

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

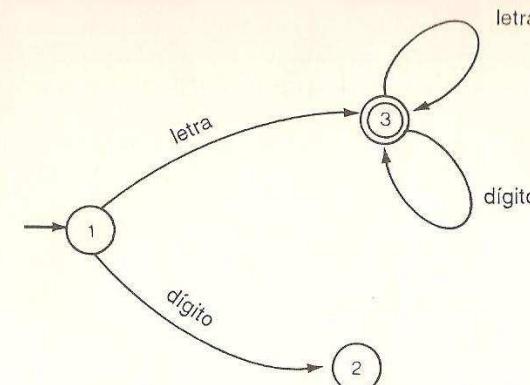


Figura 1.1 Diagrama de transiciones que representa la sintaxis de un nombre de variable

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

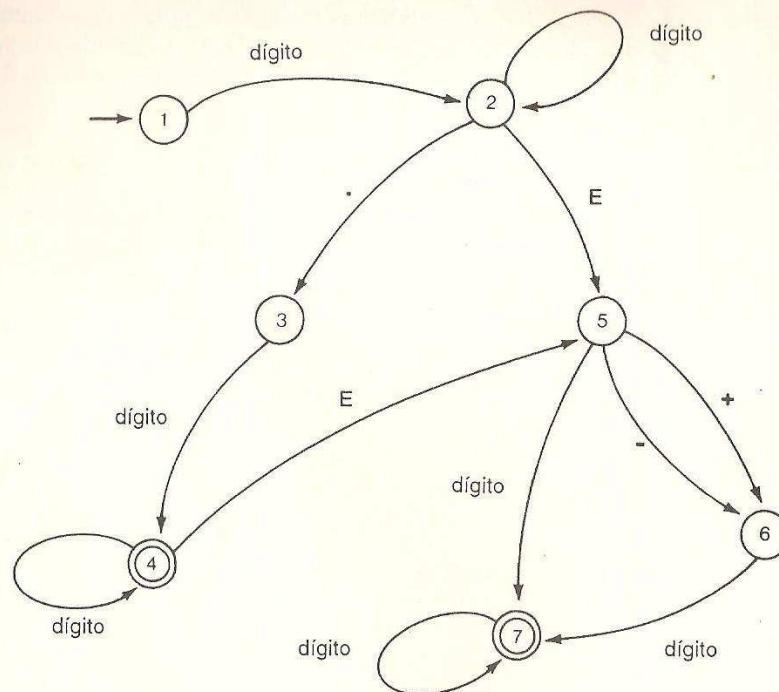


Figura 1.5 Diagrama de transiciones que representa la sintaxis de un número real

	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

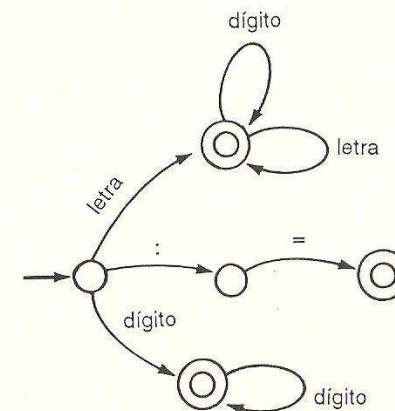
Estado: = 1;
repeat

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

- 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.
- Escriba un analizador léxico directamente a partir del siguiente diagrama de transiciones.



- Construya una tabla de transiciones a partir del diagrama del ejercicio 2 y escriba un analizador léxico basado en esa tabla.
- 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 enérgica 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 computador, 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 analiza 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 abandonar 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 permanecer en el estado 5. En análisis subsecuentes se verá la importancia en que los sistemas de reconocimiento basados en estos diagramas de transiciones no puedan recordar cómo llegaron al lugar donde se encuentran.

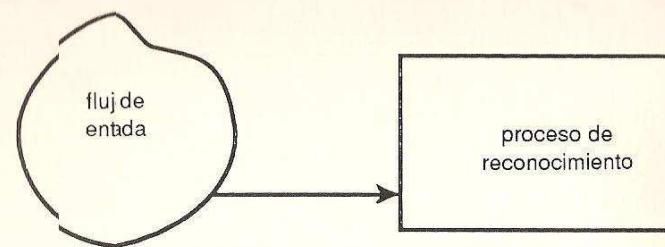


Figura 1.8 Flujo de entrada como fuente de símbolos.

Los conceptos que tenemos hasta ahora se fortalecen definiendo una máquina conceptual conocida como **autómata finito determinista**. Esta máquina consta en un dispositivo que puede estar en cualquiera de un número finito de estados, 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 mecanismo de control de la máquina, programado para reconocer 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 estados finitos.

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, porque 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 lectora 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

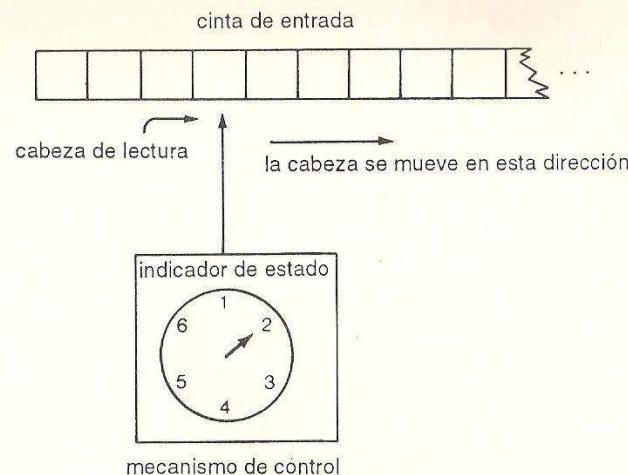


Figura 1.9 Representación de un autómata finito determinista

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 λ . Recalcamos que la cadena vacía no contiene símbolos; por lo tanto, un autómata finito determinista aceptará λ si y sólo si su estado inicial es también un estado de aceptación.

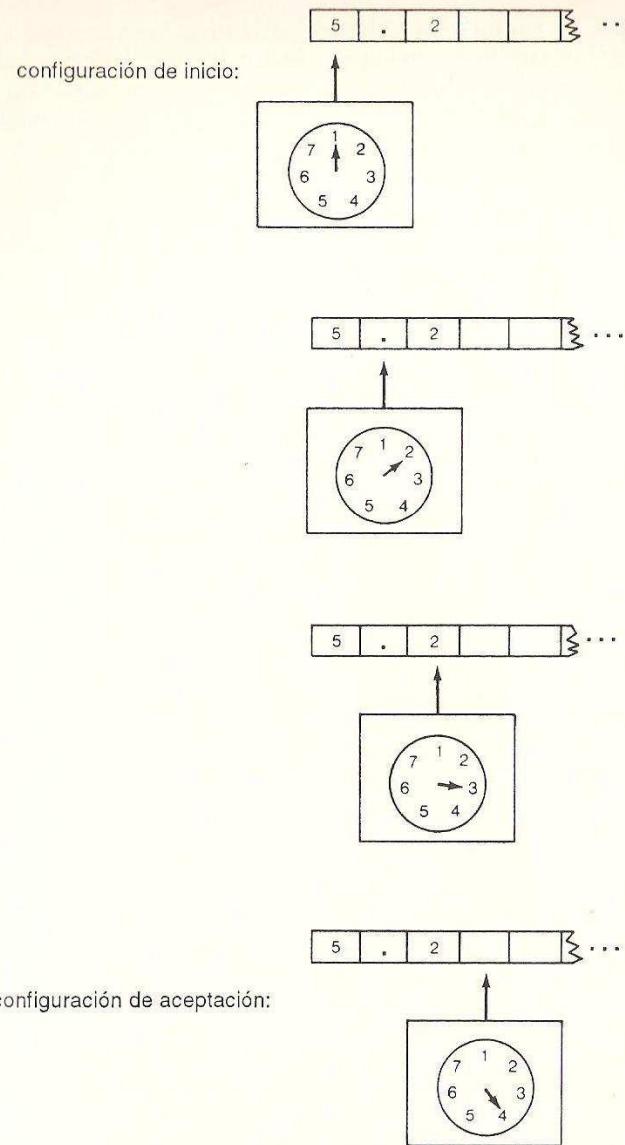


Figura 1.10 Pasos efectuados por un autómata finito determinista (programado por el diagrama de la Fig. 1.5) al procesar la cadena "5.2"

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.
- Σ es el alfabeto de la máquina.
- δ 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 δ 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 δ 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 encuentra 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.

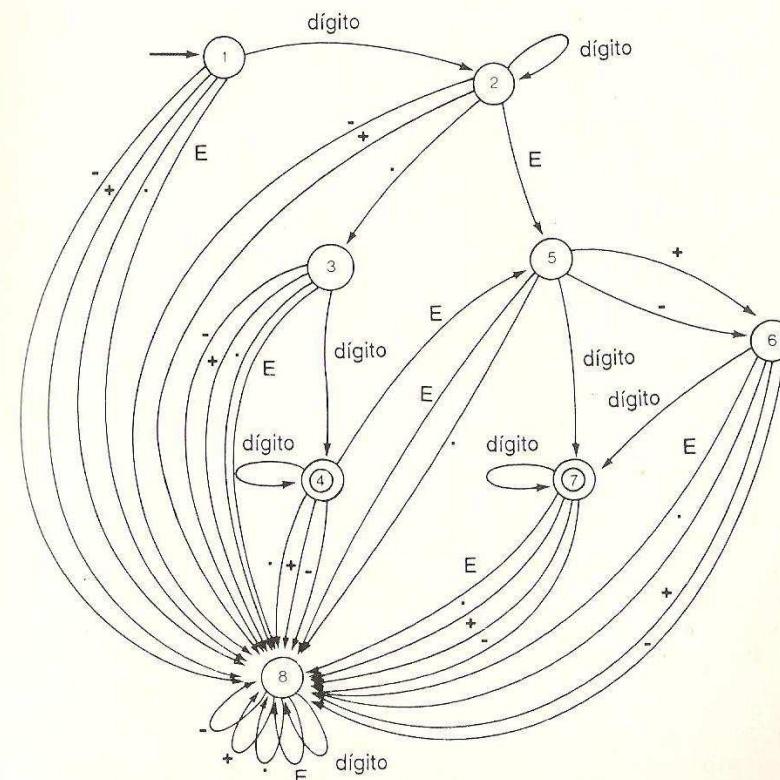


Figura 1.11 Diagrama "completo" de la figura 1.5

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 sobre pago, 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

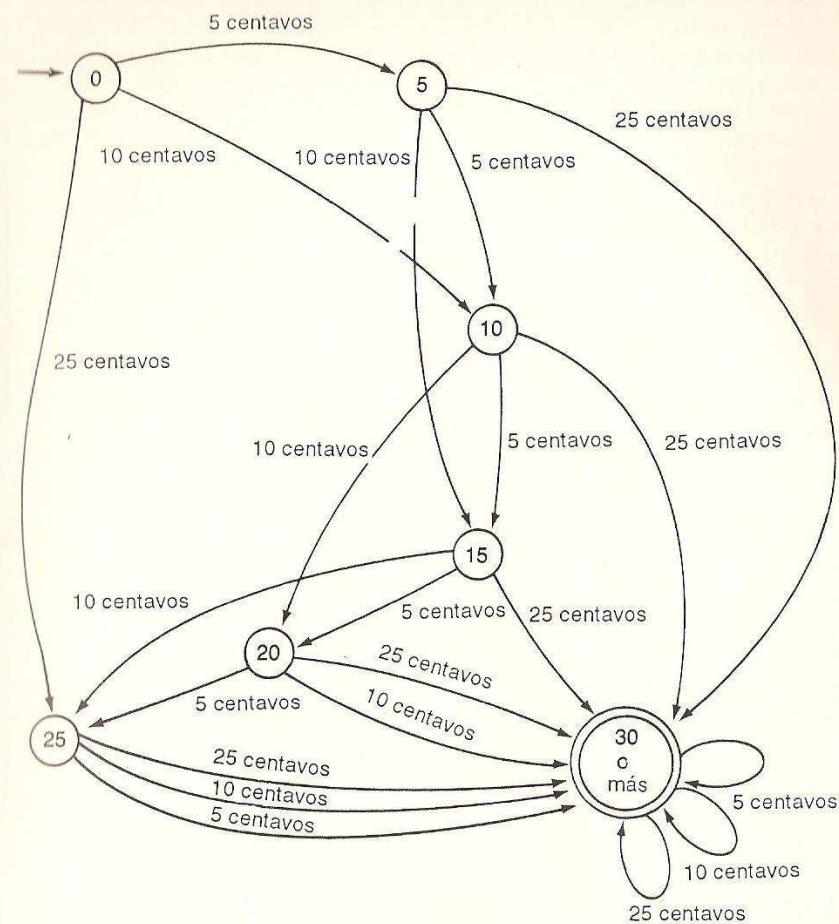
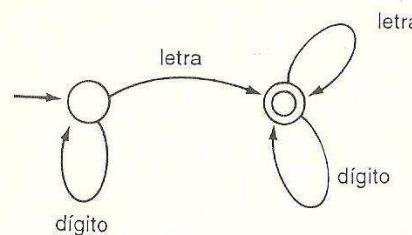


Figura 1.12 Diagrama de transiciones para una máquina vendedora

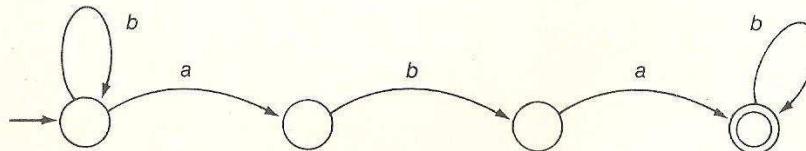
los analizadores léxicos. Sin embargo, al reconocer que estos dispositivos son autómatas finitos deterministas nos percatamos de que su acción es, en esencia, la de un análisis léxico. La máquina vendedora analiza cadenas de monedas; el teléfono analiza cadenas de dígitos. Así, si abstraemos los ingredientes 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, consideramos 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 Σ , este conjunto de cadenas está representado por Σ^* . Por lo tanto, si Σ fuera $\{a, b\}$, entonces Σ^* sería $\{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$.

Un subconjunto de Σ^* se llama lenguaje (del alfabeto Σ), de acuerdo con nuestro empleo tradicional del término. De hecho, si Σ 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 Σ^* , mientras que otro subconjunto consistiría en las frases en latín. Por supuesto, existirían también subconjuntos de Σ^* 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 Σ . 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

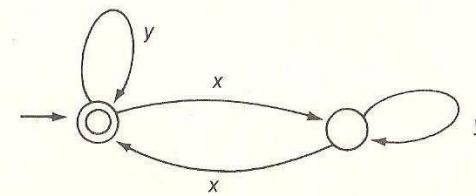


Figura 1.13 Diagrama de transiciones para un autómata finito determinista que acepta exactamente aquellas cadenas de x y y que contienen un número par de x

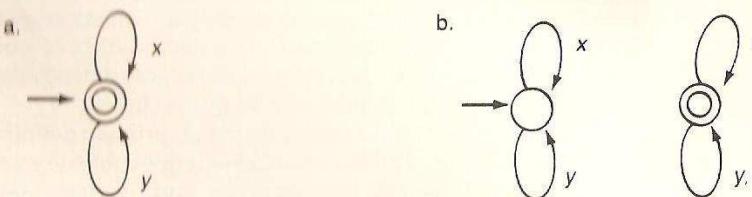


Figura 1.14 Diagramas de transiciones para autómatas finitos deterministas que aceptan a) el lenguaje $\{x, y\}^*$ y b) el lenguaje \emptyset

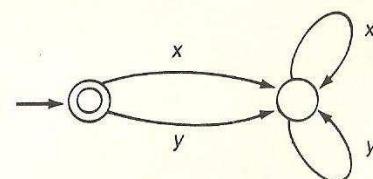


Figura 1.15 Diagrama de transiciones para un autómata finito determinista que acepta el lenguaje $\{\lambda\}$

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 Σ , que ya representamos con Σ^* , 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 estas 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 Σ , existe un lenguaje que no es igual a $L(M)$ para cualquier autómata finito determinista M .

DEMOSTRACIÓN

Puesto que cualquier arco de un autómata finito determinista que esté rotulado con un símbolo fuera de Σ no tendrá efecto alguno sobre el procesamiento de una cadena en Σ^* , podemos considerar sólo las máquinas con alfabeto Σ . Sin embargo, la colección de autómatas finitos deterministas con alfabeto Σ 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 Σ es incontable, ya que el conjunto infinito Σ^* 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 Σ^* 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.

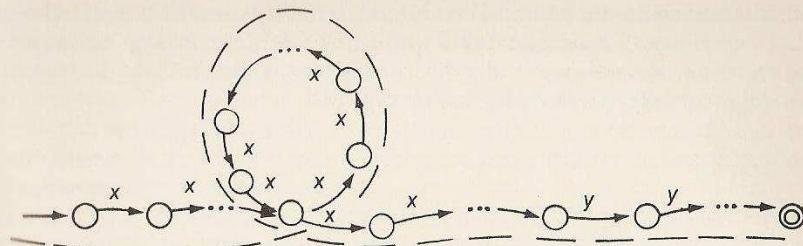
TEOREMA 1.2

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

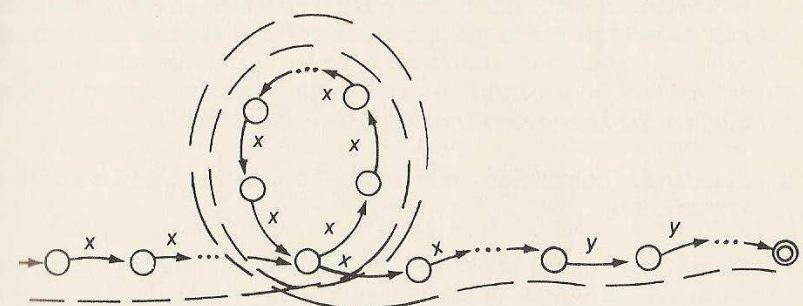
DEMOSTRACIÓN

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 $(^n)$ 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 $x^{k+j} y^k$

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

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 $M = (S, \Sigma, \delta, i, F)$ para obtener una máquina que acepte el lenguaje $\Sigma^* - L(M)$? (Entonces, el complemento de un lenguaje regular con respecto a Σ^* 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: Consideré 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, examinaremos 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 retroceso.

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 deba 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.

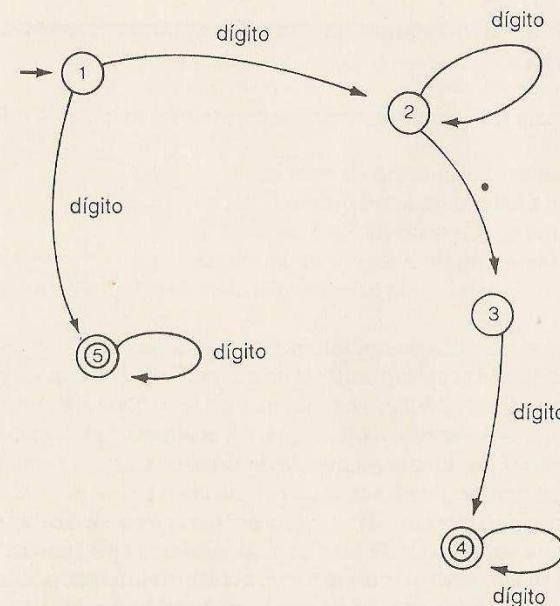


Figura 1.17 Diagrama de transiciones que acepta cadenas que representan enteros o cadenas que representan números reales en notación decimal

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 $(S, \Sigma, \rho, \iota, F)$, donde

- S es un conjunto finito de estados.
- Σ es el alfabeto de la máquina.
- ρ es un subconjunto de $S \times \Sigma \times S$.
- ι (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 ρ representa las posibles transiciones de la máquina. Es decir, la tupla (p, x, q) está en ρ 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 $(S, \Sigma, \rho, \iota, F)$, acepta la cadena no vacía $x_1 x_2 \dots x_n \in \Sigma^*$ si y sólo si existe una secuencia de estados s_0, s_1, \dots, s_n tal que $s_0 = \iota$, $s_n \in F$ y para cada entero j de 1 a n , $(s_{j-1}, x_j, s_j) \in \rho$.

Observe también que tanto (p, x, q_1) como (p, x, q_2) pueden estar en ρ 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

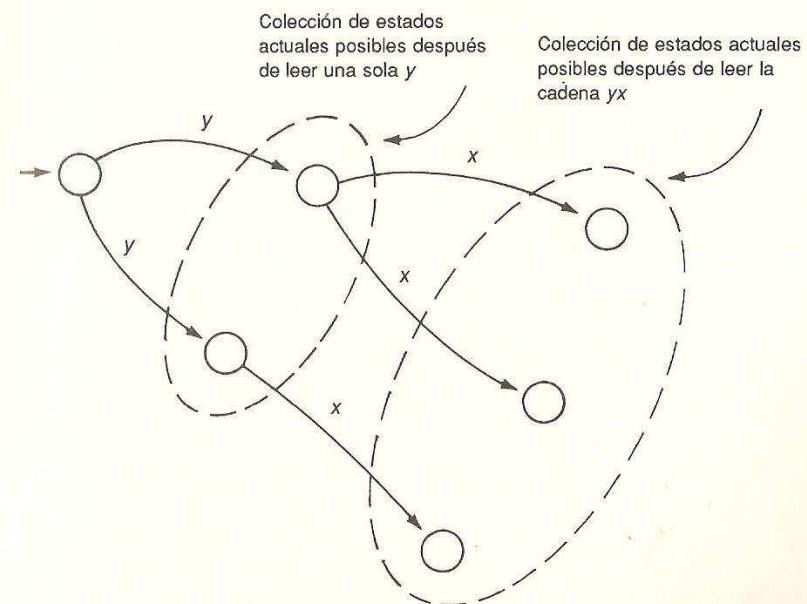


Figura 1.18 Análisis de una cadena con un autómata finito no determinista, siguiendo las opciones en paralelo

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, Σ, p, ι, 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 la quíntupla $(S', \Sigma, \delta, \iota', F')$, donde $S' = \mathcal{P}(S)$, $\iota' = \{\iota\}$, F' es la colección de subconjuntos de S que contienen por lo menos un estado de F , y δ es la función de $S' \times \Sigma$ a S' tal que para cada x en Σ 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 δ 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 i al estado s_n , que recorre arcos rotulados w_1, w_2, \dots, w_n , existe una ruta en M' del estado i' 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 i' 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 i a s_n que recorre arcos rotulados w_1, w_2, \dots, w_n .

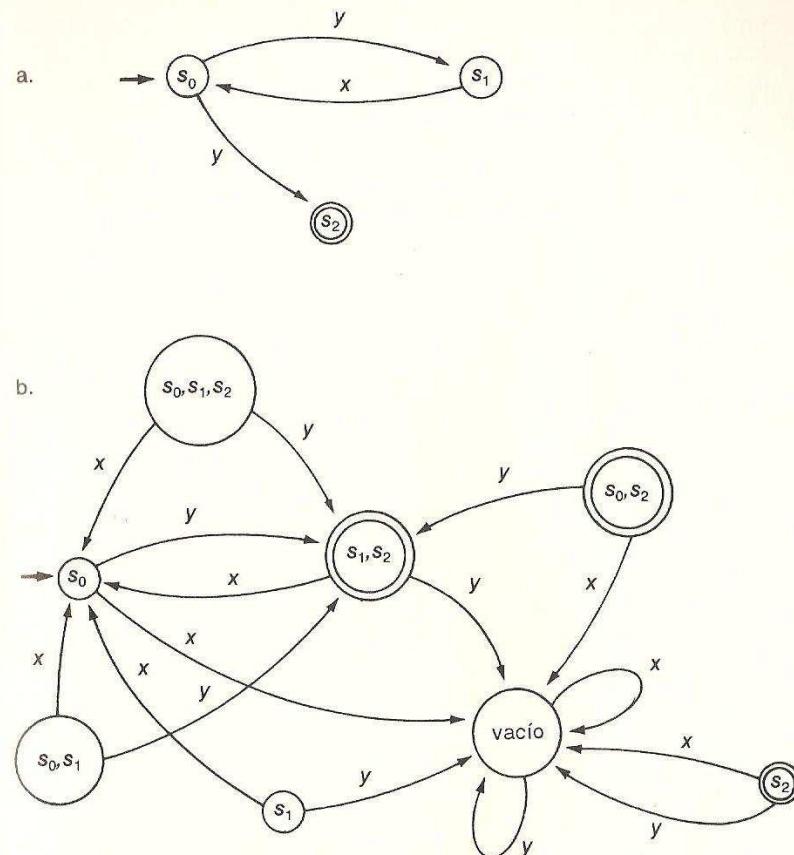


Figura 1.19 a. Diagrama de transiciones para un autómata finito no determinista
b. Diagrama de transiciones para el autómata finito determinista equivalente, construido de acuerdo con la demostración del teorema 1.3

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 ι y s'_n debe ser $\iota' = \{\iota\}$, 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 ι 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 ι' 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 ι' 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 ι' 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 ι 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 ι a s , que recorre arcos rotulados w_1, w_2, \dots, w_{n+1} , como se requiere.

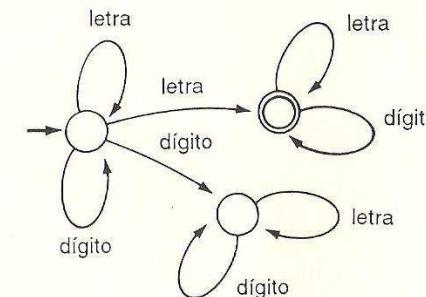
■
inversa, si $M = (S, \Sigma, \delta, \iota, F)$ es un autómata finito determinista, entonces el autómata finito no determinista $(S, \Sigma, \rho, \iota, F)$, donde $(p, x, q) \in \rho$ si y sólo si $\delta(p, x) = q$, acepta el mismo lenguaje.

■

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 $(S, \Sigma, \delta, \iota, F)$ es un autómata finito no determinista, demostrando que la función de transición δ 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.

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.

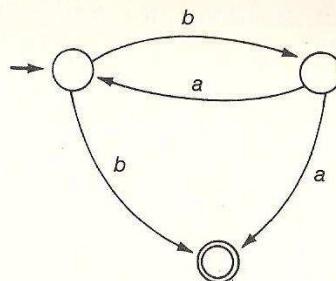
Para concluir esta sección, presentamos un teorema que resume de manera explícita la relación entre los lenguajes aceptados por los autómatas finitos deterministas y no deterministas.

TEOREMA 1.4

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

DEMOSTRACIÓN

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



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 y 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 que se 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 λ .

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 los 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 que 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 Σ , decimos que G es una gramática del alfabeto Σ . En este caso, las cadenas generadas por G son en realidad cadenas de Σ^* . Por lo tanto, una gramática G de un alfabeto Σ especifica un lenguaje de Σ 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 $\{x^m y^n z^m : m, n \in \mathbb{N}\}$.

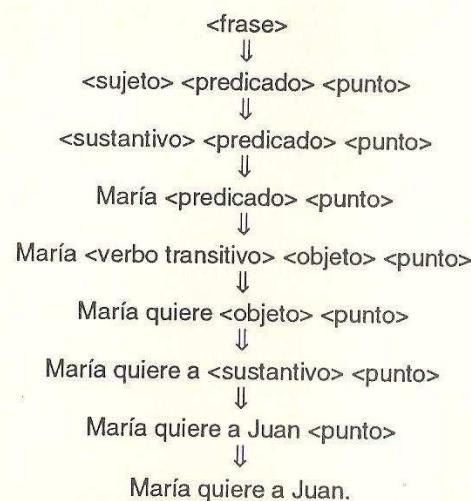


Figura 1.21 Derivación de la frase “María quiere a Juan.”

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

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

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

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

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

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

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

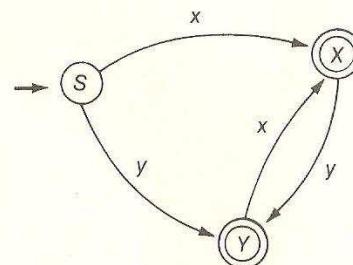


Figura 1.25 Gramática regular G y diagrama de transiciones para un autómata finito M tal que $L(G) = L(M)$

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

DEMOSTRACIÓN

Si G es una gramática regular de Σ , 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' , ι es el símbolo inicial de G' , F es la colección de no terminales de G' que aparecen en el lado izquierdo de alguna regla λ , y ρ 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 Σ para la cual los no terminales son los estados de S , el símbolo inicial es ι y las reglas de reescritura son de la forma $P \rightarrow xQ$ si (P, x, Q) está en ρ y $Q \rightarrow \lambda$ si Q está en 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).

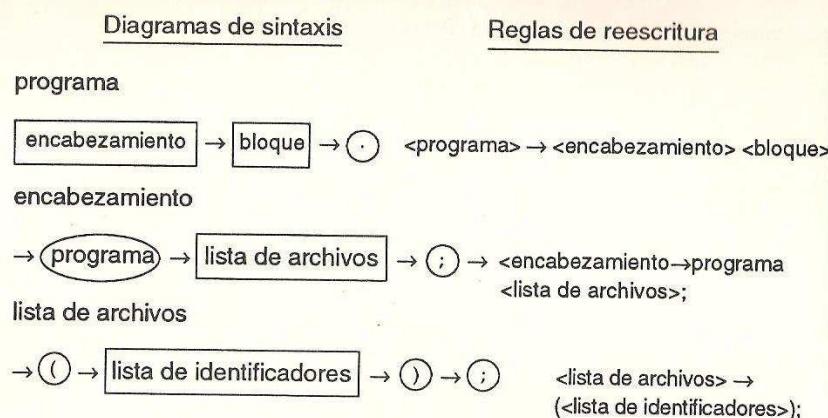


Figura 1.26 Algunos diagramas de sintaxis que describen partes del lenguaje de programación Pascal, comparados con sus formas gramaticales equivalentes

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?

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

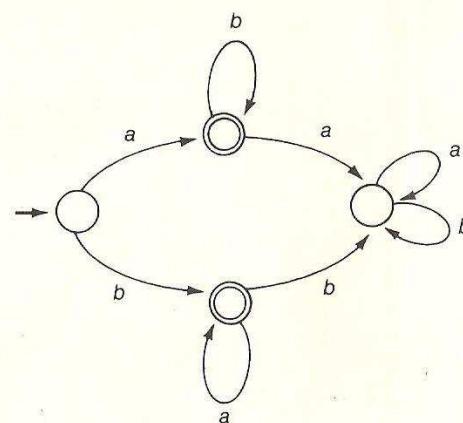
3. Convierta la siguiente gramática regular en otra gramática regular que no contenga reglas de reescritura cuyos lados derechos consistan en un solo terminal, pero que a la vez genere el mismo lenguaje. Describa con sus propias palabras el lenguaje generado.

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

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

ábeto Σ . Es decir, nos interesan los subconjuntos más simples de Σ^* . Quizásiste conjeturarás que serían los subconjuntos que consisten en cadenas simples de longitud uno, y de hecho éstos son los lenguajes que utilizamos 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 Σ .

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 combinar 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 tendría que identificar los estados iniciales de los dos diagramas, como se muestra en la figura 1.27b. Sin embargo, el diagrama resultante acepta la cadena $xyxyx$, que no está en la unión de los dos lenguajes originales.

Aquél 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 permitirnos intercambiar entre ambas. La respuesta se encuentra en la introducción a un nuevo estado inicial a partir del cual podemos 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; déjelo como un estado de aceptación si y sólo si uno de los estados iniciales era de aceptación; para cada estado que sea punto de destino de uno de los estados iniciales originales, dibuje un arco, con la misma etiqueta, a partir del nuevo estado inicial; 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 sobre las máquinas originales, y viceversa. Por lo tanto, la unión de dos lenguajes regulares cualesquiera también es regular.

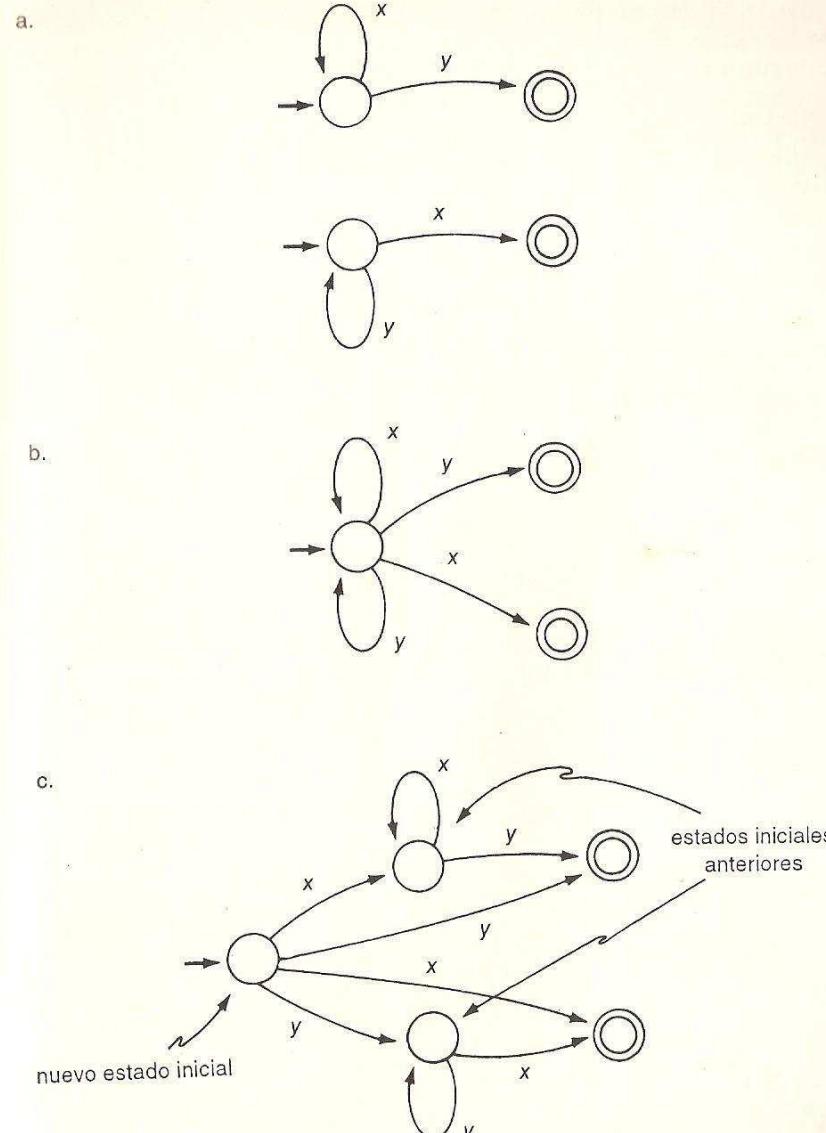


Figura 1.27 Formación de un diagrama de transiciones para un autómata finito que acepta la unión de dos lenguajes regulares

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 es 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 Σ^* el conjunto de todas las cadenas finitas que pueden formarse a partir del alfabeto Σ).

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 λ pertenece a la estrella de Kleene de cualquier lenguaje, debe pertenecer a \emptyset^* . Por ende, $\emptyset^* = \{\lambda\}$ (la razón por la cual no incluimos $\{\lambda\}$ como uno de nuestros bloques de construcción es que $\{\lambda\}$ 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

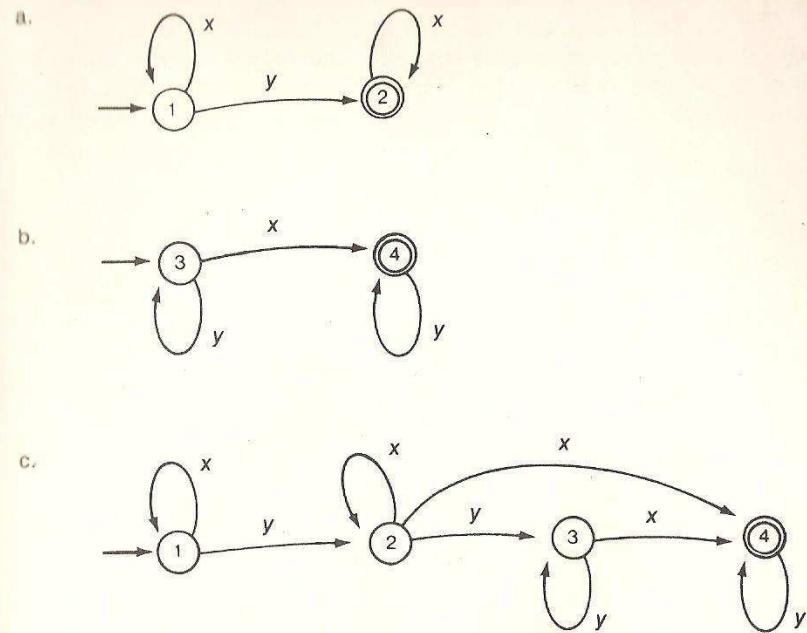


Figura 1.28 Formación de un diagrama de transiciones para un autómata finito que acepta la concatenación de dos lenguajes regulares

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 desde el 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 corresponde 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**.

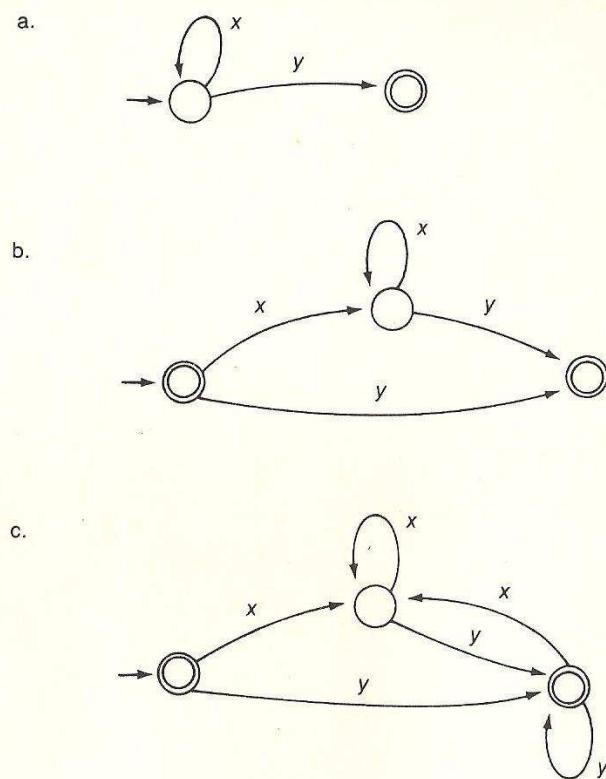


Figura 1.29 Formación de un diagrama de transiciones para un autómata finito que acepta la estrella de Kleene de un lenguaje regular

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 Σ) se define como sigue:

- \emptyset es una expresión regular.
- Cada miembro de Σ 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 Σ 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 Σ 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 e 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 Σ , los lenguajes regulares de Σ son exactamente los lenguajes representados por las expresiones regulares de Σ .

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 los lenguajes aceptados por estos diagramas genera-

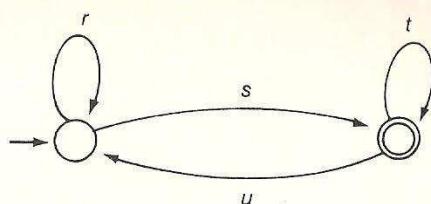


Figura 1.30 Arcos posibles en un diagrama de transiciones con dos estados, uno inicial y otro de aceptación

lizados 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. Representamos 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 (uv 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 en 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

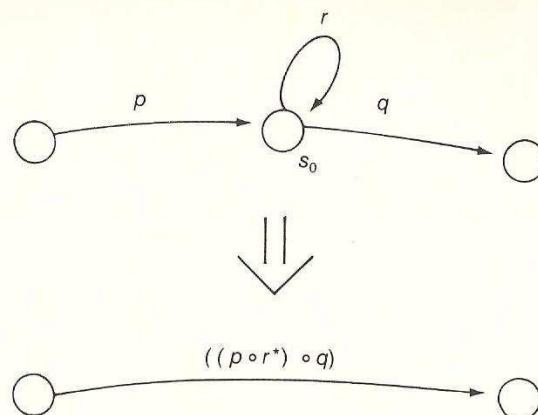


Figura 1.31 Eliminación de un estado s_0 de un diagrama de transiciones generalizado para un autómata finito

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*.

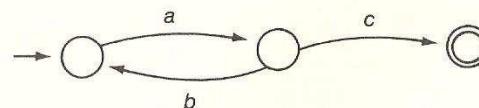
Por último, una consecuencia de los teoremas 1.3 y 1.6 es que los sistemas de comparación de patrones que se basan exclusivamente en la unión, la concatenación y la estrella de Kleene para la representación de patrones, son un tanto restrictivos en cuanto a los patrones que se pueden solicitar. Por otra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

Ejercicios

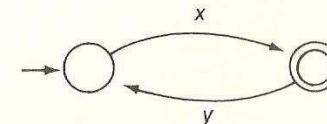
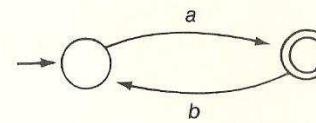
1. ¿Cuáles de los lenguajes descritos por las siguientes expresiones regulares para el alfabeto $\{x, y, z\}$ son infinitos? Describa, en una sola frase, el contenido de cada uno de estos lenguajes infinitos y elabore una lista exhaustiva de las cadenas de los lenguajes que sean finitos.

- a. $(x \circ (y \circ z^*))$
- b. $(x^* \circ (y \circ z))$
- c. $((z \cup y) \circ x)$
- d. $(z \cup y)^*$
- e. $(y \circ y)^*$
- f. $(x^* \cup y^*)$
- g. $((x \circ x) \cup z)$
- h. $((z \cup y) \cup x)$

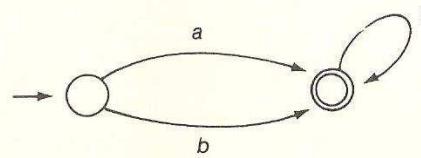
2. Dibuje un diagrama de transiciones que acepte la estrella de Kleene del lenguaje aceptado por el siguiente diagrama.



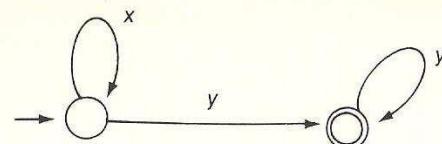
3. Dibuje un diagrama de transiciones que acepte la unión de los lenguajes aceptados por los siguientes diagramas.



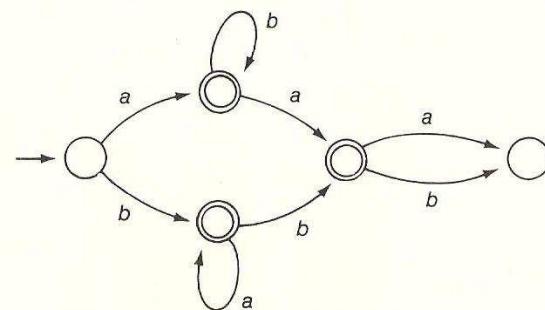
4. Dibuje un diagrama de transiciones que acepte la concatenación del lenguaje aceptado por



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 escrita 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

- 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 $\{x, y, z\}$.
- Muestre que si L_1 y L_2 son lenguajes regulares, entonces $L_1 \cap L_2$ es regular.
- Muestre que si L_1 y L_2 son lenguajes regulares, entonces $L_1 - L_2$ es regular.
- a. Si Σ es un alfabeto, ¿es la colección de palíndromos de Σ^* un lenguaje regular? ¿Por qué?
b. Si L es un lenguaje regular, ¿es la colección de palíndromos de L un lenguaje regular? ¿Por qué?
- Muestre que si L es un lenguaje regular del alfabeto Σ , entonces también es regular el lenguaje que consiste en todas las cadenas de la forma wv , donde $w \in L$ y $v \in \Sigma^* - L$.
- 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 .
- Muestre que si L es un lenguaje regular, entonces también es regular la colección de cadenas cuyas inversas se encuentran así mismo en L .