



Procesadores de Lenguajes

Cap. III Análisis Sintáctico



LENGUAJES Y
CIENCIAS DE LA
COMPUTACIÓN
UNIVERSIDAD DE MÁLAGA



ANÁLISIS SINTÁCTICO (AS)

- ¿Qué es analizar sintácticamente?
 - *Encontrar un árbol sintáctico*
- El Análisis Sintáctico, además puede:
 - Recopilar información sobre los tokens, y almacenarla en la tabla de símbolos
 - Realizar la comprobación de tipos “type checking”
 - Generar código intermedio
 - Informar sobre errores detectados
- Tipos de Análisis
 - Descendente
 - Descendente con retroceso
 - Gramáticas LL
 - Ascendente
 - Con retroceso
 - De precedencia
 - Gramáticas LR
- Descendente

Axioma inicial -> Sentencia
Parse Izquierdo

- Ascendente

Sentencia -> Axioma Inicial
Parse Derecho

AS. Pares Izquierdo y Derecho

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

- Parse izquierdo, derivaciones Izquierdas

- Ejemplo de derivaciones cadena $a * (a + b)$

$E \Rightarrow_I^1 T \Rightarrow_I^4 F * T \Rightarrow_I^5 a * T \Rightarrow^3 a * F \Rightarrow^7$

$a * (E) \Rightarrow^2 a * (T + E) \Rightarrow^3 a * (F + E) \Rightarrow^5$

$a * (a + E) \Rightarrow^1 a * (a + T) \Rightarrow^3 a * (a + F) \Rightarrow^6 a * (a + b)$

Parse izdo: 1 – 4 – 5 – 3 – 7 – 2 – 3 – 5 – 1 – 3 – 6

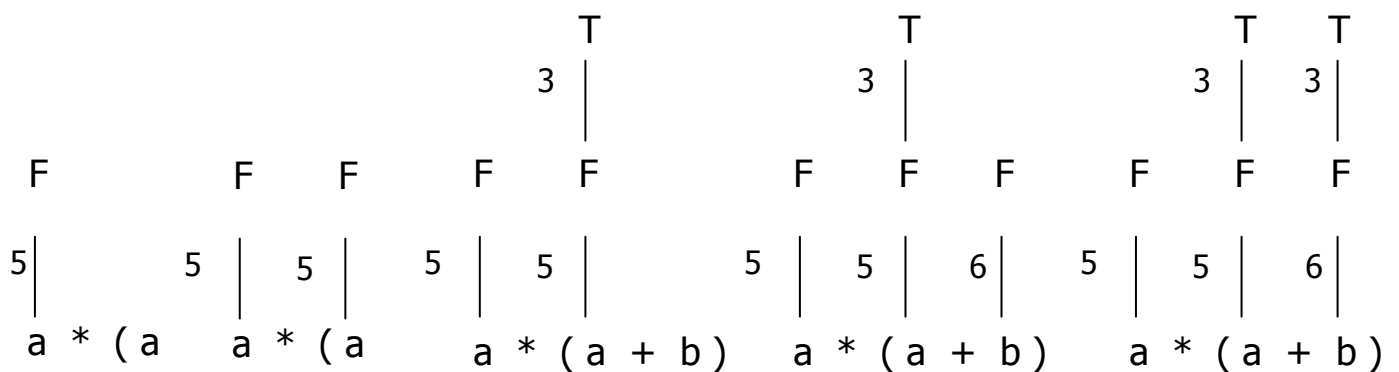
- Parse derecho (derivaciones derecha e invertir)

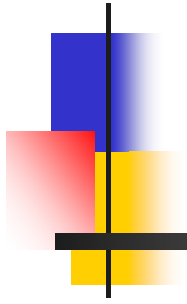
$E \Rightarrow_D^1 T \Rightarrow_D^4 F * T \Rightarrow_D^3 F * F \Rightarrow^7 F * (E) \Rightarrow^2$

$F * (T + E) \Rightarrow^1 F * (T + T) \Rightarrow^3 F * (T + F) \Rightarrow^6$

$F * (T + b) \Rightarrow^3 F * (F + b) \Rightarrow^5 F * (a + b) \Rightarrow^5 a * (a + b)$

Parse dcho: 5 – 5 – 3 – 6 – 3 – 1 – 2 – 7 – 3 – 4 – 1





ANÁLISIS SINTÁCTICO

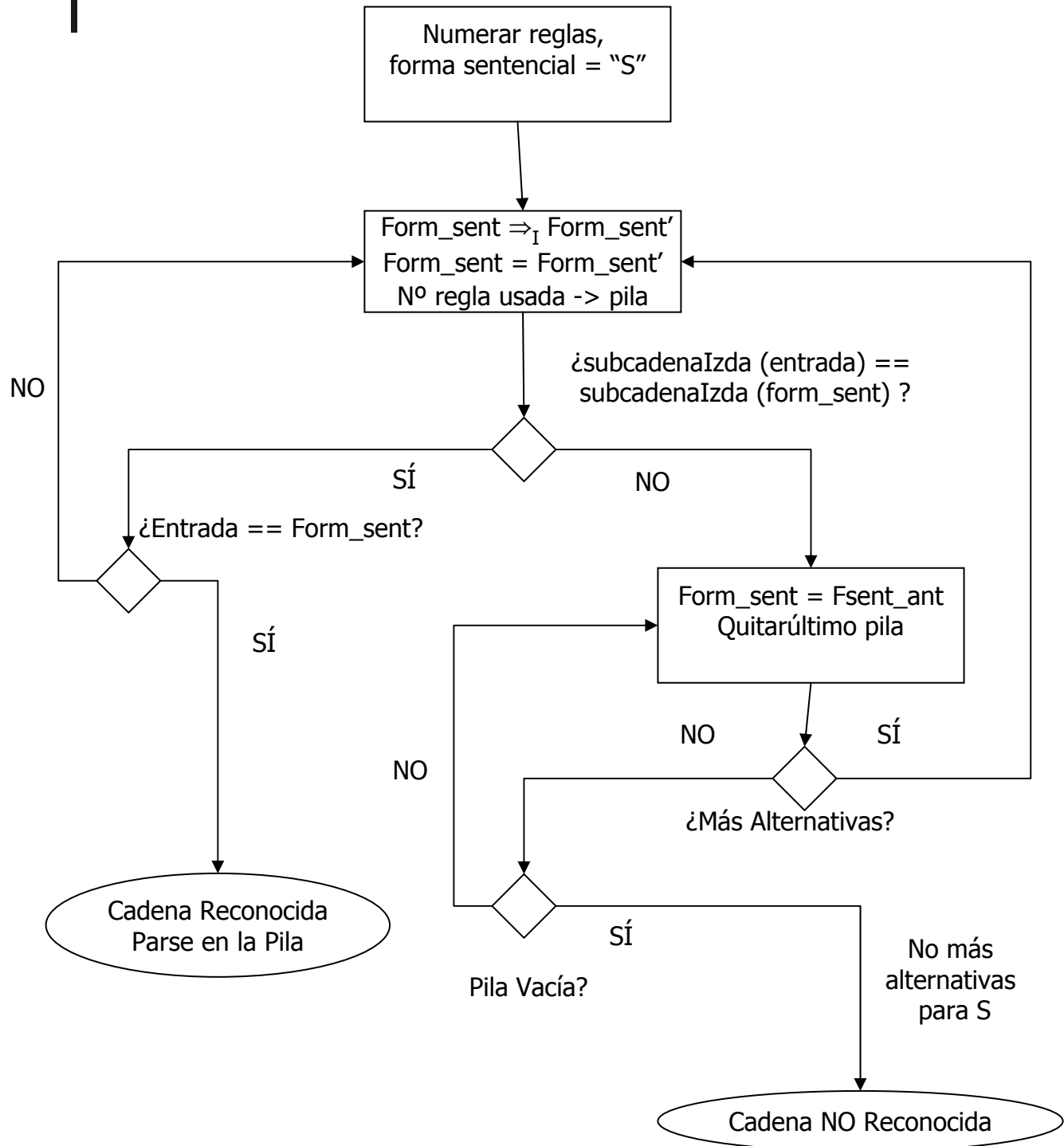
ANÁLISIS DESCENDENTE



Análisis Descendente con retroceso

- Prueba todas las **alternativas** empezando desde el axioma.
- **Problemas** con el retroceso en general:
 - Emplean mucho tiempo
 - Dependen del orden de las reglas
 - No realizan un buen diagnóstico sobre los errores
 - Difícil generación de código si se hace simultáneamente con el análisis sintáctico.

AS. Análisis Descendente con Retroceso



ASDR. Ejemplo

$E \rightarrow T + E \text{ (1) } | T \text{ (2)}$

$T \rightarrow F * T \text{ (3) } | F \text{ (4)}$

$F \rightarrow a \text{ (5) } | b \text{ (6) } | (E) \text{ (7)}$

Analizar: $(a + b) * a + b$

Forma Sentencial	Pila	
E		
T + E	1	
F*T+E	1-3	
a*T+E	1-3-5	Retroceso
F*T+E	1-3	
b*T+E	1-3-6	Retroceso
F*T+E	1-3	
(E)*T+E	1-3-7	
(T+E)*T+E	1-3-7-1	
(F*T+E)*T+E	1-3-7-1-3	
(a*T+E)*T+E	1-3-7-1-3-5	
(F*T+E)*T+E	1-3-7-1-3	
(b*T+E)*T+E	1-3-7-1-3-6	
(F*T+E)*T+E	1-3-7-1-3	
((E)*T+E)*T+E	1-3-7-1-3-7	
(F*T+E)*T+E	1-3-7-1-3	NO hay más alternat.
(T+E)*T+E	1-3-7-1	
(F+E)*T+E	1-3-7-1-4	
(a+E)*T+E	1-3-7-1-4-5	
(a+T+E)*T+E	1-3-7-1-4-5-1	
(a+F*T+E)*T+E	1-3-7-1-4-5-1-3	
(a+a*T+E)*T+E	1-3-7-1-4-5-1-3-5	
(a+F*T+E)*T+E	1-3-7-1-4-5-1-3	
(a+b*T+E)*T+E	1-3-7-1-4-5-1-3-6	
(a+F*T+E)*T+E	1-3-7-1-4-5-1-3	
(a+(E)*T+E)*T+E	1-3-7-1-4-5-1-3-7	
(a+F*T+E)*T+E	1-3-7-1-4-5-1-3	
(a+T+E)*T+E	1-3-7-1-4-5-1
(a+b)*a+b	1-3-7-1-4-5-2-4-6-4-5-2-4-6	

ASDR. Problemas en un ASD con retroceso

■ Recursividad por la izquierda

S \rightarrow aA (1) | bB (2)

A \rightarrow Aa (3) | ϵ (4)

B \rightarrow Bb (5) | ϵ (6)

Analizar aa

Forma Sentencial	Pila
------------------	------

S	-
aA	1-3
aAaa	1-3-3

■ Ciclos por la izquierda

S \rightarrow aA (1)

A \rightarrow Bb (2) | ϵ (3)

B \rightarrow Aa (4) | ϵ (5)

Analizar aa

Forma Sentencial	Pila
------------------	------

S	-
aA	1
aBb	1-2
aAab	1-2-4
aBbab	1-2-4-2
aAabab	1-2-4-2-4

■ ¿Cómo solucionar estos problemas?

1. Eliminar ciclos por la izda.
2. Realizar más comprobaciones para decidir si realizar o no un retroceso.

El número de terminales en la forma sentencial debe ser menor o igual que la longitud de la cadena a reconocer.

Todavía No funcionará

S \rightarrow SA (1) | A (2)

A \rightarrow a (3)

Analizar aa

Forma Sentencial	Pila
------------------	------

S	-
SA	1
SAA	1-1

3. La longitud de la forma sentencial ha de ser menor que la longitud de la cadena a reconocer (la gramática debes ser entonces sin ϵ)



A. Descendente sin Retroceso. Gr. LL(1)

■ LL(1)

- Left: Leer la cadena de izquierda a derecha
- Left: Derivaciones izquierda
- (1): Inspeccionar un símbolo de la entrada

■ Ejem. Gram. LL(1)

S -> cAd (1)

A -> bcB (2)

A -> a (3)

B -> b (4)

Analizar cad

Forma Sentencial	Pila
---------------------	------

S

-

cAd

1

cad

1-3

- Existen gramáticas a las que **no puede hacerse** este tipo de análisis

S -> cAd (1)

A -> aB (2)

A -> a (3)

B -> b (4)

Definiciones ASDSR-LL(1). Cabecera

Cabecera

$a \in (N \cup T)^*$, $\text{Cabecera}(a) = \text{Cab}(a) = \text{First}(a) = \{a \in T / a \Rightarrow^* a\beta, \beta \in (N \cup T)^*\} \cup \{\varepsilon \text{ si } a \Rightarrow^* \varepsilon\}$

Algoritmo para calcular $\text{Cab}(a)$

1. Si $a = \varepsilon$ entonces $\text{CAB}(a) = \varepsilon$
2. Si $a = X \in N \cup T$
 1. Si $X \in T$, $\text{Cab}(X) = \{X\}$
 2. Si $X \in N$, $\text{Cab}(X) = \bigcup_{i=1}^n \text{Cab}(a_i)$
 $X \rightarrow a_1, X \rightarrow a_2, \dots, X \rightarrow a_n$
3. Si $a = X_1 X_2 \dots X_n$ $\text{CAB}(a) =$

Si $\varepsilon \notin \text{CAB}(X_1)$ entonces
 $\text{CAB}(X_1)$
en caso contrario si $\varepsilon \notin \text{CAB}(X_2)$ entonces
 $\text{CAB}(X_1) \cup \text{CAB}(X_2) - \{\varepsilon\}$
en caso contrario si $\varepsilon \notin \text{CAB}(X_3)$ entonces
 $\text{CAB}(X_1) \cup \text{CAB}(X_2) \cup \text{CAB}(X_3) - \{\varepsilon\}$
.....
en caso contrario si $\varepsilon \notin \text{CAB}(X_n)$ entonces

$$\bigcup_{i=1}^n \text{CAB}(X_i) - \{\varepsilon\}$$
en caso contrario

$$\bigcup_{i=1}^n \text{CAB}(X_i)$$

Definiciones ASDSR-LL(1). Siguiente

- $SIGUIENTE(A) = SIG(A) = FOLLOW(A)$
 $A \in N$

$$\{a \in T / S \Rightarrow^+ aAa\beta \quad a, \beta \in (NUT)^*\} \cup \{\$, \text{ si } S \Rightarrow^* aA\}$$

- ALGORITMO SIG (A)

1. Si $A=S$ entonces $\$ \in SIG(A)$
2. Si $\exists B \rightarrow aA\beta$ entonces
 $\forall a \neq \varepsilon \in CAB(\beta) \quad a \in SIG(A)$
3. Si $\exists B \rightarrow aA$ ó $\exists B \rightarrow aA\beta$ con $\varepsilon \in CAB(\beta)$
Entonces $SIG(B) \subset SIG(A)$

- EJEMPLO

$S \rightarrow ABC$ (1)

$A \rightarrow aAa$ (2)

$A \rightarrow Bd \quad B \rightarrow b \mid \varepsilon \quad C \rightarrow c \mid \varepsilon$

$SIG(A) = \{b, c, \$, a\}$

SIG(A)

(1) $CAB(BC) =$

$CAB(B) = \{b, \varepsilon\}$

$CAB(C) = \{c, \varepsilon\}$

$SIG(S) = \{\$\}$

(2) $CAB(a) = \{a\}$

Símbolos Directores, Gramática LL(1)

- Cabecera de un conjunto de cadenas

$$\text{CAB}\{\alpha_1, \alpha_2, \dots, \alpha_n\} \stackrel{\text{def}}{=} \bigcup_{i=1}^n \text{CAB}(\alpha_i) \quad \alpha_i \in (N \cup T)^*$$

- Yuxtaposición / Producto "cadena" o "conjunto cadenas"

$$\alpha \circ \{\alpha_1, \alpha_2, \dots, \alpha_n\} \stackrel{\text{def}}{=} \{\alpha\alpha_1, \alpha\alpha_2, \dots, \alpha\alpha_n\}$$

- Símbolos Directores

$$\text{SD}(A \rightarrow \alpha) \stackrel{\text{def}}{=} \text{CAB}(\alpha \circ \text{SIG}(A))$$

$$= \text{CAB}(\alpha \circ \{a_1, a_2, \dots, a_n\}) = \text{CAB}(\alpha a_1, \alpha a_2, \dots, \alpha a_n) =$$

$$= \bigcup_{i=1}^n \text{CAB}(\alpha a_i) = \begin{cases} \text{CAB}(\alpha) & \text{si } \epsilon \notin \text{CAB}(\alpha) \\ \text{CAB}(\alpha) - \{\epsilon\} \cup \{a_1, a_2, \dots, a_n\} & \text{si } \epsilon \in \text{CAB}(\alpha) \end{cases}$$

\downarrow
 $\text{SIG}(A)$

- Una Gramática es LL(1) sii

$$\forall A \in N \quad \forall A \rightarrow \alpha, A \rightarrow \beta \quad \text{SD}(A \rightarrow \alpha) \cap \text{SD}(A \rightarrow \beta) = \emptyset$$



Ejemplo Gramática NO LL(1)

(1) $S \rightarrow i E t S S'$

(2) $S \rightarrow a$

(3) $S' \rightarrow e S$

(4) $S' \rightarrow \varepsilon$

(5) $E \rightarrow b$

$SD (S \rightarrow i E t S S') = \{i\}$

$SD (S \rightarrow a) = \{ a \}$

$SD (S' \rightarrow eS) = \{e\}$

$SD (E \rightarrow b) = \{b\}$

$SD (S' \rightarrow \varepsilon) = SIG (S') = SIG (S), \$ \in SIG (S)$

Por **(1)**: $CAB(S') = \{e, \varepsilon\}$, $SIG(S)$ ya está.

Por **(3)**: $SIG(S')$ que también está.

$\Rightarrow SIG (S) = SD (S' \rightarrow \varepsilon) = \{e, \$\}$

Por tanto: $SD (S' \rightarrow eS) \cap SD (S' \rightarrow \varepsilon) = \{e\}$

La gramática no es LL(1)

Resultados LL(1)

■ Resultado-1

- Si una gramática es LL(1) \Rightarrow No es recursiva por la izda (ni tiene ciclos por la izda)

Recursividad por la izda:

$A \rightarrow A a \mid \beta$

1- Si $\varepsilon \notin \text{CAB}(\beta)$ ó $\varepsilon \in \text{CAB}(\beta)$ pero $\text{CAB}(\beta) \neq \{\varepsilon\}$

$\text{SD}(A \rightarrow A a) \supset \text{CAB}(A a) - \{\varepsilon\} \supset \text{CAB}(A) - \{\varepsilon\} \supset (A \rightarrow \beta) \text{CAB}(\beta) - \{\varepsilon\}$

2- Si $\text{CAB}(\beta) = \{\varepsilon\} \Rightarrow A \rightarrow \varepsilon, A \rightarrow A a$

$\text{SD}(A \rightarrow A a) \supset \text{CAB}(A a) - \{\varepsilon\} \supset$

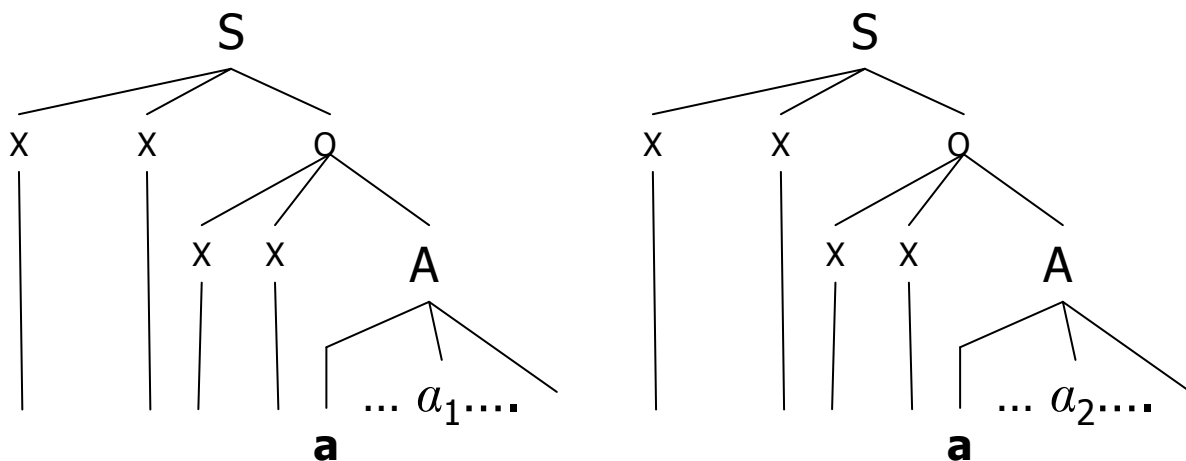
$(\varepsilon \in \text{CAB}(A)) \text{CAB}(a) - \{\varepsilon\}$

$\text{SD}(A \rightarrow \varepsilon) = \text{SIG}(A) \supset (A \rightarrow A a) \text{CAB}(a) - \{\varepsilon\}$

■ Resultado-2

- Si una gramática es LL(1) \Rightarrow No es ambigua

Dem. Si fuera ambigua:



$\Rightarrow a \in \text{SD}(A \rightarrow a_1) \cap \text{SD}(A \rightarrow a_2)$

Ejemplos Resultados

■ Recursividad

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

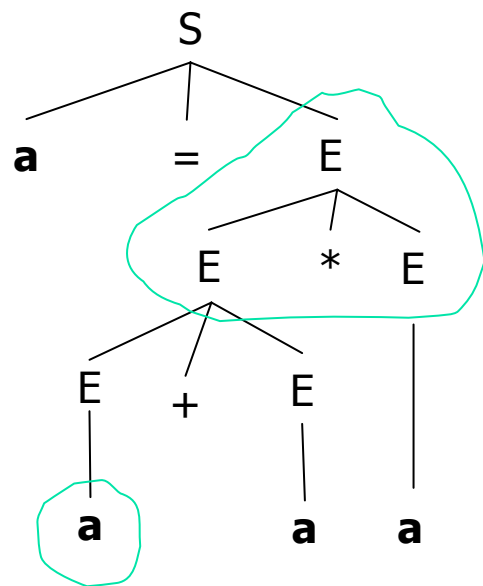
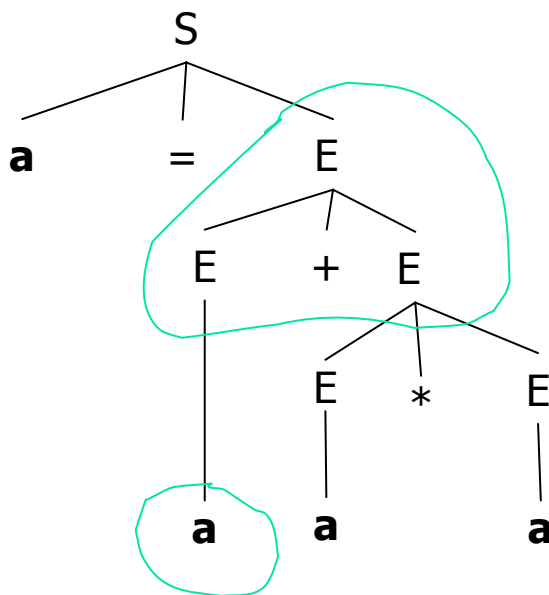
$F \rightarrow a \mid (E)$

- $SD(E \rightarrow E + T) = CAB(E+T) = CAB(E) = CAB(T) = CAB(T * F) \cup CAB(F) = CAB(F) = \{ (, a \}$
- $SD(E \rightarrow T) = CAB(T) = \{ (, a \}$

■ Ambigüedad

$S \rightarrow a = E$

$E \rightarrow E + E \mid E * E \mid a$



$$a \in SD(E \rightarrow E + E) \cap SD(E \rightarrow E * E)$$

Análisis Descendente No Recursivo Predictivo

- Se puede aplicar a Gramáticas LL(1)
- No es recursivo (utilizar una pila)
- Se “predice” la regla a usar, según el símbolo siguiente.

- Tabla de un Análisis Predictivo (M)

$$M(A, a) = \{a \in (N \cup T)^* / a \in SD(A \rightarrow \alpha)\}$$

$$A \in N, a \in T \cup \{\$ \}$$

- Ejemplo

$$S \rightarrow iEtSS' \quad | \quad a$$

$$S' \rightarrow eS \quad | \quad \varepsilon$$

$$E \rightarrow b$$

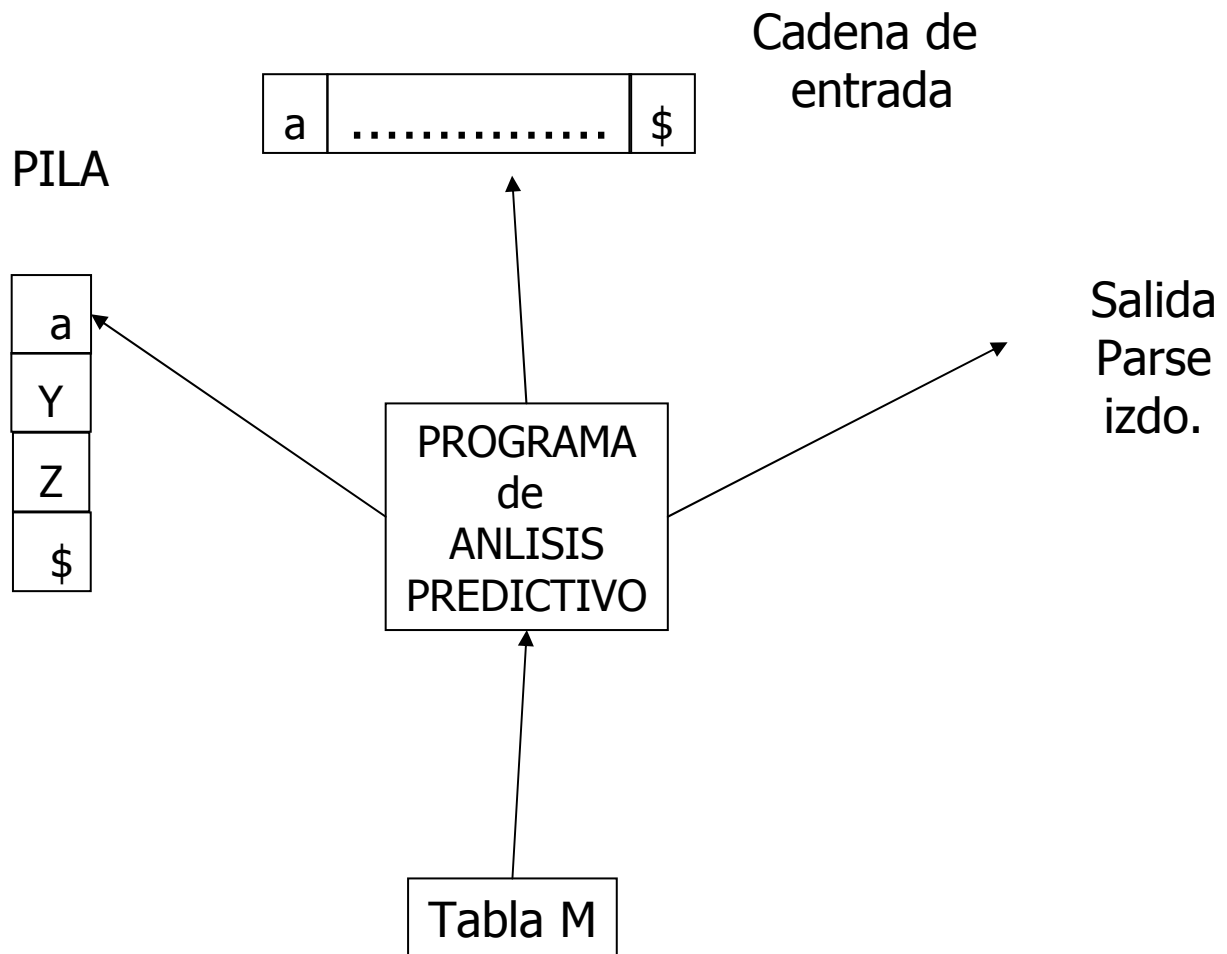
	a	b	e	i	t	\$
S	a			iEtS S'		
S'			eS ε			ε
E		b				

$$e \in SD(S' \rightarrow eS) \cap SD(S' \rightarrow \varepsilon)$$

- **Teorema.** Una gramática es LL(1) sii M tiene como mucho una cadena en cada celda.
(Definición de Aho-Ullman de gram. LL(1))

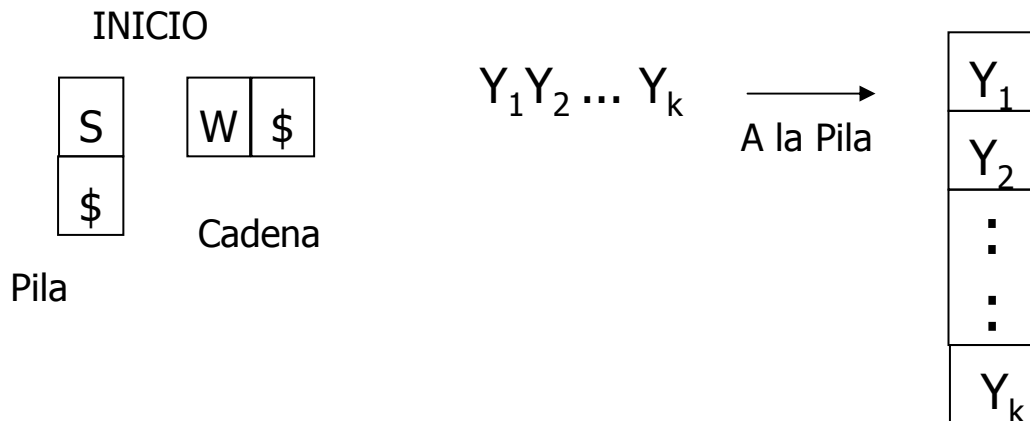
Reconocedor LL(1). Elementos

- Análisis predictivo no recursivo (Gramática LL(1))



Reconocedor LL(1). Algoritmo

■ Algoritmo Análisis Predictivo No Recursivo:



ptr – puntero al primer símbolo de w \$

REPETIR

X pila , ptr \rightarrow a

Si $X \in T$ ó $X = \$$ entonces

Si $X = a$ entonces

sacar X

avanzar ptr

else

error

else

Si $M(X, a) = Y_1 Y_2 \dots Y_k$ entonces

sacar X ; meter $Y_1 Y_2 \dots Y_k$

anotar en SALIDA $X \rightarrow Y_1 Y_2 \dots Y_k$

else error

HASTA que la pila esté vacía

Ejemplo. Reconocedor LL(1). A.P.N.R.

$E \rightarrow T E' \quad (1)$

$E' \rightarrow + T E' \quad (2) \mid \varepsilon \quad (3)$

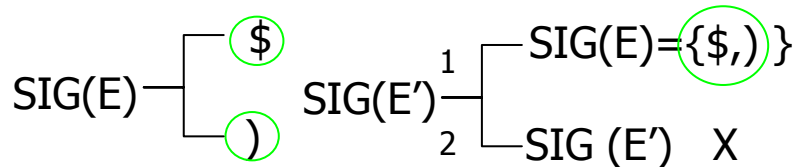
$T \rightarrow F T' \quad (4)$

$T' \rightarrow * F T' \quad (5) \mid \varepsilon \quad (6)$

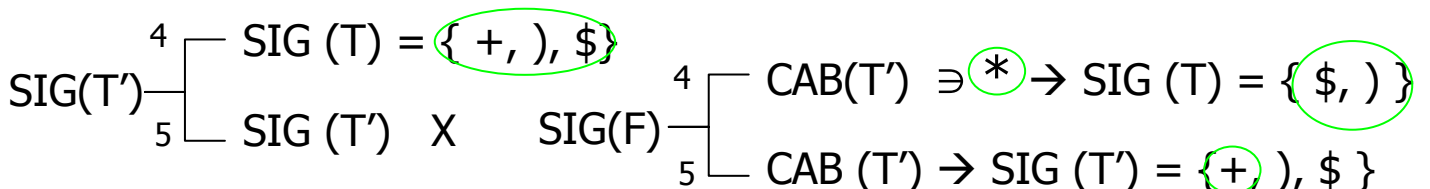
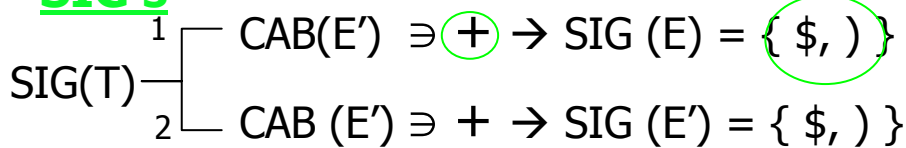
$F \rightarrow (E) \quad (7) \mid \text{id} \quad (8)$

SIG	E	E'	T	T'	F
))	+	+	+
	\$	\$))	*
			\$	\$)
					\$

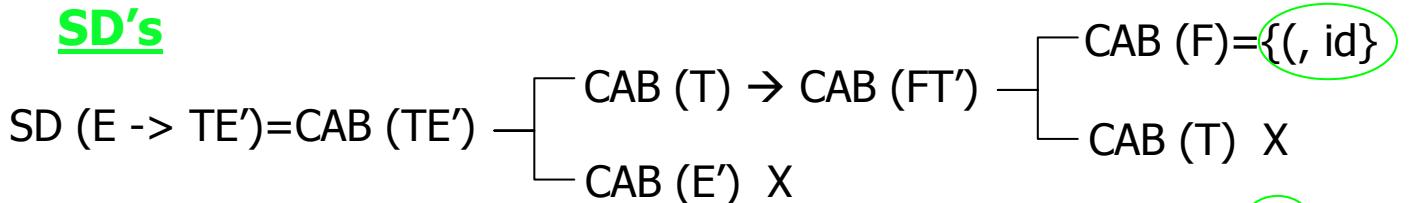
	id	+	*	()	\$
E	TE'			TE'		
E'		+TE'			ε	ε
T	FT'			FT'		
T'		ε	*FT'		ε	ε
F	id			(E)		



SIG's



SD's



$\text{SD}(E' \rightarrow +TE') = \{ + \}$

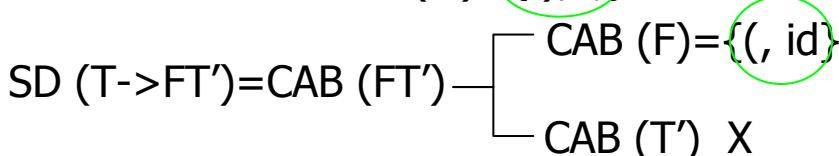
$\text{SD}(E' \rightarrow \varepsilon) \rightarrow \text{CAB}(\varepsilon) = \varepsilon$

$\text{SD}(E' \rightarrow \varepsilon) \rightarrow \text{SIG}(E') = \{), \$ \}$

$\text{SD}(T' \rightarrow *FT') = \{ * \}$

$\text{SD}(F \rightarrow (E)) = \{ (\}$

$\text{SD}(F \rightarrow \text{id}) = \{ \text{id} \}$



$\text{SD}(T' \rightarrow \varepsilon) = \text{SIG}(T') = \{ +,), \$ \}$

Ejemplo.Reconocedor aplicado a cadena

$E \rightarrow T E' \quad (1)$

$E' \rightarrow +T E' \quad (2) \mid \varepsilon \quad (3)$

$T \rightarrow F T' \quad (4)$

$T' \rightarrow * F T' \quad (5) \mid \varepsilon \quad (6)$

$F \rightarrow (E) \quad (7) \mid \text{id} \quad (8)$

	id	+	*	()	\$
E	TE'			TE'		
E'		+TE'			ε	ε
T	FT'			FT'		
T'		ε	*FT'		ε	ε
F	id			(E)		

PILA	CADENA	REGLA	ACCIÓN
E\$	id+id*id\$		
TE'\$	id+id*id\$	1	Sustituir(S)
FT'E'\$	id+id*id\$	4	S
idT'E'\$	id+id*id\$	8	S
T'E'\$	+id*id\$	-	Avanzar(A)
E'\$	+id*id\$	6	S
+TE'\$	+id*id\$	2	S
TE'\$	id*id\$	-	A
FT'E'\$	id*id\$	4	S
idT'E'\$	id*id\$	8	S
T'E'\$	*id\$	-	A
*FT'E'\$	*id\$	5	S
FT'E'\$	id\$	-	A
idT'E'\$	id\$	8	S
T'E'\$	\$	-	A
E'\$	\$	6	S
\$	\$	3	S
-	-	-	A

Parse Izdo: 1-4-8-6-2-4-8-5-8-6-3

ANÁLISIS DESCENDIENTE RECURSIVO (PREDICTIVO)

```

■ GRAMATICA LL(1)
ana_desc_rec ( )
{
    leer_simbolo ( ) ;
    funcion_S ( ) ;
    printf("OK\n") ;
}
    
```

```

■ TERMINALES
∀a ∈ T
funcion_a ( )
{
    if simbolo == 'a'
    {
        leer_simbolo ( ) ;
    }
    else
    {
        error
    }
}
    
```

a_1

$$A \rightarrow X_{11} X_{12} X_{13} \dots X_{1M1}$$

a_2

$$A \rightarrow X_{21} X_{22} X_{23} \dots X_{2M2}$$

a_n

$$A \rightarrow X_{n1} X_{n2} X_{n3} \dots X_{nMn}$$

```

genérica
funcion_term ( char a)
{
    if simbolo == a
    {
        leer_simbolo ( )
    }
    else
    {
        error
    }
}
    
```

```

■ NO TERMINALES
funcion_A ( ) {
    switch (simbolo) {
        case SD(A ->a1) :
            X11; X12; ... ; X1M1;
        case SD(A ->a2) :
            X21; X22; ... ; X2M2;
        .....
        case SD(A ->an) :
            Xn1; Xn2; ... ; XnMn;
        default:
            error( ) ;
    }
}
    
```

Ejemplo. Implementación A.D. Recursivo.

```
ana_desc_rec ( )
{
    leer_simbolo ( ) ;
    funcion_E ( ) ;
    printf("OK\n") ;
}
```

```
funcion_E() { SD(E->TE')={id, (}
    switch (simbolo) {
        case id, (:
            funcion_T();
            funcion_E'();
        default: error();
    }
}
```

```
funcion_T() {SD(T->FT')={id, (}
    switch (simbolo) {
        case id, (:
            funcion_F();
            funcion_T'();
        default: error();
    }
}
```

```
funcion_T'() {
    switch (simbolo) {
        case *:
            funcion_term('*');
            /*leer_simbolo()*/
            funcion_F();
            funcion_T'();
        case +, , $: ;
        default: error();
    }
    SD(T'->*FT')={*}
    SD(T'->ε)={+, , $}
}
```

$E \rightarrow T E' \quad (1)$

$E' \rightarrow +T E' \quad (2) \mid \varepsilon \quad (3)$

$T \rightarrow F T' \quad (4)$

$T' \rightarrow * F T' \quad (5) \mid \varepsilon \quad (6)$

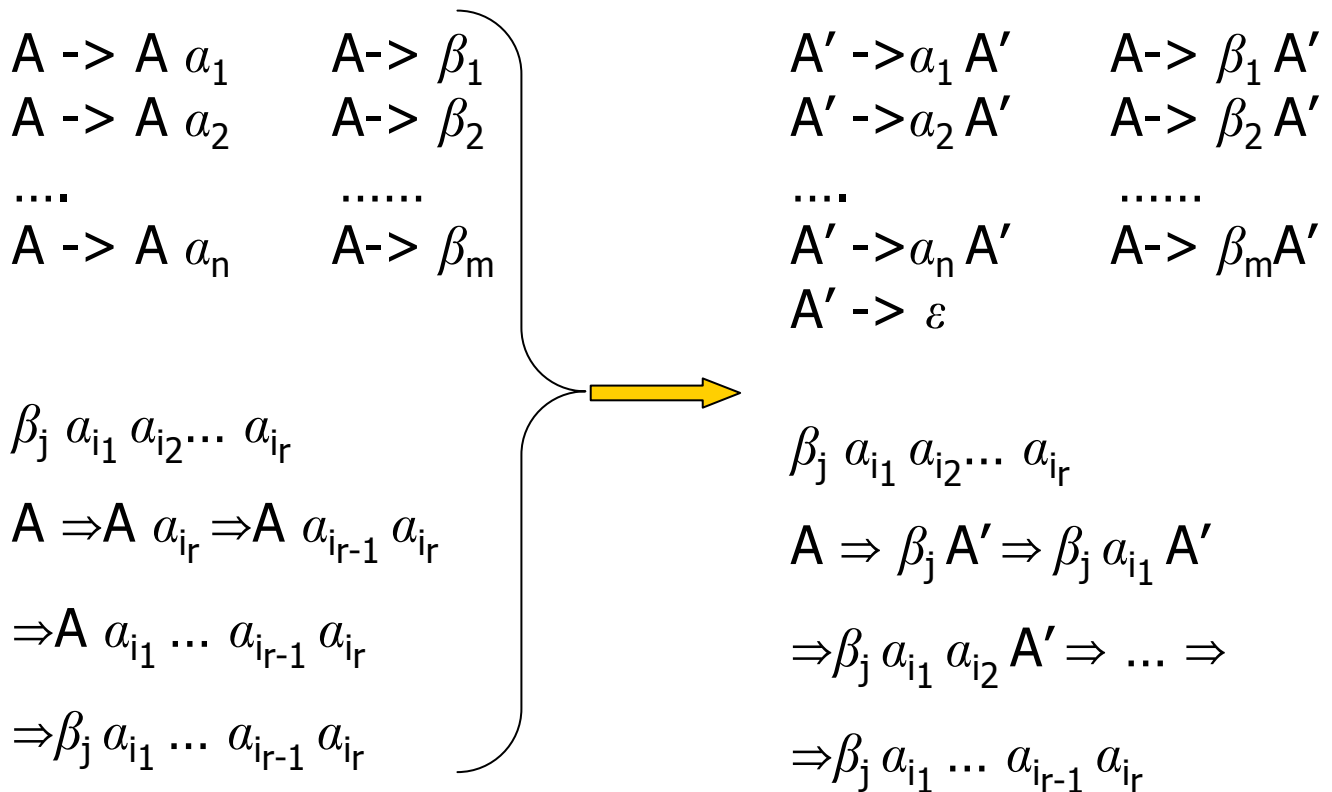
$F \rightarrow (E) \quad (7) \mid id \quad (8)$

```
funcion_E'() {
    switch (simbolo) {
        case +:
            funcion_Term('+');
            /* leer_simbolo()*/
            funcion_T();
            funcion_E'();
        case ), $: ;
        default: error();
    }
    SD(E'->+TE')={+}
    SD(E'->ε)={}, $}
}
```

```
funcion_F'() {
    switch (simbolo) {
        case (:
            funcion_Term('(');
            /* leer_simbolo()*/
            funcion_E();
            funcion_term(')');
        case id:
            funcion_term(id) ;
            /* leer_simbolo()*/
        default: error();
    }
    SD(F->(E))={ ( }
    SD(F->id)={ id }
}
```

Reglas Heurísticas para obtener una G.LL(1)

■ Eliminar Recursividad por la Izda.



EJEMPLOS

$A \rightarrow A a \mid a$

$A' \rightarrow aA' \mid \varepsilon$

$A \rightarrow aA'$

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid id$

$E' \rightarrow + T E' \mid \varepsilon$

$E \rightarrow T E'$

$T' \rightarrow * F T' \mid \varepsilon$

$T \rightarrow F T'$

$F \rightarrow (E) \mid id$

Reglas Heurísticas para obtener una G.LL(1)

■ Eliminar ciclos por la izquierda

- Gramática sin ε (puede convertirse)
- Sin ciclos de la forma $A \Rightarrow^+ A$, si existe se puede eliminar)
- Algoritmo

Ordenar los no terminales A_1, A_2, \dots, A_N

for $i=1$ to N {

for $j=1$ to $i-1$ {

reemplazar reglas $A_i \rightarrow A_j \gamma$ ($i > j$)

$A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$

donde $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$

}

Eliminar la recursividad izda si existe para A_i

}

- No debe tener reglas- ε , por ejemplo:

$S \rightarrow ASc, A \rightarrow Bb \mid \varepsilon,$

si $S < A < B$, entonces no hay que hacer nada, pero sigue habiendo ciclos: $S \Rightarrow ASc \Rightarrow Sc$

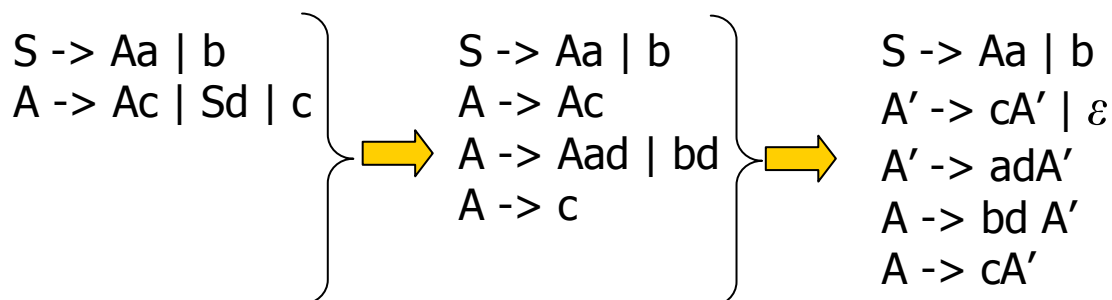
- No debe existir tampoco $A \Rightarrow^+ A$, por ejemplo:

$A \rightarrow B, B \rightarrow A$

si $A < B$, entonces aplicando el algoritmo,

$A \rightarrow B, B \rightarrow B, B' \rightarrow B'$, sigue habiendo recursividad.

EJEMPLO



Reglas Heurísticas para obtener una G.LL(1)

■ Factorización por la izquierda

- "Modificar las A-reglas para retrasar la selección de la regla a aplicar hasta que se haya visto suficiente"

$$\begin{array}{lcl}
 A \rightarrow & \left. \begin{array}{l} a \beta_1 \\ | \\ a \beta_2 \\ | \\ \dots \\ | \\ a \beta_n \end{array} \right\} & \rightarrow \begin{array}{l} A \rightarrow a A' \\ A' \rightarrow \begin{array}{l} \beta_1 \\ | \\ \beta_2 \\ | \\ \beta_3 \\ \dots \\ | \\ \beta_n \end{array} \end{array}
 \end{array}$$

EJEMPLOS

1) $\langle \text{sent} \rangle \rightarrow \text{if } \langle \text{expr} \rangle \text{ then } \langle \text{sent} \rangle \text{ else } \langle \text{sent} \rangle$
 $\text{if } \langle \text{expr} \rangle \text{ then } \langle \text{sent} \rangle$



$\langle \text{sent} \rangle \rightarrow \text{if } \langle \text{expr} \rangle \text{ then } \langle \text{sent} \rangle \langle \text{parte_else} \rangle$
 $\langle \text{parte_else} \rangle \rightarrow \text{else } \langle \text{sent} \rangle$

$| \epsilon$

(sigue sin ser LL(1))

2) $S \rightarrow abB \mid abC \mid aDh \mid c$
 $B \rightarrow b, C \rightarrow c, D \rightarrow d$

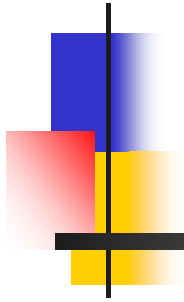


$S \rightarrow aA' \mid c$
 $A' \rightarrow bB \mid bC \mid Dh$
 $B \rightarrow b, C \rightarrow c, D \rightarrow d$



$S \rightarrow aA' \mid c$
 $A' \rightarrow bB' \mid Dh$
 $B' \rightarrow B \mid C, B \rightarrow b,$
 $C \rightarrow c, D \rightarrow d$

Nota. Si añadimos $D \rightarrow ba$, la factorización no resolvería el problema



ANÁLISIS SINTÁCTICO

ANÁLISIS ASCENDENTE

ANÁLISIS ASCENDENTE

■ Análisis ascendente con retroceso

- Construir el árbol sintáctico de abajo a arriba
- Aplicar una regla de producción:
reducción
- $A_k \rightarrow X_p X_{p+1} \dots X_n$

$$\overbrace{X_1 \dots X_{p-1} X_p \dots X_n}^{\alpha} \overbrace{Y_1 \dots Y_m}^{\beta}$$

$\underbrace{X_p \dots X_n}_{A_k}$

↓ Reducción

$$\overbrace{X_1 \dots X_{p-1} A_k}^{\alpha} \overbrace{Y_1 \dots Y_m}^{\beta}$$

- Lectura de un símbolo de la cadena de entrada: **desplazamiento**

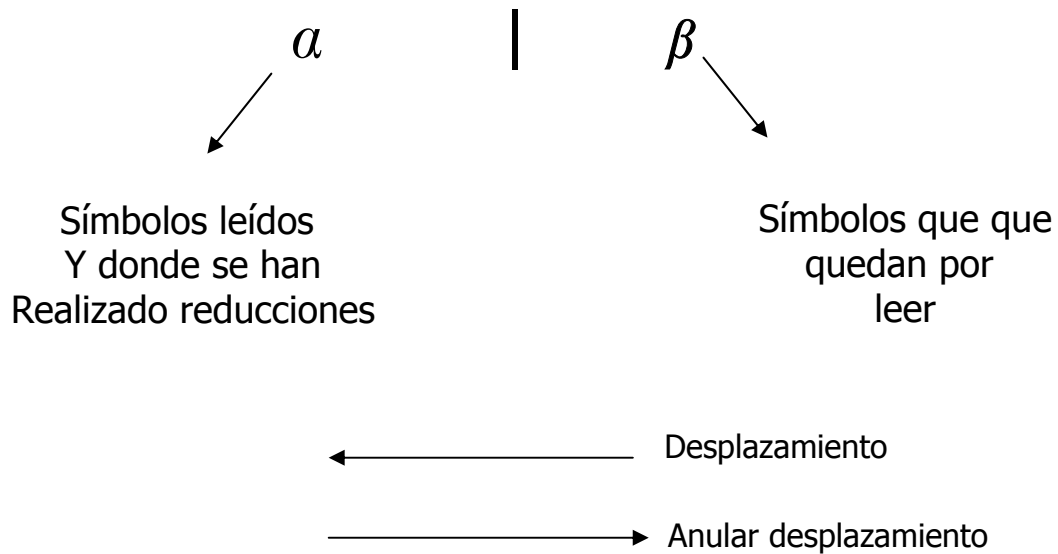
$$\overbrace{X_1 \dots X_p \dots X_n}^{\alpha} \overbrace{Y_1 \dots Y_m}^{\beta}$$

↓ Desplazamiento

$$\overbrace{X_1 \dots X_n Y_1}^{\alpha} \overbrace{Y_2 \dots Y_m}^{\beta}$$

- Dos operaciones más: **aceptar** y **error**

- 
- Las formas sentenciales se dividen siempre en dos



- **La gramática debe ser sin ε para poder realizar un análisis ascendente con retroceso**

ALGORITMO ASCENDENTE CON RETROCESO

Ensayar (a, β)

```
{
  for k=1 hasta el número de reglas
  {
    if (consecuente (regla k) ==
        alguna subcadena cola de  $a$ )
    {
      (reducir regla k en esa subcadena  $a$ 
      if (( $a$  == Axioma) && ( $\beta$  ==  $\varepsilon$ ))
      {
        ACEPTAR
      }
      else
      {
        Ensayar ( $a, \beta$ )
      }
      Anular reducción
    } /* if consecuente */
  } ( * for * /
  if ( $\beta \neq \varepsilon$ )
  {
    Desplazar
    Ensayar ( $a, \beta$ )
    Anular desplazamiento
  }
}
```

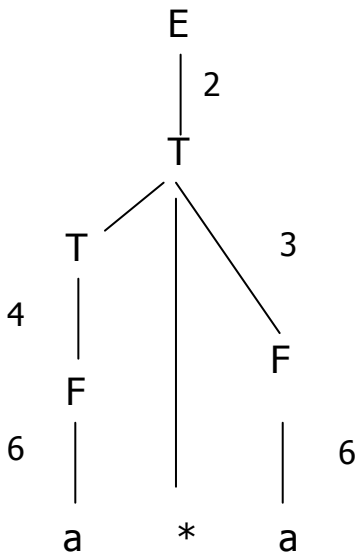
ERROR

Ejemplo A.S.A. con Retroceso

$E \rightarrow E + T (1) \mid T (2) \quad T \rightarrow T * F (3) \mid F (4) \quad F \rightarrow (E) (5) \mid a (6)$

$E \Rightarrow_D^2 T \Rightarrow_D^3 T * F \Rightarrow_D^6 T * a \Rightarrow^4 F * a \Rightarrow^6 a * a$

Parse Derecho: 6-4-6-3-2



Cadena	Pila	Acción
a *a	-	Desp.(D)
F *a	6	Red.(R)
T *a	6-4	R
E *a	6-4-2	R
E* a	6-4-2	D
E*a ε	6-4-2	D
E*F ε	6-4-2-6	R
E*T ε	6-4-2-6-4	R
E*E ε	6-4-2-6-4-2	R
E*T ε	6-4-2-6-4	AR
E*F ε	6-4-2-6	AR
E*a ε	6-4-2	AR
E* a	6-4-2	AD
E *a	6-4-2	AD
T *a	6-4	AR
T* a	6-4	D
T*a ε	6-4	D
T*F ε	6-4-6	R
T ε	6-4-6-3	R
E ε	6-4-6-3-2	R

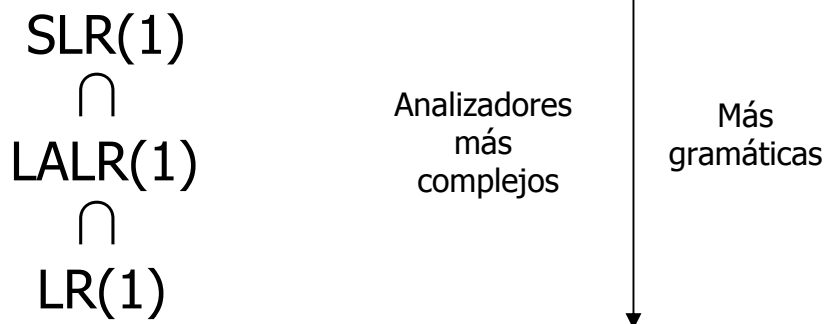


ANALIZADORES LR

- Analizadores ascendentes sin retroceso
- **LR(k)**
 - Left: Leer la cadena de izquierda a derecha
 - Right: Derivaciones derechas
 - (k): Basta con analizar los k símbolos siguientes para decidir qué acción realizar
- **Ventajas**
 - Los analizadores LR pueden reconocer la inmensa mayoría de los lenguajes de programación de contexto libre.
 - $LL(k) \subset LR(k)$ Si una gramática es $LL(k)$, entonces tiene que ser $LR(k)$
 - Localiza un error en el mismo instante que se produce.
 - Existen generadores automáticos
- **Inconvenientes**
 - Es necesario tener un generador, ya que es compleja una construcción directa

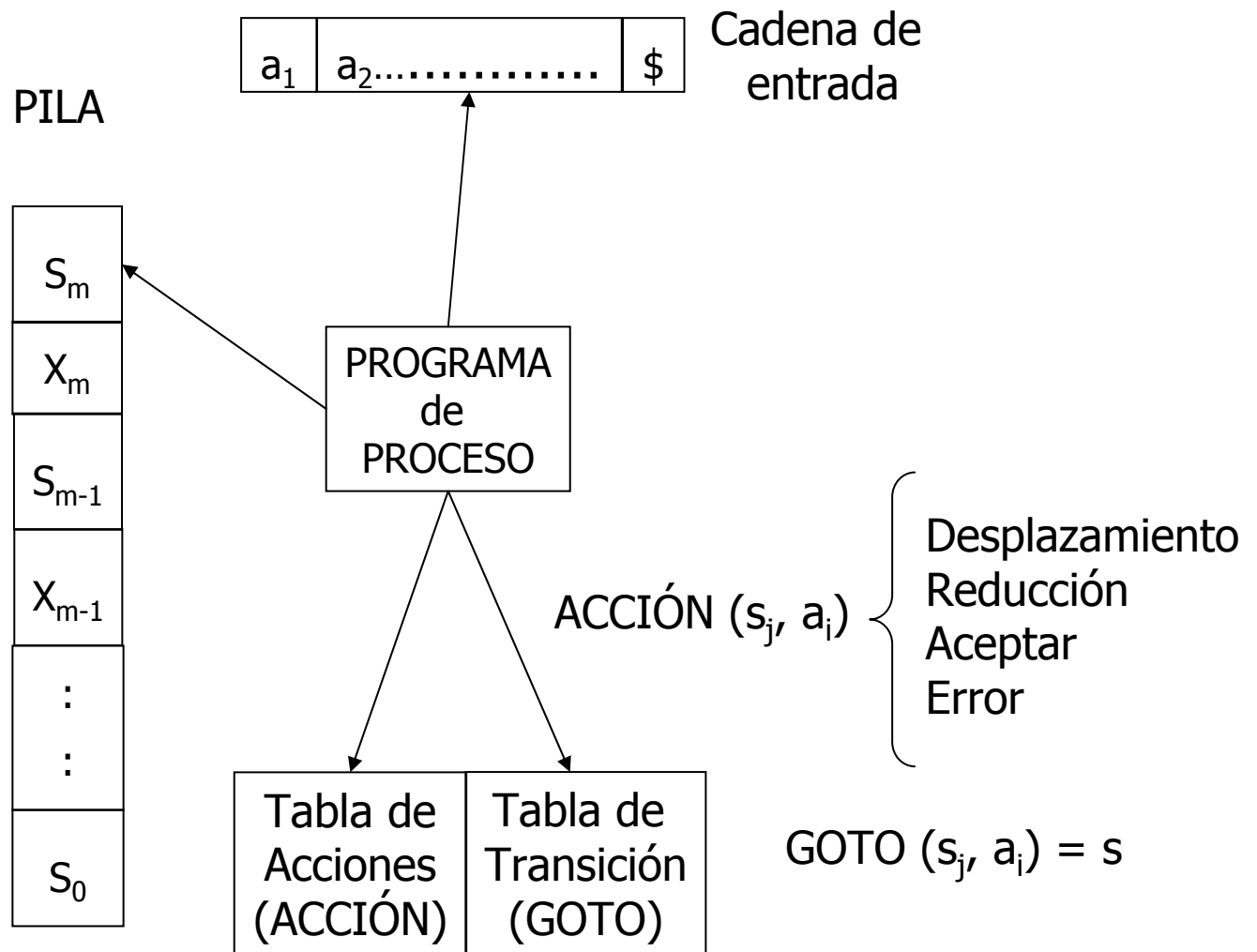
Analizadores LR

- En la práctica, casi todos los lenguajes de programación pueden analizarse mediante LR(0) o LR(1)
- LR(0) no es útil ya que pocas gramáticas cumplen la restricción de este tipo de gramáticas
- Tipos de gramáticas (analizadores)



- Yacc admite gramáticas LALR(1)

Funcionamiento de un analizador LR



S_i = estados del analizador (llevan la información sobre lo que ha ocurrido hasta entonces)

Configuración del analizador

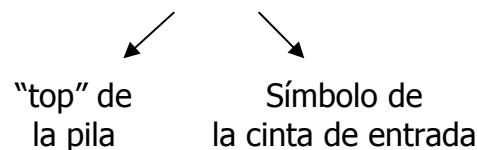
$$\underbrace{(s_0 X_1 s_1 \dots X_m s_m)}_{\text{PILA}}, \underbrace{a_i a_{i+1} \dots a_n \$}_{\text{CINTA ENTRADA}}$$

■ Configuración Inicial

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

■ Operaciones sobre el analizador

$$\blacksquare \text{ACCION}(s_m, a_i)$$



1. Desplazamiento

$$\text{GOTO}(s_m, a_i) = s$$

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

2. Reducción $A \rightarrow X_{r+1} X_{r+2} \dots X_m$

$$(s_0 X_1 s_1 \dots X_m s_m, a_i a_{i+1} \dots a_n \$) \vdash (s_0 X_1 s_1 \dots X_r s_r A, a_i a_{i+1} \dots a_n \$) \vdash$$

$$(s_0 X_1 s_1 \dots X_r s_r A s, a_i a_{i+1} \dots a_n \$) \text{ donde } \text{GOTO}(s_r, A) = s$$

3. Aceptar: cadena reconocida

4. Error: Se llamará a una rutina de recuperación, en caso de querer continuar

Ejemplo de Análisis LR-1

$S \rightarrow S + T (1) \mid T (2)$
 $T \rightarrow T * F (3) \mid F (4)$
 $F \rightarrow (S) (5) \mid a (6)$

Pas o	Pila	Cadena Entrada
1	0	a*(a+a)\$
2	0a5	*(a+a)\$
3	0F3	*(a+a)\$
4	0T2	*(a+a)\$
5	0T2*7	(a+a)\$
6	0T2*7(4	a+a)\$
7	0T2*7(4a5	+a)\$
8	0T2*7(4F3	+a)\$
9	0T2*7(4T2	+a)\$
10	0T2*7(4S8	+a)\$
11	0T2*7(4S8+6	a)\$
12	0T2*7(4S8+6a5)\$
13	0T2*7(4S8+6F3)\$
14	0T2*7(4S8+6T9)\$
15	0T2*7(4S8)\$
16	0T2*7(4S8)11	\$
17	0T2*7F10	\$
18	0T2	\$
19	0S1	\$
20	Aceptación	

Esta do	ACCIÓN (GOTO)						GOTO		
	a	+	*	()	\$	S	T	F
0	D5			D4			1	2	3
1		D6				Ace			
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			

ANÁLISIS SLR ("SIMPLE LR")

- **ITEM LR(0)** (o simplemente ITEM):

si $\exists A \rightarrow a_1 a_2$, entonces $[A \rightarrow a_1 \circ a_2]$

- $A \rightarrow \varepsilon \Rightarrow [A \rightarrow \circ]$

- Puede **representarse**:

(NÚMERO DE REGLA, POSICIÓN DEL PUNTO)

- **INTUITIVAMENTE**: Cuánto de una regla se ha visto en un punto concreto del proceso de análisis.

$A \rightarrow \circ XYZ$ Se espera ver una cadena derivable de "XYZ"

$A \rightarrow X \circ YZ$ Se ha visto una cadena derivable de "X", y se espera ver una cadena derivable de "YZ".

- **ITEM COMPLETO**: $[A \rightarrow \circ \tau]$

- **CONJUNTO ϕ** : Cjto de Todos los items de una gramática. Ejemplo:

$S \rightarrow E, E \rightarrow E+T \mid E-T \mid T, T \rightarrow (E) \mid a$

$\phi = \{ [S \rightarrow \circ E], [S \rightarrow E \circ], [E \rightarrow \circ E+T], [E \rightarrow E \circ +T], [E \rightarrow E+ \circ T], [E \rightarrow E+T \circ], \dots \}$

- **GRAMÁTICA AUMENTADA**: de G a G'

redefinición del axioma S, nuevo axioma $S' \in N, S' \rightarrow S$

- **CIERRE**: $\wp(\phi) \rightarrow \wp(\phi)$

$I \subset \phi \rightarrow \text{CIERRE}(I) \subset \phi$

1) $I \subset \text{CIERRE}(I)$

2) Si $[A \rightarrow a \circ B \beta] \in \text{CIERRE}(I)$ entonces

$\forall B \rightarrow \gamma \text{ regla}, [B \rightarrow \circ \gamma] \in \text{CIERRE}(I)$

- **EJEMPLO**

$G = \{ E' \rightarrow E, E \rightarrow E+T | T, T \rightarrow T+F | F, F \rightarrow (E), F \rightarrow id \}$

CIERRE $([E' \rightarrow \circ E]) = \{ [E' \rightarrow \circ E], [E' \rightarrow \circ E+T], [E \rightarrow \circ T], [T \rightarrow \circ T+F], [T \rightarrow \circ F], [F \rightarrow \circ (E)], [F \rightarrow \circ id] \}$

- **SUBCADENA VIABLE α**

$\alpha \in (N \cup T)^*$ es una cadena viable sii (def)

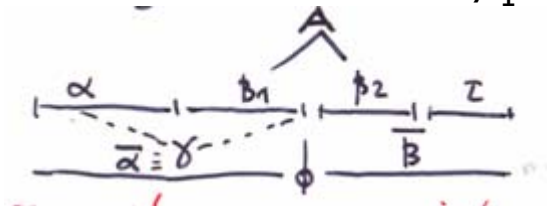
$S \Rightarrow^* \alpha \gamma \quad \gamma \in (N \cup T)^*$

(subcadena cabeza de alguna forma sentencial)

- **ITEM VÁLIDO PARA UNA SUBCADENA VIABLE**

- $[A \rightarrow \beta_1 \circ \beta_2]$ para una subcadena viable " $\alpha \beta_1$ "

$S \Rightarrow^* \alpha A \tau \Rightarrow \alpha \beta_1 \circ \beta_2 \tau$



$A \rightarrow \beta_1 \beta_2$ es una regla potencialmente candidata para una reducción futura

- **FUNCIÓN DE TRANSICIÓN = GOTO**

δ (goto) : $\wp(\phi) \times (N \cup T) \rightarrow \wp(\phi)$

$I \subset \phi \quad (I, X) \rightarrow \delta(I, X) \equiv \text{CIERRE}(\mathcal{R})$

donde $\mathcal{R} = \{ [A \rightarrow \alpha X \circ \beta] / [A \rightarrow \alpha \circ X\beta] \in I \}$

- **EJEMPLO-1**

$G = \{ E' \rightarrow E, E \rightarrow E+T | T, T \rightarrow T+F | F, F \rightarrow (E), F \rightarrow id \}$

$\delta(\{ [E' \rightarrow E \circ], [E \rightarrow E \circ +T] \}, +) =$

CIERRE $\{ [E \rightarrow E+ \circ T] \} = \{ [E \rightarrow E+ \circ T], [T \rightarrow \circ T+F], [T \rightarrow \circ F], [F \rightarrow \circ (E)], [F \rightarrow \circ id] \}$

Por qué la Función Goto (δ) ?

- $I \equiv$ el conjunto de items válidos para la subcadena viable " γ ", entonces

$\delta(I, X) \equiv$ el conjunto de items válidos para la subcadena viable " γX "

DEMOSTRACIÓN

1) Sea $[A \rightarrow a \circ X \beta] \in I$. Si es item válido para γ , $\exists \tau / \gamma = \tau a$ y

$$S \Rightarrow^* \tau A \sigma \Rightarrow \underline{\tau a} X \beta \sigma = \gamma X \beta \sigma$$

Ahora, esto quiere decir que

$[A \rightarrow aX \circ \beta] \in \delta(I, X)$ es item válido para " γX "

2) Sea $[A \rightarrow a \circ X \beta] \in I$, y $\beta = B\theta$ entonces

$[A \rightarrow aX \circ \beta] \in \delta(I, X)$ y $[B \rightarrow \circ \mu_i] \in \delta(I, X)$,

$\forall B \rightarrow \mu_i$

Ya que $[A \rightarrow a \circ X \beta]$ es item válido para γ ,

$\exists \tau / \gamma = \tau a$ y

$$S \Rightarrow^* \tau A \sigma \Rightarrow \tau a X \beta \sigma \equiv \tau a X B\theta \sigma$$

$$\Rightarrow \underline{\tau a} X \mu_i \underline{\theta} \sigma \equiv \gamma X \mu_i \pi, \text{ es decir:}$$

$$S \Rightarrow^* \gamma X B \pi \Rightarrow \gamma X \mu_i \pi, \text{ y entonces}$$

$[B \rightarrow \circ \mu_i]$ es un item válido para la subcadena viable " γX "



Colección de conjuntos de items LR(0)

$\mathbf{C} = \{I_0, I_1, I_2, I_3, \dots\}$ $I_j \subset \phi$ Cada I_j representará un estado del análisis LR.

1. La gramática tiene que ser aumentada (en caso de no serlo se aumenta con $S' \rightarrow S$)

2. Algoritmo

3. $\mathbf{C} = \{I_0\}$ donde $I_0 = \text{CIERRE}\{[S' \rightarrow \circ S]\}$

REPETIR

Para cada $I \in \mathbf{C}$ y cada $X \in N \cup T$,

si $\delta(I, X) \neq \emptyset$ y $\delta(I, X) \notin \mathbf{C}$

entonces añadir $I_j \equiv \delta(I, X)$ a \mathbf{C}

HASTA que no puedan añadirse a \mathbf{C} nuevos I_j

ALTERNATIVA PARA LA CONSTRUCCIÓN DE C

1. Construcción de un Autómata Finito No Determinista (AFND) (Q, T_e, δ, q_0, F) ,

$Q \equiv \phi \equiv$ Conjunto de todos los items LR(0)

$T_e = N \cup T$

$q_0 = [S' \rightarrow \circ S]$ $F = \{[A \rightarrow \tau \circ] \text{ —es decir items completos—}\}$

$[A \rightarrow a \circ \beta] \in \delta([A \rightarrow a \circ a \beta], a)$

$[A \rightarrow a B \circ \beta] \in \delta([A \rightarrow a \circ B \beta], B)$

$[B \rightarrow \circ \tau] \in \delta([A \rightarrow a \circ B \beta], \epsilon)$

2. Hacer este automáta determinista



Construcción de un Análisis LR(0)

| Gramática aumentada \Rightarrow Construcción de **C**

1. $S_i \equiv I_i \in \mathbf{C}$
 2. $\delta \equiv \text{Goto}$
 3. La tabla de acciones No depende del símbolo en la cinta de entrada
 - a. Si $I_j = \{[A \rightarrow \tau \circ]\} \Rightarrow \text{ACCION}(I_j) = \text{Reducir por } A \rightarrow \tau$
 - b. Si $I_j = \{[S' \rightarrow S \$ \circ]\} \Rightarrow \text{ACCION}(I_j) = \text{Aceptar}$
 - c. Si I_j no tiene ningún ítem completo $\Rightarrow \text{ACCION}(I_j) = \text{Desplazar}$
- Si existe algún estado (s_j) cuyo conjunto de ítems (I_j) contiene más de un ítem, con alguno de ellos completo \Rightarrow la gramática no es LR(0)

Análisis SLR(1)

- Supongamos que hay que analizar "tas" ($t, s \in T^*$, $a \in T$).
- Tras leer t , $\alpha = \gamma \tau$, quedando por leer "as", supongamos $\exists [A \rightarrow \tau \circ]$ ítem completo válido para la subcadena viable " $\gamma \tau$ ".
- Si reducimos por esta regla, entonces " γAas " debería ser una forma sentencial, por lo que " a " debería estar en $SIG(A)$.
- Podría ser que " a " estuviera en $SIG(A)$, pero no por esta forma sentencial " γAas ", sino por otra. Es por tanto una condición necesaria para aplicar la regla $A \rightarrow \tau$, aunque no suficiente.
- Esta consulta a los conjuntos SIGs permitirá eliminar duplicidades de la tabla de acciones, dando lugar al análisis SLR(1).

CONSTRUCCIÓN DE UNA MÁQUINA SLR(1)

1. Gramática aumentada \Rightarrow Construir **C** (Igual que LR(0))
2. Estados I_j (igual que LR(0))
3. Función $\delta \equiv$ Goto (igual que LR(0))
4. Tabla de ACCIÓN
 - a) ACCIÓN (I_j, a)=D, si $\exists [A \rightarrow \alpha \circ a \beta] \in I_j$
 - b) ACCIÓN (I_j, a)=R por $A \rightarrow \tau$ ($A \neq S'$) si $[A \rightarrow \tau \circ] \in I_j$ y $a \in SIG(A)$
 - c) ACCIÓN ($I_j, \$$)=ACEPTAR si $[S' \rightarrow S \circ] \in I_j$
 - d) Resto de acciones = Error
5. Si no hay duplicidad en la Tabla Acción \Rightarrow Gramát. es SLR(1)