capítulo 3. Análisis sintáctico.

Marquez Garcia Jonathan Gael

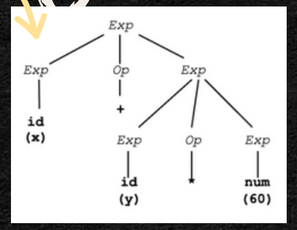
Análisis Sintáctico

indica la secuencia de componentes léxicos

Gramáticas libres de contexto

especificación para la estructura sintáctica de un lenguaje de programación

$Exp \rightarrow Exp\ Op\ Exp$
$Exp \rightarrow \mathbf{id}$
$Exp \rightarrow \mathbf{num}$
$Op \rightarrow + \mid - \mid * \mid /$



Por ejemplo, en la gramática 3.1 se tiene:

Terminales: { **id**, **num**, +, -, *, / }

No terminales: { *Exp*, *Op* }

Producciones: { $Exp \rightarrow Exp\ Op\ Exp$; $Exp \rightarrow \mathbf{id}$; $Exp \rightarrow \mathbf{num}$; $Op \rightarrow + \mid - \mid * \mid /$ }

Símbolo inicial: { *Exp* }

*Una derivación para la expresión: x + y * 60*

1)	$Exp \Rightarrow Exp\ Op\ Exp$	$[Exp \rightarrow Exp\ Op\ Exp]$
2)	$\Rightarrow Exp\ Op\ Exp\ Op\ Exp$	$[Exp \rightarrow Exp\ Op\ Exp]$
3)	$\Rightarrow Exp\ Op\ Exp\ Op\ \mathbf{num}$	$[Exp \rightarrow \mathbf{num}]$
4)	$\Rightarrow Exp\ Op\ Exp\ *\ \mathbf{num}$	$[Op \rightarrow *]$
5)	$\Rightarrow Exp\ Op\ \mathbf{id}\ *\ \mathbf{num}$	$[Exp \rightarrow \mathbf{id}]$
6)	$\Rightarrow Exp\ +\ \mathbf{id}\ *\ \mathbf{num}$	$[Op \rightarrow +]$
7)	$\Rightarrow \mathbf{id}\ +\ \mathbf{id}\ *\ \mathbf{num}$	$[Exp \rightarrow \mathbf{id}]$

Árboles de análisis sintáctico

muestra de forma gráfica la manera en que el símbolo inicial de una gramática deriva a una secuencia de componentes.

En un ejemplo, la derivación:

$Exp \Rightarrow Exp\ Op\ Exp$
$\Rightarrow \mathbf{id}\ Op\ Exp$
$\Rightarrow \mathbf{id} + Exp$
$\Rightarrow \mathbf{id} + \mathbf{id}$

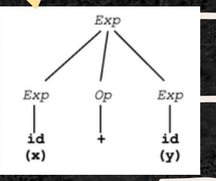
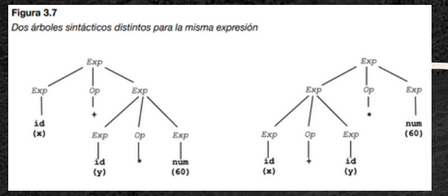
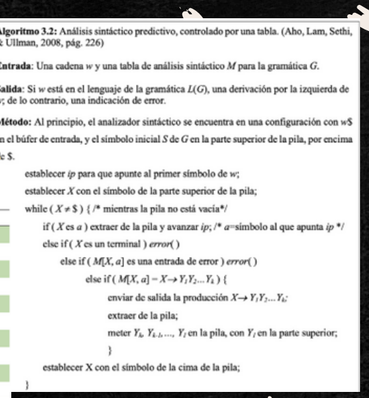


Figura 3.7
Dos árboles sintácticos distintos para la misma expresión



Análisis sintáctico predictivo no recursivo

Se puede construir a partir de una cadena w y una tabla de análisis sintáctico M para la gramática G, mediante el uso de una pila.



Algoritmo 3.2: Análisis sintáctico predictivo, controlado por una tabla. (Aho, Lam, Sethi, & Ullman, 2008, pág. 226)

Entrada: Una cadena *w* y una tabla de análisis sintáctico *M* para la gramática *G*.

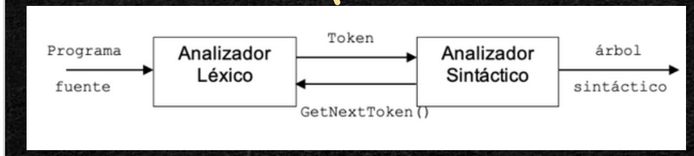
Salida: Si *w* está en el lenguaje de la gramática *L(G)*, una derivación por la izquierda de *w*, de lo contrario, una indicación de error.

Método: Al principio, el analizador sintáctico se encuentra en una configuración con *w*$ en el búfer de entrada, y el símbolo inicial *S* de *G* en la parte superior de la pila, por encima de $.

establecer *ip* para que apunte al primer símbolo de *w*;
establecer *X* con el símbolo de la parte superior de la pila;
while (*X* ≠ $) { /* mientras la pila no está vacía*/
  if (*X* es *a*) extraer de la pila y avanzar *ip*; /* *a*=símbolo al que apunta *ip* */
  else if (*X* es un terminal) *error*( )
    else if (*M*[*X*, *a*] es una entrada de error) *error*( )
      else if (*M*[*X*, *a*] = *X*→ *Y*1*Y*2...*Yk*) {
        enviar de salida la producción *X*→ *Y*1*Y*2...*Yk*;
        extraer de la pila;
        meter *Yk*, *Yk*-1, ..., *Y*1 en la pila, con *Y*1 en la parte superior;
        }
  establecer X con el símbolo de la cima de la pila;
}

Figura 3.13
*Movimientos que realiza un analizador sintáctico predictivo con la entrada id * id + id*

Pila	Entrada	Acción
$E	id*id+id$	E → TE'
$E' T	id*id+id$	T → FT'
$E' T' F	id*id+id$	F → id
$E' T' id	id*id+id$	concuerda(id)
$E' T'	*id+id$	T' → *FT'
$E' T' F*	*id+id$	concuerda(*)
$E' T' F	id+id$	F → id
$E' T' id	id+id$	concuerda(id)
$E' T'	+id$	T' → ε
$E'	+id$	E' → +TE'
$E' T +	+id$	concuerda(+)
$E' T	id$	T → FT'
$E' T' F	id$	F → id
$E' T' id	id$	concuerda(id)
$E' T'	$	T' → ε
$E'	$	E' → ε
$	$	aceptar( )



Gramáticas LL(1)

Otra manera de implementar analizadores sintácticos descendentes en por medio de las gramáticas LL(1).

Algoritmo 3.1: Construcción de una tabla de análisis sintáctico predictivo (Aho, Lam, Sethi, & Ullman, 2008, p. 224).

Entrada: La gramática *G*.

Salida: La tabla de análisis sintáctico *M*.

Método: Para cada producción $A \rightarrow \alpha$ de la gramática, hacer lo siguiente:

1. Para cada terminal *a* en Primero(*A*), agregar $A \rightarrow \alpha$ a *M*[*A*, *a*].
2. Si ε está en Primero(α), entonces para cada terminal *b* en Siguiente(*A*), se agrega $A \rightarrow \alpha$ a *M*[*A*, *b*]. Si ε está en Primero(α) y $ se encuentra en Siguiente(*A*), se agrega $A \rightarrow \alpha$ a *M*[*A*,$] también.

Tabla 3.1
Tabla de análisis sintáctico para la gramática 3.3

No terminal	Símbolo de entrada					
	id	+	*	(	)	$
E	E → TE'			E → TE'		
E'		E' → +TE'			E' → ε	E' → ε
T	T → FT'			T → FT'		
T'		T' → ε	T' → *FT'		T' → ε	T' → ε
F	F → id			F → (E)		

Conjuntos Primero y Siguiente

Es auxiliada por dos funciones, Primero y Siguiente, asociadas con una gramática G.

1. Si *X* es un terminal, entonces Primero(*X*) = {*X*}.

2. Si *X* es un no terminal y $X \rightarrow Y_1 Y_2 \ldots Y_k$ es una producción para cierta $k \geq 1$, entonces se coloca *a* en Primero(*X*) si para cierta *i*, *a* está en Primero(Y_i), y ε está en todas las funciones Primero(Y_1), ..., Primero(Y_{i-1}); es decir, $Y_1 \ldots Y_{i-1} \Rightarrow^* \varepsilon$. Si ε está en Primero($Y_j$) para todas las *j* = 1, 2, ..., *k*, entonces se agrega ε a Primero(*X*). Por ejemplo, todo lo que hay en Primero(Y_1) se encuentra sin duda en Primero(*X*). Si Y_1 *no* deriva a ε, entonces no se agrega nada más a Primero(*X*), pero si $Y_1 \Rightarrow^* \varepsilon$, entonces se agrega Primero(Y_2), y así sucesivamente.

3. Si $X \rightarrow \varepsilon$ es una producción, entonces se agrega ε a Primero(*X*).

Para calcular Siguiente(*A*) para todos los no terminales *A*,se aplican las siguientes reglas hasta que no pueda agregarse nada a cualquier conjunto Siguiente.

1. Colocar $ en Siguiente(*S*), en donde *S* es el símbolo inicial y $ es el delimitador derecho de la entrada.

2. Si hay una producción $A \rightarrow \alpha B \beta$, entonces todo lo que hay en Primero(β) excepto ε está en Siguiente(*B*).

3. Si hay una producción $A \rightarrow \alpha B$, o una producción $A \rightarrow \alpha B \beta$, en donde Primero(β) contiene a ε,entonces todo lo que hay en Siguiente(*A*) está en Siguiente(*B*).

Análisis sintáctico descendente recursivo

definir un procedimiento para cada no terminal de la gramática.

Figura 3.12
Código en Java para la regla gramatical F → (E) | id en un analizador sintáctico descendente

```
1   public static int F()
2   {
3       int error=0;
4       complex = componentes[i];
5       i++;
6       if (complex=='(') // F -> (E)
7       {
8           error = E();
9           if (error==0) //si no hay error; continua verificando
10          {
11              complex = componentes[i];
12              i++;
13              if (complex!=')')
14              {
15                  error = 1;    //No se encontró ')'
16                  i--; /*manejar retroceso*/
17              }
18          }
19      }
20      else
21      {
22          if (complex!=id) //si no es id
23          {
24              error = 2;
25              i--; /*manejar retroceso*/
26          }
27      }
28      return error;
```

Análisis sintáctico descendente

analiza una cadena de componentes léxicos de entrada mediante e los pasos para una derivación por la izquierda

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \varepsilon$
$F \rightarrow (E) \mid \mathbf{id}$ (3.3)

Figura 3.10
*Análisis sintáctico descendente para id*id+id*