Compiladores:
Análisis Sintáctico

Pontificia Universidad Javeriana Cali
Ingenieria de Sistemas y Computación
Prof. Gloria Inés Alvarez V.

Parsing LR: Ejemplo:

- $E \rightarrow E + T$
- $E \rightarrow T$
- $T \rightarrow T * id$
- $T \rightarrow id$

S_n : shift al estado n

R_m: reduce con la regla (m)

Acc: accept

Espacio en blanco: error.

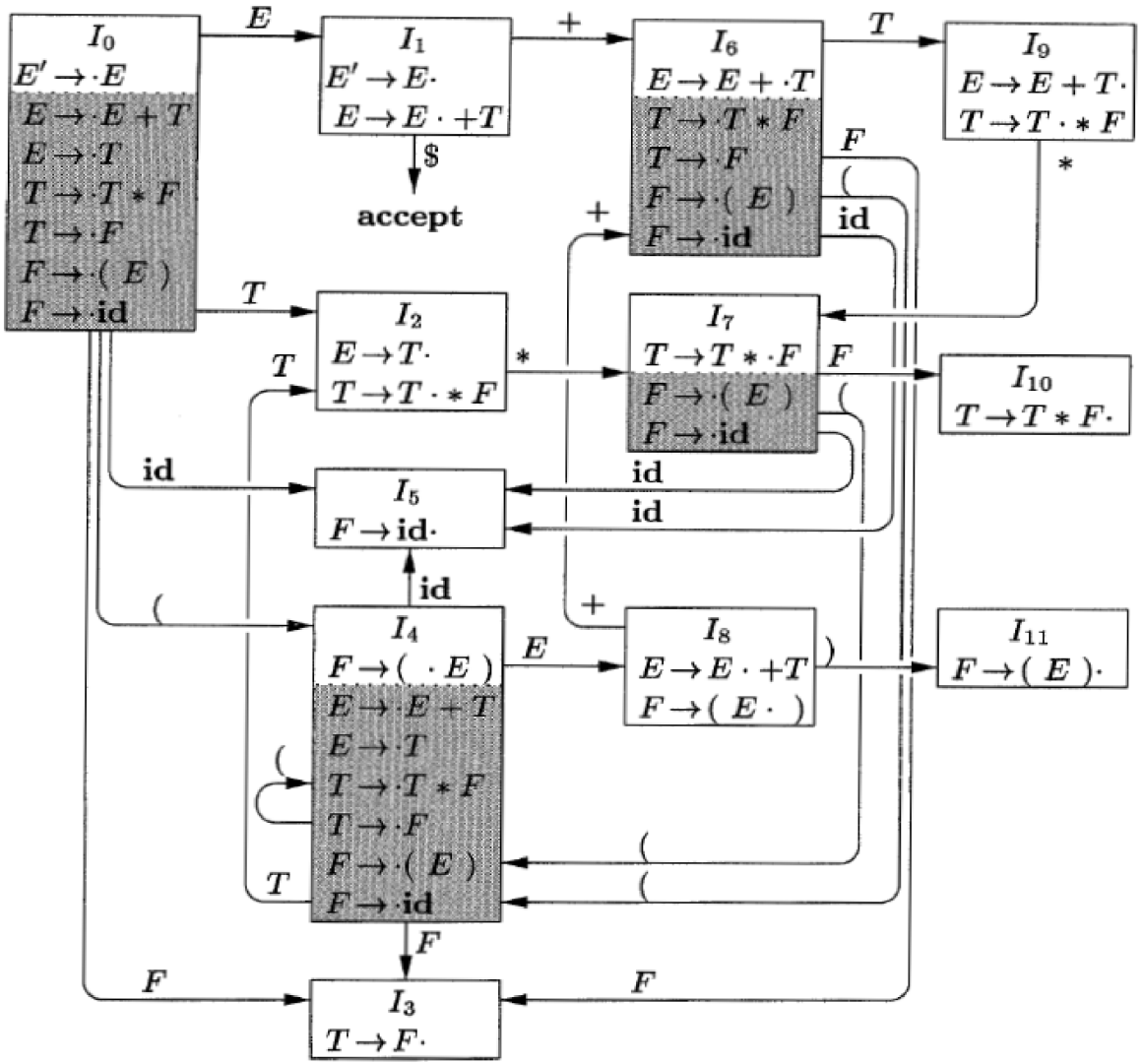
	Action				Goto	
	Id	**+**	*****	**$**	**E**	**T**
I_0	S_3				1	2
I_1		S_4		Acc		
I_2		r_2	S_5	r_2		
I_3		r_4	r_4	r_4		
I_4	S_3					6
I_5	S_7					
I_6		r_1	S_5	r_1		
I_7		r_3	r_3	r_3		

Algoritmo LR [Aho p. 251]

METHOD: Initially, the parser has s_0 on its stack, where s_0 is the initial state, and $w\$$ in the input buffer. The parser then executes the program in Fig. 4.36. □

```
let a be the first symbol of w$;
while(1) { /* repeat forever */
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break; /* parsing is done */
        else call error-recovery routine;
}
```

Autómata LR(0) [Aho p.244]



Construcción tabla SLR(0)

- Para I_0 Action[0,(] = desplazar 4

  Action[0,id] = desplazar 5
- Para I_1 Action[1,$] = aceptar

  Action[1,+] = desplazar 6
- Para I_2 se sabe que follow(E) = {$,+,)}

  Action[2,$] = reducir $E \rightarrow T$

  Action[2,+] = reducir $E \rightarrow T$

  Action[2,)] = reducir $E \rightarrow T$

  Action[2,*] = desplazar 7

Construcción de Tabla LR Canonica

- Es la técnica más general para construir tablas de parsing LR.
- Permite llevar mas información en los estados, lo cual evitará realizar reducciones invalidas.
- Este item es llamado Item LR(1), “1” se refiere a la longitud del segundo componente (el “lookahead” del item).

Items Canónicos LR(1)

- Se extienden los items para incluir información acerca del símbolo siguiente en la entrada.
- La información adicional se incorpora incluyendo en el item símbolos terminales como segundos componentes:

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

donde $A \rightarrow \alpha\beta$ es una regla de producción y

a es un símbolo terminal o $ (marcador de finalización)

Construcción de los conjuntos de items LR(1)

Clausura (I):

repita

para cada item [$A \rightarrow \alpha \bullet B\beta$, a] en I

para cada producción $B \rightarrow \gamma$ en G'

para cada terminal b en FIRST(βa)

agregue [$B \rightarrow \bullet\gamma$, b] a I

hasta que no sea posible agregar mas items a I

retorne I

Construcción de los conjuntos de items LR(1)

Goto (I, X):

J = conjunto de items $[A \rightarrow \alpha X \bullet \beta, a]$, tales que
$[A \rightarrow \alpha \bullet X \beta, a]$ está en I

retorne (Clausura (J));

Construcción de los conjuntos de items LR(1)

Items (G'):

C = Clausura ({ [S'→•S, $] })

repita

para cada conjunto de items I en C

para cada símbolo gramatical X tal que

goto(I, X) no es vacío, y no está en C

agregue goto(I, X) a C

hasta que no se puedan agregar más conjuntos de items a C

Tabla LR(1)[Aho p. 265]

INPUT: An augmented grammar G'.

OUTPUT: The canonical-LR parsing table functions ACTION and GOTO for G'.

METHOD:

1. Construct $C' = \{I_0, I_1, \cdots, I_n\}$, the collection of sets of LR(1) items for G'.

2. State i of the parser is constructed from I_i. The parsing action for state i is determined as follows.

   (a) If $[A \to \alpha \cdot a\beta, b]$ is in I_i and $\text{GOTO}(I_i, a) = I_j$, then set ACTION$[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \to \alpha \cdot, a]$ is in I_i, $A \neq S'$, then set ACTION$[i, a]$ to "reduce $A \to \alpha$."

   (c) If $[S' \to S \cdot, \$]$ is in I_i, then set ACTION$[i, \$]$ to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not LR(1). The algorithm fails to produce a parser in this case.

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then GOTO$[i, A] = j$.

4. All entries not defined by rules (2) and (3) are made "error."

5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S, \$]$.

Construcción de Tabla LALR

- Produce tablas más pequeñas que el LR Canonico.
- Las construcciones sintácticas mas comunes de lenguajes de programación pueden ser expresadas con gramáticas LALR.

Construcción de tabla LALR(1) [Aho p.268]

INPUT: An augmented grammar G'.

OUTPUT: The LALR parsing-table functions ACTION and GOTO for G'.

METHOD:

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(1) items.

2. For each core present among the set of LR(1) items, find all sets having that core, and replace these sets by their union.

3. Let $C' = \{J_0, J_1, \ldots, J_m\}$ be the resulting sets of LR(1) items. The parsing actions for state i are constructed from J_i in the same manner as in Algorithm 4.56. If there is a parsing action conflict, the algorithm fails to produce a parser, and the grammar is said not to be LALR(1).

4. The GOTO table is constructed as follows. If J is the union of one or more sets of LR(1) items, that is, $J = I_1 \cap I_2 \cap \cdots \cap I_k$, then the cores of $\text{GOTO}(I_1, X)$, $\text{GOTO}(I_2, X), \ldots, \text{GOTO}(I_k, X)$ are the same, since $I_1, I_2, \ldots, I_k$ all have the same core. Let K be the union of all sets of items having the same core as $\text{GOTO}(I_1, X)$. Then $\text{GOTO}(J, X) = K$.

Tabla LALR(1)[Aho p. 265]

INPUT: An augmented grammar G'.

OUTPUT: The canonical-LR parsing table functions ACTION and GOTO for G'.

METHOD:

1. Construct $C' = \{I_0, I_1, \cdots, I_n\}$, the collection of sets of LR(1) items for G'.

2. State i of the parser is constructed from I_i. The parsing action for state i is determined as follows.

   (a) If $[A \to \alpha \cdot a\beta, b]$ is in I_i and $\text{GOTO}(I_i, a) = I_j$, then set ACTION$[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \to \alpha \cdot, a]$ is in I_i, $A \neq S'$, then set ACTION$[i, a]$ to "reduce $A \to \alpha$."

   (c) If $[S' \to S \cdot, \$]$ is in I_i, then set ACTION$[i, \$]$ to "accept."

   If any conflicting actions result from the above rules, we say the grammar is not LR(1). The algorithm fails to produce a parser in this case.

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then GOTO$[i, A] = j$.

4. All entries not defined by rules (2) and (3) are made "error."

5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S, \$]$.

Construcción de Tabla LALR. Ejemplo

- Gramatica aumentada:

  $S' \rightarrow S$

  $S \rightarrow C\ C$

  $C \rightarrow e\ C$

  $C \rightarrow d$

Construcción de Tabla LALR. Ejemplo: Estados

$I_0 = \{ \; S' \rightarrow \bullet S, \$$
$S \rightarrow \bullet CC, \$$
$C \rightarrow \bullet eC, e/d$
$C \rightarrow \bullet d, e/d \; \}$

$I_1 = \{ \; S' \rightarrow S\bullet, \$ \; \}$

$I_2 = \{ \; S \rightarrow C\bullet C, \$$
$C \rightarrow \bullet eC, \$$
$C \rightarrow \bullet d, \$ \; \}$

$I_3 = \{ \; C \rightarrow e\bullet C, e/d$
$C \rightarrow \bullet eC, e/d$
$C \rightarrow \bullet d, e/d \; \}$

$I_4 = \{ \; C \rightarrow d\bullet, e/d \; \}$

$I_5 = \{ \; S \rightarrow CC\bullet, \$ \; \}$

$I_6 = \{ \; C \rightarrow e\bullet C, \$$
$C \rightarrow \bullet eC, \$$
$C \rightarrow \bullet d, \$ \; \}$

$I_7 = \{ \; C \rightarrow d\bullet, \$ \; \}$

$I_8 = \{ \; C \rightarrow eC\bullet, e/d \; \}$

$I_9 = \{ \; C \rightarrow eC\bullet, \$ \; \}$

Construcción tabla LALR Ejemplo [Aho p.262]

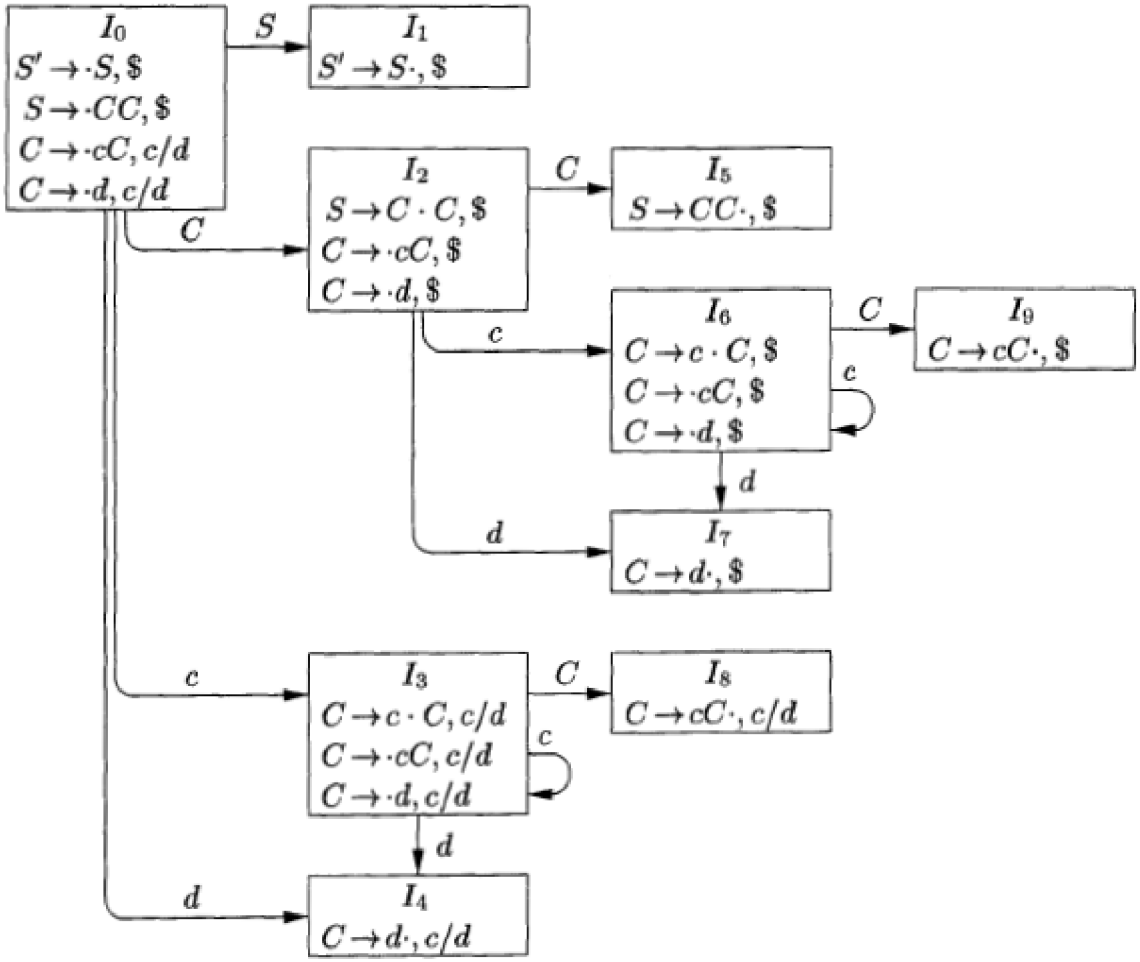


Figure 4.41: The GOTO graph for grammar (4.55)

Construcción tabla LALR Ejemplo [Aho p.269]

STATE	ACTION			GOTO	
	c	*d*	$	*S*	*C*
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		

FIN

Recuperación de Error en parsing LR

- ***Los errores se detectan en la tabla de acciones (no en la tabla de goto)***
- ***El parsing LR canónico no realiza reducciones antes de anunciar el error.***
- ***Los parsing SLR y LALR realizan reducciones, pero no hacen shift de un símbolo de entrada erroneo en la pila.***

Recuperación de Error en parsing LR

- ***Recuperación de error en modo pánico:***
  - ***Desciende por la pila buscando un estado s que tenga un goto a un no terminal A (que represente una pieza importante del programa ej. Instrucción, bloque, expresión...)***
  - ***Se descartan símbolos de la entrada, hasta encontrar un símbolo a que pertenece a FOLLOW (A)***
  - ***El parser realiza goto[s,A]***

Recuperación de Error en parsing LR

- ***Recuperación de error a nivel de frase:***
  - ***Crea rutinas de recuperación de error para cada entrada erronea de la tabla de parsing, decidiendo con base en los errores usuales que los programadores cometen.***
  - ***La rutina modifica los símbolos en el tope de la pila.***
  - ***Los cambios deben garantizar que el parser no entre en un ciclo infinito.***

Análizador Sintáctico Ascendente: Conflictos

- Una gramática ambigua no puede ser LR(k)
- Pero el analizador puede ser adaptado para reconocer ciertas gramáticas ambiguas, usando reglas que especifican la solución al conflicto:
  - Usando precedencia y asociatividad
  - Resolver el conflicto desplazamiento-reducción optando por el desplazamiento