Contenido

Ejemplo 3.7 Considere la siguiente gramática *G* para una secuencia de sentencias:

3.3 ÁRBOLES DE ANÁLISIS GRAMATICAL Y ÁRBOLES SINTÁCTICOS ABSTRACTOS

3.3.1 Árboles de análisis gramatical

Una derivación proporciona un método para construir una cadena particular de terminales a partir de un no terminal inicial. Pero las derivaciones no sólo representan la estructura de las cadenas que construyen. En general, existen muchas derivaciones para la misma cadena. Por ejemplo, construyamos la cadena de tokens

```
( número - número ) * número
```

a partir de nuestra gramática de expresión simple utilizando la derivación de la figura 3.1. Una segunda derivación para esta cadena se proporciona en la figura 3.2. La única diferencia

entre las dos derivaciones es el orden en el cual se suministran los reemplazos, y ésta es de hecho una diferencia superficial. Para aclarar esto necesitamos una representación para la estructura de una cadena de terminales que abstraiga las características esenciales de una derivación mientras se factorizan las diferencias superficiales de orden. La representación que hace esto es una estructura de árbol, y se conoce como árbol de análisis gramatical.

Figura 3.2
Otra derivación para la expresión `(34-3)*42`

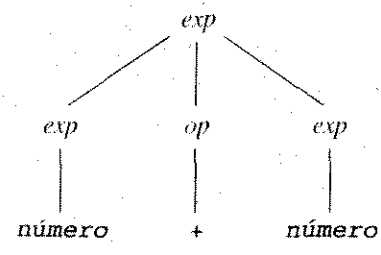
(1)	*exp* ⇒ *exp op exp*	[*exp* → *exp op exp*]
(2)	⇒ **(** *exp* **)** *op exp*	[*exp* → **(** *exp* **)**]
(3)	⇒ **(** *exp op exp* **)** *op exp*	[*exp* → *exp op exp*]
(4)	⇒ **(número** *op exp***)** *op exp*	[*exp* → **número**]
(5)	⇒ **(número** − *exp***)** *op exp*	[*op* → **-**]
(6)	⇒ **(número - número)** *op exp*	[*exp* → **número**]
(7)	⇒ **(número - número) *** *exp*	[*op* → *****]
(8)	⇒ **(número - número) * número**	[*exp* → **número**]

Un **árbol de análisis gramatical** correspondiente a una derivación es un árbol etiquetado en el cual los nodos interiores están etiquetados por no terminales, los nodos hoja están etiquetados por terminales y los hijos de cada nodo interno representan el reemplazo del no terminal asociado en un paso de la derivación.

Para dar un ejemplo simple, la derivación

exp ⇒ *exp op exp*
⇒ **número** *op exp*
⇒ **número** + *exp*
⇒ **número** + **número**

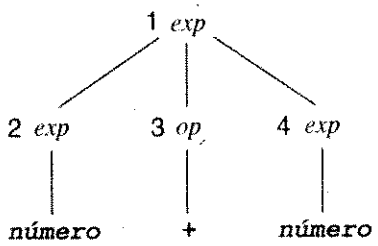
corresponde al árbol de análisis gramatical



El primer paso en la derivación corresponde a los tres hijos del nodo raíz. El segundo paso corresponde al hijo **número** de la *exp* en el extremo izquierdo debajo de la raíz, y de manera similar para los dos pasos restantes. Podemos hacer esta correspondencia explícita al numerar los nodos internos del árbol de análisis gramatical mediante el número de paso en el cual se reemplaza su no terminal asociado en una derivación correspondiente. De este modo, si numeramos la derivación anterior como sigue:

(1) $exp \Rightarrow exp\ op\ exp$
(2) $\Rightarrow$ **número** $op\ exp$
(3) $\Rightarrow$ **número** + exp
(4) $\Rightarrow$ **número** + **número**

podemos numerar los nodos internos del árbol de análisis gramatical respectivamente:



Advierta que esta numeración de los nodos internos del árbol de análisis gramatical es en realidad una **numeración preorden**.

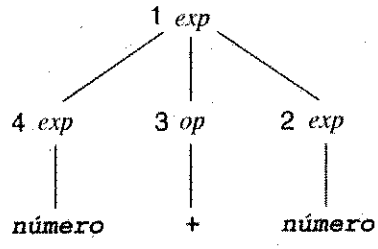
Este mismo árbol de análisis gramatical también corresponde a las derivaciones

$$\begin{aligned} exp &\Rightarrow exp\ op\ exp \\ &\Rightarrow exp\ op\ \textbf{número} \\ &\Rightarrow exp\ +\ \textbf{número} \\ &\Rightarrow \textbf{número}\ +\ \textbf{número} \end{aligned}$$

y

$$\begin{aligned} exp &\Rightarrow exp\ op\ exp \\ &\Rightarrow exp\ +\ exp \\ &\Rightarrow \textbf{número}\ +\ exp \\ &\Rightarrow \textbf{número}\ +\ \textbf{número} \end{aligned}$$

pero se aplicarían diferentes numeraciones de los nodos internos. En realidad la primera de estas dos derivaciones corresponde a la numeración siguiente:



(Dejamos al lector construir la numeración de la otra.) En este caso la numeración es la inversa de una **numeración postorden** de los nodos internos del árbol de análisis gramatical. (Un recorrido postorden visitaría los nodos internos en el orden 4, 3, 2, 1.)

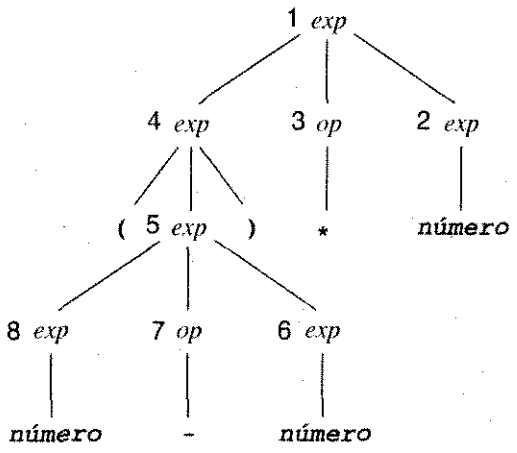
Un árbol de análisis gramatical corresponde en general a muchas derivaciones, que en conjunto representan la misma estructura básica para la cadena de terminales analizada gramaticalmente. Sin embargo, se pueden distinguir derivaciones particulares que están asociadas de manera única con el árbol de análisis gramatical. Una **derivación por la izquierda** es aquella en la cual se reemplaza el no terminal más a la izquierda en cada paso en la deriva-

ción. Por consiguiente, una **derivación por la derecha** es aquella en la cual el no terminal más a la derecha se reemplaza en cada paso de la derivación. Una derivación por la izquierda corresponde a la numeración preorden de los nodos internos de su árbol de análisis gramatical asociado, mientras que una derivación por la derecha corresponde a una numeración postorden en reversa.

En realidad, vimos esta correspondencia en las tres derivaciones y el árbol de análisis gramatical del ejemplo más reciente. La primera de las tres derivaciones que dimos es una derivación por la izquierda, mientras que la segunda es una derivación por la derecha. (La tercera derivación no es por la izquierda ni por la derecha.)

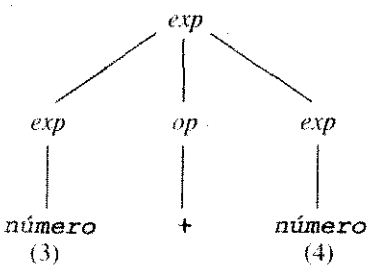
Como un ejemplo más complejo de un árbol de análisis gramatical y de las derivaciones por la izquierda y por la derecha, regresemos a la expresión `(34-3)*42` y las derivaciones que dimos en las figuras 3.1 y 3.2. El árbol de análisis gramatical para esta expresión está dado en la figura 3.3, donde también numeramos los nodos de acuerdo con la derivación de la figura 3.1. Ésta es de hecho una derivación por la derecha, y la numeración correspondiente del árbol de análisis gramatical es una numeración postorden inversa. La derivación de la figura 3.2, por otra parte, es una derivación por la izquierda. (Invitamos al lector a proporcionar una numeración preorden del árbol de análisis gramatical correspondiente a esta derivación.)

Figura 3.3
Árbol de análisis gramatical para la expresión aritmética `(34-3)*42`

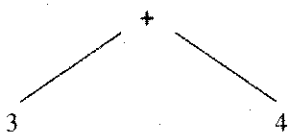


3.3.2 Árboles sintácticos abstractos

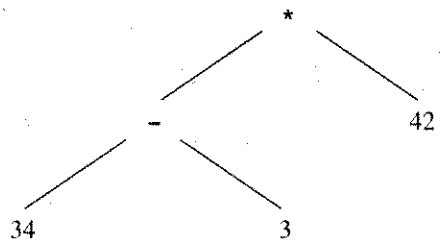
Un árbol de análisis gramatical es una representación útil de la estructura de una cadena de tokens, ya que los tokens aparecen como las hojas del árbol de análisis gramatical (de izquierda a derecha) y los nodos internos del árbol de análisis gramatical representan los pasos en una derivación (en algún orden). Sin embargo, un árbol de análisis gramatical contiene mucha información más de la que es absolutamente necesaria para que un compilador produzca código ejecutable. Para ver esto, considere el árbol de análisis gramatical para la expresión `3+4` de acuerdo con nuestra gramática de expresión simple:



Éste es el árbol de análisis gramatical de un ejemplo anterior. Aumentamos el árbol para mostrar el valor numérico real de cada uno de los tokens **`número`** (éste es un atributo del token que se calcula mediante el analizador léxico o por medio del analizador gramatical). El **principio de la traducción dirigida por sintaxis** establece que el significado, o semántica, de la cadena `3+4` debería relacionarse directamente con su estructura sintáctica como se representa mediante el árbol de análisis gramatical. En este caso el principio de la traducción dirigida por medio de sintaxis significa que el árbol de análisis gramatical debería presuponer que el valor 3 y el valor 4 se van a sumar. En realidad, podemos ver que el árbol da a entender esto de la siguiente manera. La raíz representa la operación de suma de los valores de los dos subárboles *exp* hijos. Cada uno de estos subárboles, por otro lado, representa el valor de su hijo **`número`** único. Sin embargo, existe una manera mucho más simple de representar esta misma información, a saber, mediante el árbol



Aquí, el nodo raíz simplemente se etiqueta por la operación que representa, y los nodos hoja se etiquetan mediante sus valores (en lugar de los tokens **`número`**). De manera similar, la expresión `(34-3)*42`, cuyo árbol de análisis gramatical está dado en la figura 3.3, se puede representar en forma más simple por medio del árbol



En este árbol los tokens de paréntesis en realidad han desaparecido, aunque todavía representan precisamente el contenido semántico de restar 3 de 34, para después multiplicarlo por 42.

Tales árboles representan abstracciones de las secuencias de token del código fuente real, y las secuencias de tokens no se pueden recobrar a partir de ellas (a diferencia de los árboles de análisis gramatical). No obstante, contienen toda la información necesaria para traducir de una forma más eficiente que los árboles de análisis gramatical. Tales árboles se conocen como **árboles sintácticos abstractos**, o para abreviar **árboles sintácticos**. Un analizador sintáctico recorrerá todos los pasos representados por un árbol de análisis gramatical, pero por lo regular sólo construirá un árbol sintáctico abstracto (o su equivalente).

Los árboles sintácticos abstractos pueden imaginarse como una representación de árbol de una notación abreviada denominada **sintaxis abstracta**, del mismo modo que un árbol de análisis gramatical es una representación para la estructura de la sintaxis ordinaria (que también se denomina **sintaxis concreta** cuando se le compara con la abstracta). Por ejemplo, la sintaxis abstracta para la expresión `3+4` debe escribirse como *OpExp*(*Plus*, *ConstExp*(3),*ConstExp*(4)) y la sintaxis abstracta para la expresión `(34-3)*42` se puede escribir como

OpExp(*Times*,*OpExp*(*Minus*,*ConstExp*(34),*ConstExp*(3)),*ConstExp*(42))

En realidad, la sintaxis abstracta puede proporcionar una definición formal utilizando una notación semejante a la BNF, del mismo modo que la sintaxis concreta. Por ejemplo, podríamos escribir las siguientes reglas tipo BNF para la sintaxis abstracta de nuestras expresiones aritméticas simples como

$$exp \rightarrow OpExp(op,exp,exp) \mid ConstExp(integer)$$
$$op \rightarrow Plus \mid Minus \mid Times$$

No investigaremos más al respecto. Nuestro principal interés radica en la estructura de árbol sintáctico real que utilizará el analizador sintáctico, la cual se determinará por medio de una declaración de tipo de datos.[3] Por ejemplo, los árboles sintácticos abstractos para nuestras expresiones aritméticas simples pueden ser determinados mediante las declaraciones de tipo de datos en C

```
typedef enum {Plus,Minus,Times} OpKind;
typedef enum {OpKind,ConstKind} ExpKind;
typedef struct streenode
      { ExpKind kind;
        OpKind op;
        struct streenode *lchild,*rchild;
        int val;
      } STreeNode;
typedef STreeNode *SyntaxTree;
```

Advierta que empleamos tipos enumerados para las dos diferentes clases de nodos de árbol sintáctico (operaciones y constantes enteras), así como para las operaciones (suma, resta, multiplicación) mismas. De hecho, probablemente usaríamos los tokens para representar las operaciones, más que definir un nuevo tipo enumerado. También podríamos haber usado un tipo `union` de C para ahorrar espacio, puesto que un nodo no puede ser al mismo tiempo un nodo de operador y un nodo de constante. Finalmente, observamos que estas tres declaraciones de nodos sólo incluyen los atributos que son directamente necesarios para este ejemplo. En situaciones prácticas habrá muchos más campos para los atributos de tiempo de compilación, tales como tipo de datos, información de tabla de símbolos y así sucesivamente, como se dejará claro con los ejemplos que se presentarán más adelante en este capítulo y en capítulos subsiguientes.

Concluimos esta sección con varios ejemplos de árboles de análisis gramatical y árboles sintácticos utilizando gramáticas que ya consideramos en ejemplos anteriores.

Ejemplo 3.8

Considere la gramática para las sentencias if simplificadas del ejemplo 3.4:

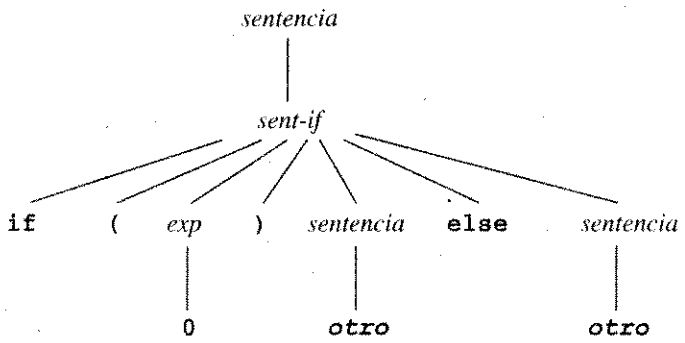
$$sentencia \rightarrow sent\text{-}if \mid \texttt{otro}$$
$$sent\text{-}if \rightarrow \texttt{if (}\ exp\ \texttt{)}\ sentencia$$
$$\mid \texttt{if (}\ exp\ \texttt{)}\ sentencia\ \texttt{else}\ sentencia$$
$$exp \rightarrow \texttt{0} \mid \texttt{1}$$

3. Existen lenguajes para los que la sintaxis abstracta recién dada es esencialmente una declaración de tipo. Véanse los ejercicios.

El árbol de análisis gramatical para la cadena

```
if (0) otro else otro
```

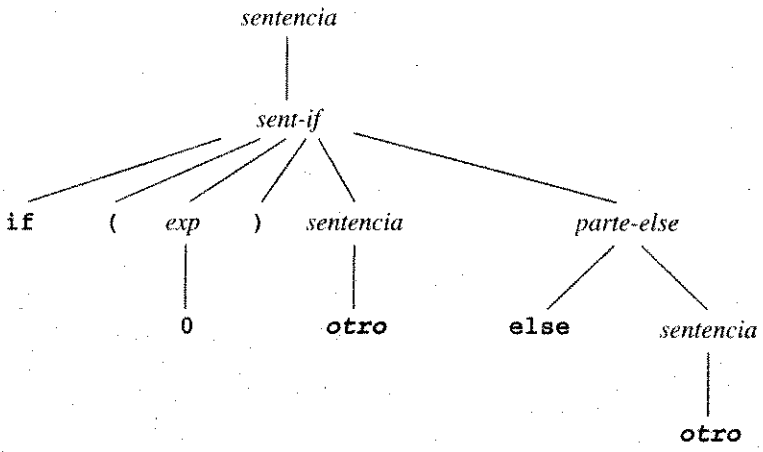
es como se presenta a continuación:



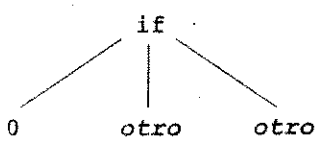
Al utilizar la gramática del ejemplo 3.6,

sentencia → *sent-if* | **otro**
sent-if → **if** **(** *exp* **)** *sentencia parte-else*
parte-else → **else** *sentencia* | ε
exp → **0** | **1**

esta misma cadena tiene el siguiente árbol de análisis gramatical.



Un árbol sintáctico abstracto apropiado para las sentencias if desecharía todo, excepto las tres estructuras subordinadas de la sentencia if: la expresión de prueba, la parte de acción (parte "then") y la parte alternativa (parte "else"). De este modo, un árbol sintáctico para la cadena anterior (usando la gramática del ejemplo 3.4 o la del ejemplo 3.6) sería:



Aquí empleamos los tokens restantes `if` y *`otro`* como etiquetas para distinguir la clase del nodo de sentencia en el árbol sintáctico. Esto se efectuaría más apropiadamente empleando un tipo enumerado. Por ejemplo, un conjunto de declaraciones en C apropiado para la estructura de las sentencias y expresiones en este ejemplo sería como el que se muestra a continuación:

```
typedef enum {ExpK,StmtK} NodeKind;
typedef enum {Zero,One} ExpKind;
typedef enum {IfK,OtherK} StmtKind;
typedef struct streenode
  {NodeKind kind;
   ExpKind ekind;
   StmtKind skind;
   struct streenode
      *test,*thenpart,*elsepart;
  } STreeNode;
typedef STreeNode * SyntaxTree;
```

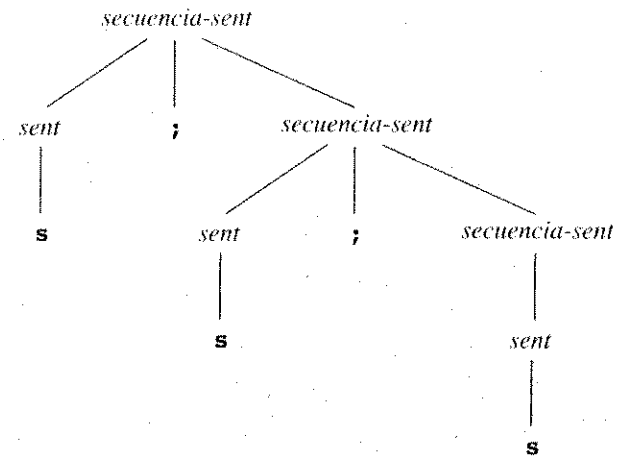
§

Ejemplo 3.9

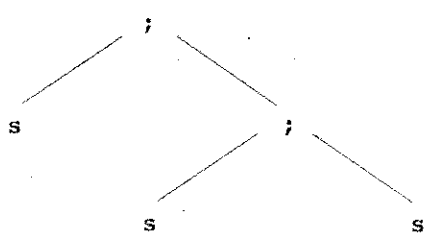
Considere la gramática de una secuencia de sentencias separadas por signos de punto y coma del ejemplo 3.7:

$$\textit{secuencia-sent} \longrightarrow \textit{sent}\ \texttt{;}\ \textit{secuencia-sent} \mid \textit{sent}$$
$$\textit{sent} \longrightarrow \texttt{s}$$

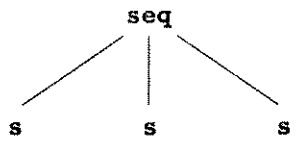
La cadena `s;s;s` tiene el siguiente árbol de análisis gramatical respecto a esta gramática:



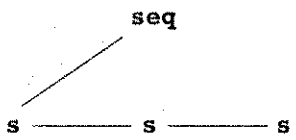
Un posible árbol sintáctico para esta misma cadena es



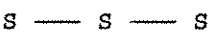
En este árbol los nodos de punto y coma son semejantes a los nodos de operador (como los nodos + de expresiones aritméticas), salvo por la diferencia de que éstos sólo "funcionan" al unirse entre sí las sentencias en una secuencia. En su lugar podríamos intentar unir todos los nodos de sentencia en una secuencia con solamente un nodo, de modo que el árbol sintáctico anterior se convertiría en



El problema con esto es que un nodo `seq` puede tener un número arbitrario de hijos y es difícil tomar precauciones al respecto en una declaración de tipo de datos. La solución es utilizar la representación estándar con **hijo de extrema izquierda y hermanos a la derecha** para un árbol (esto se presenta en la mayoría de los textos sobre estructuras de datos). En esta representación el único vínculo físico desde el padre hasta sus hijos es hacia el hijo de la extrema izquierda. Los hijos entonces se vinculan entre sí de izquierda a derecha en una lista de vínculos estándar, los que se denominan vínculos **hermanos** para distinguirlos de los vínculos padre-hijo. El árbol anterior ahora se convierte en el arreglo de hijo de extrema izquierda y hermanos a la derecha:



Con este arreglo también se puede eliminar el nodo `seq` de conexión, y entonces el árbol sintáctico se convierte simplemente en:



Ésta, como es evidente, es la representación más sencilla y más fácil para una secuencia de cosas en un árbol sintáctico. La complicación es que aquí los vínculos son vínculos hermanos que se deben distinguir de los vínculos hijos, y eso requiere un nuevo campo en la declaración del árbol sintáctico. §