

MARZO, 2022

# ANÁLISIS SINTÁCTICO



# Tipos de análisis sintáctico

## Forma Sentencial

Es cualquier secuencia de terminales y no terminales obtenida mediante derivaciones a partir del axioma inicial.

De la forma de construir el árbol sintáctico se desprenden dos tipos o clases de analizadores sintácticos. Pueden ser descendentes o ascendente.

# Tipos de análisis sintáctico

## ***Descendentes***

Parten del axioma inicial, y van efectuando derivaciones a izquierda hasta obtener la secuencia de derivaciones que reconoce a la sentencia.

Pueden ser:

Con retroceso.

Con recursión.

LL(1)

# Tipos de análisis sintáctico

## ***Ascendentes***

Parten de la sentencia de entrada, y van aplicando reglas de producción hacia atrás (desde el consecuente hasta el antecedente), hasta llegar al axioma inicial.

Pueden ser:

Con retroceso.

LR(1)

## Análisis descendente con retroceso.

*Objetivo* : El método parte del axioma inicial y aplica todas las posibles reglas al no terminal más a la izquierda.

*Ejemplo*: Utilizaremos la siguiente gramática (No recursiva por la izquierda)

1.  $E \rightarrow T + E$

2.  $E \rightarrow T$

3.  $T \rightarrow F * T$

4.  $T \rightarrow F$

5.  $F \rightarrow a$

6.  $F \rightarrow b$

7.  $F \rightarrow (E)$

para reconocer la cadena de entrada:  $(a + b) * a + b$

# Análisis descendente con retroceso.

*Ejemplo:* Utilizaremos la siguiente gramática (No recursiva por la izquierda)

1.  $E \rightarrow T + E$

2.  $E \rightarrow T$

3.  $T \rightarrow F * T$

4.  $T \rightarrow F$

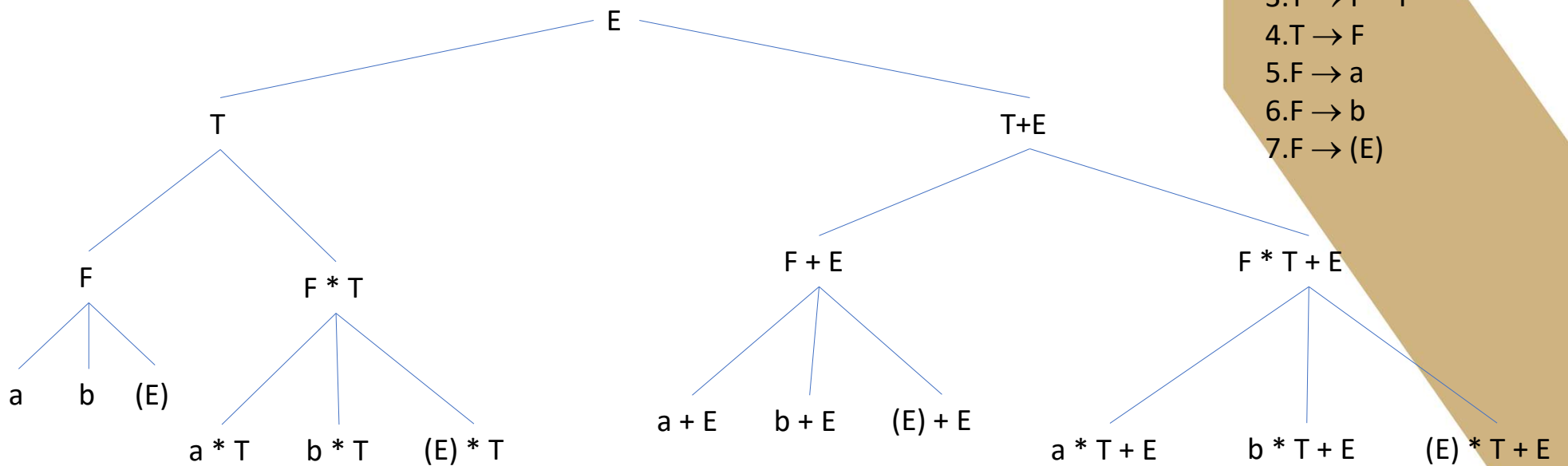
5.  $F \rightarrow a$

6.  $F \rightarrow b$

7.  $F \rightarrow (E)$

para reconocer la cadena de entrada:  $(a + b) * a + b$

# Análisis descendente.



1.  $E \rightarrow T + E$
2.  $E \rightarrow T$
3.  $T \rightarrow F * T$
4.  $T \rightarrow F$
5.  $F \rightarrow a$
6.  $F \rightarrow b$
7.  $F \rightarrow (E)$

## Análisis descendente con retroceso.

Mediante este árbol se pueden derivar todas las posibles sentencias reconocibles por esta gramática y el objetivo de este algoritmo es hacer una búsqueda en este árbol de la rama que culmine en la sentencia a reconocer. El mecanismo funciona mediante una búsqueda primero en profundidad.

Mira si todos los tokens a la izquierda de un No Terminal coincide con la cabeza de la secuencia a reconocer.

En todo el árbol de derivaciones, se pretende profundizar por cada rama hasta llegar a encontrar una forma sentencial que no puede coincidir con lo que se busca, en cuyo caso se desecha, o que coincide con lo buscado, momento en que se acepta la sentencia. Si por ninguna rama se puede reconocer, se rechaza la sentencia.



# Algoritmo

Sea  $k \in N$  el no terminal más a la izquierda de la forma sentencial.

Sea  $\tau \in T^*$  la secuencia de tokens en la izquierda de  $k$ .

Ensayar (forma\_sentencial, entrada)

for  $\{p_i \in P / p_i \equiv k \rightarrow \alpha\}$

    forma\_sentencial' = Aplicar( $p_i$ ,  $k$ , forma\_sentencial)

    if  $\tau' = \text{Parte\_izquierda}(\text{entrada})$

        if forma\_sentencial == entrada

            ¡ Sentencia reconocida !

        else

            Ensayar (forma\_sentencial', entrada)

        endif

    endif

endfor;

Ensayar (S, cadena a reconocer)

if not ¡ Sentencia reconocida !

    ¡¡ Sentencia **no** reconocida !!

endif;



## Problemas

Este método no funciona con gramáticas recursivas a la izquierda, ya que puede ocurrir que entre en un bucle infinito.

No existen muchos analizadores sintácticos con retroceso. En parte, porque casi nunca se necesita el retroceso para analizar sintácticamente las construcciones de los lenguajes de programación.


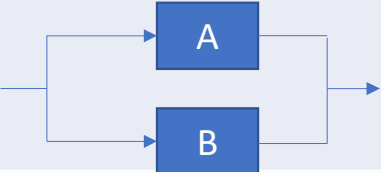
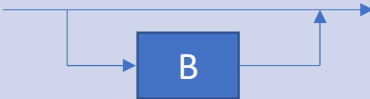

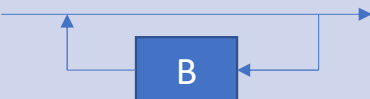
En casos como el análisis sintáctico del lenguaje natural, el retroceso tampoco es muy eficiente, y se prefieren otros métodos.

## Análisis descendente con recursión. Diagramas de Conway.

Una gramática de contexto libre puede expresar un lenguaje al igual que puede hacerlo la notación BNF, y los diagramas de Conway.

*Definición:* Un diagrama de Conway es un grafo dirigido donde los elementos no terminales aparecen como rectángulos, y los terminales como círculos.

Para demostrar que permite representar las mismas gramáticas que la BNF, se hace por inducción sobre las operaciones básicas de BNF:

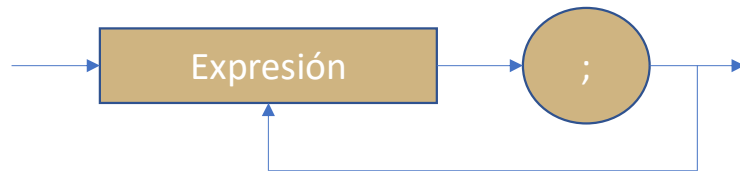
Operación	BNF	Diagrama de Conway
Yuxtaposición	$AB$	
Opción	$A   B$	
	$\varepsilon   B$	
Repetición	1 o más veces $\{B\}$	
	0 o más veces $[B]$	

# Análisis descendente con recursión.

## Diagramas de Conway.

De esta forma todos los posibles caminos desde el inicio del grafo hasta el final, representan formas sentenciales válidas.

En todo diagrama de Conway hay un origen y un destino.



En este caso se considera al ';' como un token de seguridad, lo que permite hacer una recuperación de errores mediante el método **panic mode**.

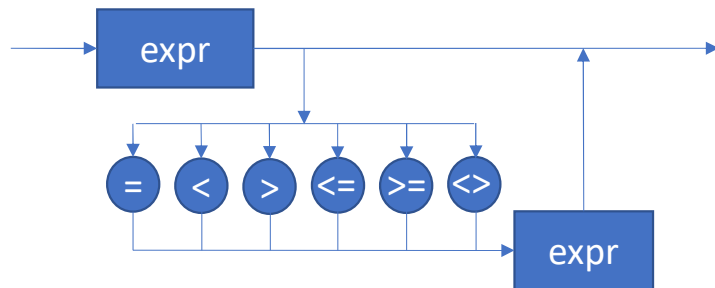
```
main ( ) {  
    get_token ( );  
    do {  
        expresión ( );  
        while (token != PUNTOYCOMA) {  
            !Error en expresión;  
            get_token ( );  
        };  
        get_token ( );  
    } while (token != EOF);  
};
```

# Análisis descendente con recursión.

## Diagramas de Conway.

El analizador sintáctico pide al lexicográfico tokens a través de `get_token ( )`, y que el lexicográfico deja el token actual en la variable global `token`.

Antes de entrar a una función, en `token` debemos tener el token de lookahead, que esa función necesita consultar.



```
expresión ( ){  
    expr ( );  
    if ((token == IGUAL)|| (token == ME)|| (token == MEI)||  
        (token == DIST)|| (token == MAI)|| (token == MA))  
        get_token ( ) ;  
        expr ( ) ;  
    }  
}
```

## Análisis descendente de gramáticas LL(1)

Una gramática LL(1) es aquella en la que su tabla de chequeo de sintaxis no posee entradas múltiples, o sea, es suficiente con examinar sólo un símbolo a la entrada, para saber qué regla aplicar. Toda gramática reconocible mediante el método de los diagramas de Conway es LL(1)

El método consiste en seguir un algoritmo partiendo de:

- La cadena a reconocer, junto con un apuntador, que nos indica cual es el token actual.
- Una pila de símbolos ( terminales y no terminales)
- Una tabla asociada de forma unívoca a una gramática.
- La cadena de entrada acabará en el símbolo \$, que consideramos como si fuese un EOF.

## Análisis descendente de gramáticas LL(1)

Sea  $X$  el elemento encima de la pila, y  $a$ , el apuntado en la entrada. El algoritmo consiste en:

- 1.- Si  $X = a = \$$  entonces ACEPTAR.
- 2.- Si  $X = a \neq \$$  entonces
  - se quita  $X$  de la pila
  - y se avanza el apuntador.
- 3.- Si  $X \in T$  y  $X \neq a$  entonces RECHAZAR.
- 4.- Si  $X \in N$  entonces consultar la entrada  $M[X,a]$  de la tabla:
  - $M[X,a]$  es vacía : RECHAZAR.
  - $M[X,a]$  no es vacía, se quita a  $X$  de la pila y se inserta el consecuente en orden inverso.



## Análisis Ascendente

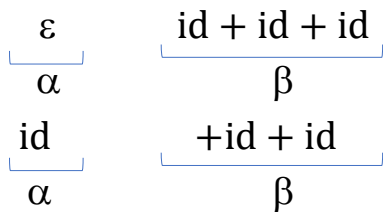
Árbol sintáctico de abajo hacia arriba, lo cual disminuye el número de reglas mal aplicadas con respecto al caso descendente.

Tanto si hay retroceso como si no, en un momento dado, la cadena de entrada estará dividida en dos partes  $\alpha$  y  $\beta$ .

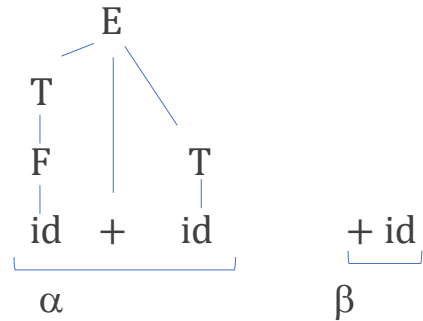
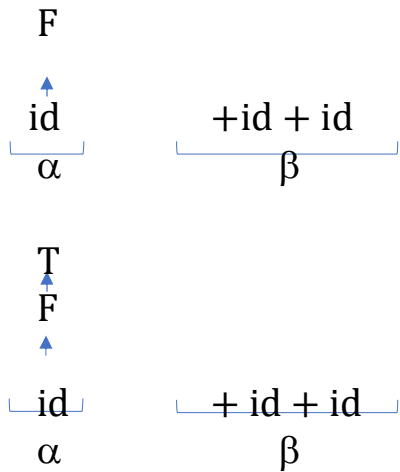
$\beta$ : El trozo de la cadena de entrada (secuencia de tokens) por reconocer. Coincidirá siempre con algún trozo de la parte derecha de la cadena de entrada:  $\beta \in T^*$ . Vamos consumiendo tokens, y todos los tokens que nos queden por consumir constituyen  $\beta$ .

$\alpha$ : Coincidirá siempre con el resto de la cadena de entrada, trozo al que se habrán aplicado algunas reglas de producción,  $\alpha \in (N \cup T)^*$  en sentido inverso.

# Análisis Ascendente, ejemplo



Ahora  $\alpha$  es F



En un momento dado, el analizador sintáctico se encuentra en con un par  $\alpha, \beta$  concreto, al que se llama configuración.

## Análisis Ascendente

El analizador sintáctico para poder trabajar puede realizar una de las cuatro operaciones:

*Aceptar* : Cadena reconocida.

*Rechazar* : La entrada no es válida.

*Reducir* : Aplicar una regla de producción a los elementos de  $\alpha$ .

*Desplazar* : Se desplaza el terminal más de la izquierda de  $\beta$  a la derecha de  $\alpha$ .

## Análisis Ascendente

- ✓ Mediante reducciones y desplazamientos, tenemos que llegar a aceptar o rechazar la cadena de entrada.
- ✓ Antes de hacer los desplazamientos tenemos que hacerles todas las reducciones posibles a  $\alpha$ .
- ✓ Cuando  $\alpha$  es el axioma inicial y  $\beta$  es la tira nula, se acepta la cadena de entrada.
- ✓ Cuando  $\beta$  no es la tira nula o  $\alpha$  no es el axioma inicial y no se puede aplicar ninguna regla, entonces se rechaza la cadena de entrada.

## Análisis Ascendente

Cuando se da cuenta que llega a una situación en la que no puede continuar, entonces vuelve atrás deshaciendo todos los cambios.

En el análisis con retroceso no se permiten las reglas  $\varepsilon$ , puesto que estas se podrán aplicar de forma indefinida.

```
Ensayar( $\varepsilon$ , cadena a reconocer);  
if NOT se ha aceptado then  
    RECHAZAR  
endif;
```

```
Ensayar ( $\alpha, \beta$ )  
    for  $\pi_i \in P$  hacer  
        if consecuente ( $\pi_i$ ) == cola( $\alpha$ )  
             $\alpha' =$  Reducir la cola de  $\alpha$  por  $\pi_i$   
            if ( $\alpha' ==$  Axioma inicial) AND ( $\beta == \varepsilon$ )  
                ACEPTAR  
            else  
                Ensayar ( $\alpha', \beta$ )  
            endif  
        endif  
    endfor  
    if ( $\beta \neq \varepsilon$ )  
         $\alpha', \beta' =$  Desplazar de  $\beta$  a  $\alpha$ .  
        Ensayar ( $\alpha', \beta'$ )  
    endif  
endEnsayar
```

## Análisis Ascendente con retroceso,

ejemplo, no se permiten reglas  $\varepsilon$

1.  $E \rightarrow T + E$

2.  $E \rightarrow T$

3.  $T \rightarrow F * T$

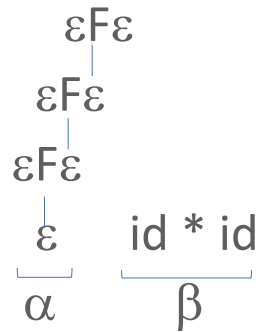
4.  $T \rightarrow F$

5.  $F \rightarrow (E)$

6.  $F \rightarrow \text{id}$

7.  $F \rightarrow \text{num}$

8.  $F \rightarrow \varepsilon$



No puede aparecer  $\varepsilon$  en una gramática ascendente con retroceso porque da lugar a recursión infinita.

# Análisis Ascendente con retroceso,

Reconocer  $a^*a \equiv id^*id$

1.  $E \rightarrow T + E$

2.  $E \rightarrow T$

3.  $T \rightarrow F * T$

4.  $T \rightarrow F$

5.  $F \rightarrow (E)$

6.  $F \rightarrow id$

7.  $F \rightarrow num$

Pila	$\alpha$	$\beta$	Acción
-	$\epsilon$	$id^*id$	Desplazar
-	$Id$	$*id$	$F \rightarrow id$
6	$F$	$*id$	$T \rightarrow F$
6-4	$T$	$*id$	$E \rightarrow T$
6-4-2	$E$	$*id$	Desplazar
6-4-2	$E *$	$Id$	Desplazar
6-4-2	$E * id$	$\epsilon$	$F \rightarrow id$
6-4-2-6	$E * F$	$\epsilon$	$T \rightarrow F$
6-4-2-6-4	$E * T$	$\epsilon$	$E \rightarrow T$
6-4-2-6-4-2	$E * E$	$\epsilon$	Retroceso
6-4	$T$	$*id$	Desplazar
6-4	$T *$	$Id$	Desplazar
6-4	$T * id$	$\epsilon$	$F \rightarrow id$
6-4-6	$T * F$	$\epsilon$	$T \rightarrow T * F$
6-4-6-3	$T$	$\epsilon$	$E \rightarrow T$
6-4-6-3-2	$E$	$\epsilon$	Aceptar

## Análisis Ascendente con retroceso, Analizador LR

Vamos a analizar una técnica eficiente de análisis sintáctico ascendente que se puede utilizar para analizar una amplia clase de gramáticas de contexto libre.

La técnica se denomina análisis sintáctico **LR(k)**; la “L” es por el examen de la entrada de izquierda a derecha, la “R” por construir una derivación por la derecha en orden inverso, y la  $k$  por el número de símbolos de entrada de examen por anticipado utilizados para tomar las decisiones del análisis sintáctico.

Cuando se omite, se asume que  $k$ , es 1.



## Analizador LR, ventajas

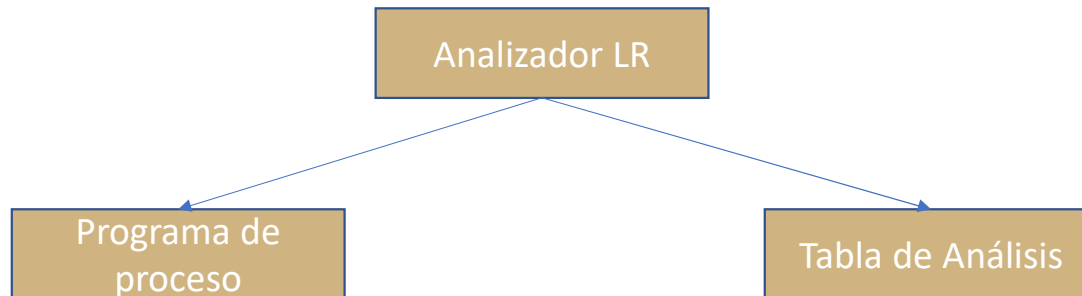
Pueden reconocer la inmensa mayoría de los lenguajes de programación que puedan ser generados mediante gramáticas de contexto-libre.

El método de funcionamiento de estos analizadores posee la ventaja de localizar un error sintáctico en el mismo instante que se produce con lo que se adquiere una gran eficiencia de tiempo de compilación frente a procedimientos menos adecuados como puedan ser los de retroceso.

## Técnicas para construir una tabla de análisis sintáctico LR para una gramática.

- 1.LR sencillo (SLR, en inglés) es el más fácil de implantar, pero el menos poderoso de los tres. Puede que no consiga producir una tabla de análisis sintáctico para algunas gramáticas que otros métodos si consiguen.
- 2.LR canónico, es el más poderoso y costoso.
- 3.LR con examen por anticipado (LALR, en inglés), está entre los otros dos en cuanto a poder y costo. El método LALR funciona con las gramáticas de la mayoría de los lenguajes de programación y, con un poco de esfuerzo, se puede implantar en forma eficiente.

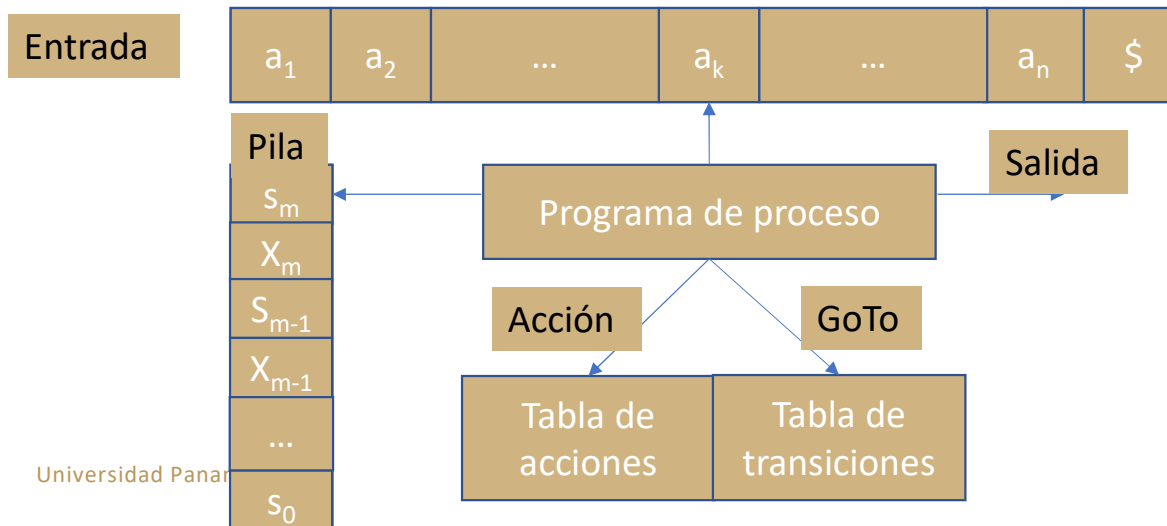
## Técnicas para construir una tabla de análisis sintáctico LR para una gramática.



## Análisis sintáctico LR

El programa de proceso posee un funcionamiento muy simple y permanece invariable de analizador a analizador.

Según sea la gramática a procesar deberá variarse el contenido de la tabla de análisis que es la que identifica plenamente al analizador.



## Análisis sintáctico LR

El contenido de la pila tiene la forma  $s_0 X_1 s_1 X_2 s_2 \dots X_m s_m$  donde el símbolo  $s_m$  se encuentra en la cabeza tal y como se muestra en la figura.

Cada uno de los  $X_i$  son símbolos de la gramática y los  $s_i$  estados del analizador.

Los estados se utilizan para representar toda la información contenida en la pila y situada antes del propio estado.

Es mediante el estado en cabeza de la pila por el que se decide qué reducción ha de efectuarse o bien qué desplazamiento.

Tradicionalmente, una tabla de análisis para un reconocedor LR consta de dos partes, la función GOTO y la función ACCION.

## Funcionamiento del analizador LR

1. Se determina el estado  $s_m$  en cabeza de la pila y el símbolo actual  $a_i$  en el instante de la cadena de entrada.
2. Se consulta en la tabla de análisis la función acción con los parámetros anteriores y que puede dar como resultado.

$$\text{Acción}(s_m, a_i) = \begin{cases} \text{Desplazar } S \\ \text{Reducir } A \rightarrow \beta \\ \text{Aceptar} \\ \text{Rechazar} \end{cases}$$

La función GOTO actúa igualmente con un estado y un símbolo de la gramática produciendo un nuevo estado.

## Configuración de un analizador LR

$$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m, a_i a_{i+1} \dots a_n \$)$$

El primer componente es el contenido actual de la pila, y el segundo la subcadena de entrada que resta por reconocer,  $a_i$  es el símbolo de entrada actual de análisis.

El movimiento del analizador se realiza teniendo en cuenta:

- 1.El símbolo leído  $a_i$ .
- 2.El símbolo en cabeza de la pila  $s_m$ .

## Configuración de un analizador LR

Actuando con la función acción y dependiendo de las cuatro posibles alternativas pueden obtenerse las configuraciones que seguidamente se detallan.

1. Si acción  $(s_m, a_i) = \text{desplazar } s$ .

entonces se introducen en la pila el símbolo actual analizado de la cadena de entrada y en la cabeza de la pila el nuevo estado obtenido mediante la función  $\text{GOTO}(s_m, a_i) = S$ .

La configuración así obtenida es

$$(s_0 X_1 s_1 X_2 s_2 \dots X_m s_m a_i s, a_{i+1} \dots a_n \$)$$

pasando  $s$  a estar situado en cabeza de la pila y  $a_{i+1}$  el siguiente símbolo a explorar en la cinta de entrada.



## Configuración de un analizador LR

2. Si acción  $(s_m, a_i) = \text{reducir } A \rightarrow \beta$

entonces el analizador ejecuta la reducción oportuna donde el nuevo estado en cabeza de la pila se obtiene mediante la función  $\text{GOTO}(s_{m-r}, a_i) = s$  donde  $r$  es precisamente la longitud de la cadena reducida.

Aquí el analizador extrajo primero  $2r$  símbolos de la pila ( $r$  símbolos de estados y  $r$  símbolos de la gramática), exponiendo el estado  $s_{m-r}$ . Luego introdujo  $A$ , el lado izquierdo de la regla de producción, y  $s$ , la entrada de  $\text{GOTO}(s_{m-r}, A)$ , en la pila.

La configuración así obtenida es

$$(s_0 X_1 s_1 X_2 s_2 \dots X_{m-r} s_{m-r} A s, a_i a_{i+1} \dots a_n \$)$$

donde  $s$  es el nuevo estado en cabeza de la pila y no se ha producido variación en la tira de entrada que aun queda por analizar.

## Configuración de un analizador LR

**3.** Si *acción* ( $s_m, a_i$ ) = *aceptar*  
entonces se ha llegado a la finalización en el proceso de reconocimiento y el análisis termina reconociendo la tira de entrada.

## Configuración de un analizador LR

### 4. Si acción $(s_m, a_i) = \text{error}$

entonces es muestra de que el analizador LR ha descubierto un error sintáctico y procederá en consecuencia activando las rutinas de corrección de errores.

Una de las ventajas de este tipo de análisis es que , cuando se produce una acción de error, el token erróneo suele estar al final de  $\alpha$  o al principio de  $\beta$ , lo que permite depurar con cierta facilidad las cadenas de entrada.

La configuración inicial del analizador es

$$(s_0, a_1 a_2 \dots a_n \$)$$

donde  $s_0$  es el estado inicial del reconocedor

## Configuración de un analizador LR

Los sucesivos movimientos se realizan en base a los cuatro puntos anteriores hasta que se acepta la cadena de entrada o bien hasta la aparición de un error.

# Ejemplo: Expresiones aritméticas y tablas LR

Sea la gramática:

1.  $S \rightarrow S + T$

2.  $S \rightarrow T$

3.  $T \rightarrow T * F$

4.  $T \rightarrow F$

5.  $F \rightarrow (S)$

6.  $F \rightarrow id$

1. Di significa desplazar y meter en la pila el estado i,
2. Rj significa reducir por la regla de producción con número j,
3. ACEP significa aceptar,
4. Las entradas en blanco significan un error sintáctico.

Tabla de análisis:

Estado	función ACCIÓN						función GOTO		
	id	+	*	(	)	\$	S	T	F
0	D5			D4			1	2	3
1		D6				ACEP			
2		R2	D7		R2	R2			
3		R4	R4		R4	R4			
4	D5			D4			8	2	3
5		R6	R6		R6	R6			
6	D5			D4				9	3
7	D5			D4					10
8		D6			D11				
9		R1	D7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

$$a * (a + a) \equiv id * (id + id)$$

(suponemos que el estado s0 queda representado por 0).

PASO	PILA	Cadena de entrada
1	0	id * (id + id)\$
2	0 a 5	* (id + id)\$
3	0 F 3	* (id + id)\$
4	0 T 2	* (id + id)\$
5	0 T 2 * 7	(id + id)\$
6	0 T 2 * 7 ( 4	id + id)\$
7	0 T 2 * 7 ( 4 a 5	+ id)\$
8	0 T 2 * 7 ( 4 F 3	+ id)\$
9	0 T 2 * 7 ( 4 T 2	+ id)\$
10	0 T 2 * 7 ( 4 S 8	+ id)\$
11	0 T 2 * 7 ( 4 S 8 + 6	id)\$
12	0 T 2 * 7 ( 4 S 8 + 6 a 5	)\$
13	0 T 2 * 7 ( 4 S 8 + 6 F 3	)\$
14	0 T 2 * 7 ( 4 S 8 + 6 T 9	)\$
15	0 T 2 * 7 ( 4 S 8	)\$
16	0 T 2 * 7 ( 4 S 8 ) 11	\$
17	0 T 2 * 7 F 10	\$
18	0 T 2	\$
19	0 S 1	\$
20	Aceptación de la cadena	

## Análisis sintáctico LR

Puede observarse como el estado que siempre se encuentra en cabeza de la pila señala en todo momento la información necesaria para la reducción, si esto procede.

Dada la estructura de los analizadores LR, con la sola inspección de  $k$  símbolos de la cadena de entrada a la derecha del símbolo actual puede decidirse con toda exactitud cual es el movimiento (reducción, desplazamiento, etc) a realizar.

Es por este motivo por lo que suele denominarse a este tipo de gramáticas como LR( $k$ ). En la práctica casi todos los lenguajes de programación pueden ser analizados mediante gramáticas LR(0) o LR(1).

# Análisis sintáctico LR, conflictos

Ejemplo:

1.  $S \rightarrow aaBdd$

2.  $S \rightarrow aCd$

3.  $B \rightarrow a$

4.  $C \rightarrow aa$

$\begin{array}{c} C \\ \swarrow \downarrow \\ aaad \\ \hline \alpha \end{array}$

$\begin{array}{c} B \\ \downarrow \\ aaad \\ \hline \alpha \end{array}$

Esta gramática reconoce estas dos secuencia de tokens

aaadd

aaad

1.  $S \rightarrow aaBdd$

2.  $S \rightarrow aCd$

3.  $B \rightarrow a$

4.  $C \rightarrow aa$

$S \rightarrow aaadd$   
 $S \rightarrow aaad$

Añadir las secuencias que producen el conflicto como parte de la gramática.