

Ítem LR(1)

Un ítem LR(1) es un elemento de la forma

$$[A \rightarrow \alpha.\beta, a]$$

donde $A \rightarrow \alpha\beta \in P$, $a \in V_T$

- Representa un posible estado del parser.
- Lleva información adicional para evitar reducciones erróneas.
- Usamos un token de lookahead que indique exactamente que símbolos de entrada pueden seguir a un handle.
- Nuestros estados del autómata consisten de un conjuntos de estos posibles estados.

Se dice que un ítem LR(1) $[A \rightarrow \alpha.\beta, a]$ es válido para un prefijo viable $\delta\alpha$ si existe una derivación más a la derecha $S \xRightarrow{*}_d \delta A a w \Rightarrow_d \delta \alpha \beta a w$

Clausura de ítems LR(1)

```

 $J \leftarrow I;$ 
repeat
  | for ítem  $[A \rightarrow \alpha \cdot B\beta, a] \in J$  y cada producción  $B \rightarrow \gamma \in G'$  do
  |   | agregar  $[B \rightarrow \cdot \gamma, c]$  a  $J$  para cada  $c \in \text{Primeros}(\beta a);$ 
  | end
until No cambia  $J;$ 
return  $J;$ 

```

Observemos que nos guardamos sólo aquellos posibles no terminales que puedan aparecer, no todo $\text{Siguietes}(B)$. Vale que:

$$\text{Primeros}(\beta a) = \begin{cases} \text{Primeros}(\beta) \cup \{a\} & \text{Si } \beta \text{ es anulable} \\ \text{Primeros}(\beta) & \text{Si no} \end{cases}$$

Autómata, tablas y algoritmo de parsing

- Notación: si tenemos en un mismo estado a los ítems $[A \rightarrow \alpha \cdot \beta, a]$ y $[A \rightarrow \alpha \cdot \beta, b]$ los podemos escribir agrupados como $[A \rightarrow \alpha \cdot \beta, a/b]$
- Las transiciones en el autómata se arman como antes, pero si ya tenía el ítem $[A \rightarrow \alpha \cdot \beta, a]$ en un estado y ahora tengo que transicionar al $[A \rightarrow \alpha \cdot \beta, b]$ (con $a \neq b$) tengo que ir a un nuevo estado.
- Reducimos por $A \rightarrow \beta$ en $\text{Action}[i][a]$ sii en el estado i tenemos al ítem $[A \rightarrow \beta \cdot, a]$
- El algoritmo de parsing se mantiene igual.

Ejercicio

Armar la tabla LR(1) para:

$$S' \rightarrow S$$

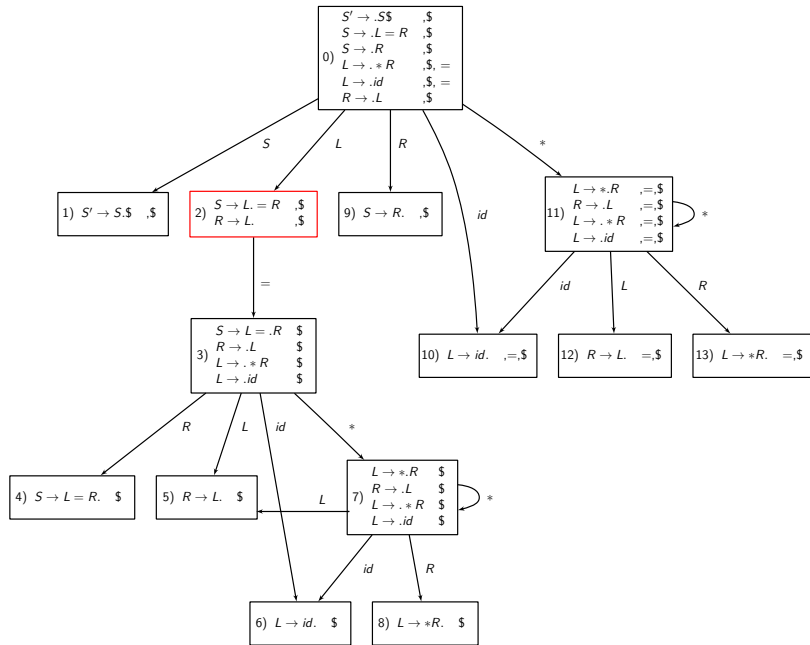
$$S \rightarrow L = R$$

$$S \rightarrow R$$

$$L \rightarrow *R$$

$$L \rightarrow id$$

$$R \rightarrow L$$



Tablas Action y Goto

	=	*	id	\$	S	L	R
0		s11	s10		1	2	9
1				acepto			
2	s3			$r R \rightarrow L$			
3		s7	s6			5	4
4				$r S \rightarrow L = R$			
5				$r R \rightarrow L$			
6				$r L \rightarrow id$			
7		s7	s6			5	8
8				$r L \rightarrow *R$			
9				$r S \rightarrow R$			
10	$r L \rightarrow id$			$r L \rightarrow id$			
11		s11	s10			12	13
12	$r R \rightarrow L$			$r R \rightarrow L$			
13	$r L \rightarrow *R$			$r L \rightarrow *R$			

LALR

- LR(1) no es práctico porque al codificar la información contextual el autómata (y por ende las tablas) quedan demasiado grandes.
- LALR es un intermedio entre SLR y LR(1). LALR tiene la misma cantidad de estados que SLR pero mantiene un contexto.
 - ▶ $LR(1) \supset LALR \supset SLR \supset LR(0)$
- La mayoría de las gramáticas LR(1) son LALR.
- LALR acepta las gramáticas de la mayoría de los lenguajes de programación.
- LALR une los estados cuyos ítems tienen los mismos cores (alcanza con mirar los ítems kernel).
 - ▶ A cada nuevo ítem fusionado se le asigna la unión de los lookahead existentes.
- Se actualiza la tabla con los estados que fueron fusionados.

Ejercicio

Armar la tabla LALR para la misma gramática de antes:

$$S' \rightarrow S$$

$$S \rightarrow L = R$$

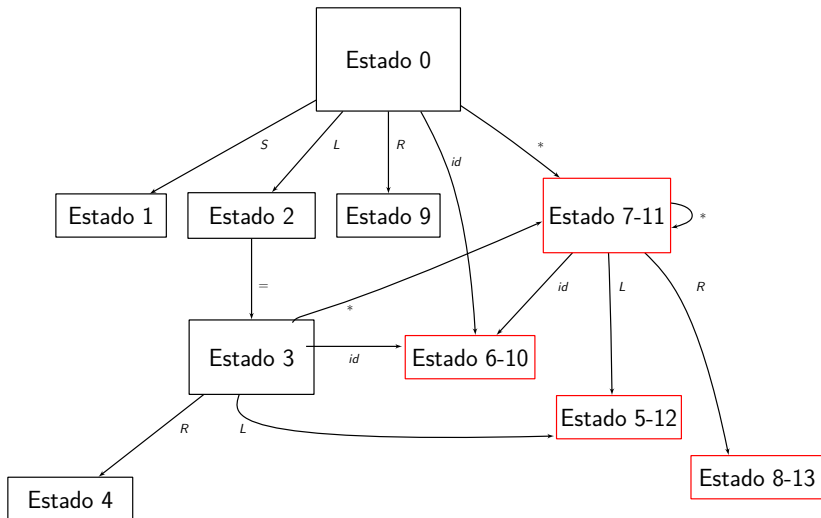
$$S \rightarrow R$$

$$L \rightarrow *R$$

$$L \rightarrow id$$

$$R \rightarrow L$$

Autómata LALR resultante



Tablas Action y Goto

	=	*	id	\$	S	L	R
0		s7-11	s6-10		1	2	9
1				acepto			
2	s3			$r R \rightarrow L$			
3		s7-11	s6-10			5-12	4
4				$r S \rightarrow L = R$			
5-12	$r R \rightarrow L$			$r R \rightarrow L$			
6-10	$r L \rightarrow id$			$r L \rightarrow id$			
7-11		s7-11	s6-10			5-12	8-13
8-13	$r L \rightarrow *R$			$r L \rightarrow *R$			
9				$r S \rightarrow R$			

Conflictos Shift/Reduce

- La fusión de estados LR(1) no puede introducir un nuevo conflicto Shift/Reduce, porque ya hubiera ocurrido anteriormente, por el core.
- Las acciones de Shift sólo dependen del core, no del lookahead.

Conflictos Reduce/Reduce

- La fusión de estados LR(1) puede introducir nuevos conflictos de tipo Reduce/Reduce.
- Por ejemplo, dados los siguientes grupos de ítems:

$$l_1 = \begin{array}{l} A \rightarrow \alpha \cdot, a \\ B \rightarrow \beta \cdot, b \end{array}$$

$$l_2 = \begin{array}{l} A \rightarrow \alpha \cdot, c \\ B \rightarrow \beta \cdot, a \end{array}$$

- En LALR se agrupa los estados l_1 e l_2 por tener mismo core y se obtiene

$$l_{1-2} = \begin{array}{l} A \rightarrow \alpha \cdot, a/c \\ B \rightarrow \beta \cdot, a/b \end{array}$$

- Lo que genera un conflicto R/R, que antes no se tenía, en Action[1-2][a]

Conclusiones

- LR(1) es poderoso porque lleva información de contexto, pero esto lo hace muy grande.
- LALR mantiene parte de esa información logrando reducir el tamaño de la tabla a lo mismo que SLR, con la desventaja que en ocasiones puede introducir nuevos conflictos.
- A los fines prácticos, no se arma el automáta LALR basandose en el LR(1) ya que esto exige más tiempo y espacio del necesario. En la teórica se mencionaron 4 algoritmos más eficientes.
- $\text{Gram} \supset \text{Gram no ambiguas} \supset \text{LR}(1) \supset \text{LALR} \supset \text{SLR} \supset \text{LR}(0)$