7. Análisis Sintáctico.[i]

Definición.

El **análisis sintáctico (parsing)** es el proceso de determinar cómo puede derivarse una cadena de terminales mediante una gramática.

La función del **analizador sintáctico** es la de obtener una cadena de tokens del analizador léxico y verificar que la cadena pueda generarse mediante la gramática.

Para ello, el analizador construye un árbol de análisis (árbol de derivación) en el cual la raíz se etiqueta con el símbolo inicial y cada hoja se etiqueta con un terminal o con λ

Métodos de análisis sintáctico.

En todos los métodos, la entrada se explora de izquierda a derecha, un símbolo a la vez.

Los métodos descendentes y ascendentes más eficientes (LL y LR) sólo funcionan para subclases de gramáticas, pero dichas gramáticas son lo bastante expresivas como para describir las construcciones sintácticas en los lenguajes de programación modernos.

Descendentes:

Los métodos descendentes construyen árboles de análisis sintáctico desde la parte superior (raíz) a la parte inferior (hojas).

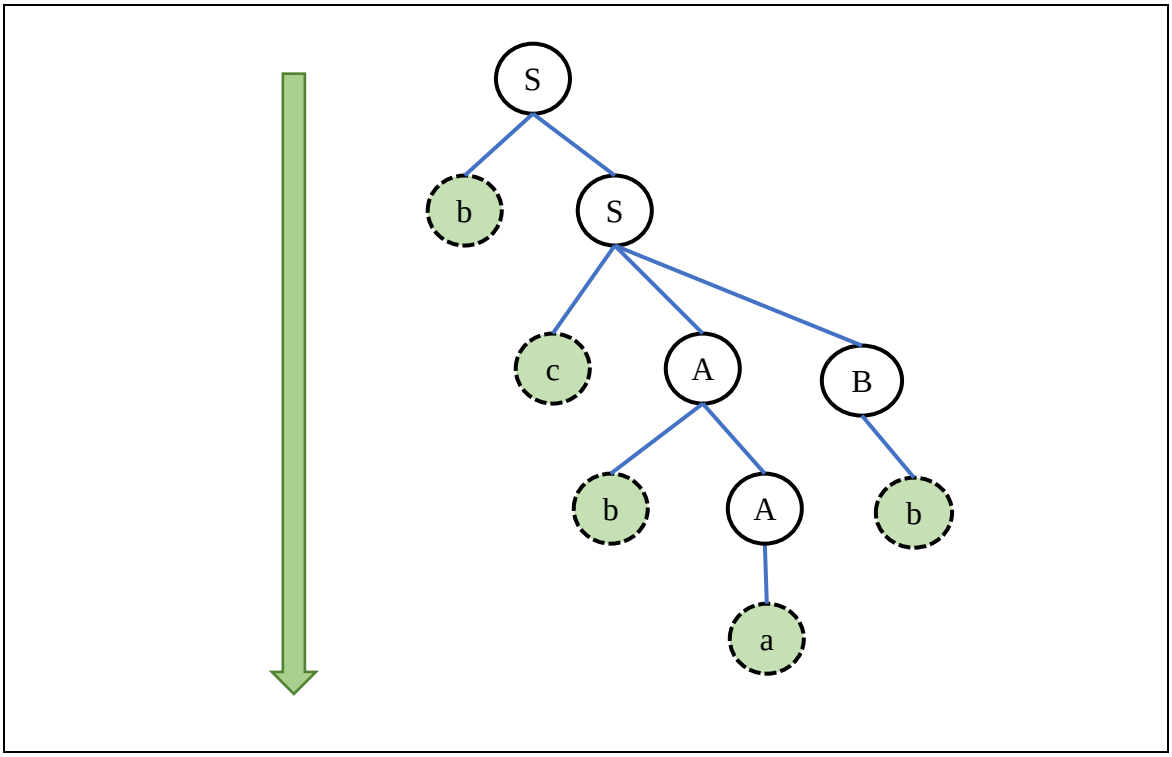
Produce derivaciones por izquierda.

Ejemplo:

$G =< \{S, A, B\}, \{a, b, c\}, S, P >$, donde:

$$P = \{S \rightarrow bS|cAB$$
$$A \rightarrow bA|a$$
$$B \rightarrow aBc|b\}$$

¿Reconoce **bcbab**?



$$S \Rightarrow bS \Rightarrow bcAB \Rightarrow bcbAB \Rightarrow bcbaB \Rightarrow bcbab$$

Ascendentes:

Los métodos ascendentes construyen árboles de análisis sintáctico comenzando por las hojas y avanzando hasta la raíz.

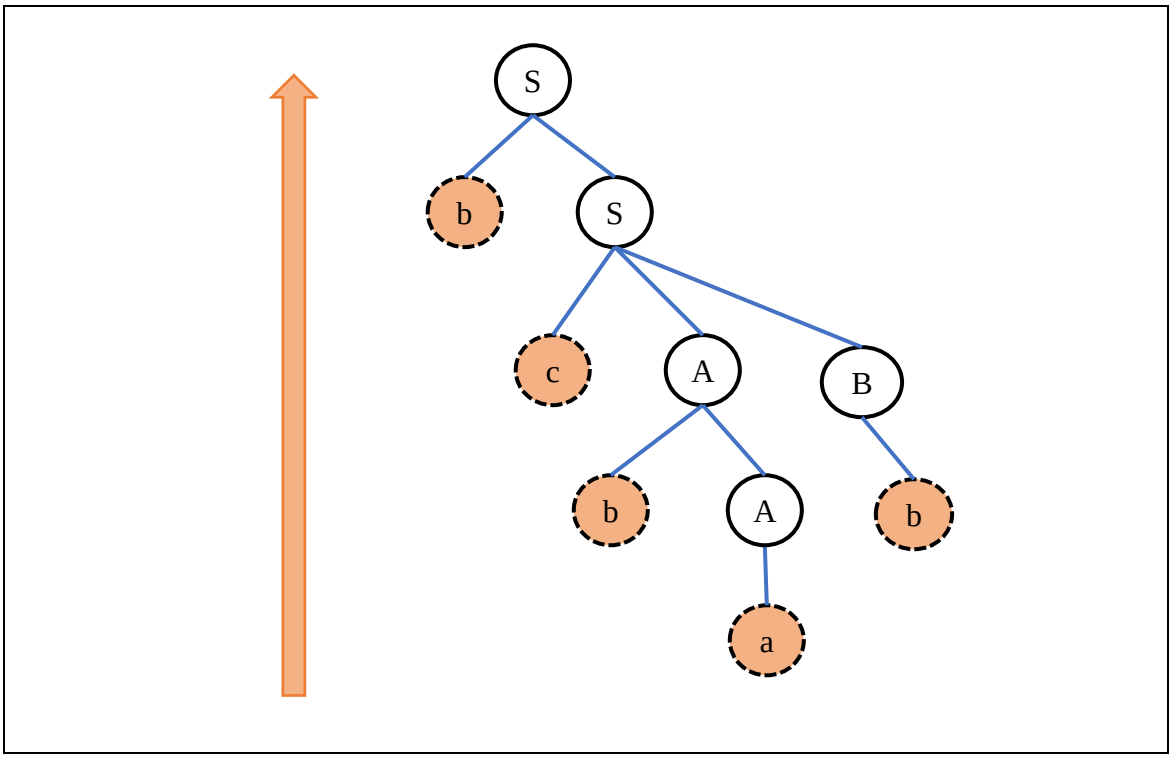
Producen derivaciones inversas.

Ejemplo:

$G =< \{S, A, B\}, \{a, b, c\}, S, P >$, donde:

$$P = \{S \rightarrow bS|cAB$$
$$A \rightarrow bA|a$$
$$B \rightarrow aBc|b\}$$

¿Reconoce **bcbab**?



$$bcbab \Rightarrow bcbAb \Rightarrow bcAb \Rightarrow bcAB \Rightarrow bS \Rightarrow S$$

Análisis Sintáctico Descendente.

En cada paso del **análisis sintáctico descendente** el problema clave es el de determinar cuál producción debe aplicarse para un no terminal.

La forma más general del Análisis Sintáctico Descendente es conocida como ***Análisis Sintáctico de Descenso Recursivo*** y puede requerir *backtracking*.

Un caso especial que no requiere *backtracking* es el ***Análisis Sintáctico Predictivo***.

Las gramáticas para las cuales se puede construir Analizadores Sintácticos Predictivos que analicen k símbolos por adelantado de la entrada se las conoce como gramáticas LL(k).

Otra forma de Análisis Sintáctico Descendente es la conocida como ***Análisis Sintáctico Predictivo No Recursivo***.

Análisis Sintáctico De Descenso Recursivo.

Consiste en un conjunto de funciones, una por cada símbolo no terminal.

La ejecución empieza con el procedimiento principal, que invoca a la función para el símbolo inicial, y se detiene y anuncia que tuvo éxito si el cuerpo de su procedimiento explora toda la cadena completa de entrada.

La función principal es:

```
Principal
Inicio
      t apunta al inicio de ω.
      Si S(t) y t==EOF
            Acepta ω
      Sino
            No acepta ω
fin
```

La función asociada a un no terminal A, siendo $A \rightarrow \alpha_1 |\alpha_2| \ldots |\alpha_k$

```
Funcion A(t) retorna booleano
inicio
      i←1
      hacer
            resultado ←procesar(A → α_i, t)
            i←i+1
      mientras resultado es False y i <= k
      retorna resultado
fin
```

Donde procesar, siendo $A \rightarrow \alpha = X_1 X_2 \ldots X_n$ es:

```
Funcion procesar(A → α, t) retorna booleano
Inicio
      resultado ← True
      para j = 1 ..n
            según X_j:
                  X_j ∈ Σ: si t == X_j:
                        consumir(t)//esto es, avanza t en ω
                        sino
                        retorna False
                  X_j ∈ N: resultado ←X_j(t)
                        Si resultado == False
                              retorna False
                  X_j = λ: resultado ← True //si es vacío, no se hace nada.
      retorna resultado
fin
```

Ejemplo: Sea la gramática cuyas producciones son:

$$S \rightarrow cAd$$
$$A \rightarrow ab|a$$

Las funciones asociadas a cada no terminal son:

```
Funcion S(t) retorna booleano
inicio
      resultado ← procesarS → cAd(t)
      retorna resultado

fin
```

```
Funcion A(t) retorna booleano
inicio
      resultado ← procesarA → ab(t)
      si resultado == False
            resultado ← procesarA → a(t)
      retorna resultado

fin
```

```
Funcion procesarS → cAd(t)retorna booleano
inicio
      resultado←True
      si(t==’c’)
            consumir(t)
            si A(t) ∧ t == 'd'
                  consumir(t)
            sino
                  retorna False
      retorna resultado
fin
```

```
Funcion procesarA → ab(t)retorna booleano
inicio
      resultado←True
      si(t==’a’)
            consumir(t)
            si t == 'b'
                  consumir(t)
            sino:
                  retorna False
      sino:
            retorna False
      retorna resultado
fin
```

```
Funcion procesarA → a(t)retorna booleano
inicio
      resultado←True
      si(t==’a’)
            consumir(t)
      sino:
            retorna False
      retorna resultado
fin
```

De manera más simple, se puede también escribir:

```
Principal
Inicio
      t apunta al inicio de ω.
      Si S(t) y t==EOF
            Acepta ω
      Sino
            No acepta ω
fin
```

```
Funcion S(t) retorna booleano
inicio
       si t==’c’:
             consumir(t)
             si A(t) y t ==’d’:
                   consumir(t)
                   retorna True
             retorna False
       retorna False

fin
```

```
Funcion A(t) retorna booleano
inicio
      si t == ‘a’:
            consumir(t)
            si t == ‘b’:
                  consumir(t)
                  retorna True
            retorna True
      retorna False

fin
```

Entonces, sea la cadena $\omega = cad$.

El programa principal hace que t apunte al inicio de ω, es decir a ‘c’.

Se invoca así a S('c').

S('c')→consume(t)→A('a')→consume(t)→retorna True

Vuelve entonces a donde fue invocada A, y como t ahora apunta a 'd', consume (t), retorna True.

Vuelve al programa principal, con True y t apuntando a EOF, entonces la cadena es aceptada.

En este método:

- la cadena se analiza de izquierda a derecha.
- siempre se elige la derivación más a la izquierda
- si la gramática es recursiva por izquierda hará que el analizador entre en un ciclo infinito.

Eliminación de Recursividad Izquierda:

Una gramática es recursiva por la izquierda si tiene un no terminal A tal que haya una derivación $A \Rightarrow^+ A\alpha$.

Los métodos de análisis sintáctico descendente no pueden manejar las gramáticas recursivas a izquierda, por lo que hay que transformarlas.

Una gramática tiene ***recursividad inmediata*** por la izquierda si tiene alguna producción de la forma $A \to A\alpha$.

La ***recursividad inmediata*** por la izquierda se elimina de la siguiente manera:

1. Se agrupan las producciones de la siguiente manera:
$$A \to A\alpha_1 | A\alpha_2 | \dots A\alpha_m | \beta_1 | \beta_2 \dots | \beta_n$$
En donde ninguna β_i termina en A.
2. Luego, se sustituyen las producciones A mediante:
$$A \to \beta_1 A' | \beta_2 A' | \dots | \beta_n A''$$
$$A' \to \alpha_1 A' | \alpha_2 A' | \dots | \alpha_m A' | \lambda$$

Algoritmo para eliminar la recursividad por izquierda.

ENTRADA:

$G =< N, \Sigma, P, S >$ sin ciclos (derivaciones $A \Rightarrow^+ A$) ni producciones λ

SALIDA:

G', equivalente a G sin recursividad a izquierda.

PASOS:

1. Ordenar los no terminales de cierta forma $A_1, A_2, \dots A_n$
2. Para cada i de 1 a n:
   Para cada j de 1 a i-1
      i. Sustituir a cada producción de la forma $A_i \to A_j\gamma$ por las producciones $A_i \to \delta_1\gamma | \delta_2\gamma | \dots | \delta_k\gamma$, en donde:
      $A_j \to \delta_1 | \delta_2 | \dots | \delta_k$ sean todas producciones A_j actuales.
   Eliminar la recursividad inmediata por la izquierda entre las producciones A_i

Ejemplo: Sea la gramática cuyas producciones son:

$$S \to Aa \mid b$$
$$A \to Ac \mid Sd \mid \lambda$$

Si consideramos $A_1 = S\ y\ A_2 = A$, aplicando el algoritmo:
Primero se considera $A \to Sd \wedge S \to Aa|b$, donde $\gamma = d$, y $\boldsymbol{\delta_1 = Aa}$ y $\boldsymbol{\delta_2 = b}$.
Luego del reemplazo $A_i \to \delta_1\gamma|\delta_2\gamma \dots$ queda $A \to \boldsymbol{Aa}d|\boldsymbol{b}d|Ac|\lambda$
Sobre ese conjunto de producciones, se elimina la recursividad a izquierda.
La gramática sin recursividad a izquierda resulta:

$$S \to Aa \mid b$$
$$A \to bdA' | A''$$
$$A' \to cA' | adA' | \lambda$$

Análisis sintáctico predictivo.

Una forma simple de análisis sintáctico de descenso recursivo es el conocido como ***análisis sintáctico predictivo***, que no requiere backtracking y en el cual el símbolo de preanálisis determina sin ambigüedad el flujo de control a través del cuerpo del procedimiento para cada no terminal.
El análisis sintáctico predictivo se basa en información acerca de los primeros símbolos que pueden generarse mediante el cuerpo de una producción. Ese conjunto de PRIMEROS se complementa con el conjunto de SIGUIENTES y nos permiten elegir la producción a aplicar, con base en el siguiente símbolo de entrada.

Factorización por la Izquierda

Algoritmo para factorizar por izquierda una gramática.

ENTRADA:

$G =< N, \Sigma, P, S >$ con producciones $A \rightarrow \alpha\beta_1 | \alpha\beta_2 \ldots$

SALIDA:

G', equivalente a G factorizada por izquierda.

PASOS:

1. Por cada no terminal A encontrar el prefijo α más largo que sea común para una o más de sus alternativas.
2. Si $\alpha \neq \lambda$, se sustituyen todas las producciones A, $A \rightarrow \alpha\beta_1 | \alpha\beta_2 \ldots | \alpha\beta_n | \gamma$ mediante lo siguiente:

$$A \rightarrow \alpha A' | \gamma$$
$$A' \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n$$

Ejemplo: Sea la gramática cuyas producciones son:

$$S \rightarrow iEtS | iEtSeS | a$$
$$E \rightarrow b$$

(donde i significa **if**, t significa **then**, e significa **else**, a es una expresión y b una condición)

La gramática factorizada resulta:

$$S \rightarrow iEtSS' | a$$
$$S' \rightarrow eS | \lambda$$
$$E \rightarrow b$$

Conjunto de PRIMEROS

Definimos PRIMERO(α) como el conjunto de terminales que empiezan las cadenas derivadas a partir de α.

PRIMERO(α)=$\{t \in \Sigma | \alpha \Rightarrow^* t\beta\} \cup \{\lambda | \alpha \Rightarrow^* \lambda\}$

Algoritmo para obtener el conjunto de PRIMEROS.

ENTRADA:

$G =< N, \Sigma, P, S >$

SALIDA:

$PRIMERO(X), \forall X \in N,$

REGLAS:

1. $\forall t \in \Sigma, PRIMEROS(t) = \{t\}$
2. $Si\ X \in N \wedge X \rightarrow Y_1 Y_2 \ldots Y_k$, entonces $t \in PRIMEROS(X)$si para algún i, $t \in PRIMEROS(Y_i)$ y $\lambda \in PRIMEROS(Y_1) \ldots . PRIMEROS(Y_{\{i-1\}})$
3. $Si\ X \rightarrow \lambda \in P$, agregar λ a $PRIMEROS(X)$

Conjunto de SIGUIENTES

Definimos SIGUIENTE(A) como el conjunto de terminales que pueden aparecer a la derecha de A en alguna forma sentencial.

SIGUIENTE(A)=$\{t \in \Sigma | S \Rightarrow^* \alpha A t \beta\} \cup \{\$ | S \Rightarrow^* \alpha A\}$

El símbolo $ significa el EOF o fin de la cadena.

Algoritmo para obtener el conjunto de SIGUIENTES.

ENTRADA:

$$G =< N, \Sigma, P, S >$$

SALIDA:

$$SIGUIENTES(X), \forall X \in N,$$

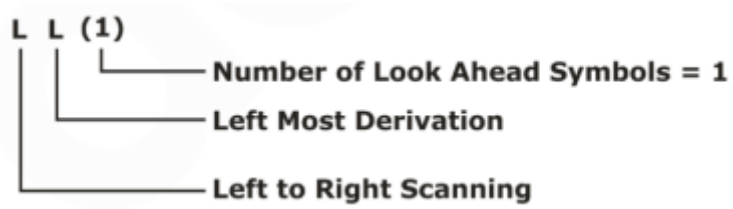
REGLAS:

1. Agregar \$ a $SIGUIENTES(S)$, donde S es el símbolo inicial y \$ el delimitador derecho de la entrada (fin de cadena)
2. Si hay una producción $A \rightarrow \alpha B \beta$, entonces todo lo que está en $PRIMEROS(\beta)$, excepto λ está en $SIGUIENTES(B)$
3. Si hay una producción $A \rightarrow \alpha B$, o una producción $A \rightarrow \alpha B \beta$, en donde en $PRIMEROS(\beta)$ está λ entonces todo lo que hay en $SIGUIENTES(A)$ está en $SIGUIENTES(B)$

Gramáticas LL(1)

Los ***analizadores sintácticos predictivos*** pueden construirse a partir de una clase de gramáticas llamadas LL(1).

LL(1) significa **Left to Right** (la entrada se explora de izquierda a derecha), **Leftmost derivation** (derivaciones por izquierda), tomando un único símbolo de entrada de anticipación en cada paso para tomar las decisiones de acción.



Una gramática G es LL(1) si , para todas las producciones distintas $X \rightarrow \alpha_1 | \alpha_2 | .. | \alpha_n$

- $PRIMEROS(\alpha_i) \cap PRIMEROS(\alpha_j) = \emptyset$
- Si $X \Rightarrow^* \lambda$, $PRIMEROS(X) \cap SIGUIENTES(X) = \emptyset$

Para eso, la gramática:

- No debe ser ambigua
- No debe tener recursividad a izquierda.
- Debe estar factorizada

Algoritmo para obtener una tabla de análisis sintáctico predictivo.

ENTRADA:

$$G =< N, \Sigma, P, S >$$

SALIDA:

Tabla de análisis sintáctico M, donde las filas están etiquetadas con los símbolos no terminales y las columnas están etiquetadas con los símbolos terminales y \$ (marca de fin de cadena)

PASOS:

Para cada producción $A \rightarrow \alpha$:

1. Por cada terminal $a \in PRIMEROS(\alpha)$, agregar $A \rightarrow \alpha$ a $M[A, a]$.
2. Si $\lambda \in PRIMEROS(\alpha)$, entonces por cada terminal $b \in SIGUIENTES(A)$, agregar $A \rightarrow \alpha$ a $M[A, b]$. Si $\lambda \in PRIMEROS(\alpha)$, y \$ $\in SIGUIENTES(A)$, agregar $A \rightarrow \alpha$ a $M[A, \$]$.
   Si después de realizar lo anterior no hay producción en alguna celda $M[A, a]$, entonces se establece $M[A, a] = error$.

Ejemplo: Sea la gramática $G =< \{E, T, E', T', F\}, \{+, *, (,), id\}, E, P >$cuyas producciones son:

$$E \rightarrow TE'$$
$$E' \rightarrow +TE' | \lambda$$
$$T \rightarrow FT'$$
$$T' \rightarrow * FT' | \lambda$$
$$F \rightarrow (E) | \boldsymbol{id}$$

Los conjuntos de PRIMEROS son:

$$PRIMEROS(E) = PRIMEROS(T) = PRIMEROS(F) = \{(, \boldsymbol{id}\}$$
$$PRIMEROS(E') = \{+, \lambda\}$$
$$PRIMEROS(T) = PRIMEROS(F) = \{(, \boldsymbol{id}\}$$
$$PRIMEROS(T') = \{*, \lambda\}$$
$$PRIMEROS(F) = \{(, \boldsymbol{id}\}$$

Los conjuntos de SIGUIENTES son:

$$SIGUIENTES(E) = \{\$\} \cup \{)\} = \{\$,)\}$$
$$SIGUIENTES(E') = SIGUIENTES(E) = \{\$,)\}$$
$$SIGUIENTES(T) = PRIMEROS(E') = \{+\} \cup SIGUIENTES(E) = \{+, \$,)\}$$
$$SIGUIENTES(T') = SIGUIENTES(T) = \{+, \$,)\}$$
$$SIGUIENTES(F) = PRIMEROS(T') = \{*\} \cup SIGUIENTES(T) = \{*, +, \$,)\}$$

La tabla queda:

NO TERMINAL	SÍMBOLO DE ENTRADA					
	id	+	*	(	)	$
E	E→TE'	**error**	**error**	E→TE'	**error**	**error**
E'	**error**	E'→+TE'	**error**	**error**	E'→λ	E'→λ
T	T→FT'	**error**	**error**	T→FT'	**error**	**Error**
T'	**Error**	T'→λ	T'→*FT'	**Error**	T'→λ	T'→λ
F	F→**id**	**error**	**error**	F→(E)	**error**	**error**

Para cada gramática LL(1) cada entrada en la tabla de análisis sintáctico identifica en forma única a una producción.

Si G es recursiva a izquierda o no está factorizada por izquierda, M tendrá por lo menos una entrada con múltiples definiciones.

Hay algunas gramáticas para las cuales ningún tipo de modificación producirá una gramática LL(1)

Ejemplo: Sea la gramática cuyas producciones son:

$$S \rightarrow iEtSS'|a$$
$$S' \rightarrow eS|\lambda$$
$$E \rightarrow b$$

Esta gramática:

- No tiene recursividad izquierda
- No necesita factorización
- Es ambigua. Eso hará que no sea LL(1)

Los conjuntos de PRIMEROS son:

$$PRIMEROS(S) = \{\boldsymbol{i}, \boldsymbol{a}\}$$
$$PRIMEROS(S') = \{\boldsymbol{e}, \lambda\}$$
$$PRIMEROS(E) = \{\boldsymbol{b}\}$$

Los conjuntos de SIGUIENTES son:

$$SIGUIENTES(S) = \{\$, \boldsymbol{e}\}$$
$$SIGUIENTES(S') = \{\$, \boldsymbol{e}\}$$
$$SIGUIENTES(E) = \{\boldsymbol{t}\}$$

La tabla queda:

NO TERMINAL	SÍMBOLO DE ENTRADA					
	a	**b**	**e**	**i**	**t**	**$**
S	$S \to a$			$S \to iEtSS'$		
S'			$S' \to eS$ $S' \to \lambda$			$S' \to \lambda$
E		$E \to b$				

Donde se observa la ambigüedad que se presenta cuando se tiene S' en la pila y e en la entrada.

Análisis sintáctico predictivo no recursivo.

Se puede construir un ***analizador sintáctico predictivo no recursivo***, mediante el mantenimiento explícito de una pila, en vez de hacerlo mediante llamadas recursivas implícitas.

Este analizador imita una derivación por la izquierda.

Si ω es la entrada que se ha relacionado hasta ahora, entonces la pila contiene una secuencia de símbolos gramaticales α de tal forma que $S \Rightarrow_{lm}^{*} \omega\alpha$

El analizador sintáctico controlado por tabla tiene:

- Un buffer de entrada
- Una pila que contiene una secuencia de símbolos gramáticales.
- Una tabla de análisis sintáctico
- Un flujo de salida.

El buffer de entrada contiene la cadena que se va a analizar, seguida por el marcador final $.

El símbolo $ además se usa para marcar el fondo de la pila, que al principio contiene el símbolo inicial de la gramática.

El analizador es controlado por el programa que considera X, el símbolo en el tope de la pila y a, el símbolo de entrada actual.

Si X es un no terminal, el analizador sintáctico elige una producción X mediante una consulta a la entrada $M[X, a]$ de la tabla.

Sino, verifica si hay una coincidencia entre el terminal X y el símbolo de entrada actual a.

El comportamiento del analizador sintáctico puede describirse en términos de sus configuraciones, que proporcionan el contenido de la pila y el resto de la entrada.

Algoritmo de análisis sintáctico predictivo, controlado por tabla.

ENTRADA:

Una cadena ω.
Una tabla de análisis predictivo M para la gramática $G =< N, \Sigma, P, S >$

SALIDA:

Si $\omega \in L(G)$, una derivación a la izquierda de ω.
Si $\omega \notin L(G)$, señala error.

PASOS:

Configuración inicial:

- $\omega$$ en el buffer de entrada.
- S (símbolo inicial de G) en el tope de la pila, sobre $.

Pasos:

Sea a el primer símbolo de ω
Sea X el símbolo en el tope de la pila
Mientras ($X \neq \$$)
    Si ($X = a$)
        desapilar X, consumir a.
    Sino:

Si(X es un terminal)
    error()
Si($M[X, a]$ es una celda de error)
    error()
Sino:
    Si($M[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$)
        Enviar como salida la producción $X \rightarrow Y_1 Y_2 \dots Y_k$
        Desapilar X
        Apilar $Y_1 Y_2 \dots Y_k$, con Y_1 en el tope de la pila.
Sea X el símbolo en el tope de la pila.

Ejemplo: Sea la gramática $G =< \{E, T, E', T', F\}, \{+, *, (,), id\}, E, P >$cuya tabla de análisis es

NO TERMINAL	SÍMBOLO DE ENTRADA					
	id	+	*	(	)	$
E	E→TE'	**error**	**error**	E→TE'	**error**	**error**
E'	**error**	E'→+TE'	**error**	**error**	E'→λ	E'→λ
T	T→FT'	**error**	**error**	T→FT'	**error**	**Error**
T'	**Error**	T'→λ	T'→*FT'	**Error**	T'→λ	T'→λ
F	F→**id**	**error**	**error**	F→(E)	**error**	**error**

Con la entrada **id+id*id,** el funcionamiento del analizador es:

Coincidencia	Pila	Entrada	Acción
	E$	**id + id * id $**	
	TE'$	**id + id * id $**	Emitir E→TE'
	FT'E'$	**id + id * id $**	Emitir T→FT'
	id T'E'$	**id + id * id $**	Emitir F→id
id	T'E'$	**+ id * id $**	Relacionar **id**
id	E'$	**+ id * id $**	Emitir T'→ λ
id	+TE'$	**+ id * id $**	Emitir E'→+TE'
id +	TE'$	**id * id $**	Relacionar +
id +	FT'E'$	**id * id $**	Emitir T→FT'
id +	idT'E'$	**id * id $**	Emitir F → id
id + id	T'E'$	*** id $**	Relacionar **id**
id + id	*FT'E'$	*** id $**	Emitir T'→*FT'
id + id *	FT'E'$	**id $**	Relacionar *
id + id *	idT'E'$	**id $**	Emitir F→id
id + id * id	T'E'$	**$**	Relacionar id
id + id * id	E'$	**$**	Emitir T'→ λ
id + id * id	$	**$**	Emitir E'→ λ

Análisis Sintáctico Ascendente.

La forma más general del Análisis Sintáctico Ascendente es conocida como ***Análisis Sintáctico de Desplazamiento-Reducción***.

El Análisis Sintáctico Ascendente es el proceso de **reducir** una cadena ω al símbolo inicial de la gramática. En cada paso de **reducción**, una subcadena específica βque coincide con el cuerpo de alguna producción $A \rightarrow \beta$ es reemplazada por el símbolo no terminal correspondiente (A). Entonces, en el **análisis sintáctico ascendente** el problema clave es determinar en qué momento hacer una reducción y qué producción aplicar.

La clase más extensa de gramáticas para las cuales pueden construirse los analizadores Sintácticos de Desplazamiento-Reducción son las gramáticas LR.

Análisis Sintáctico de Desplazamiento-Reducción.

El Análisis Sintáctico de Desplazamiento-Reducción es una forma de análisis sintáctico ascendente en el cual, durante la lectura de la cadena de entrada, se van desplazando símbolos de la entrada a una pila hasta que en determinados momentos se pueden reemplazar ciertos símbolos por un no terminal de la gramática.

El objetivo es llegar a poner en la pila el símbolo inicial de la gramática, al mismo tiempo que se termina la lectura de la cadena.

Reducción.

Consiste en reemplazar una subcadena específica βque coincide con el cuerpo de alguna producción $A \to \beta$ por el símbolo no terminal correspondiente (es decir, A)

Es el proceso inverso al de una derivación.

Pivote

Dada $\alpha\rho\beta$ forma sentencial de la gramática, ρ es ***pivote*** si y sólo si $\exists(N \to \rho) \in P \wedge S \Rightarrow^* \alpha N\beta \Rightarrow \alpha\rho\beta$

Cada vez que tenemos ρ en el tope de la pila, se puede hacer una reducción.

Prefijos Viables

Dada $\alpha\rho\beta$, forma sentencial derecha, donde ρ es pivote,$\boldsymbol{\gamma}$ **es prefijo viable** si γ es prefijo de $\alpha\rho$

El contenido de la pila siempre es un prefijo viable.

Ejemplo:

$G =< \{S, A, B, C\}, \{a, b, c\}, S, P >$cuyas producciones son:

$$S \to ABC$$
$$A \to aa$$
$$B \to bb$$
$$C \to cc$$

Son ***pivotes:*** aa, bb, cc, ABC.

Son ***prefijos viables***: A, AB, ABc, ABcc, aab, Abbc, Abb, aa, a, Ab, etc.

Desplazamiento.

Consiste en desplazar un símbolo de la cadena de entrada al tope de la pila de análisis.

Algoritmo de análisis sintáctico de Desplazamiento - Reducción.

El algoritmo usa una pila para ir guardando símbolos de la gramática, y un buffer de entrada para el resto de la cadena que debe analizar.

ENTRADA:

Una cadena ω.

SALIDA:

Si $\omega \in L(G)$, una derivación inversa de ω.

Si $\omega \notin L(G)$, señala error.

Configuración inicial:

- $\omega\$$ en el buffer de entrada.
- $ en el fondo de la pila.

PILA	ENTRADA
$	$\omega\$$

Pasos:

Repetir:

1. Desplazar cero o más símbolos de entrada a la pila.
2. Reducir una cadena β de símbolos de la parte superior de la pila a A, si existe $A \to \beta$ en la gramática.

Hasta que sea error o (pila==$S y entrada ==$)

Configuración final: (si no hubo error)

PILA	ENTRADA
$S	$

Ejemplo:
$G =< \{E, T, F\}, \{+, *, (,), id\}, E, P >$cuyas producciones son:

$$E \rightarrow E + T|T$$
$$T \rightarrow T * F|F$$
$$F \rightarrow (E)|\boldsymbol{id}$$

PILA	ENTRADA	ACCIÓN
$\$$	$id * id\$$	Desplazar (shift)
$\$id$	$* id\$$	Reducir con F→id
$\$F$	$* id\$$	Reducir con T→F
$\$T$	$* id\$$	Desplazar
$\$T *$	$id\$$	Desplazar
$\$T * id$	$\$$	Reducir con F→id
$\$T * F$	$\$$	Reducir con T→T*F
$\$T$	$\$$	Reducir con E→T
$\$E$	$\$$	Aceptar

La derivación inversa es:

$$id * id \Rightarrow F * id \Rightarrow T * id \Rightarrow T * F \Rightarrow T \Rightarrow E$$

Que corresponde a una derivación más a la derecha:

$$E \Rightarrow T \Rightarrow T * \boldsymbol{F} \Rightarrow T * id \Rightarrow F * id \Rightarrow id * id$$

Conflictos.

Hay gramáticas libres de contexto para las cuales el análisis sintáctico de Desplazamiento-Reducción no puede usarse.

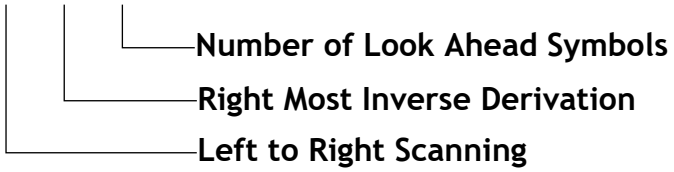
En esas gramáticas, el analizador llega a una configuración en la que, conociendo el contenido completo de la pila y el siguiente símbolo de la entrada, no puede decidir:

- Si va a desplazar o a reducir (***conflicto D-R, de Desplazamiento – Reducción***)
- Qué reducción realizar (***conflicto R-R, de Reducción – Reducción***)

Análisis Sintáctico LR.

LR(k) significa **Left to Right** (la entrada se explora de izquierda a derecha), **Rightmost derivation in reverse** (derivaciones por derecha revertidas), tomando k símbolos de anticipación en cada paso para tomar las decisiones de acción.

L R (k)

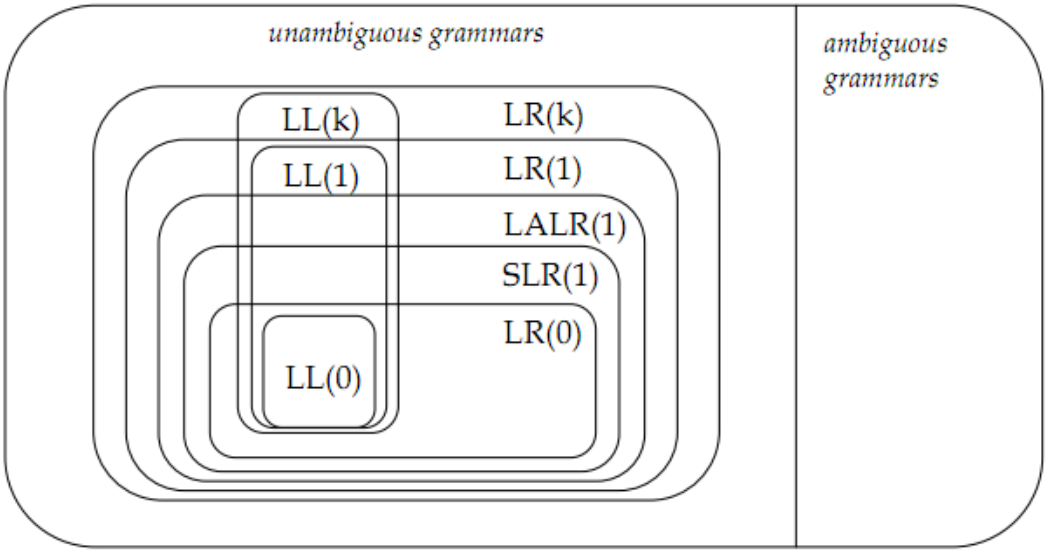


Las gramáticas para las cuales se puede construir un analizador sintáctico LR se denominan gramáticas LR.

La clase de gramáticas que pueden analizarse mediante los métodos LR (gramáticas LR) es un superconjunto de las gramáticas LL. Por lo tanto, las gramáticas LR pueden describir más lenguajes que las gramáticas LL.

LL(1) versus LR(k)

A picture is worth a thousand words:



Items y autómata LR(0).

Un analizador sintáctico LR realiza las decisiones de desplazamiento-reducción mediante un autómata que le permite llevar el registro de la ubicación en la que se encuentra en el análisis.

Un ***autómata LR(***0) es un autómata finito determinista que se utiliza para realizar decisiones en el análisis sintáctico.

Los **estados** del ***autómata LR(***0) son los conjuntos de ítems de la colección LR(0) canónica.

Las **transiciones** del ***autómata LR(***0) las proporciona la función $Ir - A(I, X)$

El **estado inicial** del ***autómata LR(***0) es $CLAUSURA(\{[S' \rightarrow \bullet S]\})$ en donde S' es el símbolo inicial de G.

Se puede construir un AFN-λ donde en cada estado hay un solo item, y hay transiciones con λ, y luego se puede convertir con los algoritmos habituales a AFD. Sin embargo, se puede hacer el AFD directamente con los algoritmos que se indican a continuación.

Un ***item*** (o ***elemento LR(0***)) es una producción de la gramática con un punto en cierta posición: $[A \rightarrow \alpha_1 \bullet \alpha_2]$

Por ejemplo, la producción $A \rightarrow XYZ$ produce los siguientes cuatro ítems:

$[\boldsymbol{A \rightarrow \bullet XYZ}]$ (item inicial, se espera encontrar una cadena derivable de XYZ en la entrada)

$[\boldsymbol{A \rightarrow X \bullet YZ}]$ (item intermedio, ya se vio una cadena derivable de X en la entrada, se espera ver una derivable de YZ)

$[\boldsymbol{A \rightarrow XY \bullet Z}]$ (item intermedio, ya se vio una cadena derivable de XY en la entrada, se espera ver una derivable de Z)

$[\boldsymbol{A \rightarrow XYZ \bullet}]$ (item completo, ya se vio todo el cuerpo XYZ, puede reducirse XYZ a A)

Y la producción $A \rightarrow \lambda$ produce un solo item: $[A \rightarrow \bullet]$

Un ***estado*** representa un conjunto de ítems.

Para construir la colección LR(0) canónica de una gramática G, se define una gramática aumentada y dos funciones: CLAUSURA (CLOSURE) e IR-A (GOTO)

Si G es una gramática con el símbolo inicial S, entonces G' es G con un nuevo símbolo inicial S' y la producción $S' \to S$.

Algoritmo para obtener la CLAUSURA(I) (CLAUSURA para un conjunto de ítems I)

ENTRADA:

Un conjunto I de ítems de la gramática G.

SALIDA:

Un conjunto J de ítems,J= CLAUSURA (I)

PASOS:

J=I

Repetir

para cada $[A \to \alpha \bullet \boldsymbol{B\beta}]$ en J:

para cada $B \to \gamma$ en G:

agregar [$B \to \bullet\boldsymbol{\gamma}$**]** $\boldsymbol{a\ J}$**:**

hasta que no se agreguen nuevos elementos a J.

retornar J.

Ejemplo: Sea la gramática $G =< \{E', E, T, F\}, \{+, *, (,), id\}, E, P >$con producciones

$$E' \to E$$
$$E \to E + T | T$$
$$T \to T * F | F$$
$$F \to (E) | \boldsymbol{id}$$

Si $I = \{[E' \to \bullet E]\}$, CLAUSURA(I) es:

$$\{$$
$$[E' \to \bullet E],$$
$$[E \to \bullet E + T],$$
$$[E \to \bullet T],$$
$$[T \to \bullet T * F],$$
$$[T \to \bullet F],$$
$$[F \to \bullet (E)],$$
$$[F \to \bullet \boldsymbol{id}]$$
$$\}$$

Definición de función IR-A (GOTO)

La función ***IR-A (goto)*** toma:

- Parámetros de entrada: una clausura(I) y un símbolo gramatical X
- Da como resultado una clausura (J)

Se define $IR - A(I, X)$ como la CLAUSURA del conjunto de todos los elementos $[A \to \alpha X \bullet \beta]$ tal que $[A \to \alpha \bullet X\beta] \in I$.

$IR\_A(I, X)$ especifica la transición del estado I con la entrada X en el autómata LR(0)

Ejemplo: Para la gramática anterior, y siendo I ={

$$[E' \to E\bullet],$$
$$[E \to E\bullet + T],$$
$$\}$$

Entonces:

$$IR\_A\ (I, +) = \{$$
$$[E \to E + \bullet T],$$
$$[T \to \bullet T * F],$$
$$[T \to \bullet F],$$
$$[F \to \bullet (E)],$$
$$[F \to \bullet \boldsymbol{id}]$$
$$\}$$

Algoritmo para obtener la colección canónica de conjuntos de ítems LR(0) para G.

ENTRADA:

Una gramática aumentada G'.

SALIDA:

Colección canónica de conjuntos de ítems LR(0)

PASOS:

$$C = CLAUSURA(\{[S' \rightarrow \bullet S]\})$$

Repetir:

Para cada conjunto de ítems I en C:

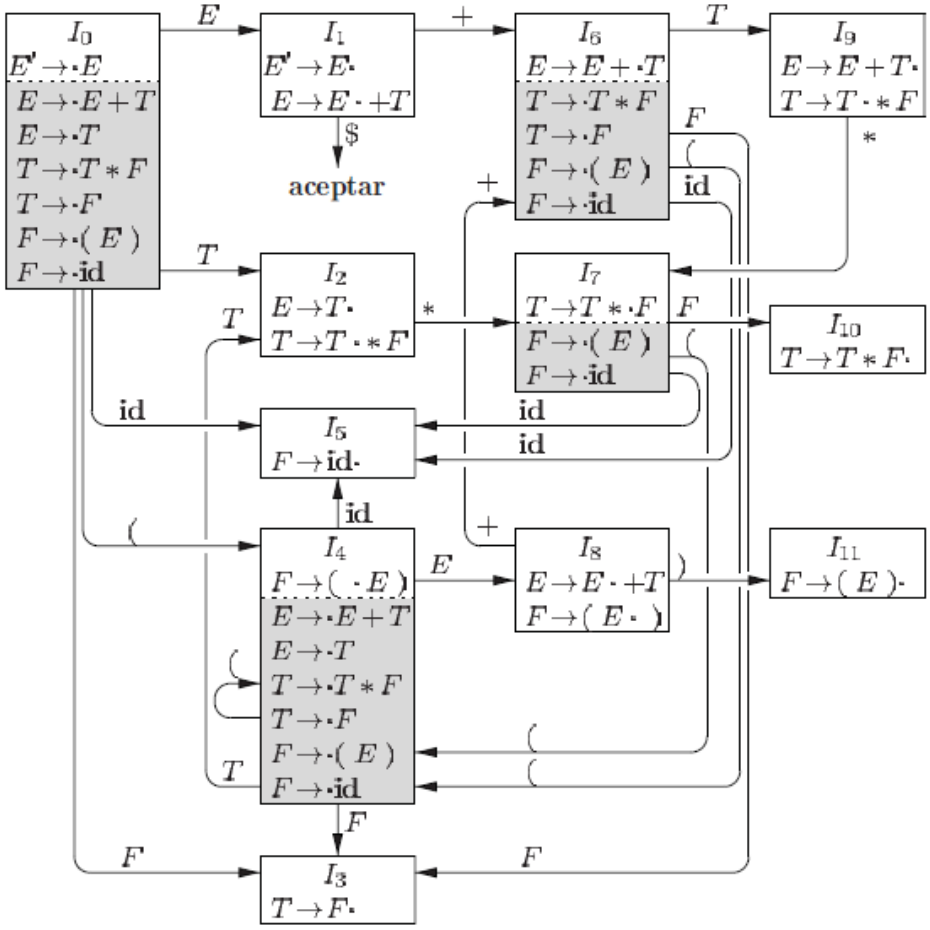
Para cada símbolo gramatical X:

Si $ir - A(I,X) \neq \emptyset \wedge ir - A(I,X) \notin C$:

Agregar $ir - A(I,X)$ a C.

Hasta que no se agreguen nuevos conjuntos de ítems a C.

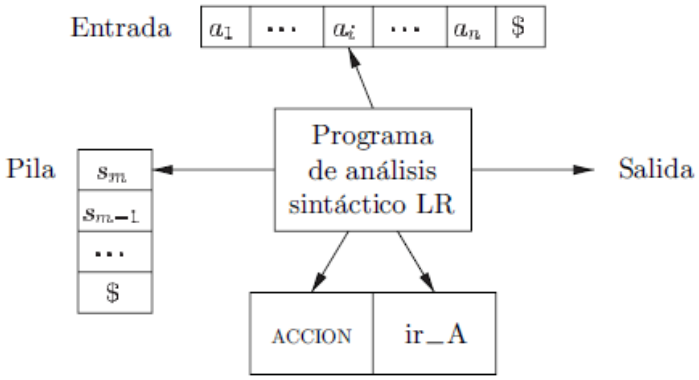
Ejemplo: Para la gramática anterior, la colección canónica de conjuntos de ítems LR(0) y la función $ir - A()$ se muestran en la figura. Los conjuntos de ítems son los estados, y la función ir-A() son las transiciones.



Algoritmo de análisis sintáctico LR.

Un analizador sintáctico LR consiste en:

- Una entrada
- Una salida
- Una pila
- Un programa controlador
- Una tabla de análisis sintáctico con dos partes: **ACCIÓN** e **IR-A.**



El programa controlador es igual para todos los analizadores sintácticos LR.

La tabla de análisis sintáctico cambia de un analizador a otro.

El programa lee caracteres de un búfer de entrada, uno por vez.

La pila contiene una secuencia de estados, $s_0, s_1, \dots, s_m$, siendo s_m el tope de la pila.

Cada estado, excepto el estado inicial, tiene un símbolo gramatical único asociado con él, es decir la transición $IR - A(I_i, X) = I_j$ asocia X al estado j.(A la inversa, no es cierto: puede haber más de un estado asociado al mismo X).

Estructura de la Tabla de Análisis Sintáctico LR.

Consiste en dos partes:

- Una función de acción de análisis sintáctico (llamada **ACCIÓN**)
- Una función **IR-A (GOTO)**

1. **La función ACCION** recibe como argumentos un estado i y un terminal a, o $.
   $ACCION[i, a]$ puede ser:
   (a) **Shift** -Desplazar j (donde j es un estado).
   Se desplaza la entrada a a la pila, usando el estado j para representar a.
   (b) **Reduce** - Reducir $A \rightarrow \beta$
   Reemplaza β en la parte superior de la pila, por A.
   (c) **Accept** - Aceptar.
   El analizador acepta la entrada y termina el análisis.
   (d) **Error**.
   El analizador descubre un error en su entrada.
2. La función $\boldsymbol{IR - A}()$.
   Si $IR - A([I_i, A] = I_j$, entonces $IR - A$ también asigna un estado i y un no terminal A, al estado j.

Algoritmo de análisis sintáctico LR.

ENTRADA:

Una cadena ω.

Una tabla de análisis sintáctico LR con las funciones $ACCION$ e $ir - A$ para la gramática G.

SALIDA:

Si $\omega \in L(G)$, los pasos de reducción de un análisis sintáctico ascendente para ω.

Si $\omega \notin L(G)$, señala error.

PASOS:

Configuración inicial:

- s_0 en la pila
- $\omega$$ en el buffer de entrada.

PILA	ENTRADA
$0	$\omega$$

Sea a el primer símbolo de $\omega$$:

Mientras(1){
Sea s el estado en el tope de la pila.
Según $ACCION[s, a]$:
Caso **desplazar** t:
Meter t en la pila
Consumir a.
Caso **reducir** $A \rightarrow \beta$:
Sacar $|\beta| = r$ símbolos de la pila
Sea t el tope de la pila, apilar $ir - A[t, A] = q$
Enviar de salida la producción $A \rightarrow \beta$
Caso **aceptar**:
Break;
Caso **error**:
Manejar Error()
}

Todos los analizadores sintácticos LR se comportan de esta manera; la única diferencia entre un analizador sintáctico LR y otro es la información en los campos ACCION e ir_A de la tabla de análisis sintáctico.

Ejemplo: Sea la gramática $G =< \{E', E, T, F\}, \{+, *, (,), id\}, E, P >$con producciones

$0. E' \rightarrow E$
$1. E \rightarrow E + T$
$2. E \rightarrow T$
$3. T \rightarrow T * F$
$4. T \rightarrow F$
$5. F \rightarrow (E)$
$6. F \rightarrow \boldsymbol{id}$

La tabla de análisis sintáctico es:

Estado	Acción						ir_A		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Donde:

- s_i=desplazar (shift) y apilar al estado i
- r_j=reducir con la producción j
- Acc = aceptar
- Casillero en blanco: error.

El análisis sintáctico de la cadena $id * id + id$ es:

	PILA	SIMBOLOS	ENTRADA	ACCIÓN
(1)	0		$id * id + id\$$	Desplazar
(2)	0 id 5	id	$* id + id\$$	Reducir F→**id**
(3)	0 F 3	F	$* id + id\$$	Reducir T→**F**
(4)	0 T 2	T	$* id + id\$$	Desplazar
(5)	0 T 2 $*$	$T *$	$id + id\$$	Desplazar
(6)	0 T 2 $*$ 7 id 5	$T * id$	$+ id\$$	Reducir F→**id**
(7)	0 T 2 $*$ 7 F 10	$T * F$	$+ id\$$	Reducir T→T*F
(8)	0 T 2	T	$+ id\$$	Reducir E→T
(9)	0 E 1	E	$+ id\$$	Desplazar
(10)	0 E 1 $+$ 6	$E +$	$id\$$	Desplazar
(11)	0 E 1 $+$ 6 id 5	$E + id$	$\$$	Reducir F→**id**
(12)	0 E 1 $+$ 6 F 3	$E + F$	$\$$	Reducir T→**F**
(13)	0 E 1 $+$ 6 T 9	$E + T$	$\$$	Reducir E→E+T
(14)	0 E 1	E	$\$$	Aceptar

Si bien no es necesario apilar el símbolo de la entrada, porque la transición determina cuál es, se puede hacer y queda más claro. Si no se apilan, queda así:

	PILA	SÍMBOLOS	ENTRADA	ACCIÓN
(1)	0		$\mathbf{id} * \mathbf{id} + \mathbf{id}\,\$$	desplazar
(2)	0 5	$\mathbf{id}$	$* \mathbf{id} + \mathbf{id}\,\$$	reducir mediante $F \to \mathbf{id}$
(3)	0 3	F	$* \mathbf{id} + \mathbf{id}\,\$$	reducir mediante $T \to F$
(4)	0 2	T	$* \mathbf{id} + \mathbf{id}\,\$$	desplazar
(5)	0 2 7	$T *$	$\mathbf{id} + \mathbf{id}\,\$$	desplazar
(6)	0 2 7 5	$T * \mathbf{id}$	$+ \mathbf{id}\,\$$	reducir mediante $F \to \mathbf{id}$
(7)	0 2 7 10	$T * F$	$+ \mathbf{id}\,\$$	reducir mediante $T \to T * F$
(8)	0 2	T	$+ \mathbf{id}\,\$$	reducir mediante $E \to T$
(9)	0 1	E	$+ \mathbf{id}\,\$$	desplazar
(10)	0 1 6	$E +$	$\mathbf{id}\,\$$	desplazar
(11)	0 1 6 5	$E + \mathbf{id}$	$\$$	reducir mediante $F \to \mathbf{id}$
(12)	0 1 6 3	$E + F$	$\$$	reducir mediante $T \to F$
(13)	0 1 6 9	$E + T$	$\$$	reducir mediante $E \to E + T$
(14)	0 1	E	$\$$	aceptar

Tabla de Análisis Sintáctico Simple LR. (SLR)

Algoritmo de construcción de la tabla SLR(1) para una gramática G.

ENTRADA:

Una gramática aumentada G'.

SALIDA:

Las funciones ACCION e Ir-A para G' de la tabla de análisis sintáctico SLR.

PASOS:

1. Construir $C = \{I_0, I_1, \ldots, I_n\}$, la colección de conjuntos de ítems LR(0) para G'

2. El estado i se construye a partir de I_i . Las acciones de análisis sintáctico para el estado i se determinan de la siguiente forma:
   (a) Si $a \in \Sigma \wedge [A \to \alpha \bullet a\beta] \in I_i \wedge ir - A(I_i, a) = I_j$, establecer $ACCION[i, a] = desplazar\ j$.
   (b) Si $[A \to \alpha \bullet] \in I_i$, establecer $ACCION[i, a] = reducir\ A \to \alpha, \forall a \in SIGUIENTES(A)$; aquí A puede o no ser S'
   (c) Si $[S' \to S \bullet] \in I_i$, establecer $ACCION[i, \$] = aceptar$

Si resulta cualquier acción conflictiva debido a las reglas anteriores, la gramática no es SLR(1) y no se produce el analizador sintáctico.

3. $\forall\ A, si\ ir - A(I_i, A) = I_j$, entonces $ir - A[i, A] = j$.
4. Todas las entradas que no estén definidas por las reglas (2) y (3) se dejan como "error".
5. El estado inicial del analizador sintáctico es el que construyó a partir del conjunto de elementos que contienen $[S' \to \bullet S]$

Ejemplo: Sea la gramática $G =< \{E', E, T, F\}, \{+, *, (,), id\}, E, P >$con producciones

$0. E' \to E$
$1. E \to E + T$
$2. E \to T$
$3. T \to T * F$
$4. T \to F$
$5. F \to (E)$
$6. F \to \boldsymbol{id}$

Los conjuntos de SIGUIENTES son:

$SIGUIENTES(E') = \{\$\}$

$SIGUIENTES(E) = \{\$, +,)\}$

$SIGUIENTES(T) = \{\$, +,), *\}$

$SIGUIENTES(F) = \{\$, +,), *\}$

Estado	Acción						ir_A		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Toda gramática SLR(1) es NO ambigua.
Pero no toda gramática NO ambigua es SLR(1).

Análizadores Sintácticos más potentes.

Hay dos métodos distintos:

1. El ***método LR Canónico*** (o LR, directamente)
   Usa ítems llamados LR(1).
2. El ***método LALR*** (Look Ahead LR)
   Tiene menos estados que los basados en ítems LR(1). Permiten manejar más gramáticas que con el método SLR y con tablas no tan grandes.

Ítems LR(1)

Un ***item LR(1)*** (o ***elemento LR(1)***) tiene la forma general: $[A \to \alpha \bullet \beta, t]$, en donde $A \to \alpha\beta$ es una producción y t es un terminal o bien es el delimitador de cadena \$

El símbolo terminal t, de longitud 1, da nombre al método (LR(1)).

Construcción del conjunto de ítems LR(1)

ENTRADA:
Una gramática aumentada G'.

SALIDA:
Conjuntos de ítems LR(1)

PASOS:

$$C = CLAUSURA(\{[S' \to \bullet S, \$]\})$$

Repetir:

Para cada conjunto de ítems I en C:

Para cada símbolo gramatical X:

Si $IR - A(I,X) \neq \emptyset \wedge IR - A(I,X) \notin C$:

Agregar $IR - A(I,X)$ a C.

Hasta que no se agreguen nuevos conjuntos de ítems a C.

Donde, el algoritmo de CLAUSURA es:

ENTRADA:
Un conjunto I de ítems.

SALIDA:
Un conjunto J de ítems,J= CLAUSURA(I)

PASOS:

J=I

Repetir

para cada ítem $[A \to \alpha \bullet \boldsymbol{B\beta}, \boldsymbol{t}]$ en J:

para cada $B \to \gamma$ en G':

para cada terminal $b \in PRIMEROS(\beta t)$:

agregar $[B \to \bullet \boldsymbol{\gamma}, \boldsymbol{b}]$ ***a J*:**

hasta que no se agreguen más elementos a J.

retornar J.

Y el algoritmo de ir-A es:

ENTRADA:
Un conjunto I de ítems.
Un símbolo gramatical X.

SALIDA:
Un conjunto J de ítems,J= CLAUSURA (I)

PASOS:

J=∅

Para cada item $[A \to \alpha \bullet X\beta, t] \in I$

Agregar item $[A \to \alpha X \bullet \beta, t]$ a J

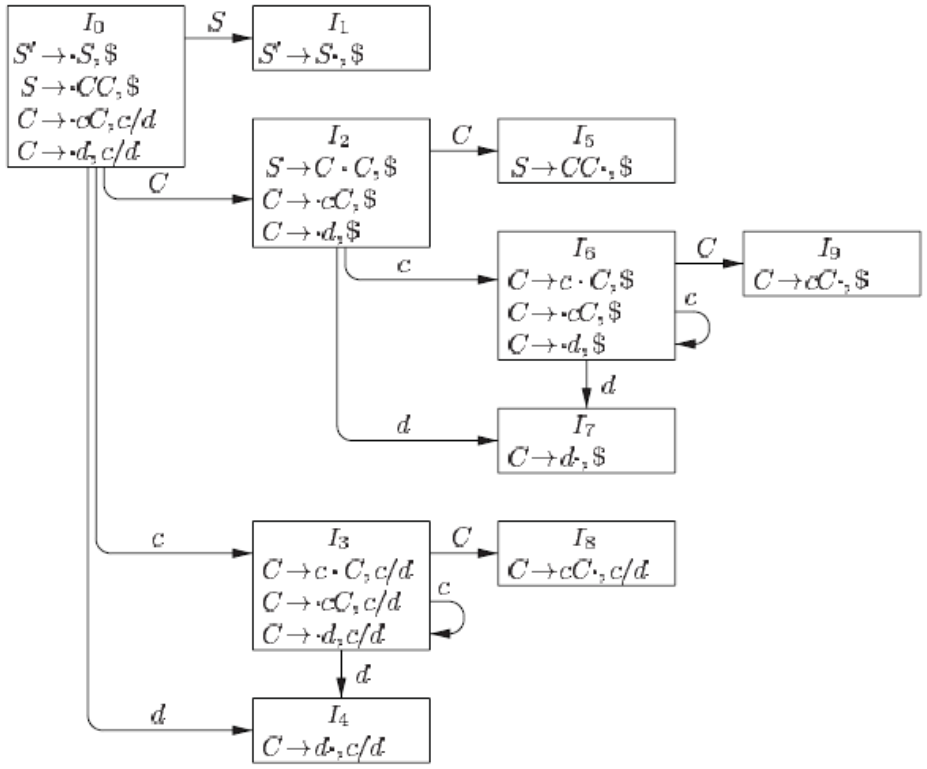
Ejemplo: Para la gramática aumentada con producciones:

$$\begin{cases} S' \to S \\ S \to CC \\ C \to cC \mid d \end{cases}$$

Los ítems LR(1) se empiezan a armar así:

$I_0 = \{$
$[S \rightarrow \bullet \boldsymbol{S}, \$]$
$[S \rightarrow \bullet \boldsymbol{CC}, \$]$
$[C \rightarrow \bullet \boldsymbol{cC}, \boldsymbol{c/d}]$
$[C \rightarrow \bullet \boldsymbol{d}, \boldsymbol{c/d}]$
$\}$

Los ítems LR(1) y la función ir-A quedan:



Algoritmo de construcción de la tabla LR(1) para una gramática G.

ENTRADA:

Una gramática aumentada G'.

SALIDA:

Las funciones ACCION e Ir-A para G' de la tabla de análisis sintáctico LR.

PASOS:

1. Construir $C = \{I_0, I_1, \ldots, I_n\}$, la colección de conjuntos de ítems LR(1) para G'
2. El estado i se construye a partir de I_i . Las acciones de análisis sintáctico para el estado i se determinan de la siguiente forma:
   (a) Si $a \in \Sigma \wedge [A \rightarrow \alpha \bullet a\beta, b] \in I_i \wedge ir - A(I_i, a) = I_j$, establecer $ACCION[i, a] = desplazar\ j$.
   (b) Si $[A \rightarrow \alpha \bullet, a] \in I_i$, A ≠ S' establecer $ACCION[i, a] = reducir\ A \rightarrow \alpha$
   (c) Si $[S' \rightarrow S \bullet, \$] \in I_i$, establecer $ACCION[i, \$] = aceptar$

   Si resulta cualquier acción conflictiva debido a las reglas anteriores, la gramática no es LR(1) y no se produce el analizador sintáctico.
3. $\forall\ A, si\ ir - A(I_i, A) = I_j$, entonces $ir - A[i, A] = j$.
4. Todas las entradas que no estén definidas por las reglas (2) y (3) se dejan como "error".
5. El estado inicial del analizador sintáctico es el que construyó a partir del conjunto de elementos que contienen $[S' \rightarrow \bullet S, \$]$

Ejemplo: Para la gramática aumentada con producciones:

$$\{$$
$$S' \to S$$
$$S \to CC$$
$$C \to cC|d$$
$$\}$$

La tabla resulta:

Estado	Acción			ir_A	
	c	d	\$	S	C
0	s3	s4		1	2
1			acc		
2	s6	s7			5
3	s3	s4			8
4	r3	r3			
5			r1		
6	s6	s7			9
7			r3		
8	r2	r2			
9			r2		

Núcleo de un Ítem LR(1)

El núcleo (core) de un ítem LR(1) $[A \to \alpha \bullet \beta, t]$ es su primer componente $A \to \alpha\beta$

Si se tienen más de un item con el mismo núcleo pueden unirse en un solo conjunto de elementos.

Por ejemplo:

Si $I_3 = \{$
$[C \to c \bullet C, c|d]$
$[C \to \bullet cC, c|d]$
$[C \to \bullet d, c|d]$
$\}$

Y $I_6 = \{$
$[C \to c \bullet C, \$]$
$[C \to \bullet cC, \$]$
$C \to \bullet . d, \$]$
$\}$

La unión quedaría:

$I_{36} = \{[C \to c \bullet C, c|d|\$]$
$[C \to \bullet cC, c|d|\$]$
$[C \to \bullet d, c|d|\$]$
$\}$

Algoritmo de construcción de la tabla LALR para una gramática G

Entrada:

Una gramática aumentada G'.

Salida:

Las funciones ACCION e Ir-A para G' de la tabla de análisis sintáctico LALR.

Pasos:

1. Construir $C = \{I_0, I_1, \dots, I_n\}$, la colección de conjuntos de ítems LR(1) para G'
2. Para cada núcleo presente entre los conjuntos de ítems LR(1), reemplazar todos los ítems que tengan ese mismo núcleo por su unión.
3. Sea $C' = \{J_0, J, \dots, J_n\}$, la colección de conjuntos de ítems LR(1) resultante del paso 2. Las acciones de análisis sintáctico para el estado i se construyen a partir de J_i de la misma manera que en el algoritmo para la tabla LR(1). Si hay un conflicto, la gramática no es LALR(1)
4. Si J es la unión de uno o más conjuntos de ítems LR(1),
   es decir $J = I_1 \cup I_2 \cup \dots I_k$,
   entonces los núcleos de $IR - A(I_1, X),\ IR - A(I_2, X) \dots\ IR - A(I_k, X)$ son los mismos, entonces $IR - A(J, X) = K$ siendo K la unión de todos los conjuntos de ítems que tienen el mismo núcleo que $IR - A(I_1, X)$

Ejemplo: Para la gramática aumentada con producciones:

$$
\begin{cases}
S' \to S \\
S \to CC \\
C \to cC \mid d
\end{cases}
$$

Estado	Acción			ir_A	
	c	d	\$	S	C
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

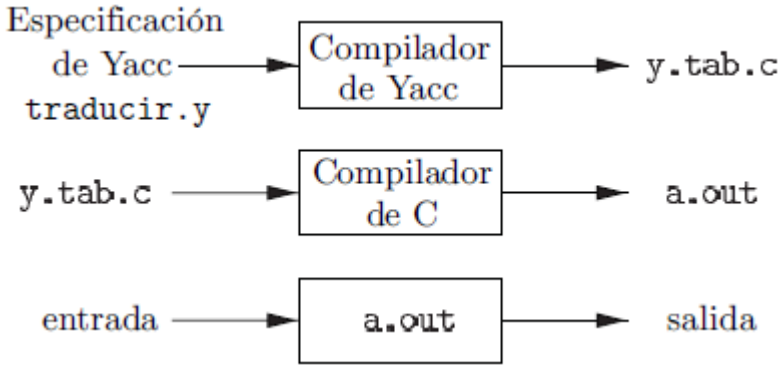
Hay varias modificaciones que se pueden realizar al algoritmo para hacerlo más eficiente.

Generador de Análizadores Sintácticos.

Un generador de analizadores sintácticos es un programa que, dada una especificación de una gramática, permite generar otro programa para analizar y traducir un archivo que corresponda a esa gramática.

En el caso de Yacc ("Yet Another Compiler-Compiler), que es el generador más conocido y usado, se puede dar como entrada una especificación de una gramática, y Yacc producirá un programa en C (**`y.tab.c`**) que usando el método LALR permitirá hacer la traducción de un programa o archivo que se corresponda con la gramática.

- El archivo **`traducir.y`** contiene la gramática de, supongamos, un lenguaje L.
- El programa **`y.tab.c`** es una representación de un analizador LALR escrito en C.
- Una vez compilado **`y.tab.c`** se obtiene un ejecutable **`a.out`**
- Ese archivo ejecutable **`a.out`** permitirá traducir una entrada escrita en lenguaje L. Es decir, **`a.out`** es un compilador/traductor.



[i] Bibliografía usada:
Aho, Sethi y Ullman. Compilers: Principles, Techniques and Tools. Second Editions.
IMPORTANTE: Evitar la edición en castellano porque tiene algunas erratas y expresiones mal traducidas.