Apéndice C

Análisis sintáctico LR generalizado

A partir del algoritmo de Earley es posible derivar la familia de algoritmos LR para el análisis de gramáticas independientes del contexto sin restricciones. Como resultado, se obtienen algoritmos LR Generalizados con complejidad $\mathcal{O}(n^3)$, donde n es la longitud de la cadena de entrada, en el peor caso pero con un mejor comportamiento en casos prácticos. La obtención de esta complejidad se debe a la utilización de programación dinámica para representar la evolución no determinista de la pila, en lugar de las representaciones de pila estructuradas en grafo utilizadas habitualmente por otros autores. El algoritmo resultante puede extenderse fácilmente al caso de las gramáticas de cláusulas definidas y de las gramáticas lineales de índices. La parte de este capítulo concerniente a las gramáticas independientes del contexto está basada en [12, 11, 10], la parte dedicada a gramáticas de cláusulas definidas está basada en [13] y la parte dedicada a gramáticas lineales de índices está basada en [16].

C.1 Introducción

Los algoritmos de análisis LR, que constituyen una de las más poderosas familias de algoritmos de análisis sintáctico para gramáticas independientes del contexto, constan de dos fases diferenciadas:

- una de compilación, en la que la gramática es compilada en una máquina de estado finito denominada *autómata LR* y dos tablas de acciones e ir-a [6, 5];
- una segunda de ejecución, en la que un autómata a pila utilizando el autómata LR como memoria de estado finito determina la pertenencia o no de las sentencias de entrada al lenguaje generado por la gramática.

La eficiencia del proceso de análisis depende de la segunda fase, en cuya descripción se centra este capítulo.

Las gramáticas que pueden ser analizadas determinísticamente mediante analizadores LR con k símbolos de preanálisis constituyen la clase de las *gramáticas LR*, muy útiles a la hora de describir lenguajes de programación pero que resultan insuficientes cuando se trata de analizar el lenguaje natural. Para ampliar el ámbito de los analizadores LR al caso de lenguajes ambiguos, debemos admitir la posibilidad de que las entradas de la tabla de acciones indiquen más de una acción a realizar en un momento dado. En este caso, nos encontramos con los algoritmos LR no deterministas, también conocidos como *LR generalizado.* Tomita ha propuesto un tipo de análisis LR generalizado [199, 200, 201] que utiliza una pila organizada en forma de grafo para representar todos los posibles caminos de análisis de una cadena de entrada dada. Dicho algoritmo presenta problemas con construcciones gramaticales cíclicas y también con aquellas en las que aparece el fenómeno de la recursión izquierda [135]. Otro inconveniente es que su complejidad con respecto a la sentencia de entrada es dependiente de la forma de la gramática; concretamente, su complejidad es $\mathcal{O}(n^{p+1})$, donde p representa la longitud del mayor lado derecho de una producción en la gramática [100].

Se han propuesto varias mejoras al algoritmo de Tomita, entre las que destacan las de Rekers [157], quien ha modificado el algoritmo original de tal modo que ha eliminado los problemas con la recursividad y la ciclicidad, aunque manteniendo la misma complejidad. Nozohoor-Farshi [135] aporta un algoritmo modificado que evita los problemas con la recursividad izquierda. Kipps [100] transforma el algoritmo original en uno de complejidad cúbica a costa de aumentar los requerimientos de espacio. Otros enfoques diferentes utilizan transformaciones en la gramática con el fin de reducir la complejidad espacial y temporal [184], o bien transformaciones en la construcción del autómata LR, aunque en estos el tratamiento de la ciclicidad es más complejo e incluso, con frecuencia, es evitado [127, 121].

Recientemente se ha despertado un gran interés en el establecimiento de relaciones entre diferentes algoritmos de análisis sintáctico y en cómo un algoritmo puede ser derivado a partir de otro [127, 189, 118]. En la mayoría de los casos, se toma el algoritmo de Earley [69] como punto de partida. En este contexto, proponemos algoritmos LR(1) y LALR(1) generalizados que son derivados mediante una secuencia de transformaciones simples del ya bien conocido algoritmo de Earley, preservando su complejidad cúbica en el peor caso pero obteniendo una mejor complejidad en el caso medio e incluso alcanzando complejidad lineal en el caso de gramáticas LR.

C.2 La relación entre los algoritmos de Earley y LR

Un analizador sintáctico Earley [69] construye, para una gramática $\mathcal{G}$ y una cadena de entrada $a_1 \ldots a_n$, una secuencia de $n+1$ conjuntos de ítems , denominados *itemsets*. Cada ítem individual es de la forma $[A \to \alpha \bullet \beta, i, j]$, donde $A \to \alpha \bullet \beta$ indica que la parte α de la producción $A \to \alpha\beta \in P$ ya ha sido reconocida durante el proceso de análisis, j indica que el ítem está en el itemset j e i es un puntero hacia atrás o *backpointer* que apunta al itemset en el cual se inició el reconocimiento de la producción $A \to \alpha\beta$. Siempre se cumple que $\alpha \stackrel{*}{\Rightarrow} a_{i+1} \ldots a_j$ y $0 \leq i \leq j$.

El análisis comienza con la creación del ítem $[S \to \bullet\alpha, 0, 0]$ del itemset 0, donde $S \to \alpha \in P$ es una producción del axioma de la gramática. A partir de entonces, se van aplicando sucesivamente las tres operaciones *scanner*, *predictor* y *completer* hasta que no se pueden generar más ítems. Estas operaciones se describen a continuación:

scanner es una operación que es aplicable a ítems de la forma $[A \to \alpha \bullet a\beta, i, j]$ cuando $a_{j+1} = a$, obteniéndose un nuevo ítem $[A \to \alpha a \bullet \beta, i, j+1]$. Es decir, se ha reconocido el símbolo terminal que estaba justo a la derecha del punto.

predictor es la operación que representa la fase descendente predictiva del algoritmo: a partir de ítems de la forma $[A \to \alpha \bullet B\beta, i, j]$ se generan ítems $[B \to \bullet\gamma, j, j]$ para toda producción $B \to \gamma \in P$, esto es, se predicen todas las producciones que potencialmente pueden ser útiles en el reconocimiento de la cadena de entrada.

completer es una operación que se aplica cuando un ítem tiene una producción con el punto en el extremo derecho. Dado un ítem $[B \to \gamma\bullet, k, j]$ se buscan todos los posibles $[A \to \alpha \bullet B\beta, i, k]$ y se generan nuevos ítems $[A \to \alpha B \bullet \beta, i, j]$ que representan que la subcadena $a_{k+1} \ldots a_j$ puede ser reducida a B y por consiguiente, como la subcadena $a_{i+1} \ldots a_k$ se reduce a α, la subcadena $a_{i+1} \ldots a_j$ se reduce a αB.

El final del proceso de análisis se alcanza cuando se genera un ítem de la forma $[S \rightarrow \alpha\bullet, 0, n]$, indicando que la cadena de entrada pertenece al lenguaje generado por la gramática.

Para describir este algoritmo así como los que restan en este capítulo, haremos uso de los *esquemas de análisis* propuestos por Sikkel [189].

Esquema de análisis sintáctico C.1 El sistema de análisis $\mathbb{P}_{\text{Earley}}$ correspondiente al algoritmo de Earley para una gramática independiente del contexto $\mathcal{G}$ y una cadena de entrada $a_1 \dots a_n$ es el siguiente [191]:

$$\mathcal{I}_{\text{Earley}} = \{ \ [A \rightarrow \alpha \bullet \beta, i, j] \mid A \rightarrow \alpha\beta \in P, \ 0 \leq i \leq j \ \}$$

$$\mathcal{H}_{\text{Earley}} = \{ \ [a, i, i+1] \mid a = a_i \ \}$$

$$\mathcal{D}^{\text{Init}}_{\text{Earley}} = \{ \ \vdash [S \rightarrow \bullet\alpha, 0, 0] \ \}$$

$$\mathcal{D}^{\text{Scan}}_{\text{Earley}} = \{ \ [A \rightarrow \alpha \bullet a\beta, i, j], [a, j, j+1] \vdash [A \rightarrow \alpha a \bullet \beta, i, j+1] \ \}$$

$$\mathcal{D}^{\text{Pred}}_{\text{Earley}} = \{ \ [A \rightarrow \alpha \bullet B\beta, i, j] \vdash [B \rightarrow \bullet\gamma, j, j] \ \}$$

$$\mathcal{D}^{\text{Comp}}_{\text{Earley}} = \{ \ [A \rightarrow \alpha \bullet B\beta, i, k], [B \rightarrow \gamma\bullet, k, j] \vdash [A \rightarrow \alpha B \bullet \beta, i, j] \ \}$$

$$\mathcal{D}_{\text{Earley}} = \mathcal{D}^{\text{Init}}_{\text{Earley}} \cup \mathcal{D}^{\text{Scan}}_{\text{Earley}} \cup \mathcal{D}^{\text{Pred}}_{\text{Earley}} \cup \mathcal{D}^{\text{Comp}}_{\text{Earley}}$$

$$\mathcal{F}_{\text{Earley}} = \{ \ [S \rightarrow \alpha\bullet, 0, n] \ \}$$

§

La definición de las hipótesis realizada en este sistema de análisis sintáctico se corresponde con la estándar y es la misma que se utilizará en los restantes sistemas de análisis del capítulo. Por consiguiente, no las volveremos a definir explícitamente.

En la figura C.1 se muestra una representación gráfica del funcionamiento del algoritmo que facilita la comprensión del mismo. Cada arco en el dibujo representa la parte de la cadena de entrada abarcada por un ítem . Supongamos que la gramática contiene las dos producciones

$$A \rightarrow \alpha B \gamma$$
$$B \rightarrow abc$$

Supongamos también que ya existe un ítem $[A \rightarrow \alpha \bullet B\gamma, i, k]$ indicando que la parte α de la producción reduce la subcadena $a_i \dots a_{k-1}$, hecho que se representa el el dibujo mediante el arco de línea continua fina de la izquierda. A partir de este ítem se predecirán ítems con producciones de B, hecho que se refleja en el dibujo por el arco de línea discontinua. En este caso, sólo se generará el ítem $[B \rightarrow \bullet abc, k, k]$. La sucesiva aplicación de tres operaciones scanner para reconocer los terminales a, b y c dará lugar, respectivamente, a los ítems $[B \rightarrow a \bullet bc, k, k+1]$, $[B \rightarrow ab \bullet c, k, k+2]$ y $[B \rightarrow abc\bullet, k, j]$, que se corresponden con los tres arcos de línea continua

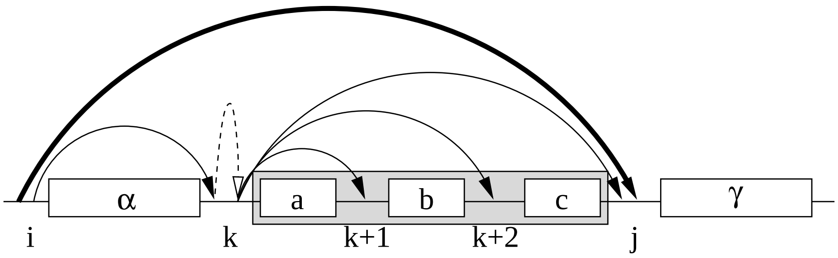

Figura C.1: Representación gráfica del algoritmo de Earley

de la derecha. Combinando este último ítem con $[A \to \alpha \bullet B\gamma, i, k]$ mediante una operación completer, obtenemos el ítem $[A \to \alpha B \bullet \gamma, i, j]$, representado en el dibujo por el arco de línea continua gruesa.

Los pasos $\mathcal{D}^{\mathrm{Pred}}_{\mathrm{Ealrey}}$ del esquema de análisis que acabamos de ver se corresponden con la operación *cerradura* que se utiliza en la construcción del control de estado finito que guía el análisis en los analizadores LR(0). Concretamente, en LR(0) la cerradura de un estado st se realiza incluyendo en st, para cada ítem LR(0) $[A \to \alpha \bullet B\beta]$ de st y para cada producción $B \to \gamma$, todos los ítems $[B \to \bullet\gamma]$. El proceso se repite hasta que no se puede añadir ningún nuevo ítem en st. La misma operación es realizada en tiempo de ejecución en el caso del algoritmo de Earley mediante los pasos Pred si nos percatamos de que las posiciones de la cadena de entrada realmente no intervienen en este tipo de operación. Por tanto, podríamos decir que el esquema de análisis de Earley se corresponde con una "descompilación" de un analizador LR(0).

Siguiendo con la analogía entre Earley y LR(0), los pasos $\mathcal{D}^{\mathrm{Scan}}_{\mathrm{Earley}}$ y $\mathcal{D}^{\mathrm{Comp}}_{\mathrm{Earley}}$ del primero se corresponden con las acciones de desplazamiento y reducción de los analizadores LR(0) clásicos. Una diferencia entre los analizadores Earley y LR estriba en la utilización de gramáticas aumentadas por parte de estos últimos. Este hecho, que se corresponde con la utilización de una producción inicial adicional $\Phi \to S$, es irrelevante en la práctica. En el resto del capítulo, consideraremos que ambos algoritmos utilizan gramáticas aumentadas.

Con el fin de obtener un esquema de análisis más parecido a las operaciones sobre pilas que aparecen en los algoritmos LR clásicos, haremos algunos cambios menores en los pasos Scan y Comp: los componentes i y j de un ítem $[A \to \alpha \bullet \beta, i, j]$ representarán ahora la parte de la cadena de entrada reconocida por el elemento $X \in V$ que aparece inmediatamente antes del punto. Si $\alpha = \varepsilon$, entonces $i = j$. Formalmente, diremos que el algoritmo LR(0) trabaja con ítems

$$\left\{ \ [A \to \alpha' X \bullet \beta, i, j] \mid \exists k, k \leq i, \ \alpha' \overset{*}{\Rightarrow} a_{k+1} \ldots a_i, \ X \overset{*}{\Rightarrow} a_{i+1} \ldots a_j \ \right\}$$

Este cambio guarda cierta relación con el propuesto por Nederhof y Satta en la variante del algoritmo de Earley presentada en [131], en donde, con el fin de posibilitar la compartición de los sufijos comunes entre producciones, se suprime durante la fase predictiva la referencia al primer índice presente en los ítems Earley. Con la modificación que nosotros proponemos no es posible compartir sufijos de producciones distintas pero sí es posible compartir ítems que sólo se diferencian por la posición en la que se comenzó a reconocer una producción dada. Consideremos una producción $A \to \alpha B\beta$, y un conjunto D conteniendo d valores de posiciones de la cadena de entrada. En el algoritmo de Earley, si se han generado d ítems $[A \to \alpha \bullet B\beta, i, \mathrm{j}]$, para $i \in D$ y j fijo, y $B \overset{*}{\Rightarrow} a_{\mathrm{j}} \ldots a_{\mathrm{k}}$ entonces el paso Comp generará un nuevo conjunto de d ítems de la forma $[A \to \alpha B \bullet \beta, i, \mathrm{k}]$.

Como consecuencia del cambio en las posiciones, el paso Comp necesitará tener ahora m ítems como antecedentes, donde m es la longitud del lado derecho de la producción que se pretende reducir. En el nuevo esquema de análisis $LR(0)$, los pasos Scan y Comp han sido renombrados a Shift y Reduce puesto que muestran un comportamiento idéntico a las operaciones de desplazamiento y de reducción de los analizadores LR.

Esquema de análisis sintáctico C.2 El sistema de análisis $\mathbb{P}_{\mathrm{LR}(0)}$ correspondiente al algoritmo LR(0) para una gramática independiente del contexto $\mathcal{G}$ y una cadena de entrada $a_1 \ldots a_n$ es el que se muestra a continuación:

$$\mathcal{I}_{\mathrm{LR}(0)} = \mathcal{I}_{\mathrm{Earley}}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}(0)} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{Earley}}$$

$$\mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}(0)} = \left\{ \; [A \to \alpha \bullet a\beta, i, j], [a, j, j+1] \vdash [A \to \alpha a \bullet \beta, j, j+1] \; \right\}$$

$$\mathcal{D}^{\mathrm{Pred}}_{\mathrm{LR}(0)} = \mathcal{D}^{\mathrm{Pred}}_{\mathrm{Earley}}$$

$$\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}(0)} = \left\{ \begin{array}{l} [B \to X_1 X_2 \cdots X_m \bullet, j_{m-1}, j_m], \ldots, [B \to \bullet X_1 X_2 \cdots X_m, j_0, j_1], [A \to \alpha \bullet B\beta, i, j_0] \\ \vdash [A \to \alpha B \bullet \beta, j_0, j_m] \end{array} \right\}$$

$$\mathcal{D}_{\mathrm{LR}(0)} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}(0)} \cup \mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}(0)} \cup \mathcal{D}^{\mathrm{Pred}}_{\mathrm{LR}(0)} \cup \mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}(0)}$$

$$\mathcal{F}_{\mathrm{LR}(0)} = \mathcal{F}_{\mathrm{Earley}}$$

§

En la figura C.2 se muestra una representación gráfica del modo en que procedería un analizador LR(0) para las dos producciones utilizadas en la figura C.1. En principio, habría un ítem $[A \to \alpha \bullet B\gamma, ?, k]$ con la segunda posición de la cadena de entrada igual a k, mientras que la primera no podemos deducirla de la información existente en el gráfico. En este punto se predice el ítem $[B \to \bullet abc, k, k]$. La aplicación de desplazamientos sobre los terminales a, b y c conlleva la generación de los ítems $[B \to a \bullet bc, k, k+1]$, $[B \to ab \bullet c, k+1, k+2]$ y $[B \to abc \bullet, k+2, j]$ respectivamente. Estos ítems se representan en la figura por arcos de línea continua fina. Precisamente, la principal diferencia entre los algoritmos de Earley y LR(0) queda reflejada en la diferente porción de la cadena de entrada que es abarcada por los ítems correspondientes a los pasos deductivos Scan y Shift. Siguiendo con el proceso de análisis, la reducción de la producción $B \to abc$ conlleva la necesidad de reagrupar los ítems que corresponden al reconocimiento de cada uno de los elementos del lado derecho de la producción, para producir un ítem $[A \to \alpha B \bullet \gamma, k, j]$, representado en la figura por el arco de línea continua gruesa.

C.3 Análisis LR con preanálisis: SLR(1) y LR(1)

A partir del esquema de análisis anterior, podemos obtener el correspondiente al algoritmo SLR(1) mediante la inclusión de un *filtro dinámico* en los pasos Reduce que se encargue de comprobar que el símbolo de preanálisis sea correcto. Para ello debemos utilizar la función *siguiente*, que se define con respecto a la función *primero*. Ambas funciones se muestran a continuación.

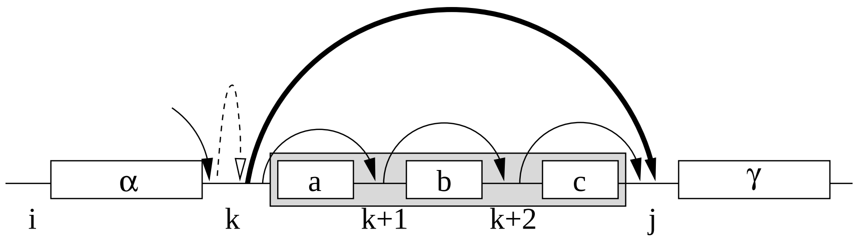

Figura C.2: Representación gráfica del algoritmo LR(0)

Definición C.1 *Un elemento* $a \in V_T \cup \{\epsilon\}$ *pertenece a* $\text{primero}(X)$*, donde* $X \in V$*, si se cumple alguna de las siguientes condiciones:*

- $X = a$
- $X \to \varepsilon \in P$ *y* $a = \varepsilon$
- $X \to Y_1 \cdots Y_i \cdots Y_m \in P$ *y* $a \in \text{primero}(Y_i)$ *y* $\forall_{j=1}^{i-1}\ \varepsilon \in \text{primero}(Y_j)$

La extensión a $\text{primero}(\alpha)$*, donde* $\alpha = X_1 \cdots X_i \cdots X_n \in V$*, es directa:* $a \in \text{primero}(\alpha)$ *si* $a \in \text{primero}(X_1) \cup \cdots \cup \text{primero}(X_i)$ *y* $\varepsilon \notin \text{primero}(X_i)$ *y* $\forall_{j=1}^{i-1}\ \varepsilon \in \text{primero}(X_j)$*. Si* $\alpha \stackrel{*}{\Rightarrow} \varepsilon$ *entonces* $\varepsilon \in \text{primero}(\alpha)$*.*

Definición C.2 *Un elemento* $a \in V_T \cup \{\$\}$ *pertenece a* $\text{siguiente}(A)$*, donde* $A \in V_N$ *y* \$ *es un carácter especial que no pertenece a* V_T *que indica que se ha alcanzado el final de la cadena de entrada, si se cumple alguna de las siguientes condiciones:*

- $a = \$$ *y* A *es el símbolo inicial*
- $A' \to \alpha A \beta \in P$ *y* $a \in (\text{primero}(\beta) - \{\varepsilon\})$
- $A' \to \alpha A \beta \in P$ *y* $\varepsilon \in \text{primero}(\beta)$ *y* $a \in \text{siguiente}(A')$*.*

La comprobación del símbolo de preanálisis se realiza en los pasos deductivos Reduce, a los cuales se les ha incorporado el filtro dinámico $\exists[a, j, j+1] \in \mathcal{H}_{\text{SLR}},\ a \in \text{siguiente}(B)$. Dichos pasos quedan ahora como sigue:

$$\mathcal{D}_{\text{SLR}(1)}^{\text{Reduce}} = \left\{ \begin{array}{l} [B \to X_1 X_2 \cdots X_m \bullet, j_{m-1}, j_m], \\ \vdots \\ [B \to \bullet X_1 X_2 \cdots X_m, j_0, j_1], \\ [A \to \alpha \bullet B\beta, i, j_0] \\ \vdash [A \to \alpha B \bullet \beta, j_0, j_m] \mid \\ \exists[a, j_m, j_m + 1] \in \mathcal{H}_{\text{SLR}}, \\ a \in \text{siguiente}(B) \end{array} \right\}$$

Esquema de análisis sintáctico C.3 El sistema de análisis $\mathbb{P}_{\text{SLR}(1)}$ correspondiente al algoritmo SLR(1) para una gramática independiente del contexto $\mathcal{G}$ y una cadena de entrada $a_1 \ldots a_n$ queda definido por:

$$\mathcal{I}_{\text{SLR}(1)} = \mathcal{I}_{\text{lR}(0)}$$

$$\mathcal{D}_{\text{SLR}(1)} = \mathcal{D}_{\text{LR}(0)}^{\text{Init}} \cup \mathcal{D}_{\text{LR}(0)}^{\text{Shift}} \cup \mathcal{D}_{\text{LR}(0)}^{\text{Pred}} \cup \mathcal{D}_{\text{SLR}(1)}^{\text{Reduce}}$$

$$\mathcal{F}_{\mathrm{SLR}(1)} = \mathcal{F}_{\mathrm{Earley}}$$

§

Proposición C.1 LR(0) $\stackrel{\mathbf{df}}{\Longrightarrow}$ SLR(1).

Demostración:

Consideraremos los sistemas de análisis sintáctico $\mathbb{P}_{\mathrm{LR}(0)}$ y $\mathbb{P}_{\mathrm{SLR}(1)}$ para una gramática $\mathcal{G}$ y una cadena de entrada $a_1 \ldots a_n$ cualquiera. Trivialmente $\mathcal{I}_{\mathrm{LR}(0)} \supseteq \mathcal{I}_{\mathrm{SLR}(1)}$ puesto que $\mathcal{I}_{\mathrm{LR}(0)} = \mathcal{I}_{\mathrm{SLR}(1)}$. Para demostrar que $\vdash_{LR(0)} \supseteq \vdash_{SLR(1)}$ bastará con mostrar que $\vdash_{LR(0)} \supseteq \mathcal{D}_{\mathrm{SLR}(1)}$. Los pasos Init, Shift y Pred de $\mathbb{P}_{\mathrm{SLR}(1)}$ son los mismos que en $\mathbb{P}_{\mathrm{LR}(0)}$, mientras que para cada paso $\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{SLR}(1)}$ existe trivialmente un paso $\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}(0)}$ y por tanto una inferencia. □

El análisis LR(1) hace un uso más elaborado del preanálisis que el SLR(1). En vez de limitarse a comprobar si el siguiente elemento de la cadena de entrada es compatible con la reducción en curso, infiere los posibles símbolos de preanálisis para cada ítem en el momento de su creación [6]. Para ello es necesario modificar la estructura de los ítems, ya que se incorpora un nuevo elemento: el símbolo de preanálisis. En consecuencia, los ítems del esquema de análisis LR(1) son el resultado de refinar los ítems del esquema LR(0), ya que cada uno de éstos últimos puede ser considerado como el representante del conjunto de ítems LR(1) que poseen la misma producción con punto y las mismas posiciones pero diferente símbolo de preanálisis:

$$\{\ [A \rightarrow \alpha \bullet \beta, b, i, j]\ \}$$

donde b es el símbolo de preanálisis. El esquema de análisis, en el cual el símbolo de preanálisis es inferido en $\mathcal{D}^{\mathrm{Pred}}_{\mathrm{LR}}$ y comprobado en $\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}}$ por medio de un filtro dinámico, es el siguiente:

Esquema de análisis sintáctico C.4 El sistema de análisis $\mathbb{P}_{\mathrm{LR}}$ correspondiente al algoritmo LR(1) para una gramática independiente del contexto $\mathcal{G}$ y una cadena de entrada $a_1 \ldots a_n$ queda definido por:

$$\mathcal{I}_{\mathrm{LR}} = \{\ [A \rightarrow \alpha \bullet \beta, b, i, j] \mid A \rightarrow \alpha\beta \in P,\ \ b \in V_T,\ \ 0 \leq i \leq j\ \}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}} = \{\vdash [S \rightarrow \bullet\alpha, \$, 0, 0]\}$$

$$\mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}} = \{\ [A \rightarrow \alpha \bullet a\beta, b, i, j], [a, j, j+1] \vdash [A \rightarrow \alpha a \bullet \beta, b, j, j+1]\ \}$$

$$\mathcal{D}^{\mathrm{Pred}}_{\mathrm{LR}} = \{\ [A \rightarrow \alpha \bullet B\beta, b, i, j] \vdash [B \rightarrow \bullet\gamma, b', j, j] \mid b' \in \mathrm{primero}(\beta b)\ \}$$

$$\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}} = \left\{ \begin{array}{l} [B \rightarrow X_1 X_2 \cdots X_m \bullet, b', j_{m-1}, j_m], \ldots, [B \rightarrow \bullet X_1 X_2 \cdots X_m, b', j_0, j_1], \\ [A \rightarrow \alpha \bullet B\beta, b, i, j_0] \\ \vdash [A \rightarrow \alpha B \bullet \beta, b, j_0, j_m] \mid \exists [b', j_m, j_m+1] \in \mathcal{H}_{\mathrm{LR}},\ \ b' \in \mathrm{primero}(\beta b) \end{array} \right\}$$

$$\mathcal{D}_{\mathrm{LR}} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}} \cup \mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}} \cup \mathcal{D}^{\mathrm{Pred}}_{\mathrm{LR}} \cup \mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}}$$

$$\mathcal{F}_{\mathrm{LR}} = \{\ [S \rightarrow \alpha\bullet, \$, 0, n]\ \}$$

§

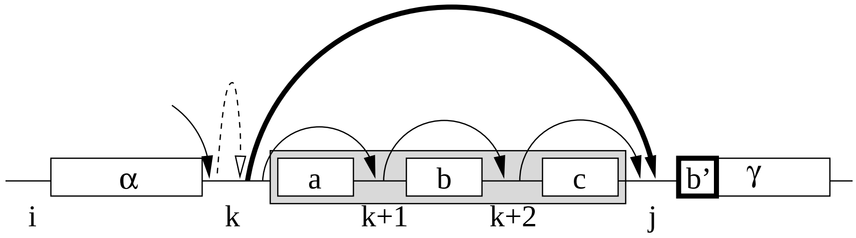

Figura C.3: Representación gráfica del algoritmo LR(1)

En la figura C.3 se representa gráficamente el proceso de análisis LR(1) para el caso de las 2 producciones utilizadas en la figura C.1. La única diferencia relevante con respecto a la figura C.2 es que ahora los pasos Pred calculan b' como posible símbolo de preanálisis, que será comparado con el primer símbolo de γ en el momento de aplicar la reducción de la producción $B \rightarrow abc$.

Proposición C.2 $\mathbf{SLR(1)} \stackrel{\text{ir}}{\Longrightarrow} \mathbf{SLR(1)'} \stackrel{\text{df}}{\Longrightarrow} \mathbf{LR}$.

<u>Demostración</u>:

Como primer paso definiremos el sistema de análisis $\mathbb{P}_{\mathrm{SLR(1)'}}$ para una gramática $\mathcal{G}$ y una cadena de entrada $a_1 \ldots a_n$:

$$\mathcal{I}_{\mathrm{SLR(1)'}} = \left\{ \; [A \rightarrow \alpha \bullet \beta, b, i, j] \mid A \rightarrow \alpha\beta \in P, \;\; b \in V_T, \;\; 0 \leq i \leq j \; \right\}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{SLR(1)'}} = \{\vdash [S \rightarrow \bullet\alpha, \$, 0, 0]\}$$

$$\mathcal{D}^{\mathrm{Shift}}_{\mathrm{SLR(1)'}} = \left\{ \; [A \rightarrow \alpha \bullet a\beta, b, i, j], [a, j, j+1] \vdash [A \rightarrow \alpha a \bullet \beta, b, j, j+1] \; \right\}$$

$$\mathcal{D}^{\mathrm{Pred}}_{\mathrm{SLR(1)'}} = \left\{ \; [A \rightarrow \alpha \bullet B\beta, b, i, j] \vdash [B \rightarrow \bullet\gamma, b', j, j] \; \right\}$$

$$\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{SLR(1)'}} = \left\{ \begin{array}{l} [B \rightarrow X_1X_2\cdots X_m\bullet, b', j_{m-1}, j_m], \ldots, [B \rightarrow \bullet X_1X_2\cdots X_m, b', j_0, j_1], \\ [A \rightarrow \alpha \bullet B\beta, b, i, j_0] \\ \vdash [A \rightarrow \alpha B \bullet \beta, b, j_0, j_m] \mid \exists [a, j_m, j_m+1] \in \mathcal{H}_{\mathrm{SLR}}, \;\; a \in \mathrm{siguiente}(B) \end{array} \right\}$$

$$\mathcal{D}_{\mathrm{SLR(1)'}} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{SLR(1)'}} \cup \mathcal{D}^{\mathrm{Shift}}_{\mathrm{SLR(1)'}} \cup \mathcal{D}^{\mathrm{Pred}}_{\mathrm{SLR(1)'}} \cup \mathcal{D}^{\mathrm{Reduce}}_{\mathrm{SLR(1)'}}$$

$$\mathcal{F}_{\mathrm{SLR(1)'}} = \left\{ \; [S \rightarrow \alpha\bullet, \$, 0, n] \; \right\}$$

Para demostrar que $\mathbf{SLR(1)} \stackrel{\text{ir}}{\Longrightarrow} \mathbf{SLR(1)'}$, definiremos la siguiente función

$$f([A \rightarrow \alpha \bullet \beta, b, i, j]) = \{ \; [A \rightarrow \alpha \bullet \beta, i, j] \; \}$$

de la cual se obtiene directamente que $\mathcal{I}_{\mathrm{SLR(1)}} = f(\mathcal{I}_{\mathrm{SLR(1)'}})$ y que $\Delta_{\mathrm{SLR(1)}} = f(\Delta_{\mathrm{SLR(1)'}})$ por inducción en la longitud de las secuencias de derivación. En consecuencia, $\mathbb{P}_{\mathrm{SLR(1)}} \stackrel{\text{ir}}{\longrightarrow} \mathbb{P}_{\mathrm{SLR(1)'}}$, con lo que hemos probado lo que pretendíamos.

Para demostrar que $\mathbf{SLR(1)'} \stackrel{\text{df}}{\Longrightarrow} \mathbf{LR}$, deberemos demostrar que para todo esquema de análisis $\mathbb{P}_{\mathrm{SLR(1)'}}$ y $\mathbb{P}_{\mathrm{LR}}$ se cumple que $\mathcal{I}_{\mathrm{SLR(1)'}} \supseteq \mathcal{I}_{\mathrm{LR}}$ y $\vdash_{SLR(1)'} \supseteq \vdash_{LR}$. Lo primero es trivial, puesto que $\mathcal{I}_{\mathrm{SLR(1)'}} = \mathcal{I}_{\mathrm{LR}}$. Para lo segundo debemos mostrar que $\vdash_{SLR(1)'} \supseteq \mathcal{D}_{\mathrm{LR}}$. Los pasos Init y Scan son idénticos en ambos sistemas de análisis. Por otra parte, es claro que $\mathcal{D}^{\mathrm{Pred}}_{\mathrm{SLR(1)'}} \supseteq \mathcal{D}^{\mathrm{Pred}}_{\mathrm{LR}}$ y $\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{SLR(1)'}} \supseteq \mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}}$ puesto que primero(βb) y primero(βb) son condiciones más restrictivas que $\emptyset$ y siguiente(B), respectivamente. $\square$

C.4 LR(1) y LALR(1) con tablas precompiladas

Se puede obtener un algoritmo más eficiente mediante la compilación de los pasos Pred en un autómata finito, llamado *autómata LR*, tal y como se hace en los algoritmos LR clásicos [6]. Dicha compilación se realiza mediante la aplicación de una función *cerradura*. Con ello se consigue evitar la aplicación de pasos Pred en tiempo de ejecución. El conjunto de ítems del nuevo esquema de análisis LR^{c} es equivalente al conjunto de ítems del esquema LR puesto que los ítems $[A \rightarrow \alpha \bullet \beta, b, i, j]$ son simplemente reemplazados por ítems $[st, i, j]$, donde st representa la clase de equivalencia de todos los ítems que contienen la producción con punto $A \rightarrow \alpha \bullet \beta$ y el símbolo de preanálisis b'. Además de en eficiencia, también se gana en flexibilidad, puesto que ahora para aplicar un esquema LR(1) o LALR(1) tan sólo es necesario cambiar la fase de compilación, manteniendo sin cambios los pasos deductivos del esquema de análisis. La prueba de corrección del esquema de análisis se fundamenta en la corrección de la estragia de análisis LR utilizada [6].

Esquema de análisis sintáctico C.5 El sistema de análisis $\mathbb{P}_{\mathrm{LR}^{\mathrm{c}}}$, correspondiente al algoritmo LR(1) utilizando tablas precompiladas, dada una gramática independiente del contexto $\mathcal{G}$ y una cadena de entrada $a_1 \ldots a_n$ queda definido por:

$$\mathcal{I}_{\mathrm{LR}^{\mathrm{c}}} = \left\{ \ [st, i, j] \mid st \in \mathcal{S}, \ \ 0 \leq i \leq j \ \right\}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^{\mathrm{c}}} = \{\vdash [st_0, 0, 0]\}$$

$$\mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^{\mathrm{c}}} = \left\{ \ [st, i, j], [a, j, j+1] \vdash [st', j, j+1] \mid \mathrm{shift}_{st'} \in \mathrm{acción}(st, a) \ \right\}$$

$$\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}^{\mathrm{c}}} = \left\{ \begin{array}{l} [st^m, j_{m-1}, j_m], , \ldots, [st^1, j_0, j_1], [st^0, i, j_0] \vdash [st, j_0, j_m] \mid \\ \exists [a, j_m, j_m+1] \in \mathcal{H}_{\mathrm{LR}^{\mathrm{c}}}, \ \mathrm{reduce}_r \in \mathrm{acción}(st^m, a), \\ st^i \in \mathrm{revela}(st^{i+1}), \ \ st \in \mathrm{ir\_a}(st^0, \mathrm{lhs}(r)), \\ m = \mathrm{longitud}(\mathrm{rhs}(r)) \end{array} \right\}$$

$$\mathcal{D}_{\mathrm{LR}^{\mathrm{c}}} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^{\mathrm{c}}} \cup \mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^{\mathrm{c}}} \cup \mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}^{\mathrm{c}}}$$

$$\mathcal{F}_{\mathrm{LR}^{\mathrm{c}}} = \left\{ \ [st_f, 0, n] \ \right\}$$

donde $\mathcal{S}$ es el conjunto de estados en el autómata LR, $st_0 \in \mathcal{S}$ es el estado inicial, $st_f \in \mathcal{S}$ es el estado final, $\mathrm{lhs}(r)$ es el no-terminal del lado izquierdo de la producción r, y donde $st^i \in \mathrm{revela}(st^{i+1})$ es equivalente a $st^{i+1} \in \mathrm{ir\_a}(st^i, X)$ cuando $X \in V_N$ y a $\mathrm{shift}_{st^{i+1}} \in \mathrm{acción}(st^i, X)$ cuando $X \in V_T$. Finalmente *acción* e *ir_a* se refieren a las tablas[1] en las que se ha codificado el comportamiento del autómata LR:

- La tabla de acciones determina qué acciones se deben realizar para cada combinación de estado y símbolo de preanálisis. Concretamente, en el caso de se deba realizar un desplazamiento indica el estado al que hay que desplazar y en el caso de acciones de reducción la producción que deberá ser aplicada.

[1] Se puede incrementar la velocidad del analizador sintáctico transformando dichas tablas en código, obteniendo en contrapartida un ejecutable de mayor tamaño [86].

- La tabla de ir_a determina cual será el nuevo estado del autómata LR después de realizar una reducción. Para acceder a una entrada se utiliza el estado actual y el no-terminal situado en el lado izquierdo de la producción que se ha reducido.

§

C.5 LR(1) y LALR(1) con complejidad $\mathcal{O}(n^3)$

La utilización de los pasos Reduce en los esquemas de análisis anteriores incrementa la complejidad de los algoritmos a $\mathcal{O}(n^{p+1})$, donde p es la máxima longitud de la parte derecha de una producción en la gramática que se esté considerando. Por consiguiente, sólo se obtendrá una complejidad de orden cúbico cuando toda producción tenga a lo sumo dos elementos en su parte derecha. Para obtener una complejidad de orden $\mathcal{O}(n^3)$ sin restringir la longitud de las producciones, seguiremos la sugerencia de Johnson [88] de utilizar el método denominado *binarización implícita de producciones*, descrito por Lang[2] en [107]. Básicamente, este proceso consiste en transformar una reducción de una producción con m elementos en su parte derecha en m reducciones de producciones que poseen a lo sumo 2 elementos en su parte derecha.

Siguiendo este enfoque, la reducción de la producción

$$A_{r,0} \to A_{r,1} \ldots A_{r,n_r}$$

se transformaría en la reducción de las siguientes $n_r + 1$ producciones:

$$\begin{array}{l} A_{r,0} \to \nabla_{r,0} \\ \nabla_{r,0} \to A_{r,1}\ \nabla_{r,1} \\ \vdots \\ \nabla_{r,n_r-1} \to A_{r,n_r}\ \nabla_{r,n_r} \\ \nabla_{r,n_r} \to \varepsilon \end{array}$$

donde los símbolos nabla son *frescos*, esto es, diferentes de cualquier otro símbolo de la gramática. Un aspecto importante a considerar es que no es necesario tratar explícitamente la existencia de esas $n_r + 1$ nuevas producciones. Bien al contrario, el algoritmo trabaja siempre sobre las producciones originales. Esto se consigue introduciendo los símbolos ∇ directamente en el interior de las producciones. En efecto, la producción

$$A_{r,0} \to A_{r,1} \ldots A_{r,n_r}$$

pasa a ser vista por el algoritmo como constituida por los elementos

$$A_{r,0} \to \nabla_{r,0}\ A_{r,1}\ \nabla_{r,1}\ \ldots A_{r,n_r}\ \nabla_{r,n_r}$$

de tal modo que los símbolos ∇ sirven de indicadores para señalar la parte de la producción que ha sido reducida, o equivalentemente, cuáles de las reducciones binarias han sido aplicadas. Por ejemplo, un ítem conteniendo ∇_{r,n_r} indicará que se ha reducido la producción $\nabla_{r,n_r} \to \varepsilon$, mientras que un ítem conteniendo ∇_{r,n_r-1} indicará que ya han sido reducidas las producciones $\nabla_{r,n_r} \to \varepsilon$ y $\nabla_{r,n_r-1} \to A_{r,n_r}\ \nabla_{r,n_r}$. La presencia de $\nabla_{r,0}$ indicará que toda la parte derecha de la producción original ha sido reducida y que ya sólo queda por generar el símbolo $A_{r,0}$, correspondiente al lado izquierdo de la producción[3].

[2] Un método equivalente llamado *bilinearización* es descrito por Leermakers en [111].

[3] Es interesante señalar la similitud entre $\nabla_{r,i}$ y la producción con punto $A_{r,0} \to \alpha \bullet \beta$, donde $\alpha = A_{r,1} \ldots A_{r,i}$ y $\beta = A_{r,i+1} \ldots a_{r,n_r}$.

De lo anterior se deduce que este nuevo tratamiento de las reducciones lleva aparejado un cambio en la forma de los ítems , puesto que se deberá incluir un nuevo elemento que representará o bien un símbolo ∇, indicando que los elementos $A_{r,i+1} \ldots A_{r,n_r}$ de la producción ya han sido reducidos, o bien un símbolo perteneciente a la gramática, que puede ser un terminal si el ítem ha sido generado como resultado de un desplazamiento o un no-terminal si el ítem ha sido generado al finalizar una reducción. En este último caso, dicho no-terminal se corresponderá con el símbolo situado en el lado izquierdo de la producción reducida. En resumen, siguiendo la terminología de los esquemas de análisis diremos que aplicamos un refinamiento a los ítems.

Con respecto a los pasos deductivos, aplicaremos un refinamiento a los pasos Shift, puesto que ahora deberemos diferenciar si desplazamos el primer elemento del lado derecho de una producción (InitShift) o cualquier otro elemento del lado derecho (Shift). Por su parte, los pasos Reduce también deben ser refinados y reemplazados por los siguientes tres pasos: Sel para indicar que una producción a sido seleccionada para reducción, Red para realizar la reducción implícita de una producción binaria y Head para indicar la finalización de la reducción y el reconocimiento del lado izquierdo de la producción.

Esquema de análisis sintáctico C.6 El sistema de análisis $\mathbb{P}_{\mathrm{LR}^3}$, correspondiente al algoritmo LR(1) con complejidad cúbica, dada una gramática independiente del contexto $\mathcal{G}$ y una cadena de entrada $a_1 \ldots a_n$ queda definido por:

$$\mathcal{I}_{\mathrm{LR}^3} = \left\{ \; [A, st, i, j] \cup [\nabla_{r,s}, st, i, j] \mid A \in V_N \cup V_T, \;\; st \in \mathcal{S}, \;\; 0 \leq i \leq j \; \right\}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^3} = \left\{ \; \vdash [-, st_0, 0, 0] \; \right\}$$

$$\mathcal{D}^{\mathrm{InitShift}}_{\mathrm{LR}^3} = \left\{ \begin{array}{l} [A, st, i, j] \vdash [A_{r,1}, st', j, j+1] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\mathrm{LR}^3}, \;\; A_{r,1} = a, \;\; \mathrm{shift}_{st'} \in \mathrm{acción}(st, a), \;\; A \in V \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^3} = \left\{ \begin{array}{l} [A_{r,s}, st, i, j] \vdash [A_{r,s+1}, st', j, j+1] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\mathrm{LR}^3}, \;\; A_{r,s+1} = a, \;\; \mathrm{shift}_{st'} \in \mathrm{acción}(st, a) \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Sel}}_{\mathrm{LR}^3} = \left\{ \begin{array}{l} [A, st, i, j] \vdash [\nabla_{r,n_r}, st, j, j] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\mathrm{LR}^3}, \;\; \mathrm{reduce}_r \in \mathrm{acción}(st, a), \;\; A \in V \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Red}}_{\mathrm{LR}^3} = \left\{ \begin{array}{l} [\nabla_{r,s}, st, k, j], [A_{r,s}, st, i, k] \vdash [\nabla_{r,s-1}, st', i, j] \mid \\ st' \in \mathrm{revela}(st) \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Head}}_{\mathrm{LR}^3} = \left\{ \; [\nabla_{r,0}, st, i, j] \vdash [A_{r,0}, st', i, j] \mid st' \in \mathrm{ir\_a}(st, A_{r,0}) \; \right\}$$

$$\mathcal{D}_{\mathrm{LR}^3} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^3} \cup \mathcal{D}^{\mathrm{InitShift}}_{\mathrm{LR}^3} \cup \mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^3} \cup \mathcal{D}^{\mathrm{Sel}}_{\mathrm{LR}^3} \cup \mathcal{D}^{\mathrm{Red}}_{\mathrm{LR}^3} \cup \mathcal{D}^{\mathrm{Head}}_{\mathrm{LR}^3}$$

$$\mathcal{F}_{\mathrm{LR}^3} = \left\{ \; [S, st_f, 0, n] \; \right\}$$

§

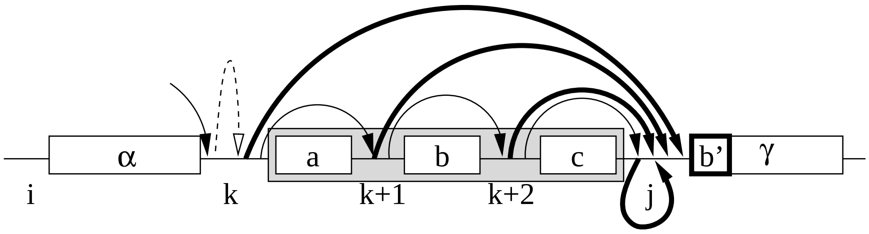

Figura C.4: Representación gráfica del algoritmo LR(1) con complejidad cúbica

En la figura C.4 se muestra cómo procede el algoritmo que se acaba de proponer en el caso del análisis de las producciones utilizadas en la figura C.1. El funcionamiento es prácticamente idéntico al de los algoritmos anteriores en lo que respecta a las acciones de desplazamiento de los terminales a, b y c. Las diferencias surgen en el proceso de la reducción de la producción $B \to abc$. Ahora, esta operación de reducción se realiza en varias fases, comenzando por la aplicación de un paso Sel que genera el ítem representado en la figura C.4 por el arco de trazo grueso de la parte inferior. Este ítem se combina con el resultante del desplazamiento del terminal c mediante un paso Red para realizar la primera reducción binaria implícita, dando lugar al arco de trazo grueso situado más a la derecha en la parte superior de la figura. Este ítem se combina a su vez con el resultante del desplazamiento del terminal b mediante otro paso Red para generar el ítem representado por el segundo arco de trazo grueso de la parte superior de la figura. Este ítem también se combina mediante un paso Red, esta vez con el ítem resultante del desplazamiento del terminal a, dando lugar al ítem correspondiente a la finalización de la reducción de la producción $B \to abc$, representado por el primer arco de trazo grueso en la figura C.4. Para finalizar, ya sólo queda aplicar un paso Head.

Proposición C.3 $\mathbf{LR^c} \overset{\mathbf{ir}}{\Longrightarrow} \mathbf{LR^{c\prime}} \overset{\mathbf{sr}}{\Longrightarrow} \mathbf{LR^3}$.

Demostración:

Como primer paso definiremos el sistema de análisis $\mathbb{P}_{\mathrm{LR}^{c\prime}}$ para una gramática $\mathcal{G}$ y una cadena de entrada $a_1 \dots a_n$:

$$\mathcal{I}_{\mathrm{LR}^{c\prime}} = \left\{ \; [A, st, i, j] \cup [\nabla_{r,s}, st, i, j] \mid A \in V_N \cup V_T, \;\; st \in \mathcal{S}, \;\; 0 \leq i \leq j \; \right\}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^{c\prime}} = \{\vdash [-, st_0, 0, 0]\}$$

$$\mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^{c\prime}} = \left\{ \; [A, st, i, j], [a, j, j+1] \vdash [a, st', j, j+1] \mid \mathrm{shift}_{st'} \in \mathrm{acción}(st, a), \;\; A \in V \; \right\}$$

$$\mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}^{c\prime}} = \left\{ \begin{array}{l} [A_{r,m}, st^m, j_{m-1}, j_m], \dots, [A_{r,1} st^1, j_0, j_1], [A, st^0, i, j_0] \vdash [A_{r,0}, st, j_0, j_m] \mid \\ \exists [a, j_m, j_m+1] \in \mathcal{H}_{\mathrm{LR}^{c\prime}}, \;\; \mathrm{reduce}_r \in \mathrm{acción}(st^m, a), \;\; st^i \in \mathrm{revela}(st^{i+1}), \\ st \in \mathrm{ir\_a}(st^0, \mathrm{lhs}(r)), \;\; m = \mathrm{longitud}(\mathrm{rhs}(r)), \;\; A \in V \end{array} \right\}$$

$$\mathcal{D}_{\mathrm{LR}^{c\prime}} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^{c\prime}} \cup \mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^{c\prime}} \cup \mathcal{D}^{\mathrm{Reduce}}_{\mathrm{LR}^{c\prime}}$$

$$\mathcal{F}_{\mathrm{LR}^{c\prime}} = \left\{ \; [st_f, 0, n] \; \right\}$$

Para demostrar que $\mathbf{LR^c} \overset{\mathbf{ir}}{\Longrightarrow} \mathbf{LR^{c\prime}}$, definiremos la siguiente función

$$f([A, st, i, j]) = \{ \; [st, i, j] \; \}$$

de la cual se obtiene directamente que $\mathcal{I}_{\mathrm{LR}^c} = f(\mathcal{I}_{\mathrm{LR}^{c'}})$ y que $\Delta_{LR^c} = f(\Delta_{LR^{c'}})$. En consecuencia, $\mathbb{P}_{\mathrm{SLR}(1)} \xrightarrow{\mathbf{ir}} \mathbb{P}_{\mathrm{SLR}(1)'}$, con lo que hemos probado lo que pretendíamos.

Para demostrar que $\mathbf{LR^{c'}} \stackrel{\mathbf{ir}}{\Longrightarrow} \mathbf{LR^3}$, deberemos demostrar que para todo esquema de análisis $\mathbb{P}_{\mathrm{LR}^{c'}}$ y $\mathbb{P}_{\mathrm{LR}^3}$ se cumplen $\mathcal{I}_{\mathrm{LR}^{c'}} \subseteq \mathcal{I}_{\mathrm{LR}^3}$ y $\overset{*}{\vdash}_{LR^{c'}} \subseteq \overset{*}{\vdash}_{LR^3}$. Lo primero es trivial, puesto que $\mathcal{I}_{\mathrm{LR}^{c'}} = \mathcal{I}_{\mathrm{LR}^3}$. Para lo segundo debemos mostrar que $\mathcal{D}_{\mathrm{LR}^{c'}} \subseteq \overset{*}{\vdash} LR^3$. Los pasos Init son idénticos en ambos casos. Un paso Shift de $\mathbb{P}_{\mathrm{LR}^{c'}}$ se corresponde bien con un paso Shift o bien con un paso InitShift de $\mathbb{P}_{\mathrm{LR}^3}$. Un paso Reduce de $\mathbb{P}_{\mathrm{LR}^{c'}}$ es igual a la secuencia de derivaciones de $\mathbb{P}_{\mathrm{LR}^3}$ encabezada por un paso Sel, seguida de m pasos Red, uno por cada elemento del lado derecho de la producción que se está reduciendo, y finalmente un paso Head. □

C.5.1 Análisis de complejidad

Tomamos la longitud n de la cadena de entrada como parámetro de la complejidad puesto que tanto el tamaño de la gramática como el del autómata LR son fijos para una gramática dada. A partir del esquema de análisis sintáctico $\mathbf{LR^3}$ es fácil ver que la complejijdad del algoritmo con respecto a la cadena de entrada es $\mathcal{O}(n^3)$ puesto que los pasos deductivos que involucran un mayor número de posiciones de la cadena de entrada son los del conjunto $\mathcal{D}^{\mathrm{Red}}_{\mathrm{LR}^3}$, cada uno de los cuales relaciona las posiciones i, j y k.

Este resultado es equivalente al que obtendríamos si siguiesemos un método clásico de cálculo de complejidad, como el que se relata a continuación. Puesto que cada ítem posee dos posiciones de la cadena de entrada, habrá $\mathcal{O}(n^2)$ ítems . Cada paso deductivo ejecuta un número limitado de operaciones por cada ítem. A este respecto, el peor caso corresponde a los pasos $\mathcal{D}^{\mathrm{Red}}_{\mathrm{LR}^3}$, que pueden combinar $\mathcal{O}(n^2)$ ítems de la forma $[\nabla_{r,s}, st, k, j]$ con $\mathcal{O}(n)$ ítems de las forma $[A_{r,s}, st, i, k]$ y por consiguiente estos pasos deductivos presentan una complejidad $\mathcal{O}(n^3)$.

Al igual que se hacía en el caso del algoritmo de Earley original [69] podemos agrupar los ítems en conjuntos de ítems denominados *itemsets*. Existe un itemset por cada uno de los caracteres en la cadena de entrada[4]. En este caso, para la clase de las gramáticas con número limitado de ítems [5], en las cuales el número máximo de ítems que puede contener un itemset está acotado, se obtiene complejidad lineal, tanto espacial como temporal. El interés práctico de este resultado se debe a que esta clase de gramáticas incluye a la familia de las gramáticas LR y en consecuencia, es posible realizar el análisis sintáctico en tiempo lineal cuando la cadena de entrada es localmente determinista.

Si considerásemos el tamaño de la gramática como variable en el cálculo de la complejidad, los análisis efectuados por Johnson [88] con respecto al algoritmo de Tomita serían también aplicables al algoritmo propuesto.

C.6 Tabulación del autómata a pila LR

El esquema de análisis LR^3 corresponde a una interpretación en programación dinámica o tabulación de los algoritmos de análisis sintáctico LR(1) o LALR(1) utilizando un sistema de inferencia basado en ítems S^1 [52, pp. 173–175]. Para incorporar el algoritmo en la estructura común de análisis sintáctico propuesta por Lang en [107] es necesario transformarlo en un conjunto de transiciones de autómata a pila. Por [52] sabemos que utilizando ítems S^1, es decir,

[4] ítems con el cuarto componente igual a j pertenecen al itemset j.

[5] *bounded item grammars*, llamadas *bounded state grammars* en [69].

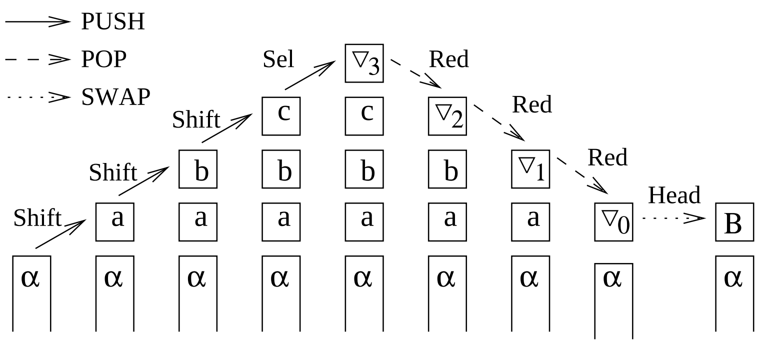

Figura C.5: Representación gráfica de la evolución de la pila en un algoritmo LR

ítems que contienen únicamente el elemento en la cima de la pila, podemos obtener una interpretación correcta en programación dinámica de los autómata débilmente predictivos, clase a la que pertenecen los autómatas a pila LR.

En la figura C.5 podemos observar cómo el algoritmo descrito en la sección C.5 trabaja sobre una pila. En esta figura, las cajas representan ítems. Para facilitar su comprensión, en cada caja hemos situado únicamente el primer elemento del ítem que representa. Por la misma razón los símbolos nabla poseen un solo índice, que indica su posición en la producción $B \to abc$. Los pasos deductivos Shift y Sel apilan un nuevo ítem en la cima de la pila. Los pasos Red desapilan los dos ítems de la cima y apilan uno en lo más alto de la pila. Los pasos Head simplemente reemplazan el ítem de la cima por otro ítem.

Las transiciones de la figura C.5 se corresponden exactamente con las transiciones de autómatas a pila descritas en [107]. En efecto, cada autómata a pila puede ser descrito utilizando las siguientes transiciones:

$$\begin{array}{lll}
\text{SWAP:} & (B \longmapsto C)(A) = C & \text{tal que } B = A \\
\text{PUSH:} & (B \longmapsto BC)(A) = C & \text{tal que } B = A \\
\text{POP:} & (DB \longmapsto C)(E, A) = C & \text{tal que } (D, B) = (E, A)
\end{array}$$

donde A, B, C, D y E son ítems y donde las pilas crecen hacia la derecha.

Esquema de compilación C.7 Si consideramos los pasos deductivos Head como transiciones SWAP, los pasos Init, InitShift, Shift y Sel como transiciones PUSH y los pasos Red como transiciones POP, obtenemos un esquema de compilación que, para una gramática independiente del contexto $\mathcal{G}$ y una estrategia de análisis sintáctico LR(1) o LALR(1), queda definido por el siguiente conjunto de reglas de compilación y el elemento final $[S, st_f]$.

$$\begin{array}{lll}
\textbf{[INIT]} & \$_0 \longmapsto \$_0[-, st_0] & \\
\textbf{[INITSHIFT]} & [A, st] \overset{a}{\longmapsto} [A, st]\ [A_{r,1}, st'] & A_{r,1} = a,\ \text{shift}_{st'} \in \text{acción}(st, a),\ A \in V \\
\textbf{[SHIFT]} & [A_{r,s}, st] \overset{a}{\longmapsto} [A_{r,s}, st]\ [A_{r,s+1}, st'] & A_{r,s+1} = a,\ \text{shift}_{st'} \in \text{acción}(st, a) \\
\textbf{[SEL]} & [A, st] \longmapsto [A_{r,n_r}, st]\ [\nabla_{r,n_r}, st] & \text{reduce}_r \in \text{acción}(st, \text{lookahead}),\ A \in V \\
\textbf{[RED]} & [A_{r,s}, st]\ [\nabla_{r,s}, st] \longmapsto [\nabla_{r,s-1}, st'] & st' \in \text{revela}(st) \\
\textbf{[HEAD]} & [\nabla_{r,0}, st] \longmapsto [A_{r,0}, st'] & st' \in \text{ir_a}(st, A_{r,0})
\end{array}$$

§

Esquema de análisis sintáctico C.8 La interpretación en programación dinámica del autómata a pila correspondiente a los algoritmos LR(1) y LALR(1) para una gramática independiente del contexto $\mathcal{G}$ y una cadena de entrada $a_1 \dots a_n$ queda definido por el el sistema de análisis $\mathbb{P}_{\mathrm{LR}^{\mathrm{S1}}}$ que se muestra a continuación:

$$\mathcal{I}_{\mathrm{LR}^{\mathrm{S1}}} = \left\{ \; [A, st, i, j] \cup [\nabla_{r,s}, st, i, j] \mid A \in V_N \cup V_T, \;\; st \in \mathcal{S}, \;\; 0 \leq i \leq j \; \right\}$$

$$\mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^{\mathrm{S1}}} = \left\{ \; \vdash [-, st_0, 0, 0] \; \right\}$$

$$\mathcal{D}^{\mathrm{InitShift}}_{\mathrm{LR}^{\mathrm{S1}}} = \left\{ \begin{array}{l} [A, st, i, j] \vdash \quad [A_{r,1}, st', j, j+1] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\mathrm{LR}^{\mathrm{S1}}}, \;\; A_{r,1} = a, \;\; \mathrm{shift}_{st'} \in \mathrm{acción}(st, a), \;\; A \in V \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^{\mathrm{S1}}} = \left\{ \begin{array}{l} [A_{r,s}, st, i, j] \vdash [A_{r,s+1}, st', i, j+1] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\mathrm{LR}^{\mathrm{S1}}}, \;\; A_{r,s+1} = a, \;\; \mathrm{shift}_{st'} \in \mathrm{acción}(st, a) \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Sel}}_{\mathrm{LR}^{\mathrm{S1}}} = \left\{ \begin{array}{l} [A, st, i, j] \vdash [\nabla_{r,n_r}, st, i, j+1] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\mathrm{LR}^{\mathrm{S1}}}, \;\; \mathrm{reduce}_r \in \mathrm{acción}(st, a), \;\; A \in V \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Red}}_{\mathrm{LR}^{\mathrm{S1}}} = \left\{ \begin{array}{l} [\nabla_{r,s}, st, i, k] \quad [A_{r,s}, st, k, j] \vdash [\nabla_{r,s-1}, st', i, j] \mid \\ st' \in \mathrm{revela}(st) \end{array} \right\}$$

$$\mathcal{D}^{\mathrm{Head}}_{\mathrm{LR}^{\mathrm{S1}}} = \left\{ \begin{array}{l} [\nabla_{r,0}, st, i, j] \vdash [A_{r,0}, st', i, j] \mid \\ st' \in \mathrm{ir\_a}(st, A_{r,0}) \end{array} \right\}$$

$$\mathcal{D}_{\mathrm{LR}^{\mathrm{S1}}} = \mathcal{D}^{\mathrm{Init}}_{\mathrm{LR}^{\mathrm{S1}}} \cup \mathcal{D}^{\mathrm{InitShift}}_{\mathrm{LR}^{\mathrm{S1}}} \cup \mathcal{D}^{\mathrm{Shift}}_{\mathrm{LR}^{\mathrm{S1}}} \cup \mathcal{D}^{\mathrm{Sel}}_{\mathrm{LR}^{\mathrm{S1}}} \cup \mathcal{D}^{\mathrm{Red}}_{\mathrm{LR}^{\mathrm{S1}}} \cup \mathcal{D}^{\mathrm{Head}}_{\mathrm{LR}^{\mathrm{S1}}}$$

$$\mathcal{F}_{\mathrm{LR}^{\mathrm{S1}}} = \left\{ \; [S, st_f, 0, n] \; \right\}$$

§

A continuación trataremos de mostrar mediante ejemplos el funcionamiento de un analizador sintáctico que implementa el esquema de análisis LR^{S1}. En primer lugar, veremos como la utilización de símbolos de preanálisis ayuda a mejorar la eficiencia evitando la exploración de caminos infructuosos. Posteriormente veremos cómo se tratan los ciclos y las producciones recursivas.

Ejemplo C.1 Tomemos la gramática independiente del contexto $\mathcal{G}_1$, simple pero de alto valor didáctico:

$$\begin{array}{ll} (0) & \Phi \rightarrow S \\ (1) & S \rightarrow Aa \\ (2) & S \rightarrow Bb \\ (3) & A \rightarrow cd \\ (4) & B \rightarrow cd \end{array}$$

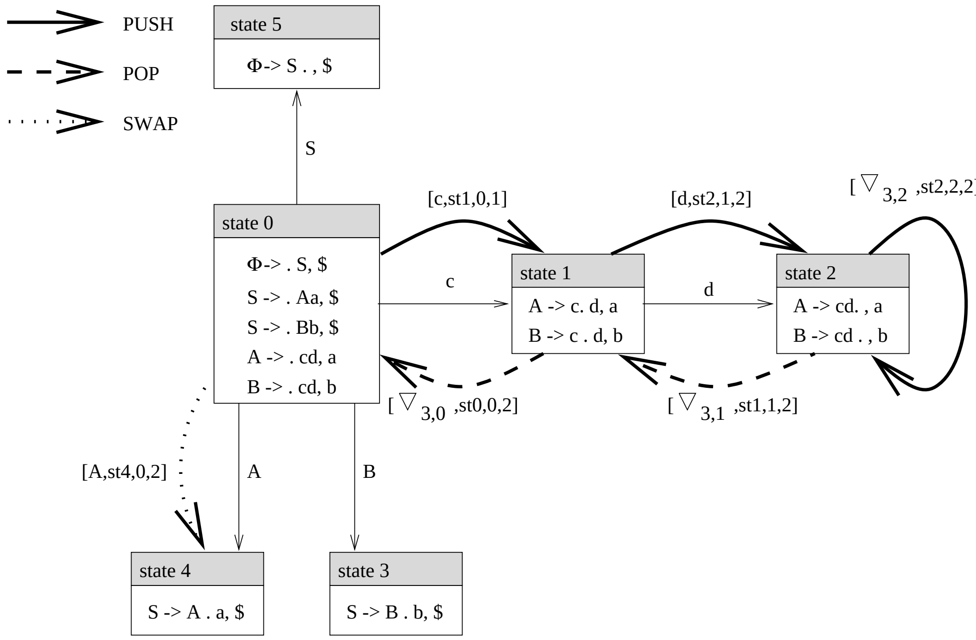

Figura C.6: Transiciones para la cadena de entrada cda en el autómata LALR(1) de la gramática $\mathcal{G}_1$

El lenguaje generado por $\mathcal{G}_1$ es el conjunto $\{cda, cdb\}$. En la figura C.6 mostramos el autómata LALR(1) para esta gramática y las transiciones correspondientes al análisis del prefijo cd de la cadena de entrada cda. El análisis comienza con el autómata en el estado 0. La primera acción a realizar es el desplazamiento del terminal c, aplicando para ello un paso InitShift. Como resultado, el ítem $[c, st1, 0, 1]$ es apilado y el nuevo estado del autómata LR es el 1. Una vez en este estado, sabemos que estamos intentando analizar la entrada mediante las producciones 3 y/o 4 (en un analizador Earley, esto correspondería a predecir las producciones 3 y 4). La siguiente acción a realizar en el desplazamiento del terminal d, para lo cual aplicamos un paso Shift, apilando el ítem $[d, st2, 1, 2]$ y convirtiendo al estado 2 en el nuevo estado actual del autómata LR. El estado 2 indica que tanto al producción 3 como la 4 reconocen el prefijo cd, pero el símbolo de preanálisis nos permite determinar que continuar el análisis por la producción 4 es infructuoso, puesto que no es compatible con el siguiente símbolo en la cadena de entrada. Un analizador sintáctico sin preanálisis se hubiese visto obligado a explorar las dos alternativas representadas por las producciones 3 y 4, para descubrir más tarde que sólo la producción 3 puede ser aplicada. La utilización de preanálisis incrementa el dominio determinista del analizador, permitiéndole obtener una mayor eficiencia.

En nuestro caso, el análisis continua mediante la reducción de la producción 3

1. Sel: PUSH $[\nabla_{3,2}, st2, 2, 2]$
2. Red: POP $[\nabla_{3,2}, st2, 2, 2]$ y $[d, st2, 1, 2]$ para dar $[\nabla_{3,1}, st1, 1, 2]$
3. Red: POP $[\nabla_{3,1}, st1, 1, 2]$ y $[c, st1, 0, 1]$ para dar $[\nabla_{3,0}, st0, 0, 2]$
4. Head: SWAP $[\nabla 3, 0, st0, 0, 2]$ para dar $[A, st4, 0, 2]$

¶

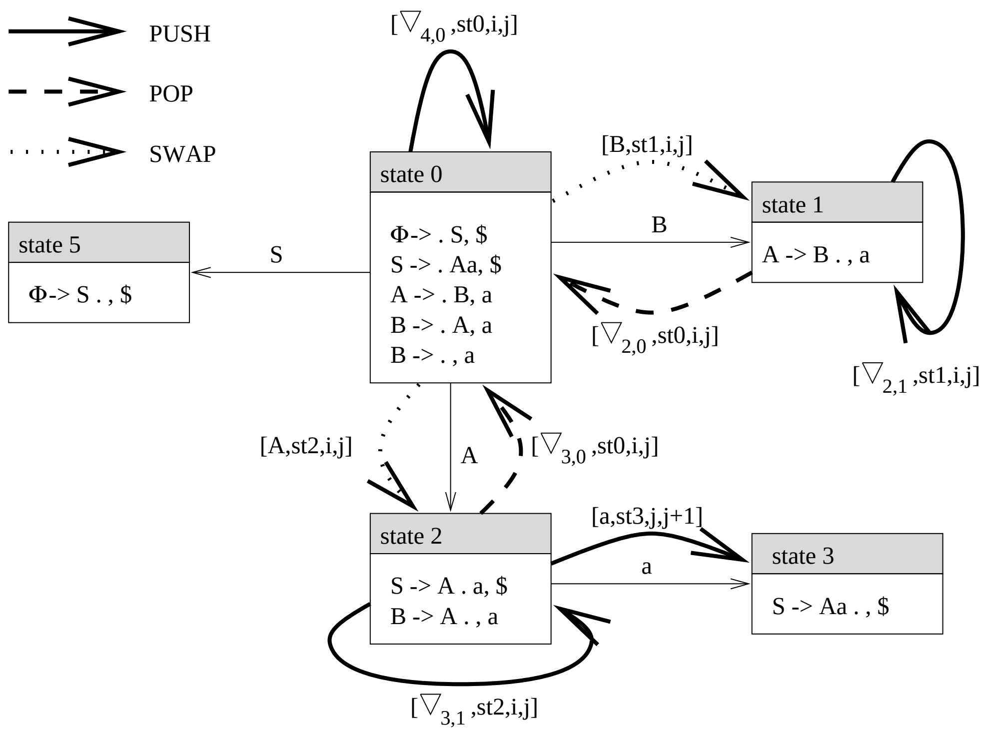

Figura C.7: Transiciones de un ciclo en el autómata LALR(1) de la gramática $\mathcal{G}_2$

Ejemplo C.2 Para mostrar el análisis de gramáticas cíclicas y con producciones recurrentes, utilizaremos la gramática $\mathcal{G}_2$, una vez más simple pero didáctica:

$$\begin{array}{ll} (0) & \Phi \rightarrow S \\ (1) & S \rightarrow Aa \\ (2) & A \rightarrow B \\ (3) & B \rightarrow A \\ (4) & B \rightarrow \varepsilon \end{array}$$

El lenguaje generado por $\mathcal{G}_2$ es $\{a\}$. En la figura C.7 podemos ver el autómata LALR(1) para esta gramática junto con las transiciones correspondientes a varios análisis cíclicos.

El análisis comienza en el estado 0. La primera acción a realizar es la reducción de la producción 4:

1. Sel: PUSH $[\nabla_{4,0}, st0, i, j]$
2. Head: SWAP $[\nabla_{4,0}, st0, i, j]$ para dar $[B, st1, i, j]$

El estado actual es el 1 y la siguiente acción a realizar la reducción de la producción 2:

1. Sel: PUSH $[\nabla_{2,1}, st1, i, j]$
2. Red: POP $[\nabla_{2,1}, st1, i, j]$ y $[B, st1, i, j]$ para dar $[\nabla_{2,0}, st0, i, j]$
3. Head: SWAP $[\nabla_{2,0}, st0, i, j]$ para dar $[A, st2, i, j]$

En este punto, el estado actual del autómata LALR(1) es el 2 y la acción a realizar es la reducción de la producción 3:

1. Sel: PUSH $[\nabla_{3,1}, st2, i, j]$

2. Red: POP $[\nabla_{3,1}, st2, i, j]$ y $[A, st2, i, j]$ para dar $[\nabla_{3,0}, st0, i, j]$

3. Head: SWAP $[\nabla_{3,0}, st0, i, j]$ para dar $[B, st1, i, j]$

El ítem resultante de la última transición, $[B, st1, i, j]$, ya había sido generado anteriormente y en consecuencia no es preciso volver a realizar las acciones derivadas a partir de él. Como resultado, todos los ítems generados por la aplicación de las producciones 2 y 3 con calculados una sola vez, en la primera iteración.

El análisis de la cadena de entrada continúa mediante la aplicación de un paso Shift en el estado 2, apilando el ítem $[a, st3, j, j+1]$. ¶

C.7 Análisis sintáctico LR Generalizado para Gramáticas de Cláusulas Definidas

La Gramáticas de Cláusulas Definidas (DCG) [143] son una extensión de las gramáticas independientes del contexto en las cuales se asocia un conjunto de atributos a los símbolos gramaticales, de tal modo que en vez de un conjunto de producciones sobre elementos de $(V_T \cup V_N)^*$ tenemos un conjunto de cláusulas sobre átomos lógicos. Frente a los enfoques que tratan de implantar analizadores LR para DCG mediante alteraciones en la estrategia de búsqueda de lenguajes lógicos [134, 196] o mediante transformaciones de la gramática [162], presentamos aquí un enfoque basado en la extensión de un algoritmo LR generalizado para gramáticas independientes del contexto.

Formalmente, una DCG está definida por la tupla $(V_N, V_T, \mathcal{P}, S, \mathcal{V}, F)$, donde $\mathcal{V}$ es un conjunto finito de variables, F es un conjunto finito de funtores y $\mathcal{P}$ es un conjunto de cláusulas definidas de la forma

$$A_{r,0}(t^1_{r,0}, \ldots, t^{m_0}_{r,0}) \rightarrow A_{r,1}(t^1_{r,1}, \ldots, t^{m_1}_{r,1}) \ \ldots \ A_{r,n_r}(t^1_{r,n_r}, \ldots, t^{m_{n_r}}_{r,n_r})$$

donde $A_{r,0} \in V_N$, $A_{r,i} \in (V_N \cup V_T)^*$ para $1 \geq i \geq n_r$ y $t^{m_j}_{r,i}$ son términos definidos inductivamente como sigue: un término es bien un funtor constante de aridad 0, bien una variable o bien un término compuesto $f(t_1, \ldots, t_l)$, donde f es un funtor de aridad l y $t_1, \ldots, t_l$ son términos. Si $A_{r,i} \in V_T$, los términos $t^1_{r,i}, \ldots, t^{m_i}_{r,i}$ se consideran relacionados con la entrada léxica $A_{r,i}$, pudiendo los valores variables ser obtenidos a partir de los valores almacenados en dicha entrada. Para cada cláusula γ_r definimos el vector $\overrightarrow{T_r}$ que contiene todas las variables que aparecen en γ_r. Cuando la mención explícita de los términos lógicos no sea precisa, denotaremos una cláusula o sus elementos mediante el símbolo de la gramática independiente del contexto que le corresponde escrito en negrita. Así, la cláusula anterior se escribirá en forma abreviada como $\mathbf{A_{r,0}} \rightarrow \mathbf{A_{r,1}} \ldots \mathbf{A_{r,n_r}}$.

Una diferencia fundamental desde el punto de visto operativo entre las gramáticas independientes del contexto y las gramáticas de cláusulas definidas es que mientras las primeras poseen un número finito de símbolos gramaticales, en las últimas el número de elementos gramaticales es potencialmente infinito ya que no existe un límite para el tamaño de los términos lógicos. En consecuencia, no se puede garantizar la terminación de las operaciones para la construcción de las tablas de análisis LR en el caso de DCG [120].

Una posible solución a este problema pasa por el uso de restrictores positivos [185] o negativos [202] con el fin de definir un número finito de clases de equivalencias en las cuales poder ordenar el número infinito de no-terminales[6]. Dichos restrictores deben aplicarse en todas aquellas

[6] Aunque los restrictores fueron originalmente definidos para ser aplicados a estructuras de rasgos [40], en general pueden ser aplicados a todos aquellos formalismos basados en restricciones [186], incluyendo las gramáticas de cláusulas definidas.

operaciones involucradas en la obtención de información precompilada a partir de la gramática: funciones primero y siguiente, cerradura de los estados del autómata LR y construcción de las tablas de acciones e ir_a.

Generalmente existen varios restrictores para gramática de cláusulas definidas pero sin embargo no puede asegurarse la terminación de las operaciones mencionadas anteriormente para cada uno de ellos. Es más, no existe ningún método automático para la selección del mejor restrictor puesto que este depende de la cantidad de información gramatical que deba ser preservada. En la práctica, con el fin e conseguir un equilibrio entre la bonanza del restrictor y la garantía de terminación, durante la fase de compilación se suele considerar únicamente la gramática independiente del contexto subyacente [226].

En el proceso de binarización de las cláusulas definidas, debemos tener en cuenta la transmisión de la información almacenada en los términos lógicos, de tal modo que una cláusula definida $\mathbf{A_{r,0}} \rightarrow \mathbf{A_{r,1}} \ldots \mathbf{A_{r,n_r}}$ se transforma en el siguiente conjunto de $n_r + 1$ cláusulas:

$$\begin{array}{l}
\mathbf{A_{r,0}} \rightarrow \nabla_{r,0}(\overrightarrow{T_r}) \\
\nabla_{r,0}(\overrightarrow{T_r}) \rightarrow \mathbf{A_{r,1}}\ \nabla_{r,1}(\overrightarrow{T_r}) \\
\vdots \\
\nabla_{r,n_r-1}(\overrightarrow{T_r}) \rightarrow \mathbf{A_{r,n_r}}\ \nabla_{r,n_r}(\overrightarrow{T_r}) \\
\nabla_{r,n_r}(\overrightarrow{T_r}) \rightarrow \varepsilon
\end{array}$$

Como mecanismo operacional utilizaremos los *autómatas lógicos a pila*[56, 52], esencialmente autómatas a pila que almacenan en la pila predicados lógicos en vez de simples elementeos gramaticales. A continuación definimos el esquema de compilación.

Esquema de compilación C.9 El esquema de compilación para una gramática de cláusulas definidas $\mathcal{G}$ y una estrategia de análisis sintáctico LR(1) o LALR(1) queda definido por el siguiente conjunto de producciones y el elemento final $[\mathbf{S}, st_f]$.

[INIT]	$\$_0 \longmapsto \$_0[-, st_0]$	
[INITSHIFT]	$[\mathbf{A}, st] \overset{a}{\longmapsto} [\mathbf{A}, st]\ [\mathbf{A_{r,1}}, st']$	$\mathbf{A_{r,1}} = a(t^1_{r,1}, \ldots, t^{m_1}_{r,1})$ $\text{shift}_{st'} \in \text{acción}(st, a)$
[SHIFT]	$[\mathbf{A_{r,s}}, st] \overset{a}{\longmapsto} [\mathbf{A_{r,s}}, st]\ [\mathbf{A_{r,s+1}}, st']$	$\mathbf{A_{r,s+1}} = a(t^1_{r,1}, \ldots, t^{m_1}_{r,1})$ $\text{shift}_{st'} \in \text{acción}(st, a)$
[SEL]	$[\mathbf{A}, st] \longmapsto [\mathbf{A_{r,n_r}}, st]\ [\nabla_{r,n_r}(\overrightarrow{T_r}), st]$	$\text{reduce}_r \in \text{acción}(st, \text{lookahead})$
[RED]	$[\mathbf{A_{r,s}}, st]\ [\nabla_{r,s}(\overrightarrow{T_r}), st] \longmapsto [\nabla_{r,s-1}(\overrightarrow{T_r}), st']$	$st' \in \text{revela}(st)$
[HEAD]	$[\nabla_{r,0}(\overrightarrow{T_r}), st] \longmapsto [\mathbf{A_{r,0}}, st']$	$st' \in \text{ir\_a}(st, A_{r,0})$

§

Esquema de análisis sintáctico C.10 La interpretación en programación dinámica del autómata lógico a pila correspondiente a los algoritmos LR(1) y LALR(1) para una gramática de cláusulas definidas $\mathcal{G}$ y una cadena de entrada $a_1 \dots a_n$ queda definido por el el sistema de análisis $\mathbb{P}_{\text{LR}^{\text{S1}}(\text{DCG})}$ que se muestra a continuación:

$$\mathcal{I}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \ [\mathbf{A}, st, i, j] \cup [\nabla_{r,s}(\overrightarrow{T_r}), st, i, j] \ \right\}$$

$$\mathcal{D}^{\text{Init}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \{\vdash [-, st_0, 0, 0]\}$$

$$\mathcal{D}^{\text{InitShift}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \begin{array}{l} [\mathbf{A}, st, i, j] \vdash [\mathbf{A_{r,1}}, st', j, j+1] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\text{LR}^{\text{S1}}(\text{DCG})}, \ \mathbf{A_{r,1}} = a(t^1_{r,1}, \dots, t^{m_1}_{r,1}), \ \text{shift}_{st'} \in \text{acción}(st, a) \end{array} \right\}$$

$$\mathcal{D}^{\text{Shift}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \begin{array}{l} [\mathbf{A_{r,s}}, st, i, j] \vdash [\mathbf{A_{r,s+1}}, st', i, j+1] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\text{LR}^{\text{S1}}(\text{DCG})}, \ \mathbf{A_{r,s+1}} = a(t^1_{r,s+1}, \dots, t^{m_{s+1}}_{r,s+1}), \\ \text{shift}_{st'} \in \text{acción}(st, a) \end{array} \right\}$$

$$\mathcal{D}^{\text{Sel}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \begin{array}{l} [\mathbf{A_{r,n_r}}, st, i, j] \vdash [\nabla_{r,n_r}(\overrightarrow{T_r}), st, j, j] \mid \\ \exists [a, j, j+1] \in \mathcal{H}_{\text{LR}^{\text{S1}}(\text{DCG})}, \ \text{reduce}_r \in \text{acción}(st, a) \end{array} \right\}$$

$$\mathcal{D}^{\text{Red}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \ [\mathbf{A_{r,s}}, st, i, k] \ \ [\nabla_{r,s}(\overrightarrow{T_r}), st, k, j] \ \ \vdash [\nabla_{r,s-1}(\overrightarrow{T_r}), st', i, j] \mid st' \in \text{revela}(st) \ \right\}$$

$$\mathcal{D}^{\text{Head}}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \ [\nabla_{r,0}(\overrightarrow{T_r}), st, i, j] \vdash [\mathbf{A_{r,0}}, st', i, j] \mid st' \in \text{ir\_a}(st, A_{r,0}) \ \right\}$$

$$\mathcal{D}_{\text{LR}^{\text{S1}}(\text{DCG})} = \mathcal{D}^{\text{Init}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \mathcal{D}^{\text{InitShift}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \mathcal{D}^{\text{Shift}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \mathcal{D}^{\text{Sel}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \mathcal{D}^{\text{Red}}_{\text{LR}^{\text{S1}}(\text{DCG})} \cup \mathcal{D}^{\text{Head}}_{\text{LR}^{\text{S1}}(\text{DCG})}$$

$$\mathcal{F}_{\text{LR}^{\text{S1}}(\text{DCG})} = \left\{ \ [\mathbf{S}, st_f, 0, n] \ \right\}$$

§

C.8 Análisis sintáctico LR Generalizado para Gramáticas Lineales de Índices

Las técnicas creadas para el desarrollo de analizadores sintácticos de tipo LR generalizado para DCG pueden aplicarse al caso de las gramáticas lineales de índices, puesto que estas pueden verse como un caso particular de DCG en el que a cada no terminal de una CFG se le añade un único atributo en forma de pila y se restringe la posibilidad de copia de su contenido (véase la sección 8.1 para más detalles). A continuación mostramos cómo especializar las técnicas desarrolladas en la sección precedente para la construcción de analizadores sintácticos LR para gramáticas lineales de índices.

El primer paso para realizar un analizador de tipo LR consiste en construir el autómata LR, un autómata finito en el cual se almacena cierta información obtenida mediante un análisis estático de la gramática. Al igual que en el caso de las gramáticas de cláusulas definidas, existen dos opciones técnicas para construir dicho autómata:

1. Considerar únicamente el esqueleto independiente del contexto de la gramática lineal de índices.

2. Incluir información relativa a la evolución de las pilas de índices en el autómata LR.

La primera opción es más simple pero menos efectiva puesto que no se considera toda la información disponible en la gramática durante la construcción del autómata LR y por consiguiente habrá más conflictos en la fase de ejecución del algoritmo de análisis LR. Estos conflictos se hubiesen podido evitar de haber considerado la información acerca de la constitución de las pilas durante la construcción del autómata LR. Para ello sería necesario sustituir las funciones *primero* y *siguiente* [6] por las que se muestran a continuación.

Definición C.3 *Un elemento* $a \in V_T \cup \{\epsilon\}$ *pertenece a* $\text{primero}(\Gamma)$*, donde* $\Gamma \in V_T \cup V_N[V_I{}^*]$*, si se cumple alguna de las condiciones siguientes:*

- $\Gamma = a$
- $\Gamma' \to \epsilon \in P$*, existe* $\sigma = \text{mgu}(\Gamma, \Gamma')$ *y* $a = \epsilon$
- $\Gamma' \to \Gamma_1 \ldots \Gamma_i \ldots \Gamma_n \in P$*, existe* $\sigma = \text{mgu}(\Gamma, \Gamma')$ *y* $a \in \text{primero}(\Gamma_i \sigma)$ *y* $\forall_{j=1}^{i-1} \epsilon \in \text{primero}(\Gamma_j \sigma)$

donde mgu se refiere al unificador más general.

A partir de la definición anterior obtenemos el siguiente método de cálculo de la función $\text{primero}(\Gamma)$:

1. Si $\Gamma = a \in V_T$ entonces $\text{primero}(a) = \{a\}$.

2. Si $\Gamma \to \varepsilon$ entonces $\varepsilon \in \text{primero}(\Gamma)$.

3. Si $\Gamma' \to \Gamma_1 \ldots \Gamma_i \ldots \Gamma_n$ y existe $\sigma = \text{mgu}(\Gamma, \Gamma')$, entonces $\text{primero}(\Gamma_1 \sigma) \subseteq \text{primero}(\Gamma)$ y $\forall_{j=1}^{j=i-1} \Gamma_j$ tal que $\varepsilon \in \text{primero}(\Gamma_j)$ tenemos que $\text{primero}(\Gamma_{j+1}\sigma) \subseteq \text{primero}(\Gamma)$

Definición C.4 *Un elemento* $a \in V_T \cup \{\$\}$ *pertenece a* $\text{siguiente}(\Gamma)$*, donde* $\Gamma \in V_T \cup V_N[V_I{}^*]$ *y \$ es un carácter especial que no pertenece a* V_T *que indica que se ha alcanzado el final de la cadena de entrada, si se cumple alguna de las siguientes condiciones:*

- $a = \$$ *y* $\Gamma = S[\,]$
- $\Gamma'' \to \Upsilon_1 \Gamma' \Upsilon_2$*, existe* $\sigma = \text{mgu}(\Gamma, \Gamma')$ *y* $a \in \text{primero}(\Upsilon_2 \sigma) - \{\epsilon\}$
- $\Gamma'' \to \Upsilon_1 \Gamma' \Upsilon_2$*, existe* $\sigma = \text{mgu}(\Gamma, \Gamma')$ *y* $\epsilon \in \text{primero}(\Upsilon_2 \sigma)$ *y* $a \in \text{siguiente}(\Gamma'' \sigma)$

A partir de la definición anterior obtenemos el siguiente método de cálculo de la función $\text{siguiente}(\Gamma)$:

1. Si $\Gamma = S[\,]$ entonces $\$ \in \text{siguiente}(\Gamma)$.

2. Si $\Gamma'' \to \Upsilon_1 \Gamma' \Upsilon_2$ y $\sigma = mgu(\Gamma, \Gamma')$ entonces $\text{primero}(\Upsilon_2 \sigma) \subseteq \text{siguiente}(\Gamma)$.

3. Si $\Gamma'' \to \Upsilon_1 \Gamma' \Upsilon_2$ y $\sigma = mgu(\Gamma, \Gamma')$ y $\varepsilon \in \text{primero}(\Upsilon_2 \sigma)$ entonces $\text{siguiente}(\Gamma'' \sigma) \subseteq \text{siguiente}(\Gamma)$.

El cierre de los estados del autómata se efectúa como en el algoritmo clásico [6] con la salvedad de que las funciones *primero* y *siguiente* son reemplazadas por las que acabamos de definir. Puesto que la operación de cierre es esencialmente predictiva pueden surgir problemas de no terminación si el proceso de unificación introduce un número infinito de nuevos elementos a considerar. Este es un problema bien conocido en el ámbito de la programación lógica y para solucionarlo se han propuesto varias alternativas. De entre todas ellas, la mejor adaptada a las características de LIG consiste en aplicar la noción de *restrictor* propuesta por Shieber en [185]. En efecto, puesto que únicamente es necesario considerar un número acotado de elementos de la cima de cada pila de índices para determinar si una producción puede ser o no utilizada en una derivación, podemos restringir el alcance de la unificación a los primeros elementos de la pila de índices y considerar el resto como una variable lógica. En la tabla C.1 se define la operación de unificación para el caso de que sólo se consulte el primer elemento de la cima de las pilas LIG. Para todos aquellos casos no mostrados en el tabla, la unificación fracasa. En la tabla C.2 se define la operación de subsumción para el mismo caso.

$\text{mgu}(A[\,], A[\,]) = A[\,]$
$\text{mgu}(A[\,], A[\circ\circ]) = A[\,]$
$\text{mgu}(A[\circ\circ], A[\,]) = A[\,]$
$\text{mgu}(A[\circ\circ], A[\circ\circ]) = A[\circ\circ]$
$\text{mgu}(A[\circ\circ], A[\circ\circ\gamma]) = A[\circ\circ\gamma]$
$\text{mgu}(A[\circ\circ\gamma], A[\circ\circ]) = A[\circ\circ\gamma]$
$\text{mgu}(A[\circ\circ\gamma_1], A[\circ\circ\gamma_2]) = A[\circ\circ\gamma_1]$ si y sólo si $\gamma_1 = \gamma_2$

Tabla C.1: Unificación mediantes restrictores de símbolos LIG

$A[\,] \preceq A[\,]$
$A[\gamma] \preceq A[\gamma]$
$A[\circ\circ] \preceq A[\,]$
$A[\circ\circ] \preceq A[\circ\circ]$
$A[\circ\circ] \preceq A[\circ\circ\gamma]$
$A[\circ\circ\gamma] \preceq A[\gamma]$
$A[\circ\circ\gamma] \preceq A[\circ\circ\gamma]$
$A[\circ\circ\gamma] \preceq A[\circ\circ\gamma'\gamma]$

Tabla C.2: Subsumción mediante restrictores de símbolos LIG

Mediante la utilización de esta técnica sólo se pueden generar un número finito de elementos durante el proceso de construcción de los estados del autómata LR. Una vez que el autómata ha sido construido, se construyen las tablas de *acción* e *ir_a* como para los algoritmos LR clásicos, teniendo en cuenta que las transiciones entre estados están etiquetadas no ya por un símbolo terminal o no-terminal sino por un terminal o por un elemento del conjunto

$V_N[\,] \cup V_N[\circ\circ] \cup V_N[\circ\circ V_I]$. Dependiendo del tratamiento del símbolo de preanálisis obtendremos un autómata finito LR(1) o LALR(1).

A continuación se detalla el esquema de compilación que dada una gramática lineal de índices genera el conjunto de transiciones del autómata lógico a pila que aplica el autómata finito LR previamente construido junto con un mecanismo de desplazamiento y reducción para analizar sintácticamente una cadena de entrada de acuerdo con la gramática. Los elementos de la pila del autómata son pares $\langle A, st\rangle$, donde A es un elemento de la gramática y st es un estado del autómata LR. En este esquema se ha aplicado una binarización implícita de las producciones de la gramática de tal modo que una producción

$$A_{r,0}[\circ\circ\gamma] \to A_{r,1}[\,] \dots A_{r,l}[\circ\circ\gamma'] \dots A_{r,n_r}[\,]$$

se ha descompuesto en las siguientes $n_r + 1$ producciones

$$\begin{aligned}
&A_{r,0}[\circ\circ\gamma] \to \nabla_{r,0}[\circ\circ\gamma] \\
&\nabla_{r,0}[\circ\circ\gamma] \to A_{r,1}[\,]\ \nabla_{r,1}[\circ\circ\gamma] \\
&\vdots \\
&\nabla_{r,l-1}[\circ\circ\gamma] \to A_{r,l}[\circ\circ\gamma']\ \nabla_{r,l}[\,] \\
&\nabla_{r,l}[\circ\circ] \to A_{r,l+1}[\,]\ \nabla_{r,l+1}[\circ\circ] \\
&\vdots \\
&\nabla_{r,n_r-1}[\circ\circ] \to A_{r,n_r}[\,]\ \nabla_{r,n_r}[\circ\circ] \\
&\nabla_{r,n_r}[\,] \to \epsilon
\end{aligned}$$

donde la pila de índices asociada los $\nabla_{r,i}$ con $i \in [l \dots n_r]$ está vacía al ser heredada de $\nabla_{r,n_r}[\,]$, puesto que el algoritmo LR reduce las producciones de derecha a izquierda.

Esquema de compilación C.11 El esquema de compilación para una gramática lineal de índices $\mathcal{G}$ y una estrategia de análisis sintáctico LR(1) o LALR(1) queda definido por el siguiente conjunto de reglas de compilación y el elemento final $\langle S, st_f\rangle$.

[INIT]	$\langle \$_0[\circ\circ], -\rangle \longmapsto \langle \$_0[\circ\circ], -\rangle\ \langle -, st_0\rangle$	
[SHIFT]	$\langle A[\circ\circ], st\rangle \overset{a}{\longmapsto} \langle A[\circ\circ], st\rangle\ \langle A_{r,1}[\,], st'\rangle$	$A_{r,1} = a,\ \text{shift}_{st'} \in \text{acción}(st, a)$
[SEL]	$\langle A_{r,n_r}[\circ\circ], st\rangle \longmapsto \langle A_{r,n_r}[\circ\circ], st\rangle\ \langle \nabla_{r,n_r}[\,], st\rangle$	$\text{reduce}_r \in \text{acción}(st, \text{lookahead})$
[RED]	$\langle A_{r,s}[\,], st\rangle\langle \nabla_{r,s}[\circ\circ], st\rangle \longmapsto \langle \nabla_{r,s-1}[\circ\circ], st'\rangle$	$st \in \text{ir\_a}(st', A_{r,s})$
[SRED]	$\langle A_{r,s}[\circ\circ\gamma], st\rangle\langle \nabla_{r,s}[\,], st\rangle \longmapsto \langle \nabla_{r,s-1}[\circ\circ\gamma'], st'\rangle$	$st \in \text{ir\_a}(st', A_{r,s})$
[HEAD]	$\langle \nabla_{r,0}[\circ\circ], st\rangle \longmapsto \langle A_{r,0}[\circ\circ], st'\rangle$	$st' \in \text{ir\_a}(st, A_{r,0})$

donde, si $A_{r,s}$ es el hijo dependiente de la producción r se aplica **[SRED]** en el proceso de reducción, en otro caso se aplica **[RED]**. El estado inicial del autómata LR es st_0 y el final es st_f. §

Las transiciones del esquema de compilación C.11 son un subconjunto de las transiciones de los autómatas lógicos a pila restringidos al caso de LIG que incorporan estrategias $*$-ascendentes [14]. Puesto que la estrategia LR es completamente ascendente al no realizar la fase de llamada ninguna predicción sobre la parte independiente del contexto, podemos utilizar la técnica de tabulación para la estrategia ascendente-ascendente propuesta en la sección 8.5.2, obteniendo la interpretación tabular del autómata que se muestra en el esquema de compilación C.12. La complejidad temporal con respecto a la longitud de la cadena de entrada es $\mathcal{O}(n^6)$ y la complejidad espacial es $\mathcal{O}(n^4)$.

Esquema de análisis sintáctico C.12 La interpretación en programación dinámica del autómata lógico a pila correspondiente a los algoritmos LR(1) y LALR(1) para una gramática lineal de índices $\mathcal{G}$ y una cadena de entrada $a_1 \dots a_n$ queda definido por el el sistema de análisis $\mathbb{P}_{\mathrm{LR}^{S1}(\mathrm{LIG})}$ que se muestra a continuación:

$$\mathcal{I}_{\mathrm{LR}} = \left\{ \begin{array}{l} [i, A, st_1, j, \gamma_1 \mid p, B, st_2, q] \mid \\ A, B \in V_N,\ \ \gamma \in V_I,\ \ st_1, st_2 \in \mathcal{A},\ \ 0 \leq i \leq j,\ \ (p,q) \leq (i,j) \end{array} \right\}$$

$$\mathcal{D}_{\mathrm{LR}}^{\mathrm{Init}} = \frac{}{[0, -, st_0, 0, - \mid -, -, -, -]}$$

$$\mathcal{D}_{\mathrm{LR}}^{\mathrm{Shift}} = \frac{[i, A, st_1, j, \gamma \mid p, B, st_2, q]}{[j, A_{r,1}, st_3, j+1, - \mid -, -, -, -]} \quad \begin{array}{l} \exists [a, j, j+1] \in \mathcal{H}_{\mathrm{LR}},\ \ A_{r,1} = a, \\ \mathrm{shift}_{st_3} \in \mathrm{acción}(st_1, a),\ \ A \in V \end{array}$$

$$\mathcal{D}_{\mathrm{LR}}^{\mathrm{Sel}} = \frac{[i, A_{r,n_r}, st_1, j, \gamma \mid p, B, st_2, q]}{[j, \nabla_{r,n_r}, st_1, j, - \mid -, -, -, -]} \quad \begin{array}{l} \exists [a, j, j+1] \in \mathcal{H}_{\mathrm{LR}}, \\ \mathrm{reduce}_r \in \mathrm{acción}(st_1, a_j), \end{array}$$

$$\mathcal{D}_{\mathrm{LR}}^{\mathrm{Red}} = \frac{\begin{array}{c} [k, \nabla_{r,s}, st_1, k, j, \gamma \mid p, B, st_2, q], \\ [A_{r,s}, st_1, i, k, - \mid -, -, -, -] \end{array}}{[i, \nabla_{r,s-1}, st_3, j, \gamma \mid p, B, st_2, q]} \quad \begin{array}{l} \boldsymbol{A}_{r,0} \to \Upsilon_1\ A_{r,s}[\,]\ \Upsilon_2, \\ st_3 \in \mathrm{revela}(st_1) \end{array}$$

$$\mathcal{D}_{\mathrm{LR}}^{\mathrm{SRed}_1} = \frac{\begin{array}{c} [k, \nabla_{r,s}, st_1, j, -, \mid -, -, -, -], \\ [i, A_{r,s}, st_1, k, \gamma \mid p, B, st_2, q] \end{array}}{[i, \nabla_{r,s-1}, st_3, j, \gamma \mid p, B, st_2, q]} \quad \begin{array}{l} A_{r,0}[\circ\circ] \to \Upsilon_1\ A_{r,s}[\circ\circ]\ \Upsilon_2, \\ st_1 \in \mathrm{ir\_a}(st_3, A_{r,s}) \end{array}$$

$$\mathcal{D}_{\mathrm{LR}}^{\mathrm{SRed}_2} = \frac{\begin{array}{c} [k, \nabla_{r,s}, st_1, j, -, \mid -, -, -, -], \\ [i, A_{r,s}, st_1, k, \gamma' \mid p, B, st_2, q] \end{array}}{[i, \nabla_{r,s-1}, st_3, j, \gamma \mid i, A_{r,s}, st_1, k]} \quad \begin{array}{l} A_{r,0}[\circ\circ\gamma] \to \Upsilon_1\ A_{r,s}[\circ\circ]\ \Upsilon_2, \\ st_1 \in \mathrm{ir\_a}(st_3, A_{r,s}) \end{array}$$

$$\mathcal{D}_{\mathrm{LR}}^{\mathrm{Red}_2} = \frac{\begin{array}{c} [k, \nabla_{r,s}, st_1, j, - \mid -, -, -, -], \\ [i, A_{r,s}, st_1, k, \gamma' \mid p, B, st_2, q], \\ [p, B, st_2, q, \gamma'' \mid p', C, st_3, q'] \end{array}}{[i, \nabla_{r,s-1}, st_4, j, \gamma'' \mid p', C, st_3, q']} \quad \begin{array}{l} A_{r,0}[\circ\circ] \to \Upsilon_1\ A_{r,s}[\circ\circ\gamma']\ \Upsilon_2, \\ st_4 \in \mathrm{revela}(st_1) \end{array}$$

$$\mathcal{D}_{\mathrm{LR}}^{\mathrm{Head}} = \frac{[i, \nabla_{r,0}, st_1, j, \gamma, \mid p, B, st_2, q]}{[i, A_{r,0}, st_3, j, \gamma \mid p, B, st_2, q]} \quad st_1 \in \mathrm{revela}(st_3)$$

$$\mathcal{D}_{\mathrm{LR}} = \mathcal{D}_{\mathrm{LR}}^{\mathrm{Init}} \cup \mathcal{D}_{\mathrm{LR}}^{\mathrm{Shift}} \cup \mathcal{D}_{\mathrm{LR}}^{\mathrm{Sel}} \cup \mathcal{D}_{\mathrm{LR}}^{\mathrm{Red}} \cup \mathcal{D}_{\mathrm{LR}}^{\mathrm{SRed}_1} \cup \mathcal{D}_{\mathrm{LR}}^{\mathrm{SRed}_2} \cup \mathcal{D}_{\mathrm{LR}}^{\mathrm{SRed}_3} \cup \mathcal{D}_{\mathrm{LR}}^{\mathrm{Head}}$$

$$\mathcal{F}_{\mathrm{LR}} = \{ \ [0, S, st_f, -, n \mid -, -, -, -] \ \}$$

donde $\mathcal{A}$ es el conjunto de estados del autómata LR, con st_0 su estado inicial y st_f su estado final. §

C.9 Conclusiones

Hemos mostrado como el algoritmo de Earley es un punto de arranque adecuado para el diseño de otros algoritmos de análisis sintáctico más complejos. En esta dirección, hemos derivado en sucesivas etapas un analizador LR generalizado capaz de analizar gramáticas independientes del contexto sin restricciones. Cada algoritmo intermedio ha sido descrito utilizando un esquema de análisis y hemos pasado de uno a otro aplicando transformaciones que podemos calificar de sencillas. Como resultado hemos obtenido un algoritmo en programación dinámica que se integra perfectamente en la estructura general de análisis propuesta por Lang, mostrando una complejidad con respecto a la cadena de entrada del orden de $\mathcal{O}(n^3)$ en el peor caso, aunque puede llegar a ser lineal en ciertos casos, tal y como sucede en el algoritmo de Earley.

La técnica propuesta ha sido también aplicada a formalismos gramaticales que aun no siendo independientes del contexto poseen un esqueleto independiente del contexto. Es el caso de las gramáticas de cláusulas definidas y de las gramáticas lineales de índices, ya que se ha mostrado que existe una extensión directa de las técnicas de análisis independientes del contexto que permite la evaluación de cláusulas de Horn [52]. Vilares Ferro y Alonso Pardo describen en [222, 221] la implementación de un analizador de gramáticas de cláusulas definidas basada en la especificación del algoritmo LALR(1) generalizado mostrada en este capítulo. Vilares Ferro et al. muestran en [223] una extensión de dicho algoritmo que permite representar de forma finita ciertas clases de términos lógicos infinitos, ampliando de esta manera el dominio de aplicación.

En los analizadores de lenguajes naturales resulta de gran interés disponer de la posibilidad de revisar dinámicamente la corrección sintáctica de un texto cuando ha sido editado sin necesidad de rehacer totalmente el análisis sintáctico sino cambiando únicamente las partes de las estructuras de análisis que se vean afectadas por el cambio. En este contexto, el algoritmo presentado aquí puede ser extendido con el fin de obtener un analizador sintáctico totalmente incremental, esto es, un analizador sintáctico que recupera partes del análisis anterior siempre que se realiza un nuevo análisis. Mediante el concepto de totalmente incremental se quiere indicar que se permiten modificaciones en cualquier punto de la cadena de entrada [232]. En [219, 227, 220] se describe un analizador sintáctico incremental basado en nuestra especificación del algoritmo LALR(1) para gramáticas independientes del contexto mientras que en [7] se analiza la integración de dicho analizador incremental en el entorno Galena [224] de desarrollo de herramientas para lenguaje natural, incluyendo la interacción con los etiquetadores diseñados en dicho enotrno [76].

Finalmente, señalar que en ciertos casos es posible aumentar la eficiencia del algoritmo obtenido aplicando técnicas para la compartición de ítems que se corresponden con el análisis de un mismo sufijo común a varias producciones gramaticales [122, 121]. Nederhof y Satta han estudiado esta vía en [130] para el caso de analizadores tabulares LR. El algoritmo resultante es tan complejo, ya que además del mecanismo de tabulación requiere la realización de un transformación a la gramática y de la aplicación de una función de filtrado, que el valor práctico de tales optimizaciones depende en gran medida de las características propias de las gramáticas utilizadas.