



# Parseo y Generación de Código

27 de septiembre de 2018

**Análisis sintáctico ascendente**

Licenciatura en Informática con Orientación en Desarrollo de Software  
Universidad Nacional de Quilmes

# Análisis sintáctico ascendente

# Análisis sintáctico

*Recordemos:* dada una gramática  $G = (N, \Sigma, P, S)$  y una cadena  $\alpha$ , el problema de **análisis sintáctico** consiste en determinar si  $\alpha \in L(G)$  y, en tal caso, dar una derivación  $S \Rightarrow^* \alpha$ .

# Análisis sintáctico descendente (repaso)

La clase pasada nos enfocamos en las técnicas de análisis sintáctico **descendente**.

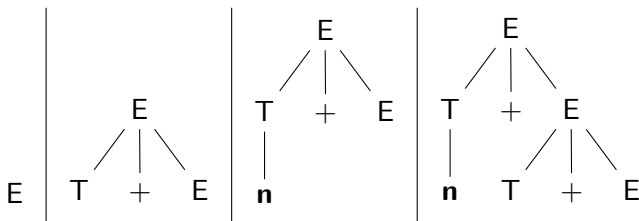
El parser en cada paso tiene un árbol de derivación parcialmente construído, que se va completando **desde la raíz hacia las hojas**, consumiendo símbolos de la entrada y **expandiendo** producciones.

# Análisis sintáctico descendente (repaso)

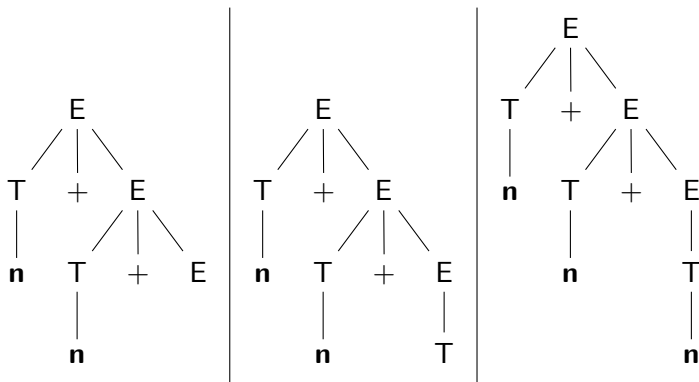
Por ejemplo, con la gramática  $G = (\{E\}, \{\mathbf{n}, +\}, P, E)$ :

$$\begin{aligned} E &\rightarrow T \mid T + E \\ T &\rightarrow \mathbf{n} \end{aligned}$$

y la cadena de entrada  $\mathbf{n} + \mathbf{n} + \mathbf{n}$  el proceso de análisis sintáctico descendente podría ser:



## Análisis sintáctico descendente (repaso)



Observar que la derivación obtenida es una derivación **más a la izquierda**.

# Análisis sintáctico ascendente

En esta clase nos enfocamos en las técnicas de análisis sintáctico **ascendente**.

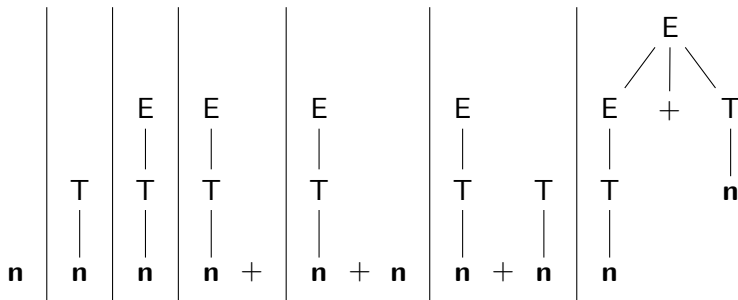
El parser en cada paso tiene un árbol de derivación parcialmente construido, que se va completando **desde las hojas hacia la raíz**, consumiendo símbolos de la entrada y **reduciendo** producciones.

# Análisis sintáctico ascendente

Por ejemplo, con la gramática  $G' = (\{E\}, \{\mathbf{n}, +\}, P, E)$ :

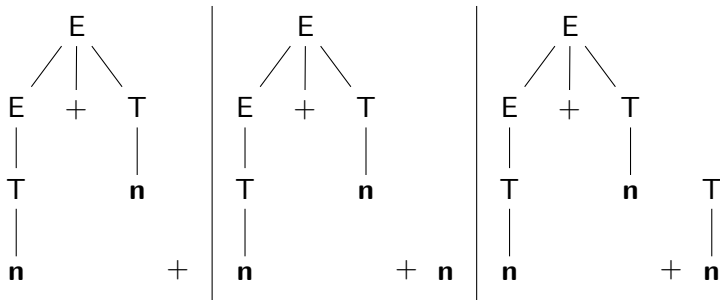
$$\begin{aligned} E &\rightarrow T \mid E + T \\ T &\rightarrow \mathbf{n} \end{aligned}$$

y la cadena de entrada  $\mathbf{n} + \mathbf{n} + \mathbf{n}$  el proceso de análisis sintáctico ascendente podría ser:

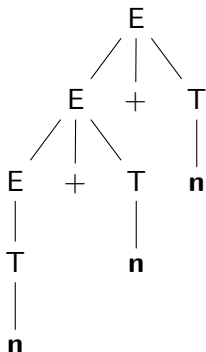




## Análisis sintáctico ascendente



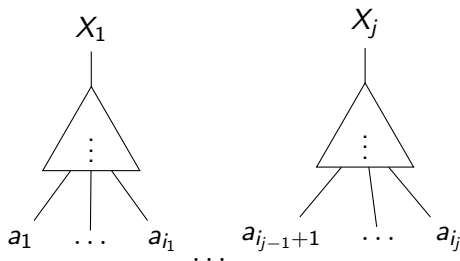
## Análisis sintáctico ascendente



Observar que la derivación obtenida es una derivación **más a la derecha**.

# Análisis sintáctico ascendente

En general en un momento dado del análisis sintáctico para la cadena  $\alpha = a_1 \dots a_n$ , ya se construyeron  $j$  subárboles del árbol de derivación. Es decir, se tiene una pila de  $j$  árboles:

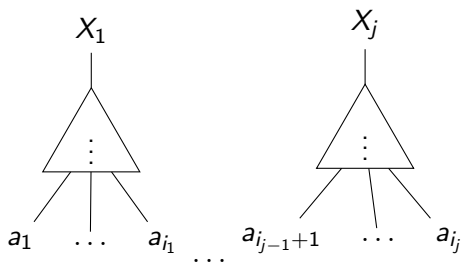


Hay tres acciones posibles:

**Caso 1:** si existe un  $k$  tal que los  $k$  símbolos  $X_{j-k+1} \dots X_j$  en el tope de la pila corresponden al lado derecho de una producción  $A \rightarrow X_{j-k+1} \dots X_j$ , se aplica un paso **reduce**, creando un nuevo nodo  $A$  con dichos  $k$  árboles como hijos.

# Análisis sintáctico ascendente

En general en un momento dado del análisis sintáctico para la cadena  $\alpha = a_1 \dots a_n$ , ya se construyeron  $j$  subárboles del árbol de derivación. Es decir, se tiene una pila de  $j$  árboles:

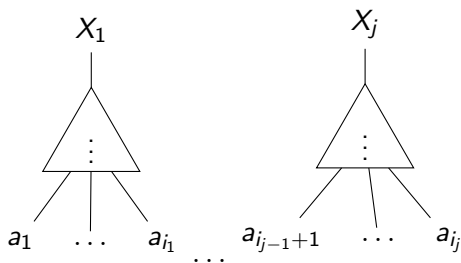


Hay tres acciones posibles:

**Caso 2:** si todavía hay símbolos de la entrada para consumir, se aplica un paso **shift**, incorporando el siguiente símbolo de la entrada  $a_{i_j+1}$  a la pila.

## Análisis sintáctico ascendente

En general en un momento dado del análisis sintáctico para la cadena  $\alpha = a_1 \dots a_n$ , ya se construyeron  $j$  subárboles del árbol de derivación. Es decir, se tiene una pila de  $j$  árboles:



Hay tres acciones posibles:

**Caso 3:** en caso contrario, no hay símbolos para consumir y no es posible reducir la cadena en la pila. Si queda un único árbol de derivación con raíz  $S$ , es decir,  $X_1 \dots X_j = S$ , la cadena se acepta. De lo contrario, se rechaza.

## Análisis sintáctico ascendente con oráculo

El siguiente algoritmo genérico de análisis sintáctico ascendente asume que se dispone de un oráculo que siempre puede determinar si hay que hacer **shift** o **reduce**, y que en caso de **reduce** elige siempre la producción correcta.

# Análisis sintáctico ascendente con oráculo

*Entrada:* Una gramática  $G = (N, \Sigma, P, S)$ ,  
una cadena  $\alpha$ .

*Salida:* Un booleano indicando si  $\alpha \in L(G)$  y en tal caso  
una derivación más a la derecha de  $S \Rightarrow^* \alpha$ .

*pila* := []

**loop**

**if** el oráculo elige una producción  $A \rightarrow X_1 \dots X_k$  donde  
         $X_1 \dots X_k$  son los  $k$  símbolos en el tope de la  
        pila ( $X_k$  es el símbolo en el tope)

        Sacar  $k$  símbolos de *pila*.

*pila*.push( $A$ )

        Agregar la producción  $A \rightarrow X_1 \dots X_k$  como primer  
        paso de la derivación construida.

**elseif**  $\alpha$  no es vacía,  $a\beta := \alpha$ .

*pila*.push( $a$ )

$\alpha := \beta$

**elseif** *pila* = [ $S$ ]

**return** Éxito:  $\alpha \in L(G)$  con la derivación construida.

**else**

**return** Falla:  $\alpha \notin L(G)$ .

**end**

**end**

# Análisis sintáctico ascendente con oráculo

**Ejercicio.** Usar el algoritmo de análisis sintáctico ascendente con la gramática  $G = (\{E, T, F\}, \{+, *, \mathbf{n}, (, )\}, P, E)$ :

$$\begin{array}{lcl} E & \rightarrow & T \mid E + T \\ T & \rightarrow & F \mid T * F \\ F & \rightarrow & \mathbf{n} \mid (E) \end{array}$$

para analizar la cadena  $\mathbf{n} * \mathbf{n} + \mathbf{n}$ .



## Análisis sintáctico ascendente

- Notar que las acciones **reduce** por distintas producciones no son mutuamente excluyentes *a priori*. Por ejemplo, la cadena  $aa$  con la gramática (ambigua)  $G_1 = (\{S, A, B\}, \{a, b\}, P, S)$ :

$$S \rightarrow Aa \mid Bb$$

$$A \rightarrow a$$

$$B \rightarrow a$$

- Notar que las acciones **shift** y **reduce** no son mutuamente excluyentes *a priori*. Por ejemplo, la cadena  $aab$  con la gramática (no ambigua)  $G_2 = (\{S, A\}, \{a, b\}, P, S)$ :

$$S \rightarrow aAb \mid Ab$$

$$A \rightarrow a$$

## Análisis sintáctico ascendente

En el caso anterior de la gramática  $G_2 = (\{S, A\}, \{a, b\}, P, S)$ :

$$\begin{aligned} S &\rightarrow aAb \mid Ab \\ A &\rightarrow a \end{aligned}$$

vemos que para la cadena  $aab$  no corresponde hacer un **reduce**  
 $A \rightarrow a$  para la primera  $a$ .

Podemos refinar el criterio para aplicar una acción **reduce**.

# Análisis sintáctico ascendente

- ▶ Si  $S \Rightarrow^* \alpha A \gamma \Rightarrow \alpha \beta \gamma$  es una derivación más a la derecha, se dice que la producción  $A \rightarrow \beta$  junto con la posición  $|\alpha|$  en la que se aplica es un **pivote** de  $\alpha \beta \gamma$ .
- ▶ Si el analizador sintáctico ascendente está en la siguiente situación:
  1. la pila es de la forma  $\alpha \beta$ ,
  2. el fragmento de la entrada todavía no consumido es una cadena  $\gamma$ ,

el parser debe aplicar una acción **reduce** solamente si hay una producción  $A \rightarrow \beta$  que es un pivote para  $\alpha \beta \gamma$  en la posición  $|\alpha|$ .

# Análisis sintáctico LR

- ▶ La clave de los parsers de la familia **LR** es que los pivotes se pueden identificar usando un autómata finito.
- ▶ Usando el autómata se puede decidir si corresponde ejecutar una acción **shift** o **reduce**.
- ▶ Veremos primero la técnica de análisis sintáctico LR(0), es decir, sin *lookahead*.

# Análisis sintáctico LR(0)

- ▶ Si  $G = (N, \Sigma, P, S)$  es una gramática, llamamos **ítem** a una producción que tiene un “puntito” en algún lado, es decir a un elemento de la forma  $A \rightarrow \alpha \bullet \beta$  donde  $(A \rightarrow \alpha\beta) \in P$  es una producción.
- ▶ Formalmente se puede modelar como un par  $(A \rightarrow \alpha, i)$  donde  $A \rightarrow \alpha$  es una producción. y  $0 \leq i \leq |\alpha|$  es la posición del puntito.
- ▶ Para implementar el analizador sintáctico, un ítem se puede representar como un par de números  $(j, i)$  donde  $j$  identifica la producción (e  $i$  la posición del puntito).
- ▶ Por ejemplo, la producción  $E \rightarrow T + E$  tiene cuatro ítems asociados:

$$E \rightarrow \bullet T + E$$

$$E \rightarrow T \bullet + E$$

$$E \rightarrow T + \bullet E$$

$$E \rightarrow T + E \bullet$$

# Análisis sintáctico LR(0)

## Clausura de un conjunto de ítems.

Si  $I$  es un conjunto de ítems, escribimos  $\text{CLAUSURA}(I)$ , y llamamos **clausura de  $I$** , al conjunto de ítems tal que:

1.  $I \subseteq \text{CLAUSURA}(I)$
2. Siempre que  $\text{CLAUSURA}(I)$  incluye un ítem  $A \rightarrow \alpha \bullet B\beta$ , también incluye a todos los ítems de la forma  $B \rightarrow \bullet \gamma$ .
3.  $\text{CLAUSURA}(I)$  es el conjunto de ítems más chico que verifica las condiciones 1. y 2.

Se puede calcular de manera sencilla siguiendo la definición:

*Entrada:* Una gramática  $G = (N, \Sigma, P, S)$ ,  
un conjunto de ítems  $I$ .

*Salida:* El conjunto de ítems  $\text{CLAUSURA}(I)$ .

$J := I$

**while** hay cambios

**foreach** ítem  $(A \rightarrow \alpha \bullet B\beta) \in J$

$J := J \cup \{(B \rightarrow \bullet \gamma) \mid (B \rightarrow \gamma) \in P\}$

**end**

**end**

**return**  $J$

# Análisis sintáctico LR(0)

**Ejercicio.** Calcular la clausura de  $\{E' \rightarrow \bullet E\}$  en la gramática  $G = (\{E', E, T\}, \{\mathbf{n}, +, (, ), \$\}, P, S)$ :

$$\begin{aligned} E' &\rightarrow E\$ \\ E &\rightarrow T \mid E + T \\ T &\rightarrow \mathbf{n} \mid (E) \end{aligned}$$

# Análisis sintáctico LR(0)

## Construcción de la tabla de análisis sintáctico LR(0).

- ▶ Una tabla de análisis sintáctico LR(0) es esencialmente un **autómata finito** determinístico en el alfabeto  $N \cup \Sigma$  (símbolos terminales y no terminales).
- ▶ Sirve para detectar cuándo hay un pivote en la pila, es decir, funciona como un oráculo que indica si corresponde hacer *shift* o *reduce*.
- ▶ Los estados son **conjuntos de ítems**.
- ▶ Vamos a suponer que la gramática está extendida con un símbolo inicial  $S'$ , un símbolo  $\$$  y una producción  $S' \rightarrow S\$$ .
- ▶ La función de transición del autómata tradicionalmente se separa en dos partes:
  - ▶ A cada estado  $q$  y cada símbolo no terminal  $A \in N$ , la tabla asocia un estado  $\text{GOTO}[q, A]$ .
  - ▶ A cada estado  $q$  y cada símbolo terminal  $a \in \Sigma$ , la tabla asocia una acción  $\text{ACTION}[q, a]$ . Las acciones posibles son:
    - ▶ **Shift**( $q$ ) donde  $q$  es un estado.
    - ▶ **Reduce**( $A \rightarrow \alpha$ ) donde  $A \rightarrow \alpha$  es una producción.
    - ▶ **Accept**.



# Análisis sintáctico LR(0)

## Construcción de la tabla de análisis sintáctico LR(0).

*Entrada:* Una gramática  $G = (N, \Sigma, P, S)$ .

*Salida:* Una tabla de análisis sintáctico LR(0).

$q_0 := \text{CLAUSURA}(\{S' \rightarrow \bullet S\})$

$Q := \{q_0\}$

**while** hay un estado  $q \in Q$  no visitado

    Marcar  $q$  como visitado.

**foreach** símbolo  $x \in N \cup \Sigma$

$q' := \text{CLAUSURA}(\{(A \rightarrow \alpha x \bullet \beta) \mid (A \rightarrow \alpha \bullet x \beta) \in q\})$

$Q := Q \cup \{q'\}$

**if**  $x$  es un símbolo no terminal

$\text{GOTO}[q, x] := q'$

**else**

**if**  $x = \$$  and  $(S' \rightarrow S \bullet \$) \in q$

$\text{ACTION}[q, x] := \text{Accept}$

**if** hay algún ítem  $(A \rightarrow \alpha \bullet) \in q$

$\text{ACTION}[q, x] := \text{Reduce}(A \rightarrow \alpha)$

**if** hay algún ítem  $(A \rightarrow \alpha \bullet x \beta) \in q$

$\text{ACTION}[q, x] := \text{Shift}(q')$

**end**

**end**

**end**

# Análisis sintáctico LR(0)

- ▶ Si a cada entrada de la tabla ACTION le corresponde a lo sumo una acción, se dice que la gramática es LR(0).
- ▶ Si alguna entrada de la tabla ACTION tiene dos o más acciones, la gramática no es LR(0) y se dice que hay un conflicto.

# Análisis sintáctico LR(0)

**Ejercicio.** Construir la tabla de análisis sintáctico LR(0) para la gramática  $G = (\{E', E, T\}, \{\mathbf{n}, +, (, ), \$\}, P, S)$ :

$$\begin{aligned} E' &\rightarrow E\$ \\ E &\rightarrow T \mid E + T \\ T &\rightarrow \mathbf{n} \mid (E) \end{aligned}$$

# Análisis sintáctico LR(0)

La tabla de análisis sintáctico LR(0) puede tener conflictos **reduce/reduce**.

Esto se da cuando hay dos ítems de la forma  $A \rightarrow \alpha \bullet$  en el mismo estado. Por ejemplo, para  $G = (\{S', S, A\}, \{a, \$\}, P, S')$ :

$$S' \rightarrow S\$$$

$$S \rightarrow a \mid A$$

$$A \rightarrow a$$

# Análisis sintáctico LR(0)

La tabla también puede tener conflictos **shift/reduce**.

Esto se da cuando hay un ítem de la forma  $A \rightarrow \alpha \bullet$  y un ítem que no tiene esa forma en el mismo estado. Por ejemplo, para  $G = (\{S', S, A\}, \{a, b, \$\}, P, S')$ :

$$\begin{array}{lcl} S' & \rightarrow & S\$ \\ S & \rightarrow & A \mid Ab \\ A & \rightarrow & a \mid ab \end{array}$$

(El conflicto evidencia una ambigüedad en la gramática).

## Análisis sintáctico LR(0)

Los conflictos generalmente corresponden a una ambigüedad en la gramática, pero pueden ser un artefacto de la construcción:

Por ejemplo, para  $G = (\{S', S, A\}, \{a, b\}, P, S')$ :

$$\begin{array}{lcl} S' & \rightarrow & S \\ S & \rightarrow & a \mid Ab \\ A & \rightarrow & a \end{array}$$

(Esta gramática no es ambigua pero tiene un conflicto).

# Análisis sintáctico LR(0)

## Algoritmo de análisis sintáctico LR.

*Entrada:* Una gramática  $G = (N, \Sigma \cup \{\$, \}, P, S')$  y una cadena  $\alpha$ .

*Salida:* Un booleano indicando si  $\alpha\$ \in L(G)$ ; en tal caso, una derivación más a la derecha para  $S' \Rightarrow^* \alpha\$$ .

```
pila := [q0] // pila de estados de la tabla LR(0)
a := primer símbolo de la entrada α$
repeat forever
  q := pila.tope()
  if ACTION[q, a] = Shift(q')
    pila.push(q')
    a := siguiente símbolo de la entrada
  elseif ACTION[q, a] = Reduce(A → β)
    Sacar |β| símbolos de pila.
    q' := pila.tope()
    pila.push(GOTO[q', A])
    Agregar la producción A → β como primer paso.
  elseif ACTION[q, a] = Accept
    return Éxito: con la derivación construida.
  else return Falla: error de sintaxis.
end
```

## Análisis sintáctico LR(0)

**Ejercicio.** Analizar sintácticamente  $n + (n + n)$  con la gramática  $G = (\{E', E, T\}, \{n, +, (, )\}, P, S)$ :

$$E' \rightarrow E\$ \qquad E \rightarrow T \mid E + T \qquad T \rightarrow n \mid (E)$$

usando la tabla de análisis sintáctico LR(0) ya construida:

	$E$	$T$	$n$	$($	$)$	$+$	$\$$
1	2	4	<b>Shift(9)</b>	<b>Shift(7)</b>			
2						<b>Shift(5)</b>	<b>Accept</b>
3			$T \rightarrow (E)$	$T \rightarrow (E)$	$T \rightarrow (E)$	$T \rightarrow (E)$	$T \rightarrow (E)$
4			$E \rightarrow T$	$E \rightarrow T$	$E \rightarrow T$	$E \rightarrow T$	$E \rightarrow T$
5		6	<b>Shift(9)</b>	<b>Shift(7)</b>			
6			$E \rightarrow E + T$	$E \rightarrow E + T$	$E \rightarrow E + T$	$E \rightarrow E + T$	$E \rightarrow E + T$
7	10	8	<b>Shift(9)</b>	<b>Shift(7)</b>			
8			$E \rightarrow T$	$E \rightarrow T$	$E \rightarrow T$	$E \rightarrow T$	$E \rightarrow T$
9			$T \rightarrow n$	$T \rightarrow n$	$T \rightarrow n$	$T \rightarrow n$	$T \rightarrow n$
10					<b>Shift(3)</b>	<b>Shift(5)</b>	



# Análisis sintáctico SLR

- ▶ El algoritmo SLR es una ligera variante de LR(0).
- ▶ Las acciones **reduce** no ocupan toda la “fila” de la tabla SLR.
- ▶ Solamente se reduce por una producción  $A \rightarrow \beta$  para los símbolos terminales que pueden aparecer a continuación de  $A$ .
- ▶ Más precisamente, se pone  $\text{ACTION}[q, a] = \text{Reduce}(A \rightarrow \beta)$  solamente cuando  $a \in \text{FOLLOW}(A)$ .
- ▶ La ventaja es que esto puede evitar conflictos **shift/reduce** y conflictos **reduce/reduce**.

# Análisis sintáctico SLR

**Ejercicio.** Ya vimos que la tabla LR(0) para la gramática  $G = (\{S', S, A\}, \{a, b, \$\}, P, S')$ :

$$\begin{array}{lcl} S' & \rightarrow & S\$ \\ S & \rightarrow & a \mid Ab \\ A & \rightarrow & a \end{array}$$

tenía un conflicto **reduce/reduce**.

Ver que la tabla SLR para  $G$  no tiene conflictos.

# Análisis sintáctico SLR

**Ejercicio.** ¿Cómo se modificaría la tabla LR(0) para la gramática de expresiones estudiada hace algunas diapositivas en el caso SLR?

# Análisis sintáctico SLR

Las gramáticas SLR no son ambiguas.

Pero hay gramáticas no ambiguas que no son SLR. Por ejemplo, considerar  $G = (\{S, L, R\}, \{\mathbf{id}, =, *\}, P, S)$ :

$$\begin{aligned} S &\rightarrow L = R \mid R \\ L &\rightarrow *R \mid \mathbf{id} \\ R &\rightarrow L \end{aligned}$$

- ▶ La entrada  $\mathbf{id} = \mathbf{id}$  primero consume  $\mathbf{id}$  y lo reduce a  $L$ .
- ▶ Acá se alcanza un conflicto **shift/reduce**:
  - ▶ ¿Se debe consumir  $=$ ?
  - ▶ ¿O se debe reducir  $R \rightarrow L$ ?
  - ▶ Observar que el símbolo  $=$  está en  $\text{FOLLOW}(R)$  de modo que el conflicto se da incluso en la gramática no es SLR.
  - ▶ Por otro lado, observar que no tiene sentido reducir  $R \rightarrow L$ .

# Análisis sintáctico LR(1) canónico

- ▶ Otro algoritmo de la familia de LR es el algoritmo **LR(1) canónico**.
- ▶ Usa un token de *lookahead*.
- ▶ La idea es refinar el autómata: si el siguiente símbolo de la entrada es  $a$ , sólo se debe reducir por una producción  $A \rightarrow \beta$  cuando  $a$  podría venir a continuación de  $A$ .

# Análisis sintáctico LR(1) canónico

- ▶ Un **ítem LR(1)** es un par  $[A \rightarrow \alpha \bullet \beta, a]$  donde  $A \rightarrow \alpha\beta$  es una producción y  $a \in \Sigma \cup \{\$ \}$  es un símbolo terminal o \$.
- ▶ Los estados de la tabla de análisis sintáctico LR(1) son ahora conjuntos de ítems LR(1).
- ▶ Nos limitaremos a presentar la construcción de la tabla LR(1). Las modificaciones clave son:
  - ▶ La definición de ítem.
  - ▶ La clausura de un conjunto de ítems.

# Análisis sintáctico LR(1) canónico

## Clausura de un conjunto de ítems LR(1).

El cálculo de la clausura se modifica como sigue:

*Entrada:* Una gramática  $G = (N, \Sigma, P, S)$ ,  
un conjunto de ítems LR(1)  $I$ .

*Salida:* El conjunto  $\text{CLAUSURA}(I)$ .

```
 $J := I$ 
while hay cambios
    foreach ítem  $[A \rightarrow \alpha \bullet B\beta, a] \in J$ 
        foreach producción  $B \rightarrow \gamma$ 
            foreach símbolo terminal  $b \in \text{FIRST}(\beta a)$ 
                 $J := J \cup \{[B \rightarrow \bullet \gamma, b]\}$ 
            end
        end
    end
end
return  $J$ 
```

# Análisis sintáctico LR(1) canónico

**Ejercicio.** Calcular la tabla de análisis sintáctico LR(1) canónico para la gramática  $G = (\{S', S, A\}, \{a, b\}, P, S')$ :

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow AA \\ A &\rightarrow aA \mid b \end{aligned}$$

Analizar sintácticamente la cadena *aabab*.



# Análisis sintáctico LALR(1)

- ▶ El algoritmo LR(1) canónico es muy poderoso<sup>1</sup>, pero el tamaño de las tablas de análisis sintáctico puede ser exponencial.
- ▶ El núcleo de un conjunto de ítems es el subconjunto de los ítems que **no** son de la forma  $[A \rightarrow \bullet\beta, a]$ .
- ▶ La explosión del tamaño de las tablas se puede controlar unificando todos los estados que tienen el mismo núcleo.
- ▶ En este caso la tabla de análisis sintáctico se conoce como **LALR(1)**.
- ▶ Hay algoritmos para construir directamente la tabla LALR(1) sin pasar por medio de la tabla LR(1).

---

<sup>1</sup>La clase de gramáticas LR(1) incluye a las LR( $k$ ) y a las LL( $k$ ) para todo  $k$ .

# Análisis sintáctico LALR(1)

**Ejemplo.** Unificar los estados que comparten el mismo núcleo y obtener la tabla de análisis sintáctico LALR(1) para la gramática  $G = (\{S', S, A\}, \{a, b\}, P, S')$ :

$$\begin{array}{lcl} S' & \rightarrow & S \\ S & \rightarrow & AA \\ A & \rightarrow & aA \mid b \end{array}$$