

Tema 6 | Análisis Sintáctico Ascendente LR

Contenidos

6.1 Analizadores LR(k).

6.2 Descripción Funcional.

6.3 Método de Construcción de la Tabla de Análisis SLR.

6.3.1 Conceptos de *prefijo viable*, funciones *clausura* y *goto*.

6.3.2 Construcción del Autómata Finito Determinista.

6.3.3 Algoritmo para la Construcción de la Tabla de Análisis SLR.

6.4 Construcción de la Tabla de Análisis LR(1).

6.5 Construcción de la Tabla de Análisis LALR(1).

6.6 Tratamiento de la Ambigüedad.

6.7 Detección y Recuperación de los errores.

6.8 YACC (Generador de Analizadores Sintácticos)

6.8.1 Entorno de YACC.

6.8.2 Utilización de YACC.

6.8.3 Especificación de YACC.

Bibliografía básica

- [Aho90] Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman
Compiladores. Principios, técnicas y herramientas. Addison-Wesley
Iberoamericana 1990.
- [Levi92] J.R. Levine, T. Manson, D. Brown
Lex & Yacc. O'Reilly & Associates, Inc. 1992.
- [Benn90] J.P. Bennet
*Introduction to compiling techniques: A first course using ANSI C,
LEX and YACC*. McGraw-Hill 1990.

20-Feb-2011

6.1 Analizadores LR(k)

Surgen como alternativa a los problemas que presentan las restricciones de gramáticas LL(n) para análisis descendente y las gramáticas de precedencia en análisis ascendente. En el siguiente ejemplo se ilustra el problema del error de reducción en análisis de precedencia simple.

Ejemplo 6.1: Sea G una gramática definida por las siguientes producciones, dada la secuencia de entrada **a f c**, aplicando la estrategia de análisis ascendente mediante precedencia simple y dadas las relaciones de precedencia obtenidas por medio del método deductivo:

$$\begin{array}{lcl} S & \rightarrow & aAb \\ & | & bAc \\ & | & aAd \\ A & \rightarrow & h \\ & | & f \end{array}$$

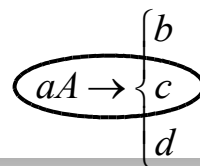
Relaciones de Precedencia

$$\begin{array}{ll} \$ < \cdot a & c \cdot > \$ \\ a < \cdot f & a = \cdot A \\ a < \cdot h & A = \cdot c \\ & f \cdot > c \end{array}$$

Pila	Acción	Entrada
\$	<·	a f c \$
\$ a	<·	f c \$
\$ a f	·>	c \$
\$ a A	=·	c \$
\$ a A c	ERROR de Reducción	\$

En precedencia sólo se mira el símbolo que está en el tope de la pila con el símbolo de la entrada, ignorando los anteriores a la pila. De esa forma, puede suceder que no se forme una secuencia correcta de símbolos para reducir.

Proceso de análisis seguido
(**Error de reducción**)



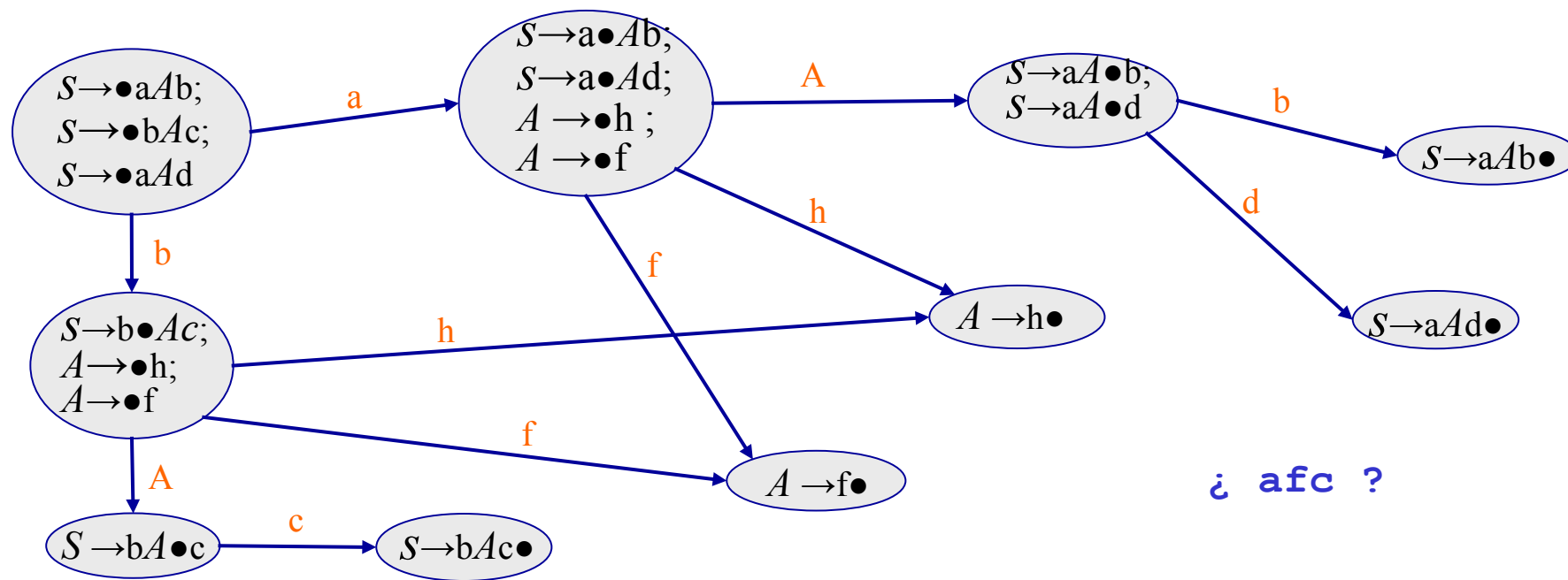
Proceso correcto
(**Secuencias válidas**)

$$aA \rightarrow \begin{cases} b \\ d \end{cases}$$

6.1 Analizadores LR(k) (2)

$S \rightarrow aAb$
 $\quad \mid bAc$
 $\quad \mid aAd$
 $A \rightarrow h$
 $\quad \mid f$

¿Cómo podemos tener en cuenta lo que precede a cada símbolo terminal?
 Observando la gramática ejemplo las cadenas del lenguaje pueden comenzar por: **a b ; ah af bh bf ; ahb afb ahd afd bhc bfc**
 Esto es como si pasáramos por distintas situaciones de la forma siguiente:



6.1 Analizadores LR(k) (3)

Es el método más general y el más usado.

L - Analiza los símbolos de la entrada de izquierda a derecha (left-right)

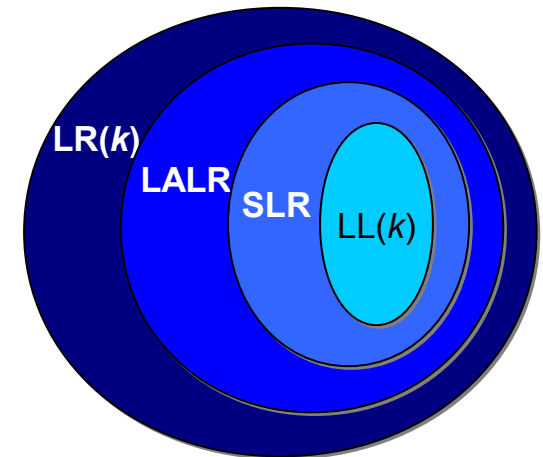
R – Reduce a la derecha (right).

k – Número de símbolos de anticipación de la entrada.

La confección de la tabla de análisis es algo más compleja que en los métodos predictivos presentados con anterioridad, si bien, resulta muy mecánica la forma de obtenerla.

Existen tres técnicas para la confección de la tabla de análisis. Cada una de ellas caracteriza a tres tipos de gramáticas LR que son:

1. **LR Simple (SLR)**: Es la más fácil de implantar pero la menos potente.
2. **LR Canónica o LR(1)**: Es la más potente pero la más compleja.
3. **LALR** (LR con anticipación o look-ahead): Es la técnica intermedia y más usada. Se trata de una simplificación del método general LR(1) y usado en la implementación de los generadores de analizadores sintácticos basados en YACC.



La técnica de análisis LR es válido para gramáticas de contexto libre no ambiguas.

6.1 Analizadores LR(k) (4)

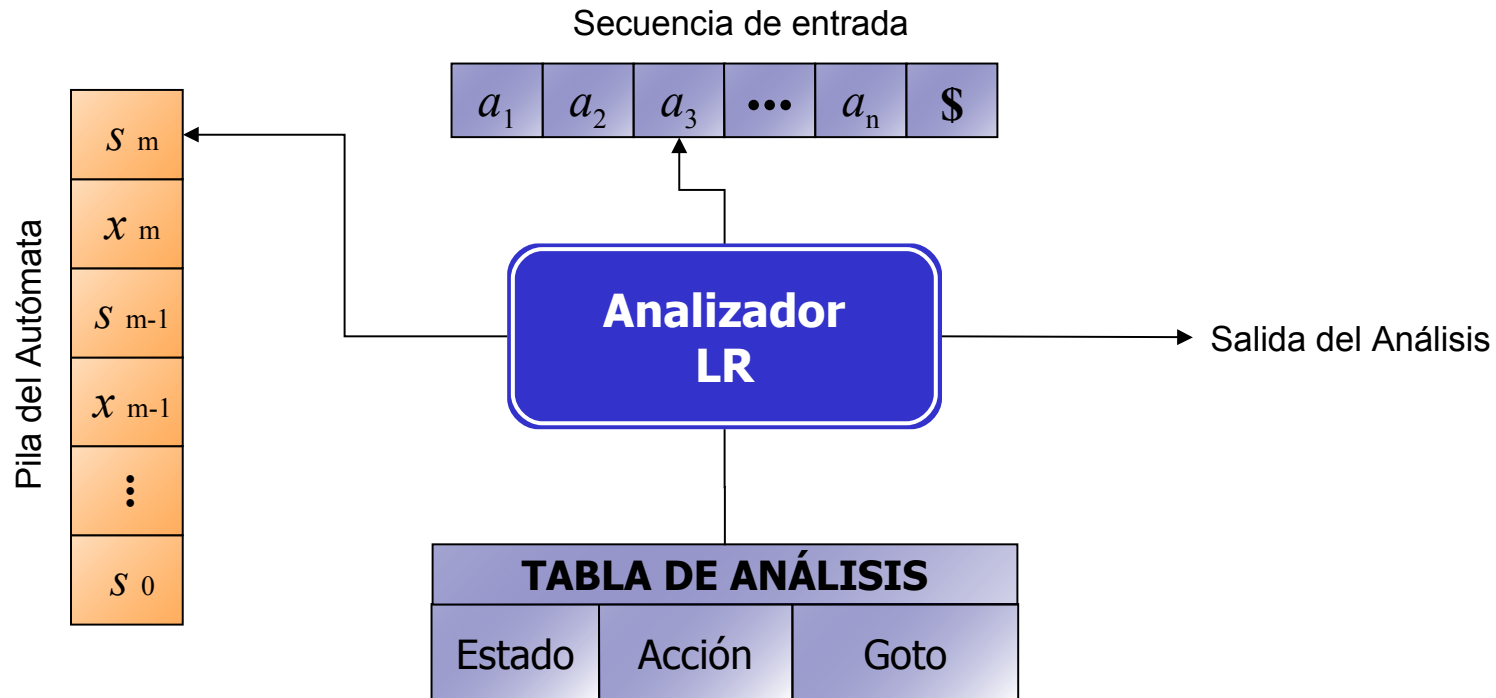
Ventajas de los analizadores LR(k)

1. Puede reconocer virtualmente todos los lenguajes que obedecen a una gramática libre de contexto.
2. Es el método conocido más general sin retroceso, así como el más fácil de implementar de los conocidos de Reducción y Desplazamiento.
3. Las clases de gramáticas que pueden analizarse mediante el analizador LR son el superconjunto de las gramáticas que pueden analizarse por analizadores predictivos.
4. El analizador LR puede detectar errores conforme se realiza el análisis, así como aplicar recuperación del análisis con pérdida controlada.

6.2 Descripción Funcional (1)

La pila del autómata de pila utilizado para el análisis LR está formado por parejas de **símbolos** (terminales y no terminales) y **estados**.

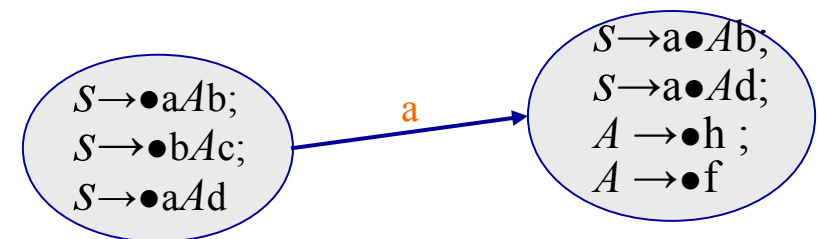
$$\underbrace{(S_0 X_1 S_1 X_2 S_2 \cdots X_m S_m)}_{\text{Pila}}, \underbrace{a_i a_{i+1} \cdots a_n \$}_{\text{Entrada}}$$



6.2 Descripción Funcional (2)

1. Si $\text{acción}[S_m, a_i] = \text{Desplaza}$, el analizador inserta en la pila tanto el símbolo de entrada a_i como el siguiente estado S que se obtiene fruto de la evaluación $\text{Goto}[S_m, a_i]$. Ahora, a_{i+1} pasa a ser el símbolo de la entrada a leer y el analizador se encuentra ahora en el estado S .

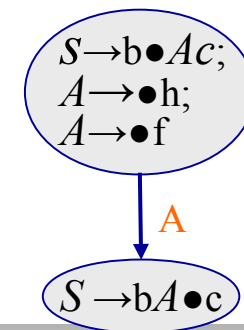
$$\begin{array}{c} (S_0 X_1 S_1 X_2 S_2 \cdots X_m S_m, a_i a_{i+1} \cdots a_n \$) \\ \Downarrow \\ (S_0 X_1 S_1 X_2 S_2 \cdots X_m S_m a_i S, a_{i+1} \cdots a_n \$) \end{array}$$



2. Si $\text{acción}[S_m, a_i] = \text{Reduce}$, entonces el analizador ejecuta una reducción utilizando la regla $A \rightarrow \beta$ resultando la siguiente configuración:

$$\begin{array}{c} (S_0 X_1 S_1 X_2 S_2 \cdots X_m S_m a_i S, a_{i+1} \cdots a_n \$) \\ \Downarrow \\ (S_0 X_1 S_1 X_2 S_2 \cdots X_{m-r} S_{m-r} A S, a_{i+1} \cdots a_n \$) \end{array}$$

Donde $S = \text{Goto}[S_{m-r}, A]$ y r la longitud de la cadena de símbolos de β . Se extraen $2r$ elementos de la pila (estados y símbolos) y después se inserta el símbolo no terminal A .



3. Si $\text{acción}[S_m, a_i] = \text{Acepta}$, el análisis se ha terminado con éxito.
4. Si $\text{acción}[S_m, a_i] = \text{Error}$, se ha descubierto un error. El analizador entonces llamará al procedimiento de recuperación de errores.

6.2 Descripción Funcional (3)

Algoritmo para el Analizador LR

Hacer que p apunte al primer símbolo de $w\$$;

repeat

Sea S el estado situado en la parte superior de la pila;

Sea a el símbolo al que apunta p ;

if $accion[S, a] = Desplaza S'$ **then**

Cargar a y luego S' en la pila;

Hacer que p apunte al siguiente símbolo;

else if $accion[S, a] = Reduce A \rightarrow \beta$ **then**

Sacar de la pila $2|\beta|$ símbolos de la pila;

Sea S' el estado situado en el tope de la pila;

Cargar A y el resultado de $goto[S', A]$ en la pila;

Ejecutar las acciones semánticas de $A \rightarrow \beta$;

else if $accion[S, a] = Aceptar$ **then**

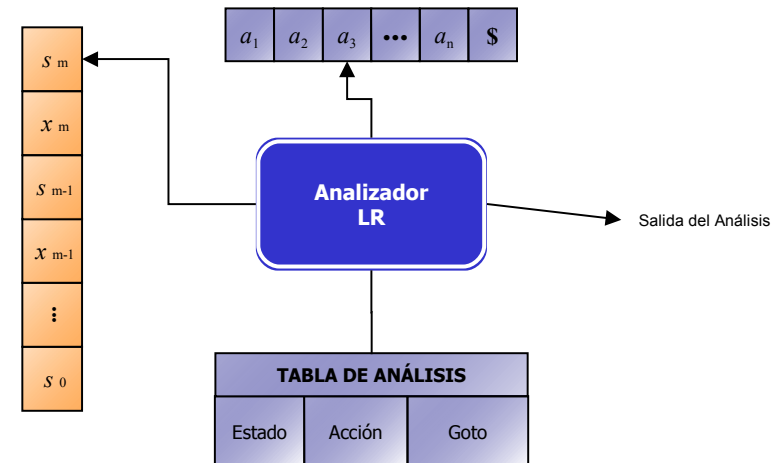
return;

else

error();

end if

until forever



Ejemplo 6.2: Dada la gramática descrita por las producciones enumeradas siguientes, la tabla de análisis asociada y dada la secuencia de entrada **id*id+id**, la evolución del análisis resulta como sigue:

Nº	Producciones
1	$E \rightarrow E + T$
2	$E \rightarrow T$
3	$T \rightarrow T * F$
4	$T \rightarrow F$
5	$F \rightarrow (E)$
6	$F \rightarrow id$

Estado	acción							goto		
	id	+	*	()	\$		E	T	F
0	s ₅			s ₄				1	2	3
1		s ₆				acc				
2		r ₂	s ₇		r ₂	r ₂				
3		r ₄	r ₄		r ₄	r ₄				
4	s ₅			s ₄				8	2	3
5		r ₆	r ₆		r ₆	r ₆				
6	s ₅			s ₄					9	3
7	s ₅			s ₄						10
8		s ₆			s ₁₁					
9		r ₁	s ₇		r ₁	r ₁				
10		r ₃	r ₃		r ₃	r ₃				
11		r ₅	r ₅		r ₅	r ₅				

s_i : Desplaza al estado i .

r_j : Reduce usando la producción j .

acc : Aceptar.

Pila	Entrada	Acción
0	id * id + id \$	Desplaza
0 id 5	* id + id \$	Reduce: $F \rightarrow id$
0 F 3	* id + id \$	Reduce: $T \rightarrow F$
0 T 2	* id + id \$	Desplaza
0 T 2 * 7	id + id \$	Desplaza
0 T 2 * 7 id 5	+ id \$	Reduce: $F \rightarrow id$
0 T 2 * 7 F 10	+ id \$	Reduce: $T \rightarrow T * F$
0 T 2	+ id \$	Reduce: $E \rightarrow T$
0 E 1	+ id \$	Desplaza
0 E 1 + 6	id \$	Desplaza
0 E 1 + 6 id 5	\$	Reduce: $F \rightarrow id$
0 E 1 * 6 F 3	\$	Reduce: $T \rightarrow F$
0 E 1 + 6 T 9	\$	Reduce: $E \rightarrow E + T$
0 E 1	\$	Aceptar

6.3 Método de Construcción de la Tabla de Análisis SLR (1)

Pivote: Es una subcadena que coincide con la parte derecha de una producción.

Prefijo viable: Sea $\phi\beta t$ una cadena de símbolos y sea β una subcadena pivote y t un símbolo terminal. Se define un prefijo viable como la secuencia de símbolos que se pueden formar de izquierda a derecha hasta el símbolo t .

Sea $\phi\beta = u_1 u_2 \cdots u_r$ y dada $B \rightarrow \beta$ una producción de la gramática, un prefijo viable será cualquier subcadena de la forma $u_1 u_2 \cdots u_i$, donde $1 \leq i \leq r$, es decir:

$$u_1 ; u_1 u_2 ; u_1 u_2 u_3 \cdots$$

Teorema 7.1 *El conjunto de todos los prefijos viables asociados con las partes derechas de las producciones de una gramática pueden ser reconocidas por una máquina de estados finitos.*

(Ver demostración en [Knuth65])

Items o configuración de una gramática: Será una producción representada entre $[,]$ y con un "." en la parte derecha de la producción.

Dada la producción $A \rightarrow \alpha_1 \alpha_2$, el conjunto de items asociados será:

$$\left\{ \begin{array}{l} A \rightarrow \cdot \alpha_1 \alpha_2 \\ A \rightarrow \alpha_1 \cdot \alpha_2 \\ A \rightarrow \alpha_1 \alpha_2 \cdot \end{array} \right.$$

6.3 Método de Construcción de la Tabla de Análisis SLR (2)

Objetivo: Construir un **autómata finito determinista** que reconozca los **prefijos viables** a partir de una gramática. Los estados del autómata finito determinista se obtienen en base a agrupar los items asociados con cada producción de la gramática.

Construcción del **AFD** para el analizador **SLR**.

Se basa en los conceptos de **Gramática aumentada**, **operación de clausura** y función **goto**.

1. **Gramática aumentada.** Sea S el símbolo inicial de una gramática G . Se define otra gramática G' llamada aumentada, tal que el símbolo inicial de la gramática es S' y aparece la producción $S' \rightarrow S$.
2. **Operación de clausura** de un conjunto de items de una gramática: Sea I el conjunto de items. La clausura de I , que notaremos como $\text{clausura}(I)$, será otro conjunto de items obtenidos por las reglas siguientes:
 - a) $\{I\} \subset \{\text{clausura}(I)\}$,
 - b) Si $[A \rightarrow \alpha \cdot B \gamma]$ y existe la producción $B \rightarrow \beta$, entonces $[B \rightarrow \cdot \beta]$ y $[A \rightarrow \alpha \cdot B \gamma] \in \{\text{clausura}(I)\}$.

Ejemplo 6.3: Sean las producciones siguientes:

$$E' \rightarrow E ; E \rightarrow E + T \mid T ; T \rightarrow T * F \mid F ; F \rightarrow (E) \mid id$$

$$I = \{ [E' \rightarrow \cdot E] \}$$

$$\text{clausura}(I) = \{ [E' \rightarrow \cdot E], [E \rightarrow \cdot E + T], [E \rightarrow \cdot T], [E \rightarrow \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \}.$$

6.3 Método de Construcción de la Tabla de Análisis SLR (3)

3. Función goto. Es una función de transición de estados alcanzables a partir de uno dado y según el símbolo que haya en la entrada.

Sea I el conjunto de items, sea X un símbolo cualquiera de la gramática. La función goto del conjunto de items I dado un símbolo X de la gramática, notado como $\text{goto}[I, X]$ como la clausura de items del tipo:

$$[A \rightarrow \alpha X \cdot \beta] \text{ tal que } [A \rightarrow \alpha X \cdot \beta] \in I.$$

Ejemplo 6.4: Dada la gramática aumentada tomada del ejemplo anterior.

$$\begin{array}{l} E' \rightarrow E \\ E \rightarrow E+T \mid T \\ T \rightarrow T^*F \mid F \\ F \rightarrow (E) \mid id \end{array}$$

y sea el conjunto de items siguiente:

$$I = \{ [E' \rightarrow E \cdot], [E \rightarrow E \cdot + T] \}$$

Entonces:

$$\text{goto}[I, +] = \{ [E \rightarrow E + \cdot T] \}$$

Posteriormente se debe obtener el siguiente estado que se alcanza. Para ello calculamos la función $\text{clausura}(\text{goto}[I, +])$ que resultaría:

$$\text{clausura}([E \rightarrow E + \cdot T]) = \{ [E \rightarrow E + \cdot T], [T \rightarrow \cdot T^* F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \}$$

6.3 Método de Construcción de la Tabla de Análisis SLR (4)

Estados asociados con la gramática SLR.

Proceso de obtención:

1. El estado inicial I_0 se define como la *clausura* ($[S \rightarrow \cdot \alpha]$), donde S es el símbolo inicial de la gramática.
2. Los estados sucesivos se obtiene como $I_n = goto(I_{n-1}, N)$ para todo símbolo terminal o no terminal con el punto delante que aparecen en los items en I_{n-1} .
3. Se aplica el paso (2) hasta que no se generen nuevos estados.

Tema 6. Análisis Sintáctico Ascendente LR

Ejemplo 6.5: Sea la gramática aumentada de los ejemplos anteriores. Obtener el conjunto de estados asociados con la gramática SLR.

En primer lugar calculamos el estado inicial I_0 obteniendo lo siguiente:

$$I_0 = \{ [E' \rightarrow \cdot E], [E \rightarrow \cdot E + T], [E \rightarrow \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \}$$

$ \begin{aligned} &E' \rightarrow E \\ &E \rightarrow E + T \mid T \\ &T \rightarrow T * F \mid F \\ &F \rightarrow (E) \mid id \end{aligned} $

El resto de estados son:

$$\begin{aligned}
 I_1 &= goto(I_0, E) = \{ [E' \rightarrow E \cdot], [E \rightarrow E \cdot + T] \} \\
 I_2 &= goto(I_0, T) = \{ [E \rightarrow T \cdot], [T \rightarrow T \cdot * F] \} \\
 I_3 &= goto(I_0, F) = \{ [T \rightarrow F \cdot] \} \\
 I_4 &= goto(I_0, () = \{ [F \rightarrow (\cdot E)], [E \rightarrow \cdot E + T], [E \rightarrow \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], \\
 &\quad [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \} \\
 I_5 &= goto(I_0, id) = \{ [F \rightarrow id \cdot] \} \\
 I_6 &= goto(I_1, +) = \{ [E \rightarrow E + \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \} \\
 I_7 &= goto(I_2, *) = \{ [T \rightarrow T * \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \} \\
 I_8 &= goto(I_4, E) = \{ [F \rightarrow (E \cdot)], [E \rightarrow E \cdot + T] \} \\
 I_9 &= goto(I_6, T) = \{ [E \rightarrow E + T \cdot], [T \rightarrow T \cdot * F] \} \\
 I_{10} &= goto(I_7, F) = \{ [T \rightarrow T * F \cdot] \} \\
 I_{11} &= goto(I_8,)) = \{ [F \rightarrow (E) \cdot] \}
 \end{aligned}$$

Conjunto de estados y **AFD** asociado de la gramática anterior:

$$I_0 = \{ [E' \rightarrow \cdot E], [E \rightarrow \cdot E + T], [E \rightarrow \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \}$$

$$I_1 = goto(I_0, E) = \{ [E' \rightarrow E \cdot], [E \rightarrow E \cdot + T] \}$$

$$I_2 = goto(I_0, T) = \{ [E \rightarrow T \cdot], [T \rightarrow T \cdot * F] \}$$

$$I_3 = goto(I_0, F) = \{ [T \rightarrow F \cdot] \}$$

$$I_4 = goto(I_0, () = \{ [F \rightarrow (\cdot E)], [E \rightarrow \cdot E + T], [E \rightarrow \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \}$$

$$I_5 = goto(I_0, id) = \{ [F \rightarrow id \cdot] \}$$

$$I_6 = goto(I_1, +) = \{ [E \rightarrow E + \cdot T], [T \rightarrow \cdot T * F], [T \rightarrow \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \}$$

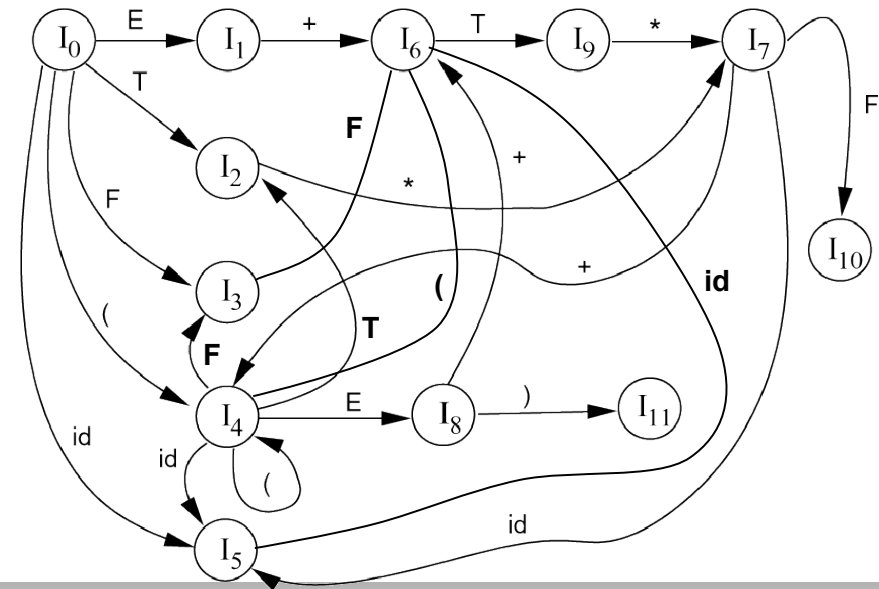
$$I_7 = goto(I_2, *) = \{ [T \rightarrow T * \cdot F], [F \rightarrow \cdot (E)], [F \rightarrow \cdot id] \}$$

$$I_8 = goto(I_4, E) = \{ [F \rightarrow (E \cdot)], [E \rightarrow E \cdot + T] \}$$

$$I_9 = goto(I_6, T) = \{ [E \rightarrow E + T \cdot], [T \rightarrow T \cdot * F] \}$$

$$I_{10} = goto(I_7, F) = \{ [T \rightarrow T * F \cdot] \}$$

$$I_{11} = goto(I_8,)) = \{ [F \rightarrow (E) \cdot] \}$$



6.3 Método de Construcción de la Tabla de Análisis SLR (5)

Algoritmo para construir la tabla de análisis SLR.

Entrada: La gramática aumentada.

Salida: Las funciones *acción* y *goto*.

Método:

1. Construir $C = \{I_1, I_2, \dots, I_n\}$ el conjunto de estados asociados con la gramática aumentada G' .
2. Desde el estado I_i se determina la acción del analizador según los casos siguientes:
 - a) Si $[A \rightarrow \alpha \cdot a \beta] \in I_i$ y $\text{goto}(I_i, a) = I_j$, entonces
Acción $[i, a] = (\text{Desplaza}, j)$, donde a es símbolo terminal.
 - b) Si $[A \rightarrow \alpha \cdot] \in I_i$, entonces para todo $a \in \text{Seguidores}(A)$ y $A \neq S'$,
Acción $[i, a] = (\text{Reduce}, A \rightarrow \alpha)$.
 - c) Si $[S' \rightarrow S \cdot] \in I_i$, entonces
Acción $[i, \$] = \text{Aceptar}$.
 - d) Si no corresponde a ningún otro caso, se dice que la gramática no es SLR.
3. Para todo símbolo no terminal se construye la función $\text{goto}(i, A)$.
Si $\text{goto}(I_i, A) = I_j$, entonces $\text{goto}(i, A) = j$.
4. Las entradas en la tabla de análisis sin acciones serán etiquetadas con *error*.
5. El estado inicial se construye como la *clausura* $([S' \rightarrow \cdot S])$.

SLR vs. LR/LALR (1)

En gramáticas SLR, las reducciones se aplican ante cualquier símbolo seguidor del símbolo no terminal. En algunos casos podría provocar situaciones de error en el análisis (gramática no sería SLR).

Ejemplo: Considerar la siguiente gramática y realizar el cálculo del AFD y la tabla de análisis SLR.

- (1) $S' \rightarrow S$
- (2) $S \rightarrow Aa$
- (3) $S \rightarrow bAc$
- (4) $S \rightarrow dc$
- (5) $S \rightarrow bda$
- (6) $A \rightarrow d$

$I_0 = \text{clausura}([S' \rightarrow \cdot S]) = \{[S' \rightarrow \cdot S], [S \rightarrow \cdot Aa], [S \rightarrow \cdot bAc], [S \rightarrow \cdot dc], [S \rightarrow \cdot bda], [A \rightarrow \cdot d]\}$

$I_1 = \text{goto}(I_0, S) = \{[S' \rightarrow S \cdot]\}$ (R_{aceptar})

$I_2 = \text{goto}(I_0, A) = \{[S \rightarrow A \cdot a]\}$

$I_3 = \text{goto}(I_0, b) = \{[S \rightarrow b \cdot Ac], [S \rightarrow b \cdot da], [A \rightarrow \cdot d]\}$

$I_4 = \text{goto}(I_0, d) = \{[S \rightarrow d \cdot c], [A \rightarrow d \cdot]\}$ (R_6)

$I_5 = \text{goto}(I_2, a) = \{[S \rightarrow Aa \cdot]\}$ (R_2)

$I_6 = \text{goto}(I_3, A) = \{[S \rightarrow bA \cdot c]\}$

$I_7 = \text{goto}(I_3, d) = \{[S \rightarrow bd \cdot a], [A \rightarrow d \cdot]\}$ (R_6)

$I_8 = \text{goto}(I_4, c) = \{[S \rightarrow dc \cdot]\}$ (R_4)

$I_9 = \text{goto}(I_6, c) = \{[S \rightarrow bAc \cdot]\}$ (R_3)

$I_{10} = \text{goto}(I_7, a) = \{[S \rightarrow bda \cdot]\}$ (R_5)

SLR vs. LR/LALR (2)

Tenemos 2 conflictos deslaza/reduce en la tabla de análisis.

$I_0 = \text{clausura}([S' \rightarrow \cdot S]) = \{[S' \rightarrow \cdot S], [S \rightarrow \cdot Aa], [S \rightarrow \cdot bAc], [S \rightarrow \cdot dc], [S \rightarrow \cdot bda], [A \rightarrow \cdot d]\}$

$I_1 = \text{goto}(I_0, S) = \{[S' \rightarrow S \cdot]\}$ (R_{aceptar})

$I_2 = \text{goto}(I_0, A) = \{[S \rightarrow A \cdot a]\}$

$I_3 = \text{goto}(I_0, b) = \{[S \rightarrow b \cdot Ac], [S \rightarrow b \cdot da], [A \rightarrow \cdot d]\}$

$I_4 = \text{goto}(I_0, d) = \{[S \rightarrow d \cdot c], [A \rightarrow d \cdot]\}$ (R_6)

$I_5 = \text{goto}(I_2, a) = \{[S \rightarrow Aa \cdot]\}$ (R_2)

$I_6 = \text{goto}(I_3, A) = \{[S \rightarrow bA \cdot c]\}$

$I_7 = \text{goto}(I_3, d) = \{[S \rightarrow bd \cdot a], [A \rightarrow d \cdot]\}$ (R_6)

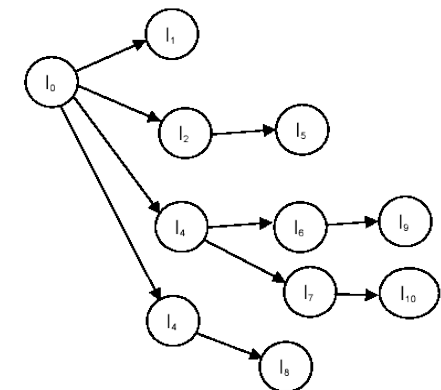
$I_8 = \text{goto}(I_4, c) = \{[S \rightarrow dc \cdot]\}$ (R_4)

$I_9 = \text{goto}(I_6, c) = \{[S \rightarrow bAc \cdot]\}$ (R_3)

$I_{10} = \text{goto}(I_7, a) = \{[S \rightarrow bda \cdot]\}$ (R_5)

- (1) $S' \rightarrow S$
- (2) $S \rightarrow Aa$
- (3) $S \rightarrow bAc$
- (4) $S \rightarrow dc$
- (5) $S \rightarrow bda$
- (6) $A \rightarrow d$

Estado	Acción					Goto	
	a	b	c	d	\$	S	A
0		s3		s4		1	2
1					acc		
2	s5						
3				s7			6
4	r6		s8 /r6				
5					r2		
6			s9				
7	s1 0/r 6		r6				
8					r4		
9					r3		
10					r5		



SLR vs. LR/LALR (3)

Ejemplo de análisis de entrada: **b d a**

- (1) $S' \rightarrow S$
- (2) $S \rightarrow Aa$
- (3) $S \rightarrow bAc$
- (4) $S \rightarrow dc$
- (5) $S \rightarrow bda$
- (6) $A \rightarrow d$

Estado	Acción					Goto	
	a	b	c	d	\$	S	A
0		s3		s4		1	2
1					acc		
2	s5						
3				s7			6
4	r6		s8 /r6				
5					r2		
6			s9				
7	s1 0/r 6		r6				
8				r4			
9				r3			
10				r5			

PILA	ENTRADA	ACCIÓN
0	b d a \$	Desplaza (3)
0 b 3	d a \$	Desplaza (7)
0 b 3 <u>d</u> 7	a \$	Desplaza (10) o Reduce [A→d]

Si optamos por <i>desplazar</i>		
0 b 3 d 7 a <u>10</u>	\$	Reduce [S→bda]
0 S 1	\$	Aceptar

Si optamos por <i>reducir</i>		
0 b 3 A 6	a \$	ERROR

SLR vs. LR/LALR (4)

Ejemplo de análisis de entrada: **b d c**

Estado	Acción					Goto	
	a	b	c	d	\$	S	A
0		s3		s4		1	2
1					acc		
2	s5						
3				s7			6
4	r6		s8 /r6				
5					r2		
6			s9				
7	s1 0/r 6		r6				
8					r4		
9					r3		
10					r5		

- (1) $S' \rightarrow S$
- (2) $S \rightarrow Aa$
- (3) $S \rightarrow bAc$
- (4) $S \rightarrow dc$
- (5) $S \rightarrow bda$
- (6) $A \rightarrow d$

PILA	ENTRADA	ACCIÓN
0	b d c \$	Desplaza (3)
0 b 3	d c \$	Desplaza (7)
0 b 3 <u>d</u> 7	c \$	Reduce [$A \rightarrow d$]
0 b 3 A 6	c \$	Desplaza (9)
0 <u>b</u> 3 A 6 c 9	\$	Reduce [$S \rightarrow bAc$]
0 S 1	\$	Aceptar

SLR vs. LR/LALR (5)

Otro problema que surge en SLR es el ejemplo de sentencias para llamada a procedimiento y la sentencia de asignación. En ambos casos se comienza con un identificador. Hasta llegar al siguiente token (en este caso la asignación “=”) no se sabría aplicar la acción adecuada.

Ejemplo 6.6: Sea la gramática aumentada que opera sobre llamada a procedimientos y sentencia de asignación.

- (0) $S' \rightarrow S$
- (1) $S \rightarrow id$
- (2) $\quad | V = E$
- (3) $V \rightarrow id$
- (4) $E \rightarrow V$
- (5) $\quad | num$

Conflictos en la tabla SLR:

Accion						goto			
Estado	id	=	num	ERROR	\$	Estado	S	V	E
0	s(2)					0	1	3	
1					Aceptar	1			
2		r(3)			r(1)	2			
3		s(4)			r(1)	3			
4	s(8)		s(7)		r(3)	4		6	5
5					r(2)	5			
6					r(4)	6			
7					r(5)	7			
8		r(3)			r(3)	8			

Conflicto reduce/reduce

6.4 Construcción de la Tabla de Análisis LR(1) (1)

El proceso de análisis LR(1) significa llevar un símbolo de anticipación para decidir el análisis.

Se extiende el concepto de **item con símbolo de anticipación**, así como, la función *clausura* y *goto*.

Items con anticipación:

Será de la forma $[A \rightarrow \alpha \cdot \beta, a]$, donde a es un símbolo terminal de la gramática.

Significado: Sea $[A \rightarrow \alpha \cdot, a]$. Se reducirá mediante $A \rightarrow \alpha$ si sólo si, el siguiente símbolo de entrada es a .

Validez de los items.

El item $[A \rightarrow \alpha \cdot \beta, a]$ es válido para el prefijo $\delta\alpha$ si $S \Rightarrow^* \delta A \omega \Rightarrow \delta \alpha \beta \omega$; y $\omega \Rightarrow a \omega'$ y $a \in \text{Iniciales}(\omega)$ en el caso que $\omega = \epsilon$ $S \Rightarrow^* \delta \alpha \beta \$$

6.4 Construcción de la Tabla de Análisis LR(1) (2)

Clausura.

Se calcula por las reglas siguientes:

1. I_0 es la clausura($[S' \rightarrow \cdot S, \$]$).
2. Sea el item $[A \rightarrow \alpha \cdot B \beta, a]$ y la producción $B \rightarrow \eta$, entonces habrá que incluir en la clausura el items $[B \rightarrow \cdot \eta, b]$ donde $b \in \text{Iniciales}(\beta a)$.

Justificación de la regla (2):

Si $\delta\alpha$ es prefijo válido de $[A \rightarrow \alpha \cdot B \beta, a]$ implica $S \Rightarrow^* \delta A \omega \Rightarrow \delta \alpha B \beta \omega$ donde $a \in \text{Iniciales}(\omega)$, también $\delta\alpha\eta$ es prefijo válido de $[B \rightarrow \cdot \eta, b]$ si $\delta\alpha B \beta \Rightarrow \delta\alpha\eta\beta$ donde $b \in \text{Iniciales}(\beta a)$.

Ejemplo 6.7: Sean las producciones siguientes:

$$S' \rightarrow S$$

$$S \rightarrow CC$$

$$C \rightarrow aC \mid d$$

$$I_0 = \text{clausura}([S' \rightarrow S]) = \{ [S' \rightarrow \cdot S, \$], [S \rightarrow \cdot CC, \$], [C \rightarrow \cdot aC, a/d], [C \rightarrow \cdot d, a/d] \}$$

$$I_1 = \text{goto}(I_0, S) = \{ [S' \rightarrow S \cdot, \$] \}$$

$$I_2 = \text{goto}(I_0, C) = \{ [S \rightarrow C \cdot C, \$], [C \rightarrow \cdot aC, \$], [C \rightarrow \cdot d, \$] \}$$

$$I_3 = \text{goto}(I_0, a) = \{ [C \rightarrow a \cdot C, a/d], [C \rightarrow \cdot aC, a/d], [C \rightarrow \cdot d, a/d] \}$$

$$I_4 = \text{goto}(I_0, d) = \{ [C \rightarrow d \cdot, a/d] \}$$

$$I_5 = \text{goto}(I_2, C) = \{ [S \rightarrow CC \cdot, \$] \}$$

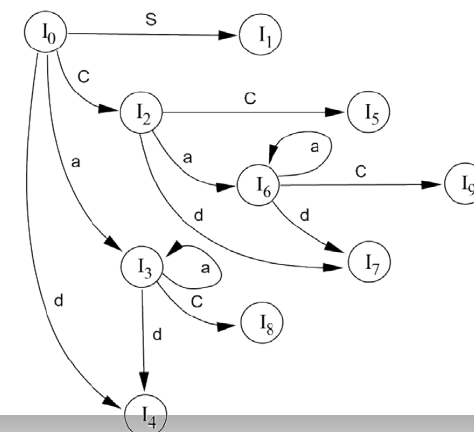
$$I_6 = \text{goto}(I_2, a) = \{ [C \rightarrow a \cdot C, \$], [C \rightarrow \cdot aC, \$], [C \rightarrow \cdot d, \$] \}$$

$$I_7 = \text{goto}(I_2, d) = \{ [C \rightarrow d \cdot, \$] \}$$

$$I_8 = \text{goto}(I_3, C) = \{ [C \rightarrow aC \cdot, a/d] \}$$

$$I_9 = \text{goto}(I_6, C) = \{ [C \rightarrow aC \cdot, \$] \}$$

Estado	Acción			Goto	
	a	d	S	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		



6.4 Construcción de la Tabla de Análisis LR(1) (3)

Algoritmo para construir la tabla LR(1).

Entrada: La gramática aumentada.

Salida: Las funciones *acción* y *goto*.

Método:

1. Construir $C = \{I_1, I_2, \dots, I_n\}$ el conjunto de estados asociados con la gramática aumentada G' .
2. Desde el estado I_i se determina la acción del analizador según los casos siguientes:
 - a) Si $[A \rightarrow \alpha \cdot a\beta, b] \in I_i$ y $goto(I_i, a) = I_j$, entonces
Acción $[i, a] = (\text{Desplaza}, j)$, a símbolo terminal.
 - b) Si $[A \rightarrow \alpha \cdot, a] \in I_i$ y $A \neq S'$, entonces
Acción $[i, a] = (\text{Reduce}, A \rightarrow \alpha)$.
 - c) Si $[S' \rightarrow S \cdot, \$] \in I_i$, entonces
Acción $[i, \$] = \text{Aceptar}$.
 - d) Si no corresponde a ningún otro caso, se dice que la gramática no es LR.
3. Para todo símbolo no terminal se construye la función $goto(i, A)$.
Si $goto(I_i, A) = I_j$, entonces $goto(i, A) = j$.
4. Las entradas en la tabla de análisis sin acciones serán etiquetadas con *error*.
5. El estado inicial se construye como la *clausura* $([S' \rightarrow \cdot S, \$])$.

6.5 Construcción de la Tabla de Análisis LALR(1)

Algoritmo para construir la tabla LALR(1)

Entrada: La gramática aumentada.

Salida: Las funciones *acción* y *goto*.

Método:

1. Construir el conjunto de estados como si fuese LR(1).
2. Unir los estados formados por los mismos items con independencia de los símbolos de anticipación en los items, obteniendo un nuevo estado unión sin conflictos.
3. Considerar la presencia del estado j unido con i,k,h como el nuevo estado $jikh$.

Ejemplo 6.8: Dada la gramática del ejemplo anterior:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow CC \\ C &\rightarrow aC \mid d \end{aligned}$$

$$I_0 = \text{clausura}([S' \rightarrow S]) = \{ [S' \rightarrow \cdot S, \$], [S \rightarrow \cdot CC, \$], [C \rightarrow \cdot aC, a/d], [C \rightarrow \cdot d, a/d] \}$$

$$I_1 = \text{goto}(I_0, S) = \{ [S' \rightarrow S \cdot, \$] \}$$

$$I_2 = \text{goto}(I_0, C) = \{ [S \rightarrow C \cdot C, \$], [C \rightarrow \cdot aC, \$], [C \rightarrow \cdot d, \$] \}$$

$$I_3 = \text{goto}(I_0, a) = \{ [C \rightarrow a \cdot C, a/d], [C \rightarrow \cdot aC, a/d], [C \rightarrow \cdot d, a/d] \}$$

$$I_4 = \text{goto}(I_0, d) = \{ [C \rightarrow d \cdot, a/d] \}$$

$$I_5 = \text{goto}(I_2, C) = \{ [S \rightarrow CC \cdot, \$] \}$$

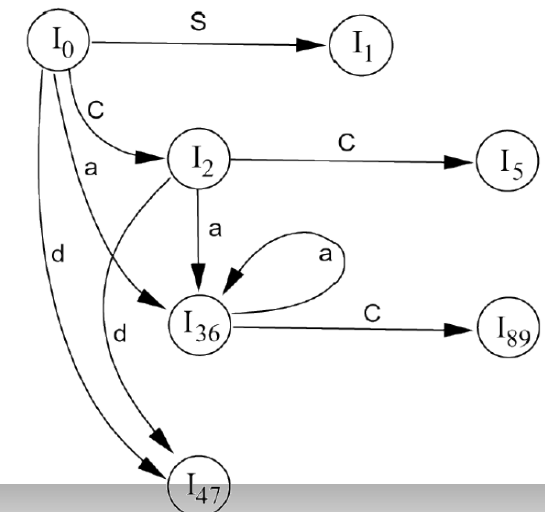
$$I_6 = \text{goto}(I_2, a) = \{ [C \rightarrow a \cdot C, \$], [C \rightarrow \cdot aC, \$], [C \rightarrow \cdot d, \$] \}$$

$$I_7 = \text{goto}(I_2, d) = \{ [C \rightarrow d \cdot, \$] \}$$

$$I_8 = \text{goto}(I_3, C) = \{ [C \rightarrow aC \cdot a/d] \}$$

$$I_9 = \text{goto}(I_6, C) = \{ [C \rightarrow aC \cdot, \$] \}$$

Estado	Acción			Goto	
	a	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		



6.6 Tratamiento de la Ambigüedad (1)

Todos los métodos de análisis sintáctico predictivos requieren que la **gramática no sea ambigua**. Sin embargo, hay algunos casos de **conflictos** que pueden resolverse adoptando algún **criterio** o en base a **información adicional** sobre el lenguaje (que no queda reflejado en la propia gramática), como ocurre con los operadores.

Conflictos reduce/desplaza:

- Si optamos por **reducir**, significa que estamos aceptando como válida la secuencia. En particular debemos asegurarnos que, bajo esa secuencia inicialmente aceptada, se alcance el símbolo inicial de la gramática, en el caso de que dicha secuencia fuese correcta.
- Si optamos por **desplazar**, actuamos de forma contraria (acción por defecto en YACC).

Conflictos reduce/reduce:

Estos conflictos se resuelven cómodamente eliminando producciones que resulten equivalentes.

6.6 Tratamiento de la Ambigüedad (2)

Ambigüedad entre los operadores: La **precedencia** y **asociatividad** de los operadores nos aportan criterios para eliminar los conflictos que puedan aparecer en la tabla de análisis.

Cuando en una celda de la tabla de análisis aparecen conflictos **desplaza/reduce** bajo un mismo símbolo operador, se actuará de la siguiente forma cuando se defina precedencia y asociatividad entre operadores:

Si $op_1 < op_2$, entonces en op_2 debe aparecer **desplaza** y en op_1 **reduce**.

Ambigüedad de los “else-balanceados”: Cuando en nuestra gramática disponemos de la sentencia *if-then-else* (sin uso de *endif*) se producirá el conflicto **desplaza/reduce**.

Para este caso ¿qué solución se debería adoptar?

Ejemplo 6.9: Dada la especificación de una sentencia que incluye la sentencia propia del *if-then-else*:

$Sentencia \rightarrow$ **if** <exp> **then** <sentencia> **else** sentencia |
if <exp> **then** <sentencia> |
 otras

Se puede diseñar una gramática aumentada de la siguiente forma:

$$S' \rightarrow S$$

$$S \rightarrow iSeS \mid iS \mid a$$

El análisis SLR arroja los siguientes estados:

$I_0 = clausura([S' \rightarrow \cdot S]) = \{ [S' \rightarrow \cdot S], [S \rightarrow \cdot iSeS], [S \rightarrow \cdot iS], [S \rightarrow \cdot a] \}$
 $I_1 = goto(I_0, S) = \{ [S' \rightarrow S \cdot] \}$
 $I_2 = goto(I_0, i) = \{ [S \rightarrow i \cdot SeS], [S \rightarrow i \cdot S], [S \rightarrow \cdot iSeS], [S \rightarrow \cdot iS], [S \rightarrow \cdot a] \}$
 $I_3 = goto(I_0, a) = \{ [S \rightarrow a \cdot] \}$
 $I_4 = goto(I_2, S) = \{ [S \rightarrow iS \cdot eS], [S \rightarrow iS \cdot] \} \leftarrow \text{Desplaza/Reduce}$
 $I_5 = goto(I_4, e) = \{ [S \rightarrow iSe \cdot S], [S \rightarrow \cdot iSeS], [S \rightarrow \cdot iS], [S \rightarrow \cdot a] \}$
 $I_6 = goto(I_4, S) = \{ [S \rightarrow iSeS \cdot] \}$

La solución que se adopta, que es la misma que adopta YACC, es tomar la asociatividad del **else** con el último **then**, de este modo, pasamos el **else** a la pila y adoptamos *Desplaza*.

6.7 Detección y Recuperación de los Errores (1)

La técnica para detección y recuperación de los errores en gramáticas LR se basan en acotar el contexto donde se ha producido el error:

- Modo Panic.
- Tabla de análisis.
- Métodos complejos basados en el algoritmo de Graham-Rhodes.

6.7 Detección y Recuperación de los Errores (2)

Modo Panic

Saltar símbolos de la entrada y/o sacar símbolos de la pila hasta alcanzar una situación válida para continuar el análisis (elimina el error y, probablemente, más información de entrada).

PROBLEMA: Decidir si se saca de la pila o se avanza en la entrada.

Como solución sería posible la inserción de **símbolos de sincronización**. Consiste en saltar símbolos de la entrada hasta alcanzar algún símbolo de sincronización.

PROBLEMA: Si se omiten algunos de los símbolos de sincronización se pueden perder demasiados símbolos de entrada.

SOLUCIÓN: Optimizar los símbolos de sincronización en base a los símbolos seguidores e iniciales de cada símbolo no terminal de la gramática.

6.7 Detección y Recuperación de los Errores (3)

Tabla de Análisis

Consiste en completar las celdas vacías de la tabla de análisis con procedimientos concretos para el tratamiento del error y recuperación del mismo para continuar con el análisis.

Estado	Acción			Goto	
	a	d	S	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		

Si el analizador se encontrase en esta situación, podría realizar las siguientes tareas:

- Informar del error diciendo: **esperaba "a" o "d"** (...que son los símbolos de entrada con transición válida en la tabla).
- Localizar aquellos símbolos que están en la pila y que rodean el error (sacarlos de la pila)
- Saltar símbolos de la entrada hasta poder continuar con el análisis.

6.7 Detección y Recuperación de los Errores (4)

Métodos Complejos Basados en el Algoritmo de Graham-Rhodes [Trem85]

Permite recuperarse ante un error y además, en algunos casos, reparar el mismo.

El método distingue dos fases:

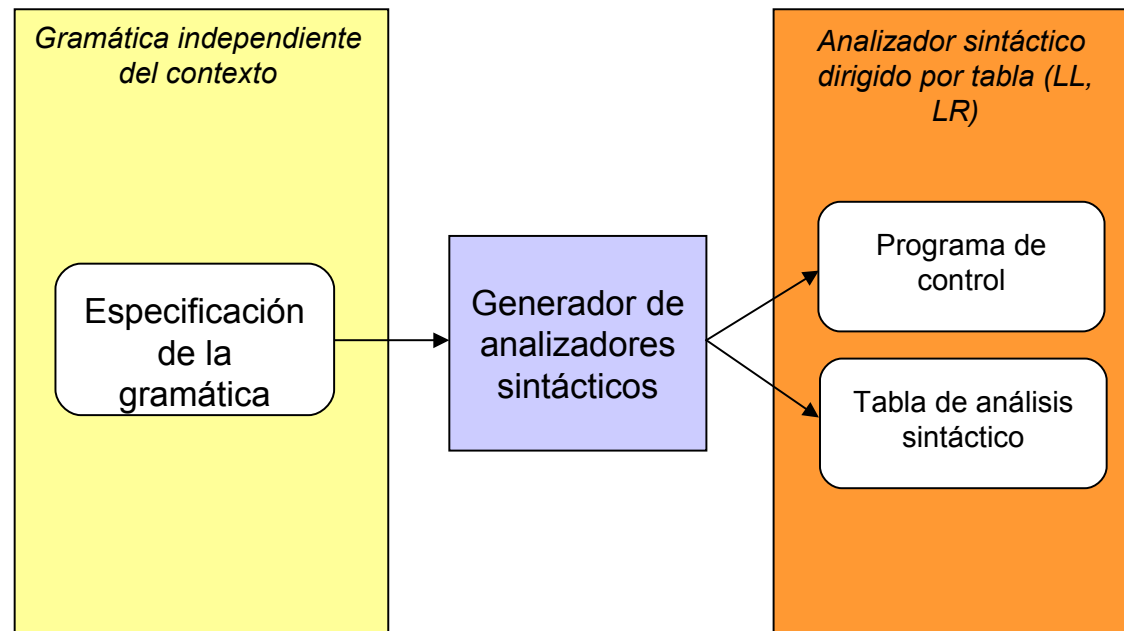
- **Fase de condensación:** donde se acota el contexto del error.
- **Fase de corrección:** donde se intenta corregir el error.

Basándose en el análisis de los símbolos próximos en el tope de la pila y en la entrada, la fase de corrección decide si hay que sacar de la pila o saltar en la entrada, actuando como si hubiese **insertado** o **ignorado** símbolos de la entrada.

Clasificación de las Gramáticas

Generalidad	LR(k)	LR(1)	LALR(1)	SLR	LL(1)	precedencia
Facilidad para escribir las gramáticas	LR	LL(1)				
Facilidad para depurar	LL(1)	LR				
Detección de errores	LR	LL(1)				
Complejidad espacial	LR	LALR o SLR	LL(1)	Precedencia		
Complejidad temporal	SLR					
Facilidad semántica	LR					

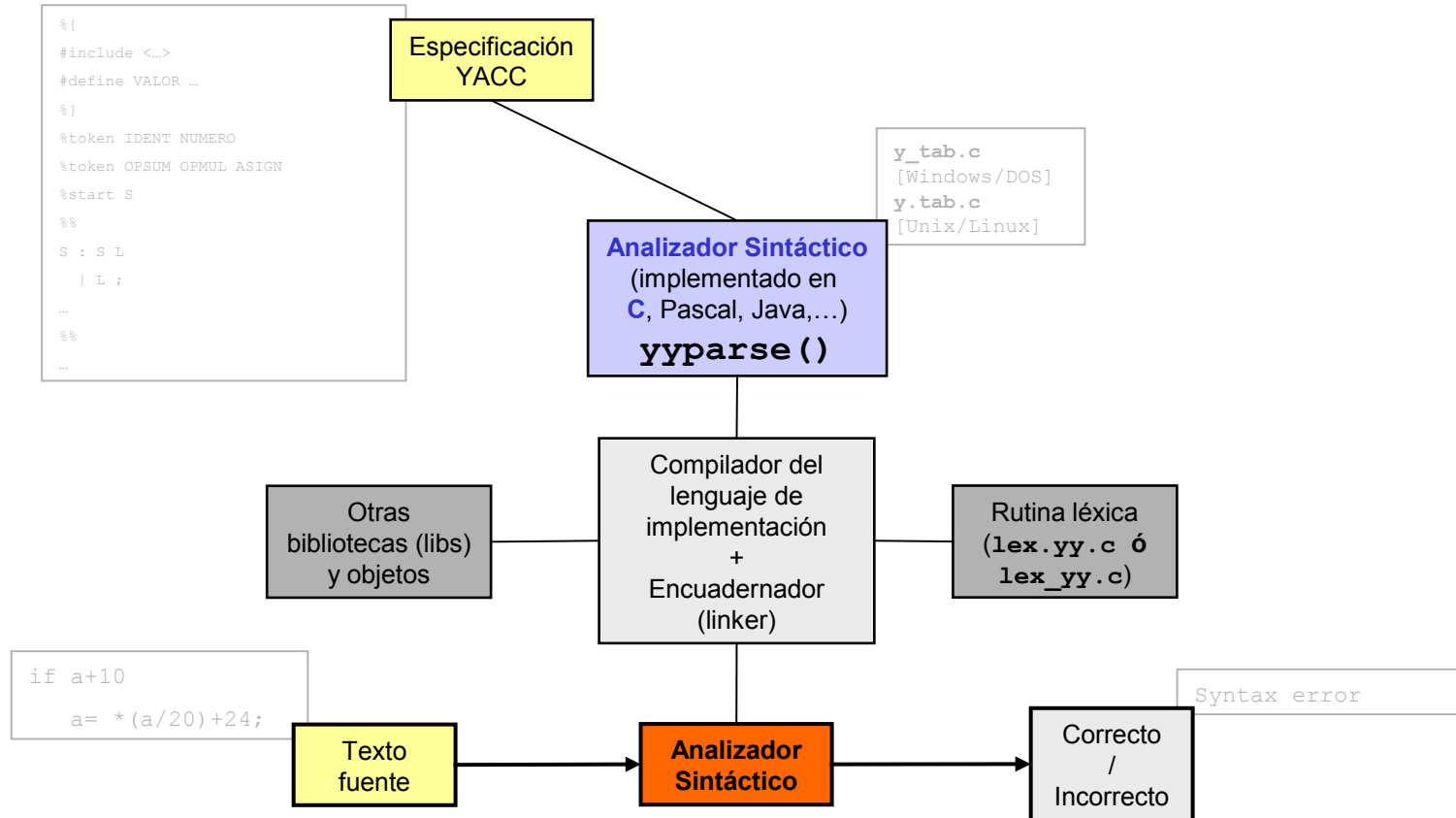
6.8. YACC (Generadores de Analizadores Sintácticos)



Generador de analizadores sintácticos: **YACC**

- Análisis sintáctico ascendente basado en gramáticas **LALR(1)**.
- Requiere la rutina léxica **yylex()** para el reconocimiento previo de los tokens (usando **LEX**).

6.8.1 Entorno de YACC (1)



6.8.1 Entorno de YACC (2)

Opciones en la llamada de YACC:

BYACC [opciones] nombre_del_fichero_de_especificaciones.

- v genera la tabla de análisis sintáctico en el fichero y.out.
- b prefijo permite cambiar el prefijo y del nombre del fichero de salida por el especificado.
- t permite realizar un "debugging" del análisis sintáctico.
- d genera el fichero de cabecera y_tab.h, que contiene la definición de los símbolos (tokens).
- r genera dos ficheros de salida, y_code.c que contiene el código y y_tab.c que contiene la tabla de análisis.

6.8.1 Entorno de YACC (3)

Analizador de léxico: Para realizar la función del analizador de léxico, el YACC utiliza una función denominada `yylex()` que devuelve un entero, cuyo valor se debe asociar a la variable `yyval`.

El analizador de léxico:

- Puede ser la función generado con LEX, o
- Una función definida por el usuario en la zona de procedimientos de usuario.

6.8.2 Utilización de YACC

Estructura de un fichero de especificación de YACC

```
{Declaraciones de los símbolos}

%%

{Reglas gramaticales} + {Acciones}

%%

{Procedimientos del usuario}
```

Se pueden incluir comentarios con la siguiente sintaxis:

```
/* ejemplo de comentario */
```


6.8.3 Especificación YACC.

Declaración de Símbolos (1)

En esta zona tenemos:

- **Código en C**, opcional (Ejemplo: declaraciones de variables e inclusión de ficheros).

```
%{  
    /* código C */  
%}
```

- Definición del **símbolo inicial** de la gramática:

```
%start simbolo_inicial
```

- Declaraciones de todos los **nombres de símbolos** (terminales) que utiliza el lenguaje, para ello podemos usar:

- Para todos los nombres de los símbolos (terminales):

```
%token nombre_simbolo1 nombre_simbolo2 ... nombre_simbolo_n
```

Ejemplo:

```
%token IDENTIFICADOR NUMERO  
%token OP_SUMA ASIGNACION
```

Declaración de Símbolos (2)

- Para los nombres de los símbolos que representan operadores:

```
%left OP          /* asociatividad a la izquierda  (X OP Y) OP Z */
%right OP          /* asociatividad a la derecha   X OP (Y OP Z) */
%nonassoc OP       /* sin asociatividad      X OP Y OP Z = error */
%prec SIMBOLO      /* cambio de precedencia. Usado en símbolos
                    que pueden disponer de doble precedencia
                    según el contexto */
```

Reglas Gramaticales

El formato general de una regla gramatical es:

A : **cuerpo** ;

Donde **A** es el nombre de un símbolo no terminal (o estructura sintáctica), y **cuerpo** es una secuencia de nombres de símbolos terminales y/o no terminales y/o literales, que indican la estructura del símbolo no terminal que se define.

Ejemplo:

A	:	B	C	D	;
A	:	E	F		;
A	:	G			;

Cuando un símbolo no terminal dispone de varias alternativas, como el caso del ejemplo anterior, es posible definirlo de la siguiente forma:

Ejemplo:

A	:	B	C	D	;
		E	F		;
		G			;

Tratamiento de la Ambigüedad (1)

Una **gramática** se dice que es **ambigua** cuando alguna de sus sentencias (conjuntos de símbolos) puede ser estructurada de distintas formas.

Si la gramática es ambigua, YACC nos informará en los siguientes términos:

- Conflicto **desplaza/reduce** (**shift/reduce conflict**).
- Conflicto **reduce/reduce** (**reduce/reduce conflict**).

Las posibles soluciones:

- Introducir **asociatividad** y **precedencia** entre los **operadores**.
- Estudiar la **tabla de análisis** en detalle y **modificarla**.

Tratamiento de la Ambigüedad (2)

Ejemplo: Especificación léxica y sintáctica.

<div> <div>Especificación Léxica</div> <div>Función yylex();</div> </div>	<div> <div>Fichero :</div> <div>tabla.h</div> </div>	<div> <div>Especificación Sintáctica</div> <div>Función yyparse</div> </div>	<div> <div>Fichero : error.y</div> </div>
<pre>%{ #include "tabla.h" }% %% "a" return (A); "b" return (B); . {printf("\n Símbolo erroneo :%s\n ", yytext); return (C); } %%</pre>	<pre>#define C 259 #define A 257 #define B 258</pre>	<pre>%token A B %START S %% S : D D ; D : A D B ; %% #include "lexyy.c" #include "error.y" int main () { yyparse (); }</pre>	<pre>void yyerror(char*s) { printf ("Error: %s",s); }</pre>

Tratamiento de la Ambigüedad (3)

Ejemplo: Archivo de salida `y.output` (tabla de análisis).

<div>0 \$accept : S \$end</div> <div>1 S : D D</div> <div>2 D : A D</div> <div>3 B</div>	
<div>state 0</div> <div>\$accept : . S \$end (0)</div> <div>A shift 1</div> <div>B shift 2</div> <div>. error</div> <div>S goto 3</div> <div>D goto 4</div> <div>state 1</div> <div>D : A . D (2)</div> <div>A shift 1</div> <div>B shift 2</div> <div>. error</div> <div>D goto 5</div> <div>state 2</div> <div>D : B . (3)</div> <div>. reduce 3</div> <div>state 3</div> <div>\$accept : S . \$end (0)</div>	<div>state 4</div> <div>S : D . D (1)</div> <div>A shift 1</div> <div>B shift 2</div> <div>. error</div> <div>D goto 6</div> <div>state 5</div> <div>D : A D . (2)</div> <div>. reduce 2</div> <div>state 6</div> <div>S : D D . (1)</div> <div>. reduce 1</div> <div>4 terminals, 3 nonterminals</div> <div>4 grammar rules, 7 states</div>

	A	B	\$	S	D
0	d1	d2	Error r	3	4
1	d1	d2	Error r		5
2	R3	R3	R3		
3			A		
4	d1	d2	Error r		6
5	R2	R2	R2		
6	R1	R1	R1		

Tratamiento de la Ambigüedad (4)

Ejemplo: Especificación YACC de la siguiente gramática.

$$E \rightarrow E + E$$

$$| E - E$$

$$| E * E$$

$$| E / E$$

$$| id$$

El significado de los símbolos en la especificación es la siguiente:

PLUS	+
MINUS	-
MULT	*
DIV	/

```
%token ID
%token PLUS MINUS MULT DIV
```

```
%start Exp
```

```
%%
```

```
Exp : Exp PLUS Exp
    | Exp MINUS Exp
    | Exp MULT Exp
    | Exp DIV Exp
    | ID
```

```
;
```

```
%%
```

Aumento de la gramática y numeración de las producciones realizada por el BYACC (obtenido en el fichero y.out):

```
0 $accept : Exp $end
1 Exp : Exp PLUS Exp
2      | Exp MINUS Exp
3      | Exp MULT Exp
4      | Exp DIV Exp
5      | ID
```

26 feb 02 19:42	y.output	Página 1/4
0	\$accept : Exp \$end	
1	Exp : Exp PLUS Exp	
2	Exp : Exp MINUS Exp	
3	Exp : Exp MULT Exp	
4	Exp : Exp DIV Exp	
5	ID	

26 feb 02 19:42	y.output	Página 2/4
state 0	\$accept : . Exp \$end (0) ID shift 1 . error Exp goto 2	
state 1	Exp : ID . (5) . reduce 5	
state 2	\$accept : Exp . \$end (0) Exp : Exp . PLUS Exp (1) Exp : Exp . MINUS Exp (2) Exp : Exp . MULT Exp (3) Exp : Exp . DIV Exp (4) \$end accept PLUS shift 3 MINUS shift 4 MULT shift 5 DIV shift 6 . error	
state 3	Exp : Exp PLUS . Exp (1) ID shift 1 . error Exp goto 7	
state 4	Exp : Exp MINUS . Exp (2) ID shift 1 . error Exp goto 8	
state 5	Exp : Exp MULT . Exp (3) ID shift 1 . error Exp goto 9	
state 6	Exp : Exp DIV . Exp (4) ID shift 1 . error	

26 feb 02 19:42	y.output	Página 3/4
<pre> Exp goto 10 7: shift/reduce conflict (shift 3, reduce 1) on PLUS 7: shift/reduce conflict (shift 4, reduce 1) on MINUS 7: shift/reduce conflict (shift 5, reduce 1) on MULT 7: shift/reduce conflict (shift 6, reduce 1) on DIV state 7 Exp : Exp . PLUS Exp (1) Exp : Exp PLUS Exp . (1) Exp : Exp . MINUS Exp (2) Exp : Exp . MULT Exp (3) Exp : Exp . DIV Exp (4) PLUS shift 3 MINUS shift 4 MULT shift 5 DIV shift 6 \$end reduce 1 8: shift/reduce conflict (shift 3, reduce 2) on PLUS 8: shift/reduce conflict (shift 4, reduce 2) on MINUS 8: shift/reduce conflict (shift 5, reduce 2) on MULT 8: shift/reduce conflict (shift 6, reduce 2) on DIV state 8 Exp : Exp . PLUS Exp (1) Exp : Exp . MINUS Exp (2) Exp : Exp MINUS Exp . (2) Exp : Exp . MULT Exp (3) Exp : Exp . DIV Exp (4) PLUS shift 3 MINUS shift 4 MULT shift 5 DIV shift 6 \$end reduce 2 9: shift/reduce conflict (shift 3, reduce 3) on PLUS 9: shift/reduce conflict (shift 4, reduce 3) on MINUS 9: shift/reduce conflict (shift 5, reduce 3) on MULT 9: shift/reduce conflict (shift 6, reduce 3) on DIV state 9 Exp : Exp . PLUS Exp (1) Exp : Exp . MINUS Exp (2) Exp : Exp . MULT Exp (3) Exp : Exp MULT Exp . (3) Exp : Exp . DIV Exp (4) PLUS shift 3 MINUS shift 4 MULT shift 5 DIV shift 6 \$end reduce 3 10: shift/reduce conflict (shift 3, reduce 4) on PLUS 10: shift/reduce conflict (shift 4, reduce 4) on MINUS 10: shift/reduce conflict (shift 5, reduce 4) on MULT 10: shift/reduce conflict (shift 6, reduce 4) on DIV </pre>		

26 feb 02 19:42	y.output	Página 4/4
<pre> state 10 Exp : Exp . PLUS Exp (1) Exp : Exp . MINUS Exp (2) Exp : Exp . MULT Exp (3) Exp : Exp . DIV Exp (4) Exp : Exp DIV Exp . (4) PLUS shift 3 MINUS shift 4 MULT shift 5 DIV shift 6 \$end reduce 4 State 7 contains 4 shift/reduce conflicts. State 8 contains 4 shift/reduce conflicts. State 9 contains 4 shift/reduce conflicts. State 10 contains 4 shift/reduce conflicts. 7 terminals, 2 nonterminals 6 grammar rules, 11 states </pre>		

Tratamiento de la Ambigüedad (7)

La tabla de análisis LALR obtenida según esta gramática es la siguiente:

```

1 Exp : Exp PLUS Exp
2     | Exp MINUS Exp
3     | Exp MULT  Exp
4     | Exp DIV   Exp
5     | ID
  
```

Estado	Acción						Goto
	ID	+	-	*	/	\$	Exp
0	D1						2
1	R5	R5	R5	R5	R5	R5	
2		D3	D4	D5	D6	Acc	
3	D1						7
4	D1						8
5	D1						9
6	D1						10
7		D3/R1	D4/R1	D5/R1	D6/R1	R1	
8		D3/R2	D4/R2	D5/R2	D6/R2	R2	
9		D3/R3	D4/R3	D5/R3	D6/R3	R3	
10		D3/R4	D4/R4	D5/R4	D6/R4	R4	

Tratamiento de la Ambigüedad (8)

Ejemplo: Especificación YACC de la gramática anterior dotando a los operadores de asociatividad para evitar los conflictos desplaza/reduce.

- a) + - * / son asociativos por la izquierda.
- b) El orden de precedencia, de mayor a menor, quedaría así: * / + -

```
%token ID
%left PLUS MINUS
%left MULT DIV

%start Exp

%%

Exp : Exp PLUS Exp
    | Exp MINUS Exp
    | Exp MULT Exp
    | Exp DIV Exp
    | ID
;
%%
```

26 feb 02 19:36	y.output	Página 1/2
0	\$accept : Exp \$end	
1	Exp : Exp PLUS Exp	
2	Exp : Exp MINUS Exp	
3	Exp : Exp MULT Exp	
4	Exp : Exp DIV Exp	
5	ID	
state 0	\$accept : . Exp \$end (0)	
	ID shift 1	
	. error	
	Exp goto 2	
state 1	Exp : ID . (5)	
	. reduce 5	
state 2	\$accept : Exp . \$end (0)	
	Exp : Exp . PLUS Exp (1)	
	Exp : Exp . MINUS Exp (2)	
	Exp : Exp . MULT Exp (3)	
	Exp : Exp . DIV Exp (4)	
	\$end accept	
	PLUS shift 3	
	MINUS shift 4	
	MULT shift 5	
	DIV shift 6	
	. error	
state 3	Exp : Exp PLUS . Exp (1)	
	ID shift 1	
	. error	
	Exp goto 7	
state 4	Exp : Exp MINUS . Exp (2)	
	ID shift 1	
	. error	
	Exp goto 8	
state 5	Exp : Exp MULT . Exp (3)	
	ID shift 1	
	. error	

26 feb 02 19:36	y.output	Página 2/2
	Exp goto 9	
state 6	Exp : Exp DIV . Exp (4)	
	ID shift 1	
	. error	
	Exp goto 10	
state 7	Exp : Exp . PLUS Exp (1)	
	Exp : Exp PLUS Exp . (1)	
	Exp : Exp . MINUS Exp (2)	
	Exp : Exp . MULT Exp (3)	
	Exp : Exp . DIV Exp (4)	
	MULT shift 5	
	DIV shift 6	
	\$end reduce 1	
	PLUS reduce 1	
	MINUS reduce 1	
state 8	Exp : Exp . PLUS Exp (1)	
	Exp : Exp . MINUS Exp (2)	
	Exp : Exp MINUS Exp . (2)	
	Exp : Exp . MULT Exp (3)	
	Exp : Exp . DIV Exp (4)	
	MULT shift 5	
	DIV shift 6	
	\$end reduce 2	
	PLUS reduce 2	
	MINUS reduce 2	
state 9	Exp : Exp . PLUS Exp (1)	
	Exp : Exp . MINUS Exp (2)	
	Exp : Exp . MULT Exp (3)	
	Exp : Exp MULT Exp . (3)	
	Exp : Exp . DIV Exp (4)	
	. reduce 3	
state 10	Exp : Exp . PLUS Exp (1)	
	Exp : Exp . MINUS Exp (2)	
	Exp : Exp . MULT Exp (3)	
	Exp : Exp . DIV Exp (4)	
	Exp : Exp DIV Exp . (4)	
	. reduce 4	
7 terminals, 2 nonterminals		
6 grammar rules, 11 states		

Tratamiento de los Errores (1)

1. **Tratamiento por defecto:** En una situación de error se produce una llamada a la función `yyerror` que imprime un mensaje en pantalla `syntax error` y finaliza la ejecución del analizador sintáctico.
2. **Introducción de producciones de error:** En la especificación YACC se pueden introducir producciones de error con el siguiente formato:

$$A \rightarrow \text{error } w \qquad \text{ó} \qquad A \rightarrow \text{error}$$

Donde **A** es un símbolo no terminal de la gramática, **error** una palabra reservada de YACC y **w** es un símbolo terminal.

YACC generará una tabla de análisis a partir de la especificación, dando a las producciones de error el mismo tratamiento que al resto de las producciones.

Tratamiento de los Errores (2)

Actuación del analizador en una situación de error:

Sea S el estado del tope de la pila:

- Si para S existe un desplazamiento con el símbolo **error** lo aplicará.
- Si para S no existe un desplazamiento con el símbolo **error**, extrae símbolos de la pila hasta que encuentra un estado que incluya un *ítem* de la forma $A \rightarrow \cdot \text{error } w$, introduce **error** en la pila.
- Si w es distinto de la cadena vacía, eliminará caracteres de la entrada hasta encontrar w , entonces pasará a reducir por la producción de **error**.
- Si w es la cadena vacía pasa a reducir por la producción de **error** cuando acontezca algún seguidor de A .

Temas: Análisis Sintáctico Ascendente LR

Símbolos terminales $\{a, b, c, d, e, f\}$ Símbolos no terminales $\{P, Q, R, S\}$

Símbolo inicial

Ejemplo: Especificación YACC de la gramática anterior dotando a los operadores de asociatividad para evitar los conflictos desplaza/reduce.

 $\{P\}$

Producciones:

$$P \rightarrow a Q R f$$

$$Q \rightarrow c$$

$$R \rightarrow b S e$$

$$S \rightarrow d f \mid d f S$$

```
%token  A B C D E F
%Start  P

%%

P  : A Q R F ;
Q  : C ;
Q  : error { printf("Recupera error con la
                  producción de error 1 \n");
          yyerrok; } ;
R  : B S E ;

R  : error E { printf("Producción de error 2\n");
          yyerrok; } ;
S  : D F
    | D F S ;

S  : error F {printf("Produccion de error 3\n");
          yyerrok; } ;

%%

#include "lexyy.c"
#include "error.y"

main ()
{
    yyparse();
}
```

```

0  $accept : P $end
1  P : a Q R f
2  Q : c
3    | error
4  R : b S e
5    | error e
6  S : d f
7    | d f S
      | error f

```

Estado	Acción								Goto			
	a	B	c	d	e	f	Error	\$	P	Q	S	R
0	D1								2			
1			D4				D3			5		
2								A				
3	R3	R3	R3	R3	R3	R3	R3	R3				
4	R2	R2	R2	R2	R2	R2	R2	R2				
5		D7					D6					8
6					D9							
7				D11			D10				12	
8						D13						
9		R5	R5	R5	R5	R5	R5	R5				
10						D14						
11						D15						
12					D16							
13	R1	R1	R1	R1	R1	R1	R1	R1				
14	R8	R8	R8	R8	R8	R8	R8	R8				
15				D11	R6		D10				17	
16	R4	R4	R4	R4	R4	R4	R4	R4				
17	R7	R7	R7	R7	R7	R7	R7	R7				

Pila	Entrada	Acción
0	acbfdddfe\$	Desplaza <i>a</i>
0a1	cbfdddfe\$	Desplaza <i>c</i>
0a1c5	bfdddfe\$	Reduce ($Q \rightarrow c$)
0a1Q5	bfdddfe\$	Desplaza <i>b</i>
0a1Q5b7	fdddfe\$	Desplaza <i>d</i>
0a1Q5b7d11	fddfe\$	Desplaza <i>f</i>
0a1Q5b7d11f15	ddfe\$	Desplaza <i>d</i>
0a1Q5b7d11f15d11	ddfe\$	ERROR, 11 no tiene desplazamiento con error, Busca en la pila un estado que si lo tenga.
0a1Q5b7d11f15	ddfe\$	Se sitúa en el estado 15 y desplaza error ($S \rightarrow \text{error } f$)
0a1Q5b7d11f15 error 10	ddfe\$	Descarta los símbolos hasta encontrar <i>f</i> y la desplaza
0a1Q5b7d11f15 error 10f14	ef\$	Reduce por $S \rightarrow \text{error } f$
0a1Q5b7d11f15S17	ef\$	Reduce por $S \rightarrow dfS$
0a1Q5b7S12	ef\$	Desplaza <i>e</i>
0a1Q5b7S12e16	f\$	Reduce por $R \rightarrow bSe$
0a1Q5R8	f\$	Desplaza <i>f</i>
0a1Q5R8f13	\$	Reduce por $P \rightarrow aQRf$
0P2	\$	Acepta

0 \$accept : P
\$end
1 P : a Q R f
2 Q : c
3 | error
4 R : b S e
5 | error e
6 S : d f
7 | d f S
| error f

Estado	Acción								Goto			
	a	B	c	d	e	f	Error	\$	P	Q	S	R
0	D1								2			
1			D4				D3			5		
2								A				
3	R3	R3	R3	R3	R3	R3	R3	R3				
4	R2	R2	R2	R2	R2	R2	R2	R2				
5		D7					D6					8
6					D9							
7				D11			D10				12	
8						D13						
9		R5	R5	R5	R5	R5	R5	R5				
10						D14						
11						D15						
12					D16							
13	R1	R1	R1	R1	R1	R1	R1	R1				
14	R8	R8	R8	R8	R8	R8	R8	R8				
15				D11	R6		D10				17	
16	R4	R4	R4	R4	R4	R4	R4	R4				
17	R7	R7	R7	R7	R7	R7	R7	R7				

Pila	Entrada	Acción
0	acbfdddfe\$	Desplaza <i>a</i>
0a1	cbfdddfe\$	Desplaza <i>c</i>
0a1c5	bfdddfe\$	Reduce (<i>Q</i> → <i>c</i>)
0a1Q5	bfdddfe\$	Desplaza <i>b</i>
0a1Q5b7	dfddfe\$	Desplaza <i>d</i>
0a1Q5b7d11	fddfe\$	Desplaza <i>f</i>
0a1Q5b7d11f15	ddfe\$	Desplaza <i>d</i>
0a1Q5b7d11f15d11	ddfe\$	ERROR, 11 no tiene desplazamiento con error, Busca en la pila un estado que si lo tenga.
0a1Q5b7d11f15	ddfe\$	Se sitúa en el estado 15 y desplaza error (<i>S</i> → error <i>f</i>)
0a1Q5b7d11f15 error 10	ddfe\$	Descarta los símbolos hasta encontrar <i>f</i> y la desplaza
0a1Q5b7d11f15 error 10f14	ef\$	Reduce por <i>S</i> → error <i>f</i>
0a1Q5b7d11f15S17	ef\$	Reduce por <i>S</i> → <i>dfS</i>
0a1Q5b7S12	ef\$	Desplaza <i>e</i>
0a1Q5b7S12e16	f\$	Reduce por <i>R</i> → <i>bSe</i>
0a1Q5R8	f\$	Desplaza <i>f</i>
0a1Q5R8f13	\$	Reduce por <i>P</i> → <i>aQRf</i>
0P2	\$	Acepta

Ejemplo: Dada la gramática siguiente, estudiar los casos de error mediante producción de **error** sin usar ningún símbolo terminal para la sincronización (lo hará con algún seguidor del no terminal).

$$P \rightarrow a Q R f$$

$$Q \rightarrow c$$

$$R \rightarrow b S e$$

$$S \rightarrow d f$$

$$| d f S$$

Una cadena válida para este lenguaje puede ser:

a c b d f d f d f f

$$P \rightarrow a Q R$$

$$f$$

$$Q \rightarrow c$$

$$| \text{error}$$

$$R \rightarrow b S e$$

$$| \text{error}$$

$$S \rightarrow d f$$

$$| d f S$$

$$| \text{error}$$

Archivo: lex-ej7.l	Archivo: yacc-ej7.y
<pre>%% [\t]* ECHO; \n ECHO; "a" { ECHO ; return aa ; } "f" { ECHO ; return ff ; } "c" { ECHO ; return cc ; } "b" { ECHO ; return bb ; } "e" { ECHO ; return ee ; } "d" { ECHO ; return dd ; } . printf ("\n(Linea %d) Error léxico leyendo %s\n", yylineno, yytext); %%</pre>	<pre>%token aa bb cc dd ee ff %start P %% P : aa Q R ff ; Q : cc error ; R : bb S ee error ; S : dd ff dd ff S error ; #include "lexyy.c" #include "msj.err" void yyerror (char *msg) { printf ("\n (Linea %d) %s y aparece %s\n", yylineno, msg, yytext); } /*****/ int main () { yyparse(); return (0); }</pre>

Detección y recuperación de errores:

- $Q \rightarrow cc$: Si no aparece una **c** entonces se recuperaría con $[Q \rightarrow \text{error}]$ y sincronizaría hasta alcanzar un seguidor de Q .

$$\text{Seguidores}(Q) = \{ b, \text{error} \}$$

- $R \rightarrow bb \ S \ ee$: Si no aparece cualquiera de estos símbolos, se recuperaría con $[R \rightarrow \text{error}]$ y sincronizaría hasta alcanzar un seguidor de R

$$\text{Seguidores}(R) = \{ f \}$$

- $S \rightarrow dd \ ff \mid dd \ ff \ S$: Si no aparece un par **d f** o secuencias de éstos, se recuperaría con $[S \rightarrow \text{error}]$ y sincronizaría hasta alcanzar un seguidor de S .

$$\text{Seguidores}(S) = \{ e \}$$

Podemos interpretar que los tokens **aa** y **ff** se podrían corresponder con marcas de comienzo y de final de la sentencia que se puede expresar con este lenguaje.

Ejecución 1. Debería detectar el error y seguir analizando, sin embargo se detiene leyendo el primer token.

```
a c c f
a c c
(Linea 1) Error () esperaba bb y aparece c
f
```

$$P \rightarrow a Q R$$

f

$$Q \rightarrow c$$

| *error*

$$R \rightarrow b S e$$

| *error*

$$S \rightarrow d f$$

| d f S

| *error*

Ejecución 2. Detecta el error y sigue analizando.

```
c c c f
c
(Linea 1) Error (0) esperaba aa y aparece c
```

Depurador de YACC: Es posible realizar depuraciones en una ejecución de análisis en YACC. Para ello es necesario fijar la variable de entorno `YYDEBUG=1`.

Texto de entrada:

a c b d f d d f e f

```
a yydebug: state 0, reading 257 (aa)
yydebug: state 0, shifting to state 1
c yydebug: state 1, reading 259 (cc)
yydebug: state 1, shifting to state 4
yydebug: state 4, reducing by rule 2 (Q : cc)
yydebug: after reduction, shifting from state 1 to state 5
b yydebug: state 5, reading 258 (bb)
yydebug: state 5, shifting to state 7
d yydebug: state 7, reading 260 (dd)
yydebug: state 7, shifting to state 10
f yydebug: state 10, reading 262 (ff)
yydebug: state 10, shifting to state 13
d yydebug: state 13, reading 260 (dd)
yydebug: state 13, shifting to state 10
d yydebug: state 10, reading 260 (dd)
```

Error (10) esperaba ff y aparece d

```
yydebug: error recovery discarding state 10
yydebug: state 13, error recovery shifting to state 9
yydebug: state 9, reducing by rule 8 (S : error)
yydebug: after reduction, shifting from state 13 to state 15
yydebug: state 15, reducing by rule 7 (S : dd ff S)
yydebug: after reduction, shifting from state 7 to state 11
yydebug: state 11, error recovery discards token 260 (dd)
f yydebug: state 11, reading 262 (ff)
yydebug: state 11, error recovery discards token 262 (ff)
e yydebug: state 11, reading 261 (ee)
yydebug: state 11, shifting to state 14
yydebug: state 14, reducing by rule 4 (R : bb S ee)
yydebug: after reduction, shifting from state 5 to state 8
f yydebug: state 8, reading 262 (ff)
yydebug: state 8, shifting to state 12
yydebug: state 12, reducing by rule 1 (P : aa Q R ff)
yydebug: after reduction, shifting from state 0 to state 2

yydebug: state 2, reading 0 (end-of-file)
```

0 \$accept : P \$end	error shift 6	state 13
1 P : aa Q R ff	bb shift 7	S : dd ff . (6)
3 error	. error	S : dd ff . S (7)
2 Q : cc	R goto 8	error shift 9
5 error	state 6	dd shift 10
6 S : dd ff	R : error . (5)	ee reduce 6
7 dd ff S	. reduce 5	S goto 15
8 error	state 7	state 14
state 0	R : bb . S ee (4)	R : bb S ee . (4)
\$accept : . P \$end	error shift 9	. reduce 4
(0)	dd shift 10	state 15
aa shift 1	. error	S : dd ff S . (7)
. error	S goto 11	. reduce 7
P goto 2	state 8	8 terminals, 5 nonterminals
state 1	P : aa Q R . ff (1)	9 grammar rules, 16 states
P : aa . Q R ff (1)	ff shift 12	
error shift 3	. error	
cc shift 4	state 9	
. error	S : error . (8)	
Q goto 5	. reduce 8	
state 2	state 10	
\$accept : P . \$end	S : dd . ff (6)	
(0)	S : dd . ff S (7)	
\$end accept	ff shift 13	
state 3	. error	
Q : error . (3)	state 11	
. reduce 3	R : bb S . ee (4)	
state 4	ee shift 14	
Q : cc . (2)	. error	
. reduce 2	state 12	
state 5	P : aa Q R ff . (1)	
P : aa Q . R ff (1)	. reduce 1	