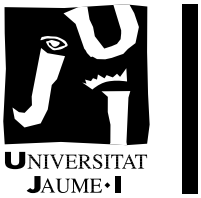


4º Ingeniería Informática
II26 Procesadores de lenguaje
Análisis ascendente

Esquema del tema

1. Introducción

Hasta ahora hemos visto cómo realizar el análisis descendente, construyendo el árbol de arriba abajo. Para ello partíamos del símbolo inicial de la gramática e íbamos buscando una derivación que justificara la entrada. El mecanismo básico era predecir cuál de las partes derechas de un no terminal íbamos a encontrar teniendo en cuenta el símbolo actual de la entrada. La aproximación ascendente es la inversa: en lugar de partir de un no terminal e intentar averiguar la parte derecha que toca, el analizador ascendente intenta averiguar si en la entrada hay una parte derecha y sustituirla por un no terminal.

Las técnicas de análisis que veremos pertenecen a lo que se conoce como familia LR. El nombre viene de que se lee la entrada de izquierda a derecha (*left to right*) y que se obtiene una derivación por la derecha (*rightmost*) de la cadena analizada.

En general, las técnicas LR son más potentes que las LL[1], pero el coste del análisis sigue siendo lineal con la talla de la entrada.

2. Análisis LR(0)

Primero estudiaremos el análisis LR(0). Aunque es el menos potente de la familia LR, tiene la ventaja de ser fácilmente comprensible y servir de base para entender el resto.

2.1. Un ejemplo

Comenzaremos por ver un ejemplo de análisis. Sea la siguiente gramática para expresiones formadas utilizando constantes, paréntesis y el operador suma:

$$\begin{array}{lcl} \langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{E} \rangle + \langle \mathsf{T} \rangle \mid \langle \mathsf{T} \rangle \\ \langle \mathsf{T} \rangle & \rightarrow & \mathsf{cte} \mid (\, \langle \mathsf{E} \rangle \,) \end{array}$$

Vamos a *aumentarla*. Esto consiste en añadir un nuevo símbolo inicial ($\langle \mathsf{S'} \rangle$) a la gramática y la regla $\langle \mathsf{S'} \rangle \rightarrow \langle \mathsf{S} \rangle \$$. Incluimos esta regla para facilitar el análisis: si descubrimos que esta regla se ha empleado para generar la entrada, podemos estar seguros de que no ha habido errores.

[1] Es decir, toda gramática LL(k) es LR(k), pero no a la inversa.

Un par de observaciones. Introducimos esta regla por comodidad, por eso es un especial. En particular, es la única regla en la que puede aparecer el fin de la entrada explícitamente. Además, será la única regla en la que aparezca el nuevo símbolo inicial.

Al aumentar nuestra gramática obtenemos:

$$\begin{array}{lcl} \langle \mathsf{S} \rangle & \to & \langle \mathsf{E} \rangle \mathbf{\$} \\ \langle \mathsf{E} \rangle & \to & \langle \mathsf{E} \rangle \mathbf{+} \langle \mathsf{T} \rangle \mid \langle \mathsf{T} \rangle \\ \langle \mathsf{T} \rangle & \to & \mathbf{cte} \mid \mathbf{(} \langle \mathsf{E} \rangle \mathbf{)} \end{array}$$

Para realizar el análisis, necesitaremos utilizar lo que se conoce como *ítems LR(0)*. Un ítem LR(0) no es más que un par (R, n) donde R es una regla $\langle \mathsf{A} \rangle \to \gamma$ y n es un número entre 0 y la talla de γ, ambos inclusive. Por comodidad, los ítems se representan como $\langle \mathsf{A} \rangle \to \alpha \cdot \beta$, con $\gamma = \alpha\beta$ y $n = |\alpha|$. Si γ es la cadena vacía, escribimos el único ítem LR(0) posible como $\langle \mathsf{A} \rangle \to \cdot$.

Intuitivamente, utilizaremos el ítem $\langle \mathsf{A} \rangle \to \alpha \cdot \beta$ para indicar que, durante nuestro análisis, hemos encontrado en la entrada una subcadena que se deriva de α y esperamos encontrar ahora una subcadena que se derive de β.

Supongamos que queremos analizar la cadena **cte+cte**. Al empezar el análisis, esperamos encontrar algo que se derive del símbolo inicial de la gramática. Esto lo podemos representar mediante el ítem $\langle \mathsf{S} \rangle \to \cdot \langle \mathsf{E} \rangle \mathbf{\$}$. El punto delante de $\langle \mathsf{E} \rangle$ indica que también podemos encontrar algo que se derive de este no terminal. Como la derivación tendrá que suceder a partir de una de las partes derechas, tendremos que añadir otros dos ítems:

$\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{E} \rangle \mathbf{+} \langle \mathsf{T} \rangle$
$\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{T} \rangle$

Y el mismo razonamiento nos lleva a añadir otros dos ítems más para las partes derechas de $\langle \mathsf{T} \rangle$: $\langle \mathsf{T} \rangle \to \cdot \mathbf{cte}$ y $\langle \mathsf{T} \rangle \to \cdot \mathbf{(} \langle \mathsf{E} \rangle \mathbf{)}$. En total, tenemos los siguientes ítems:

$\langle \mathsf{S} \rangle \to \cdot \langle \mathsf{E} \rangle \mathbf{\$}$
$\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{E} \rangle \mathbf{+} \langle \mathsf{T} \rangle$
$\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{T} \rangle$
$\langle \mathsf{T} \rangle \to \cdot \mathbf{cte}$
$\langle \mathsf{T} \rangle \to \cdot \mathbf{(} \langle \mathsf{E} \rangle \mathbf{)}$

Comenzamos el análisis apilando este conjunto de ítems (al que, por razones que luego quedarán claras, llamaremos *estado*). La situación que tenemos ahora es:

$\langle \mathsf{S} \rangle \to \cdot \langle \mathsf{E} \rangle \mathbf{\$}$ $\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{E} \rangle \mathbf{+} \langle \mathsf{T} \rangle$ $\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{T} \rangle$ $\langle \mathsf{T} \rangle \to \cdot \mathbf{cte}$ $\langle \mathsf{T} \rangle \to \cdot \mathbf{(} \langle \mathsf{E} \rangle \mathbf{)}$		▲**cte+cte**

Lo que podemos (debemos) hacer ahora es *desplazar*. Desplazar consiste en leer el siguiente componente léxico, apilarlo y apilar un nuevo estado. El nuevo estado se calcula a partir de los ítems del estado anterior y del componente que acabamos de apilar. Vamos por partes:

- El ítem $\langle \mathsf{T} \rangle \to \cdot \mathbf{cte}$ nos dice que podemos esperar ver una constante y exactamente es eso lo que ha pasado. Para actualizar nuestra información, transformamos el ítem en $\langle \mathsf{T} \rangle \to \mathbf{cte} \cdot$.
- El resto de ítems no pueden utilizarse para justificar la constante, así que desaparecen.

La situación ahora es:

$\langle \mathsf{S} \rangle \to \cdot \langle \mathsf{E} \rangle \mathbf{\$}$ $\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{E} \rangle \mathbf{+} \langle \mathsf{T} \rangle$ $\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{T} \rangle$ $\langle \mathsf{T} \rangle \to \cdot \mathbf{cte}$ $\langle \mathsf{T} \rangle \to \cdot \mathbf{(} \langle \mathsf{E} \rangle \mathbf{)}$	**cte**	$\langle \mathsf{T} \rangle \to \mathbf{cte} \cdot$	**cte**▲**+cte**

Observemos el ítem del tope de la pila. Como el punto está al final de una regla, podemos deducir que hemos encontrado en la entrada esa parte derecha. Visto de otra forma, el terminal **cte** que hemos encontrado es un término. Lo que hace ahora el analizador es sustituir la parte derecha que ha encontrado por la correspondiente parte izquierda, es lo que se llama una *reducción*. Dado que cada símbolo en la parte derecha "ocupa" dos posiciones en la pila (el símbolo y el estado correspondiente), se eliminan de la pila los $2n$ elementos superiores (n es la talla de la parte derecha) y se apila la parte izquierda de la regla. Una vez apilada, se utiliza esta parte izquierda para calcular el nuevo estado en el tope de la pila utilizando para ello el estado inmediato.

Nuestra situación ahora es:

⟨S⟩ → ·⟨E⟩**$** ⟨E⟩ → ·⟨E⟩+⟨T⟩ ⟨E⟩ → ·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	⟨T⟩	⟨E⟩ → ⟨T⟩·		**cte**▲+**cte**

Ahora, el tope de la pila nos indica que el termino que hemos encontrado antes era una expresión, por lo que hay que volver a reducir:

⟨S⟩ → ·⟨E⟩**$** ⟨E⟩ → ·⟨E⟩+⟨T⟩ ⟨E⟩ → ·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	⟨E⟩	⟨S⟩ → ⟨E⟩·**$** ⟨E⟩ → ⟨E⟩·+⟨T⟩		**cte**▲+**cte**

Dado que no tenemos más reducciones, podemos hacer un nuevo desplazamiento:

⟨S⟩ → ·⟨E⟩**$** ⟨E⟩ → ·⟨E⟩+⟨T⟩ ⟨E⟩ → ·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	⟨E⟩	⟨S⟩ → ⟨E⟩·**$** ⟨E⟩ → ⟨E⟩·+⟨T⟩	+	⟨E⟩ → ⟨E⟩+·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	**cte**+▲**cte**

Realizamos otro desplazamiento:

⟨S⟩ → ·⟨E⟩**$** ⟨E⟩ → ·⟨E⟩+⟨T⟩ ⟨E⟩ → ·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	⟨E⟩	⟨S⟩ → ⟨E⟩·**$** ⟨E⟩ → ⟨E⟩·+⟨T⟩	+	⟨E⟩ → ⟨E⟩+·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	**cte**	⟨T⟩ → **cte**·	**cte**+**cte**▲

Ahora podemos reducir un ⟨T⟩:

⟨S⟩ → ·⟨E⟩**$** ⟨E⟩ → ·⟨E⟩+⟨T⟩ ⟨E⟩ → ·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	⟨E⟩	⟨S⟩ → ⟨E⟩·**$** ⟨E⟩ → ⟨E⟩·+⟨T⟩	+	⟨E⟩ → ⟨E⟩+·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	⟨T⟩	⟨E⟩ → ⟨E⟩+⟨T⟩·	**cte**+**cte**▲

Ahora vemos que hemos encontrado una expresión, que era la suma de otra expresión y un término. Así que hacemos la reducción. Aquí se ve claramente la diferencia entre el análisis LR y el LL. De manera informal, un analizador LL tienen que decidir ante la primera **cte** si lo que está viendo es ya una expresión completa o es parte de una suma. Por eso tiene problemas con la recursividad por la izquierda. El analizador LR no tiene ese problema, espera hasta haber visto toda la parte de la entrada generada por una parte derecha antes de reducirla a la parte izquierda.

En este caso, reducimos ⟨E⟩+⟨T⟩ a ⟨E⟩:

⟨S⟩ → ·⟨E⟩**$** ⟨E⟩ → ·⟨E⟩+⟨T⟩ ⟨E⟩ → ·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	⟨E⟩	⟨S⟩ → ⟨E⟩·**$** ⟨E⟩ → ⟨E⟩·+⟨T⟩		**cte+cte**▲

Ahora "desplazaremos" el fin de la entrada. Aunque no es un movimiento del todo correcto, nos simplifica la interpretación del análisis.

⟨S⟩ → ·⟨E⟩**$** ⟨E⟩ → ·⟨E⟩+⟨T⟩ ⟨E⟩ → ·⟨T⟩ ⟨T⟩ → ·**cte** ⟨T⟩ → ·(⟨E⟩)	⟨E⟩	⟨S⟩ → ⟨E⟩·**$** ⟨E⟩ → ⟨E⟩·+⟨T⟩	**$**	⟨S⟩ → ⟨E⟩**$**·	**cte+cte**▲

Ahora podríamos reducir con la regla ⟨S⟩ → ⟨E⟩**$**, pero esto quiere decir que hemos encontrado una derivación de la entrada, así que la aceptamos.

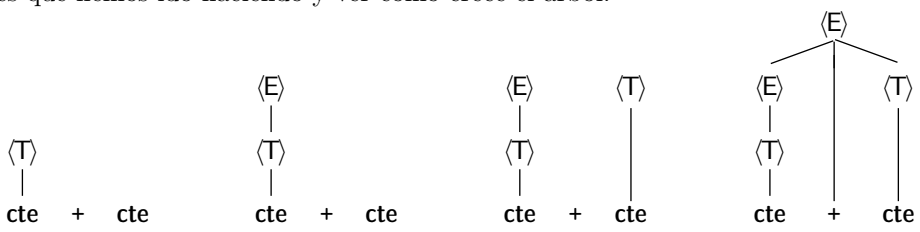
Las reglas que hemos utilizado en las reducciones son las siguientes:

$$\langle T\rangle \to \textbf{cte},\ \langle E\rangle \to \langle T\rangle,\ \langle T\rangle \to \textbf{cte},\ \langle E\rangle \to \langle E\rangle\textbf{+}\langle T\rangle,\ \langle S\rangle \to \langle E\rangle\textbf{\$}.$$

Al recorrer la lista en orden inverso, eliminando la última regla, tenemos una derivación canónica por la derecha de nuestra cadena en la gramática original:

$$\langle E\rangle \Rightarrow \langle E\rangle\textbf{+}\langle T\rangle \Rightarrow \langle E\rangle\textbf{+cte} \Rightarrow \langle T\rangle\textbf{+cte} \Rightarrow \textbf{cte+cte}.$$

Para entender por qué se dice que se construye el árbol de abajo arriba, podemos repasar las reducciones que hemos ido haciendo y ver como crece el árbol:



Ejercicio 1

Analiza la expresión **cte+(cte+cte)**. Escribe la correspondiente derivación.

2.2. Formalización

Vamos a analizar los pasos que hemos dado en el análisis anterior fijándonos sólo en los elementos de la pila que no eran estados. Podemos representar los pasos mediante la siguiente tabla:

Pila	Entrada	Concatenación	Acción
	cte+cte	**cte+cte**	Estado inicial
cte	**+cte**	**cte+cte**	Desplazamos **cte**
⟨T⟩	**+cte**	⟨T⟩**+cte**	Reducimos con ⟨T⟩ → **cte**
⟨E⟩	**+cte**	⟨E⟩**+cte**	Reducimos con ⟨E⟩ → ⟨T⟩
⟨E⟩**+**	**cte**	⟨E⟩**+cte**	Desplazamos **+**
⟨E⟩**+cte**		⟨E⟩**+cte**	Desplazamos **cte**
⟨E⟩+⟨T⟩		⟨E⟩+⟨T⟩	Reducimos con ⟨T⟩ → **cte**
⟨E⟩		⟨E⟩	Reducimos con ⟨E⟩ → ⟨T⟩
⟨E⟩**$**		⟨E⟩**$**	Desplazamos el fin de la entrada
			Aceptamos

Si observas la concatenación de la pila con la entrada, te darás cuenta de que se obtienen los pasos de la derivación por la derecha en orden inverso. Podemos ver el funcionamiento del analizador de la siguiente forma:

- La pila junto con la entrada que falta por analizar es un paso de la derivación de la entrada original.
- Si la pila termina en un sufijo que sea la parte derecha de la regla que se empleó en el paso anterior de la derivación, lo sustituye por la parte izquierda, de modo que se retrocede un paso en el análisis. Por eso, reconstruimos la derivación en orden inverso.
- Si no tenemos una parte derecha, avanzamos en la entrada hasta encontrar alguna.

La regla que empleamos para reducir es lo que se conoce como *selector* (en inglés, *handle*)[2]. Cuando tenemos una forma sentencial, el prefijo que no pasa del selector se llama *prefijo viable*. El analizador mantiene en la pila siempre un prefijo viable y cuando llega al selector, hace la reducción. La cuestión es cómo puede averiguar si en la pila se ha encontrado el final de un prefijo viable o tiene que seguir desplazando. Como los prefijos viables forman un lenguaje regular, la solución es emplear un autómata: el *autómata de prefijos viables*. Los estados de este autómata son los que hemos intercalado en la pila en el ejemplo. Y son ellos los que nos indican qué hacer en cada momento. Además, si el análisis con el autómata no es posible, es que hemos encontrado un error.

2.2.1. Construcción del autómata de prefijos viables

Para construir este autómata necesitamos en primer lugar un procedimiento para construir los estados. Antes lo hemos hecho de manera intuitiva; la formalización es muy sencilla. Si tenemos un conjunto de ítems que representa las situaciones en que "podemos estar" durante el análisis, lo que hay que hacer es aumentarlo para que incluya todos aquellos ítems en los que también podemos estar. Para ello utilizamos la operación de *clausura*.

El siguiente algoritmo hace el cálculo para un conjunto de ítems I:

Algoritmo clausura(I);
  repetir
    $I' := I$;
    para todo $\langle \mathsf{A} \rangle \to \alpha \cdot \langle \mathsf{B} \rangle \beta \in I' : \langle \mathsf{B} \rangle \in N$ **hacer**
      $I := I \cup \{\langle \mathsf{B} \rangle \to \cdot \gamma \mid \langle \mathsf{B} \rangle \to \gamma \in P\}$
    fin para todo
  hasta $I = I'$
  devolver I
fin clausura.

Ejercicio 2

Dada la gramática del ejemplo, calcula la clausura de los siguientes conjuntos de ítems:

- $\{\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{E} \rangle \mathsf{+} \langle \mathsf{T} \rangle\}$
- $\{\langle \mathsf{E} \rangle \to \langle \mathsf{E} \rangle \mathsf{+} \langle \mathsf{T} \rangle \cdot\}$
- $\{\langle \mathsf{E} \rangle \to \cdot \langle \mathsf{E} \rangle \mathsf{+} \langle \mathsf{T} \rangle, \langle \mathsf{T} \rangle \to \mathsf{(} \cdot \langle \mathsf{E} \rangle \mathsf{)}\}$

A medida que se desarrolla el análisis, hay que obtener a partir de un estado su sucesor teniendo en cuenta un símbolo (terminal o no terminal). Para ello, basta utilizar el siguiente algoritmo:

Algoritmo sucesor(I,X);
  devolver clausura($\{\langle \mathsf{A} \rangle \to \alpha X \cdot \beta \mid \langle \mathsf{A} \rangle \to \alpha \cdot X \beta \in I\}$);
fin sucesor.

[2] Técnicamente, el selector está formado por la regla y su posición en la cadena. Sin embargo, cuando la gramática es LR, hay sólo una posición posible para el selector, por lo que se suele hablar de la regla como el selector.

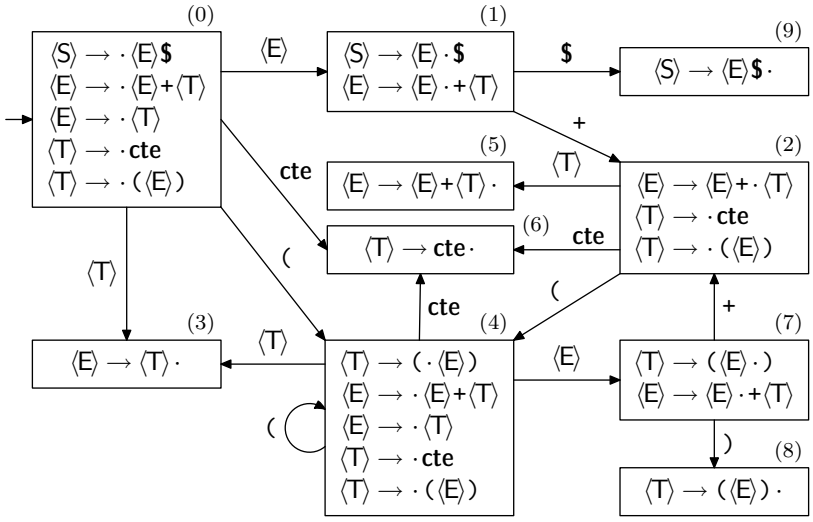


Figura 1: Autómata de prefijos viables para la gramática del ejemplo.

Utilizando estos dos algoritmos podemos construir un autómata finito determinista de la siguiente manera:

- El estado inicial es la clausura de $\{\langle S' \rangle \rightarrow \cdot \langle S \rangle \$\}$, siendo $\langle S' \rangle \rightarrow \langle S \rangle \$$ la regla utilizada para aumentar la gramática.
- Por cada estado E y cada símbolo X, calculamos el sucesor, llamémoslo F. Si F no es vacío, lo añadimos (si no está ya) al conjunto de estados y creamos un arco de E a F etiquetado con X.

Siguiendo estos pasos en nuestro ejemplo, obtenemos el autómata de la figura 1. Considerando todos los estados como finales, este autómata reconoce los prefijos viables.

Ejercicio 3

Calcula el autómata de prefijos viables para la gramática siguiente:

$$\begin{array}{lcl} \langle S \rangle & \rightarrow & \mathtt{a}\langle A \rangle \,|\, \mathtt{c} \\ \langle A \rangle & \rightarrow & \mathtt{b}\langle S \rangle \end{array}$$

2.2.2. Tablas de análisis

A partir del autómata de prefijos viables, podemos construir dos tablas que nos servirán para realizar los análisis, de manera similar a las que empleábamos en el análisis LL(1).

La primera de las tablas es la de acciones, que asocia a cada estado una acción de la siguiente forma:

- Si algún ítem del estado tiene la forma $\langle A \rangle \rightarrow \alpha \cdot \mathtt{a} \beta$, siendo $\mathtt{a}$ un terminal, la acción asociada es *desplazar*.
- Si algún ítem del estado tiene la forma $\langle A \rangle \rightarrow \alpha \cdot$, siendo $\langle A \rangle$ distinto del no terminal con el que se aumentó la gramática, la acción asociada es *reducir* con la regla $\langle A \rangle \rightarrow \alpha$.
- Si el estado es el que contiene el ítem $\langle S' \rangle \rightarrow \langle S \rangle \$ \cdot$, la acción asociada es *aceptar*.

Si siguiendo estas reglas algún estado tiene más de una acción asociada, decimos que la gramática no es LR(0).

La otra tabla que utilizaremos es la tabla de sucesores, que no es más que la representación matricial del autómata de prefijos viables.

En la práctica, suelen escribirse ambas tablas juntas. Así, para nuestro ejemplo, las tablas son:

Estado	Acción	Sucesor							
		+	cte	(	)	$	⟨S⟩	⟨E⟩	⟨T⟩
0	desplazar		6	4				1	3
1	desplazar	2				9			
2	desplazar		6	4					5
3	reducir con ⟨E⟩ → ⟨T⟩								
4	desplazar		6	4				7	3
5	reducir con ⟨E⟩ → ⟨E⟩+⟨T⟩								
6	reducir con ⟨T⟩ → cte								
7	desplazar	2			8				
8	reducir con ⟨T⟩ → (⟨E⟩)								
9	aceptar								

2.2.3. El algoritmo de análisis

Una vez hemos construido las tablas, el siguiente algoritmo nos permite realizar el análisis:

```
Algoritmo análisis LR(0);
    Π := ∅; push(Π,0); // Pila de análisis
    a := siguiente();
    mientras 2 = 1 + 1 hacer
        X := top(Π);
        opción acción[X]:
            aceptar:
                devolver cierto; // Éxito
            desplazar:
                si sucesor[X, a] está definido entonces
                    push(Π, a); push(Π, sucesor[X, a]);
                    a := siguiente();
                si no
                    devolver falso; // Error
                fin si
            reducir (⟨A⟩ → α):
                pop(Π, 2 × |α|);
                X := top(Π);
                push(Π, ⟨A⟩); push(Π,sucesor[X, ⟨A⟩]);
        fin opción
    fin mientras
fin análisis LR(0).
```

Podemos representar el análisis de nuestro ejemplo inicial de la siguiente manera:

Pila	Entrada	Acción
0	**cte+cte**	desplazar
0**cte**6	**+cte**	reducir $\langle \mathsf{T} \rangle \rightarrow$ **cte**
0$\langle$T$\rangle$3	**+cte**	reducir $\langle \mathsf{E} \rangle \rightarrow \langle \mathsf{T} \rangle$
0$\langle$E$\rangle$1	**+cte**	desplazar
0$\langle$E$\rangle$1+2	**cte**	desplazar
0$\langle$E$\rangle$1+2**cte**6		reducir $\langle \mathsf{T} \rangle \rightarrow$ **cte**
0$\langle$E$\rangle$1+2$\langle$T$\rangle$5		reducir $\langle \mathsf{E} \rangle \rightarrow \langle \mathsf{E} \rangle$+$\langle \mathsf{T} \rangle$
0$\langle$E$\rangle$1		desplazar
0$\langle$E$\rangle$1**$**9		aceptar

Ejercicio 4

Analiza la expresión `cte+(cte+cte)`.

Ejercicio 5

Comprueba que la gramática

$$
\begin{array}{lcl}
\langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{E} \rangle \langle \mathsf{E} \rangle \texttt{+} \\
\langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{E} \rangle \langle \mathsf{E} \rangle \texttt{-} \\
\langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{E} \rangle \langle \mathsf{E} \rangle \texttt{*} \\
\langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{E} \rangle \langle \mathsf{E} \rangle \texttt{/} \\
\langle \mathsf{E} \rangle & \rightarrow & \textsf{cte}
\end{array}
$$

es LR(0). Construye sus tablas de análisis y utilízalas para analizar las siguientes entradas:

- **cte cte + cte ***
- **cte cte + cte cte ***

¿Qué lenguaje representa esta gramática?

2.3. Conflictos LR(0)

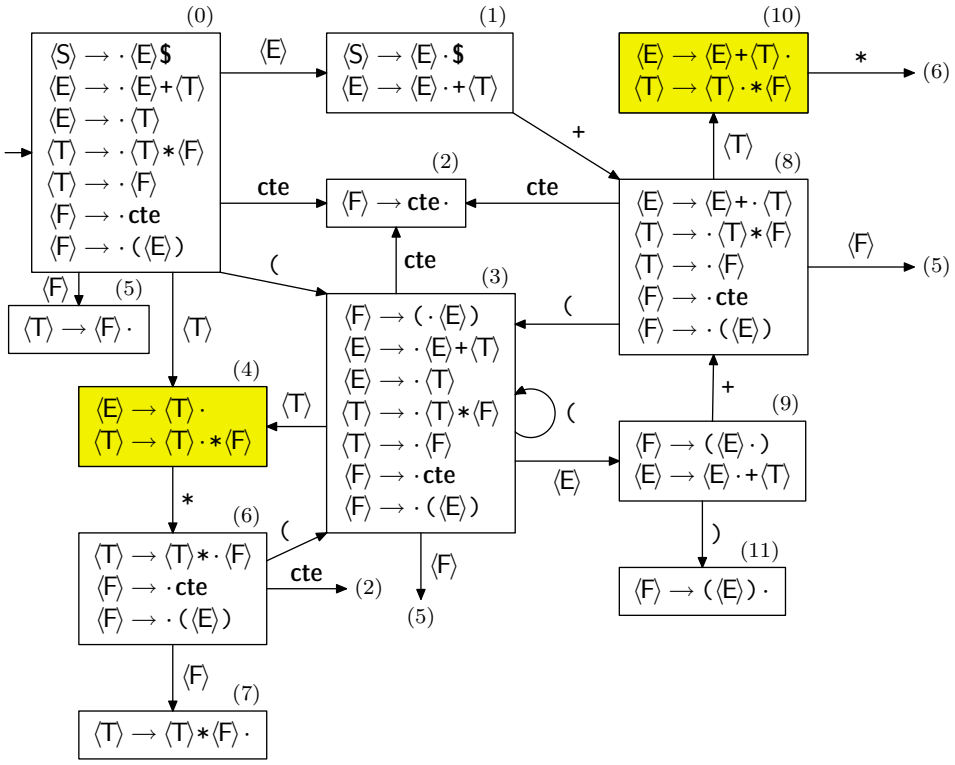
Al construir el analizador, pueden surgir conflictos: entradas en la tabla que tienen asociadas dos o más acciones. Los conflictos pueden ser de dos tipos:

- Conflicto reducción/reducción: aparece cuando en un estado tenemos dos ítems de la forma $\langle \mathsf{A} \rangle \rightarrow \alpha \cdot$. En este caso, el analizador LR(0) no puede decidir cuál de las reducciones efectuar.
- Conflicto desplazamiento/reducción: aparece cuando uno de los ítems del estado indica que hay que desplazar y otro que hay que reducir. En este caso, el analizador LR(0) no puede decidir si desplazar o reducir.

Desgraciadamente, muchas de las gramáticas que representan construcciones habituales en los lenguajes de programación no son LR(0). Al añadir multiplicaciones a la gramática de nuestro ejemplo, obtenemos la siguiente gramática aumentada:

$$
\begin{array}{lcl}
\langle \mathsf{S} \rangle & \rightarrow & \langle \mathsf{E} \rangle \texttt{\$} \\
\langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{E} \rangle \texttt{+} \langle \mathsf{T} \rangle \mid \langle \mathsf{T} \rangle \\
\langle \mathsf{T} \rangle & \rightarrow & \langle \mathsf{T} \rangle \texttt{*} \langle \mathsf{F} \rangle \mid \langle \mathsf{F} \rangle \\
\langle \mathsf{F} \rangle & \rightarrow & \textsf{cte} \mid \texttt{(} \langle \mathsf{E} \rangle \texttt{)}
\end{array}
$$

El correspondiente autómata de prefijos viables es:



Puedes ver que en los estados 4 y 10 tenemos sendos conflictos desplazamiento/reducción. Hemos eliminado el estado al que transitaríamos con el fin de la entrada desde el estado 1 porque, como veremos a continuación, con el análisis SLR no es necesario.

3. Análisis SLR

Vamos a ver ahora cómo podemos enriquecer nuestro método de análisis para tratar de eliminar algunos conflictos. Partamos del ejemplo anterior. En el estado 4, tenemos dos posibilidades: puede que hayamos encontrado un término que constituía por sí solo una expresión o puede que el término que hemos encontrado forme parte de un producto. Reflexionando acerca de esto, podemos ver que si el siguiente componente en la entrada es un $*$ no hay manera de que hayamos encontrado una expresión completa ya que en nuestra gramática una expresión nunca va seguida de un $*$ (puedes calcular siguientes de ⟨E⟩ para verificarlo). Así, si vemos un $*$, lo más razonable es desplazar. Por otro lado, si vemos alguno de los siguientes de ⟨E⟩, lo que tendremos que hacer será reducir. En el caso del estado 10, podemos utilizar un razonamiento similar: un $*$ formaría parte de un término y habría que desplazarlo y los siguientes de ⟨E⟩ llevarían a reducir.

La idea básica del análisis SLR (de *Simple LR*) es aumentar la tabla de acciones de modo que ahora esté indexada por los estados y los símbolos terminales (incluyendo el fin de entrada). Se desplazará cuando el símbolo esté inmediatamente después del punto y sólo se reducirá en caso de que en la entrada se encuentre alguno de los siguientes de la parte izquierda de la regla.

3.1. Construcción de las tablas de análisis

La tabla de sucesores se construye de la misma manera que antes. En cuanto a la de acciones, se hace lo siguiente con cada estado I:

- Por cada ítem $\langle A \rangle \rightarrow \alpha \cdot$ de I (con $\langle A \rangle \neq \langle S' \rangle$) y para cada $a \in$ siguientes($\langle A \rangle$), se inserta "reducir $\langle A \rangle \rightarrow \alpha$" en acción$[I, a]$.

- Por cada ítem $\langle \mathsf{A} \rangle \to \alpha \cdot a\beta$ de I, con $a \in \Sigma$, se inserta "desplazar" en acción$[I, a]$.
- Si $\langle \mathsf{S'} \rangle \to \langle \mathsf{S} \rangle \cdot \$ \in I$, se inserta "aceptar" en acción$[I, \$]$.

La gramática es SLR si cada casilla de la tabla tiene como mucho una acción. En caso contrario, no es SLR.

La tabla de acciones para nuestro ejemplo de expresiones con multiplicación es:

estado	+	*	cte	(	)	$
0			d	d		
1	d					aceptar
2	r ⟨F⟩ → cte	r ⟨F⟩ → cte			r ⟨F⟩ → cte	r ⟨F⟩ → cte
3			d	d		
4	r ⟨E⟩ → ⟨T⟩	d			r ⟨E⟩ → ⟨T⟩	r ⟨E⟩ → ⟨T⟩
5	r ⟨T⟩ → ⟨F⟩	r ⟨T⟩ → ⟨F⟩			r ⟨T⟩ → ⟨F⟩	r ⟨T⟩ → ⟨F⟩
6			d	d		
7	r ⟨T⟩ → ⟨T⟩*⟨F⟩	r ⟨T⟩ → ⟨T⟩*⟨F⟩			r ⟨T⟩ → ⟨T⟩*⟨F⟩	r ⟨T⟩ → ⟨T⟩*⟨F⟩
8			d	d		
9	d				d	
10	r ⟨E⟩ → ⟨E⟩+⟨T⟩	d			r ⟨E⟩ → ⟨E⟩+⟨T⟩	r ⟨E⟩ → ⟨E⟩+⟨T⟩
11	r ⟨F⟩ → (⟨E⟩)	r ⟨F⟩ → (⟨E⟩)			r ⟨F⟩ → (⟨E⟩)	r ⟨F⟩ → (⟨E⟩)

3.2. Algoritmo de análisis

El algoritmo para el análisis SLR es muy parecido al del análisis LR(0), de hecho, lo único que cambia es la manera de decidir las acciones, que ahora tiene en cuenta el próximo símbolo:

Algoritmo análisis SLR;
  Π := ∅; push(Π,0); // Pila de análisis
  a := siguiente();
  mientras $2 = 1 + 1$
    X := top(Π);
    opción acción$[X, a]$:
      aceptar:
        devolver cierto; // Éxito
      desplazar:
        push(Π, a); push(Π, sucesor$[X, a]$);
        a := siguiente();
      reducir ($\langle \mathsf{A} \rangle \to \alpha$):
        pop(Π, $2 \times |\alpha|$);
        X := top(Π);
        push(Π, ⟨A⟩); push(Π,sucesor$[X, \langle \mathsf{A} \rangle]$);
      vacía:
        devolver falso // Error
    fin opción
  fin mientras
fin análisis SLR.

Ejercicio 6

Escribe la tabla de sucesores para la gramática del ejemplo y analiza la cadena cte*(cte+cte).

Ejercicio* 7

En el algoritmo de análisis LR(0), cuando la acción es *desplazar* se comprueba si está definido el sucesor del estado y el símbolo de la entrada. Demuestra que esto no es necesario en el análisis SLR.

Ejercicio 8

Determina si las gramáticas siguientes son o no SLR. En caso afirmativo, calcula las correspondientes tablas de análisis:

- Primera gramática:

$$\begin{array}{lcl} \langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{E} \rangle + \langle \mathsf{T} \rangle \mid \langle \mathsf{T} \rangle \\ \langle \mathsf{T} \rangle & \rightarrow & \langle \mathsf{F} \rangle \uparrow \langle \mathsf{T} \rangle \mid \langle \mathsf{F} \rangle \\ \langle \mathsf{F} \rangle & \rightarrow & \mathbf{cte} \mid \mathbf{id} \mid (\langle \mathsf{E} \rangle) \end{array}$$

- Segunda gramática:

$$\begin{array}{lcl} \langle \mathsf{S} \rangle & \rightarrow & \langle \mathsf{L} \rangle = \langle \mathsf{R} \rangle \mid \langle \mathsf{R} \rangle \\ \langle \mathsf{L} \rangle & \rightarrow & * \langle \mathsf{R} \rangle \mid \mathbf{id} \\ \langle \mathsf{R} \rangle & \rightarrow & \langle \mathsf{L} \rangle \end{array}$$

3.3. Conflictos SLR

Pese a que la familia SLR es más amplia que la LR(0), sigue habiendo casos en los que se presentan conflictos. Como antes, los conflictos pueden ser:

- Reducción/reducción: se da cuando hay dos ítems que permiten la reducción y la intersección de los siguientes de las partes izquierdas de las reglas es no vacía.
- Desplazamiento/reducción: se da si alguno de los terminales asociado a un desplazamiento forma parte del conjunto de siguientes del no terminal que se puede reducir.

Aunque muchas de las construcciones habituales en los lenguajes de programación se pueden representar mediante gramáticas SLR, es posible que se presenten conflictos. En la práctica se usa un método más potente, el análisis LALR, pero para verlo, tendremos que estudiar primero el método más potente de análisis con coste lineal y un símbolo de anticipación: el análisis LR(1).

4. Análisis LR(1)

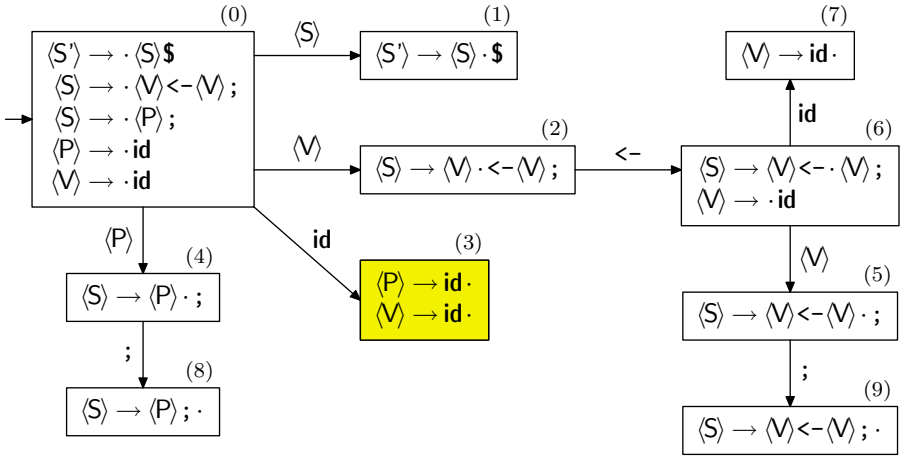
Ahora estudiaremos el análisis LR(1). La idea es muy similar a la del análisis SLR, pero en lugar de reducir cuando nos lo indiquen los siguientes lo que haremos será incorporar a los ítems la información acerca de cuándo es posible efectuar la reducción.

4.1. Un ejemplo

Supongamos que utilizamos la gramática siguiente (ya aumentada) para modelar parte de un lenguaje de programación que permite sentencias que son asignaciones entre variables y otras que son llamadas a procedimiento:

$$\begin{array}{lcl} \langle \mathsf{S'} \rangle & \rightarrow & \langle \mathsf{S} \rangle \mathbf{\$} \\ \langle \mathsf{S} \rangle & \rightarrow & \langle \mathsf{V} \rangle <\!- \langle \mathsf{V} \rangle \, ; \\ \langle \mathsf{S} \rangle & \rightarrow & \langle \mathsf{P} \rangle \, ; \\ \langle \mathsf{P} \rangle & \rightarrow & \mathbf{id} \\ \langle \mathsf{V} \rangle & \rightarrow & \mathbf{id} \end{array}$$

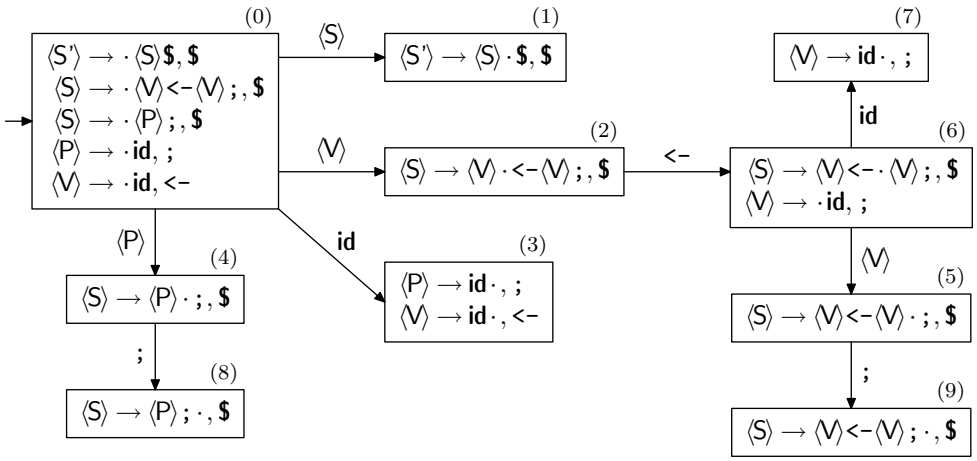
Si intentamos construir nuestro autómata LR(0), llegamos a:



Podemos ver que en el estado 3 hay un posible conflicto. Como los siguientes de ⟨P⟩ y ⟨V⟩ contienen el punto y coma, no podemos utilizar el análisis SLR. Sin embargo, un poco de reflexión nos permite darnos cuenta de que si la categoría en la entrada es ;, sólo podemos reducir con la regla ⟨P⟩ → **id**. La reducción con la regla ⟨V⟩ → **id** ante un punto y coma corresponde al estado 7.

¿Cómo podemos incorporar en nuestro método de análisis la intuición anterior? Lo que haremos será incorporar a nuestros ítems un nuevo elemento: un símbolo terminal, o el fin de la entrada, que indicará con qué símbolo de anticipación es válido el ítem. Así, podemos empezar a construir el estado inicial de nuestro autómata de prefijos viables con el ítem ⟨S'⟩ → · ⟨S⟩$, $, que nos indica que esperamos ver la expansión de ⟨S⟩, pero seguida de $. Al construir la clausura, añadiremos los ítems ⟨S⟩ → · ⟨V⟩<-⟨V⟩;, $ y ⟨S⟩ → · ⟨P⟩;, $. Ambos tienen como símbolo de anticipación el fin de la entrada ya que era lo que estaba a continuación de ⟨S⟩ inicialmente. Ahora, al añadir un ítem para ⟨V⟩, nos damos cuenta de que estará seguido del terminal <-, por lo que el ítem que añadimos es ⟨V⟩ → · **id**, <-. Finalmente, añadimos el ítem ⟨P⟩ → · **id**, ;.

Para calcular el sucesor de un estado con un símbolo, bastará con calcular la clausura como hemos hecho antes de los correspondientes ítems habiendo movido el punto, pero manteniendo los símbolos de anticipación. De esta manera, obtenemos el siguiente autómata:



Ahora, es fácil obtener la tabla de acciones. Para desplazar y aceptar, el criterio no cambia, y

para reducir, tendremos que mirar el símbolo de anticipación. En nuestro ejemplo, la tabla es:

Estado	**id**	<-	;	$
0	d			
1				aceptar
2		d		
3		r ⟨V⟩ → **id**	r ⟨P⟩ → **id**	
4			d	
5			d	
6	d			
7			r ⟨V⟩ → **id**	
8				r ⟨S⟩ → ⟨P⟩ ;
9				r ⟨S⟩ → ⟨V⟩<-⟨V⟩ ;

Como se ve, ya han desaparecido los conflictos. Ten en cuenta que los conflictos no se eliminan al añadir reglas (¿por qué?). Así pues, aunque la gramática es muy sencilla, el problema puede aparecer fácilmente al especificar un lenguaje de programación.

EJERCICIO 9

Escribe la tabla de sucesores para la gramática anterior.

4.2. Formalización

Vamos a formalizar ligeramente las intuiciones anteriores. Empezaremos por el concepto de ítem. Un *ítem LR(1)* es una tripleta formada por una producción, un índice (el punto) y un terminal (que representa una anticipación válida para el ítem).

Dado un ítem LR(1) $\langle A\rangle \to \alpha\cdot\beta, \mathtt{a}$, llamaremos *núcleo* de este ítem a $\langle A\rangle \to \alpha\cdot\beta$.

El algoritmo para el cálculo de la clausura es prácticamente igual que el que teníamos para los ítems LR(0). Lo único que tendremos que hacer es propagar adecuadamente los símbolos de anticipación: si tenemos $\langle A\rangle \to \alpha\cdot\langle B\rangle\beta, \mathtt{a}$, habrá que añadir los ítems $\langle B\rangle \to \cdot\gamma, \mathtt{b}$ con los $\mathtt{b}$ adecuados. Por un lado, podrá ir cualquiera de los primeros de β. Por otro, si β se anula, podrá ir $\mathtt{a}$. En resumen, puede ir cualquiera de los primeros de $\beta\mathtt{a}$. El algoritmo es entonces:

Algoritmo clausura(I);
  repetir
    $I' := I$;
    para todo $\langle A\rangle \to \alpha\cdot\langle B\rangle\beta, \mathtt{a} \in I' : \langle B\rangle \in N$ **hacer**
      $I := I \cup \{\langle B\rangle \to \cdot\gamma, \mathtt{b} \mid \langle B\rangle \to \gamma \in P \wedge \mathtt{b} \in \text{primeros}(\beta\mathtt{a})\}$
    fin para todo
  hasta $I = I'$
  devolver I
fin clausura.

El cálculo del sucesor es análogo al del caso LR(0):

Algoritmo sucesor(I,X);
  devolver clausura($\{\langle A\rangle \to \alpha X\cdot\beta, \mathtt{a} \mid \langle A\rangle \to \alpha\cdot X\beta, \mathtt{a} \in I\}$);
fin sucesor.

Con esta función, se obtiene la tabla de sucesores igual que en el caso LR(0), con la salvedad de que el estado inicial es el conjunto clausura de $\langle S'\rangle \to \cdot\langle S\rangle\$, \$$.

En cuanto a la tabla de acciones, la rellenamos asociando a cada estado la correspondiente acción:

- Si algún ítem del estado tiene la forma $\langle A\rangle \to \alpha\cdot\mathtt{a}\beta, \mathtt{b}$, la acción asociada al estado y el terminal $\mathtt{a}$ es *desplazar*.

- Si algún ítem del estado tiene la forma $\langle A\rangle \to \alpha\cdot, a$, siendo $\langle A\rangle$ distinto del no terminal con el que se aumentó la gramática, la acción asociada al estado y el terminal a es *reducir* con la regla $\langle A\rangle \to \alpha$ (a puede ser **$**).
- Si el estado contiene el ítem $\langle S'\rangle \to \langle S\rangle\cdot \$, \$$, la acción asociada al estado y a **$** es *aceptar*.

Si ningún par estado, terminal recibe más de una acción, se dice que la gramática es LR(1).

El algoritmo de análisis es exactamente el mismo que en el caso del SLR.

Ejercicio* 10

Construye la tabla de análisis LR(1) para la segunda gramática del ejercicio 8.

4.3. Análisis LALR

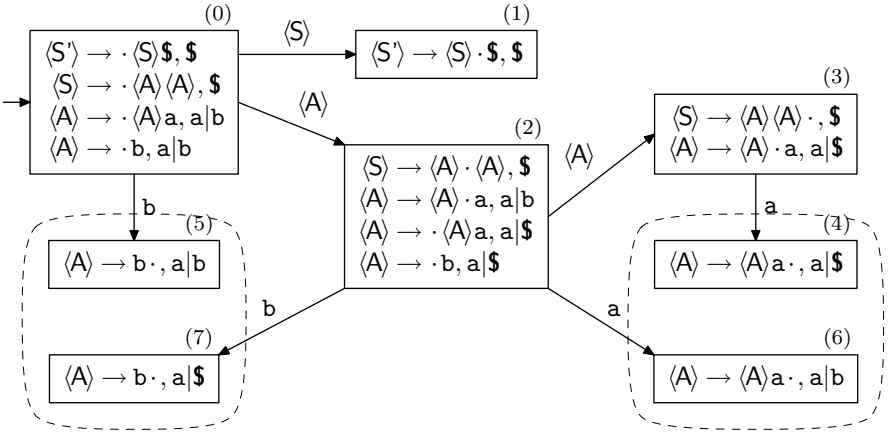
El análisis LR(1) tiene la ventaja de su gran potencia. Sin embargo, las tablas asociadas son muy grandes debido a la gran cantidad de estados necesarios para representar el autómata de prefijos viables. La técnica de análisis LALR (de Lookahead LR) intenta eliminar este problema uniendo estados que realizan la misma función. El precio es una ligera reducción en la cantidad de gramáticas analizables.

Vamos a estudiar la técnica LALR con un ejemplo sencillo. Sea la gramática:

$$\begin{array}{lcl} \langle S'\rangle & \to & \langle S\rangle \$ \\ \langle S\rangle & \to & \langle A\rangle\langle A\rangle \\ \langle A\rangle & \to & \langle A\rangle a | b \end{array}$$

Puede que te hayas dado cuenta de que esta gramática en realidad es SLR; es cierto, pero para el ejemplo servirá ya que las gramáticas realistas enseguida necesitan gran cantidad de estados.

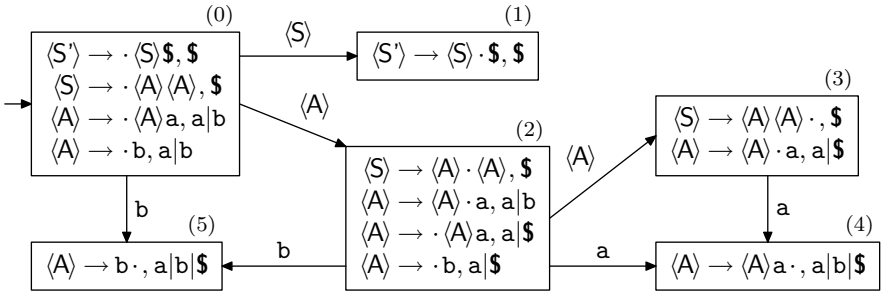
El autómata de prefijos viables LR(1) es el siguiente[3]:



Al analizar los estados 4 y 6, vemos que ambos tienen la misma función; los dos se encargan de la reducción mediante la regla $\langle A\rangle \to \langle A\rangle a$. La diferencia está en el contexto en que se da la reducción; el estado 4 reduce las $\langle A\rangle$ que están en la segunda parte de la cadena y el estado 6 las de la primera. Una relación similar se da entre el estado 5 y el 7.

[3]Hemos utilizado la notación $\langle A\rangle \to \alpha, a|b$ para indicar el par de ítems $\langle A\rangle \to \alpha, a$ y $\langle A\rangle \to \alpha, b$

Si fundimos los pares de estados anteriores, obtenemos el siguiente autómata:



Si comparamos el funcionamiento de ambos autómatas ante cadenas del lenguaje, vemos que es análogo. Ante cadenas erróneas, el segundo autómata retarda ligeramente la detección del error, ya que puede realizar algunas reducciones incorrectas, pero no consume ningún símbolo adicional de la entrada.

Ejercicio 11

Comprueba cómo funcionan ambos autómatas ante las cadenas `bb` y `bbb`.

Así pues, el autómata LALR puede verse como el resultado de crear primero el autómata LR(1) y luego unir aquellos estados que tienen los mismos núcleos (los mismos ítems cuando no se consideran los símbolos de anticipación). Lógicamente, en la práctica, el autómata se calcula utilizando algoritmos más eficientes que este proceso de construcción del autómata LR(1) y fusión de estados.

Como te podrás imaginar, para aplicaciones prácticas el autómata LALR no se calcula manualmente sino que se emplea algún metacompilador. Quizá el más conocido sea YACC, o su versión GNU: Bison. Tanto uno como otro se distribuyen con UNIX/Linux y están preparados para integrarse fácilmente con los analizadores léxicos generados por lex o flex.

Puede que te preguntes si al fusionar estados no se pueden crear conflictos. La respuesta es sí, pero afortunadamente es raro:

- No pueden aparecer conflictos desplazamiento/reducción que no estuvieran presentes en la gramática original.
- Pueden aparecer conflictos reducción/reducción, pero ocasionalmente.

Ejercicio* 12

Demuestra que al fusionar estados no pueden aparecer conflictos desplazamiento/reducción.

Ejercicio* 13

Construye la tabla de análisis LALR para la segunda gramática del ejercicio 8.

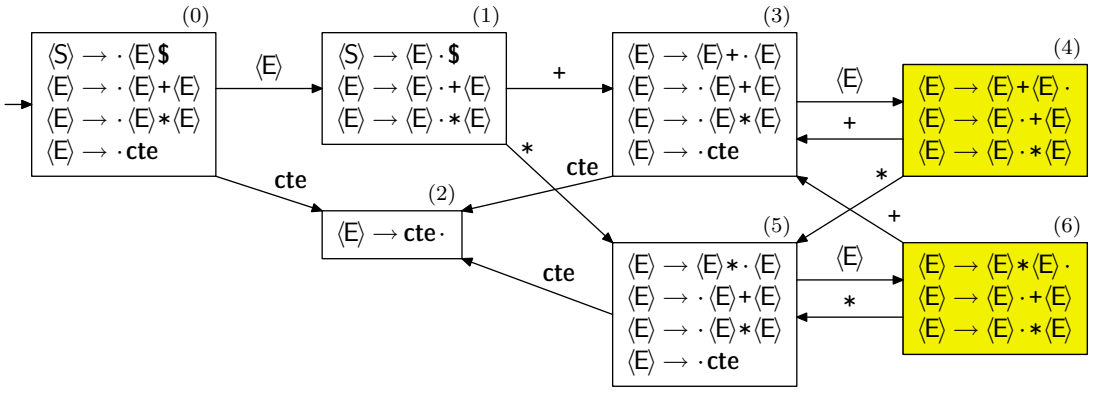
5. Uso de gramáticas ambiguas

Una de las propiedades básicas de las gramáticas LR es que no son ambiguas. Sin embargo, sabemos que determinadas construcciones se expresan con más facilidad mediante gramáticas ambiguas y normas para resolver las ambigüedades. Por ejemplo, en el caso del **if-then-else** era más fácil aplicar la regla de asociar el **else** con el **if** más cercano que escribir la gramática no ambigua.

Vamos a estudiar con un poco de detalle otro ejemplo. Sea la gramática:

$$
\begin{array}{lcl}
\langle S \rangle & \rightarrow & \langle E \rangle \$ \\
\langle E \rangle & \rightarrow & \langle E \rangle + \langle E \rangle \\
\langle E \rangle & \rightarrow & \langle E \rangle * \langle E \rangle \\
\langle E \rangle & \rightarrow & \mathbf{cte}
\end{array}
$$

Evidentemente, esta gramática es ambigua por lo que no será LR. A pesar de eso, vamos calcular el autómata de prefijos viables. Utilizando ítems LR(0), obtenemos:



Aquí vemos que la ambigüedad provoca conflictos en los estados 4 y 6. Vamos a analizar lo que pasa en el estado 4. Dado que + es un siguiente de ⟨E⟩, podemos reducir con la regla ⟨E⟩ → ⟨E⟩+⟨E⟩. Por otro lado, el ítem ⟨E⟩ → ⟨E⟩ · +⟨E⟩ nos indica que podemos desplazar.

Supongamos ahora que intentamos analizar la cadena **cte+cte+cte**. El comienzo del análisis es:

Pila	Entrada	Acción
0	**cte+cte+cte$**	desplazar
0**cte**2	**+cte+cte$**	reducir ⟨E⟩ → **cte**
0⟨E⟩1	**+cte+cte$**	desplazar
0⟨E⟩1+3	**cte+cte$**	desplazar
0⟨E⟩1+3**cte**2	**+cte$**	reducir ⟨E⟩ → **cte**
0⟨E⟩1+3⟨E⟩4	**+cte$**	¿?

La situación ahora se puede resolver teniendo en cuenta la asociatividad de la suma. Dado que queremos que sea asociativa a izquierdas, la acción en el estado 4 ante un + debe ser reducir.

Si intentamos analizar la cadena **cte+cte∗cte**, los primeros pasos son:

Pila	Entrada	Acción
0	**cte+cte∗cte$**	desplazar
0**cte**2	**+cte∗cte$**	reducir ⟨E⟩ → **cte**
0⟨E⟩1	**+cte∗cte$**	desplazar
0⟨E⟩1+3	**cte∗cte$**	desplazar
0⟨E⟩1+3**cte**2	**∗cte$**	reducir ⟨E⟩ → **cte**
0⟨E⟩1+3⟨E⟩4	**∗cte$**	¿?

Dado que el producto tiene prioridad sobre la suma, el conflicto con el símbolo ∗ se resuelve desplazando.

Ejercicio 14

Resuelve los conflictos del estado 6 y escribe la tabla de análisis.

Como puedes imaginarte, modificar directamente las tablas de análisis para resolver los conflictos necesita un conocimiento profundo de la estructura de la gramática y de las consecuencias de las modificaciones. Por eso, los metacompiladores suelen ofrecer maneras indirectas de hacer estas modificaciones. Así, en YACC/Bison, es posible establecer las prioridades y asociatividad de

los operadores de modo que las modificaciones son realizadas automáticamente.

Ejercicio 15

Diseña un analizador SLR para la gramática:

$$\begin{aligned}\langle S\rangle &\rightarrow \textbf{if cond then } \langle S\rangle \textbf{ else } \langle S\rangle\\ \langle S\rangle &\rightarrow \textbf{if cond then } \langle S\rangle\\ \langle S\rangle &\rightarrow \textbf{sentencia}\end{aligned}$$

Resuelve los conflictos aplicando la regla que asocia al **else** el **if** más próximo.

6. Tratamiento de errores

La detección de errores se realiza de la misma forma en todos los métodos LR: en un momento dado encontramos una entrada en blanco en la tabla de acciones. La diferencia que hay entre el análisis LR(1) y los LALR y SLR ya la hemos mencionado antes. Un analizador LR(1) anunciará el error sin realizar siquiera una reducción; un analizador LALR o SLR puede que efectúe alguna reducción, pero no un desplazamiento.

Habitualmente estamos interesados en informar del error y después recuperarnos. Suele ser más difícil dar un informe de error preciso con análisis LR que con LL. La razón es que un estado es una "superposición" de distintos análisis posibles. En cuanto a la recuperación, también es ligeramente más difícil que en el análisis LL.

Una posibilidad es realizar una recuperación en modo pánico. En este caso se procede ante un error de la siguiente manera:

- Se desapilan elementos hasta encontrar un estado que represente el intento de analizar un no terminal adecuado, llamémoslo $\langle A\rangle$.
- Se avanza sobre la cadena de entrada hasta encontrar un siguiente de $\langle A\rangle$.
- Se apila el sucesor de $\langle A\rangle$ con la entrada encontrada y se continúa el análisis.

Lógicamente, el problema está en encontrar el "no terminal adecuado". Normalmente se eligen aquellos que representen estructuras importantes del programa: sentencias, bloques, expresiones,...

Los compiladores generados por YACC implementan una versión refinada de esta estrategia en la que los terminales se marcan mediante producciones de error similares a las de metacomp[4].

7. Acciones semánticas

Dado que los analizadores LR no deciden hasta el último momento qué regla van a utilizar, sólo pueden ejecutar las acciones semánticas "al final". Para entenderlo, supongamos que tenemos las siguientes reglas:

$$\begin{aligned}\langle A\rangle &\rightarrow \textbf{uno } \{\text{acción1}\} \textbf{ dos tres}\\ \langle A\rangle &\rightarrow \textbf{uno } \{\text{acción2}\} \textbf{ dos cuatro}\end{aligned}$$

Cuando hemos desplazado **uno** y vemos **dos**, no es posible decidir cuál de las dos reglas será la que finalmente se reduzca. Por ello, no es posible elegir entre las dos acciones.

La consecuencia inmediata de lo anterior es que los esquemas de traducción que tengan atributos heredados presentarán problemas para su implementación. Los que únicamente tengan atributos sintetizados no tienen problemas ya que se pueden poner las correspondientes acciones al final.

¿Cómo podemos ejecutar acciones que no estén al final? Una posible solución es introducir nuevos no terminales que se reescriban como la cadena vacía y que tengan asociada la acción correspondiente. Así, la regla

$$\langle A\rangle \rightarrow \alpha\ \{\text{acción}\}\ \beta$$

[4] Como cabría esperar, fue metacomp el que se inspiró en YACC :-).

se sustituiría por el par

$$\begin{array}{lcl} \langle \mathsf{A} \rangle & \rightarrow & \alpha \langle \mathsf{M} \rangle \beta \\ \langle \mathsf{M} \rangle & \rightarrow & \lambda \ \{\text{acción}\} \end{array}$$

Por ejemplo, el esquema

$$\begin{array}{lcl} \langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{T} \rangle \langle \mathsf{R} \rangle \\ \langle \mathsf{R} \rangle & \rightarrow & \mathsf{+} \langle \mathsf{T} \rangle \ \{\text{print}(\text{'+'})\} \ \langle \mathsf{R} \rangle | \mathsf{-} \langle \mathsf{T} \rangle \ \{\text{print}(\text{'-'})\} \ \langle \mathsf{R} \rangle | \lambda \\ \langle \mathsf{T} \rangle & \rightarrow & \mathsf{cte} \ \{\text{print}(\mathsf{cte}.\text{val})\} \end{array}$$

se reescribiría como

$$\begin{array}{lcl} \langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{T} \rangle \langle \mathsf{R} \rangle \\ \langle \mathsf{R} \rangle & \rightarrow & \mathsf{+} \langle \mathsf{T} \rangle \langle \mathsf{M} \rangle \langle \mathsf{R} \rangle | \mathsf{-} \langle \mathsf{T} \rangle \langle \mathsf{N} \rangle \langle \mathsf{R} \rangle | \lambda \\ \langle \mathsf{T} \rangle & \rightarrow & \mathsf{cte} \ \{\text{print}(\mathsf{cte}.\text{val})\} \\ \langle \mathsf{M} \rangle & \rightarrow & \lambda \ \{\text{print}(\text{'+'})\} \\ \langle \mathsf{N} \rangle & \rightarrow & \lambda \ \{\text{print}(\text{'-'})\} \end{array}$$

Este es el método utilizado por YACC.

En cuanto a los atributos en sí, los podemos almacenar en la pila, junto a los no terminales o terminales correspondientes. Por ejemplo, vamos a aumentar la gramática con la que empezamos el tema para que calcule el valor de las sumas:

$$\begin{array}{lcl} \langle \mathsf{S} \rangle & \rightarrow & \langle \mathsf{E} \rangle \ \{\text{print} \ (\langle \mathsf{E} \rangle.\text{v})\} \\ \langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{E} \rangle \mathsf{+} \langle \mathsf{T} \rangle \ \{\langle \mathsf{E} \rangle.\text{v} := \langle \mathsf{E} \rangle_1.\text{v} + \langle \mathsf{T} \rangle.\text{v}\} \\ \langle \mathsf{E} \rangle & \rightarrow & \langle \mathsf{T} \rangle \ \{\langle \mathsf{E} \rangle.\text{v} := \langle \mathsf{T} \rangle.\text{v}\} \\ \langle \mathsf{T} \rangle & \rightarrow & \mathsf{cte} \ \{\langle \mathsf{T} \rangle.\text{v} := \mathsf{cte}.\text{v}\} \\ \langle \mathsf{T} \rangle & \rightarrow & \mathsf{(} \langle \mathsf{E} \rangle \mathsf{)} \ \{\langle \mathsf{T} \rangle.\text{v} := \langle \mathsf{E} \rangle.\text{v}\} \end{array}$$

La traza del análisis de $\mathsf{cte}_3\mathsf{+}\mathsf{cte}_4$ sería:

Pila	Entrada	Acción
0	**cte+cte$**	desplazar
0 **cte** 6 3	**+cte$**	reducir $\langle \mathsf{T} \rangle \rightarrow \mathsf{cte}$
0 $\langle \mathsf{T} \rangle$ 3 3	**+cte$**	reducir $\langle \mathsf{E} \rangle \rightarrow \langle \mathsf{T} \rangle$
0 $\langle \mathsf{E} \rangle$ 1 3	**+cte$**	desplazar
0 $\langle \mathsf{E} \rangle$ 1 + 2 3	**cte$**	desplazar
0 $\langle \mathsf{E} \rangle$ 1 + 2 **cte** 6 3 4	**$**	reducir $\langle \mathsf{T} \rangle \rightarrow \mathsf{cte}$
0 $\langle \mathsf{E} \rangle$ 1 + 2 $\langle \mathsf{T} \rangle$ 5 3 4	**$**	reducir $\langle \mathsf{E} \rangle \rightarrow \langle \mathsf{E} \rangle \mathsf{+} \langle \mathsf{T} \rangle$
0 $\langle \mathsf{E} \rangle$ 1 7	**$**	aceptar

Ejercicio 16

Escribe la traza para la entrada $\texttt{cte}_3\texttt{+(cte}_4\texttt{+cte}_5\texttt{)}$.

8. Resumen

- Análisis LR: se construye una derivación por la derecha.
- No se decide qué regla se emplea hasta ver todo lo que ha producido.
- Ítem LR(0):
  - Regla y posición en la parte derecha (punto).
  - A la derecha del punto, lo que hemos visto, a la izquierda, lo que esperamos.
- Elementos del analizador:
  - Autómata de prefijos viables:
    - Construido a partir de las clausuras de los conjuntos de ítems.
    - Los estados representan la situación del análisis.
  - Tabla de acciones.
- Tipos de conflicto: reducción/reducción, desplazamiento/reducción.
- Gramática LR(0): no tiene conflictos LR(0).
- Análisis SLR: ítems LR(0) y tabla de análisis con un símbolo de anticipación.
- Análisis LR(1): los ítems tienen además un símbolo de anticipación.
- Análisis SLR: se unen los estados del autómata LR(1) que tienen el mismo núcleo.
- Gramáticas ambiguas:
  - Modificando la tabla.
  - Con precaución.
  - Los metacompiladores tienen métodos indirectos de modificación.
- Tratamiento de errores:
  - Más complejo que en LL.
  - Modo pánico.
  - Producciones de error.
- Acciones semánticas:
  - Al final de las reglas.
  - En medio con reglas "virtuales".
  - Atributos en la pila.