

Autómatas de pila y lenguajes independientes del contexto

2.1 Autómatas de pila

Definición de los autómatas de pila
Autómatas de pila como aceptadores de lenguajes

2.2 Gramáticas independientes del contexto

Definición de las gramáticas independientes del contexto
Gramáticas independientes del contexto y autómatas de pila
Forma normal de Chomsky

2.3 Límites de los autómatas de pila

Alcance de los lenguajes independientes del contexto
Autómatas de pila deterministas
Principio de preanálisis

2.4 Analizadores sintácticos $LL(k)$

Proceso de análisis sintáctico LL
Aplicación del principio de preanálisis
Tablas de análisis sintáctico LL

2.5 Analizadores sintácticos $LR(k)$

Proceso de análisis sintáctico LR
Implantación de analizadores sintácticos $LR(k)$
Tablas de análisis sintáctico LR

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

2.6 Comentarios finales

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

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

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

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

2.1 AUTÓMATAS DE PILA

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

Definición de los autómatas de pila

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

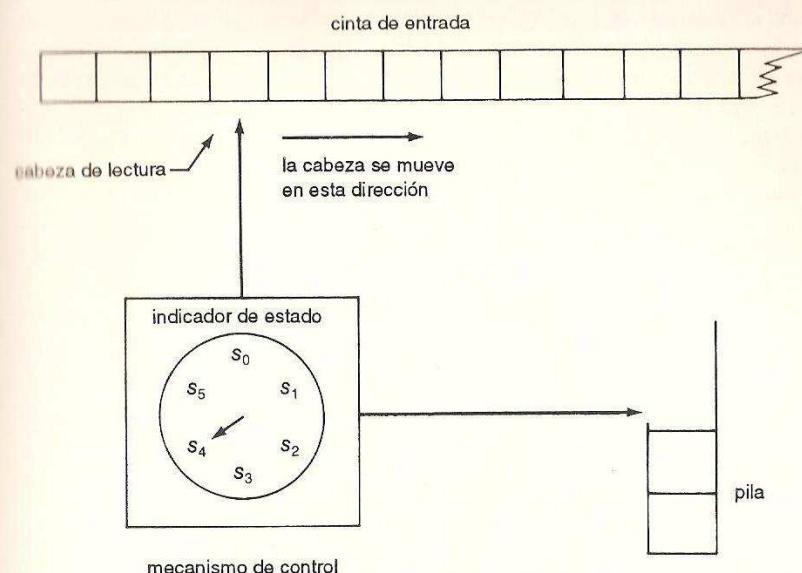


Figura 2.1 Autómata de pila

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

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

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

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

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

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

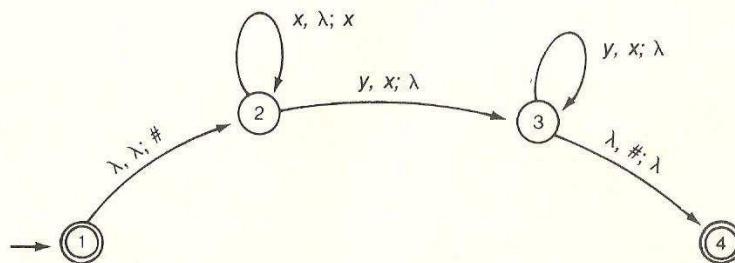


Figura 2.2 Diagrama de transiciones para un autómata de pila

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

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

Autómatas de pila como aceptadores de lenguajes

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

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

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

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

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

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

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

TEOREMA 2.1

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

DEMOSTRACIÓN

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

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

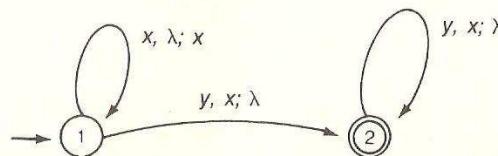


Figura 2.3 Autómata de pila que puede aceptar cadenas sin una pila vacía

2. Elimine la característica de aceptación de cada estado de aceptación de M . Luego, añada un estado p junto con las transiciones que permiten a la máquina pasar de cada uno de los antiguos estados de aceptación a p sin leer, extraer o insertar un símbolo.

3. Para cada x en Γ (sin incluir $\#$), introduzca la transición $(p, \lambda, x; p, \lambda)$.
4. Añada un nuevo estado de aceptación q y la transición $(p, \lambda, \#; q, \lambda)$.

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

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

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

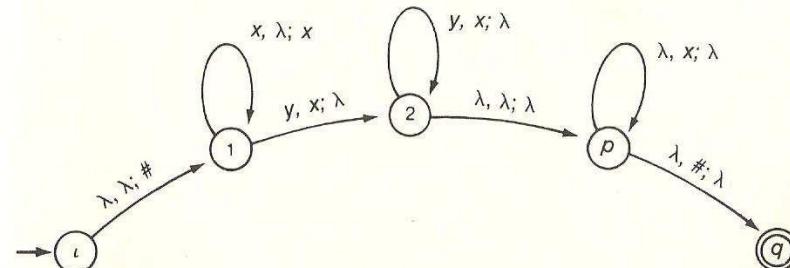
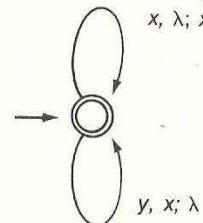


Figura 2.4 El diagrama de la figura 2.3 después de modificarlo para que vacíe su pila antes de aceptar una cadena

Ejercicios

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



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

2.2 GRAMÁTICAS INDEPENDIENTES DEL CONTEXTO

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

Definición de las gramáticas independientes del contexto

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

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

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

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

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

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

$$S \Rightarrow zMNz \Rightarrow zMbNbzbz \Rightarrow zMbzbzbz \Rightarrow zaMabzbzbz \Rightarrow zazabzbzbz$$

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

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

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

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

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

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

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

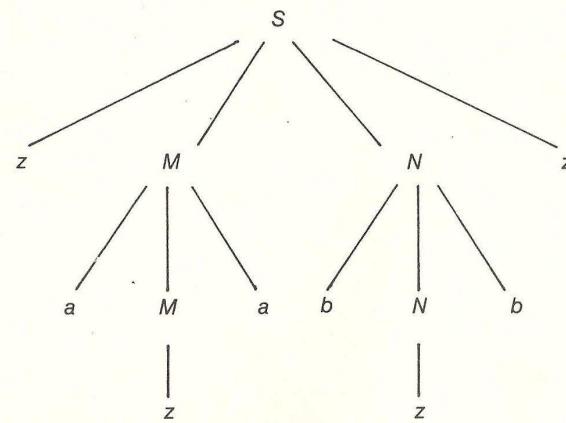


Figura 2.6 Árbol de análisis sintáctico para la cadena $zazabzbz$ utilizando la gramática de la figura 2.5

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

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

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

Gramáticas independientes del contexto y autómatas de pila

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

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

TEOREMA 2.2

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

DEMOSTRACIÓN

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

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

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

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

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

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

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

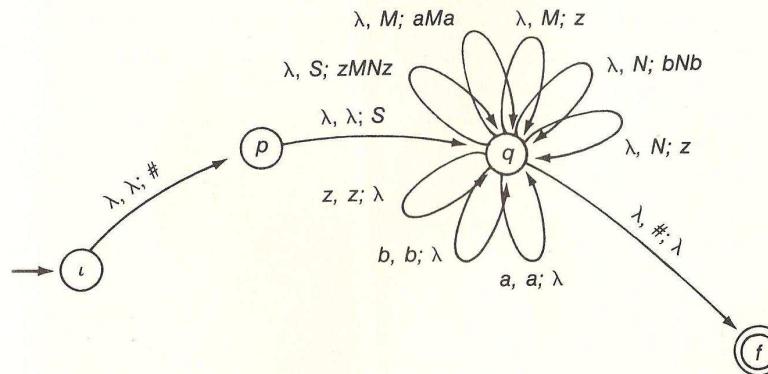


Figura 2.8 Diagrama de transiciones de un autómata de pila construido a partir de la gramática de la figura 2.5 utilizando las técnicas presentadas en la demostración del teorema 2.2

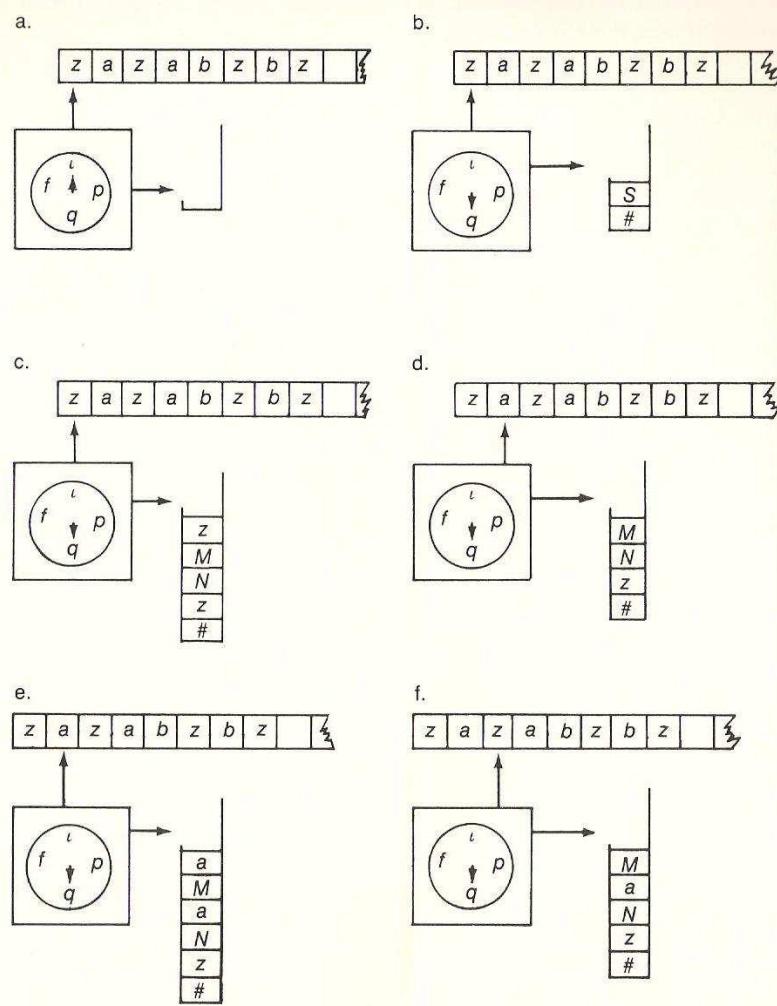


Figura 2.9 Autómata de pila en funcionamiento

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

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

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

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

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

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

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

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

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

TEOREMA 2.3

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

DEMOSTRACIÓN

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

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

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

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

1. Para cada estado de aceptación f de M , forme la regla de reescritura $S \rightarrow \langle \iota, \lambda, f \rangle$, donde ι es el estado inicial de M .

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

Se obtiene otra colección de reglas de reescritura con

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

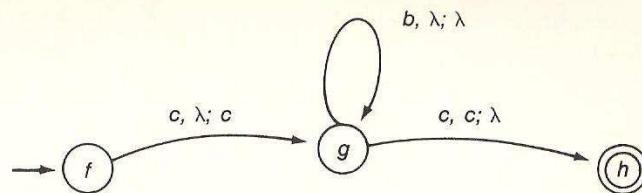


Figura 2.11 Diagrama de transiciones para un autómata de pila que acepta $\{cb^n c : n \in N^*\}$

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

Forma normal de Chomsky

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

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

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

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

Del paso 1

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

Del paso 2

$$\begin{aligned} &\langle f, \lambda, f \rangle \rightarrow \lambda \\ &\langle g, \lambda, g \rangle \rightarrow \lambda \\ &\langle h, \lambda, h \rangle \rightarrow \lambda \end{aligned}$$

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

$$\begin{aligned} &\langle g, c, f \rangle \rightarrow c \langle h, \lambda, f \rangle \\ &\langle g, c, g \rangle \rightarrow c \langle h, \lambda, g \rangle \\ &\langle g, c, h \rangle \rightarrow c \langle h, \lambda, h \rangle \end{aligned}$$

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

$$\begin{aligned} &\langle f, \lambda, f \rangle \rightarrow c \langle g, c, f \rangle \quad \langle f, \lambda, f \rangle \rightarrow \langle f, c, f \rangle \\ &\langle f, \lambda, f \rangle \rightarrow c \langle g, c, g \rangle \quad \langle g, \lambda, f \rangle \rightarrow \langle f, c, f \rangle \\ &\langle f, \lambda, f \rangle \rightarrow c \langle g, c, h \rangle \quad \langle h, \lambda, f \rangle \rightarrow \langle h, c, f \rangle \\ &\langle f, \lambda, g \rangle \rightarrow c \langle g, c, f \rangle \quad \langle f, \lambda, g \rangle \rightarrow \langle f, c, g \rangle \\ &\langle f, \lambda, g \rangle \rightarrow c \langle g, c, g \rangle \quad \langle g, \lambda, g \rangle \rightarrow \langle g, c, g \rangle \\ &\langle f, \lambda, g \rangle \rightarrow c \langle g, c, h \rangle \quad \langle h, \lambda, g \rangle \rightarrow \langle h, c, g \rangle \\ &\langle f, \lambda, h \rangle \rightarrow c \langle g, c, f \rangle \quad \langle f, \lambda, h \rangle \rightarrow \langle f, c, h \rangle \\ &\langle f, \lambda, h \rangle \rightarrow c \langle g, c, g \rangle \quad \langle g, \lambda, h \rangle \rightarrow \langle g, c, h \rangle \\ &\langle f, \lambda, h \rangle \rightarrow c \langle g, c, h \rangle \quad \langle h, \lambda, h \rangle \rightarrow \langle h, c, h \rangle \end{aligned}$$

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

$$\begin{aligned} &\langle g, \lambda, f \rangle \rightarrow b \langle g, \lambda, f \rangle \quad \langle f, \lambda, f \rangle \rightarrow \langle f, c, f \rangle \\ &\langle g, \lambda, f \rangle \rightarrow b \langle g, \lambda, g \rangle \quad \langle g, \lambda, f \rangle \rightarrow \langle g, c, f \rangle \\ &\langle g, \lambda, f \rangle \rightarrow b \langle g, \lambda, h \rangle \quad \langle h, \lambda, f \rangle \rightarrow \langle h, c, f \rangle \\ &\langle g, \lambda, g \rangle \rightarrow b \langle g, \lambda, f \rangle \quad \langle f, \lambda, g \rangle \rightarrow \langle f, c, g \rangle \\ &\langle g, \lambda, g \rangle \rightarrow b \langle g, \lambda, g \rangle \quad \langle g, \lambda, g \rangle \rightarrow \langle g, c, g \rangle \\ &\langle g, \lambda, g \rangle \rightarrow b \langle g, \lambda, h \rangle \quad \langle h, \lambda, g \rangle \rightarrow \langle h, c, g \rangle \\ &\langle g, \lambda, h \rangle \rightarrow b \langle g, \lambda, f \rangle \quad \langle f, \lambda, h \rangle \rightarrow \langle f, c, h \rangle \\ &\langle g, \lambda, h \rangle \rightarrow b \langle g, \lambda, g \rangle \quad \langle g, \lambda, h \rangle \rightarrow \langle g, c, h \rangle \\ &\langle g, \lambda, h \rangle \rightarrow b \langle g, \lambda, h \rangle \quad \langle h, \lambda, h \rangle \rightarrow \langle h, c, h \rangle \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

TEOREMA 2.4

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

DEMOSTRACIÓN

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

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

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

Ahora reemplazamos cada regla de G' de la forma

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

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

$$\begin{aligned} N &\rightarrow N_1 R_1 \\ R_1 &\rightarrow N_2 R_2 \\ &\vdots \\ R_{n-1} &\rightarrow N_{n-1} N_n \end{aligned}$$

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

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

■

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

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

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

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

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

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

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

que genera el mismo lenguaje, no la tiene.

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

Gramática para $L - \{\lambda\}$
con símbolo inicial S

$$\begin{aligned} S &\rightarrow MN \\ N &\rightarrow MS \\ N &\rightarrow x \\ M &\rightarrow x \end{aligned}$$

Gramática para L
con símbolo inicial S'

$$\left. \begin{aligned} S' &\rightarrow \lambda \\ S' &\rightarrow MN \end{aligned} \right\} \text{Nuevas reglas}$$

$$\begin{aligned} S' &\rightarrow MN \\ S' &\rightarrow MS \\ S' &\rightarrow x \\ M' &\rightarrow x \end{aligned}$$

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

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

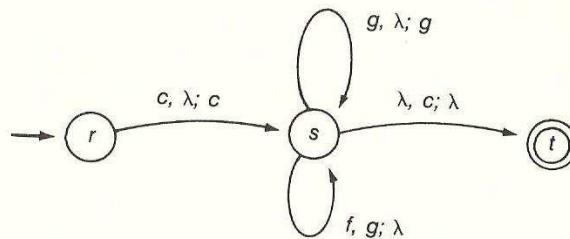
Ejercicios

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

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

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

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



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

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

2.3 LÍMITES DE LOS AUTÓMATAS DE PILA

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

Alcance de los lenguajes independientes del contexto

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

TEOREMA 2.5

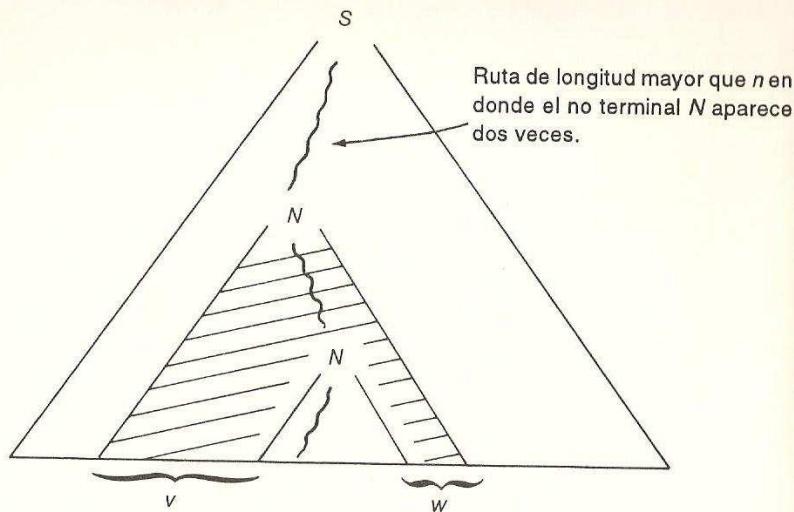
Si L es un lenguaje independiente del contexto que contiene un número infinito de cadenas, entonces debe existir en L una cadena que tenga la forma $svuw^t$, donde s, v, u, w y t son subcademas, por lo menos una de v y w es no vacía, y sv^nuw^nt está en L para cada $n \in \mathbb{N}^+$.

DEMOSTRACIÓN

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

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

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

Figura 2.18 Representación gráfica del árbol de derivación T

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

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

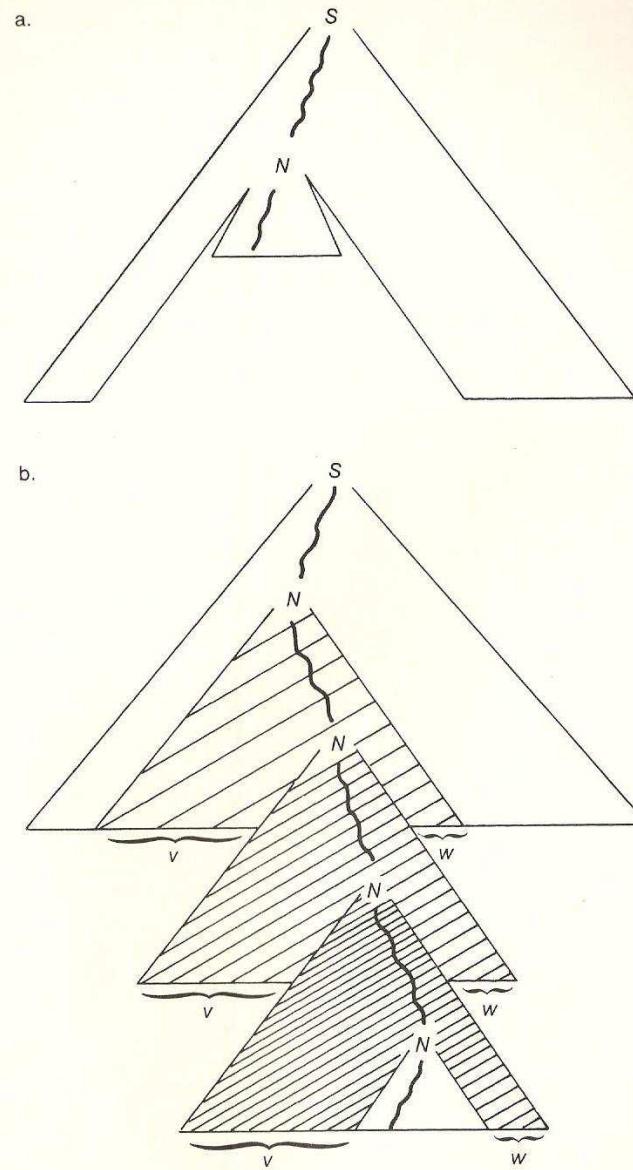


Figura 2.19 Árboles sintácticos que pueden construirse modificando el árbol de la figura 2.17

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

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

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

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

Autómatas de pila deterministas

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

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

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

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

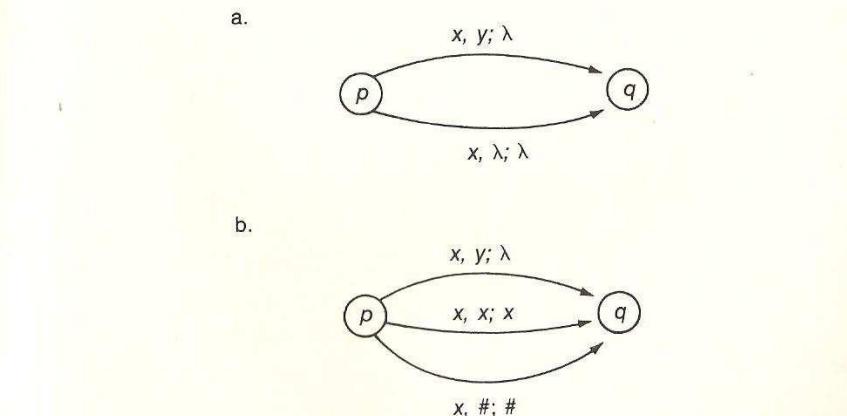


Figura 2.20 Porción del diagrama de transiciones para un autómata de pila que presenta no determinismo y su contrapartida determinista

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

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

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

TEOREMA 2.6

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

DEMOSTRACIÓN

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

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

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

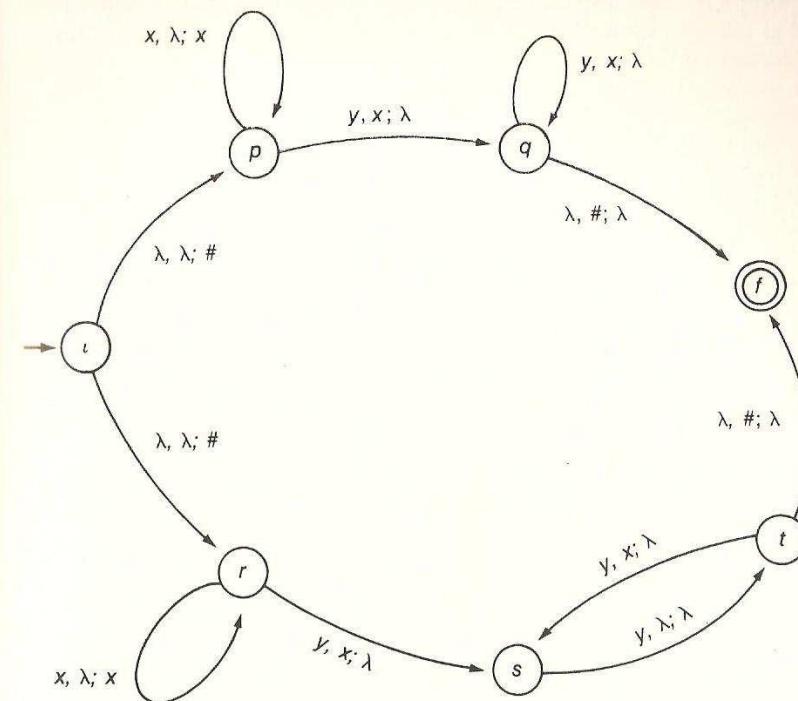


Figura 2.21 Diagrama de transiciones para un autómata de pila M para el cual $L(M) = \{x^n y^n : n \in \mathbb{N}^+\} \cup \{x^n y^{2n} : n \in \mathbb{N}^+\}$

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

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

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

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

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

■

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

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

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

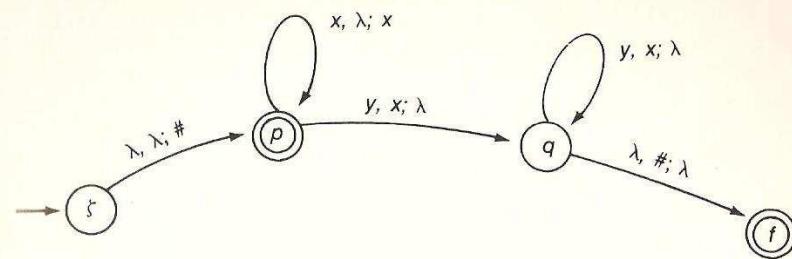


Figura 2.22 Diagrama parcial para un autómata de pila determinista M donde $L(M) = \{x^n : n \in \mathbb{N}^+\} \cup \{x^n y^n : n \in \mathbb{N}^+\}$

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

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

Principio de preanálisis

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

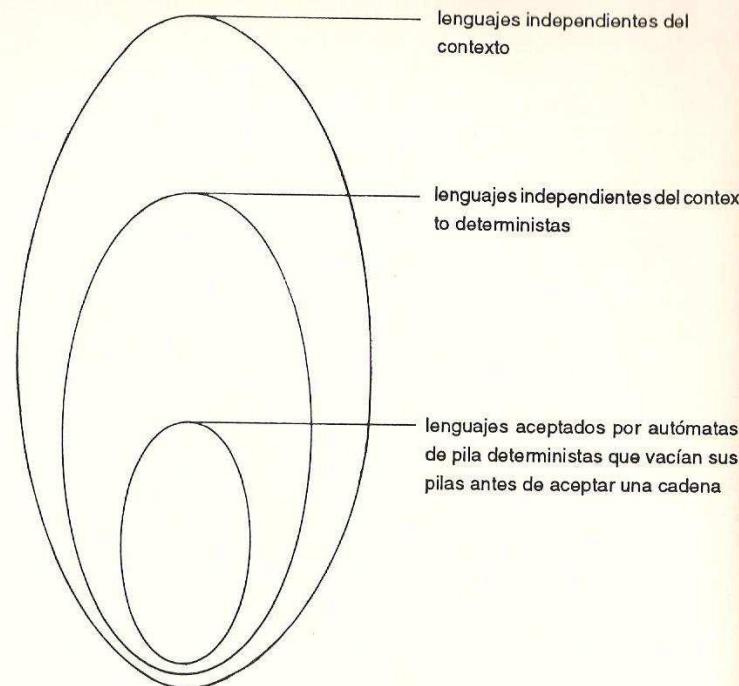


Figura 2.23 Clasificación de los lenguajes independientes del contexto

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

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

```

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

```

Figura 2.24 Ciclo while típico

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

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

Ejercicios

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

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

2.4 ANALIZADORES SINTÁCTICOS LL(k)

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

Proceso de análisis sintáctico LL

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

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

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

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

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

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

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

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

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

al final de la rutina.

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

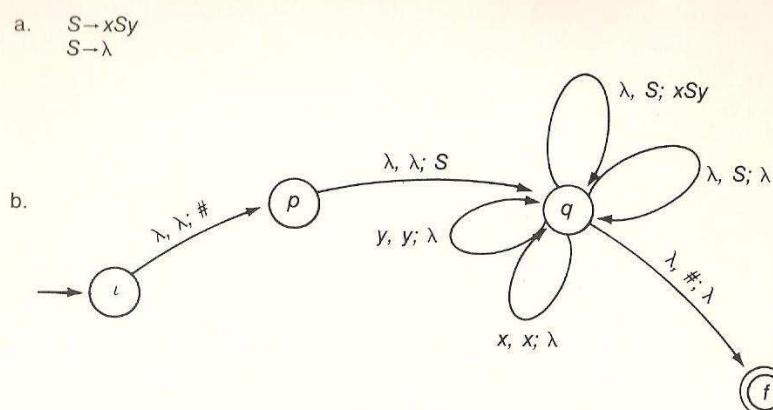


Figura 2.25 Gramática independiente del contexto y diagrama de transiciones asociado para un autómata de pila

```

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

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

Aplicación del principio de preanálisis

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

```

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

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

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

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

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

$$\begin{array}{l} S \rightarrow xSz \\ S \rightarrow xyTyz \\ T \rightarrow \lambda \end{array}$$

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

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

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

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

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

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

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

Tablas de análisis sintáctico LL

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

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

	<i>a</i>	<i>b</i>	<i>z</i>	FDC
<i>S</i>	error	error	<i>zMNz</i>	error
<i>m</i>	<i>aMa</i>	error	<i>z</i>	error
<i>N</i>	error	<i>bNb</i>	<i>z</i>	error

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

```

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

```

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

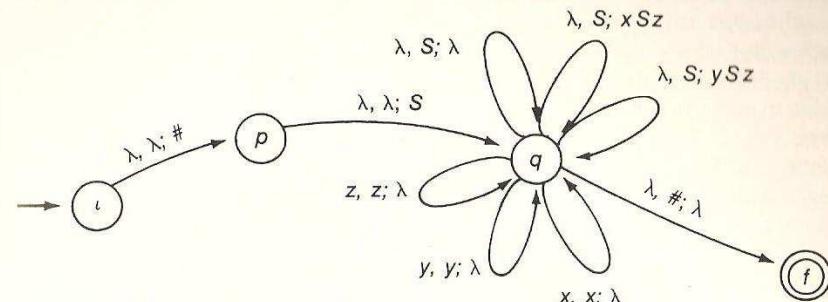
x	y	FDC
xSy	λ	λ

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

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

Ejercicios

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



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

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

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

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

2.5 ANALIZADORES SINTÁCTICOS $LR(k)$

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

Proceso de análisis sintáctico LR

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

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

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

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

1. Establezca cuatro estados: uno inicial llamado i , un estado de aceptación llamado f y otros dos estados llamados p y q .
2. Introduzca las transiciones $(i, \lambda, \lambda; p, \#)$ y $(q, \lambda, \#; f, \lambda)$, donde suponemos que $\#$ es un símbolo que no se presenta en esta gramática.
3. Para cada símbolo terminal x en la gramática, introduzca la transición $(p, x, \lambda; p, x)$. Estas transiciones permiten que el autómata transfiera a la pila los símbolos de entrada mientras se encuentra en el estado p . La ejecución de esta transición se llama **operación de desplazamiento** ya que su efecto es desplazar un símbolo de la cadena de entrada a la pila.
4. Para cada regla de reescritura $N \rightarrow w$ (donde w representa una cadena de uno o más símbolos) que exista en la gramática, introduzca la transición $(p, \lambda, w; p, N)$. (Aquí permitimos que una transición elimine más de un símbolo de la pila. Para ser más precisos, la transición $(p, i, xy; p, z)$ es una forma abreviada de transición $(p, \lambda, y; p, \lambda)$ seguida por $(p, \lambda, x; p, z)$, donde p_1 es un estado al que no puede llegar ninguna otra transición. Así, para ejecutar la transición $(p, \lambda, xy; p, z)$, un autómata debe tener una y en la cima de la pila con una x inmediatamente debajo). La presencia de estas transiciones significa que si los símbolos

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

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

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

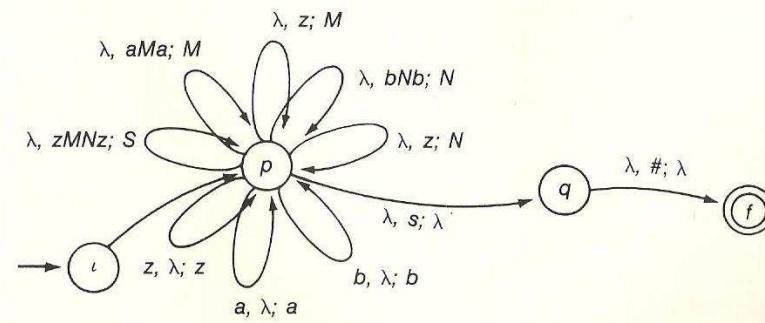


Figura 2.32 Otro diagrama de transiciones para un autómata de pila basado en la gramática de la figura 2.5

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

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

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

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

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

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

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

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

Implantación de analizadores sintácticos $LR(k)$

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

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

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

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

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

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

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

tanto, podemos investigar la estructura interna de la pila observando el símbolo especial de la cima.)

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

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

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

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

```

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

```

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

Tablas de análisis sintáctico LR

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

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

Contenido de la pila	Resto de la entrada
vacía	
①	zazabzbz
① z ②	zazabzbz
① z ② a ③	azabzbz
① z ② a ③ z ⑦	zabzbz
① z ② a ③ M ⑧	abzbz
① z ② a ③ M ⑧ a ⑫	bzbz
① z ② M ④	bzbz
① z ② M ④ b ⑤	bz
① z ② M ④ b ⑤ z ⑨	bz
① z ② M ④ b ⑤ N ⑩	z
① z ② M ④ b ⑤ N ⑩ b ⑬	z
① z ② M ④ N ⑥	
① z ② M ④ N ⑥ z ⑪	
① S ⑯	
vacía	

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

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

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

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

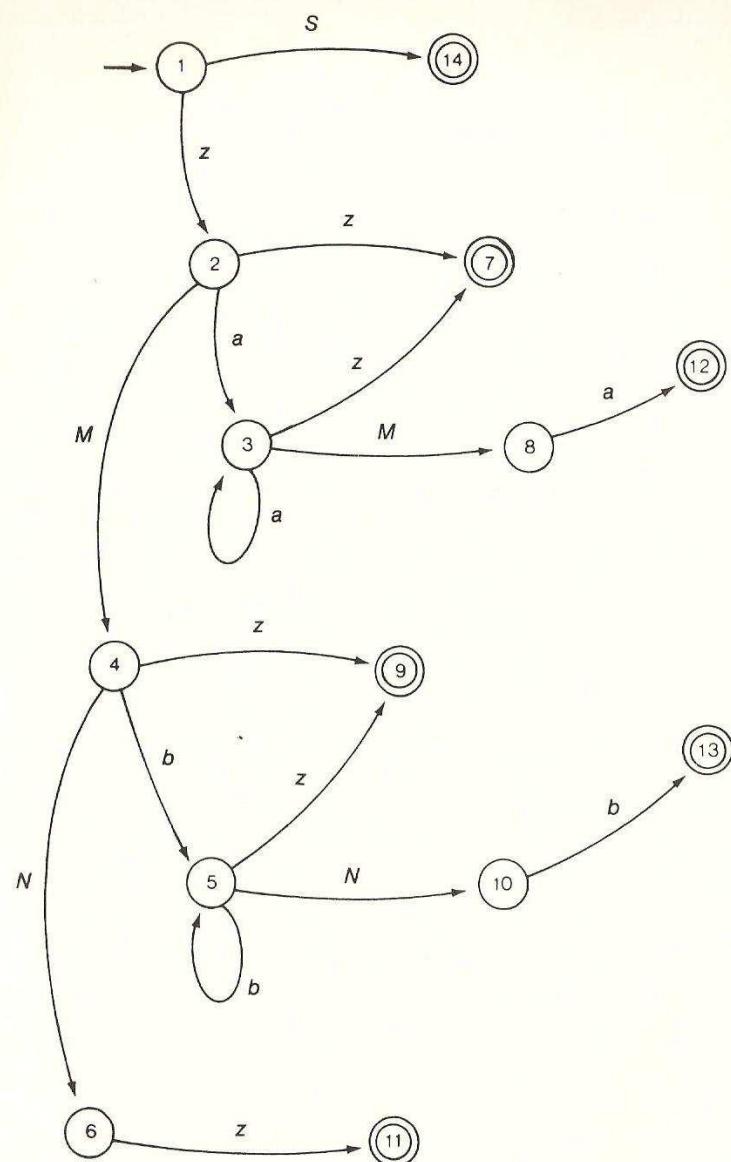


Figura 2.37 Autómata finito a partir del cual se construyó la tabla de la figura 2.34

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

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

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

repeat

analizar la sintaxis hasta ejecutar una reducción
generar código
until termina el análisis sintáctico

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

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

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

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

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

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

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

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

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

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

Ejercicios

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

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

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

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

2.6 COMENTARIOS FINALES

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

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

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

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

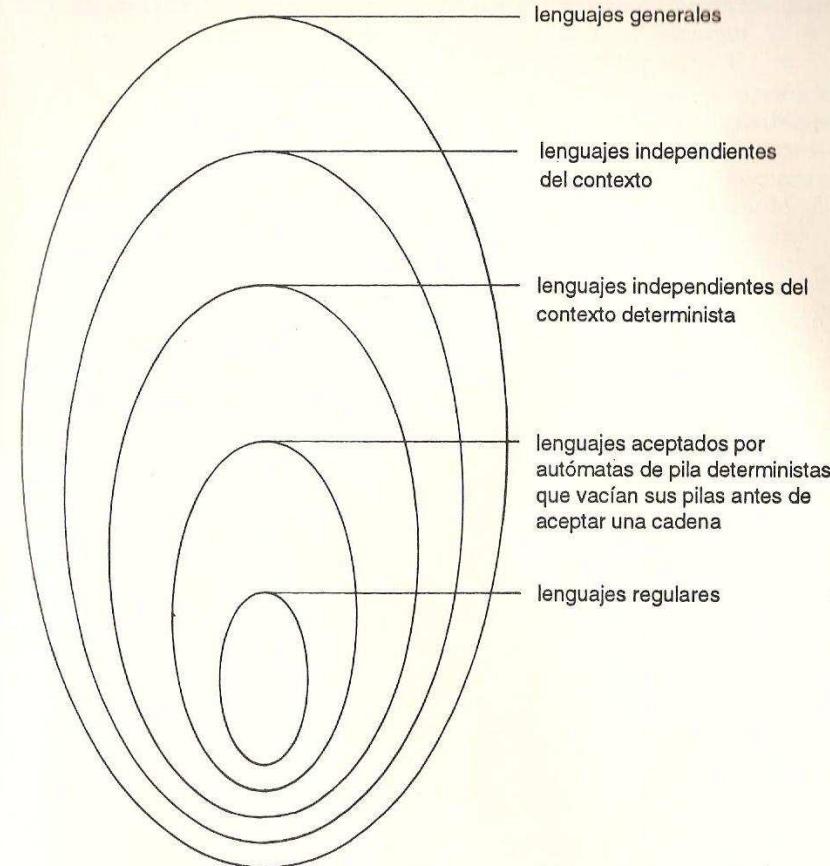


Figura 2.39 La jerarquía de los lenguajes que hemos presentado hasta ahora

Problemas de repaso del capítulo

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