LR(1) - LALR

Teoría de Lenguajes

2do. cuatrimestre 2016

Repaso

- Parsing bottom-up
- Parsing shift-reduce
- Prefijos viables. Autómata para los prefijos viables
- Tablas Action y Goto
- Algoritmo de parsing
- Conflictos
- LR(0)
- SLR

Ítem LR(1)

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

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

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

- 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) $[A \to \alpha.\beta, a]$ es válido para un prefijo viable $\delta\alpha$ si existe una derivación más a la derecha $S \overset{*}{\underset{d}{\Rightarrow}} \delta A a \omega \underset{d}{\Rightarrow} \delta\alpha\beta a\omega$

Clausura de ítems LR(1)

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

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

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

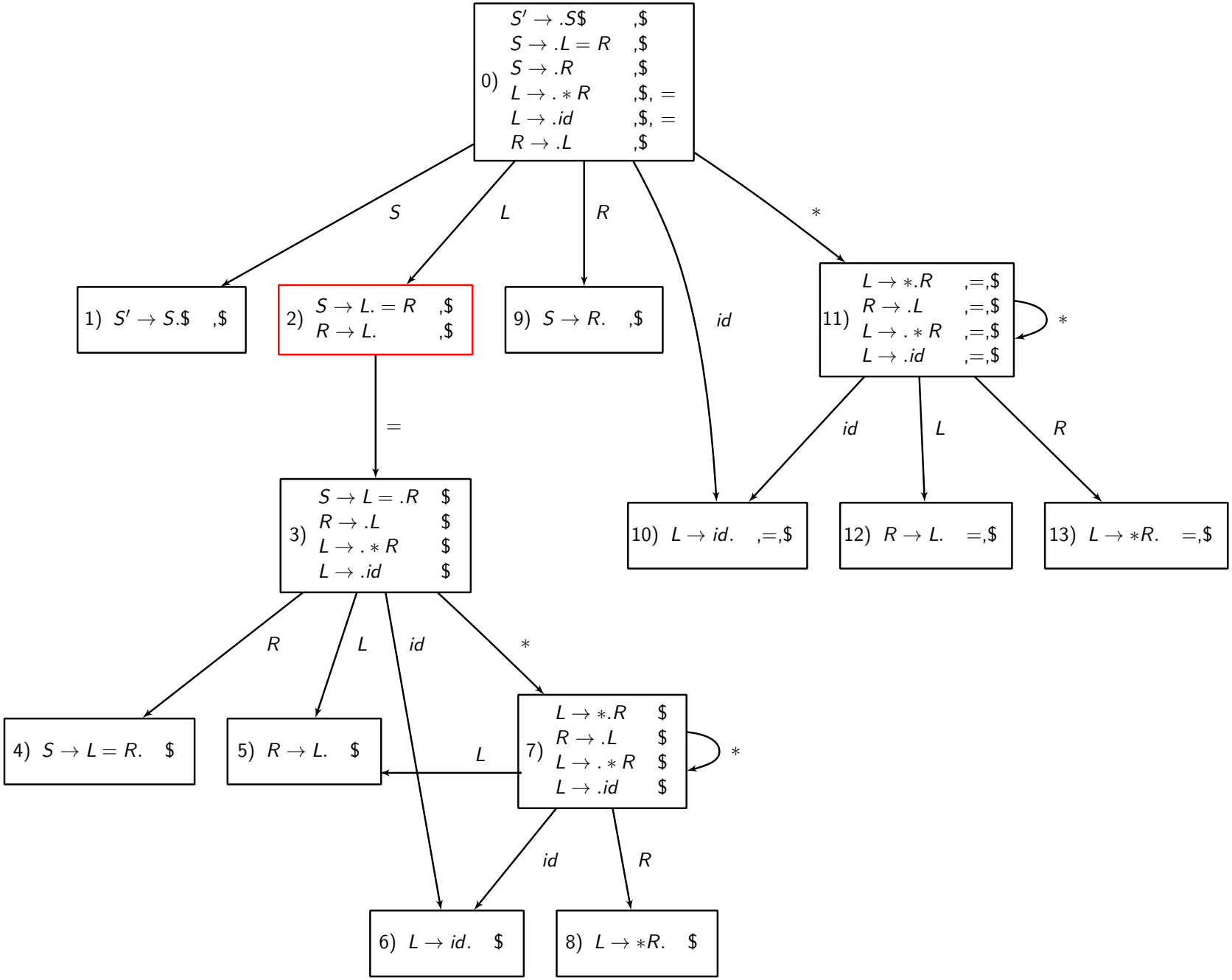
Autómata, tablas y algoritmo de parsing

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

Ejercicio

Armar la tabla LR(1) para:

$$
\begin{aligned}
S' &\to S \\
S &\to L = R \\
S &\to R \\
L &\to *R \\
L &\to id \\
R &\to L
\end{aligned}
$$


0) $S' \to .S\$$,$
$S \to .L = R$,$
$S \to .R$,$
$L \to . * R$,$, =
$L \to .id$,$, =
$R \to .L$,$
S
L
R
*
id
1) $S' \to S.\$$,$
2) $S \to L. = R$,$
$R \to L.$,$
9) $S \to R.$,$
11) $L \to *.R$,=,$
$R \to .L$,=,$
$L \to . * R$,=,$
$L \to .id$,=,$
=
id
L
R
*
3) $S \to L = .R$ $
$R \to .L$ $
$L \to . * R$ $
$L \to .id$ $
10) $L \to id.$,=,$
12) $R \to L.$ =,$
13) $L \to *R.$ =,$
R
L
id
*
4) $S \to L = R.$ $
5) $R \to L.$ $
7) $L \to *.R$ $
$R \to .L$ $
$L \to . * R$ $
$L \to .id$ $
L
*
id
R
6) $L \to id.$ $
8) $L \to *R.$ $

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

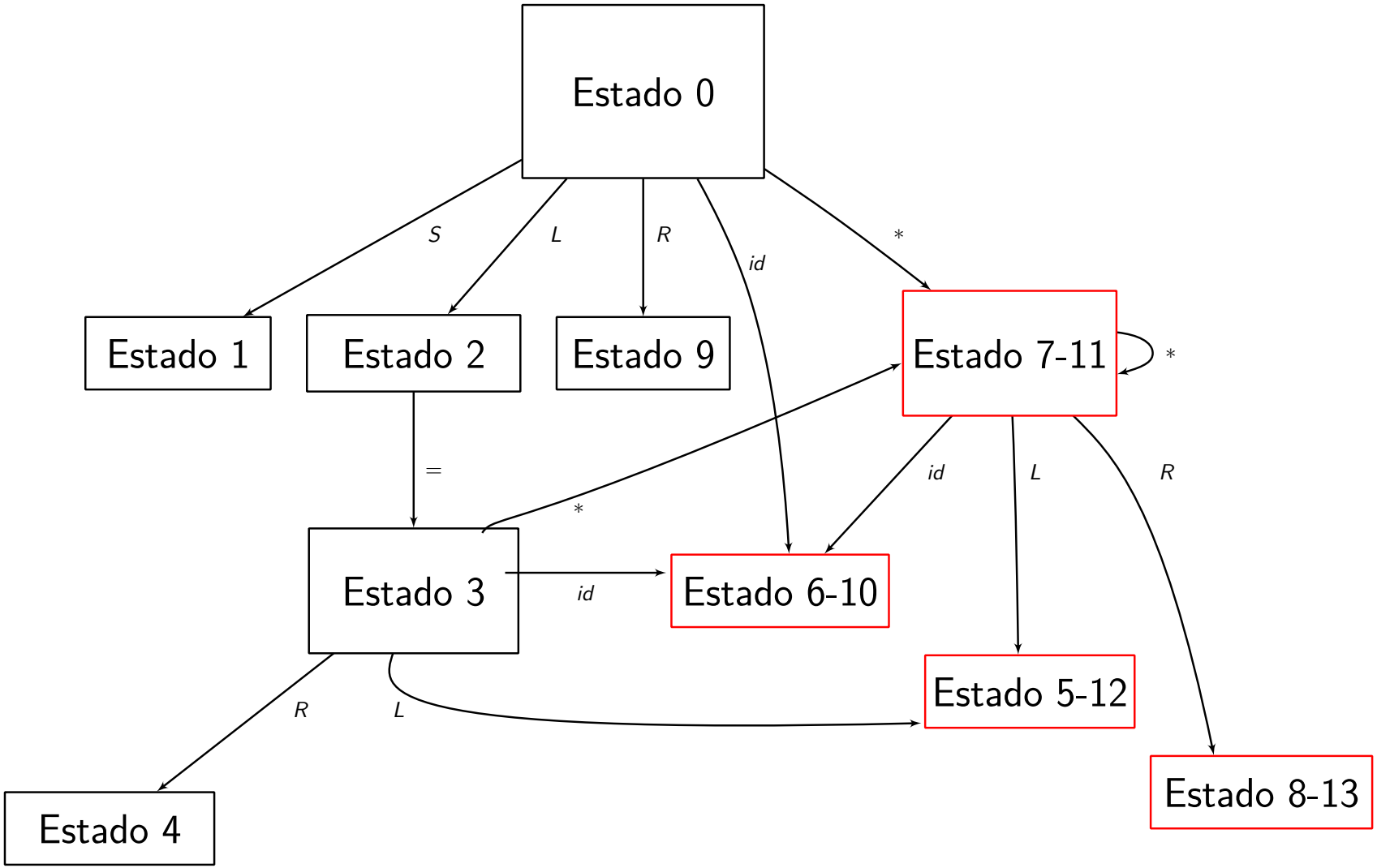
- LR(1) no es práctico porque al codificar la información contextual el autómata (y por ende las tablas) quedan demasiado grandes.
- LALR es un intermedio entre SLR y LR(1). LALR tiene la misma cantidad de estados que SLR pero mantiene un contexto.
  - LR(1) $\supset$ LALR $\supset$ SLR $\supset$ LR(0)
- 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:

$$
\begin{aligned}
S' &\to S \\
S &\to L = R \\
S &\to R \\
L &\to *R \\
L &\to id \\
R &\to L
\end{aligned}
$$

Autómata LALR resultante



Tablas Action y Goto

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

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:

$$I_1 = \begin{array}{ll} A \to \alpha\cdot & , a \\ B \to \beta\cdot & , b \end{array} \qquad I_2 = \begin{array}{ll} A \to \alpha\cdot & , c \\ B \to \beta\cdot & , a \end{array}$$

- En LALR se agrupa los estados I_1 e I_2 por tener mismo core y se obtiene

$$I_{1-2} = \begin{array}{ll} A \to \alpha\cdot & , a/c \\ B \to \beta\cdot & , a/b \end{array}$$

- 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.
- Gram $\supset$ Gram no ambiguas $\supset$ LR(1) $\supset$ LALR $\supset$ SLR $\supset$ LR(0)

¿Preguntas?

¿Preguntas?