como del estado actual del mecanismo de control de la máquina.

Si representamos con  $\Gamma$  el conjunto de símbolos de cinta de una máquina de Turing, con  $S$ , el conjunto de estados y con  $S'$ , el conjunto de estados que no son el de parada, entonces es posible representar las transiciones de la máquina por medio de una función, llamada **función de transición** de la máquina, de la forma  $\delta: (S' \times \Gamma) \rightarrow S \times (\Gamma \cup \{L, R\})$ , donde suponemos que los símbolos  $L$  y  $R$  no pertenecen a  $\Gamma$ .

La semántica de esta representación funcional es la siguiente:

- $\delta(p, x) = (q, y)$  significa "si el estado actual es  $p$  y el símbolo actual es  $x$ , reemplazar la  $x$  con el símbolo  $y$  y pasar al estado  $q$ ".
- $\delta(p, x) = (q, L)$  significa "si el estado actual es  $p$  y el símbolo actual es  $x$ , mover la cabeza una celda a la izquierda y pasar al estado  $q$ ".
- $\delta(p, x) = (q, R)$  significa "si el estado actual es  $p$  y el símbolo actual es  $x$ , mover la cabeza una celda a la derecha y pasar al estado  $q$ ".

Observe que, al describir con una función las transiciones de una máquina de Turing, la máquina es determinista. Para ser más precisos, existe una, y sólo una, transición asociada a cada par estado-símbolo donde el estado no es el de detención.

Durante la operación normal, una máquina de Turing ejecuta transiciones repetidamente hasta llegar al estado de parada (esto quiere decir que en ciertas condiciones es posible que nunca se detengan los cálculos de una máquina de Turing, ya que su programa interno puede quedar atrapado en un ciclo sin fin). Existe, sin embargo, una anomalía que puede ocurrir durante este proceso: la cabeza de la máquina puede "rebasar" el extremo izquierdo de la cinta. En este caso, la máquina abandonará los cálculos y decimos que la ejecución de la máquina sufrió una terminación anormal.

Como sucede con otros autómatas, es útil representar visualmente una máquina de Turing, como se hace en la figura 3.1. Allí, el mecanismo de control de la máquina está representado por un rectángulo con una especie de carátula de reloj que indica el estado actual de la máquina. Encima de este rectángulo se encuentra la cinta de la máquina; la posición de la cabeza de la máquina está indicada con un apuntador, como se hizo en los autómatas finitos y de pila.

Como también sucede con los otros autómatas, la colección de transiciones de una máquina de Turing se puede representar de manera conveniente por medio de un diagrama de transiciones en el cual se ilustran con pequeños

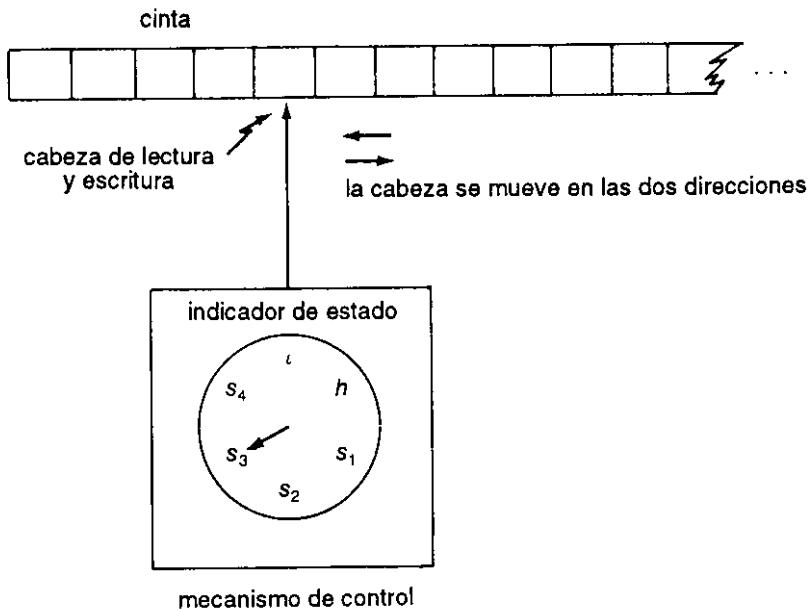
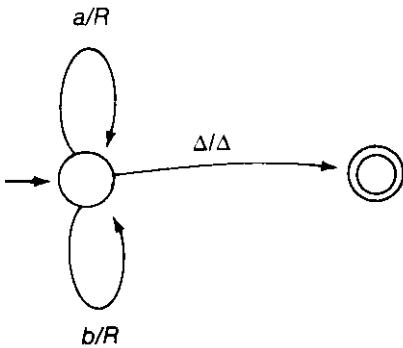


Figura 3. 1 Representación de una máquina de Turing

círculos los estados de la máquina (identificando los estados inicial y de parada con un apuntador y un doble círculo respectivamente) conectados por arcos que representan las transiciones posibles. Cada uno de los arcos del diagrama de transiciones de una máquina de Turing se etiqueta con un par de símbolos separados por una diagonal. El primer símbolo del par representa el símbolo de la cinta que debe existir en la celda actual para que la transición sea aplicable; el segundo símbolo es el que se escribirá en la celda actual (si la transición es una operación de escritura) o de lo contrario uno de los símbolos, *L* o *R* (si la transición es una operación de movimiento). De esta manera, la transición  $\delta(p, x) = (q, y)$  estaría representada por un arco de *p* a *q* con etiqueta *x/y*; o  $\delta(p, x) = (q, L)$  aparecería como un arco de *p* a *q* con etiqueta *x/L*. Como muestra, la figura 3.2 es un diagrama de transiciones completo para una máquina de Turing con símbolos de cinta  $\{a, b, \Delta\}$  que mueve su cabeza hacia la derecha hasta encontrar un espacio en blanco.

Con base en nuestras experiencias con otros autómatas, no debe sorprender que muchas veces resulta difícil manejar y comprender un diagrama de transiciones completo para una máquina de Turing. Por esto, en ocasiones dibujamos esqueletos de los diagramas, que sólo contienen los arcos pertinentes para el análisis, dándose por sentado que, de ser necesario, podrían añadirse los demás arcos.

Como pronto veremos, la "máquina" computacional original de Turing era una computación humana con lápiz y papel. Con la llegada de los equipos de cálculo electrónico, este modelo dio lugar al concepto de máquina electromecánica que lee y escribe en una cinta magnética. Lo importante es que



**Figura 3.2** Diagrama de transiciones para una máquina de Turing

las capacidades computacionales teóricas del sistema son las mismas, sin importar la tecnología con la cual se implanten sus componentes. De hecho, el concepto de máquina de Turing, al igual que los otros autómatas que hemos estudiado, es independiente de su implantación. Entonces, aislaremos las características definitivas de una máquina de Turing en la siguiente definición formal.

Una máquina de Turing es una sextupla de la forma  $(S, \Sigma, \Gamma, \delta, i, h)$ , donde:

- $S$  es una colección finita de estados.
- $\Sigma$  es un conjunto finito de símbolos distintos de espacio en blanco, llamado alfabeto de la máquina.
- $\Gamma$  es un conjunto finito de símbolos, incluidos los de  $\Sigma$ , que se conocen como símbolos de la cinta de la máquina.
- $\delta$  es la función de transición de la máquina.
- $i$  es un elemento de  $S$  llamado estado inicial.
- $h$  es un elemento de  $S$  llamado estado de parada.

En ocasiones es conveniente contar con una notación concisa que representa la configuración de la cinta de una máquina de Turing, incluyendo el contenido de sus celdas y la posición de la cabeza. En estos casos presentaremos la lista del contenido de las celdas de la cinta, subrayando la posición de la cabeza. De esta manera,  $\Deltaxyz\Delta\Delta\dots$  representará una cinta que contiene un espacio en blanco, seguido por los símbolos  $x, y$  y  $z$ , seguidos a su vez por espacios en blanco, con la cabeza sobre la celda que contiene la  $z$ .

### Los orígenes de las máquinas de Turing

El propósito para el cual se desarrollaron las máquinas de Turing es distinto del de los otros autómatas que hemos estudiado, ya que se diseñaron para contener todo el poder de los procesos computacionales. En otras palabras, la

intención de Turing fue desarrollar un sistema en el cual fuera posible modelar cualquier proceso que pudiera considerarse como un cálculo.

Recuerde que esto sucedió mucho antes del desarrollo de la maquinaria computacional que existe en la actualidad. Turing pensó en un cálculo realizado por un ser humano con lápiz y papel. Bajo este contexto, Turing razonó que, en un momento dado las personas sólo podrían concentrarse en una porción restringida del papel y que, a su vez, la colección de marcas en este trozo de papel se podía considerar como un solo símbolo. Por esto, Turing consideró dividir el papel en secciones, cada una de las cuales constituía la cantidad de papel requerida para registrar un solo símbolo. Turing también llegó a la conclusión de que cualquier proceso computacional sólo podía implicar un número finito de símbolos. De hecho, puesto que debe ser posible registrar cada símbolo en una cantidad fija de papel, la existencia de una cantidad cada vez mayor de símbolos dictaminaría que los símbolos deberían tener características distintivas arbitrariamente pequeñas. Entonces, se llegaría a un punto donde comenzaría a fallar la capacidad humana para distinguir los símbolos, lo que daría como resultado un número limitado de símbolos efectivos.

Turing planteó que al considerar una sección específica del papel, una persona podría alterar dicha sección o pasar a otra. La acción por emprender y sus detalles dependerían del símbolo existente en la sección y del estado de la mente de la persona. Turing concluyó que, al igual que con el número de símbolos, los seres humanos poseían, sólo una cantidad finita de estados de la mente distinguibles. Turing consideró también que la persona se hallaría en un estado especial inicial al comenzar los cálculos y en un estado designado como de parada al completar los cálculos.

Para evitar que la disponibilidad de papel restringiera el poder del modelo, Turing propuso que la cantidad de papel disponible para los cálculos fuera limitada.

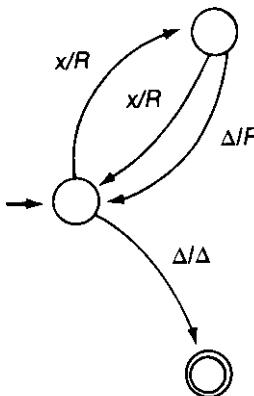
Es muy significativo que nadie haya podido producir un modelo computacional ampliamente aceptado que supere el poder del modelo de Turing. Este modelo es incluso más general que los computadores actuales, ya que una máquina de Turing nunca se ve restringida por la carencia de espacio de almacenamiento, algo que finalmente debe ocurrir en una máquina real. Por esto, la mayoría de los científicos de la computación aceptan la tesis de Turing: *el poder computacional de una máquina de Turing es tan grande como el de cualquier sistema computacional posible*. Además, esta tesis y los resultados que la apoyan hoy en día tienen importantes implicaciones con respecto a las dudas sobre la computabilidad. Para resolver cualquier problema con un computador, se requiere desarrollar un proceso computacional (o un algoritmo) que resuelva el problema. Entonces, las respuestas a las preguntas que se planteaban en la época de Turing, como “¿qué puede hacerse con un proceso computacional?” nos ayudan a responder las preguntas actuales, como “¿qué puede hacer un computador moderno?”.

En el capítulo 4 volveremos a hablar de la función de las máquinas de Turing en el estudio de los procesos computacionales, además de investigar

la relación entre estas nociones abstractas y el diseño de lenguajes de programación. Sin embargo, en este capítulo continuaremos con la presentación de una técnica útil para construir máquinas de Turing y luego veremos la relación entre máquinas de Turing, gramáticas y lenguajes.

## Ejercicios

1. ¿Con qué configuración de cinta se detendrá la máquina de Turing que se presenta a continuación si inicia con la cinta configurada como  $\underline{x}x\Delta\Delta\Delta\dots$ ?



2. Diseñe una máquina de Turing con símbolos de cinta  $x$ ,  $y$  y  $\Delta$  que busque en la cinta el patrón  $xyxy$  y se detenga si y sólo si encuentra ese patrón.
3. Diseñe una máquina de Turing que, al iniciar con la cabeza sobre la celda del extremo izquierdo de la cinta, tenga una terminación anormal si y sólo si hay una  $x$  registrada en algún lugar de la cinta. Si aplicara su máquina a una cinta que no contiene una  $x$ , ¿detectaría la máquina esta situación?
4. Muestre que las capacidades computacionales de una máquina de Turing no aumentarían si se permitiera que tuvieran más de un estado de parada, ya que esta máquina se podría simular con una máquina de Turing con un solo estado de parada.

## 3.2 CONSTRUCCIÓN MODULAR DE MÁQUINAS DE TURING

Aunque el propósito principal de este capítulo es estudiar la capacidad de aceptación de lenguajes por parte de las máquinas de Turing, un efecto

secundario importante es la presentación de los fundamentos de las máquinas de Turing, ya que estas máquinas servirán como herramientas en los capítulos subsecuentes. Por esto investigaremos las máquinas de Turing con mayor detenimiento antes de continuar con nuestro estudio de los lenguajes. En esta sección, nuestro objetivo es desarrollar técnicas por medio de las cuales puedan construirse máquinas de Turing complejas a partir de bloques elementales. Este enfoque facilitará la construcción y la comprensión de las máquinas que veremos más adelante.

## Combinación de máquinas de Turing

Aunque hablaremos en términos de la combinación de máquinas de Turing para formar otras máquinas de Turing mayores, nuestra estrategia será en realidad combinar los programas de las máquinas de Turing en una forma muy semejante a como se combinan módulos de programas para desarrollar grandes sistemas de software. Para esto, es útil representar los programas de las máquinas de Turing por medio de diagramas de transiciones y combinarlos de manera parecida a lo que hicimos para formar la unión y la concatenación de autómatas finitos.

Suponga que tenemos dos máquinas de Turing,  $M_1$  y  $M_2$ , con diagramas de transiciones  $T_1$  y  $T_2$ , respectivamente, y símbolos de cinta del conjunto  $\Gamma$ . Si queremos desarrollar un diagrama de transiciones para otra máquina que simule las actividades de  $M_1$  seguidas por las de  $M_2$ , bastaría con eliminar la designación de parada del estado de parada de  $T_1$  y la característica de inicio del estado inicial de  $T_2$ , y luego dibujar una arco con etiqueta  $x/x$  para cada  $x$  en  $\Gamma$ , del antiguo estado de parada de  $T_1$  al antiguo estado inicial de  $T_2$ .

Obviamente, el resultado se comportaría de la manera esperada, pero este sencillo enfoque tiene varias desventajas. Por ejemplo, en potencia podría introducirse un gran número de transiciones que fueran simples reescrituras del contenido de la celda (esta deficiencia presentaría obstáculos para nuestro análisis de la complejidad en el capítulo 5). Además, suponga que queremos que la máquina compuesta se detenga después de simular  $M_1$ , a menos que el símbolo actual, al llegar al estado de parada, sea  $z$ , en cuyo caso nos gustaría que la nueva máquina simulara las acciones de  $M_2$ ; o suponga que necesitamos combinar tres diagramas y controlar el paso de uno a otro dependiendo del valor del símbolo actual al llegar a cada uno de los antiguos estados de parada. En estos casos se requiere un enlace un poco más complicado.

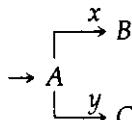
Suponga entonces que necesitamos combinar los diagramas de transiciones de varias máquinas de Turing para obtener una máquina que simule alguna combinación de las máquinas originales. Lo hacemos de la siguiente manera:

1. Elimine la característica de inicio de los estados iniciales de todas las máquinas, excepto la de aquél donde iniciará la máquina compuesta.

2. Elimine la característica de detención de los estados de parada de todas las máquinas e introduzca un nuevo estado de parada que no se encuentre en ninguno de los diagramas que se combinan.
3. Para cada uno de los antiguos estados de parada  $p$  y cada  $x$  en  $\Gamma$ , dibuje un arco de la siguiente forma:
  - a. Si la máquina compuesta debe detenerse al llegar a  $p$  con el símbolo actual  $x$ , dibuje un arco con etiqueta  $x/x$  de  $p$  al nuevo estado de parada.
  - b. Si al llegar al estado  $p$  con el símbolo actual  $x$ , la máquina compuesta debe transferir el control a la máquina  $M = (S, \Sigma, \Gamma, \delta, i, h)$ , dibuje entonces un arco con etiqueta  $x/z$  de  $p$  al estado  $q$  de  $M$ , donde  $\delta(i, x) = (q, z)$ .

La figura 3.3 proporciona un ejemplo de esta construcción. Aquí hemos comenzado con los diagramas de transiciones para tres máquinas distintas, cada una con los símbolos de cinta  $x$ ,  $y$  y  $\Delta$ . Una mueve su cabeza una celda a la derecha, otra encuentra la primera  $x$  a la derecha de la celda actual, y la tercera encuentra la primera  $y$  a la derecha de la celda actual. Con el proceso de composición antes mencionado y empleando estas máquinas como bloques de construcción, hemos construido una máquina compuesta que encuentra la segunda ocurrencia del símbolo distinto de espacio en blanco que se halla a la derecha de la posición inicial de la cabeza.

Es conveniente evitar los detalles interiores de los bloques de construcción a partir de los cuales construimos máquinas de Turing más complejas, como sucede en el caso de grandes sistemas de software. Si sabemos cuál es la tarea que realiza cada una de las máquinas más pequeñas, entonces nuestra preocupación será el adecuado enlace de estas máquinas y no cómo cada una lleva a cabo su tarea específica. Por esto, evitaremos el uso de diagramas de transiciones detallados como los que se muestran en la figura 3.3, y en vez de esto utilizaremos diagramas compuestos. En estos diagramas cada uno de los bloques de construcción se representa como un nodo, con flechas entre los nodos para indicar las transiciones entre bloques. Estas flechas se etiquetan de acuerdo con el valor que debe aparecer en la celda actual para que se recorra la flecha. Es decir, si una flecha del nodo  $A$  al nodo  $B$  tiene una etiqueta  $x$ , entonces la ejecución se transferirá a la máquina  $B$  si  $A$  llega a su antiguo estado de parada con una  $x$  en la celda actual. Como hicimos con los diagramas de transiciones, debe indicarse con un apuntador el nodo del diagrama compuesto donde debe comenzar la ejecución. Por ejemplo, la máquina compuesta presentada en la figura 3.3 podría resumirse con el diagrama compuesto



donde el nodo  $A$  representa la máquina que mueve su cabeza una celda a la derecha,  $B$  la máquina que busca una  $x$  y  $C$  la máquina que busca una  $y$ .

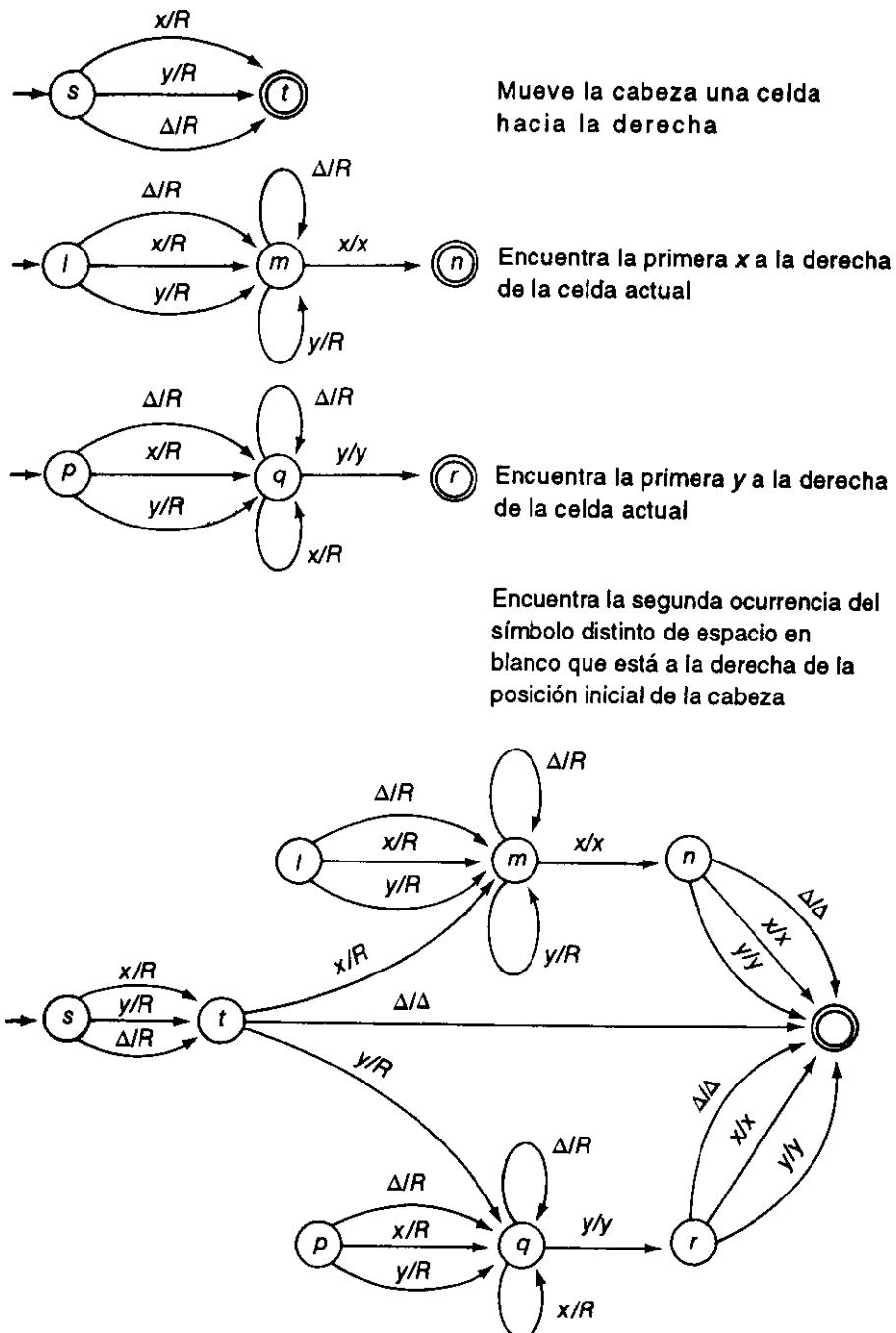


Figura 3.3 Construcción de una máquina de Turing compuesta a partir de máquinas más pequeñas

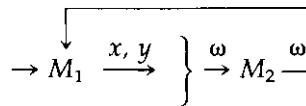
Por cuestiones de conveniencia, existen varias abreviaturas en cuanto a la notación que se emplea habitualmente al dibujar diagramas compuestos para máquinas de Turing. Uno es reemplazar varias flechas que tienen la misma fuente y el mismo destino por una sola flecha rotulada con una lista de símbolos, o quizás con  $\neg x$  (léase "no  $x$ "), si hay que recorrer la flecha para todos los símbolos actuales distintos de  $x$ . Otra abreviatura es utilizar una flecha sin etiqueta para representar una transición que debe recorrerse sin importar el valor de la celda actual. Incluso esta situación se llega a abreviar más, eliminando la flecha y colocando los nodos uno al lado del otro. Por ejemplo, la secuencia  $\rightarrow A \rightarrow B \rightarrow C$  podría simplificarse a  $\rightarrow ABC$ .

Como ejemplos, la figura 3.4a muestra un diagrama compuesto de una máquina construida a partir de  $M_1$  y  $M_2$ . La máquina compuesta simula las acciones de  $M_1$  hasta el punto donde normalmente se detendría y luego simula las acciones de  $M_2$ . La figura 3.4b representa una máquina que simula  $M_1$  y luego hace una transferencia a  $M_2$  sólo si la celda actual contiene una  $x$ ; de lo contrario, se detiene al concluir las acciones de  $M_1$ . Por último, la figura 3.4c representa la máquina compuesta que comienza con la simulación de  $M_1$ , y luego simula  $M_2$  o  $M_3$ , dependiendo de si el símbolo actual es  $x$  o no.

Otra abreviatura que utilizaremos consiste en aplicar la notación

$$\xrightarrow{x, y, z} \left. \right\} \omega \rightarrow$$

para indicar que cuando el símbolo actual es  $x$ ,  $y$  o  $z$ , la máquina deberá proseguir en esta dirección, donde  $\omega$  representa el símbolo que en realidad está presente. Esta notación evita saturar el diagrama con rutinas similares, aunque separadas, para cada uno de los símbolos  $x$ ,  $y$  y  $z$ . En vez de esto, podemos presentar una sola rutina genérica que trata con el símbolo  $\omega$ , de manera parecida a como un subprograma presenta una rutina genérica en términos de parámetros formales. Así, si la máquina



fuerá a llegar al antiguo estado de parada de  $M_1$  con el símbolo actual  $x$  o  $y$ , continuaría con la ejecución de  $M_2$ ; esto llevaría al antiguo estado de parada de  $M_2$  con el mismo símbolo actual que tenía al entrar a  $M_2$ , y luego la ejecución regresaría a  $M_1$ .

## Bloques de construcción básicos

Una vez establecido este sistema de notación, consideremos ahora las máquinas de Turing elementales que emplearemos como bloques de construcción en análisis posteriores. Obtendremos una pista de cuáles deberán ser estos

bloques de construcción si recordamos que las únicas actividades disponibles para una máquina de Turing son mover la cabeza una celda a la derecha, moverla una celda a la izquierda y escribir un símbolo en la celda actual. Por consiguiente, si construimos máquinas individuales que realizan estas sencillas tareas, entonces cualquier otra máquina de Turing debe ser una composición de estos bloques.

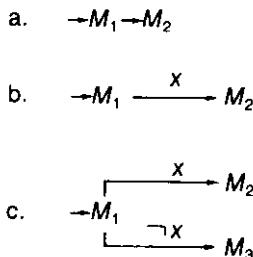


Figura 3.4 Ejemplos de máquinas de Turing compuestas

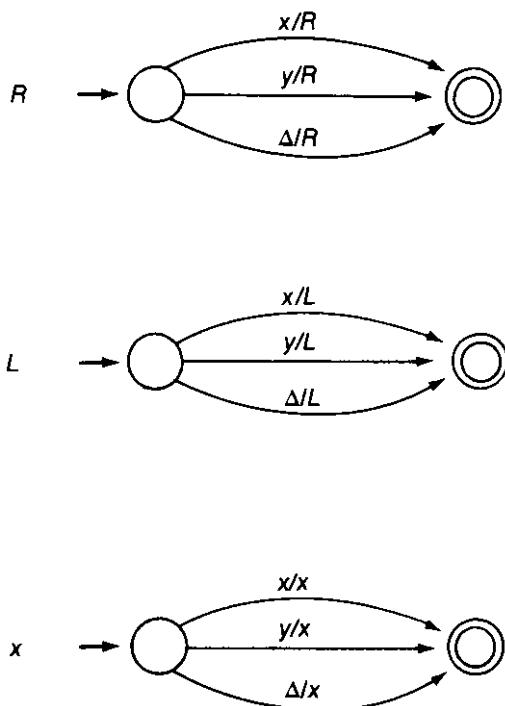


Figura 3.5 Máquinas  $R$ ,  $L$  y  $x$

La figura 3.5 muestra los diagramas de transiciones de las máquinas de Turing que efectúan cada una de estas actividades rudimentarias, suponiendo que los símbolos de la cinta son  $x$ ,  $y$  y  $\Delta$  (las máquinas para otros

símbolos no son más que simples generalizaciones de estas máquinas). Representamos con  $R$  la máquina que mueve su cabeza una celda a la derecha, con  $L$  la que la mueve una celda a la izquierda y con  $x$  la máquina que escribe un símbolo en la celda actual. De esta manera, una máquina de Turing que se mueve una celda a la derecha, escribe el símbolo  $y$ , se mueve una celda hacia la izquierda y se detiene, podría representarse con el diagrama compuesto

$$\rightarrow R \rightarrow y \rightarrow L$$

o, en forma condensada, como

$$\rightarrow RyL$$

Nuestro siguiente paso es establecer un repertorio de máquinas ligeramente más complejas. Un grupo de estas máquinas, presentado en la figura 3.6, lleva a cabo búsquedas sencillas. De manera más precisa, para cualquier símbolo  $x$  la máquina representada por  $R_x$  recorre la cinta a la derecha de su posición inicial en busca de una celda que contenga el símbolo  $x$ . Si encuentra ese símbolo, la máquina se detiene y esa celda será la actual; de no ser así, la máquina continuará su búsqueda eternamente.

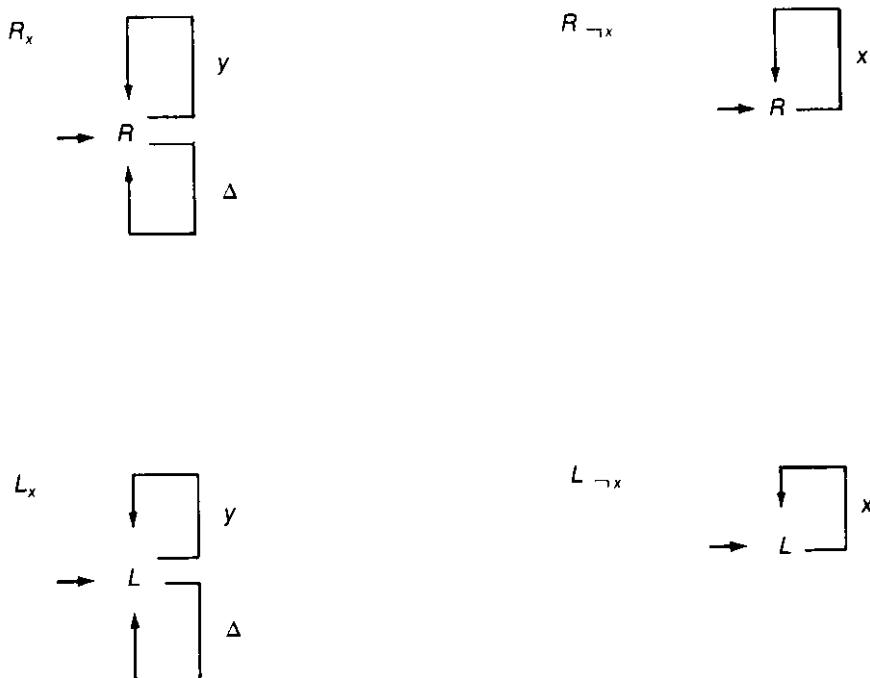
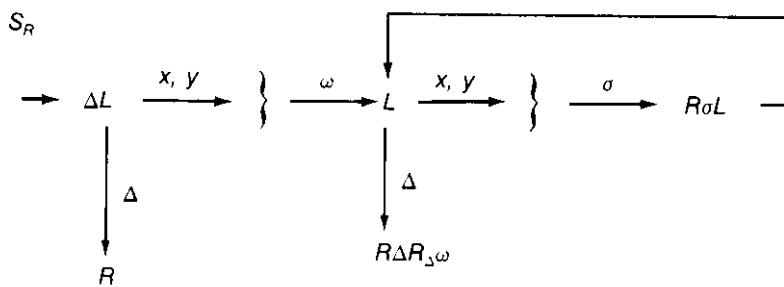
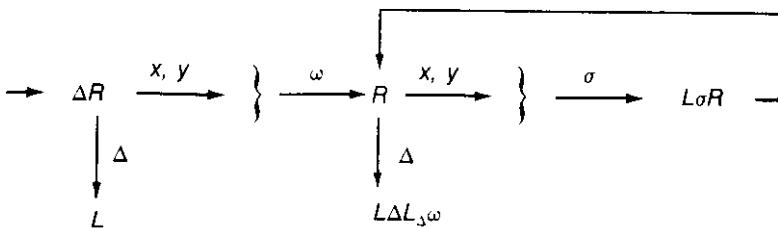


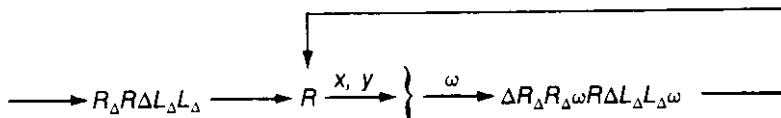
Figura 3.6 Máquinas  $R_x$ ,  $R_{\neg x}$ ,  $L_x$  y  $L_{\neg x}$

De modo similar, la máquina  $R_{\neg x}$  buscará a la derecha de la posición inicial cualquier símbolo que no sea  $x$ . Las máquinas  $L_x$  y  $L_{\neg x}$  efectúan estas mismas búsquedas hacia la izquierda de la posición inicial (observe que, a diferencia de la búsqueda hacia la derecha desde la posición inicial, la búsqueda hacia la izquierda puede ocasionar una terminación anormal si llega al extremo izquierdo de la cinta sin encontrar el objetivo de la búsqueda). Las máquinas  $R_\Delta$ ,  $R_{\neg \Delta}$ ,  $L_\Delta$  y  $L_{\neg \Delta}$  tendrán una utilidad especial para nosotros, pues pueden usarse para construir máquinas compuestas que busquen celdas en blanco o que no estén en blanco.

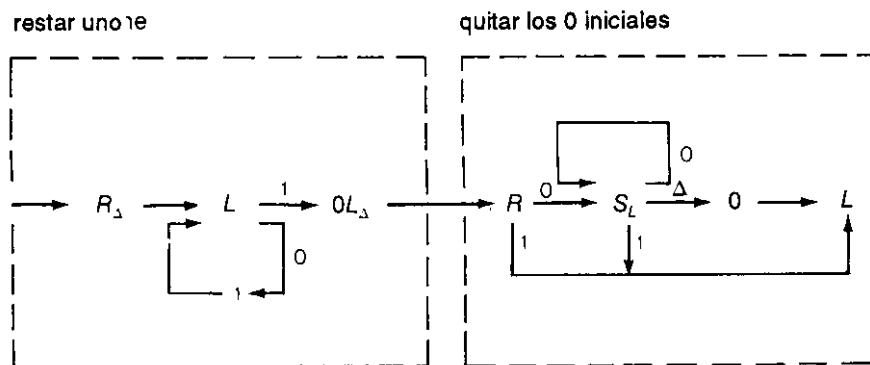
Otra colección de máquinas que será útil es la que lleva a cabo operaciones de desplazamiento; en la figura 3.7 se presentan dos de estas máquinas. La máquina  $S_R$  desplaza una celda hacia la derecha la cadena de símbolos que no están en blanco y que se encuentran a la izquierda de la celda actual. Así, si la cinta de  $S_R$  tuviera la configuración inicial  $\Delta xyyxx\Delta\Delta\Delta\Delta\cdots$ , entonces  $S_R$  se detendría con la cinta configurada como  $\Delta\Delta xy\cancel{y}x\Delta\Delta\Delta\cdots$ ; o, si se aplicara a  $\Delta xy\cancel{y}\Delta\Delta xxy\Delta\Delta\cdots$ ,  $S_R$  produciría  $\Delta\Delta y\cancel{y}x\Delta xy\Delta\Delta\cdots$ . Como un caso algo especial, si la celda a la izquierda de la actual contiene un espacio en blanco,  $S_R$  únicamente borraría la celda actual. Así,  $S_R$  convertiría la configuración  $xy\Delta yx\Delta\Delta\Delta\cdots$  de la cinta a  $xy\Delta\cancel{x}\Delta\Delta\Delta\cdots$ .

 $S_R$ **Figura 3.7** Máquinas  $S_R$  y  $S_L$

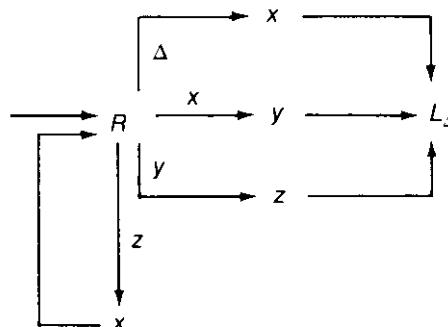
En forma parecida,  $S_L$  realiza un desplazamiento hacia la izquierda. Si la cinta de  $S_L$  tuviera la configuración inicial  $\Delta yyyxx\Delta\Delta\Delta\cdots$ , entonces  $S_L$  se detendría con su cinta configurada como  $\Delta yyxx\Delta\Delta\Delta\Delta\cdots$ ; o si se aplicara a  $\Delta yxy\Delta\Delta xxy\Delta\Delta\cdots$ ,  $S_L$  produciría  $\Delta yxy\Delta xxy\Delta\Delta\Delta\cdots$ .



**Figura 3.8** Máquina copiadora que transforma un patrón de la forma  $\Delta w\Delta$   
 $\Delta w\Delta w\Delta$



**Figura 3.9** Máquina de Turing para reducir en uno una representación binaria



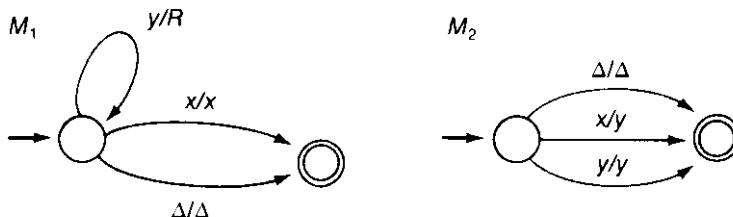
**Figura 3.10** Máquina de Turing compuesta que produce la siguiente  
cadena en la secuencia  $\lambda, x, y, z, xx, yx, zx, xy, yy, zy, xz,$   
 $yz, zz, xxx, yxx, \dots$

Concluimos con varios ejemplos de cómo pueden combinarse las máquinas elementales presentadas hasta ahora para formar máquinas compuestas más complejas. La figura 3.8 define una máquina copiadora que transforma un patrón de la forma  $\underline{\Delta}w\Delta$  a la forma  $\Delta w\underline{\Delta}w\Delta$ , donde  $w$  es cualquier cadena (posiblemente de longitud cero) de símbolos que no son espacios en blanco. La máquina que se presenta en la figura 3.9 supone que su entrada representa un entero positivo en notación binaria y reduce en uno el valor representado. Por último, la máquina de la figura 3.10 modifica cadenas del alfabeto  $\{x, y, z\}$ . Específicamente, si iniciamos la máquina con su cinta configurada como  $\underline{\Delta}w_1\Delta\Delta\Delta\dots$ , donde  $w_1$  es una cadena en  $\{x, y, z\}^*$ , entonces la máquina se detendrá con la cinta configurada como  $\underline{\Delta}w_2\Delta\Delta\Delta\dots$ , donde  $w_2$  es la cadena que sigue a  $w_1$  en la secuencia  $\lambda, x, y, z, xx, yx, zx, xy, yy, zy, xz, yz, zz, xxx, yxx, \dots$ . Más adelante veremos la utilidad de esta máquina como bloque de construcción.

## Ejercicios

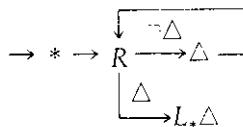
- Combine los diagramas de transiciones de las máquinas  $M_1$  y  $M_2$  que se muestran a continuación para formar el diagrama de transiciones de la máquina compuesta

$\downarrow$   
 $x$   
 $\rightarrow M_1 \rightarrow M_2$ . ¿Qué hace la máquina compuesta?



- Utilizando los bloques de construcción presentados en esta sección, construya una máquina de Turing que reemplace la cadena de ceros y unos inmediatamente a la derecha de la cabeza por el complemento de la cadena y luego regrese la cabeza a su posición original. Suponga que el extremo derecho de la cadena está marcado con un espacio en blanco (el complemento de una cadena de ceros y unos es la cadena que se forma al sustituir los ceros originales con unos, y los unos originales, con ceros).
- ¿En qué condiciones se presentaría una terminación anormal durante la ejecución de la máquina  $S_R$ ?

4. Describa los cálculos efectuados por la siguiente máquina de Turing.



### 3.3 MÁQUINAS DE TURING COMO ACEPTADORES DE LENGUAJES

Hemos presentado a los autómatas de los capítulos anteriores como aceptadores de lenguajes y los hemos utilizado para evaluar cadenas y determinar su pertenencia a un lenguaje específico. Por lo tanto, es natural que estudiemos a las máquinas de Turing desde la misma perspectiva. En esta sección analizaremos los aspectos de la aceptación de cadenas realizada por máquinas de Turing; empezaremos considerando el contexto en el cual una máquina de Turing acepta una cadena de entrada.

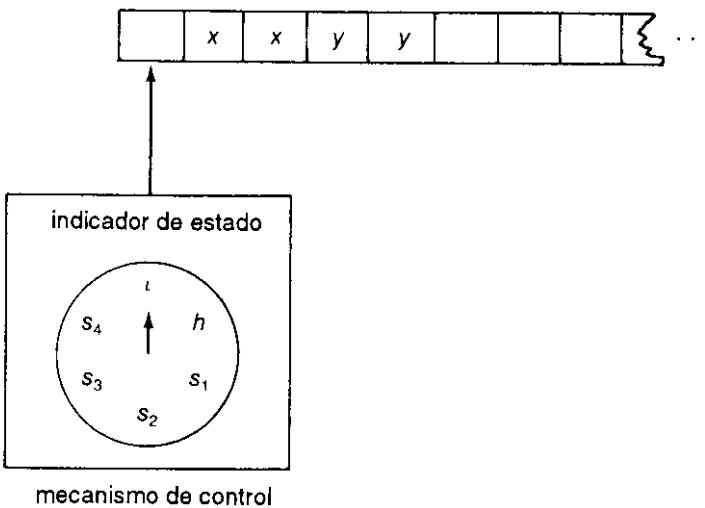
#### Procedimientos de evaluación de cadenas

Para evaluar una cadena de algún alfabeto  $\Sigma$  con una máquina de Turing, registramos la cadena en la cinta (que de otra manera estaría en blanco) de la máquina, comenzando por la segunda celda (si fuéramos a evaluar la cadena  $xxyy$ , la cinta aparecería como  $\triangle xxyy\triangle\triangle\triangle\cdots$ ). Luego colocamos la cabeza de la máquina en la celda del extremo izquierdo de la cinta y ponemos en marcha la máquina a partir de su estado inicial (véase Fig. 3.11). Decimos que la máquina acepta la cadena si, a partir de esta configuración inicial, encuentra su camino hasta su estado de parada.

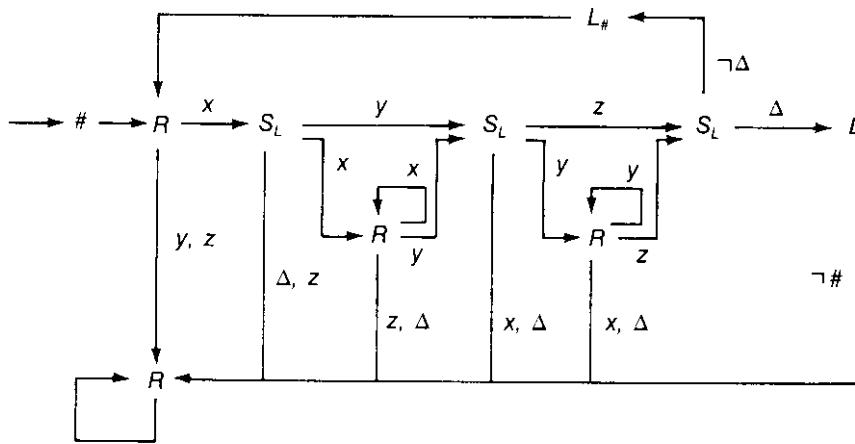
Como ejemplo, en la figura 3.12 se muestra un diagrama compuesto para una máquina que acepta las cadenas que tengan precisamente la forma  $x^n y^n z^n$ , donde  $n \in \mathbb{N}$ . Esta máquina interroga la entrada reduciendo repetidamente la longitud de una cadena no vacía de la forma  $x^n y^n z^n$  a través de la secuencia  $x^{n-1} y^{n-1} z^n$ ,  $x^{n-1} y^{n-1} z^{n-1}$  y  $x^{n-1} y^{n-1} z^{n-1}$ . La máquina se detiene si y sólo si este proceso de reducción produce una cadena vacía como resultado de la eliminación del mismo número de  $x$ ,  $y$  y  $z$  (se emplea  $\#$  como símbolo de cinta para ayudar a encontrar el extremo izquierdo de la cinta después de completar cada secuencia de reducciones).

Al igual que con los demás autómatas, la colección de cadenas aceptadas por una máquina de Turing  $M$  se llama lenguaje aceptado por la máquina y se representa con  $L(M)$ . Se dice que un lenguaje  $L$  es un lenguaje aceptado por una máquina de Turing si existe una máquina de Turing  $M$  tal que  $L = L(M)$ .

La figura 3.12 tiene una relevancia especial para nuestros fines, pues muestra que las máquinas de Turing son capaces de aceptar lenguajes que no



**Figura 3.11** Configuración inicial de una máquina de Turing al evaluar la cadena  $xxxy$ .



**Figura 3.12** Máquina de Turing  $M$  para la cual  $L(M) = \{x^n y^n z^n: n \in \mathbb{N}\}$

pueden aceptar los autómatas de pila (recuerde que  $\{x^n y^n z^n: n \in \mathbb{N}\}$  no es un lenguaje independiente del contexto). De hecho, encontraremos que la clase de los lenguajes aceptados por máquinas de Turing contiene propiamente todos los lenguajes independientes del contexto, aunque requeriremos un poco de trabajo preliminar para demostrarlo.

Hemos definido la aceptación de cadenas de las máquinas de Turing de una manera que se ajusta a la de los demás autómatas. Sin embargo, en ocasiones es conveniente requerir que una máquina de Turing escriba un

mensaje de aceptación en su cinta antes de detenerse. Por ejemplo, podríamos desear que una máquina de Turing acepte una cadena si se detiene con su cinta en la configuración  $\Delta Y \Delta \Delta \Delta \dots$ , donde el símbolo  $Y$  se usa para representar la respuesta afirmativa (en estas circunstancias, no se considera una aceptación si la máquina se detiene con cualquier otra configuración de la cinta).

Por fortuna, este criterio de aceptación adicional no afecta la clase de lenguajes que pueden aceptar las máquinas de Turing. Dada una máquina de Turing  $M$  que acepte las cadenas con sólo detenerse, existe otra máquina de Turing  $M'$  que acepta las mismas cadenas si se detiene con la cinta configurada como  $\Delta Y \Delta \Delta \Delta \dots$ , y viceversa. Para justificar esta afirmación, consideremos primero una máquina de Turing  $M$  que acepte cadenas con sólo detenerse y luego mostremos que podemos modificarla para que acepte las mismas cadenas si se detiene con una configuración de cinta  $\Delta Y \Delta \Delta \Delta \dots$ .

Básicamente, todo lo que tenemos que hacer es modificar  $M$  para que lleve el control de la porción de la cinta que altera durante la ejecución. Luego, después de terminar sus cálculos normales, podrá borrar esa porción de la cinta y escribir el mensaje adecuado en la cinta en blanco antes de detenerse. Para llevar este control de la porción alterada de la cinta, utilizamos los símbolos de cinta especiales  $\#$  y  $*$ . El símbolo  $\#$  se usará para marcar el extremo izquierdo de la cinta y el símbolo  $*$  para marcar el extremo derecho de la porción alterada. Así, la máquina modificada deberá comenzar cualquier cálculo con

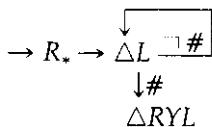
$$\rightarrow R \wedge S_R R * L \wedge L \# R$$

Es decir, debe desplazarse hacia el extremo derecho de la cinta de entrada y desplazar la cadena una celda a la derecha. Luego, debe marcar la primera celda a la derecha de la entrada con el símbolo  $*$ , regresar a la celda del extremo izquierdo de la cinta, escribir el símbolo  $\#$  y ubicar su cabeza sobre el espacio en blanco en el extremo izquierdo de la cadena de entrada. En resumen, dada la configuración inicial  $\Delta w \Delta \Delta \dots$ , donde  $w$  es la cadena de entrada, la máquina produce la configuración  $\# \Delta w * \Delta \Delta \dots$ .

A partir de esta configuración, nuestra máquina modificada debe continuar con la simulación de las acciones de  $M$ . Sin embargo, al efectuar la simulación, la máquina debe estar pendiente de la ocurrencia de dos circunstancias especiales. La primera ocurre si el símbolo  $\#$  se convierte en el símbolo actual. Observe que, como toda la entrada se ha desplazado una celda a la derecha, no se leerá la celda que contiene  $\#$  durante la simulación, a menos que el proceso simulado sufra una terminación anormal. Entonces, si el símbolo actual fuera  $\#$ , nuestra máquina modificada debería mover su cabeza una celda más hacia la izquierda para que también experimentara una terminación anormal.

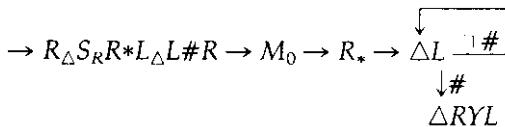
La otra circunstancia especial es la ocurrencia del símbolo  $*$  como símbolo actual. Esta situación indica que el cálculo simulado ha movido su cabeza a la derecha, hacia una celda de la cinta que no se había alterado antes. En este caso, nuestra máquina modificada debe empujar la marca  $*$  una celda a la derecha ejecutando  $\rightarrow R * L \wedge$  antes de continuar con la simulación.

Una vez que se han simulado los cálculos de  $M$  hasta el punto donde  $M$  ha llegado a su estado de parada, nuestra máquina deberá borrar su cinta y escribir el mensaje  $Y$  antes de detenerse. Esto se logra añadiendo el bloque



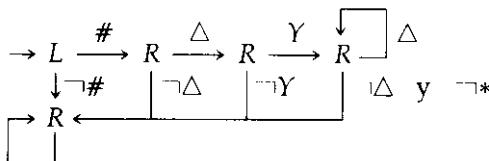
al final de nuestra máquina modificada.

En resumen, la máquina final es



donde  $M_0$  es la máquina que simula  $M$ , excepto por las dos circunstancias especiales en los cuales surge  $\#$  o  $*$  como el símbolo actual.

A la inversa, suponga que comenzamos con una máquina de Turing  $M'$  que acepta cadenas deteniéndose con la cinta configurada como  $\Delta Y \Delta \Delta \Delta \dots$ . Entonces podríamos alterar esta máquina para obtener otra que no se detuviera en ningún otro caso. Lo único que se requiere es insistir en que la máquina revise su cinta en busca de la configuración  $\Delta Y \Delta \Delta \Delta \dots$  antes de detenerse. Para esto, construimos otra máquina que marque los extremos izquierdo y derecho de la cinta en la forma antes descrita; simule las acciones de  $M'$  tomando en cuenta la ocurrencia de  $\#$  o  $*$  como símbolo actual; y, por último, si la simulación llega hasta el estado de parada original, confirme que la porción de la cinta limitada por las marcas  $\#$  y  $*$  esté configurada como  $\# \Delta Y \Delta \Delta \dots \Delta \Delta *$  antes de detenerse. Este último paso podría lograrse con una rutina como



Concluimos que se pueden establecer varias maneras para que una máquina de Turing acepte sus cadenas de entrada. Podemos permitir que simplemente se detengan o insistir en que respondan con un mensaje de aceptación. Ambas estrategias tienen sus ventajas y desventajas pero, ya que cuentan con el mismo poder, tenemos la libertad de elegir el método que más convenga para la aplicación. Sin embargo, supondremos que una máquina de Turing acepta sus cadenas con sólo detenerse, a menos que se especifique lo contrario.

## Máquinas de Turing de varias cintas

Ahora consideraremos las máquinas de Turing que tienen más de una cinta (este tipo de máquinas se conoce como máquinas de Turing de  $k$  cintas, donde  $k$  representa un entero positivo, en aquellos casos donde tenga relevancia el número de cintas). Cada una de estas cintas tiene un extremo izquierdo, se extiende indefinidamente hacia la derecha, y el acceso a cada una se logra a través de una cabeza separada de lectura y escritura. La transición que se ejecutará en un instante determinado dependerá de la colección de símbolos presentes para estas cabezas, junto con el estado actual de la máquina. La acción de una sola transición afecta sólo a una de las cintas de la máquina; esta acción puede ser escribir en la celda actual de esa cinta, mover la cabeza correspondiente una celda a la izquierda o moverla una celda a la derecha.

Para evaluar la aceptación de una cadena de símbolos utilizando una máquina de Turing de varias cintas, comenzamos con la máquina en su estado inicial y con la cadena de entrada registrada en la primera cinta (en el mismo formato que tendría en la cinta única de una máquina convencional), a la vez que las otras cintas están en blanco y todas las cabezas se encuentran en la celda del extremo izquierdo de sus cintas. La cadena es aceptada si, a partir de esta configuración inicial, la máquina se detiene.

Uno podría esperar que las máquinas de Turing de varias cintas tuvieran un potencial de procesamiento de lenguajes mayor que sus homólogos de una sola cinta; no obstante, el siguiente teorema muestra que esto no es cierto. Aunque en ocasiones conviene emplear una máquina con varias cintas, estas máquinas no pueden aceptar más lenguajes que las máquinas de Turing convencionales.

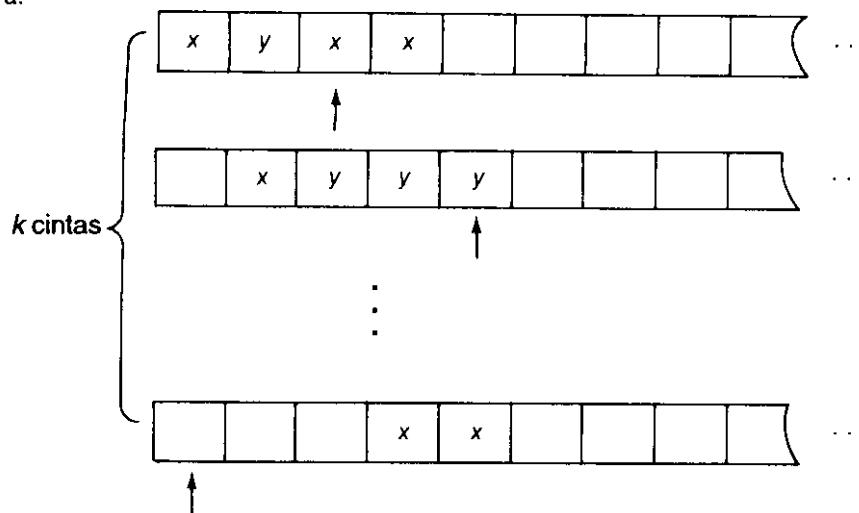
### TEOREMA 3.1

Para cada máquina de Turing de varias cintas  $M$ , hay una máquina de Turing tradicional (de una cinta)  $M'$  tal que  $L(M) = L(M')$ .

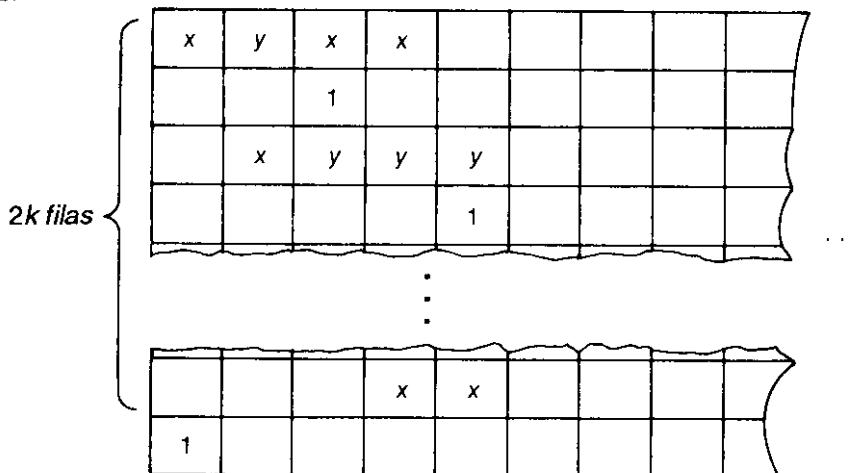
#### DEMOSTRACIÓN

Suponga que  $M$  es una máquina de Turing de  $k$  cintas que acepta el lenguaje  $L$ . Nuestra estrategia es mostrar la posibilidad de representar el contenido de las  $k$  cintas en una sola, de modo que las acciones de  $M$  puedan ser simuladas por una máquina  $M'$  de cinta única. Visualizamos las cintas de  $M$  colocadas paralelas entre sí, con los extremos izquierdos alineados, como se presenta en la figura 3.13a, donde un apuntador bajo cada cinta indica la posición de la cabeza. Con base en esta disposición, es posible representar el contenido de las  $k$  cintas y las posiciones de sus cabezas en una tabla que contiene  $2k$  filas y un número ilimitado de columnas que se extienden hacia la derecha, como se muestra en la figura 3.13b. Las filas impares de esta tabla representan las cintas; las pares se usan para indicar la posición

a.



b.



**Figura 3.13** Representación de las cintas de una máquina de Turing de  $k$  cintas con un solo arreglo

de la cabeza de la cinta de la fila anterior, lo cual se logra colocando un 1 en la columna asociada a la celda actual de la fila superior y dejando en blanco todas las demás celdas.

Observe que cada una de las columnas de la tabla que acabamos de describir es en esencia una  $2k$ -tupla, siendo los componentes impares símbolos de cinta de  $M$  y los pares elementos de  $\{\triangle, 1\}$ . Por lo tanto, existe sólo un número finito de combinaciones distintas de símbolos

que pueden aparecer en las columnas. Esto quiere decir que podemos asignar un nuevo símbolo de cinta, único, a cada una de las  $2k$ -tuplas posibles y luego representar toda la tabla como una cadena infinita de estos nuevos símbolos. De esta manera podemos almacenar en una sola cinta toda la información almacenada en las  $k$  cintas de la máquina. Aunque cada una de las celdas de la cinta única contiene sólo un símbolo, éste representa una  $2k$ -tupla. Por lo tanto, es conveniente expresarlo como si la cinta contuviera directamente la tupla, en vez del símbolo que la representa. Nosotros adoptaremos esto de manera informal.

Ahora estamos preparados para describir una máquina de Turing de cinta única  $M'$  que acepte el mismo lenguaje que la máquina de  $k$  cintas  $M$ . Los alfabetos de  $M$  y  $M'$  son los mismos; empero, la colección de símbolos de cinta de  $M'$  consiste en los símbolos de cinta de  $M$ , un símbolo para cada  $2k$ -tupla posible y un símbolo  $\#$  que se usa como marca especial. Para evaluar una cadena, la máquina  $M'$  comienza con su cinta como un duplicado exacto de la cinta 1 de la máquina  $M$  de  $k$  cintas. La primera tarea de  $M'$  es traducir el contenido de su cinta a un formato que represente las  $k$  cintas de  $M$ . Para lograrlo,  $M'$  ejecuta los pasos siguientes:

- A1. Desplaza el contenido de la cinta una celda a la derecha por medio de  $\rightarrow R_\Delta S_R L_\Delta$  (la cabeza quedará sobre la segunda celda de la cinta).
- A2. Mueve la cabeza una celda a la izquierda (es decir, hacia la celda del extremo izquierdo de la cinta), escribe la marca especial  $\#$  y luego mueve la cabeza una celda a la derecha (aquí se supone que  $\#$  no es un símbolo de cinta de  $M$ , sino que se emplea para marcar el final de la cinta. Si se lee durante la simulación de  $M$ , sabemos que  $M$  sobrepasó el extremo de su cinta).
- A3. Repite los pasos siguientes hasta que el símbolo reemplazado en el paso b sea un espacio en blanco.
  - a. Mueve la cabeza de la cinta una celda a la derecha.
  - b. Reemplaza el símbolo actual, digamos  $x$ , con el símbolo de cinta que representa a la tupla  $(x, \Delta, \Delta, \Delta, \dots; \Delta, \Delta)$ .
- A4. Ejecuta  $L_\Delta$ , lo cual moverá la cabeza de regreso a la segunda celda de la cinta. Sustituye el espacio en blanco de esta celda con el símbolo de cinta que representa a la tupla  $(\Delta, 1, \Delta, 1, \dots; \Delta, 1)$ .

Después de ejecutar estos pasos,  $M'$  habrá traducido su cinta a un formato de cintas múltiples, marcado el extremo izquierdo de la cinta con el símbolo  $\#$  y devuelto su cabeza a la tupla que representa las celdas del extremo izquierdo de las  $k$  cintas de  $M$  (véase Fig 3.14).

La tarea de  $M'$  sería ahora simular las acciones de  $M$  hasta que  $M$  se detenga o termine anormalmente. Por supuesto, para simular una

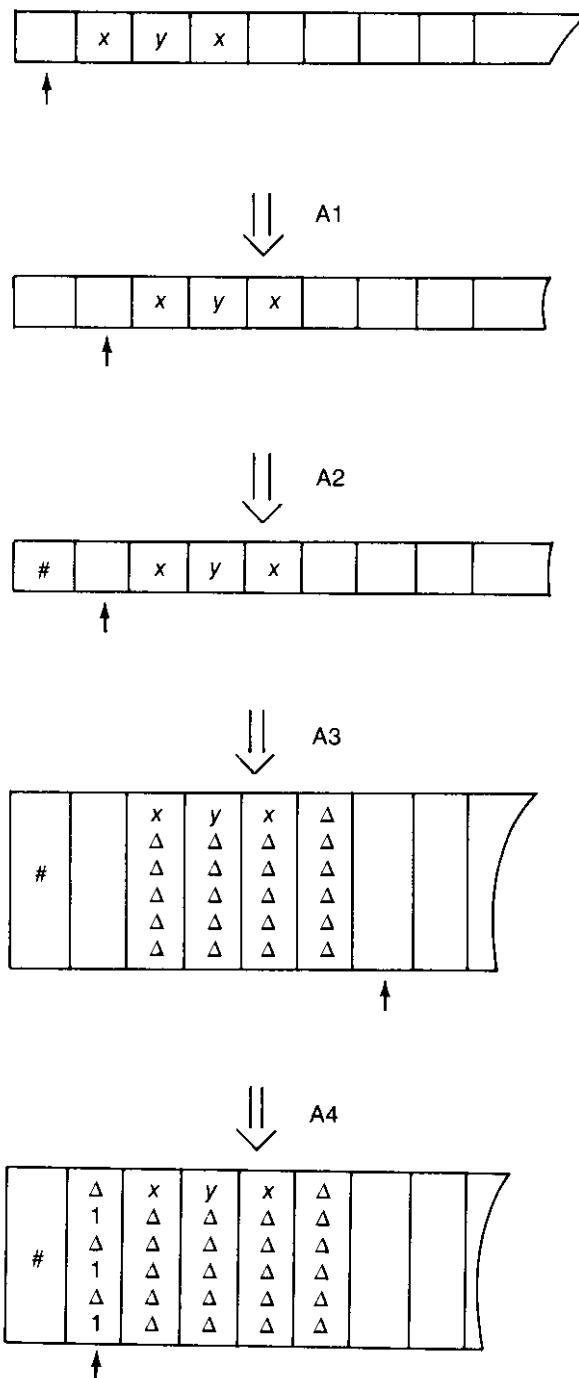


Figura 3.14 Conversión de una cinta única a un formato de tres cintas

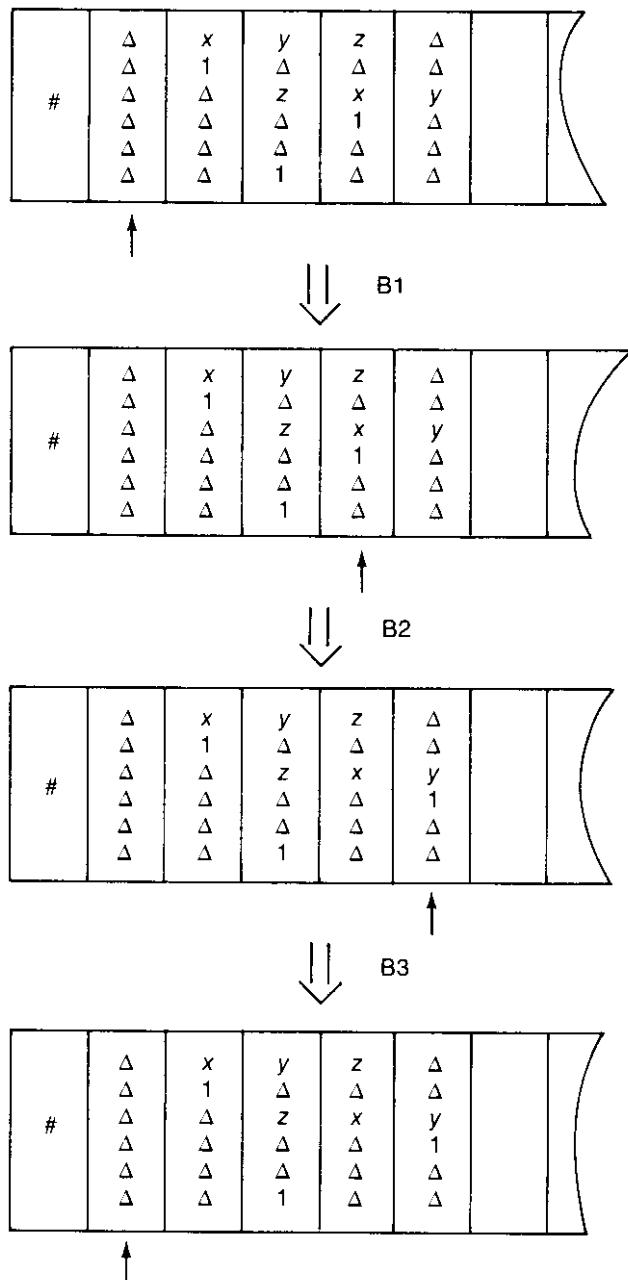
transición de  $M$  se requiere una secuencia de pasos en  $M'$ . Diseñamos  $M'$  de manera que cada secuencia comience en un estado especial, llamado estado compuesto, que refleje el estado actual de  $M$  así como la colección de  $k$  símbolos de las celdas actuales de las cintas de  $M$ . Así, cada estado compuesto es conceptualmente una  $K+1$ -tupla, donde el primer componente es un estado de  $M$  y los restantes son símbolos del alfabeto de  $M$ .

Construimos  $M'$  de forma tal que hallarse en el estado compuesto  $(p, x_1, x_2, \dots, x_k)$  corresponda a que  $M$  se encuentre en el estado  $p$  con los símbolos actuales  $x_1, x_2, \dots, x_k$ . A partir de dicho estado,  $M'$  ejecutará una secuencia de pasos diseñados para simular cada transición que ejecutaría  $M$  en la situación correspondiente (observe que esta transición es determinada en forma única por el estado compuesto). De esta manera, la decisión de cuál será la siguiente transición que se ejecute no se realiza dinámicamente durante el proceso de simulación, sino que se determina antes de la ejecución, durante la construcción de la máquina). Una vez que se ha ejecutado la secuencia de simulación de la transición,  $M'$  se desplazará al estado compuesto que corresponda a la situación donde se encontraría  $M$  después de ejecutar la transición individual.

Para establecer la base de este proceso de simulación, refinamos el paso A4 anterior para que deje a  $M'$  en el estado compuesto que representa a la  $k+1$ -tupla  $(i, \triangle, \triangle, \dots, \triangle)$ , donde  $i$  es el estado inicial de  $M$ . Así, después de terminar con el paso A4,  $M'$  se hallará en el estado compuesto asociado al estado inicial y a los símbolos actuales de  $M$ , dispuesta a iniciar el proceso de reconocimiento de la cadena.

Para simular la ejecución de la transición  $\tau$  de  $M$ ,  $M'$  ejecuta los pasos B1 a B3, donde empleamos la notación  $j_\tau$  para representar el número de la cinta que se vería afectada por la transición  $\tau$  (recuerde que una transición de la máquina de Turing de varias cintas afecta sólo a una de las cintas de la máquina).

- B1. Mueve la cabeza a la derecha hasta que el componente  $2j_\tau$  de la tupla en la celda actual sea 1 (es decir, encuentra la posición de la cabeza asociada con la cinta  $j_\tau$ ).
- B2.
  - a. Si la transición  $\tau$  es una operación de escritura, modifica el componente  $2j_\tau - 1$  según se indique.
  - b. Si la transición  $\tau$  es un movimiento hacia la derecha, reemplaza con un espacio en blanco el 1 del componente  $2j_\tau$  de la celda actual, mueve la cabeza una celda a la derecha y cambia de espacio en blanco a 1 el componente  $2j_\tau$  de la tupla, que allí se encuentra (si en lugar de una tupla detecta un espacio en blanco al moverse a la derecha, reemplaza éste con la  $2K$ -tupla  $(\triangle, \triangle, \triangle, \triangle, \dots, \triangle, \triangle)$  antes de escribir el 1).



**Figura 3.15** Simulación con una sola cinta de una máquina de Turing de tres cintas que ejecuta una operación de movimiento hacia la derecha en su segunda cinta

- c. Si la transición  $\tau$  es un movimiento hacia la izquierda, reemplaza con un espacio en blanco el 1 del componente  $2j_\tau$  de la celda actual, se mueve una celda a la izquierda y cambia de espacio en blanco a 1 el componente  $2j_\tau$  de la tupla que allí se encuentra (si en lugar de una tupla detecta el símbolo # al moverse hacia la izquierda, mueve la cabeza de nuevo hacia la izquierda; esto ocasionará que  $M'$  abandone los cálculos, como lo haría  $M$ ).
- B3. Utilizando como guía la marca especial # del extremo izquierdo de la cinta, devuelve la cabeza a la segunda celda de la cinta y pasa al estado compuesto de  $M'$  que refleje la configuración de  $M$  después de ejecutar la transición  $\tau$ .

Como ejemplo, la figura 3.15 presenta la simulación de una máquina de tres cintas que mueve la cabeza de su segunda cinta una celda hacia la derecha.

Si la ejecución de B1 a B3 conduce a un estado compuesto cuyo primer componente es el estado de parada de  $M$ , se han detenido los cálculos que se simulan. Por esto diseñamos  $M'$  de manera que también se detenga.

Al construirse de esta manera,  $M'$  únicamente simula las actividades de  $M$  y, por lo tanto, acepta una cadena si y sólo si  $M$  la hubiera aceptado. Por consiguiente, la máquina  $M'$  de cinta única acepta exactamente el mismo lenguaje que la máquina  $M$  de varias cintas.



Entre otras cosas, el teorema 3.1 refuerza nuestra confianza en la tesis de Turing. Al ampliar el potencial de los autómatas finitos con la adición de memoria para obtener la clase de los autómatas de pila, podemos vernos tentados a ampliar las máquinas de Turing de manera parecida. Sin embargo, el teorema 3.1 indica que esto no produciría un aceptador de lenguajes más poderoso. Podríamos modelar cualquier dispositivo de memoria que se quisiera agregar a una máquina de Turing mediante cintas adicionales, y por el teorema 3.1 sabemos que estas cintas adicionales no mejorarían el potencial de reconocimiento de lenguajes de la máquina. Por lo tanto, no podemos contradecir la tesis de Turing añadiendo memoria a las máquinas de Turing.

Otro fenómeno que apoya la tesis de Turing es que no se obtiene ningún poder de reconocimiento adicional si se introduce el no determinismo en el tratamiento de las máquinas de Turing. Nuestro primer paso para apoyar esta afirmación es presentar el concepto de una máquina de Turing no determinista.

## Máquinas de Turing no deterministas

Una máquina de Turing no determinista es similar a una máquina de Turing tradicional; la diferencia es que quizás una máquina no determinista no se

encuentre completamente definida o, lo que es más importante, puede que ofrezca más de una transición aplicable a un par estado-símbolo. Si una máquina de Turing no determinista llegara al par estado-símbolo actual sin que existiera una transición aplicable, la máquina abandonaría los cálculos. Si una máquina de Turing no determinista llegara al par estado-símbolo actual donde puede aplicarse más de una transición, la máquina llevaría a cabo una elección no determinista y proseguiría con sus cálculos mediante la ejecución de una de las opciones aplicables.

En resumen, una máquina de Turing no determinista se puede definir como una sexteta  $(S, \Sigma, \Gamma, \pi, \iota, h)$ , lo mismo que una máquina de Turing determinista, excepto que el cuarto componente es un subconjunto de  $((S - \{h\}) \times \Gamma) \times (S \times (\Gamma \cup \{L, R\}))$  en vez de una función de  $(S - \{h\}) \times \Gamma$  a  $S \times (\Gamma \cup \{L, R\})$ . En este contexto, es evidente que las máquinas de Turing no deterministas forman una clase de máquinas que contiene propiamente a las máquinas de Turing tradicionales (deterministas) que hemos analizado hasta ahora.

Decimos que una máquina de Turing no determinista  $M$  acepta una cadena  $w$  si es posible que  $M$  llegue a su estado de parada después de iniciar sus cálculos con la entrada  $w$ . Decimos *possible* ya que reconocemos que en el caso de una máquina no determinista, no alcanzar el estado de parada en un intento particular podría ser el resultado de una mala decisión de la máquina, más que de una cadena de entrada impropia. Definimos como lenguaje  $L(M)$  a la colección de todas las cadenas aceptadas por la máquina de Turing no determinista  $M$ . Así, una cadena  $w$  se encuentra en  $L(M)$  si y sólo si  $M$  dispone de opciones que le permitan alcanzar el estado de parada al recibir la entrada  $w$ .

Puesto que la máquina de Turing no determinista es una generalización de la máquina de Turing tradicional, puede aceptar todo lenguaje que acepta una máquina tradicional. Lo que resulta más interesante es que, como lo muestra el teorema siguiente, las máquinas de Turing no deterministas son incapaces de aceptar más lenguajes que las deterministas. Por lo tanto, como sucede con los autómatas finitos, la introducción del no determinismo no aumenta el potencial de reconocimiento de lenguajes de las máquinas de Turing.

### TEOREMA 3.2

Para cada máquina de Turing no determinista  $M$ , existe una máquina de Turing determinista  $D$  tal que  $L(M) = L(D)$ .

### DEMOSTRACIÓN

Suponga que  $M$  es una máquina de Turing no determinista que acepta el lenguaje  $L(M)$ . Debemos mostrar la existencia de una máquina de Turing determinista que acepte el mismo lenguaje. Esto se logra de manera indirecta, mostrando que existe una máquina de Turing determinista de tres cintas  $M'$  tal que  $L(M) = L(M')$  y, por el teorema

3.1, debe existir una máquina de Turing tradicional (determinista, de cinta única) que acepte  $L(M)$ .

Diseñamos  $M'$  para que pruebe de manera sistemática todas las opciones posibles para la máquina determinista  $M$ , con el objetivo de encontrar una combinación que lleve a la aceptación de la cadena de entrada. Se emplean las tres cintas de  $M'$  de la manera siguiente: la primera cinta contiene la cadena de entrada que se evalúa; la segunda cinta se emplea como "cinta de trabajo" en donde repetidamente  $M'$  copia la versión intacta de la cadena de entrada y luego, usando esta copia, simula una secuencia de transiciones de  $M$ ; la tercera cinta sirve para llevar el control de la secuencia de transiciones (de  $M$ ) que se aplica, así como las secuencias que ya se han simulado.

El proceso de copiado de la cadena de entrada de la cinta 1 en la cinta 2 implica dos aspectos sutiles, pero importantes. En primer lugar,  $M'$  debe desplazar la cadena una celda hacia la derecha durante el proceso de copiado. Es decir, hay que colocar el contenido de la celda uno de la cinta 1 en la celda dos de la cinta 2, la celda dos de la cinta 1 va a la celda tres de la cinta 2, etcétera. Esto permite que  $M'$  coloque un símbolo especial en la celda del extremo izquierdo de su segunda cinta, lo que a su vez permite que  $M'$  detecte una terminación anormal de  $M$  sin abortar sus propios cálculos. Si la secuencia de transiciones que se simula ocasiona que  $M$  rebase el extremo de la cinta,  $M'$  detectará esta marca especial, notará que la secuencia actual no es productiva y comenzará el proceso de evaluación de otra secuencia.

El segundo aspecto que está presente en el proceso de copiado es que  $M'$  debe colocar una marca especial en el extremo derecho de la cadena de su segunda cinta. Si esta marca se detecta al simular las transiciones de  $M$ , se desplaza hacia la derecha. Así, cuando  $M'$  necesita borrar esta cinta antes de comenzar otra simulación, sólo tiene que borrar hasta esta marca especial.

Por último, debemos indicar cómo  $M'$  lleva el control de la simulación de las secuencias de transiciones de  $M$ . La idea es bastante sencilla. En primer lugar, rotule cada uno de los arcos del diagrama de transiciones de  $M$  con un símbolo único. Si existieran sólo cinco arcos en el diagrama, se podrían emplear los dígitos 1, 2, 3, 4 y 5. Luego, construya un componente en  $M'$  que genere todas las cadenas de estos símbolos en forma sistemática utilizando la cinta 3 (como modelo, véase Fig 3.10). Cada una de estas cadenas representa una secuencia de transiciones y, a final de cuentas, estará representada cada una de las transiciones posibles.

Al utilizar este generador de secuencias interno, las actividades de  $M'$  se llevan a cabo de la siguiente manera:

1. Copia la cadena de entrada de la cinta 1 a la cinta 2, en la forma antes descrita.
2. Genera la siguiente secuencia de transiciones en la cinta 3.
3. Simula esta secuencia con la cinta 2.
4. Si esta simulación conduce a un estado de parada de  $M$ , se detiene. De lo contrario, borra la cinta 2 y regresa al paso 1.



En resumen, no podemos ampliar el potencial de reconocimiento de lenguajes de las máquinas de Turing añadiendo cintas o introduciendo un comportamiento no determinista, observación que apoya la tesis de Turing. Esto nos puede llevar a la conjectura de que la clase de lenguajes aceptados por máquinas de Turing representa el final de nuestra jerarquía de lenguajes que las máquinas pueden reconocer y, por lo tanto, son de importancia para nuestro estudio. Por esto, en las secciones siguientes nos dedicaremos a aprender más acerca de esta clase de lenguajes.

## Ejercicios

1. a. Diseñe una máquina de Turing que acepte el lenguaje  $\Sigma^*$ , donde  $\Sigma = \{x, y\}$ .  
b. Diseñe una máquina de Turing que acepte el lenguaje  $\emptyset$ .
2. Muestre que una máquina de Turing se puede modificar para que evite una terminación anormal pero a la vez acepte las mismas cadenas que antes.
3. Muestre que si se permite en una máquina de Turing de varias cintas que las transiciones individuales afecten a más de una cinta, entonces no aumenta el potencial de la máquina. En otras palabras, muestre que cualquier transición que opera sobre más de una cinta se puede simular con una secuencia de transiciones que operan cada una en una sola cinta.
4. Muestre que cualquier cálculo de una “máquina de Turing” cuya cinta se extiende infinitamente hacia la izquierda y hacia la derecha se puede simular con una máquina de Turing de dos cintas y, por lo tanto, con una máquina de Turing tradicional.

## 3.4 LENGUAJES ACEPTADOS POR MÁQUINAS DE TURING

En la sección anterior analizamos los rudimentos de las máquinas de Turing como aceptadores de lenguajes, y denominamos lenguajes aceptados por máquinas de Turing a los lenguajes que dos aceptan. En esta sección describiremos con mayor detenimiento esta clase de lenguajes.

## Comparación entre lenguajes aceptados por máquinas de Turing y lenguajes estructurados por frases

En el capítulo 1 presentamos las gramáticas estructuradas por frases y analizamos cómo una gramática de este tipo define un lenguaje que consiste en las cadenas de terminales generados por la gramática. Al restringir las formas de las reglas de reescritura disponibles, hemos podido identificar clases de gramáticas que generan lenguajes regulares e independientes del contexto. Ahora queremos considerar las gramáticas estructuradas por frases que no tienen restricción alguna en cuanto a sus reglas de reescritura. Así, tanto el lado izquierdo como el derecho de las reglas de reescritura pueden consistir en cualquier cadena finita de terminales y no terminales, siempre y cuando exista por lo menos un no terminal en el lado izquierdo.

Los lenguajes que generan estas gramáticas se conocen como **lenguajes estructurados por frases**. Éstos son los lenguajes que pueden definirse “gramaticalmente”, en el sentido de que sus estructuras de cadenas se pueden analizar empleando una jerarquía de estructuras de frases. Puesto que las gramáticas regulares y las independientes del contexto son casos especiales de las gramáticas sin restricciones, los lenguajes que las primeras generan están contenidos en la clase de los lenguajes estructurados por frases. Además, la figura 3.16 muestra una gramática sin restricciones que genera el lenguaje  $\{x^n y^n z^n : n \in \mathbb{N}\}$ , el cual, como ya sabemos, no es independiente del contexto. Por ello, los lenguajes estructurados por frases constituyen una clase de lenguajes más extensa que la de los independientes del contexto.

En esta sección, nuestro objetivo es caracterizar a los lenguajes estructurados por frases como aquellos que las máquinas de Turing pueden aceptar. Es decir, los *lenguajes estructurados por frases* son precisamente los *lenguajes aceptados por máquinas de Turing*. Esto se demostrará en dos etapas. Primero, en el teorema 3.3 mostramos que todo lenguaje aceptado por máquinas de Turing es un lenguaje estructurado por frases; luego, en el teorema 3.4 mostramos que todo lenguaje estructurado por frases es aceptado por máquinas de Turing.

La demostración que daremos para el teorema 3.3 se basa en la construcción de una gramática que genera el mismo lenguaje que el aceptado por una máquina de Turing determinada. Esta construcción requiere un sistema de notación para representar la configuración total de una máquina de Turing en cualquier etapa de sus cálculos. Al emplear este sistema, el contenido de la

$$\begin{aligned} S &\rightarrow xyNSz \\ S &\rightarrow \lambda \\ yNx &\rightarrow xyN \\ yNz &\rightarrow yz \\ yNy &\rightarrow yyN \end{aligned}$$

**Figura 3.16** Gramática que genera el lenguaje  $\{x^n y^n z^n : n \in \mathbb{N}\}$

cinta de la máquina se representa como una cadena de símbolos de cinta encerrados entre corchetes. Primero representamos con [ el extremo izquierdo de la cinta, luego presentamos la cadena de símbolos que se encuentran en la cinta, comenzando por la celda del extremo izquierdo e incluyendo por lo menos un espacio en blanco después del último símbolo distinto de un espacio, y por último cerramos la representación con ]. De esta manera, una cinta que contiene  $\Delta xxyx\Delta\Delta\Delta\dots$  se podría representar como  $[\Delta xxyx\Delta]$  o quizás como  $[\Delta xxyx\Delta\Delta\Delta\Delta\Delta\Delta]$ , y a una cinta que contiene  $\Delta\Delta\Delta x\alpha\Delta y\alpha\Delta x\Delta\Delta\Delta\dots$  como  $[\Delta\Delta\Delta x\alpha\Delta y\alpha\Delta x\Delta\Delta]$ .

Para completar la representación de la configuración de la máquina, insertamos en nuestra representación de la cinta el estado actual justo a la izquierda del símbolo actual. Así, si  $p$  fuera un estado de la máquina, entonces  $[\Delta xpxyxx\Delta]$  representaría la configuración en la cual  $p$  es el estado actual y la configuración de la cinta es  $\Delta xxyx\Delta\Delta\Delta\dots$ . De manera parecida,  $[p\Delta xy\Delta]$  representaría a la máquina en el estado  $p$  con la configuración  $\Delta xy\Delta\Delta\dots$ . (Podemos suponer que los símbolos empleados para representar los estados de la máquina son distintos de los símbolos de cinta de la máquina.)

La importancia que para nosotros tiene esta notación es que nos ofrece un medio para expresar los cálculos de una máquina de Turing como secuencia de cadenas de símbolos, donde cada cadena representa la configuración de la máquina en un instante determinado de los cálculos. Así, si la máquina acepta cadenas deteniéndose con la cinta configurada como  $\Delta Y\Delta\Delta\Delta\dots$ , entonces el

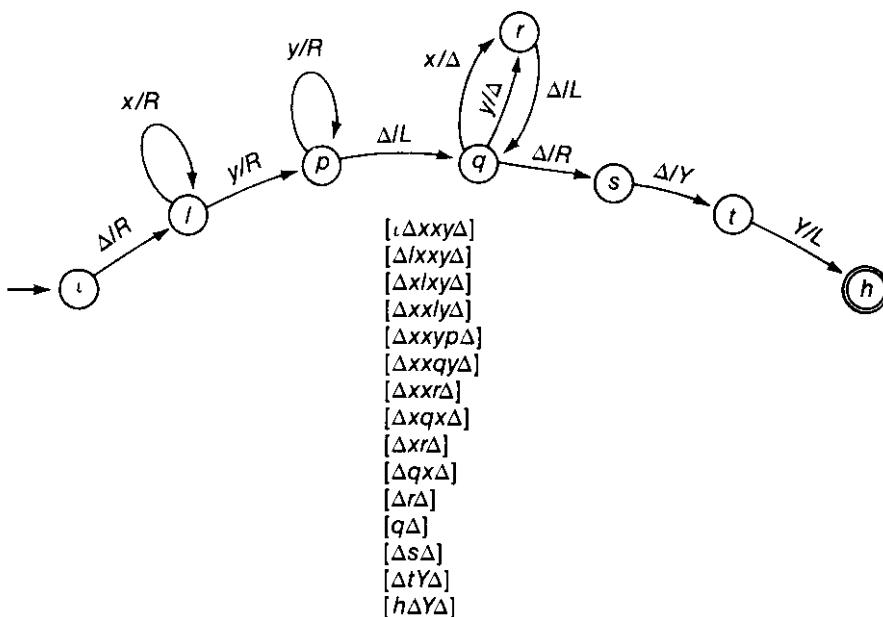


Figura 3.17 Diagrama de transiciones parcial para una máquina de Turing y la secuencia de configuraciones que produce al procesar la cadena  $xy$

proceso de aceptación de la cadena  $w$  se podría resumir como la secuencia de configuraciones de la máquina que comienza con  $[i\Delta w\Delta]$  y termina con  $[h\Delta Y\Delta]$ , donde  $i$  y  $h$  representan los estados de inicio y parada de la máquina, respectivamente.

La figura 3.17 muestra una máquina de Turing que acepta el lenguaje  $\{x^n y^n : m \in \mathbb{N}, n \in \mathbb{N}^+\}$  y la secuencia de configuraciones que produce al procesar la cadena  $xyy$ . Esta secuencia comienza con una configuración que contiene la cadena de entrada y termina con la configuración  $[h\Delta Y\Delta]$ . Así, al leerla de atrás hacia adelante obtenemos la secuencia que conduce de la configuración  $[h\Delta Y\Delta]$  a una configuración que contiene la cadena de entrada; esta secuencia es bastante similar a una derivación de la cadena. De hecho, si construyéramos una gramática en la cual toda derivación tuviera que comenzar con el patrón  $[h\Delta Y\Delta]$  y cuyas reglas de reescritura simularon la acción inversa de una transición de la máquina, entonces estaríamos en camino de construir una gramática que generara las cadenas aceptadas por la máquina. Éste es el enfoque que usaremos para demostrar el teorema 3.3.

### TEOREMA 3.3

Todo lenguaje aceptado por máquinas de Turing es un lenguaje estructurado por frases.

### DEMOSTRACIÓN

Nuestra tarea es mostrar que para cualquier lenguaje  $L$  aceptados por máquinas de Turing existe una gramática estructurada por frases que genera  $L$ . Para esto seleccionamos una máquina de Turing  $M$  que acepte cadenas sólo deteniéndose con su cinta configurada como  $\Delta Y\Delta\Delta\Delta\dots$  y para la cual  $L(M) = L$ .

A partir de esta máquina definimos una gramática  $G$  de la manera siguiente: los no terminales de  $G$  se definen como  $S$  (el símbolo inicial de la gramática),  $[ , ]$ , los símbolos que representan los estados de  $M$ , y los símbolos de cinta de  $M$ , incluidos  $\Delta$  y  $Y$ . Los terminales de  $G$  son los símbolos del alfabeto de  $M$ .

La única regla de reescritura que contiene el símbolo inicial es la regla

$$S \rightarrow [h\Delta Y\Delta]$$

Esta regla garantiza que cualquier derivación basada en esta gramática comenzará al final de alguna secuencia de configuraciones de la máquina. También introducimos la regla

$$\Delta] \rightarrow \Delta\Delta]$$

que permite que una derivación amplíe la cadena  $[h\Delta Y\Delta]$  a cualquier longitud que se desee.

A continuación, introducimos reglas de reescritura que simulan transiciones a la inversa. Para cada transición de la forma  $\delta(p, x) = (q, y)$  introducimos la regla de reescritura

$$qy \rightarrow px$$

(De esta manera,  $[\triangle z q y \triangle]$  puede reescribirse como  $[\triangle z p x \triangle]$ , lo cual refleja que si la configuración de  $M$  fuera  $[\triangle z p x \triangle]$ , la aplicación de  $\delta(p, x) = (q, y)$  desplazaría  $M$  hacia  $[\triangle z q y \triangle]$ .)

Para cada transición de la forma  $\delta(p, x) = (q, R)$ , introducimos la regla

$$xq \rightarrow px$$

(P. ej.,  $[\triangle x q y z \triangle]$  se podría reescribir como  $[\triangle p x y z \triangle]$ .)

Para cada transición de la forma  $\delta(p, x) = (q, L)$  y cada símbolo de cinta  $y$  de  $M$ , introducimos la regla

$$qyx \rightarrow ypx$$

(Así,  $[\triangle q y x \triangle \triangle]$  se podría reescribir como  $[\triangle y p x \triangle \triangle]$ .)

La lista de reglas de reescritura de  $G$  se completa introduciendo tres reglas que permiten que una derivación elimine los no terminales  $[, i, \triangle y ]$  en ciertas circunstancias. Estas reglas son

$$\begin{aligned} [i\triangle &\rightarrow \lambda \\ \triangle\triangle] &\rightarrow \triangle\triangle \end{aligned}$$

y

$$\triangle\triangle \rightarrow \lambda$$

(Por consiguiente, si una derivación produjera la configuración inicial  $[i\triangle x y x \triangle \triangle \triangle]$ , podría eliminar los no terminales para producir la cadena  $x y x$ .)

Por último, planteamos que  $L(M) = L(G)$ . Si  $w$  fuera una cadena de  $L(M)$ , existiría una secuencia de configuraciones de  $M$  que comenzaría con  $[i\triangle w \triangle]$  y terminaría con  $[h\triangle Y \triangle]$ . Por lo tanto, podemos producir una derivación de la cadena  $w$  de la forma

$$S \Rightarrow [h\triangle Y \triangle] \Rightarrow \cdots \Rightarrow [i\triangle w \triangle] \Rightarrow w\triangle \Rightarrow w$$

Comenzamos por aplicar la regla  $S \rightarrow [h\triangle Y \triangle]$  y luego la regla  $\triangle \rightarrow \triangle\triangle$  repetidamente hasta que la cadena  $[h\triangle Y \triangle \triangle \cdots \triangle]$  sea tan larga como cualquier configuración de la secuencia que represente los cálculos de  $M$ . Después aplicamos en orden inverso las reglas de reescritura correspondientes a las transiciones de la secuencia de configuraciones original. Esto producirá el patrón  $[i\triangle w \triangle \triangle \cdots \triangle]$ , el cual podemos

reducir a  $w$  aplicando las reglas  $\triangle\triangle] \rightarrow \triangle], \triangle] \rightarrow \lambda$ , y  $[\iota\triangle \rightarrow \lambda$ . Como resultado,  $w$  estaría en  $L(G)$  (como ejemplo, la figura 3.18 presenta la derivación que corresponde a la secuencia de configuraciones proporcionadas en la figura 3.17).

A la inversa, si tuviéramos una cadena en  $L(G)$ , su derivación daría origen a una secuencia de configuraciones que mostrarían a su vez cómo podría aceptar  $M$  la cadena. Así, cualquier cadena de  $L(G)$  también se encuentra en  $L(M)$ .



El siguiente teorema es lo único que nos falta para justificar nuestra afirmación de que los lenguajes estructurados por frases son exactamente los lenguajes aceptados por máquinas de Turing.

### TEOREMA 3.4

Todo lenguaje estructurado por frases es un lenguaje aceptado por máquinas de Turing.

#### DEMOSTRACIÓN

Comenzamos por observar que al aplicar la demostración del teorema 3.1 a una máquina de Turing no determinista de varias cintas se produciría una máquina no determinista de cinta única que aceptaría el mismo lenguaje que la máquina con varias cintas (es posible que la transición por simular de un estado compuesto de la forma  $(p, x_1, x_2, \dots, x_k)$  ya no esté determinada en forma única, pero entonces se conocerán todas las opciones antes de la ejecución y, por tanto, podrán incorporarse al autómata permitiendo el no determinismo). Así, para cualquier máquina de Turing no determinista  $M$  de varias cintas existe una máquina de Turing  $M'$  de una sola cinta tal que  $L(M) = L(M')$ . Tal observación nos permite demostrar este teorema mostrando que para cada gramática  $G$  existe una máquina de Turing no determinista  $N$  de dos cintas tal que  $L(G) = L(N)$ . Después,  $N$  se podría simular con una máquina de Turing no determinista de cinta única (por la observación anterior), la cual podría ser simulada a su vez por una máquina de Turing convencional (según el Teorema 3.2).

A continuación observamos que una máquina de Turing puede realizar cualquier regla de reescritura de la gramática. Es decir, si una cadena de símbolos  $v$  aparece en algún lugar de la cinta de la máquina y la gramática contiene la regla  $v \rightarrow w$ , donde  $w$  representa una cadena (posiblemente vacía) de terminales y no terminales, entonces la máquina puede reemplazar la cadena  $v$  por la cadena  $w$  aplicando operaciones de escritura y de desplazamiento hacia la izquierda y hacia la derecha.

$$\begin{aligned}
 S &\Rightarrow [h\Delta Y\Delta] \\
 &\Rightarrow [h\Delta Y\Delta\Delta] \\
 &\Rightarrow [h\Delta Y\Delta\Delta\Delta] \\
 &\Rightarrow [\Delta t Y\Delta\Delta\Delta] \\
 &\Rightarrow [\Delta s\Delta\Delta\Delta\Delta] \\
 &\Rightarrow [q\Delta\Delta\Delta\Delta\Delta] \\
 &\Rightarrow [\Delta r\Delta\Delta\Delta\Delta] \\
 &\Rightarrow [\Delta qx\Delta\Delta\Delta] \\
 &\Rightarrow [\Delta xr\Delta\Delta\Delta] \\
 &\Rightarrow [\Delta xqx\Delta\Delta] \\
 &\Rightarrow [\Delta xxr\Delta\Delta] \\
 &\Rightarrow [\Delta xxqy\Delta] \\
 &\Rightarrow [\Delta xxyp\Delta] \\
 &\Rightarrow [\Delta xxly\Delta] \\
 &\Rightarrow [\Delta xly\Delta] \\
 &\Rightarrow [\Delta/xx\Delta] \\
 &\Rightarrow [\Delta/xx\Delta] \\
 &\Rightarrow [xxy\Delta] \\
 &\Rightarrow xxy
 \end{aligned}$$

**Figura 3.18** Derivación de la cadena  $xx\bar{y}$  correspondiente a la secuencia de configuraciones de la figura 3.17

Construimos ahora una máquina no determinista de dos cintas que opera de la siguiente manera: utiliza la cinta 1 para almacenar la cadena de entrada por evaluar. Escribe el símbolo inicial de la gramática en la cinta 2. Luego aplica repetidamente y de manera no determinista las reglas de reescritura a la cadena de la cinta 2 (decimos “de manera no determinista” ya que puede existir más de una regla aplicable en un momento determinado). Si el contenido de la cinta 2 se convierte en una cadena con sólo terminales, compara esta cadena con la cadena de entrada que está almacenada en la cinta 1. Si ambas cadenas son idénticas, la máquina se detiene; de lo contrario, mueve una de las cabezas hacia la izquierda hasta que ocurra una terminación anormal.

En resumen, este proceso únicamente emplea la cinta 2 para calcular una derivación con base en las reglas de la gramática. Si la cadena de entrada se puede derivar de la gramática, entonces es posible que esta cadena sea la producida en la cinta 2. En este caso, la máquina aceptará la entrada, deteniéndose. Sin embargo, si la entrada no puede derivarse de la gramática, la cadena producida en la cinta 2 nunca podrá ser igual a la de entrada, y sería imposible que la máquina aceptara la cadena. Por consiguiente, el lenguaje aceptado por la máquina es el lenguaje generado por la gramática.



Como resumen, los lenguajes y las máquinas estudiados hasta ahora forman la jerarquía que se presenta en la figura 3.19.

### Alcance de los lenguajes estructurados por frases

La equivalencia entre los lenguajes estructurados por frases y los lenguajes aceptados por máquinas de Turing significa que las máquinas de Turing se pueden utilizar para estudiar el alcance de los lenguajes estructurados por frases. Esto es la esencia del teorema siguiente.

#### TEOREMA 3.5

Para cada alfabeto  $\Sigma$  existe al menos un lenguaje sobre  $\Sigma$  que no es un lenguaje estructurado por frases..

#### DEMOSTRACIÓN

Sea  $L$  un lenguaje estructurado por frases del alfabeto  $\Sigma$ . Entonces, por el teorema 3.4, existe una máquina de Turing  $M$  tal que  $L(M) = L$ . Comenzamos nuestra demostración señalando que existe una  $M$  cuyos símbolos de cinta se eligen del conjunto  $\Sigma \cup \{\Delta\}$ .

Si  $M$  es la máquina de Turing tal que  $L(M) = L$  y los símbolos de cinta de  $M$  incluyen más que  $\Sigma \cup \{\Delta\}$ , podemos construir otra máquina de Turing  $M'$  con símbolos de cinta de  $\Sigma \cup \{\Delta\}$ , de modo que  $L(M') = L(M) = L$ . De hecho, sea  $x$  cualquier símbolo (distinto de espacio en blanco) de  $\Sigma$ . Entonces, disponga en una lista los símbolos distintos de espacios en blanco de  $M$  y represente cada entrada de la lista con una cadena de  $x$  de longitud igual a la posición del símbolo

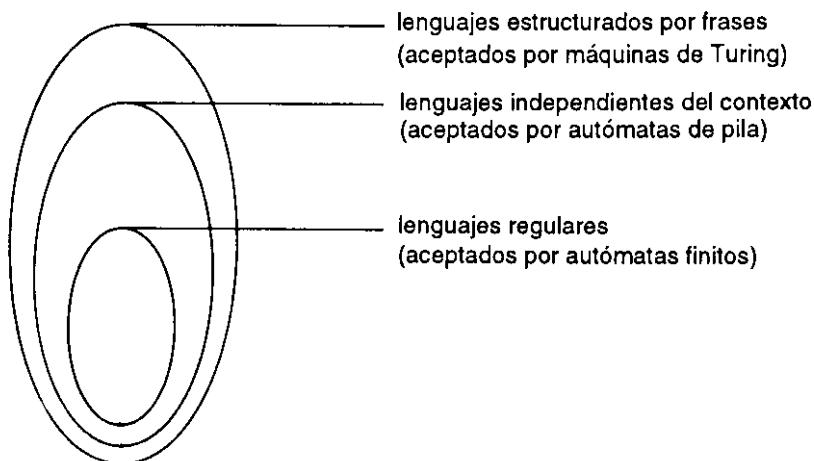


Figura 3.19 Jerarquía de máquinas y lenguajes

en esta lista (el primer símbolo se representa con  $x$ , el segundo con  $xx$ , etc.). Así, cada uno de los símbolos de la cinta de  $M$  que no sea un espacio en blanco, y por consiguiente también cada símbolo del alfabeto de  $M'$ , se representa con una cadena de  $x$  única. De esta manera, siempre es posible representar el contenido de la cinta de  $M$  por medio de cortas cadenas de  $x$  separadas por espacios en blanco (un espacio en blanco de la cinta de  $M$  se codificaría como dos espacios consecutivos, es decir, una cadena de  $x$  vacía).

Ahora construya  $M'$  para que traduzca su entrada a esta forma codificada, simule las acciones de  $M$  y se detenga únicamente si  $M$  se detiene. Así,  $M'$  aceptará exactamente las mismas cadenas que acepta  $M$ , sin emplear más que los símbolos de cinta  $\Sigma \cup \{\triangle\}$ .

Concluimos que dado un lenguaje estructurado por frases de un alfabeto  $\Sigma$ , existe una máquina de Turing con símbolos de cinta  $\Sigma \cup \{\triangle\}$  tal  $L(M) = L$ .

Lo que queda de nuestra demostración es, en esencia, lo mismo que la demostración del teorema 1.1. Podemos generar de manera sistemática una lista de todas las máquinas de Turing con símbolos de cinta  $\Sigma \cup \{\triangle\}$ , generando primero aquellas máquinas con sólo dos estados (recuerde que una máquina de Turing tiene cuando menos dos estados), seguido por las que tienen tres estados, etcétera. Por lo tanto, existe una cantidad contable de tales máquinas de Turing. Por otra parte, existe una cantidad infinita de cadenas en  $\Sigma^*$  y, debido a ello, es incontable el número de lenguajes que se pueden formar a partir de  $\Sigma$ . Por consiguiente, existen más lenguajes basados en  $\Sigma$  que máquinas de Turing con símbolos de cinta en  $\Sigma \cup \{\triangle\}$ . Así mismo, deben existir lenguajes basados en  $\Sigma$  que no son lenguajes estructurados por frases.



Aquí es donde comienza a complicarse la trama. Los teoremas 3.3, 3.4 y 3.5 nos dicen que existen lenguajes que no tienen bases gramaticales y además que las máquinas de Turing sólo pueden reconocer aquellos lenguajes que sí tienen bases gramaticales. Si aceptamos la tesis de Turing de que las máquinas de Turing engloban la esencia de cualquier proceso computacional, debemos concluir que ningún proceso algorítmico puede reconocer los lenguajes que no tienen bases gramaticales.

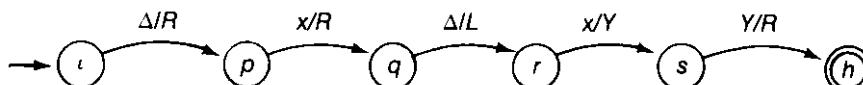
Esta conclusión no sólo implica que existen lenguajes que no podemos analizar sintácticamente con un computador, sino además tiene ramificaciones relacionadas con la búsqueda de sistemas que comprendan los lenguajes naturales, un área de gran actividad en las investigaciones recientes. De hecho, nos indica que en el desarrollo de un sistema de procesamiento de lenguajes naturales existe el requisito de que el lenguaje que se procesa tenga una estructura gramatical bien definida; si el lenguaje natural no cuenta con esta estructura, no seremos capaces de procesarlo algorítmicamente.

Este requisito tiene además un estrecho vínculo con el tema de la inteligencia natural comparada con la inteligencia artificial. Si, como muchos creen, la mente humana opera ejecutando algoritmos, entonces la tesis de Turing indica que existen lenguajes cuya sintaxis no puede analizar la mente humana. Por otra parte, si la inteligencia natural se basa en un nivel más poderoso que la ejecución de algoritmos, entonces la tesis de Turing sugiere que el desarrollo de máquinas verdaderamente inteligentes seguirá por siempre siendo un sueño.

Vemos entonces que la teoría presentada hasta ahora tiene consecuencias significativas. El único punto débil en la cadena es la tesis de Turing, la *conjetura* de que el concepto de computación con una máquina de Turing es tan poderoso (aunque quizás no tan conveniente) como cualquier otro modelo computacional. Como mencionamos antes, la mayoría de los científicos de la computación acepta actualmente esta tesis. Gran parte de esta aceptación proviene de hecho de que la tesis de Turing es congruente con otros resultados y conjeturas que han surgido al estudiar la computación desde otras perspectivas; en el capítulo siguiente consideraremos algunas de estas estrategias.

## Ejercicios

- Dibuje un diagrama de transiciones para todas las máquinas de Turing con símbolos de cinta  $x$  y  $\Delta$  que sólo tengan dos estados (uno inicial y uno de parada). ¿Cómo se relacionan su capacidad para llevar a cabo esta tarea y el hecho de que sea contable la colección de todas las máquinas de Turing cuyo conjunto de símbolos de cinta es  $\{x, \Delta\}$ ?
- Muestre que al agregar una segunda pila a un autómata de pila obtenemos una clase de máquinas con el mismo poder de aceptación de lenguajes que la clase de máquinas de Turing.
- Utilice el proceso de construcción descrito en la demostración del teorema 3.3, para desarrollar una gramática estructurada por frases que acepte el mismo lenguaje aceptado por la máquina de Turing cuyo diagrama de transiciones aparece a continuación (la máquina acepta cadenas deteniéndose con la cinta configurada como  $\Delta Y \Delta \Delta \Delta \dots$ ) Con la gramática que obtenga, muestre una derivación de la cadena  $x$ .



- Muestre que el lenguaje  $\{x^n y^{2^n} z^{4^n} : n \in \mathbb{N}\}$  es aceptado por máquinas de Turing.

### 3.5 MÁS ALLÁ DE LOS LENGUAJES ESTRUCTURADOS POR FRASES

Hemos demostrado la existencia de lenguajes que no están estructurados por frases (o, lo que es lo mismo, no son aceptados por máquinas de Turing), pero aún no identificamos explícitamente un lenguaje de éstos. Un objetivo fundamental de esta sección es llenar este hueco; la presentación de las máquinas de Turing universales y la distinción entre lenguajes aceptables y decidibles representan importantes extensiones de nuestro análisis.

#### Sistema de codificación de máquinas de Turing

Para identificar un lenguaje que no es aceptado por máquinas de Turing (y por lo tanto no es estructurado por frases) necesitaremos un sistema de codificación con el cual podamos representar las máquinas de Turing con alfabeto  $\Sigma$  y símbolos de cinta  $\Sigma \cup \{\Delta\}$  como cadenas que únicamente contienen ceros y unos. Por ello, hacemos una pausa para presentar dicho sistema antes de proseguir con el tema principal de la sección.

Dada una máquina  $M$  por representar, nuestro sistema de codificación necesita que acomodemos los estados de  $M$  en una lista cuyo primer elemento corresponda al estado inicial y el segundo al estado de parada. Con base en el orden de esta lista, podemos hablar del primer estado de  $M$ , del segundo estado de  $M$  y, en general, del estado  $j$  de  $M$ . Establecemos que el estado  $j$  de  $M$  se representará con una cadena de ceros de longitud  $j$ . Así, el estado inicial estará representado por 0, el estado de parada por 00 y el siguiente estado (si existe) por 000.

Luego, representamos los símbolos  $L$  y  $R$ , así como los símbolos en  $\Sigma$  (los símbolos de cinta de  $M$  distintos de espacio en blanco) como cadenas de ceros. Esto se hace acomodando en una lista los símbolos de  $\Sigma$  y representando  $L$  con 0,  $R$  con 00, el primer símbolo de la lista con 000, el segundo símbolo con 0000 y, en general, el símbolo  $j$  con una cadena de ceros de longitud  $j + 2$ .

Si ahora establecemos que el símbolo del espacio en blanco se representará con la cadena vacía, obtenemos un sistema en el cual podemos representar los símbolos  $L$  y  $R$ , los estados de  $M$  y los símbolos de la cinta de  $M$  por medio de cadenas de ceros. A su vez, esto nos permite representar cualquier transición de  $M$  como una cadena de ceros y unos. A fin de cuentas, se puede identificar cualquier transición (que debe tener la forma  $\delta(p, x) = (q, y)$  con una cuádrupla  $(p, x, q, y)$  donde  $p$  es el estado actual,  $x$  es el símbolo actual,  $q$  es el nuevo estado y  $y$  es o bien un símbolo de cinta (si la transición es una operación de escritura) o bien  $L$  o  $R$  (si la transición es una operación de movimiento de la cabeza)). De esta manera, es posible representar toda la transición como cuatro cadenas de ceros separados por unos. Por ejemplo, la cadena 01000100100 representaría la transición  $\delta(1, x) = (h, R)$ , donde  $x$  es el símbolo representado por 000 y  $h$  es el estado de parada de la máquina. Puesto que un espacio en blanco se representa

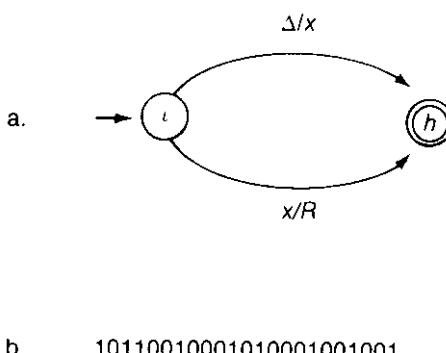
con la ausencia de ceros, la cadena 011001 representa la transición  $\delta(t, \triangle) = (h, \triangle)$ , donde  $h$  es una vez más el estado de parada de la máquina.

Observe que una lista de todas las transiciones disponibles para una máquina de Turing con símbolos de cinta  $\Sigma \cup \{\triangle\}$  constituye una descripción completa de la máquina. Por lo tanto, cualquier máquina de este tipo se puede representar como una lista de transiciones codificada. Adoptaremos la regla convencional de añadir un 1 al inicio y al final de la lista, y un solo 1 para separar las transiciones de la lista. Así, la cadena 10110010001010001001001 (un 1 introductorio seguido por el código de transición 011001000, un 1 separador, el código 01000100100 y un 1 final) representa la máquina con dos transiciones,  $\delta(t, \triangle) = (h, x)$  y  $\delta(t, x) = (h, R)$ , que se muestra en la figura 3.20.

Al representar de esta forma una máquina de Turing, establecemos que las transiciones por incluir en la lista tendrán el orden siguiente: primero presentamos todas las transiciones que surgen del estado 0 (el inicial), luego las que se originan del estado 000, seguidas por las que tienen su origen en el estado 0000, etc., hasta incluir todos los estados donde pueden originarse transiciones. Las transiciones correspondientes a un estado se acomodan de acuerdo con el símbolo que se requiere en la celda actual de la cinta, comenzando por la transición que requiere un espacio en blanco como símbolo actual, luego la transición que requiere el símbolo con código 000, después la transición cuyo símbolo actual es 0000, etcétera. Esta uniformidad hace más fácil evaluar una cadena de ceros y unos para ver si constituye una representación válida de alguna máquina de Turing (determinista y, por lo tanto, completamente definida).

### Un lenguaje no estructurado por frases

Al emplear esta forma de codificación, encontramos que cada máquina de Turing con alfabeto  $\Sigma$  y símbolos de cinta  $\Sigma \cup \{\triangle\}$  se puede representar como una cadena de ceros y unos, la cual a su vez puede interpretarse como un entero



**Figura 3.20** a. Máquina de Turing sencilla  
b. La misma máquina en forma codificada

no negativo escrito en binario. Es más, si contáramos en binario llegaríamos en algún momento al patrón que representa una máquina de Turing específica cualquiera con símbolos de cinta  $\Sigma \cup \{\Delta\}$ . Por supuesto, las representaciones binarias de muchos enteros no negativos no corresponden a representaciones de máquinas válidas. Establezcamos que cada uno de estos enteros se asociará con la sencilla máquina que se muestra en la figura 3.21. Así, tenemos entonces una función de  $\mathbb{N}$  sobre las máquinas de Turing con alfabeto  $\Sigma$  y símbolos de cinta  $\Sigma \cup \{\Delta\}$ . Empleamos la notación  $M_i$  para representar la máquina que esta función relaciona con el entero  $i$ .

Con base en esta función podemos construir otra, esta vez de  $\Sigma^*$  sobre la colección de máquinas de Turing con alfabeto  $\Sigma$  y símbolos de cinta  $\Sigma \cup \{\Delta\}$ . Basta con asociar una cadena  $w$  de  $\Sigma^*$  con la máquina  $M_{|w|}$ , donde  $|w|$  representa la longitud de  $w$ . A fin de hacer más sencilla esta notación, utilizaremos  $M_w$  para representar la máquina asociada a  $w$  por medio de esta función.

Observe que los símbolos de  $w$  también se encuentran en el alfabeto de  $M_w$ , por lo que tiene sentido aplicar  $M_w$  a la cadena de entrada  $w$ . Definimos que el lenguaje  $L_0$  es el subconjunto  $\{w: M_w \text{ no acepta } w\}$  de  $\Sigma^*$ ; una cadena  $w$  de  $\Sigma^*$  se halla en  $L_0$  si y sólo si ésta no es aceptada por su máquina  $M_w$  correspondiente.

Ahora nuestra tarea es mostrar que  $L_0$  no es aceptado por máquinas de Turing. Para esto, mostramos que se llega a una contradicción si se supone lo contrario. Si  $L_0$  fuera aceptado por alguna máquina de Turing, utilizando un argumento similar al de la demostración del teorema 3.5 podemos suponer que el alfabeto de la máquina es  $\Sigma$  y su conjunto de símbolos de cinta es  $\Sigma \cup \{\Delta\}$ . Así mismo,  $L_0$  debe ser aceptado por  $M_{w_0}$  para alguna cadena  $w_0$  en  $\Sigma^*$  (toda máquina de Turing con alfabeto  $\Sigma$  y símbolos de cinta  $\Sigma \cup \{\Delta\}$  es  $M_w$  para una  $w$  de  $\Sigma^*$ ). Por lo tanto,  $L_0 = L(M_{w_0})$ .

Ahora nos preguntamos si la cadena  $w_0$  existe o no en  $L(M_{w_0})$  (tiene que estar o no estar), pero, como veremos en un momento, ambas opciones nos llevan a contradicciones. Con base en la definición de  $L_0$ , sabemos que  $w_0 \in L(M_{w_0})$  implica que  $w_0 \notin L_0$ , y que  $w_0 \notin L(M_{w_0})$  implica que  $w_0 \in L_0$ . Sin embargo, como  $L_0 = L(M_{w_0})$ , ambos enunciados son contradictorios. En vista de esta paradoja, debemos concluir que nuestra conjectura con respecto a la aceptación de  $L_0$  es falsa; en otras palabras, el lenguaje  $L_0$  no es aceptado por su máquina de Turing.

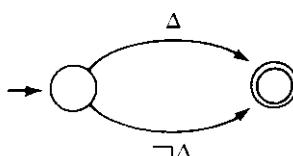


Figura 3.21 Máquina de Turing sencilla

## Máquinas de Turing universales

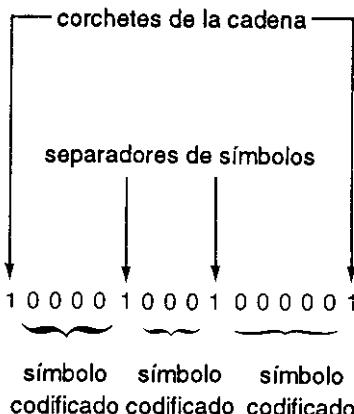
Nuestro siguiente objetivo es mostrar que el complemento de  $L_0$ , el conjunto  $\{w : M_w \text{ acepta } w\}$ , es aceptado por máquinas de Turing. Esto demostrará que la capacidad de una máquina de Turing para aceptar un lenguaje no es simétrica con la capacidad para rechazar el complemento del lenguaje. En otras palabras, existen casos donde se puede construir una máquina de Turing que identifique las cadenas de un lenguaje, pero no se puede construir una máquina de Turing que identifique las cadenas que no están en el lenguaje.

Sin embargo, para mostrar que el complemento de  $L_0$  es aceptado por máquinas de Turing, necesitamos presentar el concepto de **máquina de Turing universal**. Ésta no es más que una máquina de Turing "programable" que, dependiendo de su programa, puede simular cualquier otra máquina de Turing. De esta forma, las máquinas de Turing universales son los antepasados abstractos de nuestros modernos computadores programables que leen y ejecutan los programas almacenados en sus memorias. Es más, las máquinas de Turing universales están diseñadas para ejecutar programas que se almacenan en sus cintas.

Un programa para una máquina de Turing universal no es más que una versión codificada de una máquina de Turing que lleva a cabo la tarea que se desea ejecutar la máquina universal. Suponga que queremos programar una máquina de Turing universal para que desempeñe una actividad específica. Primero diseñaríamos una máquina de Turing tradicional que realizara la tarea; luego codificaríamos esta máquina como una cadena de ceros y unos, como ya lo hemos hecho. La cadena de código sería el programa de la máquina universal.

El siguiente paso sería codificar los datos que se usarían como entrada para los cálculos deseados. Para esto recordamos que a cada símbolo de la cinta distinto de espacio en blanco, correspondiente a la máquina que acabamos de codificar, se le asigna un código de tres o más ceros. Por consiguiente, cualquier cadena de estos símbolos se puede representar con la secuencia de códigos adecuada. En esta secuencia separamos con un 1 los códigos adyacentes y encerramos entre corchetes toda la secuencia, con un 1 en cada extremo (Fig. 3.22). Observe que la cadena vacía estaría representada por 11, una secuencia vacía de códigos.

Después de codificar la máquina que se simulará y la cadena que servirá como entrada, colocamos estos códigos en la cinta de entrada de la máquina de Turing universal, como se describe a continuación. La celda del extremo izquierdo permanece en blanco, luego se encuentra la versión codificada de la máquina que se simulará, seguida por la cadena de entrada codificada (la máquina universal puede detectar la separación entre la codificación de la máquina y los datos, ya que el código de la máquina termina con un 1 y el código de los datos comienza con 1. Para ser más precisos, toda transición codificada debe comenzar con el código de un estado, lo que requiere por lo menos un 0. Así, es posible detectar el fin de la lista de transiciones con la presencia de un código de estado incorrecto. Véase Fig. 3.23).

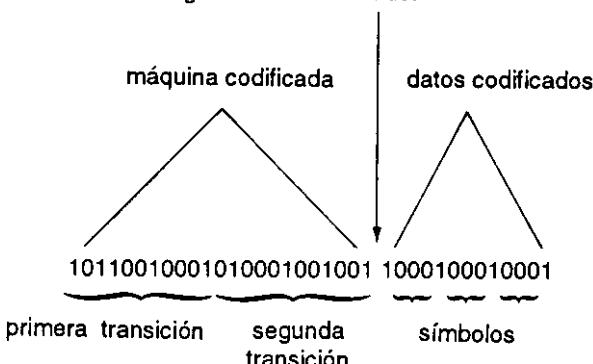


**Figura 3.22** Cadena de tres símbolos en forma codificada

Una vez que esta información se ha colocado en la cinta de la máquina universal, ubicamos la cabeza de la máquina sobre la celda del extremo izquierdo de la cinta y ponemos en marcha la máquina a partir de su estado inicial. Partiendo de esta configuración, la máquina de Turing universal extrae y simula las transiciones que encuentra en la primera porción de la cinta conforme manipula la cadena existente en la segunda porción de la cinta.

Para ser más precisos, podríamos visualizar una máquina de Turing universal como una máquina de tres cintas. La primera cinta se usa para almacenar el programa de entrada y los datos, así como para contener cualquier salida; la segunda cinta sirve como área de trabajo, en la cual se manipulan los datos; y la tercera cinta se emplea para contener una representación del estado actual de la máquina simulada.

Esto no puede ser el inicio  
de otra transición, ya que esa  
interpretación produciría un  
código de estado inválido.



**Figura 3.23** Máquina de Turing y sus datos en forma codificada

La figura 3.24 describe tal máquina como una composición de los bloques de construcción de máquinas, descritos antes. Los exponentes se emplean para indicar la cinta destino (por ejemplo,  $R^1$  mueve una celda hacia la derecha la cabeza de la cinta 1, mientras que  $R^2$  mueve la cabeza de la cinta 2). Las principales funciones de la máquina compuesta son: primero encuentra el inicio de la cadena de entrada codificada, copia esta cadena a la cinta 2 y coloca en la cinta 3 el código del estado inicial. Luego, prosigue con la búsqueda de las transiciones codificadas en la cinta hasta encontrar una que sea aplicable. Una vez que encuentra esta transición, la simula en la cinta 2 y actualiza el código de estado de la cinta 3 para reflejar el nuevo estado. Si el estado simulado se convierte en el estado de parada, borra la cinta 1, copia el contenido de la cinta 2 en la cinta 1, coloca la cabeza de la cinta 1 en el lugar donde se encontraba la cabeza de la cinta 2 al llegar al estado de parada, y se detiene.

Esta presentación de una máquina de Turing universal se ha hecho en el contexto de una máquina de tres cintas, pero sabemos que es posible simular cualquier máquina de Turing de tres cintas con una máquina de Turing que sólo tenga una. Consideramos entonces que una máquina de Turing universal es una máquina de una cinta denotada por  $T_u$ .

### Comparación entre lenguajes aceptables y decidibles

Con el apoyo de una máquina de Turing universal, ahora podemos construir una máquina de Turing que acepte el complemento del lenguaje  $L_0$ . Comenzamos por construir una máquina de Turing  $M_{pre}$  que procese una cadena de entrada  $w$  de  $\Sigma^*$  de la siguiente manera:

1. Genera una cadena de ceros y unos que represente a la máquina  $M_w$  (se trata de un proceso bastante directo, ya que la representación deseada para  $M_w$  es el código de la máquina presentada en la figura 3.21 o simplemente la representación binaria de la longitud de  $w$ ).
2. Coloca la cadena obtenida en la cinta de la máquina, seguida por la versión codificada de  $w$ .

Por lo tanto, se puede construir una máquina que acepta el complemento de  $L_0$  formando la máquina compuesta  $\rightarrow M_{pre} \rightarrow T_u$ . De hecho, dada una cadena de entrada  $w$ , esta máquina de Turing aplicaría  $M_w$  a  $w$  y se detendría si y sólo si se encontrara que  $M_w$  acepta  $w$  (véase Fig. 3.25). Así, acepta precisamente el lenguaje  $L_1 = \{w : M_w \text{ acepta } w\}$ , el cual es el complemento de  $L_0$ .

Hemos confirmado entonces que no es obligatorio que el complemento de un lenguaje aceptados por máquinas de Turing sea también aceptados por máquinas de Turing:  $L_1$  lo es, no así su complemento  $L_0$ . Esta falta de simetría no es un simple hecho curioso, sino que tiene repercusiones considerables. En nuestro caso, quiere decir que la capacidad para aceptar un lenguaje no es

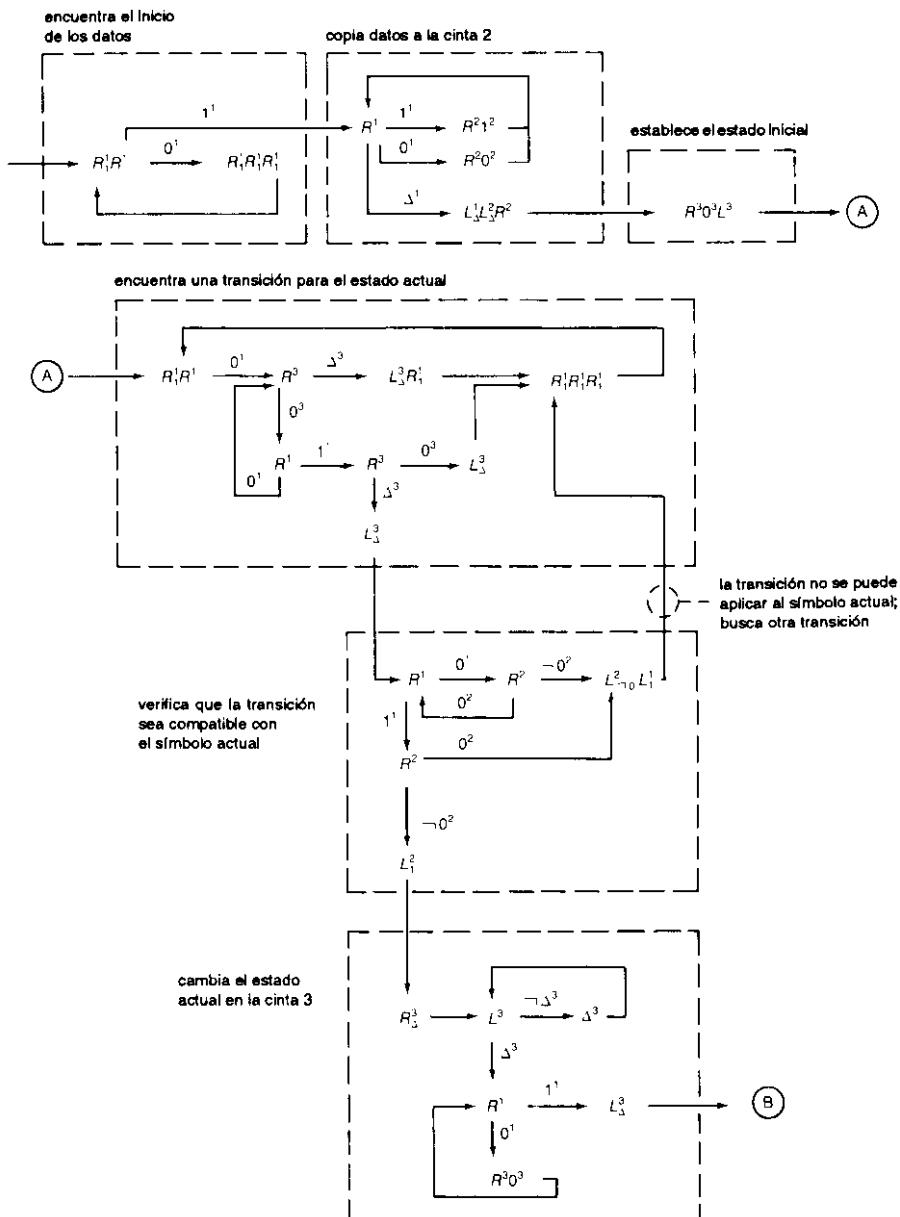


Figura 3.24 Máquina de Turing universal (continúa en la página siguiente)

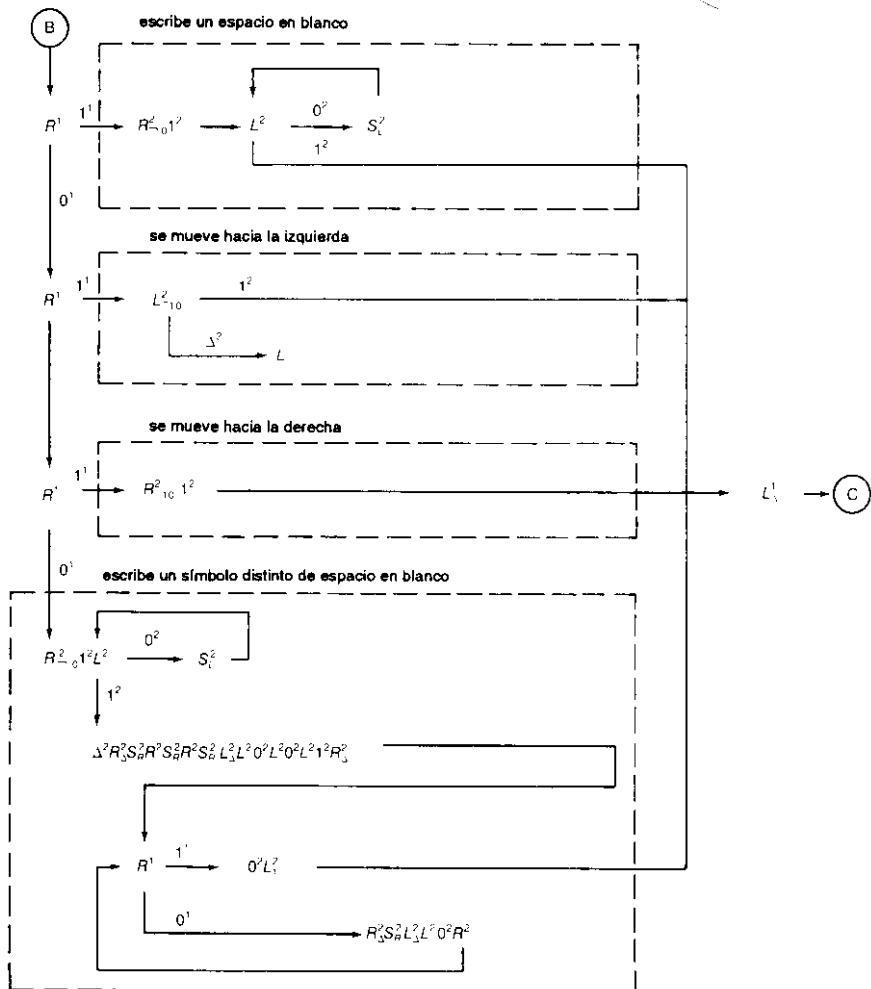


Figura 3.24 (continuación)

simétrica con la capacidad para rechazar su complemento. Existe un lenguaje  $L$  para el cual podemos construir una máquina de Turing que responda afirmativamente al recibir entradas de  $L$ , pero no se puede construir una máquina de Turing que responda afirmativamente para todas las cadenas que están en  $L$  y negativamente en el caso de todas las demás. Se dice que aquellos lenguajes donde sí se cumple esta condición son lenguajes **decidibles por máquinas de Turing** (o sólo decidibles), ya que una máquina de Turing puede decidir si una cadena se encuentra o no en el lenguaje, en vez de sólo aceptar las cadenas que pertenecen al mismo (empleando términos menos formales, decimos que la máquina de Turing decide el lenguaje en vez de únicamente aceptarlo). En resumen, el lenguaje  $L_i$  que antes definimos es aceptable, pero no decidable.

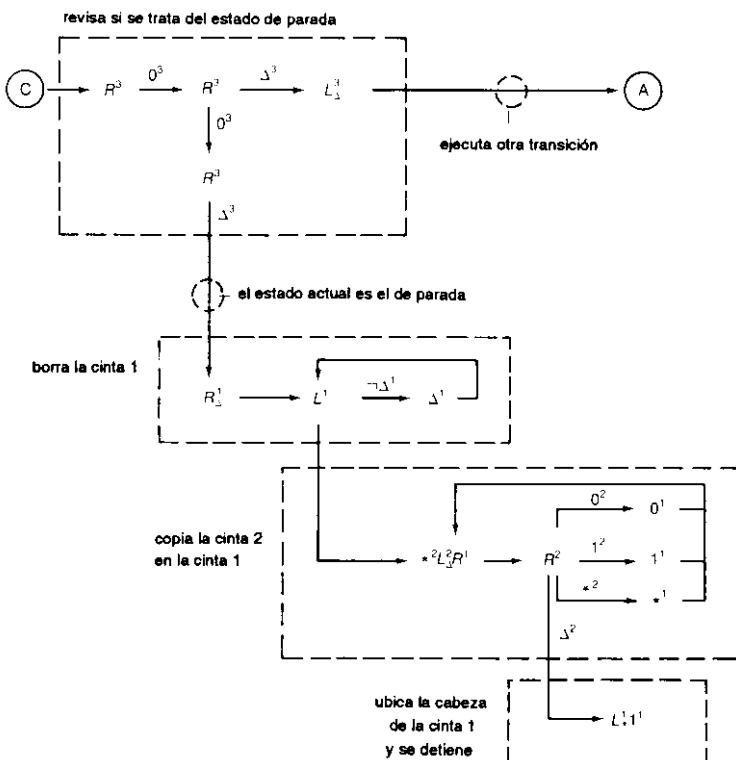
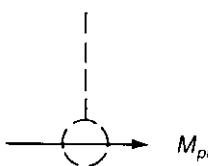


Figura 3.24 (continuación)

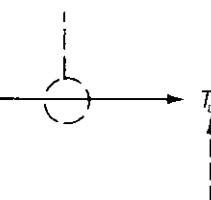
Para obtener ejemplos de lenguajes decidibles sólo tenemos que volver a los lenguajes independientes del contexto, todos los cuales son decidibles. Para sustentar esta afirmación, bosquejemos una técnica general para construir una máquina de Turing que decida un lenguaje independiente del contexto dado. Suponemos que  $L$  es un lenguaje independiente del contexto y que  $G$  es una gramática independiente del contexto, con símbolo inicial  $S$ , que genera  $L - \{\lambda\}$  y que se halla en forma normal de Chomsky. Entonces, dada una cadena  $w$ , podemos decidir si  $w \in L$  ejecutando el proceso siguiente.

Si  $w = \lambda$ , responda sí o no dependiendo de si  $\lambda \in L$ . Si  $w \neq \lambda$ , registre los no terminales de  $G$  como encabezamientos de las columnas de una tabla y llene la porción de la tabla bajo estos encabezamientos como se describe a continuación: bajo cada no terminal, registre todas las cadenas de terminales de longitud uno que pueden derivarse del no terminal; esto no es más que la colección de terminales que aparecen en el lado derecho de una regla de reescritura cuyo lado izquierdo es el no terminal en cuestión (si las reglas  $N \rightarrow xyN \rightarrow y$  estén en  $G$ , entonces registre las cadenas  $xy$  y  $y$  bajo el no terminal  $N$ ). Para construir la segunda fila de la tabla, registre bajo cada no terminal las cadenas de terminales de longitud dos que pueden derivarse de dicho no terminal (si  $N \rightarrow PQ$  es una regla de  $G$ , registre en la columna  $N$  las cadenas

Al llegar a este punto,  
la cinta de la máquina  
contiene la cadena de  
entrada  $w$ .



Al llegar a este punto, la cinta de la máquina contiene una cadena de ceros y unos que consiste en la versión codificada de  $M_w$  seguida por la versión codificada de  $w$ .



Por lo tanto, la máquina compuesta se detendrá si y sólo si  $T_u$  encuentra que  $M_w$  acepta  $w$ .

Figura 3.25 Aplicación de la máquina  $\rightarrow M_{pre} \rightarrow T_u$  a la cadena  $w$

obtenidas por la concatenación de todas las cadenas de terminales de longitud uno que se pueden derivar de  $P$  con todas las cadenas de terminales de longitud uno que se pueden derivar de  $Q$ . Observe que estas cadenas de un solo terminal están registradas en las filas anteriores de la tabla). En la tercera fila de la tabla, bajo cada no terminal, registre todas las cadenas de terminales de longitud tres que pueden derivarse de dicho no terminal (si  $N \rightarrow PQ$  es una regla de  $G$ , concatene todas las cadenas de terminales de longitud uno que pueden derivarse de  $P$  con cada una de las cadenas de terminales de longitud uno que se pueden derivar de  $Q$ . Luego, concatene todas las cadenas de terminales de longitud dos que pueden derivarse de  $P$  con las de longitud uno que se pueden derivar de  $Q$ . Una vez más, todas las cadenas que se deben concatenar se encuentran en las casillas de la tabla correspondientes a las filas por encima de la que estamos llenando.)

En general, registramos en la fila  $n$  bajo cada no terminal  $N$  todas las cadenas de terminales de longitud  $n$  que se pueden derivar de dicho no terminal. Estas cadenas se pueden encontrar localizando primero cada regla de la forma  $N \rightarrow PQ$  y luego, para cada  $i$  en  $\{1, 2, \dots, n-1\}$ , concatenando cada cadena de terminales de longitud  $i$  derivable de  $P$  con cada cadena de terminales de longitud  $n-i$  derivable de  $Q$  (véase Fig. 3.26). Continuamos con la construcción de esta tabla hasta completar la fila número  $|w|$ . Al llegar a este punto verificamos si  $w$  se ha registrado bajo el símbolo inicial de  $G$ . Si es así, entonces  $w \in L$ ; de lo contrario,  $w \notin L$ . Por lo tanto, concluye el proceso de decisión.

Para realizar este procedimiento con una máquina de Turing, podemos usar una máquina con cintas múltiples que tenga una cinta más que el número de no terminales en  $G$ . La primera cinta se puede usar para almacenar la cadena de entrada y la respuesta afirmativa o negativa. Cada una de las cintas restantes se puede emplear para representar una columna de la tabla, separando con diagonales los elementos de distintas filas. Así, el proceso de decidir si

la cadena  $xyx$  se encuentra o no en el lenguaje generado por la gramática de la figura 3.26 produciría el contenido de las cintas que está representado en la figura 3.27, de donde la máquina puede determinar que  $xyx$  no se encuentra en el lenguaje.

Por supuesto, existen también lenguajes decidibles que no son independientes del contexto. Un ejemplo es el lenguaje  $\{x^n y^n z^n : n \in \mathbb{N}\}$ , que no es independiente del contexto pero sí puede ser decidido por la máquina de Turing de la figura 3.28.

Por último, debemos señalar que la terminología “decidible por máquinas de Turing” y “aceptado por máquinas de Turing” no es universal. Ya hemos visto que un lenguaje aceptado por máquinas de Turing es idéntico a un lenguaje estructurado por frases. Sin embargo, en otros contextos un lenguaje aceptado por máquinas de Turing se conoce como **lenguaje enumerable recursivamente**. Esta terminología se debe a que los lenguajes aceptados por máquinas de Turing son precisamente los lenguajes cuyas cadenas puede enumerar –o, en otras palabras, listar– una máquina de Turing. Además, en

$$\begin{aligned} S &- MN \\ M &- MP \\ N &- PN \\ M &- x \\ N &- y \\ P &- x \\ P &- y \end{aligned}$$

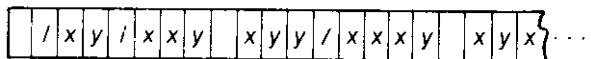
	<i>S</i>	<i>M</i>	<i>N</i>	<i>P</i>
1		<i>x</i>	<i>y</i>	<i>x</i> <i>y</i>
2	<i>xy</i>	<i>xx</i> <i>xy</i>	<i>xy</i> <i>yy</i>	
3	<i>xx</i> <i>xy</i>	<i>xxx</i> <i>xx</i> <i>xy</i> <i>yx</i> <i>yy</i>	<i>xx</i> <i>xy</i> <i>yy</i> <i>yy</i>	
4	<i>xxxx</i> <i>xyxy</i> <i>xxyy</i> <i>yyyy</i>	<i>xxxx</i> <i>xx</i> <i>xy</i> <i>yx</i> <i>xx</i> <i>xy</i> <i>yy</i> <i>yx</i> <i>yy</i>	<i>xxxx</i> <i>xyxy</i> <i>xxyy</i> <i>yyyy</i> <i>yxxx</i> <i>yyxy</i> <i>yxyy</i> <i>yyyy</i>	

Figura 3.26 Gramática independiente del contexto y primeras cuatro filas de la tabla construida a partir de ella

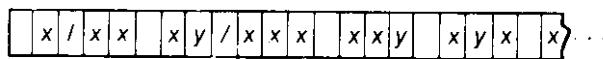
cinta uno, que contiene la cadena de entrada



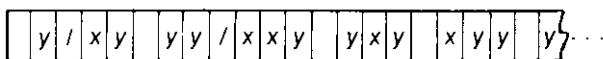
cinta dos, que representa la columna S



cinta tres, que representa la columna M



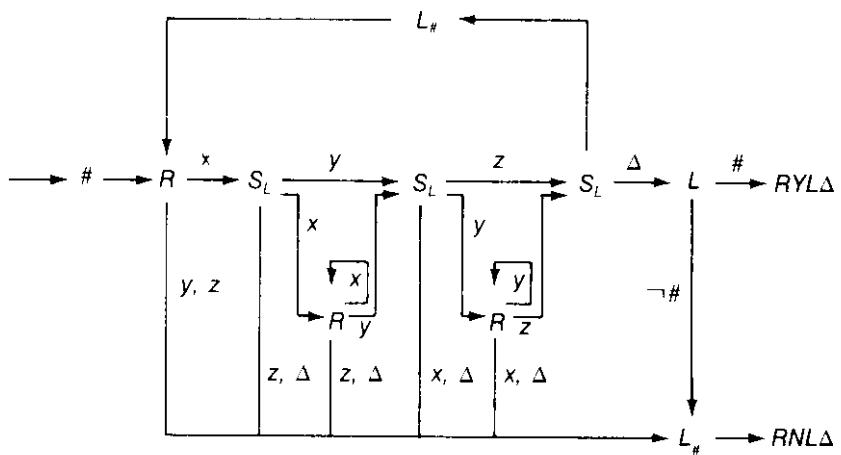
cinta cuatro, que representa la N



cinta cinco, que representa la columna P



**Figura 3.27** Realización con cinco cintas del proceso de construcción de la tabla de la figura 3.26



**Figura 3.28** Máquina de Turing que decide el lenguaje  $\{x^n y^n z^n : n \in \mathbb{N}\}$

en aquellas situaciones en las cuales se habla de lenguajes enumerables recursivamente, por lo general se hace referencia a los lenguajes decidibles por máquinas de Turing como lenguajes recursivos.

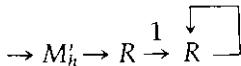
## El problema de la parada

Damos fin a esta sección con un ejemplo de otro lenguaje que no es decidable por una máquina de Turing, no sólo porque precisemos otra muestra sino porque el caso que se presentará, conocido como problema de la parada, es un ejemplo clásico en la teoría de la computación.

Recuerde que cualquier máquina de Turing se puede codificar como una cadena de ceros y unos. Representemos con  $\rho(M)$  esta versión codificada de una máquina de Turing  $M$ . Si ahora centramos nuestra atención en las máquinas con alfabeto  $\{0, 1\}$  y símbolos de cinta  $\{0, 1, \Delta\}$ , entonces la cadena de código  $\rho(M)$  se podría usar como entrada para la misma máquina  $M$ . No nos interesa si esta forma de usar  $M$  tiene relación con el fin para el cual se diseñó originalmente  $M$ . Lo único que nos importa es si  $M$  se detiene o no cuando inicia su operación con  $\rho(M)$  como entrada. Si se detiene, decimos que  $M$  es autotérminante. De esta manera, cualquier máquina de Turing con alfabeto  $\{0, 1\}$  y símbolos de cinta  $\{0, 1, \Delta\}$  es o bien autotérminante o bien no autotérminante.

Definamos ahora el lenguaje  $L_h$  de  $\{0, 1\}^*$  como  $\{\rho(M) : M \text{ es autotérminante}\}$  y preguntémonos si  $L_h$  es decidable por máquinas de Turing. Para efectuar una decisión con respecto a  $L_h$ , se requiere la capacidad para detectar si una cadena de  $\{0, 1\}^*$  es la versión codificada de una máquina autotérminante, lo que en esencia es el problema de decidir si una máquina se para cuando se le aplica una entrada específica; por consiguiente, el problema de decisión de  $L_h$  se conoce como **problema de la parada**.

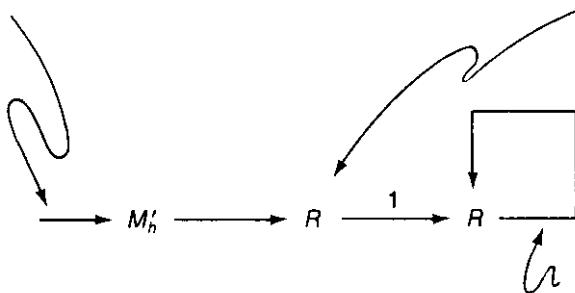
Por desgracia, el lenguaje  $L_h$  no es decidable según Turing. Para convencernos de esto, supongamos lo contrario, que existe una máquina de Turing  $M_h$  que decide  $L_h$ . Entonces, modificaríamos  $M_h$  para obtener otra máquina  $M'_h$  que responda con el mensaje 1 en caso de que  $M_h$  hubiera respondido  $Y$  y 0 cuando la respuesta de  $M_h$  fuera  $N$ . Como en la demostración del teorema 3.6, podemos establecer que los símbolos de cinta que necesita  $M'_h$  consisten sólo en  $\{0, 1, \Delta\}$ . Así, se podría emplear  $M'_h$  para construir la máquina compuesta



con símbolos de cinta  $\{0, 1, \Delta\}$ , que se detiene si y sólo si  $M'_h$  llega a su estado de parada con salida 0. Representemos con  $M_0$  a esta máquina compuesta.

A partir de aquí, nuestro argumento gira alrededor de la pregunta ¿es  $M_0$  autotérminante o no autotérminante? Puesto que se trata de una máquina de Turing con símbolos de cinta  $\{0, 1, \Delta\}$ , debe ser una cosa o la otra. Supongamos que es autotérminante. Entonces, al recibir la entrada  $\rho(M_0)$ ,  $M'_h$  se detendrá con salida 1. Esto quiere decir que  $M_0$  no se detendría si iniciara con la entrada  $\rho(M_0)$ . (La ejecución de  $M_0$  recorrería el arco  $R \xrightarrow{1} R$  y quedaría atrapada en el proceso infinito de movimiento de la cabeza a la derecha. Véase Fig. 3.29.) Sin embargo, ésta es la característica que define a una máquina que no es autotérminante. Entonces, si suponemos que  $M_0$  es autotérminante, llegamos a la contradicción de que no es autotérminante.

1. Si  $M_0$  fuera autotérminante y se pusiera en marcha con la entrada  $\rho(M_0)$ ,

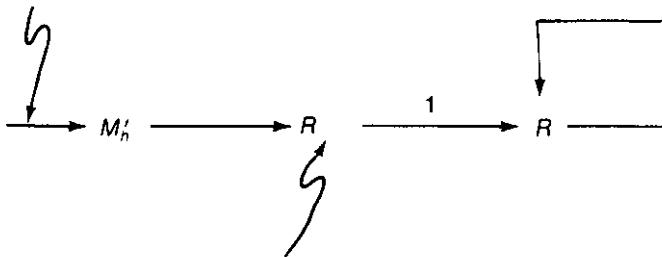


2. su ejecución llegaría a este punto con el símbolo actual 1,

3. por lo que  $M_0$  quedaría atrapada en este ciclo infinito y nunca se detendría.

**Figura 3.29** Ejecución de la máquina  $M_0$  con entrada  $\rho(M_0)$  bajo el supuesto de que  $M_0$  es autotérminante

1. Si  $M_0$  no fuera autotérminante y se pusiera en marcha con la entrada  $\rho(M_0)$



2. su ejecución llegaría a este punto con el símbolo actual 0, por lo que  $M_0$  se detendría.

**Figura 3.30** Ejecución de la máquina  $M_0$  con entrada  $\rho(M_0)$  bajo el supuesto de que no es autotérminante

Puesto que  $M_0$  debe ser autotérminante o no autotérminante, al llegar a este punto debemos concluir que no es autotérminante. Empero, si  $M_0$  no fuera autotérminante, entonces, al recibir la entrada  $\rho(M_0)$ ,  $M'_h$  se detendría con salida 0. Esto quiere decir que  $M_0$  se detendría si iniciara con la entrada  $\rho(M_0)$ .

(La ejecución de  $M_0$  no podría recorrer el arco  $R \xrightarrow{1} R$ . Véase Fig. 3.30.) Sin embargo, ésta es la característica que define a una máquina autotérminante. Por lo tanto, también debe ser falsa nuestra suposición de que  $M_0$  no es autotérminante.

Hemos llegado a la inconsistencia de una máquina  $M_0$  con símbolos de cinta  $\{0, 1, \Delta\}$  que no es autotérminante ni no autotérminante, cuando la máquina debe ser una u otra cosa. Por consiguiente, nos vemos obligados a concluir que nuestra suposición inicial es falsa; es decir, el lenguaje  $L_h$  no es decidable por máquinas de Turing.

## Ejercicios

1. Diseñe una máquina de Turing que acepte exactamente aquellas cadenas de ceros y unos que sean representaciones válidas de máquinas de Turing deterministas, de acuerdo con el sistema de codificación descrito en esta sección.
2. Identifique la paradoja que existe en el siguiente enunciado:

El cocinero de una comunidad aislada cocina para aquellas personas, y sólo aquellas personas, que no cocinan para sí mismas.

(Sugerencia: ¿Quién cocina para el cocinero?) ¿Cómo se relaciona esta pregunta con esta sección?

3. Diseñe una máquina de Turing que acepte las cadenas del lenguaje  $L = \{w^n : w = xy\}$  al detenerse con su cinta configurada como  $\underline{\Delta}Y\Delta\Delta\Delta\dots$  y rechace las cadenas que no se hallan en  $L$  al detenerse con la configuración  $\underline{\Delta}N\Delta\Delta\Delta\dots$ . Explique por qué no puede construirse esta máquina en el caso de todos los lenguajes aceptados por máquinas de Turing.
4. Presente un argumento de que cualquier lenguaje que contenga un número finito de cadenas es decible por máquinas de Turing.
5. Amplíe la tabla de la figura 3.26 para solucionar la pregunta de si la cadena  $xyyyxy$  es generada por la gramática.

## 3.6 COMENTARIOS FINALES

En este capítulo y los anteriores presentamos una jerarquía de lenguajes junto con los autómatas que se requieren para aceptarlos. Se trata de una variante de la jerarquía de lenguajes conocida como jerarquía de Chomsky (nombrada así

en honor de N. Chomsky, pionero del desarrollo de la teoría de lenguajes formales durante la década de los cincuenta). En la figura 3.31 se presenta un resumen de la jerarquía que hemos presentado. Mostramos así mismo que cada nivel de esta jerarquía contiene propiamente el siguiente nivel inferior. De hecho, en cada nivel proporcionamos un ejemplo explícito de un lenguaje que reside en ese nivel pero no en el inferior. En este momento, un excelente ejercicio de repaso sería recopilar estos ejemplos e incluirlos en el diagrama de la figura 3.31.

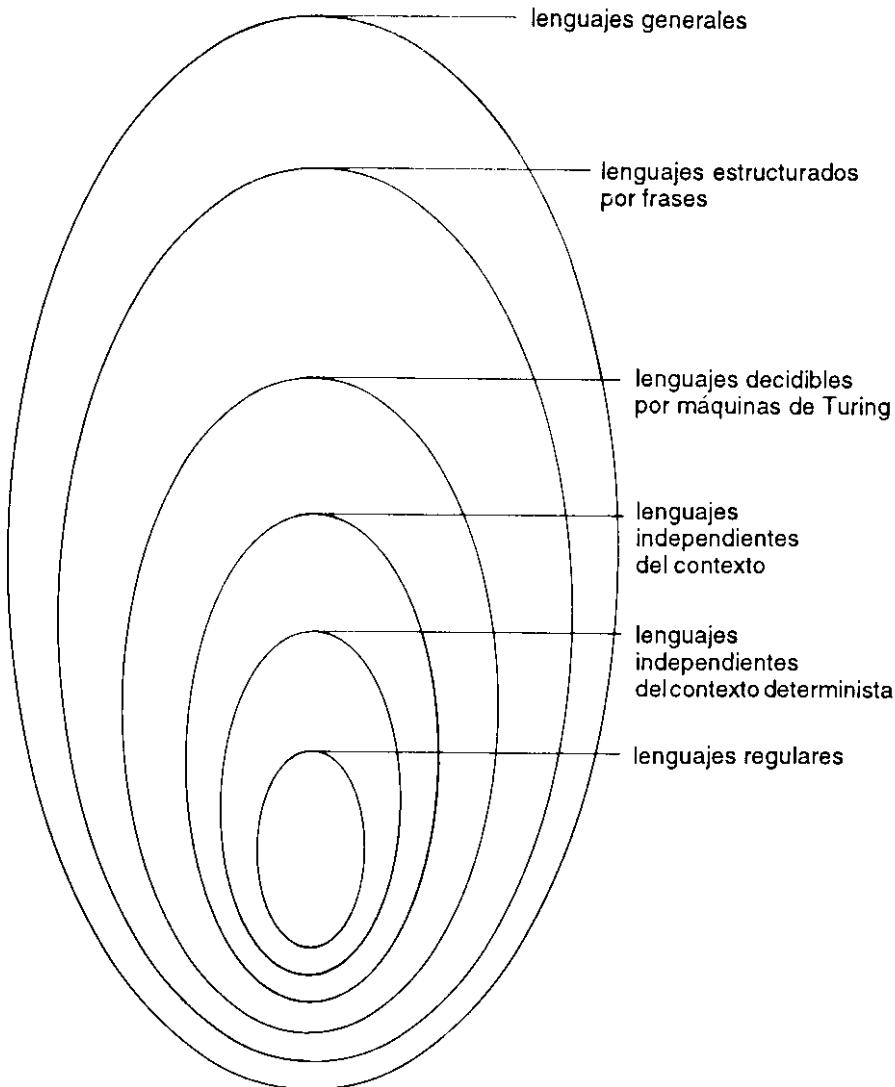
Uno de los principales resultados de este capítulo ha sido la presentación de la tesis de Turing, la cual, traducida al contexto de la figura 3.31, nos dice que un sistema computacional nunca podrá efectuar un análisis sintáctico de aquellos lenguajes que se encuentran más allá de los lenguajes estructurados por frases. Como se señaló al final de la sección 3.4, esta tesis tiene grandes repercusiones. Por esto, no es ninguna sorpresa que la tesis de Turing haya llamado la atención de muchos investigadores, quienes han tratado de apoyar sus afirmaciones o de desafiar su validez.

En el próximo capítulo veremos algunos de los resultados de las actividades de estos investigadores. Por ahora sólo señalaremos que la amplia gama de terminología que hemos empleado (como “lenguajes aceptable por máquinas de Turing”, “lenguajes estructurados por frases” y “lenguajes recursivamente enumerables”) es el resultado del alcance de la tesis de Turing. En varias disciplinas se han observado y estudiado los mismos límites aparentes del poder de los procesos computacionales que surgen de la hipótesis de Turing, incluso cuando cada quien ha atacado el problema de la computabilidad desde una perspectiva distinta y con diferente terminología.

## Problemas de repaso del capítulo

---

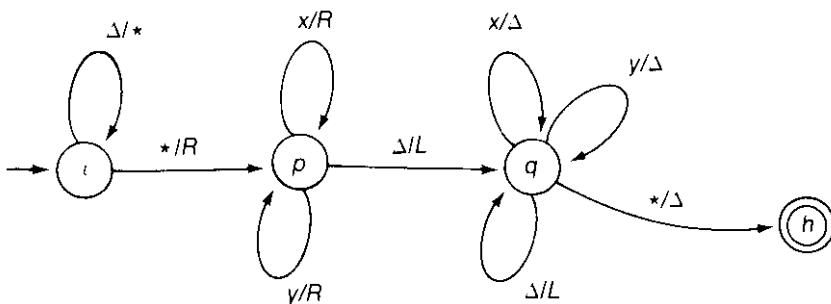
1. Con los bloques de construcción presentados en la sección 3.2, construya una máquina de Turing compuesta que convierta su cinta de la configuración  $\underline{\Delta}w\Delta\Delta\Delta\cdots$ , donde  $w$  es cualquier cadena de  $x$  y  $y$ , en la configuración  $\underline{\Delta}w\Delta v\Delta\Delta\Delta\cdots$ , donde  $v$  es la cadena  $w$  escrita a la inversa.
2. Con los bloques de construcción presentados en la sección 3.2, construya una máquina de Turing compuesta que forme la concatenación de dos cadenas  $v$  y  $w$  en  $\{x, y\}^*$  convirtiendo su cinta de la configuración  $\underline{\Delta}v\Delta w\Delta\Delta\Delta\cdots$  en  $\underline{\Delta}vw\Delta\Delta\Delta\cdots$ .
3. ¿Cómo puede el resultado de la ejecución de  $\rightarrow RL$  diferir del de  $\rightarrow LR$ ?
4. Dibuje un diagrama de transiciones para la máquina de Turing compuesta  $\rightarrow R_x\Delta L$ .



**Figura 3.31** Variante de la jerarquía de lenguajes de Chomsky

5. Diseñe una máquina de Turing  $M$  tal que  $L(M) = \{x^n y^{2n} z^n : n \in \mathbb{N}\}$ .
6. Diseñe una gramática  $G$  estructurada por frases tal que  $L(G) = \{x^m y^n x^m y^n : m, n \in \mathbb{N}\}$ .
7. Diseñe una gramática  $G$  estructurada por frases tal que  $L(G) = \{x^n y^{2n} z^n : n \in \mathbb{N}\} \cup \{x^n y^n z^n : n \in \mathbb{N}\}$ .

8. Utilizando la notación para representar toda la configuración de una máquina de Turing, como la que se usó en la demostración del teorema 3.3, rastree la ejecución de la máquina compuesta de la figura 3.3 al iniciar con la configuración de cinta  $\underline{\Delta}\Delta yx\Delta\Delta\Delta\dots$ .
9. Utilizando la notación para representar toda la configuración de una máquina de Turing, como la que se usó en la demostración del teorema 3.3, rastree la ejecución de la máquina cuyo diagrama de transiciones se presenta a continuación, suponiendo que la configuración inicial de la cinta es  $\underline{\Delta}xyx\Delta\Delta\Delta\dots$ .



10. ¿Es independiente del contexto el lenguaje generado por la gramática siguiente (con símbolo inicial  $S$ )? Describa el lenguaje.

$$\begin{aligned} S &\rightarrow xN \\ N &\rightarrow Sx \\ xNx &\rightarrow y \end{aligned}$$

11. ¿Es independiente del contexto el lenguaje generado por la gramática siguiente (con símbolo inicial  $S$ )? Describa el lenguaje.

$$\begin{aligned} S &\rightarrow SPQR \\ S &\rightarrow \lambda \\ QP &\rightarrow PQ \\ PQ &\rightarrow QP \\ PR &\rightarrow RP \\ RP &\rightarrow PR \\ QR &\rightarrow RQ \\ RQ &\rightarrow QR \\ P &\rightarrow x \\ Q &\rightarrow y \\ R &\rightarrow z \end{aligned}$$

12. Diseñe una máquina de Turing  $M$  tal que  $L(M) = \{x, y, z\}^* - \{x^n y^n z^n : n \in \mathbb{N}\}$ .

13. Muestre que si el lenguaje  $L$  es aceptado por máquinas de Turing, entonces existe una máquina de Turing que termina anormalmente si y sólo si su cadena de entrada se encuentra en  $L$ .
14. Si el lenguaje  $L$  de  $\Sigma$  es aceptable según Turing, ¿existe alguna máquina de Turing que se detenga si su entrada está en  $L$  y termine anormalmente si su entrada en  $\Sigma^* - L$ ? Explique su respuesta.
15.
  - a. ¿Forma la unión de un número finito de lenguajes estructurados por frases un lenguaje estructurado por frases? Justifique su respuesta.
  - b. ¿Forma siempre la unión de una colección de lenguajes estructurados por frases un lenguaje estructurado por frases? Justifique su respuesta.
16.
  - a. ¿Forma la intersección de un número finito de lenguajes estructurados por frases un lenguaje estructurado por frases? Justifique su respuesta.
  - b. ¿Forma siempre la intersección de una colección de lenguajes estructurados por frases un lenguaje estructurado por frases? Justifique su respuesta.
17. Podemos expresar una gramática como una cadena de símbolos en donde las reglas de reescritura se separan con diagonales ( $S \rightarrow xS / S \rightarrow \lambda$ ). Diseñe una máquina de Turing que acepte únicamente las cadenas que representen gramáticas independientes del contexto con terminales del conjunto  $\{x, y\}$  y no terminales de  $\{S, M, N\}$ .
18. Diseñe una gramática  $G$  que no sea independiente del contexto con la cual  $L(G)$  sea el lenguaje independiente del contexto  $\{x^n y^n : n \in \mathbb{N}\}$ .
19. Muestre que para cada lenguaje  $L$  estructurado por frases existe una cantidad infinita de gramáticas que generan  $L$ .
20. Diseñe una máquina de Turing (quizás con varias cintas) que ordene alfabéticamente una lista de cadenas de  $\{x, y, Z\}^*$ . Suponga que las cadenas de la lista de entrada están separadas por un espacio en blanco.
21. Muestre que la colección de lenguajes no estructurados por frases tiene mayor cardinalidad que la colección de los lenguajes estructurados por frases (por lo tanto, existen más lenguajes no aceptados por máquinas de Turing que lenguajes que sí lo son).
22. Muestre que para cualquier alfabeto  $\Sigma$  existe un lenguaje  $L$  de  $\Sigma$  tal que ni  $L$  ni  $\Sigma^* - L$  son aceptados por máquinas de Turing.

23. Muestre que la colección de lenguajes decidibles por máquinas de Turing para un alfabeto cualquiera es infinita, pero contable.
24. Muestre que si una máquina de Turing  $M$  acepta cada cadena  $w$  en  $L(M)$  sin tener que ejecutar más de  $|w| + 1$  pasos, entonces  $L(M)$  debe ser regular.
25. ¿Es un subconjunto de un lenguaje estructurado por frases siempre un lenguaje estructurado por frases? Explique su respuesta.
26. Muestre que la estrella de Kleene de un lenguaje aceptado por máquinas de Turing es también aceptable según Turing.
27. Muestre que el lenguaje  $\{x^n : n \text{ es un entero primo positivo}\}$  es aceptado por máquinas de Turing. ¿Es decidable? ¿Por qué?
28. Muestre que el lenguaje  $\{x^{n^2} : n \in \mathbb{N}\}$  es aceptado por máquinas de Turing. ¿Es decidable? ¿Por qué?
29. Dibuje un diagrama de transiciones para una máquina de Turing que decida el lenguaje  $\{x^m y^n : m, n \in \mathbb{N} \text{ y } m \geq n\}$ .
30. Construya, con los bloques de construcción presentados en la sección 3.2, una máquina de Turing que acepte el lenguaje  $\{x^m y^n x^m y^n : m, n \in \mathbb{N}\}$ .
31. Aplique el algoritmo de construcción de tablas descrito en la sección 3.5 para decidir si la cadena  $xxyyxxxxyy$  se encuentra en el lenguaje generado por la gramática siguiente, cuyo símbolo inicial es  $S$ .

$$\begin{aligned} S &\rightarrow SS \\ S &\rightarrow MN \\ N &\rightarrow SP \\ M &\rightarrow x \\ N &\rightarrow y \\ P &\rightarrow y \end{aligned}$$

- ¿Se encuentran también las cadenas  $xxxyyyxyy$ ,  $xyxyxyxy$  y  $xyxxxxyy$ ?
32. Amplíe la tabla de la figura 3.26 para determinar si la cadena  $xyyxyyy$  es generada por la gramática de dicha figura.
  33. Suponga que  $p(n)$  es una expresión polinomial en  $n$  y que  $M$  es una máquina de Turing que acepta toda cadena  $w$  de  $L(M)$  sin tener que ejecutar más de  $p(|w|)$  pasos. Muestre que el lenguaje  $L(M)$  es decidable.

34. Muestre que todo lenguaje aceptado por máquinas de Turing puede generarse a partir de una gramática estructurada por frases en la que ninguna regla de reescritura tiene una terminal como parte de su lado izquierdo.
35. Muestre que para cualquier lenguaje  $L$  recursivamente enumerable, existe una máquina de Turing que, al ponerse en marcha con una cinta en blanco, comenzará a generar en su cinta una lista de las cadenas en  $L$ , de manera que cada una de ellas aparecerá tarde o temprano en la cinta.
36. Muestre que todo lenguaje enumerable recursivamente infinito contiene un lenguaje recursivo infinito.

## Problemas de programación

---

1. Desarrolle un simulador de máquinas de Turing. Diseñe este simulador de manera que se proporcione en forma tabular la descripción de la máquina que se simulará, para que pueda reemplazarse fácilmente con la descripción de otra máquina.
2. Escriba un programa que simule la máquina de Turing universal de tres cintas descrita en este capítulo.
3. Escriba un programa para decidir si la cadena que se proporcione como entrada se halla o no en el lenguaje  $\{x^m y^n z^m : m, n \in \mathbb{N}\}$ , aplicando el algoritmo de construcción de tablas descrito en la sección 3.5.