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

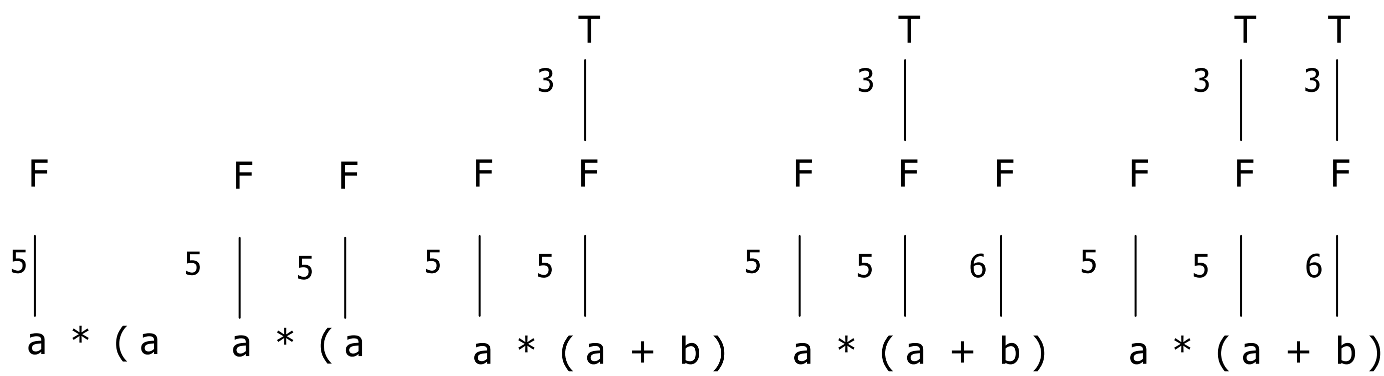
AS. Parses Izquierdo y Derecho

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

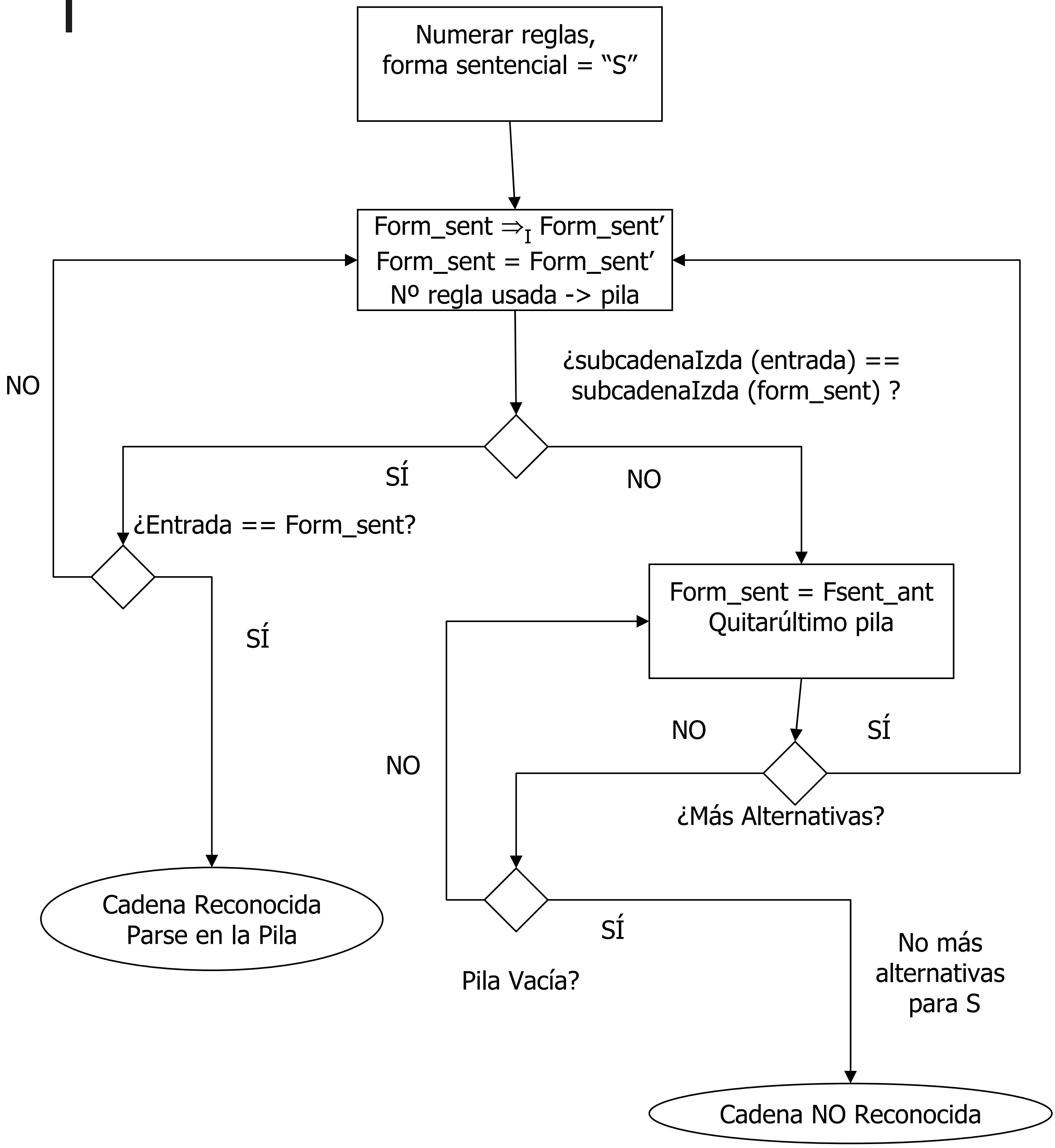
ANÁLISIS DESCENDENTE

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 relgas
  - 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 -> T + E (1) | T (2)
T -> F * T (3) | F (4)
F -> a (5) | b (6) | (E) (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 -> aA (1) | bB (2)

  A -> Aa (3) | ε (4)

  B -> Bb (5) | ε (6)

  Analizar aa

Forma Sentencial	Pila
S	-
aA	1-3
aAaa	1-3-3

- **Ciclos por la izquierda**

  S -> aA (1)

  A -> Bb (2) | ε (3)

  B -> 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 -> SA (1) | A (2)

   A -> 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

  $\alpha \in (N \cup T)^*$, Cabecera(α)=Cab (α)=First(α)=

  $\{a \in T / \alpha \Rightarrow^* a\beta, \beta \in (N \cup T)^* \} \cup \{\varepsilon$ si $\alpha \Rightarrow^* \varepsilon\}$

- Algoritmo para calcular Cab(α)

1. Si $\alpha = \varepsilon$ entonces CAB(α)= ε
2. Si $\alpha = X \in N \cup T$
   1. Si $X \in T$, Cab (X) = {X}
   2. Si $X \in N$, Cab (X) = $\bigcup_{i=1}^{n} \text{Cab}(\alpha_i)$

   $X \rightarrow \alpha_1$, $X \rightarrow \alpha_2$, ... , $X \rightarrow \alpha_n$

3. Si $\alpha = X_1 X_2 \ldots X_n$ CAB(α)=

   Si $\varepsilon \notin$ CAB(X_1) entonces

   CAB (X_1)

   en caso contrario si $\varepsilon \notin$ CAB(X_2) entonces

   CAB (X_1) $\cup$ CAB (X_2) – $\{\varepsilon\}$

   en caso contrario si $\varepsilon \notin$ CAB(X_3) entonces

   CAB (X_1) $\cup$ CAB (X_2) $\cup$ CAB (X_3) – $\{\varepsilon\}$

   ....

   en caso contrario si $\varepsilon \notin$ 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∈N
  $\{a \in T \,/\, S \Rightarrow^{+} \alpha A a \beta \quad \alpha,\beta \in (N \cup T)^{*}\} \cup$
  $\{\$, \text{ si } S \Rightarrow^{*} \alpha A\}$
  - ALGORITMO SIG (A)
  1. Si A=S entonces $\$ \in$ SIG(A)
  2. Si $\exists$ B -> $\alpha A \beta$ entonces
     $\forall\, a \neq \varepsilon \in$ CAB(β) $\quad a \in$ SIG (A)
  3. Si $\exists$ B -> αA ó $\exists$ B -> $\alpha A \beta$ con $\varepsilon \in$ CAB(β)
     Entonces SIG (B) $\subset$ SIG(A)
  - EJEMPLO

  S -> ABC (1)
  A-> aAa (2)
  A -> Bd B -> b | ε C -> c | ε
  SIG(A) = {b, c, \$, a}

SIG(A)

(1) CAB(BC) = CAB (B)={b,ε} CAB(C)={c, ε}

SIG(S) = {\$}

(2) CAB (a) = {a}

Símbolos Directores, Gramática LL(1)

- Cabecera de un conjunto de cadenas

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

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

$$\alpha \circ \{\alpha_1, \alpha_2, \ldots, \alpha_n\} \underset{def}{=} \{\alpha\alpha_1, \alpha\alpha_2, \ldots, \alpha\alpha_n\}$$

- Símbolos Directores

$$SD(A->\alpha) \underset{def}{=} CAB(\alpha \circ SIG(A))$$

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

$$= \bigcup_{i=1}^{n} CAB(\alpha a_i) = \begin{cases} CAB(\alpha) & \text{si } \varepsilon \notin CAB(\alpha) \\ CAB(\alpha) - \{\varepsilon\} \cup \underbrace{\{a_1, a_2, \ldots, a_n\}}_{\Downarrow \; SIG(A)} & \end{cases}$$

- Una Gramática es LL(1) sii

$\forall$ A $\in$ N $\forall$ A -> α, A-> β SD(A ->α) $\cap$ SD(A->β)

Ejemplo Gramática NO LL(1)

(1) S -> i E t S S′
(2) S -> a
(3) S′ -> e S
(4) S′ -> ε
(5) E -> b

SD (S -> i E t S S′) = {i}
SD (S -> a) = { a }
SD (S′ -> eS) = {e}
SD (E -> b) = {b}
SD (S′ -> ε) = SIG (S′) = SIG (S), $\$ \in$ SIG(S)
Por **(1)**: CAB(S′)={e, ε}, SIG(S) ya está.
Por **(3)**: SIG(S′) que también está.
=> SIG (S) = SD (S′ -> ε) = {e, $}
Por tanto: SD (S′ -> eS) $\cap$ SD (S′ -> ε) = {e}
La gramática no es LL(1)

Resultados LL(1)

- **Resultado-1**
  - Si una gramática es LL(1) => No es recursiva por la izda (ni tiene ciclos por la izda)

Recursividad por la izda:

A -> A α | β

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

SD (A-> A α) $\supset$ CAB(A α) – $\{\varepsilon\} \supset$ CAB(A)-$\{\varepsilon\} \supset$

(A-> β) CAB(β)-$\{\varepsilon\}$

2- Si CAB(β) = $\{\varepsilon\}$ => A -> ε, A->A α

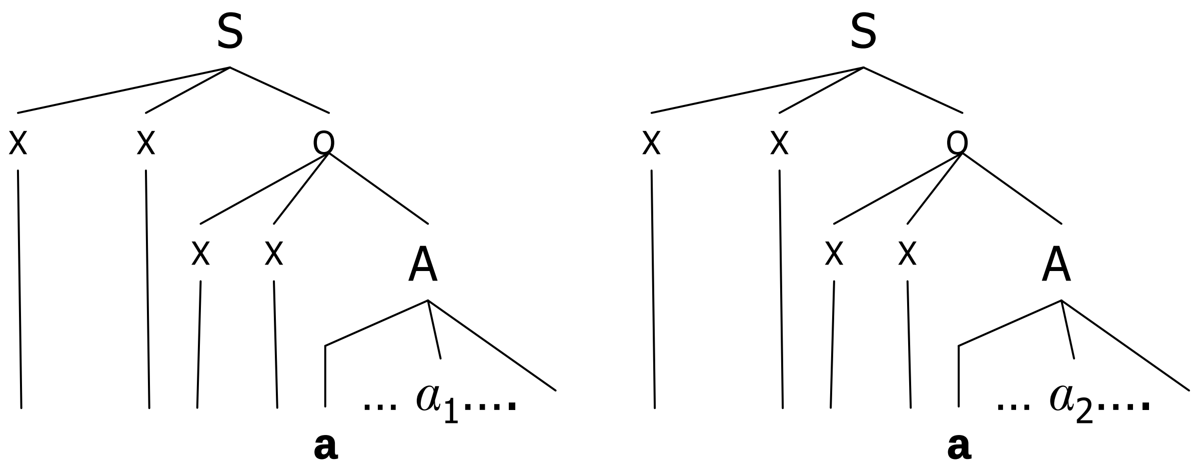
SD(A -> A α) $\supset$ CAB(A α) – $\{\varepsilon\} \supset$

($\varepsilon \in$ CAB(A)) CAB(α)-$\{\varepsilon\}$

SD(A -> ε) = SIG (A) $\supset$ (A->A α) CAB(α) – $\{\varepsilon\}$

- **Resultado-2**
  - Si una gramática es LL(1) => No es ambigua

<u>Dem.</u> Si fuera ambigua:



=> a $\in$ SD (A -> α_1) $\cap$ SD (A -> α_2)

Ejemplos Resultados

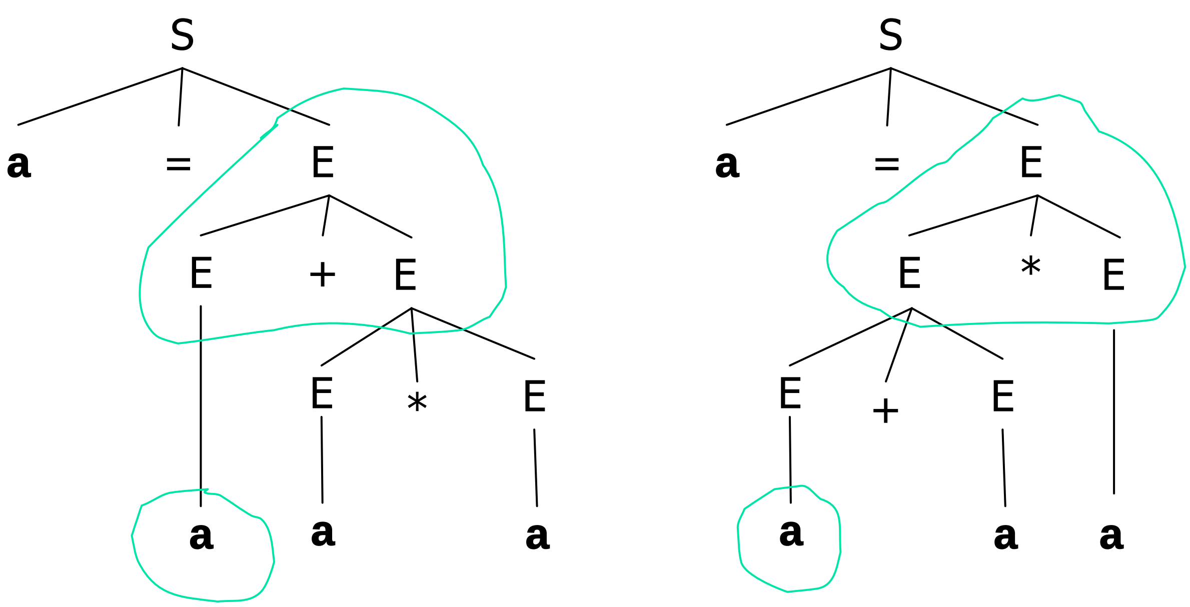
- Recursividad

  E -> E + T | T
  T -> T * F | F
  F -> a | (E)

  - SD (E -> E + T) = CAB (E+T) = CAB (E) = CAB(T) = CAB(T*F) U CAB (F) = CAB(F) = { (, a }
  - SD (E -> T) = CAB (T) = { (, a }

- Ambigüedad

  S -> a = E
  E -> E + E | E * E | a



a ∈ SD (E -> E+E) ∩ SD (E -> 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)

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

  $A \in N, \quad a \in T \cup \{\$\}$
  - Ejemplo

  S -> i E t S S′ | a

  S′ -> e S | ε

  E -> b

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

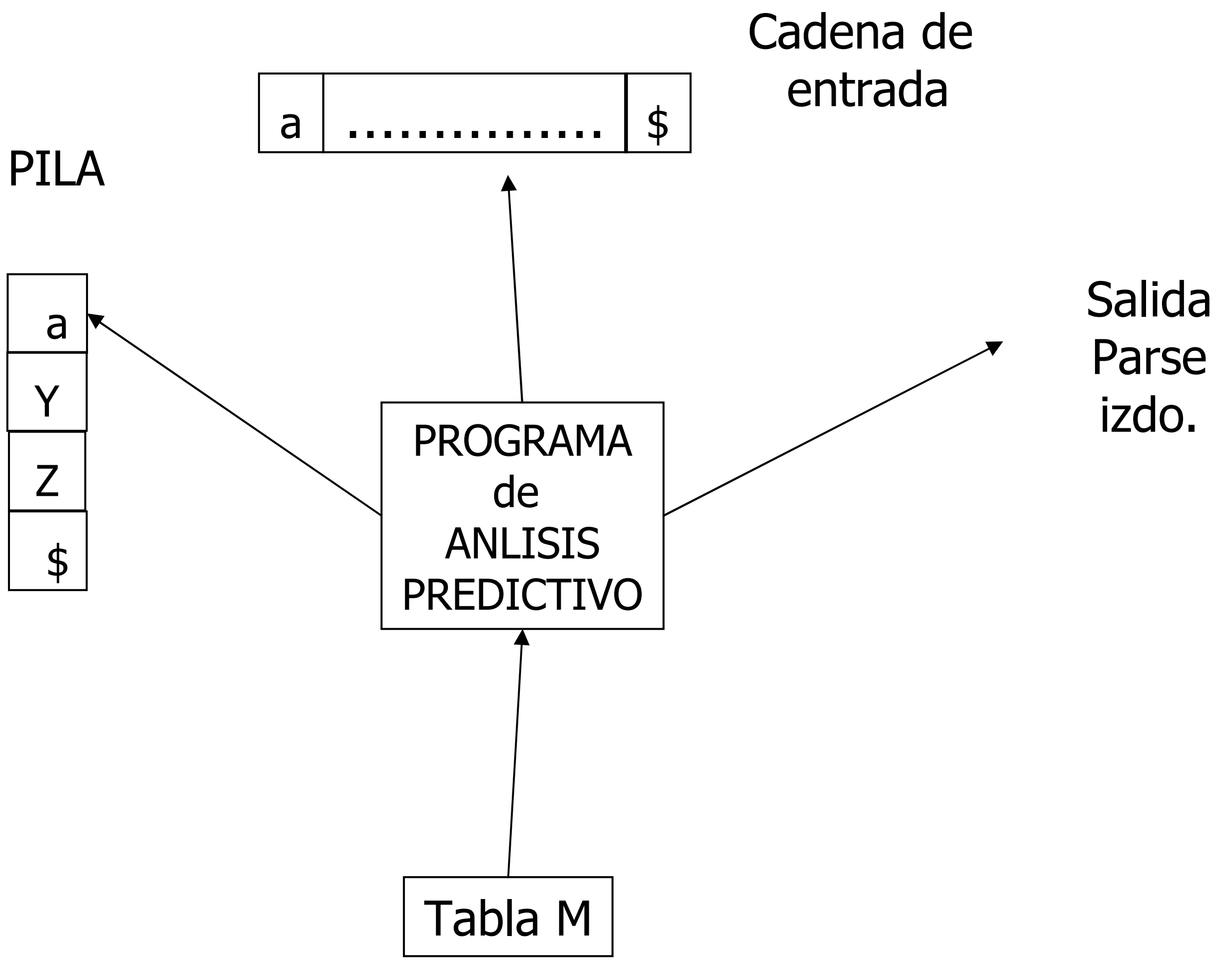
$e \in SD\ (S' \rightarrow eS) \cap SD\ (S' \rightarrow \varepsilon)$

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

  (Definición de Aho-Ullman de gram. LL(1))

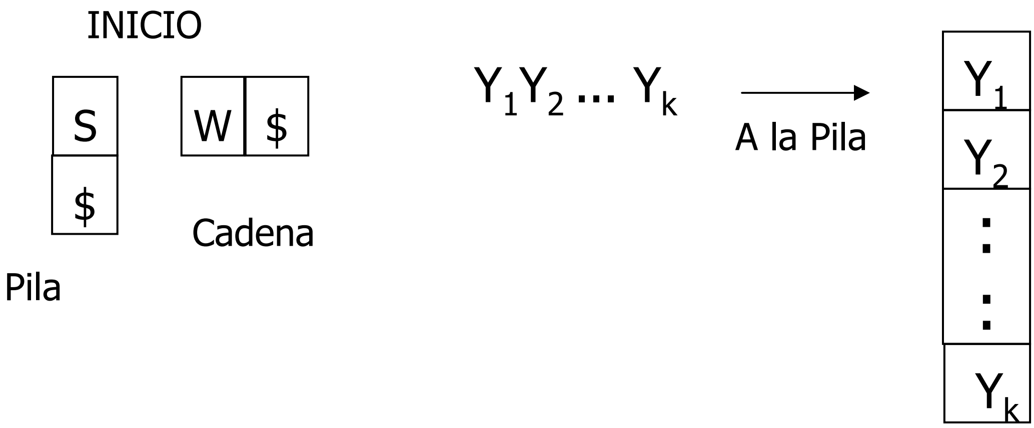
Reconocedor LL(1). Elementos

- 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 $\longrightarrow$ a

Si X $\in$ T ó X=$ entonces

Si X=a entonces

sacar X

avanzar ptr

else

error

else

Si M(X,a) = $Y_1Y_2 \ldots Y_k$ entonces

sacar X ; meter $Y_1Y_2 \ldots Y_k$

anotar en SALIDA X $\longrightarrow Y_1Y_2 \ldots Y_k$

else error

HASTA que la pila esté vacía

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

E-> T E' (1)

E' -> +T E' (2) | ε (3)

T -> F T' (4)

T' -> * F T' (5) | ε (6)

F -> (E) (7) | id (8)

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

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

SIG(E) — $;)

SIG(E') —
1 SIG(E)={$,) }
2 SIG (E') X

SIG's

SIG(T) —
1 CAB(E') ∋ + → SIG (E) = { $,) }
2 CAB (E') ∋ + → SIG (E') = { $,) }

SIG(T') —
4 SIG (T) = { +,), $}
5 SIG (T') X

SIG(F) —
4 CAB(T') ∋ * → SIG (T) = { $,) }
5 CAB (T') → SIG (T') = {+,), $ }

SD's

SD (E -> TE')=CAB (TE') —
- CAB (T) → CAB (FT') —
  - CAB (F)={(, id}
  - CAB (T) X
- CAB (E') X

SD (E' -> +TE')={+}

SD (E' -> ε) —
- CAB (ε)= ε
- SIG(E')={), $}

SD (T' ->*FT') = {*}

SD (F -> (E)) = {(}

SD (F -> id) = {id}

SD (T->FT')=CAB (FT') —
- CAB (F)={(, id}
- CAB (T') X

SD (T' -> ε)=SIG(T')={+,),$}

Ejemplo.Reconocedor aplicado a cadena

E-> T E′ (1)

E′ -> +T E′ (2) | ε (3)

T -> F T′ (4)

T′ -> * F T′ (5) | ε (6)

F -> (E) (7) | id (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") ;
}
```

$$A \rightarrow \underbrace{X_{11}\ X_{12}\ X_{13} \ldots X_{1M1}}_{\alpha_1}$$

$$A \rightarrow \underbrace{X_{21}\ X_{22}\ X_{23} \ldots X_{2M2}}_{\alpha_2}$$

$$A \rightarrow \underbrace{X_{n1}\ X_{n2}\ X_{n3} \ldots X_{nMn}}_{\alpha_n}$$

- TERMINALES

$\forall a \in T$

```
   funcion_a ( )
{
          if simbolo = = 'a'
          {
                leer_simbolo ( ) ;
          }
          else
          {
                     error
          }
}
```

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

- NO TERMINALES

```
funcion_A ( )  {
          switch (simbolo)    {
                    case    SD(A ->α1) :
                                 X11; X12 ; ... ;  X1M1 ;
                    case    SD(A ->α2) :
                                 X21; X22 ; ... ;  X2M2 ;
                    .....
                    case    SD(A ->αn) :
                                 Xn1; Xn2 ; ... ;  XnMn ;
                    default:
                                 error( ) ;
          }
}
```

Ejemplo. Implementación A.D. Recursivo.

E-> T E' (1)

E' -> +T E' (2) | ε (3)

T -> F T' (4)

T' -> * F T' (5) | ε (6)

F -> (E) (7) | id (8)

```
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 +,),$: ;
        default: error();
    }        SD(T'->*FT')={*}
}            SD(T'-> ε)={+,),$}
```

```
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.

$A \to A\,\alpha_1 \qquad A \to \beta_1$
$A \to A\,\alpha_2 \qquad A \to \beta_2$
$\ldots \qquad \ldots$
$A \to A\,\alpha_n \qquad A \to \beta_m$

$A' \to \alpha_1 A' \qquad A \to \beta_1 A'$
$A' \to \alpha_2 A' \qquad A \to \beta_2 A'$
$\ldots \qquad \ldots$
$A' \to \alpha_n A' \qquad A \to \beta_m A'$
$A' \to \varepsilon$

$\beta_j\, \alpha_{i_1}\, \alpha_{i_2} \ldots \alpha_{i_r}$

$A \Rightarrow A\, \alpha_{i_r} \Rightarrow A\, \alpha_{i_{r-1}}\, \alpha_{i_r}$

$\Rightarrow A\, \alpha_{i_1} \ldots \alpha_{i_{r-1}}\, \alpha_{i_r}$

$\Rightarrow \beta_j\, \alpha_{i_1} \ldots \alpha_{i_{r-1}}\, \alpha_{i_r}$

$\beta_j\, \alpha_{i_1}\, \alpha_{i_2} \ldots \alpha_{i_r}$

$A \Rightarrow \beta_j A' \Rightarrow \beta_j\, \alpha_{i_1} A'$

$\Rightarrow \beta_j\, \alpha_{i_1}\, \alpha_{i_2} A' \Rightarrow \ldots \Rightarrow$

$\Rightarrow \beta_j\, \alpha_{i_1} \ldots \alpha_{i_{r-1}}\, \alpha_{i_r}$

<u>EJEMPLOS</u>

A -> A a | a

A′ -> aA′ | ε
A -> aA′

E -> E + T | T
T -> T * F | F
F -> (E) | id

E′ -> + T E′ | ε
E -> T E′
T′ -> * F T′ | ε
T -> F T′
F -> (E) | 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 $A_1, A_2, \ldots, A_N$
  for i=1 to N {
      for j=1 to i-1 {
      reemplazar reglas $A_i \rightarrow A_j\gamma$ (i>j)
      $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \ldots \mid \delta_k \gamma$
      donde $A_j \rightarrow \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$
      }
      Eliminar la recursividad izda si existe para A_i
  }

- No debe tener reglas-ε, por ejemplo:

  S -> ASc, A->Bb | ε,

si S < A < B, entonces no hay que hacer nada, pero sigue habiendo ciclos: S => ASc => Sc

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

  A -> B, B -> A

si A < B, entonces aplicando el algoritmo,
A->B, B->B, B'->B', sigue habiendo recursividad.

EJEMPLO

S -> Aa | b
A -> Ac | Sd | c

$\Rightarrow$

S -> Aa | b
A -> Ac
A -> Aad | bd
A -> c

$\Rightarrow$

S -> Aa | b
A' -> cA' | ε
A' -> adA'
A -> bd A'
A -> cA'

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”

$$
\left.\begin{array}{lll}
A & \rightarrow & \alpha\,\beta_1 \\
 & | & \alpha\,\beta_2 \\
 & | & \\
 & \ldots & \\
 & | & \alpha\,\beta_n
\end{array}\right\}
\Rightarrow
\begin{array}{lll}
A \rightarrow \alpha\, A' & & \\
A' & \rightarrow & \beta_1 \\
 & | & \beta_2 \\
 & | & \beta_3 \\
 & \ldots & \\
 & | & \beta_n
\end{array}
$$

<u>EJEMPLOS</u>

1) <sent> -> if <expr> then <sent> else <sent>
   if <expr> then <sent>

   <sent> -> if <expr> then <sent> <parte_else>
   <parte_else> -> else <sent>
   | ε
   (sigue sin ser LL(1))

2) S -> abB | abC | aDh | c
   B -> b, C -> c, D -> d

   S -> aA’ | c
   A’ -> bB | bC | Dh
   B-> b, C->c, D->d

   S -> aA’ | c
   A’ -> bB’ | Dh
   B’ -> B|C, B->b,
   C->c, D->d

Nota. Si añadimos D -> 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**
  - $A_k \text{ -> } X_p\, X_{p+1} \dots X_n$

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

$$\Big\downarrow \text{ Reducción}$$

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

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

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

$$\Big\downarrow \text{ Desplazamiento}$$

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

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

- Las formas sentenciales se dividen siempre en dos

$$\alpha \quad | \quad \beta$$

Símbolos leídos
Y donde se han
Realizado reducciones

Símbolos que que
quedan por
leer

$\longleftarrow$ Desplazamiento

$\longrightarrow$ Anular desplazamiento

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

ALGORITMO ASCENDENTE CON RETROCESO

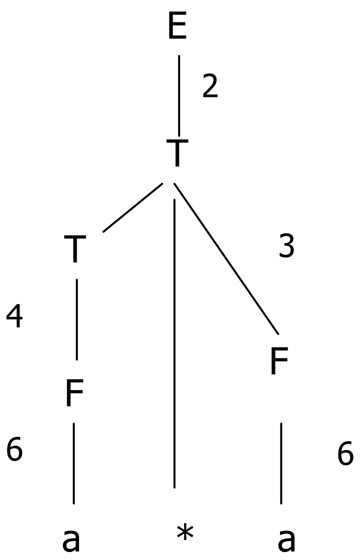
```
Ensayar  (α,  β)
{
     for    k=1    hasta el número de reglas
     {
             if (consecutente  (regla k) ==
              alguna subcadena cola de α)
           {
             (reducir regla k en esa subcadena α
              if ((α == Axioma) && ( β == ε))
              {
                      ACEPTAR
              {
               else                         }
              {
                    Ensayar (α,  β)
               }
                Anular reducción
             } /* if consecuente */
         } ( * for * /
         if   (β  ! =  ε)
         {
              Desplazar
              Ensayar (α,  β)
              Anular desplazamiento
         }
     }
     ERROR
```

Ejemplo A.S.A. con Retroceso

E -> E + T (1) | T (2)   T -> T * F (3) | F (4)   F -> (E) (5) | 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
- **L R (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)

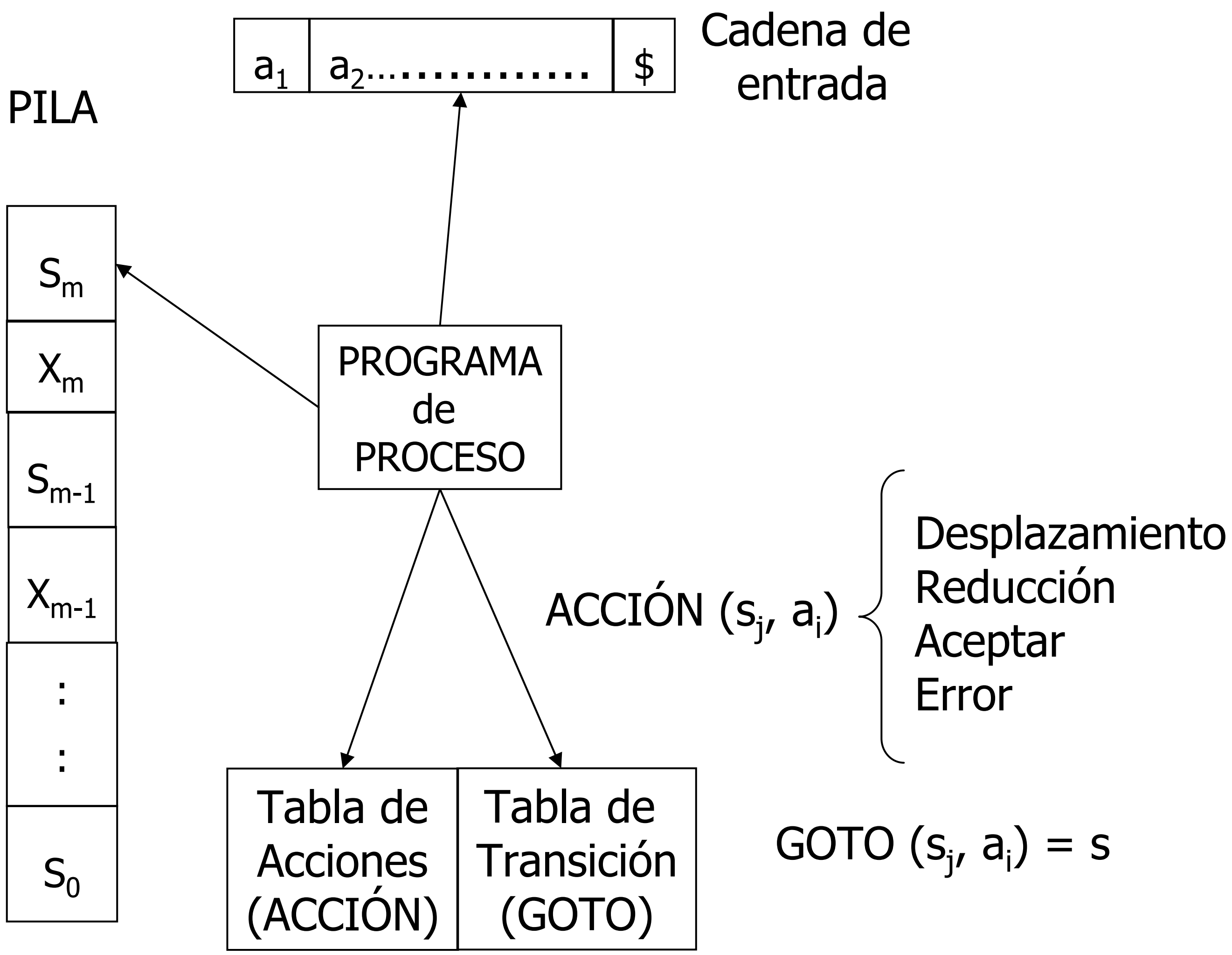
SLR(1)
∩
LALR(1)
∩
LR(1)

Analizadores más complejos

Más gramáticas

- 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

- $(\underbrace{s_0X_1s_1\ldots X_ms_m,}_{\text{PILA}}\ \underbrace{a_i\ a_{i+1}\ \ldots\ a_n\ \$}_{\text{CINTA ENTRADA}})$
- Configuración Inicial
  - $(s_0,\ a_1a_2\ldots\ a_n\ \$)$
- Operaciones sobre el analizador
  - ACCION $(s_m,\ a_i)$

    "top" de la pila

    Símbolo de la cinta de entrada

1. Desplazamiento

   GOTO $(s_m,\ a_i) = s$

$(s_0X_1s_1\ldots X_ms_m,\ a_i\ a_{i+1}\ldots a_n\$) \vdash (s_0X_1s_1\ldots X_ms_ma_is,\ a_{i+1}\ldots a_n\$)$

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

$(s_0X_1s_1\ldots X_ms_m,\ a_i\ a_{i+1}\ldots a_n\$) \vdash (s_0X_1s_1\ldots X_rs_rA,\ a_i\ a_{i+1}\ldots a_n\$) \vdash$

$(s_0X_1s_1\ldots X_rs_rAs,\ a_i\ a_{i+1}\ldots a_n\$)$ donde 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 -> S + T (1) | T (2)
T -> T * F (3) | F (4)
F -> (S) (5) | a (6)

	ACCIÓN (GOTO)							GOTO		
Estado	**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				

Paso	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	

ANÁLISIS SLR ("SIMPLE LR")

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

  si $\exists$ A -> $\alpha_1 \alpha_2$, entonces [A -> $\alpha_1 \circ \alpha_2$]
- A -> $\varepsilon \Rightarrow$ [A -> $\circ$]
- Puede **representarse**:

  (NÚMERO DE REGLA, POSICIÓN DEL PUNTO)
- **INTUTITIVAMENTE**: Cuánto de una regla se ha visto en un punto concreto del proceso de análisis.

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

- **ITEM COMPLETO**: [A -> $\circ\ \tau$]
- **CONJUNTO ϕ**: Cjto de Todos los items de una gramática. Ejemplo:

  S->E, E-> E+T | E-T | T, T->(E) | a

  ϕ={ [S -> $\circ$ E], [S -> E $\circ$], [E -> $\circ$ E+T],

  [E->E $\circ$+T] [E->E+ $\circ$T], [E->E+T $\circ$], … }
- **GRAMÁTICA AUMENTADA**: de G a G'

  redefinición del axioma S, nuevo axioma S' $\in$ N, S'->S
- **CIERRE**: $\wp(\phi) \rightarrow \wp(\phi)$

  I $\subset \phi \rightarrow$ CIERRE (I) $\subset \phi$

  1) I $\subset$ CIERRE (I)

  2) Si [A -> $\alpha \circ$ B β] $\in$ CIERRE(I) entonces

  $\forall$ B -> γ regla, [B -> $\circ\ \gamma$] $\in$ CIERRE (I)

- **EJEMPLO**

  G= { E'->E, E->E+T|T, T->T+F|F, F->(E), F->id}

  CIERRE ([E'-> ∘ E]} = { [E'-> ∘ E], [E'-> ∘ E+T],
  [E-> ∘T], [T-> ∘ T*F], [T->∘ F], [F-> ∘(E)], [F-> ∘ id] }

- **SUBCADENA VIABLE** α

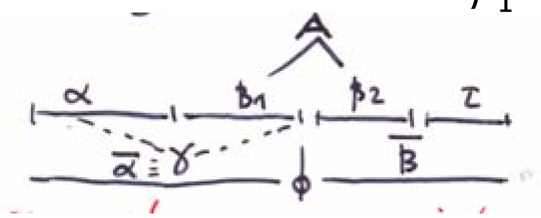
  $\alpha \in (N \cup T)^*$ es una cadena viable sii (def)

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

  (subcadena cabeza de alguna forma sentencial)

- **ITEM VÁLIDO PARA UNA SUBCADENA VIABLE**
  - [A -> $\beta_1 \circ \beta_2$] para una subcadena viable "$\alpha\ \beta_1$"

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



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

- **FUNCIÓN DE TRANSICIÓN** = GOTO

  δ (goto) : $\wp(\phi)$ x (N U T) → $\wp(\phi)$

  $I \subset \phi \qquad (I, X) \quad \rightarrow \delta\,(I,X) \equiv$ CIERRE($\Re$)

  donde $\Re$={[A-> α X ∘ β] / [A-> α ∘ Xβ] $\in$ I}

- **EJEMPLO-1**

  G= { E'->E, E->E+T|T, T->T+F|F, F->(E), F->id}

  δ ({[E'->E ∘], [E->E ∘+T]}, +) =

  CIERRE {[E->E+ ∘T]} = {[E->E+ ∘ T], [T-> ∘ T*F],
  [T-> ∘ F], [F-> ∘(E)], [F-> ∘ id]}

Por qué la Función Goto (δ) ?

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

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

  DEMOSTRACIÓN

  1) Sea [A-> $\alpha \circ$ X β] $\in$ I. Si es item válido para γ , $\exists$ τ / γ = τ α y

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

  Ahora, esto quiere decir que

  [A-> αX $\circ$ β] $\in$ δ (I,X) es item válido para "γ X"

  2) Sea [A-> $\alpha \circ$ X β] $\in$ I, y β=Bθ entonces

  [A-> αX $\circ$ β] $\in$ δ (I,X) y [B -> $\circ$ μ_i] $\in$ δ (I,X),

  $\forall$ B -> μ_i

  Ya que [A-> $\alpha \circ$ X β] es item válido para γ,

  $\exists$ τ / γ = τ α y

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

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

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

  [B -> $\circ$ μ_i] es un item válido para la subcadena viable "γ X"

Colección de conjuntos de items LR(0)

C = $\{I_0, I_1, I_2, I_3, \ldots\}$ $I_j \subset \phi$ 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'->S)
2. Algoritmo
3. **C** = $\{I_0\}$ donde I_0=CIERRE{[S' -> $\circ$ S]}

REPETIR

Para cada I $\in$ **C** y cada X $\in$ N U T,

si δ (I,X) $\neq \emptyset$ y δ (I,X) $\notin$ **C**

entonces añadir $I_j \equiv \delta$ (I,X) a **C**

HASTA que no puedan añadirse a **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 U T

   q_0=[S' -> $\circ$ S] F = {[A -> $\tau \circ$] –es decir items completos-}

   [A-> α a $\circ$ β] $\in \delta$ ([A-> α $\circ$ a β], a)

   [A-> α B $\circ$ β] $\in \delta$ ([A-> α $\circ$ B β], B)

   [B -> $\circ$ τ] $\in \delta$ ([A-> α $\circ$ B β], ε)
2. Hacer este automáta determinista

Construcción de un Análisis LR(0)

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

1. $S_i \equiv I_i \in$ **C**
2. $\delta \equiv$ Goto
3. La tabla de acciones No depende del símbolo en la cinta de entrada
   a. Si I_j= {[A -> $\tau \circ$]} $\Rightarrow$ ACCION(I_j)=Reducir por A -> τ
   b. Si I_j= {[S' -> S \$ $\circ$]} $\Rightarrow$ ACCION(I_j)=Aceptar
   c. Si I_j no tiene ningún item completo $\Rightarrow$ ACCION(I_j)=Desplazar

- Si existe algún estado (s_j) cuyo conjunto de items (I_j) contiene más de un item, 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-> $\tau \circ$] item 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 <u>necesaria</u> para aplicar la regla A -> τ, aunque <u>no suficiente</u>
- 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 -> $\alpha \circ$ a β] $\in I_j$
   b) ACCIÓN (I_j , a)=R por A -> τ ($A \neq S'$)
      si [A -> $\tau \circ$] $\in I_j$ y $a \in$ SIG(A)
   c) ACCION (I_j , \$)=ACEPTAR si [S' -> S $\circ$] $\in I_j$
   d) Resto de acciones = Error
5. Si no hay duplicidad en la Tabla Acción $\Rightarrow$ Gramát. es SLR(1)