Compiladores

Análisis Sintáctico Descendente

Adrian Ulises Mercado Martínez

Facultad de Ingeniería, UNAM

5 de septiembre de 2013

Índice

Sección 1 | Introducción

Análisis Sintáctico Descendente

Análisis Sintáctico Descendente

- Se basa en derivaciones por la izquierda.
- Construye el árbol sintáctico de la raíz hacia las hojas.
- Existen dos tipos de análisis descendente
  - Recursivo
  - LL(k)

Ejemplo:Sea G

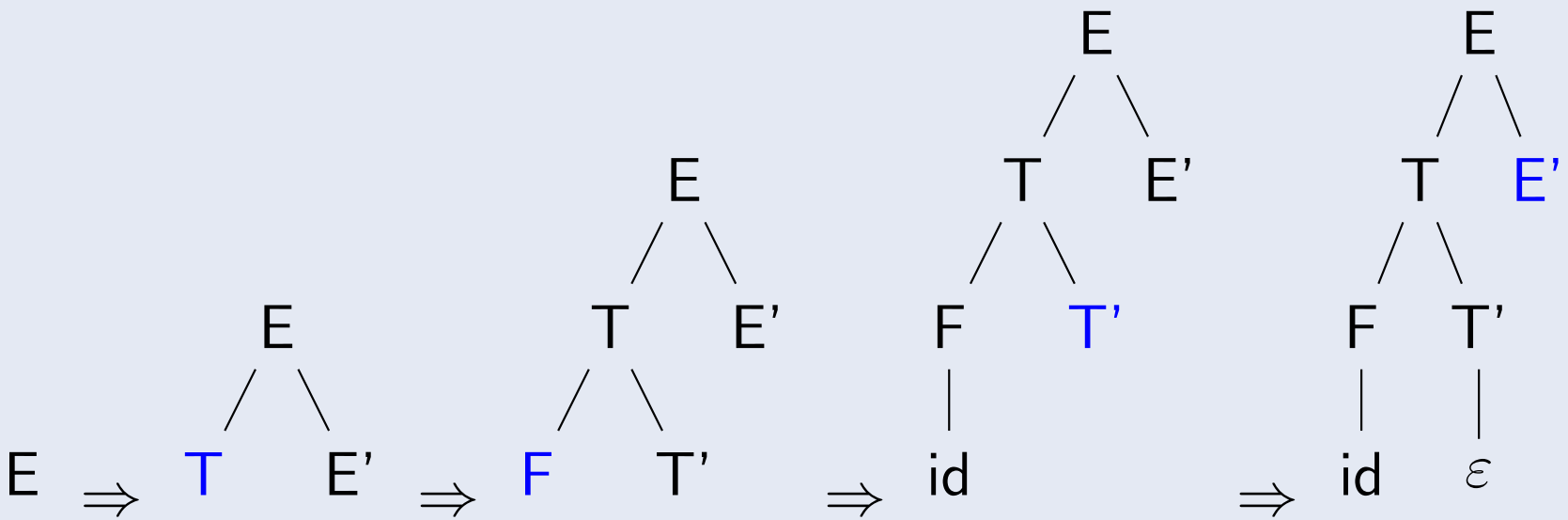
$$E \rightarrow T\ E' \mid T$$
$$E' \rightarrow +\ T\ E' \mid \varepsilon$$
$$T \rightarrow F\ T'$$
$$T' \rightarrow *\ F\ T' \mid \varepsilon$$
$$F \rightarrow (E) \mid id$$

$\omega =$ **id + id * id**

E⇒T E'⇒F T' E' ⇒id T' E'⇒id E'⇒id + T E'⇒id + F T' E'⇒id + id T' E'⇒ id +id * F T' E' ⇒id + id * id T' E' ⇒id + id * id E' ⇒ id + id * id

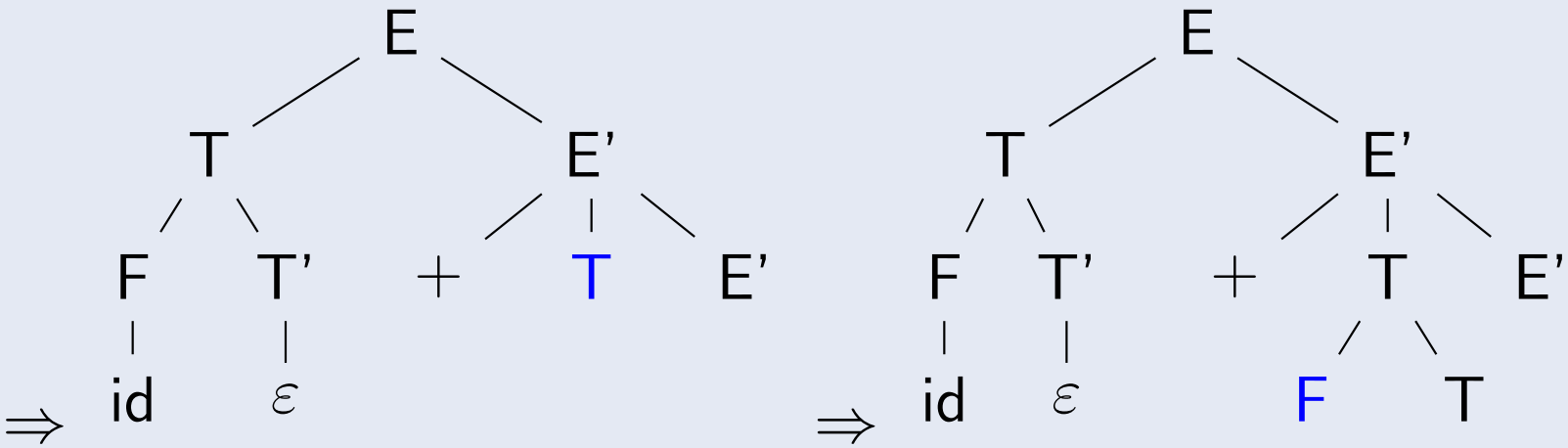
Análisis Sintáctico Descendente

Derivación por la izquierda



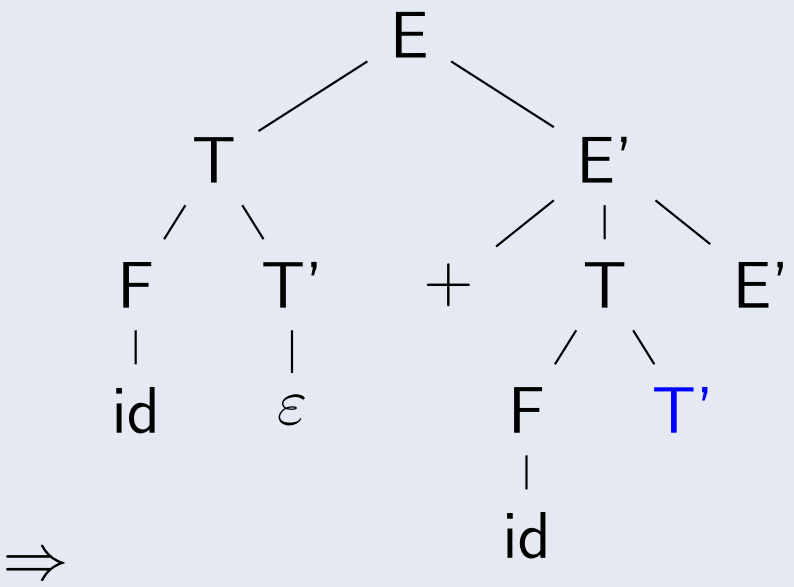
Análisis Sintáctico Descendente

Derivación por la izquierda.

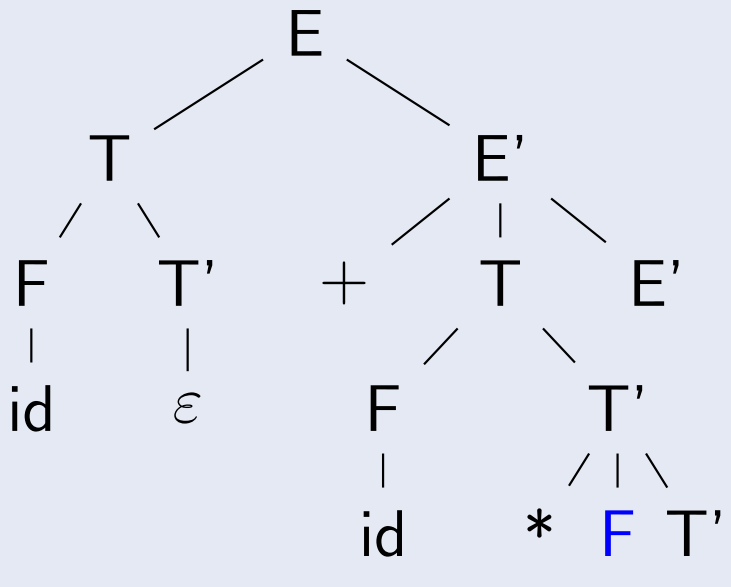


Análisis Sintáctico Descendente

Análisis Sintáctico Descendente



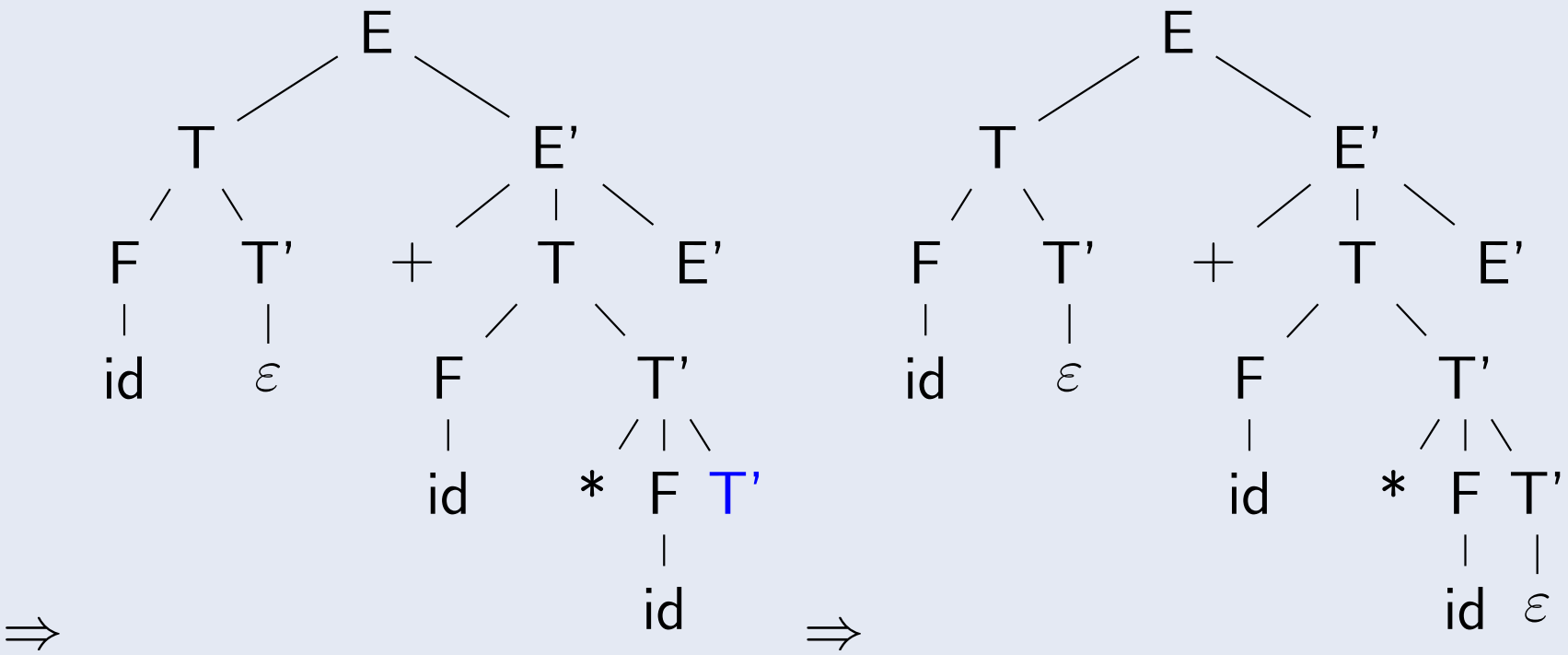
$\Rightarrow$



$\Rightarrow$

Análisis Sintáctico Descendente

Derivación por la izquierda.



Sección 2 | Recursividad

Recursividad

En general una gramática es recursiva si cumple con la siguiente definición.

Definición (Recursividad)

Decimos que una gramática G es recursiva si existe alguna derivación tal que $A \overset{+}{\Rightarrow} \alpha A \beta$ para α y β cualesquiera.

Análisis Sintáctico Descendente

Definition (Recursividad por la derecha)

Decimos que una gramática es recursiva por la derecha si tiene un no terminal A tal que haya una derivación $A \stackrel{+}{\Rightarrow} \alpha\ A$ para cierta cadena α.

La recursividad derecha se representa por:

$$A \rightarrow \alpha\ A\ |\beta$$

Definition (Recursividad por la izquierda)

Decimos que una gramática es recursiva por la izquierda si tiene un no terminal A tal que haya una derivación $A \stackrel{+}{\Rightarrow} A\alpha$ para cierta cadena α.

La recursividad izquierda se representa por:

$$A \rightarrow A\alpha\ |\beta$$

Análisis Sintáctico Descendente

Eliminación de la recursividad por la izquierda

Eliminación de la recursividad por la izquierda

- $A \rightarrow A\alpha|\beta$
- sustituimos por
- $A \rightarrow \beta\ A'$
- $A' \rightarrow \alpha\ A'|\varepsilon$

Eliminación de la recursividad por la izquierda general

- $A \rightarrow A\alpha_1|...|A\alpha_n|\beta_1|...|\beta_n$
- sustituimos por
- $A \rightarrow \beta_1\ A'|...|\beta_n A'$
- $A' \rightarrow \alpha_1\ A'|...|\alpha_n A'|\varepsilon$

Análisis Sintáctico Descendente I

Algoritmo

Algoritmo 1: Algoritmo para eliminar la Recursividad

Data: Una gramática G propia.

Result: Una gramática equivalente sin recursividad por la izquierda.

1 Ordenar los No terminales de cierta forma: $A_1, A_2, ..., A_n$;

2 **foreach** *(i=1 hasta n)* **do**

3 | **foreach** *(j=1 hasta i-1)* **do**

4 | | sustituir cada producción de la forma $A_i \rightarrow A_j\gamma$ por las producciones $A_i \rightarrow \delta_1\gamma|\delta_2\gamma|...|\delta_k\gamma$ donde $A_j \rightarrow \delta_1|\delta_2|...|\delta_n$;

5 | eliminar la recursividad inmediata;

Sección 3 | Factorización por la izquierda.

Análisis Sintáctico Descendente

Factorización por la izquierda.

Definition

Si en una gramática G algún no terminal A tiene reglas de producción $A \rightarrow \alpha\beta_1|\alpha\beta_2|...|\alpha\beta_n|\gamma$ decimos que las producciones tienen un factor común α, ese factor es el prefijo común más largo.

Factorización por la izquierda

- Si $A \rightarrow \alpha\beta_1|\alpha\beta_2|...|\alpha\beta_n|\gamma$
- sustituimos por
- $A \rightarrow \alpha A'|\gamma$
- $A' \rightarrow \beta_1|\beta_2|...|\beta_n$

Algoritmo

Algoritmo 2: Algoritmo para factorizar una Gramática

Data: Una gramática G

Result: Una gramática equivalente factorizada por la izquierda

1 **foreach** *(No terminal en G)* **do**

2 Buscar el prefijo más largo α que sea común para dos o más alternativas;

3 **if** $(\alpha \neq \varepsilon)$ **then**

4 sustituir $A \rightarrow \alpha\beta_1|\alpha\beta_2|...|\alpha\beta_n|\gamma$ por $A \rightarrow \alpha A'|\gamma$ y $A' \rightarrow \beta_1|\beta_2|...|\beta_n$;

Sección 4 Analizador Sintáctico Descendente Recursivo

Análisis Sintáctico Descendente Recursivo

Se basa en la idea de construir una función por cada símbolo *no terminal* de la gramática. Dentro de la función se revisan cada una de las producciones.

Para un no terminal A

```
void A(){
   Elegir una producción de A tal que A::= X1 X2 ... Xk
   for( i= 1 a k){
          if(Xi es un no terminal)
              llamar al procedimiento Xi();
          else if(Xi es igual a un símbolo de entrada
                  actual)
              avanzar la entrada al símbolo siguiente;
          else
             error();
     }
```

Ejemplo:

- S $\rightarrow$ if E then S else S
- S $\rightarrow$ begin S L
- S $\rightarrow$ print E
- L $\rightarrow$ end
- L $\rightarrow$: S L
- E $\rightarrow$ num = num

Ejemplo

```
final int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5,
                    PRINT=6,SEMI=7, NUM=8, EQ=9;
int tok = getToken();
void avanzar(){ tok = getToken()};
void consumir(int i){if tok==i) avanzar(); else error();}
void S(){
    switch(tok){
       case IF: consumir(IF); E(); consumir(THEN); S();
                consumir(ELSE); S(); break;
       case BEGIN: consumir(BEGIN); S(); L(); break;
       case PRINT: consumir(PRINT); E(); break;
       default: error();
    }
}
```

Ejemplo

```
void L(){
    switch(tok){
        case END: consumir(END); break;
        case SEMI: consumir(SEMI); S(); L(); break;
        default: error();
    }
}
void E() consumir(NUM); consumir(EQ); consumir(NUM);}
```

Sección 5 Análisis Sintáctico Descendente Predictivo

Análisis Sintáctico Descendente

Definition (FIRST)

El conjunto FIRST(α), en donde α es cualquier cadena de símbolos gramaticales, como el conjunto de terminales con los que puede empezar las cadenas derivadas de α. Si $\alpha \stackrel{*}{\Rightarrow} \varepsilon$ entonces ε también se encuentra en FIRST(α)

Definition (FOLLOW)

El conjunto FOLLOW(A), en donde A es un no terminal, como el conjunto de terminales que pueden aparecer de inmediato a la derecha de A en cierta forma de frase; es decir, el conjunto de terminales a, de tal forma que exista una derivación de la forma $S \stackrel{*}{\Rightarrow} \alpha A a \beta$, para algunas α y β

Algoritmo 3: Algoritmo para calculo de FIRST y FOLLOW

```
1  Iniciar FIRST, FOLLOW en vacío y anulable en "no";
2  repeat
3      foreach (producción X → Y1 Y2 ... Yk) do
4          if (Y1 Y2 ... Yk son todos anulables o ε) then
5              anulable[X]= "sí";
6          if (X es el símbolo inicial) then
7              agregar a FOLLOW(X) a ▽;
8          foreach (i= 0 hasta k y j=i+1 hasta k) do
9              if (Y1 ... Yi−1 son todos anulables o i=1) then
10                 FIRST[X] = FIRST[X] ⋃ FIRST[Yi];
11             if (Yi+1 .... Yk son todos anulables o k=i) then
12                 FOLLOW[Yi]=FOLLOW[Yi] ⋃ FOLLOW[X];
13             if (Yi+1 .... Yj− son todos anulables o i+1=j) then
14                 FOLLOW[Yi]=FOLLOW[Yi] ⋃ FOLLOW[Yj];
15 until FIRST, FOLLOW y anulable no cambien;
```

Ejemplo

Sea la gramática G

$$S \rightarrow ABCD$$
$$A \rightarrow aAz|\varepsilon$$
$$B \rightarrow bBAy|Ey|\varepsilon$$
$$C \rightarrow cCAx|\varepsilon$$
$$D \rightarrow dDBAw|\varepsilon$$
$$E \rightarrow eECe|\varepsilon$$

Ejemplo

	anulable	FIRST	FOLLOW
S	no		
A	no		
B	no		
C	no		
D	no		
E	no		

Ejemplo

Vamos a calcular los anulables primero

	anulable	FIRST	FOLLOW
S	sí		
A	sí		
B	sí		
C	sí		
D	sí		
E	sí		

Ejemplo

	anulable	FIRST	FOLL
S	sí	FRIST(A)∪ FIRST(B)∪ FIRST(C)∪ FIRST(D)	
A	sí	a ε	
B	sí	b FIRST(E) ε	
C	sí	c ε	
D	sí	d ε	
E	sí	e ε	

Ejemplo

	anulable	FIRST	FOLLOW
S	sí	a b e c d ε	
A	sí	a ε	
B	sí	b e ε	
C	sí	c ε	
D	sí	d ε	
E	sí	e ε	

Ejemplo

	anulable	FIRST	FOLLOW
S	sí	a b e c d ε	$\bigtriangledown$
A	sí	a ε	b e c d $\bigtriangledown$ y x w
B	sí	b e ε	c d $\bigtriangledown$ a y w
C	sí	c ε	d $\bigtriangledown$ e a x
D	sí	d ε	$\bigtriangledown$ b e a w
E	sí	e ε	c e y

Algoritmo 4: Construcción de la Tabla de Análisis Sintáctio LL(1)

Data: Una Gramática G

Result: La tabla de Análisis Sintáctico LL(1) M

1 **foreach** *(producción* $A \rightarrow \alpha$ *en la gramática G)* **do**
2 **foreach** *(terminal a en FIRST*(α) *)* **do**
3 M[A, a]= $A \rightarrow \alpha$;
4 **if** *(* $\varepsilon \in$ *FIRST*(α)*)* **then**
5 M[A, b]= $A \rightarrow \alpha$, donde $b \in$ FOLLOW(A);
6 **if** *(*$\bigtriangledown \in$ *FOLLOW(A))* **then**
7 M[$A, \bigtriangledown$]= $A \rightarrow \alpha$;

Ejemplo:

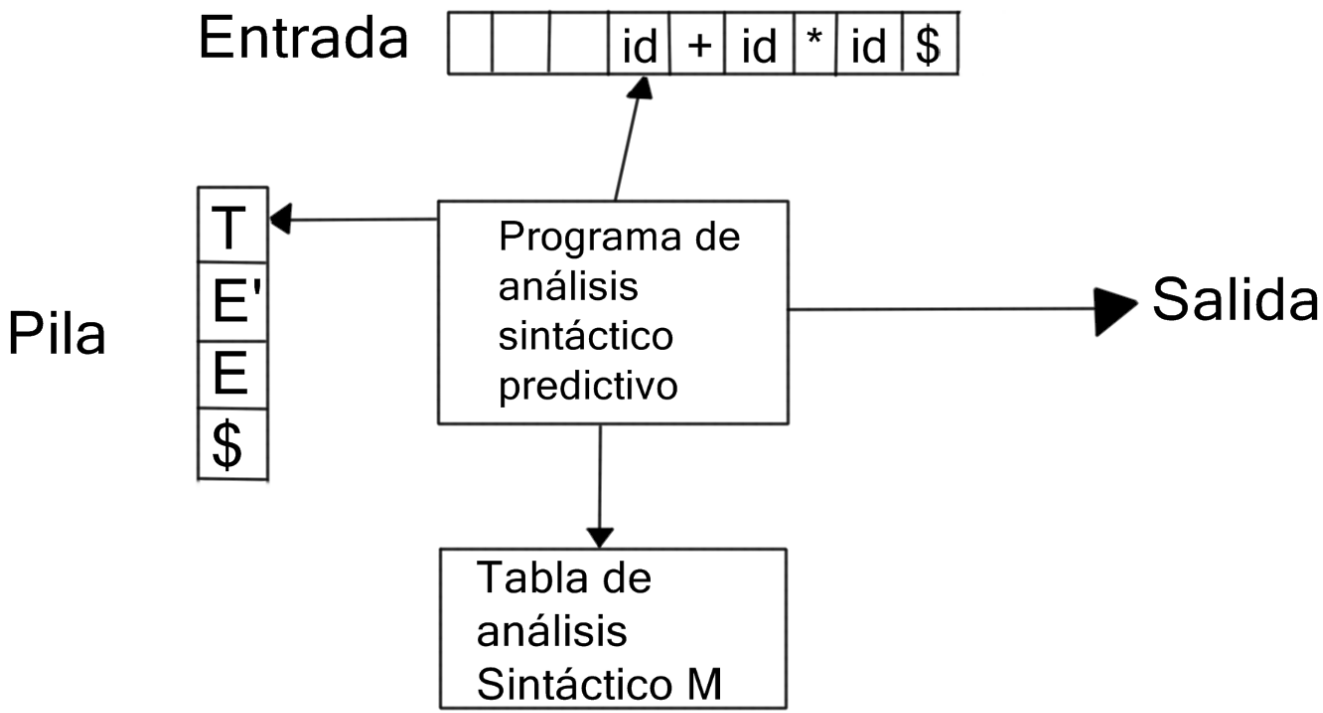
Sea G:

$$E \rightarrow TE'$$
$$E' \rightarrow +TE'|\varepsilon$$
$$T \rightarrow FT'$$
$$T' \rightarrow *FT'|\varepsilon$$
$$F \rightarrow (E)|id$$

No Terminal	id	+	*	(	)	▽
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		

Sección 6 | Análisis Sintáctico LL(1)

Análisis Sintáctico LL(1)



Algoritmo 5: Análisis Sintáctio LL(1)

Data: Una cadena ω y una tabla de análisis sintáctico M para la gramática G

Result: Si $\omega \in$ L(G), una derivación por la izquierda de ω; en caso contrario, una indicación de error.

```
1  ip es el apuntador al primer símbolo de ω;
2  X es la cima de la pila;
3  Hacer ω = ωV;
4  Meter en la pila a V S, S es el símbolo inicial;
5  while (X ≠ V ) do
6      if (X = ip) then
7          pop de la pila y avanzar ip;
8      else if (X es un terminal diferente de ip ) then
9          error();
10     else if (M[X,ip]= error) then
11         error();
12     else if (M[X,ip]= X → Y_i Y_2 ... Y_k) then
13         pop();
14         push(Y_k .... Y_2 Y_1) de modo que Y_1 este en la cima de la pila;
15 if (X = V y ip =V) then
16     aceptar;
```

PILA	ENTRADA	ACCIÓN
E $\bigtriangledown$	id + id * id $\bigtriangledown$	
TE'$\bigtriangledown$	id+ id *id $\bigtriangledown$	$E \rightarrow TE'$
FT'E'$\bigtriangledown$	id + id * id $\bigtriangledown$	$T \rightarrow FT'$
id T' E' $\bigtriangledown$	id + id * id $\bigtriangledown$	$F \rightarrow id$
T' E' $\bigtriangledown$	+ id * id $\bigtriangledown$	pop (id) y avanzar
E' $\bigtriangledown$	+ id * id $\bigtriangledown$	$T' \rightarrow \varepsilon$
+ T E' $\bigtriangledown$	+ id * id $\bigtriangledown$	$E' \rightarrow +TE'$
T E' $\bigtriangledown$	id * id $\bigtriangledown$	pop(+) y avanzar
F T' E' $\bigtriangledown$	id * id	$T \rightarrow FT'$
id T' E' $\bigtriangledown$	id * id $\bigtriangledown$	$F \rightarrow id$
T' E' $\bigtriangledown$	* id $\bigtriangledown$	pop(id) y avanzar
* F T' E' $\bigtriangledown$	* id $\bigtriangledown$	$T' \rightarrow *FT'$
F T' E' $\bigtriangledown$	id $\bigtriangledown$	pop(*) y avanzar
id T' E' $\bigtriangledown$	id $\bigtriangledown$	$F \rightarrow id$
T' E' $\bigtriangledown$	$\bigtriangledown$	pop(id) y avanzar
E' $\bigtriangledown$	$\bigtriangledown$	$T' \rightarrow \varepsilon$
$\bigtriangledown$	$\bigtriangledown$	$E' \rightarrow \varepsilon$
$\bigtriangledown$	$\bigtriangledown$	aceptar