# Procesadores de Lenguajes I:

# Una introducción a la fase de análisis

*J. M. Benedí Ruíz*
*V. Gisbert Giner*
*L. Moreno Boronat*
*E. Vivancos Rubio*

# Capítulo 4

# Análisis sintáctico ascendente

En este capítulo se muestran las técnicas de construcción de un analizador sintáctico ascendente a partir de la especificación sintáctica. Para ello, tras mostrar los conceptos teóricos básicos empleados en las técnicas de análisis sintáctico ascendente, se describe paso a paso el proceso de construcción de un analizador sintáctico de tipo LR(0), SLR(1), LR(1) y LALR(1). También se incluye un apartado dedicado a resolver los conflictos que pueden aparecer al construir un analizador sintáctico ascendente para una gramática ambigua. El capítulo finaliza con una breve descripción de la relación entre los tipos de gramáticas definidas a lo largo de los capítulos de análisis sintáctico descendente y ascendente.

## 4.1 Conceptos fundamentales

### *4.1.1. Análisis sintáctico ascendente*

En el capítulo anterior se ha visto como en el análisis sintáctico descendente se realiza un proceso de derivación que, a partir del símbolo inicial de la gramática, permite obtener la cadena analizada. En cambio, en el análisis sintáctico descendente, veremos que se realizará un proceso que, a partir de la cadena a analizar, permite obtener el símbolo inicial de la gramática. Este proceso se llevará a cabo aplicando la operación inversa de la derivación, a la que llamaremos reducción.

Dada una forma sentencial $\delta\ \beta\ \gamma$, aplicar la ***reducción*** correspondiente a la regla de producción $A \rightarrow \beta$, consiste en sustituir en la forma sentencial el lado derecho de la producción, por su no-terminal del lado izquierdo:

$$\delta\ \beta\ \gamma \leftarrow \delta\ A\ \gamma \qquad \underline{\text{sii}} \quad \exists\ (A \rightarrow \beta) \in P; \qquad\qquad \delta, \gamma \in (N \cup \Sigma)^*$$

Dada una gramática $G=(N, \Sigma, P, S)$ independiente del contexto y una cadena $\omega \in \Sigma^*$ podemos ver el análisis sintáctico ascendente (ASA) de $\omega$ como el proceso de construir el árbol de análisis

sintáctico asociado a la cadena desde las hojas hasta la raíz. Esto es equivalente a aplicar sobre ω una secuencia de reducciones que permita llegar al símbolo inicial de la gramática: S. El orden de aplicación de las reducciones coincide con la inversa de la derivación a derechas para la cadena.

*Ejemplo 4.1.*- Sea la gramática $G_1$:

$S \rightarrow aABe$ (1)

$A \rightarrow Abc$ (2)

$\mid b$ (3)

$B \rightarrow b$ (4)

La traza del análisis sintáctico ascendente para la cadena ω = abbcbe según la gramática $G_1$ será

$$abbcbe \leftarrow aAbcbe \leftarrow aAbe \leftarrow aABe \leftarrow S$$

A partir de la cadena de entrada, y mediante la secuencia de reducciones 3,2,4,1 se ha llegado al símbolo inicial de la gramática.

La realización de un análisis sintáctico ascendente implica decidir para cada frase sentencial a derechas qué producción aplicar. Esto presenta dos tipos de problema:

a) Dos producciones tienen la misma parte derecha. Esto ocurre en el ejemplo anterior con las producciones $A \rightarrow b$ y $B \rightarrow b$.

b) Aparentemente se pueden aplicar reducciones sobre más de una subcadena de una forma sentencial. Esto aparece en el ejemplo anterior cuando sobre la forma sentencial aAbcbe se podrían aplicar aparentemente las reducciones $A \rightarrow b$ y $A \rightarrow Abc$.

En este capítulo se presentan diversas técnicas que permiten al analizador tomar la decisión adecuada cuando se enfrente a alguno de los dos problemas anteriores, obteniéndose así analizadores sintácticos ascendentes deterministas.

### *4.1.2 ASA por desplazamiento-reducción*

El análisis ascendente mediante reducciones por la izquierda puede implementarse mediante el empleo de una pila. La idea básica consiste en desplazar símbolo a símbolo la cadena de entrada a la pila. Antes de realizar el desplazamiento de un símbolo de la cadena de entrada a la pila, se observa si "se debe" aplicar una reducción sobre los símbolos de la cima de la pila. Si esto es así, se aplica la reducción, sino se realiza el desplazamiento. Este algoritmo general para realizar el análisis sintáctico ascendente se llama algoritmo de ***desplazamiento-reducción***.

En el algoritmo 4.1 se muestra una versión de este algoritmo, que emplea vuelta atrás (backtracking) para solucionar el problema del indeterminismo. En [Aho 72] se presenta formalmente este algoritmo para un autómata de pila.

En el Ejemplo 4.2 podemos observar que la aplicación de la reducción r-2 correspondiente a la regla número 2 en (EE, a$, 2) no ha permitido llegar al símbolo inicial, y ha sido necesario realizar una vuelta atrás para deshacer la reducción. En cambio la aplicación de esa misma reducción en (E, aba$, ) si que permite llegar al símbolo inicial. El objetivo de los siguientes apartados es encontrar algún método que permita diferenciar cuándo se debe aplicar una reducción y cuándo se debe desplazar, para poder tener de esta manera un analizador sintáctico ascendente (A.S.A.) predictivo.

*Ejemplo 4.2.*- Sea la gramática $G_2$

$$S \rightarrow EF \quad (1)$$
$$E \rightarrow ab \quad (2)$$
$$F \rightarrow aba \quad (3)$$

La traza del A.S.A. mediante el algoritmo de desplazamiento-reducción para la cadena $\omega$ = ababa sería:

| **PILA** | **ENTRADA** | |
|---|---|---|
| | ababa$ | |
| a | baba$ | |
| ab | aba$ | |
| E | aba$ | r-2 |
| Ea | ba$ | |
| Eab | a$ | |
| EE | a$ | r-2 |
| EEa | $ | vuelta atrás |
| Eab | a$ | |
| Eaba | $ | r-3 |
| EF | $ | r-1 |
| S | $ | Aceptación |

```
ALGORITMO ASA desplazamiento-reducción no predictivo
ENTRADA     ω ∈ Σ* y G=(N, Σ, P, S)
SALIDA      Si ω ∈ L(G) entonces aceptación sino error
USA         Pila de (N ∪ Σ)*

MÉTODO
Repetir

      Repetir
            Si en la cima de la pila está la parte derecha de una producción
                  entonces    aplicar la reducción
                  sino Desplazar un símbolo de la cadena de entrada a la cima
                        de la pila.
      Hasta no queden símbolos en la cadena de entrada

      Si en la pila está solo el símbolo inicial entonces fin(aceptación)
      Sino VUELTA ATRÁS hasta deshacer la última reducción aplicada

Hasta Probadas todas las posibilidades

fin(error)
```

**Algoritmo 4.1**: Algoritmo desplazamiento-reducción no predictivo

### *4.1.3. Pivote*

Dada una gramática incontextual G = (N, Σ, P, S), y dada una forma sentencial a derechas $\gamma \in (N \cup \Sigma)^*$. Un ***pivote*** de $\gamma$ es: (r, j), donde $r \in P$ y $j \geq 0$, con $A \in N$; $\alpha, \beta \in (N \cup \Sigma)^*$; $\omega \in \Sigma^*$, si cumple:

$$S \rightarrow^* \alpha A \omega \rightarrow \alpha \beta \omega = \gamma$$

$$\text{con} \quad r = (A \rightarrow \beta) \quad \text{y} \quad j = |\alpha \beta|$$

Como se puede apreciar, el pivote de una forma sentencial a derechas es básicamente el lado derecho de una producción que, reducido, permite obtener la siguiente forma sentencial a derechas del proceso que acabará obteniendo el símbolo inicial de la gramática.

Si somos capaces de detectar de forma determinista los pivotes, podremos realizar de forma predictiva el ASA: Basta con ir desplazando símbolos a la pila del analizador hasta que detectemos en la cima de la pila un pivote y en ese momento aplicar la reducción asociada al pivote. El problema ahora consiste en detectar los pivotes en la cima de la pila.

Notar que en una forma sentencial a derechas, a la derecha del pivote solo puede haber símbolos terminales. Además, se cumple que si una gramática no es ambigua, cada forma sentencial tendrá un único pivote.

*Ejemplo 4.3.*- A.S.A. con gramática ambigua

Sea la gramática $G_3$:

$E \rightarrow E + E$
$\quad | \; E * E$
$\quad | \; ( E )$
$\quad | \; id$

La gramática $G_3$ es ambigua, porque podemos encontrar dos derivaciones a derechas distintas para la cadena $\omega$ = id + id * id

$$E \rightarrow \underline{E + E} \rightarrow E + \underline{E * E} \rightarrow E + E * \underline{id} \rightarrow E + \underline{id} * id \rightarrow \underline{id} + id * id$$

$$E \rightarrow \underline{E * E} \rightarrow E * \underline{id} \rightarrow \underline{E + E} * id \rightarrow E + \underline{id} * id \rightarrow \underline{id} + id * id$$

Esto se traduce en que la forma sentencial E + E * id tiene dos pivotes: ( $E \rightarrow$ id , 5 ) y ( $E \rightarrow E + E$ , 3 ), ya que cualquiera de estas dos reducciones aplicadas sobre la misma forma sentencial a derechas (E + E * id), permite obtener la siguiente forma sentencial a derechas del proceso de reducción que nos lleva al símbolo incial de la gramática.

### *4.1.4. Gramáticas LR(k)*

Se dice que una gramática G= (N, Σ, P, S) es ***reducida*** si el axioma S (símbolo inicial de la gramática) no aparece en la parte derecha de ninguna regla de G. En este capítulo, salvo que se indique lo contrario, se utilizan siempre gramáticas reducidas. Para garantizar que toda gramática con la que se trabaje es reducida, basta con ampliarla con un nuevo símbolo inicial y una nueva producción para éste símbolo, obteniendo una nueva gramática G'= (N', Σ, P', S'), donde N'=N $\cup$ { S' }, P' = P $\cup$ {(S'$\rightarrow$S$)} y S' $\notin$ N $\cup$ Σ. Al añadir la producción S'$\rightarrow$S$ además, se garantiza que todas las sentencias terminarán con el símbolo $, que pasa a representar de esta forma el fin de cadena de entrada.

Hay un tipo de gramáticas independientes del contexto conocidas como gramáticas LR para las que es posible realizar el análisis sintáctico ascendente de forma determinista. Si conociendo k símbolos de anticipación (además de los símbolos analizados) es posible reconocer de forma determinista los pivotes de cualquier forma sentencial, diremos que la gramática es LR(k)[1].

[1] La L hace referencia a que la cadena de entrada se lee de izquierda a derecha ("Left-to-right"). La R indica que se realiza una derivación a derechas ("Rightmost-derivation") propia del análisis ascendente.

Más formalmente podemos decir que, dado un $k \geq 0$, y una gramática incontextual G=(N,Σ,P,S) reducida G es ***LR(k)*** si para $\gamma, \alpha, \alpha', \beta, \beta' \in (N \cup \Sigma)^*$; $\omega, \omega' \in \Sigma^*$; $A, A' \in N$ se tiene que,

si se cumple:

i) $S \rightarrow^* \alpha A \omega \rightarrow \alpha \beta \omega = \gamma \omega$

ii) $S \rightarrow^* \alpha' A' \omega' \rightarrow \alpha' \beta' \omega' = \gamma \omega'$

iii) Los pivotes $(A \rightarrow \beta, |\alpha \beta|)$ y $(A' \rightarrow \beta', |\alpha'\beta'|)$ son distintos

entonces $primeros_k(\omega) \neq primeros_k(\omega')$

En general se dice que una gramática G es LR si y solo si $\exists k \mid$ G es LR(k).

### *4.1.5. Modelo de analizador LR*

Para realizar el análisis ascendente que pretendemos, emplearemos un analizador LR. Un analizador LR será un autómata de pila como el mostrado en la Fig. 4.1.

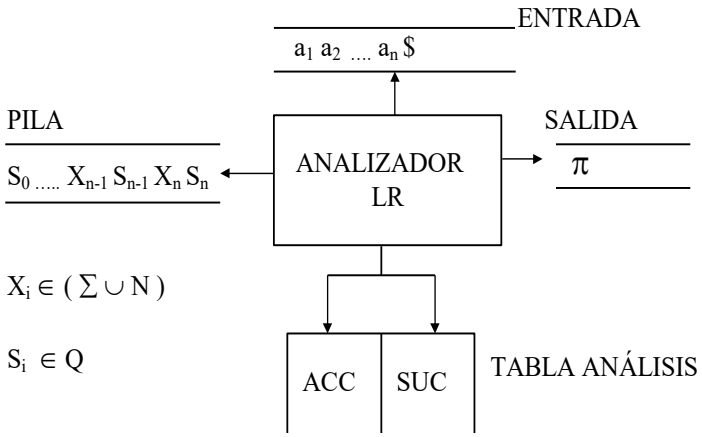



**Figura 4.1.** Esquema de un analizador LR

Este autómata dispone de:

- Una cinta de entrada, donde estará la cadena a analizar.
- Una cinta de salida.
- Una función de transición representada por una tabla de análisis.
- Una pila donde se apilarán símbolos de la gramática y estados por los que ha pasado el autómata.

La configuración del autómata viene dada por ($S_0$ $X_1$ $S_1$ …. $X_n$ $S_n$ , $a_1a_2...a_n$ \$, π ), y la configuración inicial es ($S_0$ , ω \$, ε ).

Al igual que ocurre en los analizadores LL, la función de transición de un reconocedor LR viene dada por una tabla de análisis. Esta tabla de análisis está dividida en dos subtablas:

- Una subtabla ***acción*** empleada para conocer la acción (desplazamiento o reducción) que debe realizar el autómata. En el caso de un analizador LR(1) está subtabla se define

$$acc: Q \times (\Sigma \cup \{\$\}) \rightarrow \{des, red\text{-}r, err, acep\}$$

donde *des* indica que la acción a realizar es un desplazamiento de un símbolo de la cadena de entrada a la pila, *red-r* que se debe aplicar la reducción correspondiente a la producción número *r*, *err* que se ha producido un error (la cadena analizada no pertenece al lenguaje reconocido por el autómata), y *acep* indica que ha finalizado el análisis reconociendo la cadena.

- Una subtabla sucesor[2], donde se indica el estado al que debe pasar el autómata tras realizar cada acción de desplazamiento o reducción.

$$suc: Q \times (\Sigma \cup N) \rightarrow Q \cup \{err\}$$

Los cuatro tipos de acciones de la subtabla acción aplicadas sobre un autómata reconocedor LR(1) producen los siguientes movimientos:

a) *desplazamiento*:

$$(S_0\ X_1\ S_1\ \ldots.\ X_n\ S_n\ ,\ a_i...a_n\ \$,\ \pi\ )\ \vdash\ (S_0\ X_1\ S_1\ \ldots.\ X_n\ S_n\ a_i\ S_{n+1},\ a_{i+1}...a_n\ \$,\ \pi\ )$$

si $acc[S_n,a_i] = des$ y $suc[S_n,a_i] = S_{n+1}$

Mediante esta acción se desplaza a la pila el símbolo de la cadena de entrada ($a_i$) y, tras él, se apila el estado al que pasa el autómata ($S_{n+1}$).

b) *reducción*:

$$(S_0\ X_1\ S_1\ \ldots.\ X_n\ S_n\ ,\ a_i...a_n\ \$,\ \pi\ )\ \vdash\ (S_0\ X_1\ S_1\ \ldots.\ X_k\ S_k\ A\ S_{n+1},\ a_i...a_n\ \$,\ \pi{\cdot}j\ )$$

si $acc[S_n,a_i] = red\text{-}j$ y $suc[S_k,A] = S_{n+1}$ siendo la producción j: $A \rightarrow X_{k+1}\ X_{k+2}\ ...\ X_n$

[2] La subtabla sucesor está parcialmente definida puesto que se puede demostrar que hay situaciones estado-terminal que no se pueden presentar nunca.

Mediante esta acción, se desapila el lado derecho de una producción y los estados que siguen a cada símbolo ($X_{k+1}$ $S_{k+1}$ $X_{k+2}$ $S_{k+2}$... $X_n$ $S_n$) y apilamos el no-terminal del lado izquierdo de la producción (A).

c) *aceptación*: La configuración final de autómata será $(S_0X_n S_n , \$, \pi )$ si $acc[S_n,\$]$= acept

d) *error*: La cadena de entrada no pertenece al lenguaje reconocido por el autómata.

A modo de ejemplo en la Fig. 4.2 se muestra una tabla de análisis ascendente SLR(1) para la gramática $G_4$. En las próximas secciones veremos como construir este tipo de tablas.

$G_4$:

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

En un análisis ascendente con un símbolo de anticipación, frecuentemente las subtablas Acción y Sucesor se representan fusionadas, apareciendo junto a cada operación de desplazamiento, el estado al que debe pasar el autómata. De esta forma no es necesario que aparezcan los símbolos terminales en la subtabla sucesor. No obstante, sigue siendo necesario que aparezcan los símbolos no-terminales. La tabla resultante de esta fusión se denomina ***tabla compactada***. En la Fig. 4.3 se muestra la tabla de análisis de la Fig. 4.2 compactada.

***Prefijo viable***

A cada uno de los prefijos de formas sentenciales a derechas que pueden aparecer en la pila durante el análisis por desplazamiento-reducción, se le llama ***prefijo viable***.

Más formalmente podemos definir un prefijo viable para una forma sentencial a derechas $\alpha \beta \omega$ con $\alpha, \beta \in (N \cup \Sigma)^*$; $\omega \in \Sigma^*$, siendo su pivote asociado $(A \rightarrow \beta, |\alpha \beta|)$ y $\alpha \beta = u_1 ... u_n$, como cualquier cadena $u_1 ... u_i$ con $0 \leq i \leq n$. De esta definición se deduce que un pivote contiene cualquier prefijo de una forma sentencial a derechas sin sobrepasar al pivote.

| | ***ACC*** | | | | | | ***SUCC*** | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | id | + | * | ( | ) | $ | id | + | * | ( | ) | E | T | F |
| 0 | d | | | d | | | 5 | | | 4 | | 1 | 2 | 3 |
| 1 | | d | | | | Ac | | 6 | | | | | | |
| 2 | | r-2 | d | | r-2 | r-2 | | | 7 | | | | | |
| 3 | | r-4 | r-4 | | r-4 | r-4 | | | | | | | | |
| 4 | d | | | d | | | 5 | | | 4 | | 8 | 2 | 3 |
| 5 | | r-6 | r-6 | | r-6 | r-6 | | | | | | | | |
| 6 | d | | | d | | | 5 | | | 4 | | | 9 | 3 |
| 7 | d | | | d | | | 5 | | | 4 | | | | 10 |
| 8 | | d | | | d | | | 6 | | | 11 | | | |
| 9 | | r-1 | d | | r-1 | r-1 | | | 7 | | | | | |
| 10 | | r-3 | r-3 | | r-3 | r-3 | | | | | | | | |
| 11 | | r-5 | r-5 | | r-5 | r-5 | | | | | | | | |

**Figura 4.2.** Tablas acción y sucesor sin compactar

| | ***ACC*** | | | | | | ***SUCC*** | | |
|---|---|---|---|---|---|---|---|---|---|
| | id | + | * | ( | ) | $ | E | T | F |
| 0 | d-5 | | | d-4 | | | 1 | 2 | 3 |
| 1 | | d-6 | | | | Ac | | | |
| 2 | | r-2 | d-7 | | r-2 | r-2 | | | |
| 3 | | r-4 | r-4 | | r-4 | r-4 | | | |
| 4 | d-5 | | | d-4 | | | 8 | 2 | 3 |
| 5 | | r-6 | r-6 | | r-6 | r-6 | | | |
| 6 | d-5 | | | d-4 | | | | 9 | 3 |
| 7 | d-5 | | | d-4 | | | | | 10 |
| 8 | | d-6 | | | d-11 | | | | |
| 9 | | r-1 | d-7 | | r-1 | r-1 | | | |
| 10 | | r-3 | r-3 | | r-3 | r-3 | | | |
| 11 | | r-5 | r-5 | | r-5 | r-5 | | | |

**Figura 4.3.** Tablas acción y sucesor compactadas

Salvo que se indique lo contrario siempre construiremos y utilizaremos tablas compactadas.

***Teorema de Knuth***

*El conjunto de todos los prefijos viables de cualquier forma sentencial a derechas de una gramática LR(k) puede ser reconocido por un Autómata de Estados Finitos.*

A partir del teorema de Knuth, nuestro objetivo se transforma en construir un autómata que reconozca todos los prefijos viables de una gramática, y además poder determinar para cada uno de ellos si cuando aparezcan en la pila durante el análisis se debe realizar una reducción o un desplazamiento. Si conseguimos esto, es decir, poder reconocer durante el análisis cuándo aparece un pivote en la cima de la pila, tendremos un método para realizar el análisis sintáctico ascendente de forma predictiva.

En el próximo apartado nos ocuparemos de encontrar un método que nos permita construir este autómata reconocedor de prefijos viables y así reconocer los pivotes cuando aparezcan en la cima de la pila.

### *ASA desplazamiento-reducción predictivo*

El algoritmo de análisis por desplazamiento-reducción visto al principio del capítulo, puede adaptarse para realizar el análisis de forma predictiva. Este algoritmo para el caso de un analizador que utilice un solo símbolo de anticipación y partiendo de las tablas de acción y sucesor compactadas queda tal y como se muestra en el algoritmo 4.2.

```
ALGORITMO  A.S.A: Desplazamiento-Reducción
/* Con las tablas acción y sucesor compactadas */
ENTRADA   G' = (N', Σ, P', S');     ω ∈ Σ*;     TA (acc,suc)
acc: Q x (Σ ∪ {$}) → {des-s, red-r, err, acep}
suc: Q x N → Q ∪{err}
SALIDA    Si ω ∈ L(G) entonces π else error( )
METODO
apilar(s0); sim := obtsym;  π := ε;   fin := falso;
repetir
 caso acc[cima,sim] sea
   "des-s":  apilar(sim); apilar(s); sim := obtsym;
   "red-r: A→β": para i:=1 hasta 2*|β|  hacer desapilar;
                 si   suc[cima,A] = err   entonces error( )
                 sino  s':= suc[cima,A]; apilar(A); apilar(s');  π = π · r;
   "acep":  fin := verdad;
   "err":    error( );
 end;
hasta fin
FIN
```

**Algoritmo 4.2.** Algoritmo de análisis LR por desplazamiento-reducción

Como se ve en el algoritmo 4.2, las acciones que se pueden aplicar son presentadas anteriormente para un algoritmo de desplazamiento-reducción.

*Ejemplo 4.4.*- En este ejemplo se va a realizar la traza de análisis ascendente por desplazamiento-reducción para la cadena ω = id * id + id según la tabla de la Fig. 4.3 y el algoritmo 4.2 de análisis sintáctico ascendente por desplazamiento-reducción:

La configuración inicial es: (0, id*id+id$,  )

Consultando la subtabla acción (fila 0, columna “id”) se obtiene que la acción a aplicar es un desplazamiento al estado 5 (d-5). Por lo tanto, se desplaza el “id” de la cadena de entrada a la pila, y se apila el estado 5 al que pasa el autómata:  (0id5, *id+id$,  )

Ahora debe consultarse la fila 5 y la columna “*”, obteniendo la acción “r6”. Aplicar la reducción número 6 (F → id) consiste en desapilar “id” y el estado que le sigue (5) y en su lugar apilar “F”. Seguidamente debe apilarse el estado al que pasa el autómata tras realizar la reducción. Para ello se debe consultar la subtabla sucesor, columna “F” (no-terminal apilado), fila “0” (estado que aparece bajo el no-terminal “F” recién apilado). Por lo tanto se apila el estado “3” indicado por la celda:  (0F3, *id+id$, 6)

El resto del proceso quedaría:

(0F3, *id+id$, 6)
├— (0T2, *id+id$, 6 4)
├— (0T2*7, id+id$, 6 4)
├— (0T2*7id5, +id$, 6 4)
├— (0T2*7F10, +id$, 6 4 6)
├— (0T2, +id$, 6 4 6 3)
├— (0E1, +id$, 6 4 6 3 2)
├— (0E1+6, id$, 6 4 6 3 2)
├— (0E1+6id5, $, 6 4 6 3 2)
├— (0E1+6F3, $, 6 4 6 3 2 6)
├— (0E1+6T9, $, 6 4 6 3 2 6 4)
├— (0E1, $, 6 4 6 3 2 6 4 1)
├— Aceptada

## 4.2. Análisis LR(0)

En este apartado se muestra la forma más simple de análisis sintáctico ascendente determinista: el análisis LR(0). Para obtener las tablas de análisis (acción y sucesor) que determinen la función de transición del analizador sintáctico LR(0), vamos a construir el autómata reconocedor de prefijos viables mencionado en el apartado anterior. Para construir este autómata partiremos de la definición de “elemento LR(0)”. A partir de dos funciones (clausura y sucesor) aplicadas sobre conjuntos de elementos LR(0), iremos construyendo el autómata reconocedor de prefijos viables.

Sea G = (N, Σ, P, S) una gramática incontextual reducida. Un *elemento LR(0)* para G es [A $\rightarrow \beta_1 \cdot \beta_2$]; siendo (A $\rightarrow \beta_1 \beta_2$) $\in$ P.

Es decir, un elemento LR(0) no es más que una producción de la gramática con una marca (punto) entre los símbolos de su lado derecho.

La marca (el punto) que aparece en un elemento LR(0) separa la parte que se puede haber analizado (y por lo tanto está en la cima de la pila), de la que se espera analizar (resto de la entrada). Así por ejemplo, el elemento LR(0) [E→E+.T] para la gramática $G_4$ indica que en la cima de la pila se encuentra E +, y en la cinta de entrada se espera encontrar una secuencia de terminales ω tal que T$\rightarrow^*$ ω.

Puesto que el número de producciones de una gramática es finito, y el número de símbolos de la parte derecha de las producciones es finito, el número de elementos LR(0) de una gramática es finito.

Para las producción vacías, como A $\rightarrow \varepsilon$, el único elemento LR(0) posible se representa por [A $\rightarrow \cdot$ ].

Sea G=(N,Σ,P,S) una gramática incontextual reducida. Un elemento LR(0) [A $\rightarrow \beta_1 \cdot \beta_2$] es ***un elemento LR(0) válido para un prefijo viable*** ($\alpha \beta_1$); si dado $\alpha, \beta_1, \beta_2 \in (N \cup \Sigma)^*$; A $\in$ N; ω $\in \Sigma^*$, cumple que:

$$S \rightarrow^* \alpha A \omega \rightarrow \alpha \beta_1 \beta_2 \omega$$

siendo el pivote $(A \rightarrow \beta_1 \beta_2, |\alpha \beta_1 \beta_2|)$

A continuación se definen dos operaciones sobre conjuntos de elementos LR(0), clausura y sucesor, que permitirán calcular la colección canónica de conjuntos de elementos LR(0) y consecuentemente el autómata reconocedor de prefijos viables.

El algoritmo 4.3 define la operación ***clausura*** de un conjunto I de elementos LR(0). Se puede ver que la clausura de un conjunto de elementos LR(0) I contiene a todos los elementos del conjunto I. Si alguno de los elementos incluidos tienen un no-terminal (B) inmediatamente a la derecha del punto, habrá que añadir nuevos elementos LR(0). Estos nuevos elementos LR(0) serán las producciones del no-terminal (B) con la marca al principio de su lado derecho. Este proceso se repetirá hasta que no se añadan nuevos elementos al conjunto I.

```
FUNCION Clausura (I)   /* Para elementos LR(0) */

Clausura (I) := I;

repetir

      para todo [A → α . B β] ∈ clausura (I) hacer
            para todo (B → γ ) ∈ P: [B → . γ ] ∉ clausura (I) hacer
                  clausura (I) := clausura (I) ∪ {[B → . γ ]};

hasta   no se incorporen nuevos elementos LR(0) a clausura(I);
```

**Algoritmo 4.3.** Definición de la operación clausura

Si [A → α **.** B β] pertenece a la clausura de I es porque en algún momento del proceso de análisis sintáctico se espera encontrar en la entrada alguna cadena derivable a partir de B. Consecuentemente, si (B → γ) es una producción de la gramática, en ese mismo momento del análisis se puede encontrar en la entrada cualquier cadena derivable a partir de γ. Esto es equivalente a decir que si [A → α **.** B β] es un elemento LR(0) válido para un prefijo viable θα, el elemento LR(0) [B → **.** γ] también lo será.

La operación ***sucesor*** de un conjunto I de elementos LR(0), para un símbolo x ∈ (N ∪ Σ), se define mediante el algoritmo 4.4. En este algoritmo se puede ver que sucesor de un conjunto de elemento LR(0) I para un símbolo x, se obtiene buscando los elementos LR(0) de I que tienen inmediatamente a la derecha del punto el símbolo x. Se cogen estos elementos, se desplaza el punto a continuación del símbolo x, y se calcula la clausura de estos nuevos elementos.

```
FUNCION Sucesor (I , x ) /* Elementos LR(0)*/

J := ∅;

para todo [A → α . x β] ∈ I
  hacer J = J ∪ {[A → α x . β]};

sucesor (I,x) := clausura (J)
```

**Algoritmo 4.4.** Definición de la operación sucesor

Si I es el conjunto de elementos LR(0) válidos para el prefijo viable θα, sucesor (I, x), con x∈(N∪Σ) será el conjunto de elementos LR(0) válidos para el prefijo viable θαx .

A partir del elemento inicial [S' → **.** S], y mediante la operación sucesor, podemos calcular la ***colección canónica de conjuntos de elemento LR(0)*** (algoritmo 4.5). Para ello se calcula un primer conjunto de elementos LR(0) como clausura de {[S' → **.** S]}. Seguidamente se obtienen

nuevos conjuntos de elementos LR(0) como resultado de calcular sucesor de ese primer conjunto para todos los símbolos de la gramática. Posteriormente se calcula sucesor de los nuevos conjuntos para todos los símbolos de la gramática, y así sucesivamente hasta que no quede ningún conjunto sin calcular sus sucesores. Este proceso terminará porque muchos de los conjuntos sucesor serán el conjunto vacío, y otros conjuntos se repetirán.

```
ALGORITMO   Colección Canónica de Cjtos. de elementos LR (0)
ENTRADA     G' = (N', Σ, P', S');
SALIDA      C: Colección canónica de conjuntos de elementos LR(0)
METODO

C = { clausura ({ [S' → . S] }) };          /* Sin marcar */

repetir
      para todo I no marcado ∈ C hacer
            para todo x ∈ (N' ∪ Σ) hacer
                  si sucesor(I, x) ≠ ∅ entonces
                        C := C ∪ sucesor (I, x) ;
            marcar I
hasta  no queden en C conjuntos no marcados ;
FIN
```

**Algoritmo 4.5.** Cálculo de la colección canónica de conjuntos de elementos LR(0)

*Ejemplo 4.5.*- Sea la gramática $G_5$:

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

Según el algoritmo 4.3 podemos calcular la clausura del elemento LR(0) como:

Clausura( { [E'→ . E] ) } )= { [ E'→ . E ], [ E → . E +T ], [ E → . T ], [ T → . T * F ],
[ T → . F ], [ F → . ( E ) ], [ F → . id ] }

Si llamamos $I_0$ al conjunto de elementos LR(0) obtenido, $I_0$ = Clausura( { [E'→ . E] ) } ), podemos calcular el resultado de la función sucesor sobre el conjunto de elementos LR(0) $I_0$, para el símbolo E:

Sucesor( $I_0$, E )= Clausura ( { [ E'→ E . ], [ E → E . +T ] } ) = {[ E'→ E . ], [ E → E . +T ] }

Según al algoritmo 4.5 se puede calcular toda la colección canónica de elementos LR(0), obteniéndose el autómata LR(0) reconocedor de prefijos viables de la Fig. 4.4.

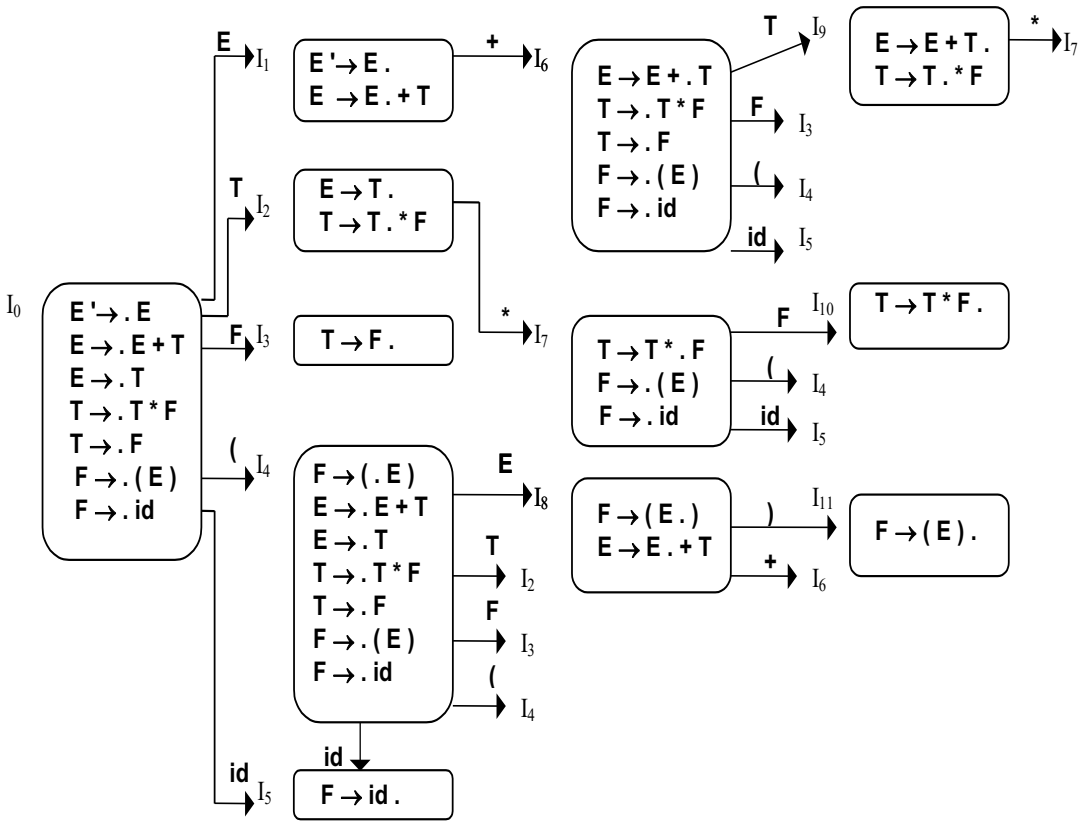



**Figura 4.4.** Autómata LR(0) reconocedor de prefijos viables para la gramática $G_5$

Al calcular de esta forma la colección canónica de conjuntos de elementos LR(0), implícitamente estamos construyendo un autómata que reconoce los prefijos viables de la gramática. El estado inicial se corresponde con la Clausura de { [S'→ **.** S] }. Cada uno de los estados de este autómata se considera estado final, y está formado por el conjunto de elementos LR(0) válidos para el prefijo viable reconocido hasta ese estado. Así por ejemplo, el estado $I_8$ de la Fig. 4.4 está formado por el conjunto de elementos LR(0) válidos para el prefijo viable "(E" de la gramática $G_5$. Esos mismos elementos LR(0) también son válidos para, entre otros, los prefijos viables "((E", "E+(E", o "T*(E".

Si tenemos en cuenta que la marca de cada elemento LR(0) separa la parte analizada (que se encuentra en la cima de la pila) de la no analizada (entrada), podemos encontrarnos con estados que contengan elementos LR(0) de la forma:

a) $[N \rightarrow \alpha \,.\,]$

En este caso se trata de elementos que indican que toda la parte derecha de la producción $N \rightarrow \alpha$ está en la cima de la pila, y por lo tanto se ha detectado un pivote. En estos casos se debe aplicar la reducción indicada por la producción.

b) $[N \rightarrow \alpha_1 \,.\, \alpha_2\,]$, con $\alpha_2 \neq \varepsilon$

Los elementos con esta forma indican que todavía no está el pivote correspondiente a la producción $N \rightarrow \alpha_1 . \alpha_2$ en la cima de la pila (falta toda la cadena $\alpha_2$), y por lo tanto se debe realizar un desplazamiento.

c) [S' → S **.** ]

Este elemento indica que en la pila del analizador se encuentra el símbolo inicial de la cadena S. Si se ha analizado toda la cadena de entrada (en la cinta de entrada se ha llegado al $), se ha finalizado el análisis sintáctico ascendente y se puede concluir que la cadena pertenece al lenguaje reconocido por el autómata.

Con estos criterios, se puede determinar en función del contenido de la pila del analizador (prefijos viables) si se debe reducir, desplazar o aceptar la cadena.

Si en un mismo estado del autómata aparece algún elemento LR(0) que indique reducción (la marca al final de su parte derecha), junto con algún otro elemento que indique desplazamiento, diremos que aparece un *conflicto reducción/desplazamiento* en ese estado. Así por ejemplo, en la Fig. 4.4 se pueden encontrar conflictos reducción/desplazamiento en los estados $I_2$, e $I_9$[3]: En $I_2$ hay un elemento que indica reducción ([ E → T **.** ]) y otro que indica desplazamiento ([ T → T **.** * F ]),. Si en un mismo estado aparecen dos o más elementos LR(0) que indiquen reducciones distintas (elementos con su marca al final de la parte derecha) diremos que aparece un *conflicto reducción/reducción*. En la Fig. 4.4 no aparece ningún conflicto reducción/reducción. Si en algún estado aparece un conflicto desplazamiento/reducción o reducción/reducción la gramática no es LR(0). Se puede demostrar que la no aparición de conflictos es equivalente a la condición LR(0).

En el caso de que aparezca un conflicto en algún conjunto de elementos LR(0), como ocurre en el Ejemplo 4.5, la gramática no es LR(0) pero se puede intentar construir un analizador sintáctico ascendente con una técnica más potente, como la SLR(1) que se presenta en la siguiente sección.

## 4.3. Análisis SLR(1)

Frecuentemente aparecen conflictos al emplear la técnica LR(0) descrita anteriormente. Para solucionar en la medida de lo posible estos conflictos se puede emplear un símbolo de anticipación: Para decidir si se debe desplazar o reducir en un estado, se considera el primer símbolo que hay en la cinta de entrada.

[3] El caso del estado $I_1$ es especial puesto que el elemento LR(0) [E'→E **.** ] indica la aceptación de la cadena de entrada para el símbolo $.

La técnica SLR(1)[4] consiste en calcular la colección canónica de elementos LR(0), y para intentar evitar la aparición de conflictos, cuando un elemento LR(0) indica reducción [N→ α. ], se aplicará la reducción solo cuando en la cadena de entrada haya un símbolo perteneciente a Siguientes del no-terminal N.

Según este criterio, los conflictos que aparecían con la técnica LR(0) en los conjuntos $I_2$ e $I_9$ de la Fig. 4.4 no aparecerán con la técnica de análisis SLR(1). El conjunto $I_9$ estaba formado por los elementos LR(0) [E → E +T.] [T → T. *F], por lo tanto el autómata SLR(1) en el estado 9 desplazará cuando en la cadena de entrada haya un *, y reducirá por la regla número 1 solo cuando haya un símbolo perteneciente a Siguientes (E) = {+, $, )}. De igual forma en el conjunto $I_2$ aparecían los elementos [E → T. ] y [T→ T.* F], por lo tanto cuando el autómata SLR(1) se encuentre en el estado 2 y en la cadena de entrada haya un * desplazará, mientras que cuando haya un símbolo terminal perteneciente a Siguientes(E) = {+, $, )} reducirá según la regla de producción número 2.

Este criterio queda reflejado en la forma de construir la tabla de análisis SLR(1) que muestra el algoritmo 4.6.

```
ALGORITMO   Construcción de la T.A. SLR(1)
ENTRADA     G' = (N', Σ, P', S');
SALIDA      TA (acc,suc)
            acc: Q x (Σ ∪ {$}) → {des-s, red-r, err, acep}
            suc: Q x N → Q
METODO
1.    C := {I0, I1, ..., In};
      /*Colección canónica de cjtos. de elementos LR(0) */
2.    Inicializar TA (acc,suc) con la acción "err";
3.    para todo Ii ∈ C
        para todo elemento LR(0) ∈ Ii hacer
          si ([A → α . a β] ∈ Ii: a ∈ Σ) ∧ sucesor (Ii, a) = Is
                  entonces acc[i,a] := des-s;
          si [S' → S . ] ∈ Ii entonces acc[i,$] := acep;
          si ([A → α . ] ∈ Ii) ∧ (r: A → α ) ∈ P entonces
                  paratodo a ∈ siguientes (A) hacer acc[i,a] := red-r;
4.   paratodo A ∈ N hacer
       Si sucesor (Ii,A) = Is entonces suc [i,A] := S ;
FIN
```

**Algoritmo 4.6.** Construcción tabla de análisis SLR(1)

[4] Simple LR (1)

En la Fig. 4.3 se muestra la tabla completa de análisis SLR(1) para la gramática $G_4$ construida a partir de la colección canónica de conjuntos de elementos LR(0) de la Fig. 4.4 utilizando el algoritmo 4.6.

*Ejemplo 4.6.*- Sea la gramática $G_6$:

$$S' \rightarrow S$$
$$S \rightarrow aSA$$
$$S \rightarrow \varepsilon$$
$$A \rightarrow Bb$$
$$B \rightarrow Ac$$
$$B \rightarrow \varepsilon$$

La colección canónica de elementos LR(0) quedaría tal y como se muestra en la Fig. 4.5.

Si calculamos el conjunto de siguientes de los símbolos no-terminales de la gramática:

SIG (S) = { $, b }
SIG (A) = { $, b, c }
SIG (B) = { b }

La tabla de análisis SLR(1) según el algoritmo 4.6 quedará tal y como aparece en la Fig. 4.6.

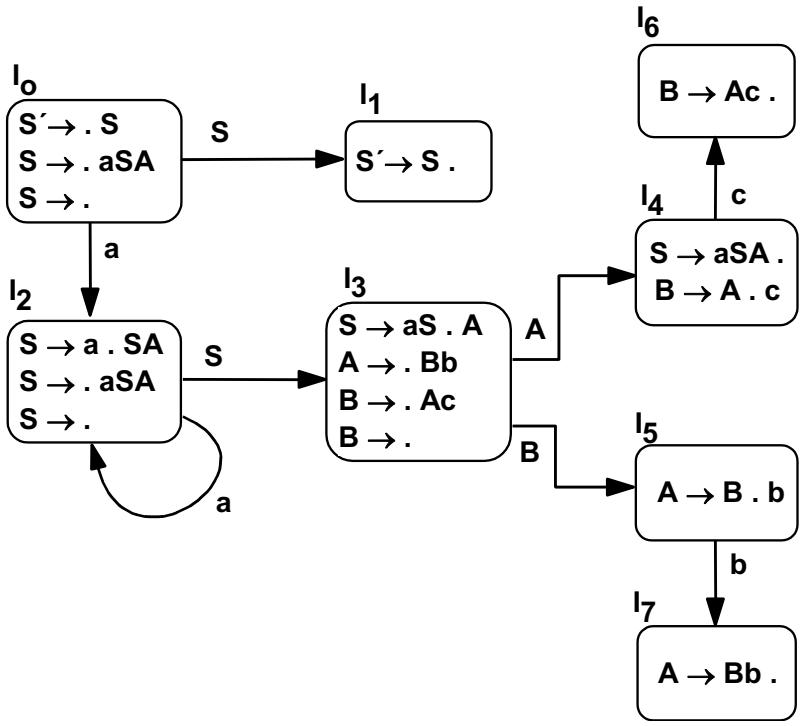



**Figura 4.5.** Colección canónica de elementos LR(0) para $G_6$

| | a | b | c | $ | S | A | B |
|---|---|---|---|---|---|---|---|
| 0 | d2 | r2 | | r2 | 1 | | |
| 1 | | | | Acep | | | |
| 2 | d2 | r2 | | r2 | 3 | | |
| 3 | | r5 | | | | 4 | 5 |
| 4 | | r1 | d6 | r1 | | | |
| 5 | | d7 | | | | | |
| 6 | | r4 | | | | | |
| 7 | | r3 | r3 | r3 | | | |

**Figura 4.6.** Tabla de análisis SLR(1) para $G_6$

Al igual que ocurre en al análisis LR(0), es posible que aparezcan conflictos. Si en un mismo estado aparece algún elemento LR(0) que indique reducción para un símbolo terminal x junto con algún otro elemento que indique desplazamiento para ese mismo símbolo terminal x, diremos que aparece un *conflicto reducción/desplazamiento*. De igual forma, puede aparecer un *conflicto reducción/reducción* si dos elementos LR(0) de un estado indican para un mismo símbolo de anticipación dos reducciones distintas. En cualquiera de los dos casos, no se podrá realizar el análisis SLR(1): La gramática no es SLR(1). En ese caso se puede intentar construir un analizador sintáctico ascendente siguiendo la técnica propuesta en el siguiente apartado.

## 4.4. Análisis LR(1)

Cuando la técnica SLR(1) no permite obtener un analizador ascendente predictivo debido a la aparición de conflictos, se puede intentar emplear una técnica más potente. En este apartado introducimos la técnica de análisis LR(1). La técnica LR(1) se basa en la definición de un nuevo tipo de elemento, los elementos LR(1), y en la redefinición de las funciones sucesor y clausura para que actúen con este nuevo tipo de elementos.

Sea G = (N, Σ, P, S) una gramática incontextual reducida. Un ***elemento LR(1)*** para G es [A $\rightarrow \beta_1 \cdot \beta_2$, a], siendo $(A \rightarrow \beta_1 \beta_2) \in P$ y $a \in \Sigma \cup \{\$\}$.

A $A \rightarrow \beta_1 \cdot \beta_2$ se le llama *núcleo*, y al símbolo terminal a *símbolo de anticipación*.

La marca de un elemento LR(1), al igual que en los elementos LR(0) separa la parte analizada de la que se espera analizar.

El símbolo de anticipación representa un símbolo terminal o $ que puede seguir a la parte derecha del núcleo en una forma sentencial. Precisamente esto será de utilidad cuando se trate de

un elemento que indique reducción (con la marca al final de la parte derecha del núcleo): *solo se aplica la reducción cuando en la cadena de entrada se encuentre el símbolo de anticipación indicado por el elemento LR(1)*.

Sea G = (N,Σ,P,S) una gramática incontextual reducida. Un elemento LR(1) $[A \rightarrow \beta_1 . \beta_2, a]$ es ***un elemento LR(1) válido para un prefijo*** *viable* $(\alpha \beta_1)$; si dado $\alpha, \beta_1, \beta_2 \in (N \cup \Sigma)^*$; $A \in N$; $\omega \in \Sigma^*$ cumple que:

a) $S \rightarrow^* \alpha A \omega \rightarrow \alpha \beta_1 \beta_2 \omega \quad \wedge$

b) $\{a\} = \text{primeros}(\omega) \quad \vee \quad (a = \$ \ \underline{\text{si}} \ \omega = \varepsilon)$

siendo el pivote $(A \rightarrow \beta_1 \beta_2, |\alpha \beta_1 \beta_2|)$

Con el objetivo de construir la colección canónica de conjuntos de elementos LR(1), es necesario redefinir las operaciones clausura y sucesor para elementos LR(1) (algoritmos 4.7 y 4.8)

```
FUNCION Clausura ( I ) /* Para elementos LR(1) */

clausura (I) := I;
repetir
      para todo [A → α . B β, a] ∈ clausura (I) hacer

            para todo (B → γ ) ∈ P :   b ∈ Primeros(βa)  ∧
                [B → . γ, b] ∉ clausura(I) hacer

                  clausura (I) := clausura (I) ∪ {[B → . γ, b]};

hasta no se incorporen nuevos elementos LR(1) a clausura (I);
```

**Algoritmo 4.7.** Función clausura para elementos LR(1)

El algoritmo 4.7 define la operación ***clausura*** de un conjunto I de elementos LR(1). Esta clausura incluirá a todos los elementos del conjunto I. Si alguno de los elementos incluidos tienen un no-terminal (B) inmediatamente a la derecha del punto ($[A \rightarrow \alpha . B \beta , a]$), habrá que añadir nuevos elementos LR(1) formados por las producciones del no-terminal (B) con la marca al principio de su lado derecho. Los símbolos de anticipación para estos nuevos elementos LR(1) se obtienen calculando el conjunto de Primeros de la cadena que sigue al no-terminal (β) concatenada con el símbolo de anticipación del elemento LR(1) (a): Primeros(βa)

El algoritmo 4.8 define la operación sucesor de un conjunto I de elementos LR(1), para un símbolo $x \in (N \cup \Sigma)$. Como puede observarse, esta operación es similar a la definida para un conjunto de elementos LR(0), excepto que en este caso se mantiene el símbolo de anticipación.

```
FUNCION Sucesor (I , x)  /* Elementos LR(1) */

J := ∅;
para todo [A → α . x β , c] ∈ I
 hacer J = J ∪ {[A → α x . β , c]};
       sucesor (I,x) := clausura (J)
```

**Algoritmo 4.8.** Función sucesor para elementos LR(1)

Habitualmente se suelen representar varios elementos LR(1) que solo difieren en el símbolo de anticipación en una forma más compacta:

```
[A → α . β  ,a]
[A → α . β  ,b]          =     [A → α . β  ,a / b / c]
[A → α . β  ,c]
```

A partir del elemento inicial [S' → . S, $], y mediante la operación sucesor, podemos calcular la colección canónica de conjuntos de elementos LR(1), y el autómata reconocedor de prefijos viables LR(1). Este proceso es similar al presentado en el Algoritmo 4.6 pero aplicado ahora a elementos LR(1).

A partir de esta colección canónica de conjuntos de elementos LR(1), y mediante el algoritmo 4.9, se puede construir la tabla de análisis LR(1).

ALGORITMO  Construcción de la T.A. LR(1)

```
ENTRADA     G' = (N', Σ, P', S');
SALIDA      TA (acc,suc)
            acc: Q x (Σ ∪ {$}) → {des-s, red-r, err, acep}
            suc: Q x N → Q ∪ {err}
METODO
1. C := {I0, I1, ..., In};
      /*Colección canónica de cjtos. de elementos LR(1) */
   Inicializar TA (acc,suc) con la acción "err";
2. para todo Ii ∈ C y para todo elemento LR(1) ∈ Ii hacer
      si ([A → α . a β, b] ∈ Ii: a ∈ Σ) ∧ sucesor (Ii, a) = Is entonces
            acc[i,a] := des-s;
      Si [S' → S . , $] ∈ Ii entonces acc[i,$] := acep;
      Si ([A → α . , a] ∈ Ii) ∧ (r: A → α ) ∈ P ∧ A ≠ S' entonces
            acc[i,a] := red-r;
3. paratodo A ∈ N hacer
      Si sucesor (Ii,A) = Is entonces suc [i,A] := S;
FIN
```

**Algoritmo 4.9.** Construcción de la tabla de análisis LR(1)

*Ejemplo 4.7.*- Sea la gramática reducida $G_7$:

$S' \rightarrow S$

$S \rightarrow L = R$

$S \rightarrow R$

$L \rightarrow * R$

$L \rightarrow id$

$R \rightarrow L$

Podemos calcular el autómata reconocedor de prefijos viables LR(1) para la gramática $G_7$ (Fig. 4.7) usando las funciones clausura y sucesor (algoritmos 4. 7 y 4.8) :

Tal y como puede observarse en el autómata de la Fig. 4.7, no aparece ningún tipo de conflicto por lo tanto se puede concluir que la gramática $G_7$ es LR(1). Pero al igual que ocurre con las técnicas LR(0) y SLR(1), podía haber aparecido algún conflicto al construir la tabla de análisis LR(1). En ese supuesto se hubiese concluido que la gramática no era LR(1).

El principal inconveniente de la técnica de construcción del analizador LR(1) es que el autómata resultante tiene un elevado número de estados. En el siguiente apartado se introduce una nueva técnica que permite obtener reconocedores con menos estados.

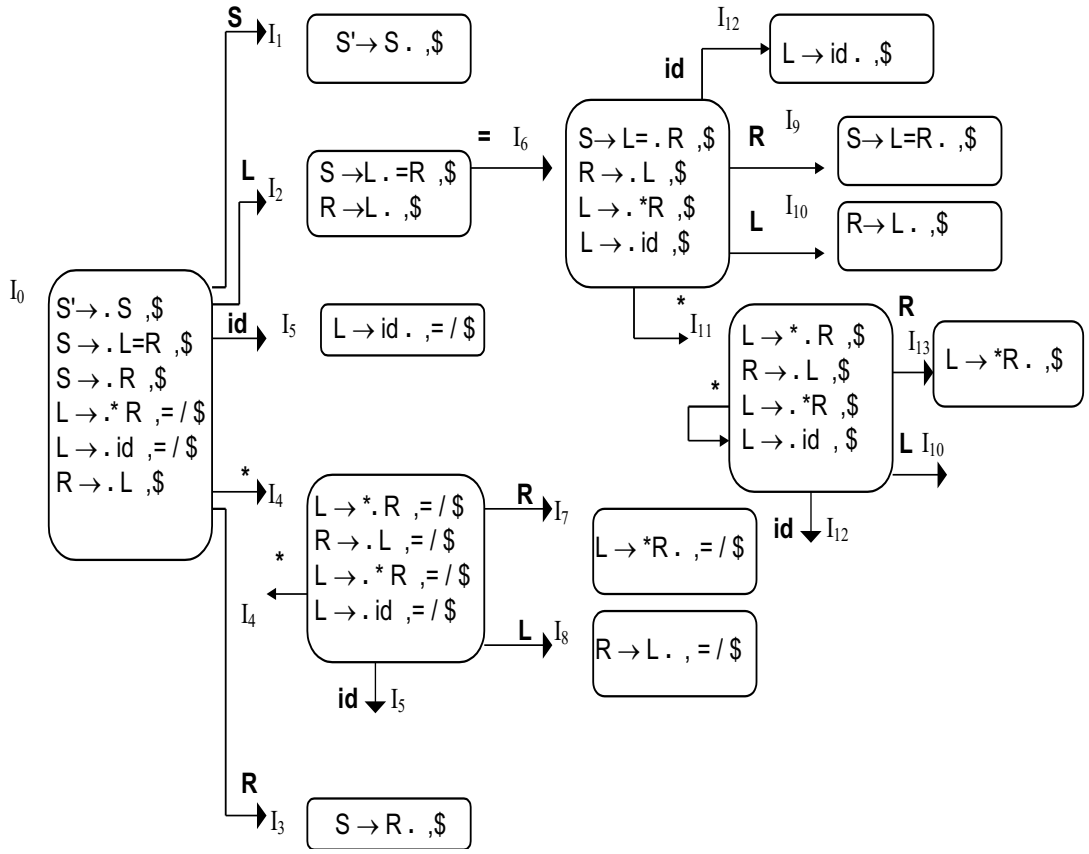



**Figura 4.7.** Autómata reconocedor de prefijos viables LR(1) para $G_7$

# 4.5. Análisis LALR(1)

En este apartado se introduce una nueva técnica que permite construir analizadores sintácticos ascendentes predictivos con menos estados que los LR(1): los analizadores LALR(1)[5]. La idea que origina esta técnica de análisis es la fusión de los estados en el autómata LR(1) que estén formados por elementos con el mismo núcleo.

En [Aho 90] se puede encontrar el algoritmo que permite construir el autómata LALR(1). En este apartado, y con el objetivo de mostrar la idea en la que se basa la técnica LALR(1), se presenta mediante un ejemplo la construcción del autómata LALR(1) a partir de la fusión de estados del autómata LR(1).

*Ejemplo 4.8.*- En el autómata del Ejemplo 4.7 se puede observar que los elementos LR(1) de los estados $I_5$ e $I_{12}$ tienen el mismo núcleo, por lo tanto pueden fusionarse. Al nuevo estado, resultante de la fusión lo llamamos $I_{5\text{-}12}$. Todas las transiciones que llegaban al estado $I_5$ o al $I_{12}$, llegarán al nuevo estado $I_{5\text{-}12}$. Igual ocurre con todas las transiciones que salían de los estados $I_5$ e $I_{12}$ , que ahora saldrán del estado $I_{5\text{-}12}$.

[5] Del inglés LookAhead-LR(1)

$I_5$: [ L → id . , =/$ ]  $I_{12}$: [ L → id . , $ ]

$I_{5\text{-}12}$: [ L → id . , =/$ ]

Lo mismo ocurre con los estados $I_8$ e $I_{10}$, $I_4$ e $I_{11}$ y los estados $I_7$ e $I_{13}$ .

El autómata LALR(1) resultante queda tal y como se muestra en la Fig. 4.8.

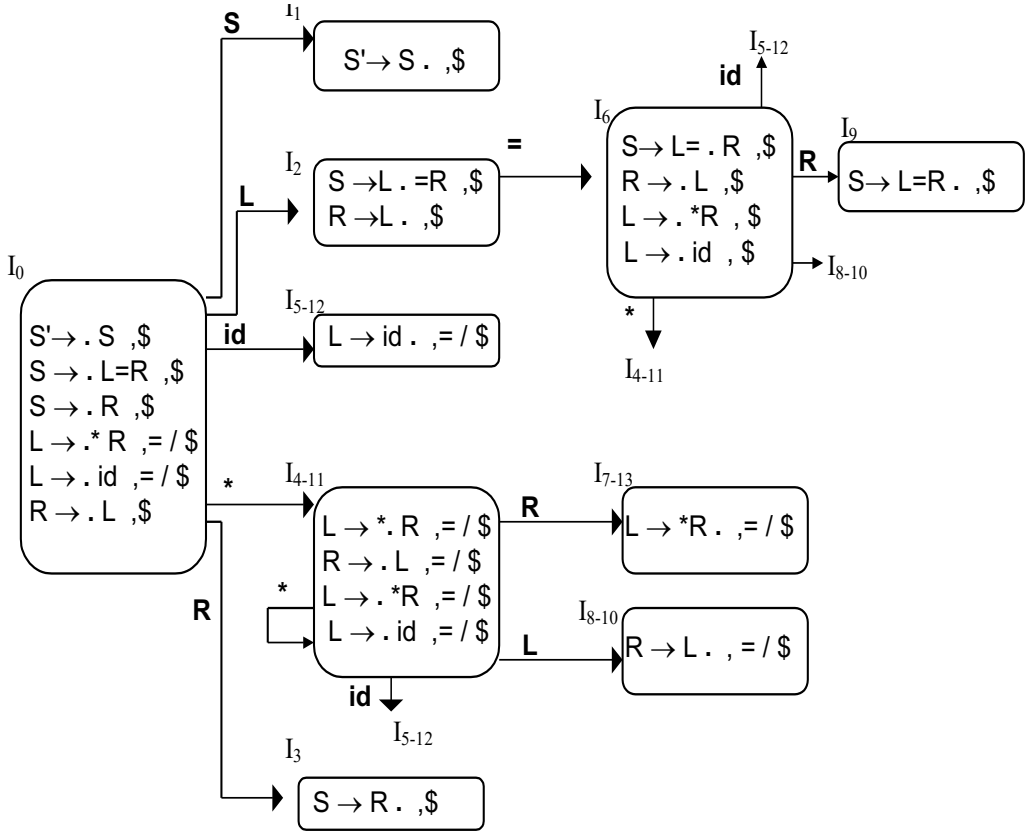



**Figura 4.8.** Autómata LALR(1) para la gramática $G_7$

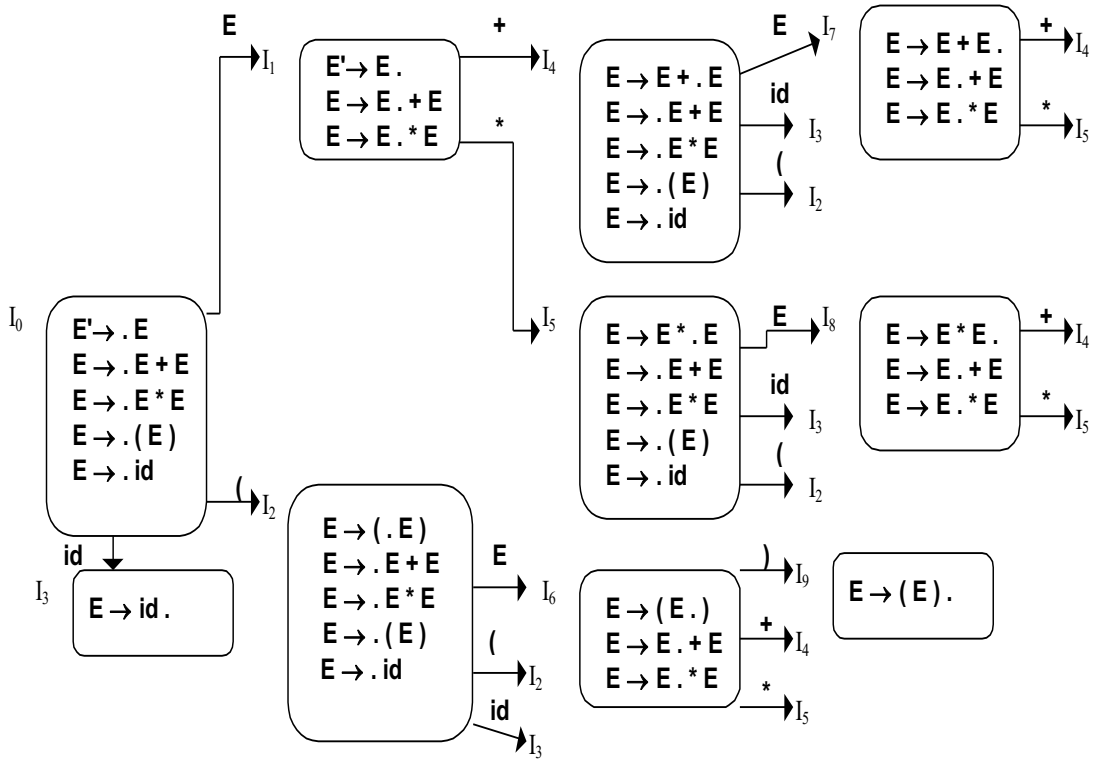
Al construir el autómata LALR(1) mediante esta técnica de fusión de estados del autómata LR(1), pueden aparecer conflictos reducción/reducción que no aparecían en el autómata LR(1). En cambio, puede demostrarse fácilmente que no pueden aparecer conflictos reducción/desplazamiento.

Es posible que el analizador LALR(1) realice alguna reducción en algún caso en el que el LR(1) detectaría un error, pero el analizador LALR(1) lo descubrirá más tarde (antes de desplazar el símbolo erróneo). Se puede observar que el autómata LALR(1) tiene menos estados que el LR(1). De hecho, dada una gramática SLR(1), un analizador LALR(1) tendrá exactamente el mismo número de estados que tendría el analizador SLR(1). Para dar una idea del orden de magnitud de esta diferencia, un analizador LALR(1) para un lenguaje como Pascal puede tener varios cientos de estados, mientras que el analizador LR(1) para el mismo lenguaje tendrá varios miles de estados [AHO90].

## 4.6. Resolución de conflictos

En los apartados anteriores hemos visto como frecuentemente aparecen conflictos en la tablas de análisis ascendente. Algunos de estos conflictos se deben a que la gramática es ambigua. En estos casos se pueden analizar los conflictos para decidir en cada caso qué acción es la más apropiada en función del comportamiento que se desea que tenga el analizador. Vamos a ver esta técnica mediante un ejemplo.

Dada la gramática $G_3$, el autómata reconocedor de prefijos viables LR(0) queda




Puede observarse que al intentar construir la tabla de análisis SLR(1), aparecen conflictos en los estados $I_7$ e $I_8$:

$I_7$: $E \rightarrow E + E .$ $\Rightarrow$ red-1: SIG(E)={ +, *, ), $}

$E \rightarrow E . + E$ $\Rightarrow$ des-4

$E \rightarrow E . * E$ $\Rightarrow$ des-5

$I_8$: $E \rightarrow E * E .$ $\Rightarrow$ red-2: SIG(E)={ +, *, ), $}

$E \rightarrow E . + E$ $\Rightarrow$ des-4

$E \rightarrow E . * E$ $\Rightarrow$ des-5

Esta gramática es ambigua, por lo tanto también aparecerán conflictos si se construye la tabla de análisis LR(1).

La ambigüedad provoca que ante cadenas como $id_1*id_2+id_3$ el autómata no sepa si debe

- Asociar el símbolo * con el $id_1$ y con el $id_2$: $(id_1*id_2)+id_3$, o
- Asociar el símbolo * con el $id_1$ y el resultado de la suma $id_2+id_3$: $id_1*(id_2+id_3)$.

Si consideramos que la interpretación adecuada es la usada habitualmente en matemáticas en la que el operador producto tiene mayor precedencia que el operador suma, el comportamiento que deseamos que tenga el autómata es el del primer caso: $(id_1*id_2)+id_3$.

Según este criterio, cuando el analizador se encuentre en el estado $I_8$, y en la cadena de entrada haya un +, deberá aplicar la reducción correspondiente a la regla 2: E→E*E, para asociar así las dos expresiones unidas por el operador * antes de pasar a analizar el símbolo +.

$I_8$: $E \rightarrow E * E .$ ⇒ ¿ red-2: SIG(E)={ +, *, ), $} ?
$E \rightarrow E . + E$ ⇒ ¿ des-4?

acción (8,+)=r-2 por mayor precedencia del operador * que del +

De esta forma, hemos resuelto uno de los conflictos que aparecían en el estado $I_8$.

Si además deseamos que los operadores * y + tengan asociatividad a izquierdas, podemos resolver el otro conflicto que aparece en el estado $I_8$. Cuando aparezcan cadenas como $id_1*id_2*id_3$, se espera que el analizador lo interprete como $(id_1*id_2)*id_3$, y no como $id_1*(id_2*id_3)$, por lo tanto en el estado $I_8$, cuando en la cadena de entrada haya un *, se debe aplicar la reducción correspondiente a la regla 2: E→E*E, para asociar así las dos primeras expresiones unidas por el operador * antes de pasar a analizar el segundo símbolo *.

$I_8$: $E \rightarrow E * E .$ ⇒ ¿ red-2: SIG(E)={ +, *, ), $} ?
$E \rightarrow E . * E$ ⇒ ¿ des-5 ?

acción (8,*)=r-2 por asociatividad a izquierdas del operador *

De la misma forma, y siguiendo estos dos mismos criterios (mayor precedencia del * frente al +, y asociatividad a izquierdas de los dos operadores) se pueden resolver los dos conflictos que aparecen en el estado $I_7$. Las acciones asociadas a los estados $I_7$ e $I_8$ quedarán una vez resueltos los conflictos

$I_7$: acción (7,+)=r-1 por asociatividad a izquierdas del operador +
acción (7,*)=d-5 por mayor precedencia del operador * que del +

$I_8$: acción (8,+)=r-2 por mayor precedencia del operador * que del +
acción (8,*)=r-2 por asociatividad a izquierdas del operador *

## 4.7. Relaciones entre gramáticas

Aunque las gramáticas LR(1) son un subconjunto de la clase de gramáticas LR(k), se ha demostrado que cualquier gramática LR(k) puede ser transformada en una gramática LR(1) equivalente. Otra relación interesante, es que las gramáticas LL(1) son un subconjunto de la clase de gramáticas LALR(1). En la Fig. 4.9 se muestran las relaciones de inclusión entre los tipos de gramáticas definidas hasta el momento.

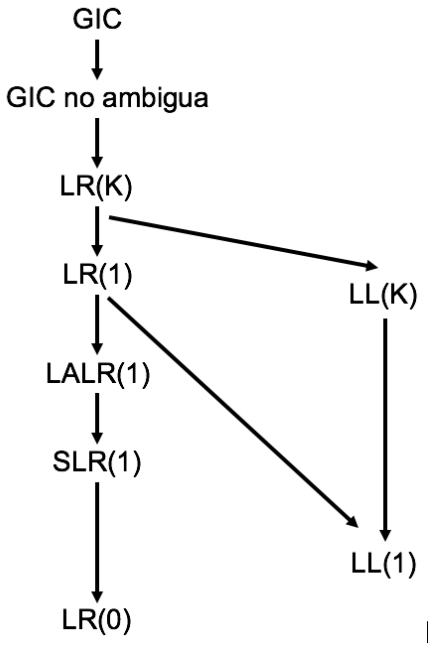



**Figura 4.9.** Relaciones entre gramáticas