

Compiladores 24-2

Análisis Sintáctico: parser bottom-up LALR

Lourdes del Carmen González Huesca

luglzhuesca@ciencias.unam.mx

Facultad de Ciencias, UNAM

2 abril 2024



Análisis sintáctico

Parser bottom-up

- La clase de gramáticas **LR** corresponden a analizadores que construyen un árbol de sintaxis concreta desde las hojas y hacia la raíz.
- Los analizadores utilizan una tabla de parsing que decide las acciones y está generada por una autómata finito entre conjuntos de items.
- La técnica para generar los árboles se llama *shift-reduce* donde la reducción se realiza para obtener un proceso inverso de una derivación paso a paso.

Estudiaremos las variantes de construcción de la tabla:

- LR(0)
- SLR *simple LR*
- LR(1) *canonical-LR*
- LALR *lookahead-LR*

Análisis sintáctico LALR

técnica look-ahead LR

- Método de parsing más usado en la práctica ya que las tablas de parsing son más pequeñas:
 - ✓ hacer más compacto el autómata finito al **unir** estados que tienen el mismo núcleo o *core*
 - ✓ las acciones *shift* dependen sólo del núcleo de un estado

Definición (Core de un estado)

El núcleo o core de un estado es el conjunto de items donde la componente izquierda es la misma, es decir los lookaheads pueden ser diferentes.

Análisis sintáctico LALR

técnica look-ahead LR

- Método de parsing más usado en la práctica ya que las tablas de parsing son más pequeñas:
 - ✓ hacer más compacto el autómata finito al **unir** estados que tienen el mismo núcleo o *core*
 - ✓ las acciones *shift* dependen sólo del núcleo de un estado

Definición (Core de un estado)

El núcleo o core de un estado es el conjunto de items donde la componente izquierda es la misma, es decir los lookaheads pueden ser diferentes.

- La unión de estados no puede producir conflictos de tipo shift/reduce si la gramática ya era de clase LR(1).
 - Los estados del AF son conjuntos (canónicos) de items usando la cerradura y comenzando por $[S' \rightarrow \bullet S, \#]$.
 - Las transiciones entre estados son calculadas por la función GoTo.

Análisis sintáctico LALR

algoritmo para calcular tabla

Input: Una gramática aumentada con un nuevo símbolo inicial S'

Output: La tabla de parsing LALR con las funciones ACTION and GoTo para la gramática aumentada.

1. Construir la colección de conjuntos de items según el método LR(1)
 $C = \{I_0, I_1, \dots, I_n\}$.
2. Para cada núcleo de los conjuntos en C , encontrar todos los conjuntos que compartan el mismo núcleo y reemplazarlos por la unión de ellos.
3. Sea $C' = \{J_0, J_1, \dots, J_m\}$ la colección resultante del paso 2.
Las acciones para el estado j están construidas desde J_j en la misma forma que el método para LR(1) (ver algoritmo para LR(1)).
Si hay un conflicto entonces el algoritmo falla y la gramática no pertenece a la clase LALR(1).
4. La parte de la tabla GoTo se construye de la siguiente forma:
Si J es la unión de núcleos de uno o más conjuntos de items,
 $J = I_0 \cap I_1 \cap \dots \cap I_k$ entonces los cores de
 $\text{GoTo}(I_1, X)$, $\text{GoTo}(I_2, X)$, \dots , $\text{GoTo}(I_k, X)$, son el mismo, dado que comparten las componentes izquierdas.
Entonces K será la unión de los conjuntos de items que tienen el mismo core que $\text{GoTo}(I_1, X)$ y es justo $K = \text{GoTo}(J, X)$.

Análisis sintáctico LALR

ejemplo

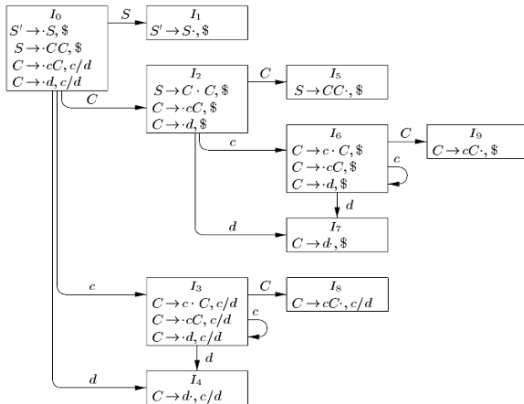
Considere la gramática

$$S' \rightarrow S$$

$$S \rightarrow CC$$

$$C \rightarrow cC \mid d$$

y el autómata construido con el método **LR(1)**:



Análisis sintáctico LALR

ejemplo

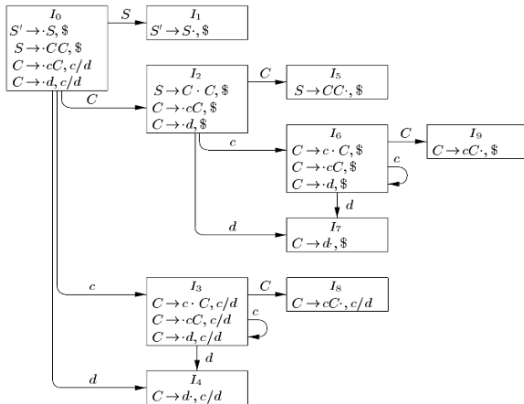
Considere la gramática

$$S' \rightarrow S$$

$$S \rightarrow CC$$

$$C \rightarrow cC \mid d$$


y el autómata construido con el método **LR(1)**:



STATE	ACTION			GOTO	
	<i>c</i>	<i>d</i>	<i>\$</i>	<i>S</i>	<i>C</i>
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Análisis sintáctico LALR

Construcción eficiente de la tabla para LALR

Usar los estados de un autómata siguiendo el método **LR(1)** y sólo tomar los núcleos (proceso largo) 

Análisis sintáctico LALR

Construcción eficiente de la tabla para LALR

Usar los estados de un autómata siguiendo el método **LR(1)** y sólo tomar los núcleos (proceso largo) ✗

- ✓ Evitar construir todos los conjuntos de items:
seguir el método de **LR(0)** y calcular los lookaheads ya sea espontáneamente o propagarlos:
 - 1) Primero es necesario calcular los conjuntos de items según el método de **LR(0)**, es decir mediante conjuntos canónicos usando la función CLOSURE para los estados y la función GOTO para las transiciones siguiendo las definiciones dadas.
 - 2) Después se deben identificar los items que sean tipo kernel para cada estado, siguiendo la definición correspondiente.
 - 3) Completar los items con los lookaheads apropiados y así generar los kernels de LALR con lookaheads espontáneos o propagados.

Análisis sintáctico LALR

lookaheads apropiados

Completar los items con los lookaheads apropiados y así generar los kernels de LALR.

Para esto existen dos tipos de lookaheads:

Análisis sintáctico LALR

lookaheads apropiados

Completar los items con los lookaheads apropiados y así generar los kernels de LALR.

Para esto existen dos tipos de lookaheads:

1. Lookaheads espontáneos

Sea I un conjunto de items con kernel $[A \rightarrow \alpha \bullet \beta, a]$ y $J = \text{GoTo}(I, X)$ para cualquier símbolo de la gramática X , b será un lookahead espontáneo donde $\text{GoTo}(\text{CLOSURE}(\{[A \rightarrow \alpha \bullet \beta, a]\}), X)$ contiene al item $[B \rightarrow \gamma \bullet \delta, b]$

El lookahead espontáneo para el item $S' \rightarrow \bullet S$ es $\#$.

Análisis sintáctico LALR

lookaheads apropiados

Completar los items con los lookaheads apropiados y así generar los kernels de LALR.

Para esto existen dos tipos de lookaheads:

1. Lookaheads espontáneos

Sea I un conjunto de items con kernel $[A \rightarrow \alpha \bullet \beta, a]$ y $J = \text{GoTo}(I, X)$ para cualquier símbolo de la gramática X , b será un lookahead espontáneo donde $\text{GoTo}(\text{CLOSURE}(\{[A \rightarrow \alpha \bullet \beta, a]\}), X)$ contiene al item $[B \rightarrow \gamma \bullet \delta, b]$

El lookahead espontáneo para el item $S' \rightarrow \bullet S$ es $\#$.

2. Lookaheads propagados

Considerando las mismas condiciones del inciso anterior se toma $a = b$ y $\text{GoTo}(\text{CLOSURE}(\{[A \rightarrow \alpha \bullet \beta, b]\}), X)$ contiene al item $[B \rightarrow \gamma \bullet \delta, b]$, es decir se propaga el lookahead de $A \rightarrow \alpha \bullet \beta$ en el kernel del estado I , hacia $B \rightarrow \gamma \bullet \delta$ que está en el kernel del estado J .

Análisis sintáctico LALR

algoritmo para determinar lookaheads

Input: El kernel K de un conjunto de items I que se obtuvo por el método LR(0) y X un símbolo cualquiera de la gramática.

Output: Los lookaheads generados espontáneamente o los propagados por los items en I para los items del kernel en $J = \text{GoTo}(I, X)$.

Este algoritmo incluye un nuevo símbolo \diamond , que servirá para indicar los lookaheads que se propagarán.

```
for each item  $A \rightarrow \alpha \bullet \beta$  in  $K$ 
{
     $J := \text{CLOSURE}(\{[A \rightarrow \alpha \bullet \beta, \diamond]\})$ ;
    if  $[B \rightarrow \gamma \bullet X \delta, a]$  is in  $J$  and  $a$  is not  $\diamond$ 
    then lookahead  $a$  is generated spontaneously for item
         $B \rightarrow \gamma X \bullet \delta$  in  $\text{GoTo}(I, X)$ ;
    if  $[B \rightarrow \gamma \bullet X \delta, a]$  is in  $J$ 
    then lookaheads propagate from item  $A \rightarrow \alpha \bullet \beta$  in  $I$ 
        to item  $B \rightarrow \gamma X \bullet \delta$  in  $\text{GoTo}(I, X)$ ;
}
```

Análisis sintáctico LALR

ejemplo para tabla eficiente

Considera la gramática extendida

$$S' \rightarrow S$$

$$L \rightarrow \star R \mid \text{id}$$

$$S \rightarrow L = R \mid R$$

$$R \rightarrow L$$

Análisis sintáctico LALR

ejemplo para tabla eficiente

Considera la gramática extendida

$$S' \rightarrow S$$

$$L \rightarrow \star R \mid \text{id}$$

$$S \rightarrow L = R \mid R$$

$$R \rightarrow L$$

1. Calcular los estados o conjuntos de items usando el método LR(0)

$$\begin{aligned} I_0: \quad & S' \rightarrow \cdot S \\ & S \rightarrow \cdot L = R \\ & S \rightarrow \cdot R \\ & L \rightarrow \cdot \star R \\ & L \rightarrow \cdot \text{id} \\ & R \rightarrow \cdot L \end{aligned}$$

$$I_1: \quad S' \rightarrow S \cdot$$

$$\begin{aligned} I_2: \quad & S \rightarrow L \cdot = R \\ & R \rightarrow L \cdot \end{aligned}$$

$$I_3: \quad S \rightarrow R \cdot$$

$$\begin{aligned} I_4: \quad & L \rightarrow \star \cdot R \\ & R \rightarrow \cdot L \\ & L \rightarrow \cdot \star R \\ & L \rightarrow \cdot \text{id} \end{aligned}$$

$$I_5: \quad L \rightarrow \text{id} \cdot$$

$$\begin{aligned} I_6: \quad & S \rightarrow L = \cdot R \\ & R \rightarrow \cdot L \\ & L \rightarrow \cdot \star R \\ & L \rightarrow \cdot \text{id} \end{aligned}$$

$$I_7: \quad L \rightarrow \star R \cdot$$

$$I_8: \quad R \rightarrow L \cdot$$

$$I_9: \quad S \rightarrow L = R \cdot$$

Análisis sintáctico LALR

ejemplo para tabla eficiente

$$S' \rightarrow S$$

$$L \rightarrow \star R \mid \text{id}$$

$$S \rightarrow L = R \mid R$$

$$R \rightarrow L$$

2. Identificar los kernels de cada conjunto (item inicial o ítems que no tienen la bandera al inicio de la producción)

$$I_0: S' \rightarrow \cdot S$$

$$I_5: L \rightarrow \text{id} \cdot$$

$$I_1: S' \rightarrow S \cdot$$

$$I_6: S \rightarrow L = \cdot R$$

$$I_2: S \rightarrow L \cdot = R \\ R \rightarrow L \cdot$$

$$I_7: L \rightarrow \star R \cdot$$

$$I_3: S \rightarrow R \cdot$$

$$I_8: R \rightarrow L \cdot$$

$$I_4: L \rightarrow \star \cdot R$$

$$I_9: S \rightarrow L = R \cdot$$

Análisis sintáctico LALR

ejemplo para tabla eficiente

$$S' \rightarrow S$$

$$L \rightarrow \star R \mid \text{id}$$

$$S \rightarrow L = R \mid R$$

$$R \rightarrow L$$

3. Calcular la cerradura siguiendo el algoritmo para LR(1):

```
repeat{
  for each item  $[A \rightarrow \alpha \bullet B \beta, b]$  in  $I$ 
    for each grammar production  $B \rightarrow \gamma$ 
      for each terminal symbol  $x$  in  $\text{FIRST}(\beta b)$ 
        add  $[B \rightarrow \bullet \gamma, x]$  to  $I$ 
}
until no more items are added to  $I$ 
```

Análisis sintáctico LALR

ejemplo para tabla eficiente

$S' \rightarrow S$ $L \rightarrow *R \mid id$ $S \rightarrow L = R \mid R$ $R \rightarrow L$

3. Calcular la cerradura siguiendo el algoritmo para LR(1):

```
repeat{
  for each item  $[A \rightarrow \alpha \bullet B \beta, b]$  in I
    for each grammar production  $B \rightarrow \gamma$ 
      for each terminal symbol  $x$  in  $FIRST(\beta b)$ 
        add  $[B \rightarrow \bullet \gamma, x]$  to I
}
until no more items are added to I
```

La cerradura del kernel para I_0 , tomando $[S' \rightarrow \bullet S, \diamond]$ es:

$[S' \rightarrow \bullet S, \diamond]$ $[L \rightarrow \bullet * R, \diamond]$ $[S \rightarrow \bullet L = R, \diamond]$

$[L \rightarrow \bullet * R, =]$ $[S \rightarrow \bullet R, \diamond]$ $[L \rightarrow \bullet id, \diamond]$

$[R \rightarrow \bullet L, \diamond]$ $[L \rightarrow \bullet id, =]$

Análisis sintáctico LALR

ejemplo para tabla eficiente

$$S' \rightarrow S$$

$$L \rightarrow \star R \mid \text{id}$$

$$S \rightarrow L = R \mid R$$

$$R \rightarrow L$$

4. Identificar los lookaheads a los kernels encontrados siguiendo el algoritmo:

K kernel of set I and X any grammar symbol

```
for each item  $A \rightarrow \alpha \bullet \beta$  in K
{
  J := CLOSURE({[ $A \rightarrow \alpha \bullet \beta$ ,  $\Diamond$ ]});
  if [ $B \rightarrow \gamma \bullet X \delta$ , a] is in J and a is not  $\Diamond$ 
  then lookahead a is generated spontaneously for item
     $B \rightarrow \gamma X \bullet \delta$  in GoTo(I, X);
  if [ $B \rightarrow \gamma \bullet X \delta$ , a] is in J
  then lookaheads propagated from item  $A \rightarrow \alpha \bullet \beta$  in I
    to item  $B \rightarrow \gamma X \bullet \delta$  in GoTo(I, X);
}
```

Análisis sintáctico LALR

ejemplo para tabla eficiente

$$S' \rightarrow S$$

$$L \rightarrow \star R \mid id$$

$$S \rightarrow L = R \mid R$$

$$R \rightarrow L$$

4. Identificar los lookaheads a los kernels encontrados siguiendo el algoritmo:

K kernel of set I and X any grammar symbol

```
for each item  $A \rightarrow \alpha \bullet \beta$  in K
{
  J := CLOSURE({[A  $\rightarrow \alpha \bullet \beta$ ,  $\diamond$ ]});
  if [B  $\rightarrow \gamma \bullet X \delta$ , a] is in J and a is not  $\diamond$ 
  then lookahead a is generated spontaneously for item
    B  $\rightarrow \gamma X \bullet \delta$  in GoTo(I, X);
  if [B  $\rightarrow \gamma \bullet X \delta$ , a] is in J
  then lookaheads propagated from item A  $\rightarrow \alpha \bullet \beta$  in I
    to item B  $\rightarrow \gamma X \bullet \delta$  in GoTo(I, X);
}
```

La cerradura del kernel para l_0 , tomando $[S' \rightarrow \bullet S, \diamond]$ es:

$$\begin{aligned} [S' \rightarrow \bullet S, \diamond] \quad [L \rightarrow \bullet \star R, \diamond] \quad [S \rightarrow \bullet L = R, \diamond] \\ [L \rightarrow \bullet \star R, =] \quad [S \rightarrow \bullet R, \diamond] \quad [L \rightarrow \bullet id, \diamond] \\ [R \rightarrow \bullet L, \diamond] \quad [L \rightarrow \bullet id, =] \end{aligned}$$

1. los lookaheads = se generaron espontáneamente
2. calcular las propagaciones

Análisis sintáctico LALR

ejemplo para tabla eficiente

$$S' \rightarrow S$$

$$L \rightarrow \star R \mid \text{id}$$

$$S \rightarrow L = R \mid R$$

$$R \rightarrow L$$

4. Propagar los lookaheads iterando el algoritmo anterior

$I_0:$ $S' \rightarrow \cdot S$
 $S \rightarrow \cdot L = R$
 $S \rightarrow \cdot R$
 $L \rightarrow \cdot \star R$
 $L \rightarrow \cdot \text{id}$
 $R \rightarrow \cdot L$

$I_1:$ $S' \rightarrow S \cdot$

$I_2:$ $S \rightarrow L \cdot = R$
 $R \rightarrow L \cdot$

$I_3:$ $S \rightarrow R \cdot$

$I_4:$ $L \rightarrow \star \cdot R$
 $R \rightarrow \cdot L$
 $L \rightarrow \cdot \star R$
 $L \rightarrow \cdot \text{id}$

$I_5:$ $L \rightarrow \text{id} \cdot$

$I_6:$ $S \rightarrow L = \cdot R$
 $R \rightarrow \cdot L$
 $L \rightarrow \cdot \star R$
 $L \rightarrow \cdot \text{id}$

$I_7:$ $L \rightarrow \star R \cdot$

$I_8:$ $R \rightarrow L \cdot$

$I_9:$ $S \rightarrow L = R \cdot$

FROM	TO
$I_0:$ $S' \rightarrow \cdot S$	$I_1:$ $S' \rightarrow S \cdot$ $I_2:$ $S \rightarrow L \cdot = R$ $I_2:$ $R \rightarrow L \cdot$ $I_3:$ $S \rightarrow R \cdot$ $I_4:$ $L \rightarrow \star \cdot R$ $I_5:$ $L \rightarrow \text{id} \cdot$
$I_2:$ $S \rightarrow L \cdot = R$	$I_6:$ $S \rightarrow L = \cdot R$
$I_4:$ $L \rightarrow \star \cdot R$	$I_4:$ $L \rightarrow \star \cdot R$ $I_5:$ $L \rightarrow \text{id} \cdot$ $I_7:$ $L \rightarrow \star R \cdot$ $I_8:$ $R \rightarrow L \cdot$
$I_6:$ $S \rightarrow L = \cdot R$	$I_4:$ $L \rightarrow \star \cdot R$ $I_5:$ $L \rightarrow \text{id} \cdot$ $I_8:$ $R \rightarrow L \cdot$ $I_9:$ $S \rightarrow L = R \cdot$

Análisis sintáctico LALR

ejemplo para tabla eficiente

$$S' \rightarrow S$$

$$L \rightarrow \star R \mid \text{id}$$

$$S \rightarrow L = R \mid R$$

$$R \rightarrow L$$

4. Propagar los lookaheads iterando el algoritmo anterior, es decir revisando cada ítem en los kernels

SET	ITEM	LOOKAHEADS			
		INIT	PASS 1	PASS 2	PASS 3
I_0 :	$S' \rightarrow \cdot S$	\$	\$	\$	\$
I_1 :	$S' \rightarrow S \cdot$		\$	\$	\$
I_2 :	$S \rightarrow L \cdot = R$		\$	\$	\$
	$R \rightarrow L \cdot$		\$	\$	\$
I_3 :	$S \rightarrow R \cdot$		\$	\$	\$
I_4 :	$L \rightarrow \star \cdot R$	=	=/\$	=/\$	=/\$
I_5 :	$L \rightarrow \text{id} \cdot$	=	=/\$	=/\$	=/\$
I_6 :	$S \rightarrow L = \cdot R$			\$	\$
I_7 :	$L \rightarrow \star R \cdot$		=	=/\$	=/\$
I_8 :	$R \rightarrow L \cdot$		=	=/\$	=/\$
I_9 :	$S \rightarrow L = R \cdot$				\$

Análisis sintáctico LALR

ejemplo para tabla eficiente

Referencias

- [1] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman.
Compilers, Principles, Techniques and Tools.
Pearson Education Inc., Second edition, 2007.
- [2] Jean-Christophe Filliâtre.
Curso Compilation (inf564) école Polytechnique, Palaiseau, Francia.
<http://www.enseignement.polytechnique.fr/informatique/INF564/>, 2018.
Material en francés.
- [3] François Pottier.
Presentaciones del curso Compilation (inf564) École Polytechnique, Palaiseau, Francia.
<http://gallium.inria.fr/~fpottier/X/INF564/>, 2016.
Material en francés.
- [4] Michael Lee Scott.
Programming Language Pragmatics.
Morgan-Kaufman Publishers, Third edition, 2009.
- [5] Yunlin Su and Song Y. Yan.
Principles of Compilers, A New Approach to Compilers Including the Algebraic Method.
Springer-Verlag, Berlin Heidelberg, 2011.
- [6] Tim Teitelbaum.
Introduction to compilers.
<http://www.cs.cornell.edu/courses/cs412/2008sp/>, 2008.
- [7] Steve Zdancewic.
Notas del curso (CIS 341) - Compilers, Universidad de Pennsylvania, Estados Unidos.
<https://www.cis.upenn.edu/~cis341/current/>, 2018.