

Sintaxis y Semántica de los lenguajes

**Métodos descendentes de análisis
sintáctico**

2020

Facultad Regional Delta,
Universidad Tecnológica Nacional

Métodos de análisis sintáctico

Métodos descendentes de análisis sintáctico

Introducción.

Analizador sintáctico descendente con retroceso.

Función Primeros.

Función Siguientes.

Símbolos directrices.

Analizador sintáctico descendente predictivo.

Analizador sintáctico descendente predictivo con tabla.

Gramáticas LL(1).

Gramática ampliada.

No recursividad de las gramáticas LL(1) (T)

Eliminación de la recursividad izquierda.

Factorización.

Gramáticas LL(k).

Problemas no decidibles y decidibles con gramáticas LL.

Introducción.

Consideremos la GLC $G = \langle V_N, V_T, P, S \rangle$ con

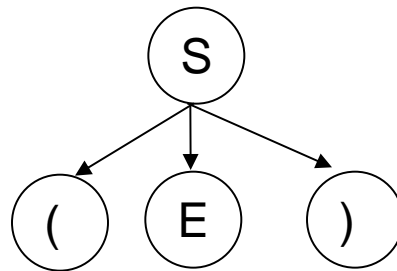
$P = \{ S \rightarrow (E),$

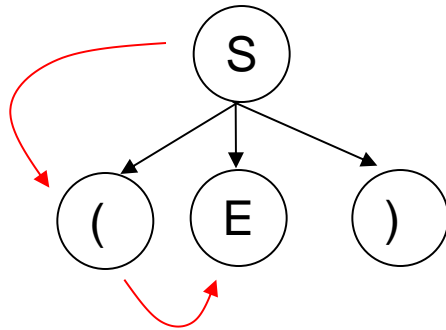
$E \rightarrow E * E,$

$E \rightarrow i \},$

la cadena $w = (i)$ y un árbol de derivación incompleto formado exclusivamente por el nodo raíz \textcircled{S} .

Al aplicar la única derivación posible ($S \rightarrow (E)$) obtenemos el siguiente árbol





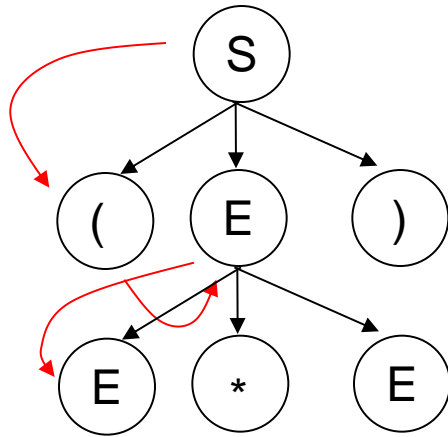
$w = (i)$

Como el primer símbolo de w coincide con la etiqueta del nodo que está más a la izquierda “(”, el puntero de w se desplaza un lugar a la derecha.

Ahora apunta a “i” y además nos desplazamos a la próxima hoja del árbol.

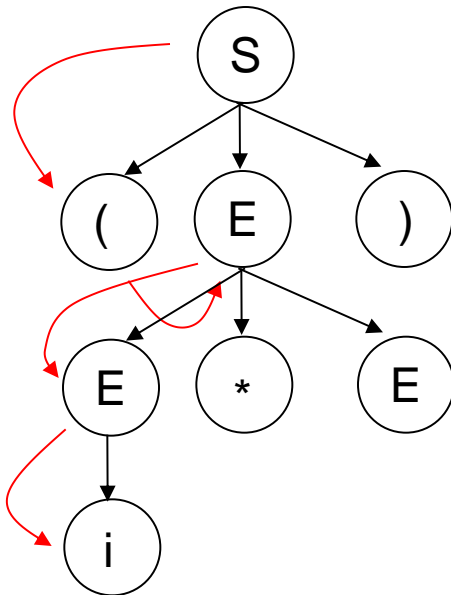
Como esta hoja está etiquetada con un no terminal E debemos expandirlo.

Como E tiene asociada más de una producción, elegimos al azar una de ellas. Consideremos $E \rightarrow E * E$, así el árbol nos queda



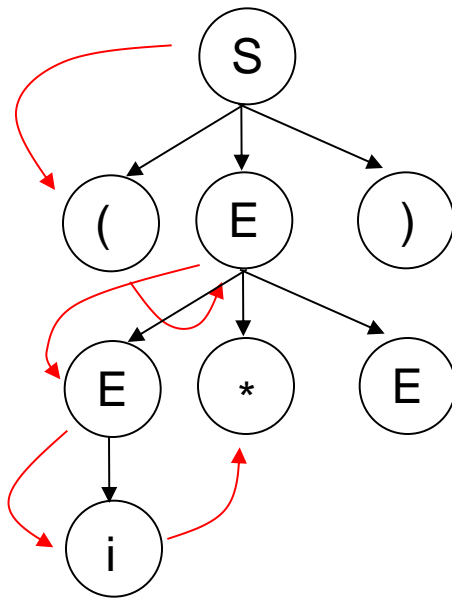
$w = (i)$
 $\uparrow \uparrow$

Nuevamente quedamos apuntando a un nodo etiquetado con un no terminal y debemos expandirlo. Elegimos al azar una producción. Consideremos $E \rightarrow i$.



$w = (i)$
 $\uparrow \uparrow \uparrow$

Como el símbolo apuntado en la cadena de entrada coincide con la etiqueta del nodo en el árbol, el puntero en w se desplaza a la derecha y visitamos el próximo nodo en el árbol.



$$w = (i)$$

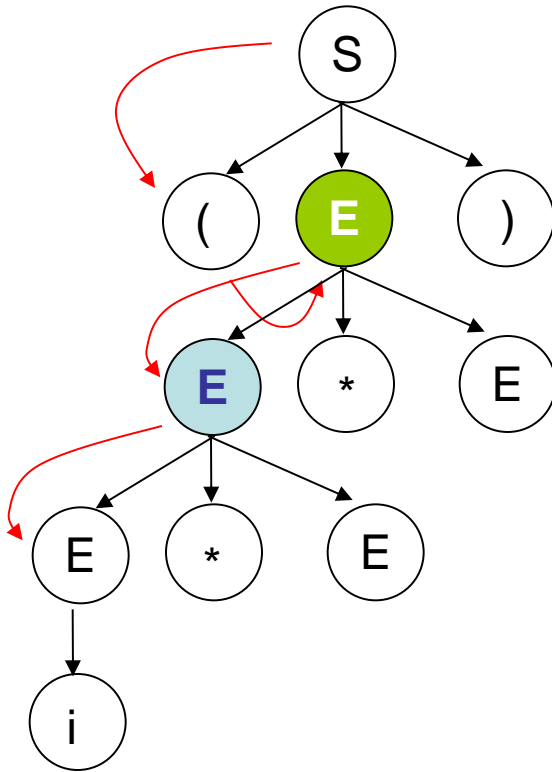
Sin embargo, **no hay coincidencia** entre la etiqueta del nodo “*” y el símbolo apuntado en w .

Estamos en una situación de **error**.

El error surgió de **haber elegido mal una expansión** o quizás, de haber considerado una cadena que **no puede ser generada por la gramática**.



Asumamos que (tal como ocurre) la cadena sí puede ser generada por la gramática, luego habrá sido una expansión mal elegida.

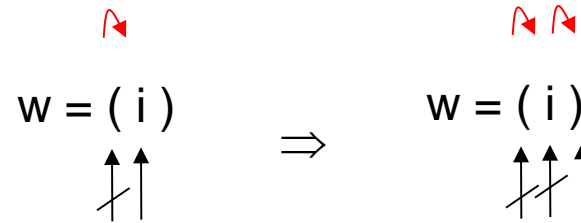
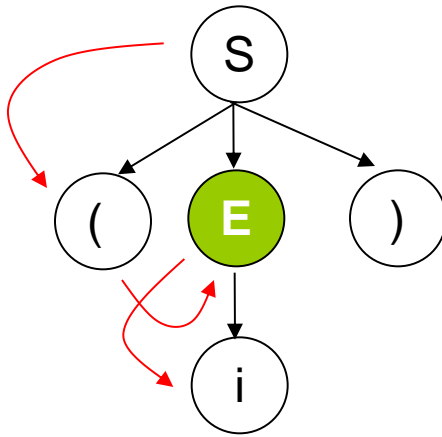
Retrocedemos hasta la última expansión y procedemos con una producción alternativa (si la hubiere). En nuestro caso tendremos



$$w = (i)$$

Y luego expandemos nuevamente E usando $E \rightarrow i$. Fijarse que a esta altura, ya tenemos 4 terminales en el árbol y si intentamos expandir el último no terminal con la alternativa $E \rightarrow E * E$, empeoraríamos la situación.

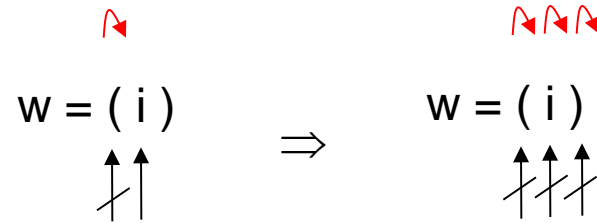
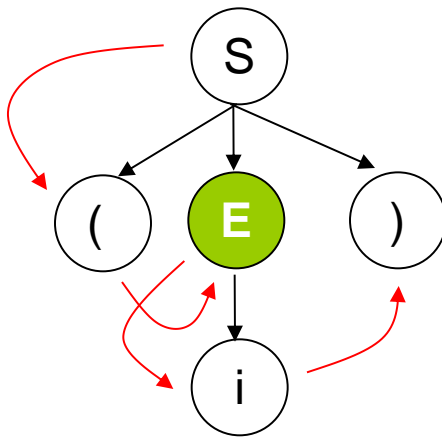
Luego, el error no estuvo en cómo expandimos  sino que deberíamos ir hacia la expansión anterior  .



Fijarse que si retrocedemos hacía expansiones previas que pudieron estar mal realizadas, también debemos hacer retroceder el puntero en w de acuerdo a cómo estaba cuando se hizo la expansión que vamos a revisar.

Para **E** antes habíamos elegido $E \rightarrow E * E$ y por lo tanto ahora elegiremos la otra $E \rightarrow i$.

El símbolo que etiqueta al nuevo nodo es igual al apuntado en w , por lo tanto desplazamos el puntero en w y visitamos el próximo nodo en el árbol.



Nuevamente hay coincidencia entre la etiqueta del nodo en el árbol y el símbolo apuntado en w .

Pero en este caso llegamos a la siguiente situación:

hemos recorrido todos los nodos del árbol, y

hemos agotado toda la cadena w ,

y por lo tanto **la cadena pertenece a $L(G)$.**

Comentarios del método:

- Se requiere retroceso (backtracking)
- La cadena se analiza de izquierda a derecha
- Siempre se elige la derivación mas a la izquierda.

Este método se denomina **análisis sintáctico descendente recursivo** o **análisis sintáctico descendente con retroceso**.

Consideremos ahora otro ejemplo:

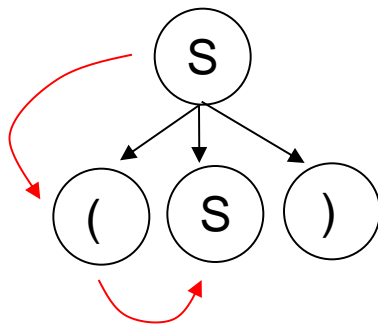
Sea la GLC $G = \langle V_N, V_T, P, S \rangle$ con

$P = \{ S \rightarrow (S),$
 $S \rightarrow a \},$

la cadena $w = (a)$ y un árbol de derivación incompleto formado

exclusivamente por el nodo raíz \textcircled{S} .

Expandimos S y llegamos a



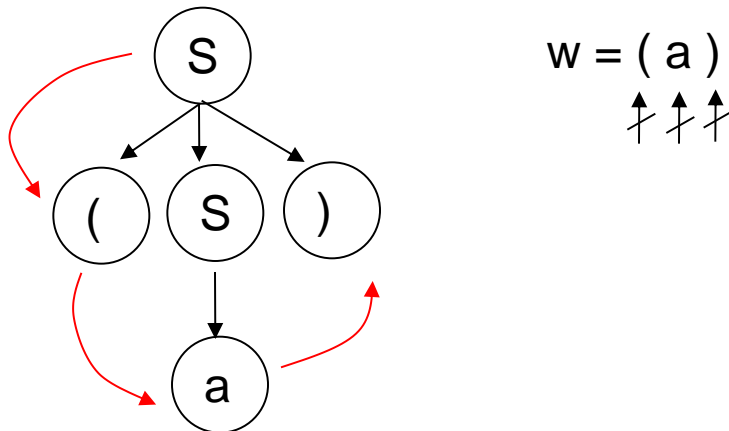
$w = (a)$
 $\uparrow \uparrow$

Ahora tenemos que expandir nuevamente S , pero nos detendremos en analizar las dos posibilidades.

Si expandimos con $S \rightarrow (S)$ el próximo nodo que visitaremos en el árbol es el etiquetado con "(" pero si lo hacemos con $S \rightarrow a$, la primera visita coincidirá con el símbolo apuntado en w y podremos seguir adelante.

Esto es, la idea es **seleccionar la producción a usar en la expansión de modo tal que la próxima comparación con el símbolo apuntado en w coincida**.

Finalmente, recorreremos el próximo nodo etiquetado con ")" que también coincidirá y concluiremos con que la cadena pertenece a $L(G)$.



Con esta idea en mente definiremos la función Primeros.

Definición (Primeros)

Primeros: $V^* \rightarrow P(V_T)$, con $V = (V_T \cup V_N)$,

$\text{Primeros}(\gamma) = \{ t \in V_T / \gamma \rightarrow^* \alpha, \alpha = t \alpha' \}$.

En el ejemplo recién visto,

para la producción $S \rightarrow (S)$, $\text{Primeros}(" (S) ") = \{ (\}$ y

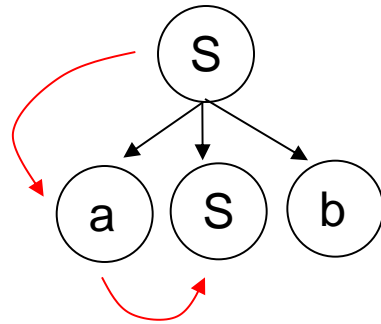
para la producción $S \rightarrow a$, $\text{Primeros}(" a ") = \{ \mathbf{a} \}$.

Consideremos ahora el siguiente ejemplo.

Sea la GLC $G = \langle V_N, V_T, P, S \rangle$ con

$P = \{ S \rightarrow aSb, S \rightarrow \lambda \}$ y $w = ab$.

Habiendo llegado al siguiente árbol de derivación



quedamos apuntando en la cadena w a la “b” ($w = ab$).



Ahora si aplico la función Primeros a las partes derechas de las producciones tengo: para $S \rightarrow aSb$, $\text{Primeros}(aSb) = \{ a \}$ y para $S \rightarrow \lambda$,

¿cuál es $\text{Primeros}(\lambda)$? $\text{Primeros}(\lambda) = \emptyset$

Al ser “S” anulable, lo que cuenta es saber cuál es el próximo símbolo que le seguirá a S en cualquier forma sentencial para saber si, en el caso que S se anule, el símbolo que le sigue coincidirá con el apuntado en la cadena de entrada.

Sea la siguiente forma sentencial $A\alpha$, con $A \in V_N$ y $\alpha \in (V_N \cup V_T)^*$.

Si A puede anularse, los siguientes de A pueden calcularse como los primeros de α .

Definición (Siguietes):

Siguietes: $V_N \rightarrow P(V_T)$,

$\text{Siguietes}(N) = \{ t \in V_T / S \rightarrow^* \dots N t \dots \}$.

Con la definición de Siguietes y Primeros estamos en condiciones de definir a los Símbolos Directrices de una producción.

Definición (Símbolos directrices ó SD):

Sea la GLC $G = \langle VN, VT, P, S \rangle$,

SD: $P \rightarrow P(VT)$,

$$SD(N \rightarrow \beta) = \begin{cases} \text{Primeros}(\beta) & \text{si } \beta \text{ no es anulable} \\ \text{Primeros}(\beta) \cup \text{Siguietes}(N) & \text{si } \beta \text{ es anulable} \end{cases}$$

Implementación procedural de analizadores sintácticos descendentes con retroceso

Una forma de implementar un analizador descendente es mediante la definición de procedimientos para cada no terminal empezando por el símbolo distinguido.

Sea una GLC $G = \langle V_N, V_T, P, N_1 \rangle$, G sin reglas borradoras y reducida, con $V_N = \{N_1, N_2, \dots, N_m\}$, $m = |V_N|$ y

sea $N_i \in V_N$ con las siguientes producciones asociadas

$$N_i \rightarrow \alpha_1 / \alpha_2 / \dots / \alpha_k$$

y t es el símbolo apuntado en la cadena de entrada, definimos

Procedimiento P_{N_i} (p_w puntero a w)

$j \leftarrow 1$

Hacer

$\text{error} \leftarrow \text{Falso}$

 Procesar ($N_i \rightarrow \alpha_j$, p_w)

$j \leftarrow j + 1$

mientras $\text{error} \wedge (j \leq k)$

Sea $N \rightarrow \alpha$ una producción en P , y $\alpha = x_1 x_2 \dots x_n$

donde $x_j \in (V_N \cup V_T)$, $1 \leq j \leq n$, definimos

Procesar ($N \rightarrow \alpha$, p_w puntero a w)

Para $j : 1 \dots n$

según x_j

$x_j \in V_T$:

si $\wedge p_w == x_j$

avanzar p_w en w

sino

error \leftarrow Verdadero

Fin

$x_j \in V_N$:

$P_{x_j}(p_w)$

si error

Fin

Luego, si G no es recursiva izquierda

Algoritmo

error \leftarrow *falso*

t apunta al primer símbolo de w

PN1(t)

si no error \wedge fin_de_cadena(w, t)

 w pertenece a $L(G)$

sino

 w no pertenece a $L(G)$

Implementación procedural de analizadores sintácticos descendentes predictivo.

En el caso que se disponga de los Símbolos Directrices de las producciones de la gramática, el procedimiento para cada no terminal puede reescribirse como

Procedimiento P_{N_i} (p_w puntero a w)
Error $\leftarrow falso$
Según t
 $\wedge p_w \in SD(N_i \rightarrow \alpha_1)$ entonces Procesar ($N_i \rightarrow \alpha_1, p_w$)
 $\wedge p_w \in SD(N_i \rightarrow \alpha_2)$ entonces Procesar ($N_i \rightarrow \alpha_2, p_w$)
...
 $\wedge p_w \in SD(N_i \rightarrow \alpha_k)$ entonces Procesar ($N_i \rightarrow \alpha_k, p_w$)

Tanto el algoritmo principal como el procedimiento Procesar siguen iguales que para el caso de analizadores sintácticos con retroceso.

La **ventaja** de los **analizadores sintácticos descendentes predictivos** es que son determinísticos (no dependen del orden en que las producciones son consideradas cuando son utilizadas para una expansión).

Sin embargo, la **desventaja** es que sólo un conjunto propio de lenguajes libres de contexto pueden ser analizados de esta manera.

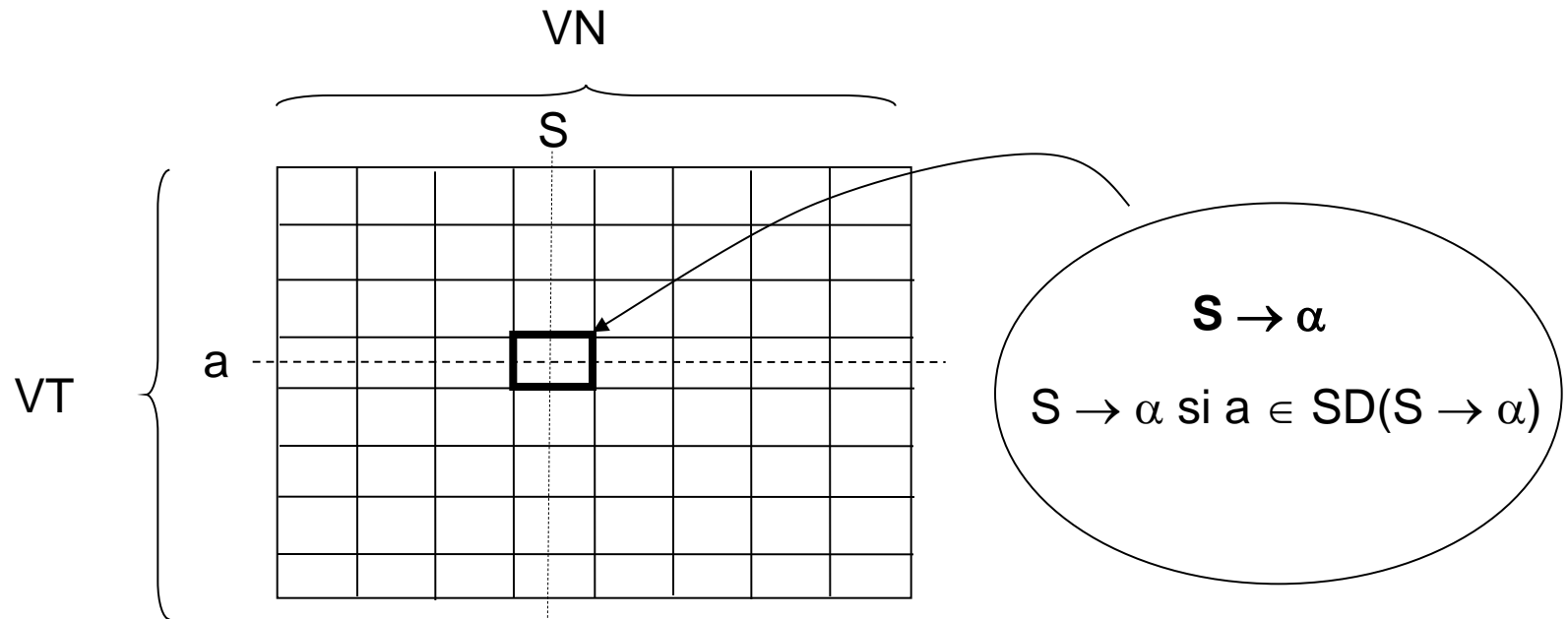
La **ventaja** de los **analizadores sintácticos descendentes con retroceso** es que pueden analizar cualquier lenguaje libre de contexto (excepto sean generados a partir de una gramática recursiva izquierda que veremos próximamente).

Sin embargo, la **desventaja** es que son no determinísticos (no se puede saber en cada momento del análisis cuál es la producción indicada para la expansión de un no terminal).

Implementación de analizadores sintácticos descendentes predictivos con Tabla.

La implementación con procedimientos no es la única posibilidad para producir analizadores sintácticos descendentes predictivos.

Consideremos que disponemos de la siguiente tabla



Cada **fila** tiene asociado **un terminal** y cada **columna** un **no terminal**.

El método dispone de dos estructuras más:

una lista donde está la cadena de entrada que tiene adicionado al final el símbolo “#”, y

una pila que está inicializada con dos símbolos: primero (en el fondo de la pila) el símbolo # y luego (en el tope de la pila) el símbolo distinguido de la gramática “S”.

Sea t es símbolo apuntado en w , M la tabla y tope el símbolo en el tope de la pila, el siguiente algoritmo permite analizar sintácticamente en forma descendente y predictiva la cadena w y determinar si la misma pertenece o no a $L(G)$.

Algoritmo de análisis sintáctico descendente predictivo con tabla.

Repetir

Si $tope \in V_T$

 si $tope == t$

 extraer el tope de la pila

 avanzar 1 en w

 sino

 error.

sino

 si $M(t, tope) == tope \rightarrow Y_1 Y_2 \dots Y_k$

 Extraer $tope$ de la pila

 Poner Y_k, Y_{k-1}, \dots, Y_1 en la pila.

 sino

 error.

hasta que $tope == \# \wedge t == \#$

Ejemplo de ejecución:

Sea $G = \langle V_N, V_T, P, S \rangle$, con $P = \{ S \rightarrow (S), S \rightarrow a \}$.

La tabla será

	S
($S \rightarrow (S)$
a	$S \rightarrow a$
)	
#	

Sea w = ((a)) #

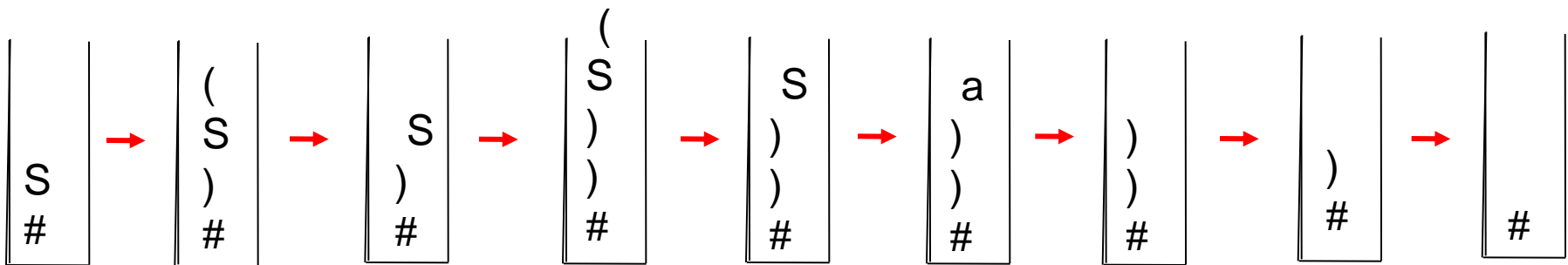


Diagram illustrating the structure of a sequence of 9 elements, each represented as $((a)) \#$. Arrows point upwards from below to the first element of each pair $((a)) \#$.

Gramática LL(1)

Sea una GLC $G = \langle V_N, V_T, P, S \rangle$, G es LL(1) si

$$\forall (N \rightarrow \beta) \in P \wedge \forall (N \rightarrow \beta') \in P \Rightarrow (\beta \neq \beta' \Rightarrow SD(N \rightarrow \beta) \cap SD(N \rightarrow \beta') = \phi)$$

Ejemplo:

Sea $G = \langle V_N, V_T, P, S \rangle$ con $P = \{ E \rightarrow E + T / T, T \rightarrow T * F / F, F \rightarrow (E) / id \}$

Los símbolos directrices para las producciones de E son

$$SD(E \rightarrow E + T) = \{ (, id \},$$

$$SD(E \rightarrow T) = \{ (, id \},$$

y por lo tanto no es LL(1) pues la intersección es distinta de vacío.

Nota: Sea G una GLC / $G = \langle V_N, V_T, P, S \rangle$ y
 G' la gramática ampliada $G' = \langle V_{N'}, V', P', S' \rangle$
con $P' = P \cup \{ S' \rightarrow S\# \}$, $V_{T'} = V_T \cup \{\#\}$ y $V_{N'} = V_N \cup \{S'\}$.

Observación: Toda forma sentencial (apartir de S') es de la forma $\beta\#$, donde
 $\beta \in (V_N \cup V_T)^*$.

Lema:

Dada una GLC G / la gramática ampliada G' sea reducida,
 $\forall U \in V_N$, $\text{Siguietes}(U) \neq \emptyset$.

Demostración:

como G' es reducida, \exists alguna forma sentencial $\alpha U \beta\#$.

Luego puede suceder

- a) $t \in \text{Primeros}(\beta) \Rightarrow t \in \text{Siguietes}(U)$,
- b) $\text{Primeros}(\beta) = \emptyset \Rightarrow \text{Siguietes}(U) = \{\#\}$.

Teorema: Sea $G = \langle V_N, V_T, P, S \rangle$ una GLC reducida y LL(1), entonces G no es recursiva izquierda.

Demostración:

Supongamos que G es recursiva izquierda.

Luego $\exists A \in V_N / A \rightarrow^+ A\alpha, \alpha \in (V_N \cup V_T)^*$.

De esto podemos escribir la siguiente derivación:

$$\mathbf{A=A_0 \rightarrow A_1\alpha_1 \rightarrow A_2\alpha_2 \alpha_1 \rightarrow \dots \rightarrow A_k \alpha_k .. \alpha_1}$$

con $A_k = A$ y $\alpha_k .. \alpha_1 = \alpha$ (en cada paso expandimos el no terminal A_i , que debe ser no terminal pues, si no lo fuera, A no podría volver a aparecer nunca a la izquierda de la cadena).

Por otro lado, dado que G es reducida, debe existir $\beta \in V_T^* / A \rightarrow^+ \beta$.

$$\mathbf{A=A'_0 \rightarrow A'_1\alpha'_1 \rightarrow A'_2\alpha'_2 \alpha'_1 \rightarrow \dots \rightarrow \beta.}$$

Dado que $A_0 = A = A'_0$ y $A\alpha \neq \beta$,

debe existir algún i / $A_{i-1} = A'_{i-1} \wedge A_i \neq A'_i$.

Luego, en el paso i de cada una de las dos derivaciones se utilizaron las producciones

$$A_{i-1} \rightarrow A_i \alpha_i \text{ y } A'_{i-1} \rightarrow A'_i \alpha'_i.$$

Hay dos casos para analizar

1. $\beta \neq \lambda$

En este caso, $\beta = a\gamma$, con $a \in V_T$ y $\gamma \in V_T^*$, luego tenemos que $a \in SD(A'_{i-1} \rightarrow A'_i \alpha'_i)$.

Por otro lado, en la derivación

$$A_{i-1} \alpha_{i-1} \dots \alpha_1 \rightarrow A_i \alpha_i \dots \alpha_1 \rightarrow \dots \rightarrow A_k \alpha_k \dots \alpha_1,$$

como $A_k = A$, luego podemos seguir derivando

$$A_{i-1} \alpha_{i-1} \dots \alpha_1 \rightarrow A_i \alpha_i \dots \alpha_1 \rightarrow \dots \rightarrow A_k \alpha_k \dots \alpha_1 \rightarrow A'_i \alpha'_i \dots \alpha'_1 \alpha_k \dots \alpha_1 \rightarrow \dots \rightarrow a\gamma \alpha_k \dots \alpha_1$$

y por lo tanto

$$a \in SD(A_{i-1} \rightarrow A_i \alpha_i).$$

Como $A'_{i-1} = A_{i-1}$ y

$$SD(A_{i-1} \rightarrow A_i \alpha_i) \cap SD(A'_{i-1} \rightarrow A'_i \alpha'_i) \neq \phi$$

entonces G no es $LL(1)$. Absurdo.

$$2. \beta = \lambda$$

Si esto sucede, entonces $A'_i \alpha'_i$ es anulable y por lo tanto

$$\text{Siguietes}(A'_{i-1}) \subseteq SD(A'_{i-1} \rightarrow A'_i \alpha'_i).$$

Pero también tenemos que $A \rightarrow^* \lambda$, luego $A_i \alpha_i \dots \alpha_1 \rightarrow^* \lambda$.

Así que, si $A_i \alpha_i$ es anulable, en $A_{i-1} \rightarrow A_i \alpha_i$ luego

$$\text{Siguietes}(A_{i-1}) \subseteq SD(A_{i-1} \rightarrow A_i \alpha_i).$$

Como $A'_{i-1} = A_{i-1}$, $\text{Siguietes}(A_{i-1}) = \text{Siguietes}(A'_{i-1})$ y por el lema anterior, como $\forall N \in V_N$, $\text{Siguietes}(N) \neq \phi$, luego

$$SD(A_{i-1} \rightarrow A_i \alpha_i) \cap SD(A'_{i-1} \rightarrow A'_i \alpha'_i) \neq \phi.$$

Eliminación de recursividad izquierda

Sean las siguientes producciones

$$A \rightarrow A\alpha_1 / A\alpha_2 / \dots / A\alpha_m / \beta_1 / \beta_2 / \dots / \beta_n$$

donde ningún β_i comienza con A.

Sustituir las producciones para A por

$$A \rightarrow \beta_1 A' / \beta_2 A' / \dots / \beta_n A'$$

y hacer

$$A' \rightarrow \alpha_1 A' / \alpha_2 A' / \dots / \alpha_m A' / \lambda.$$

El problema de este método es que no contempla la recursividad indirecta.

Por ejemplo: dadas las producciones

$$S \rightarrow Aa,$$

$$A \rightarrow Sd / b,$$

se puede derivar

$$\textcolor{red}{S} \rightarrow Aa \rightarrow \textcolor{red}{S}da \rightarrow Aada \rightarrow \textcolor{red}{S}dada \rightarrow \dots$$

Eliminación completa de recursividad izquierda.

Sea $G = \langle V_N, V_T, P, S \rangle$, $V_N = \{ A_1, A_2, \dots, A_n \} / (A_i \rightarrow \lambda) \notin P \wedge A \rightarrow^+ A$ no es una derivación posible en G .

Algoritmo

1. Para $i : 1 \dots n$

 Para $j : 1 \dots n, (j \neq i)$

 Sustituir cada producción $A_i \rightarrow A_j \gamma$ por

$A_i \rightarrow \delta_1 \gamma / \delta_2 \gamma / \dots / \delta_k \gamma$ donde $A_j \rightarrow \delta_1 / \delta_2 / \dots / \delta_k$.

 Eliminar la recursividad izquierda directa entre las producciones de A_i (si la hubiera).

2. Fin.

Ejemplo:

Sea la gramática de las expresiones con P definida

$P = \{ E \rightarrow E + T / T, \quad (1) \text{ y } (2)$

$T \rightarrow T * F / F, \quad (3) \text{ y } (4)$

$F \rightarrow (E) / i / c \} \quad (5) \text{ y } (6)$

¿Cómo eliminamos la recursividad de (1)?

$E \rightarrow T E'$

$E' \rightarrow + T E' / \lambda$

en forma similar para (3)

$T \rightarrow F T'$

$T' \rightarrow * F T' / \lambda$

Así P' queda

$P' = \{ E \rightarrow T E',$

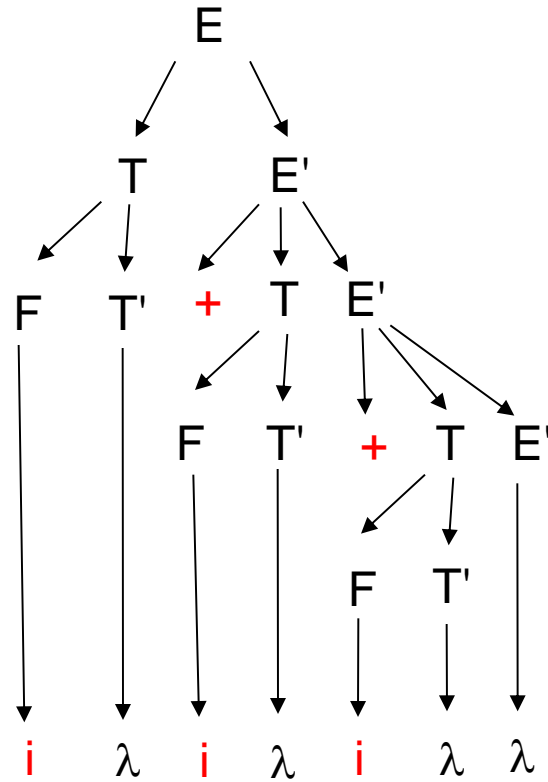
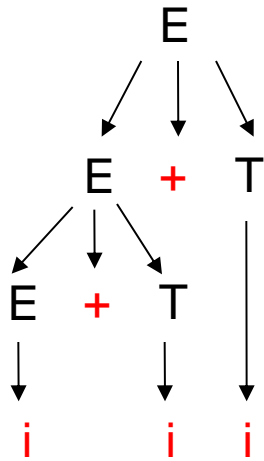
$E' \rightarrow + T E' / \lambda,$

$T \rightarrow F T'$

$T' \rightarrow * F T' / \lambda$

$F \rightarrow (E) / i / c \}.$

Veamos que sucede con los arboles de derivación



Factorización.

Sea $G = \langle V_N, V_T, P, S \rangle$ con P definida así:

$$P = \{ \dots, C \rightarrow i E t C e C / i E t C, \dots \}$$

donde $i, t, e \in V_T$ y $C, E \in V_N$.

Fijarse que G no es $LL(1)$.

Sin embargo puede realizarse factorización, esto es,

Si $A \rightarrow \alpha\beta_1 / \alpha\beta_2$ son dos producciones de A , se pueden reescribir las producciones como

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta_1 / \beta_2$$

donde α es el prefijo comun de $\alpha\beta_1$ y $\alpha\beta_2$ de mayor longitud.

En el ejemplo podemos construir P' como sigue:

$$P' = \{ C \rightarrow i E t C C', C' \rightarrow e C / \lambda \}$$

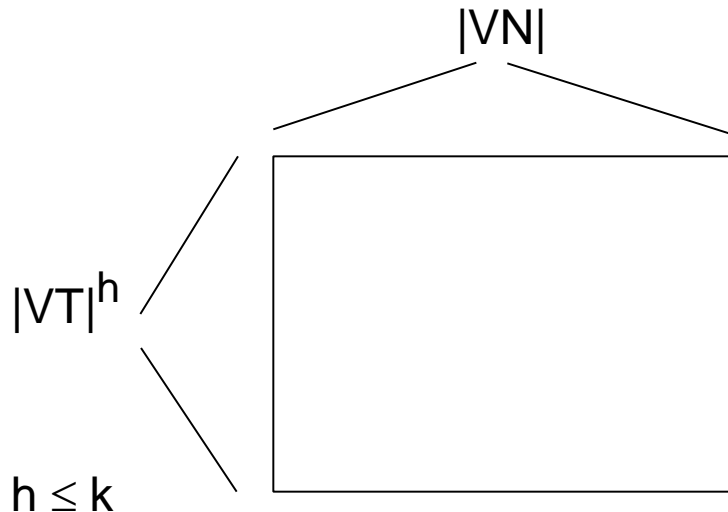
Las gramáticas LL(1) están fuertemente condicionadas a que el primer símbolo hacia adelante debe permitir predecir la derivación a usar.

¿Porqué no usar 2 símbolos hacia adelante?

¿Porqué no usar k símbolos hacia adelante?

Estos métodos se llaman LL(k).

En este caso la matriz tiene la siguiente forma



Esto requiere un mayor uso de memoria. Hay herramientas que cuando descubren un conflicto LL(1) incrementan el k a 2 para resolver dicho conflicto, y así sucesivamente.

Sea $G = \langle V_N, V_T, P, S \rangle$

Definición (Primeros_k):

Primeros_k(α) es el conjunto de todos los prefijos de longitud k o menor de los terminales que pueden derivados a partir de α . Esto es:

$$\text{Primeros}_k : V^* \rightarrow P(V_k),$$

donde $V_k = \{ x / x \in V_T^* \wedge |x| \leq k \}$ y $P(V_k)$ es el conjunto de parte de V_k ,
 $\text{Primeros}_k(\alpha) = \{ x / ((\alpha \rightarrow^* x\beta \wedge |x| = k) \vee (\alpha \rightarrow^* x \wedge |x| < k)) \}.$

Definición (Siguietes_k):

$$\text{Siguietes}_k : V_N \rightarrow P(V_k),$$

$$\text{Siguietes}_k(N) = \{ x / x \in V_T^*, S \rightarrow^* ..Nx.. \wedge |x| \leq k \}.$$

Definición (Símbolos Directrices con k símbolos, SD_k):

$SD_k : P \rightarrow P(V_k)$, donde P es el conjunto de producciones de la gramática

$$SD_k(A \rightarrow \alpha) = \begin{cases} \text{Primeros}_k(\alpha), & \text{si } \alpha \text{ no es anulable} \\ \text{Primeros}_k(\alpha) \cup \text{Siguietes}_k(A), & \text{si } \alpha \text{ es anulable.} \end{cases}$$

Definición (**gramática LL(k)**):

Una gramática $G = \langle V_N, V_T, P, S \rangle$ es LL(k) si

$$\forall (N \rightarrow \beta) \in P \wedge \forall (N \rightarrow \beta') \in P \Rightarrow (\beta \neq \beta' \Rightarrow SD_k(N \rightarrow \beta) \cap SD_k(N \rightarrow \beta') = \emptyset)$$

El problema de decidir si un lenguaje puede ser generado por una gramática LL(k) es no decidible.

Asociado a esto podemos plantearnos 3 preguntas:

- 1) ¿Es G LL(k) para algún k dado?
- 2) ¿Es G LL(k) para algún k?
- 3) Si G no es LL(1), ¿existe G' LL(1) / $L(G') = L(G)$?

La pregunta 1) es un problema decidible mientras que la 2) y la 3) no lo son.

Sin embargo, hay herramientas tales como las de eliminación de recursividad o factorización para intentar convertir una gramática que no es LL(1) en otra que sí lo sea.

En general, la metodología para obtener un analizador sintáctico descendente predictivo es la siguiente:

1. Definir el lenguaje
2. Proponer una gramática que lo genere
3. Verificar si es LL(k).