

Capítulo III

Histórico de revisões:

2010-2014 – Carlos M. Fonseca

2008-2010 – Luís Macedo

Anos anteriores – F. Amílcar Cardoso

Analizador SLR (simple LR)

2

- A construção da tabela de parsing é, no essencial, idêntica à LR(0)
- Diferença:
 - só se faz reduce usando $A \rightarrow \gamma$ se o próximo símbolo a ler pertencer a FOLLOW(A)

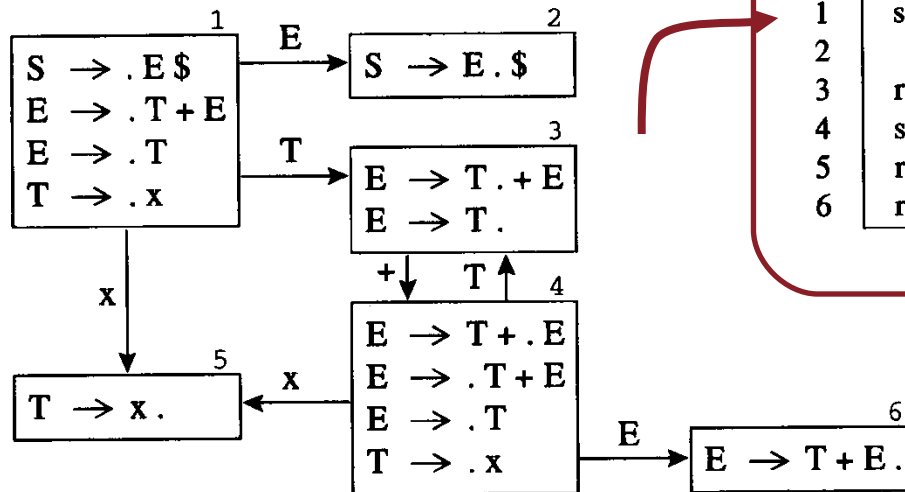
```
 $R \leftarrow \{\}$   
for each state  $I$  in  $T$   
  for each item  $A \rightarrow \alpha.$  in  $I$   
    for each token  $X$  in FOLLOW(A)  
       $R \leftarrow R \cup \{(I, X, A \rightarrow \alpha)\}$ 
```

Num parser LR(0):

```
 $R \leftarrow \{\}$   
for each state  $I$  in  $T$   
  for each item  $A \rightarrow \alpha.$  in  $I$   
     $R \leftarrow R \cup \{(I, A \rightarrow \alpha)\}$ 
```

Analizador SLR

3

 $0 \quad S \rightarrow E \$$
 $1 \quad E \rightarrow T + E$
 $2 \quad E \rightarrow T$
 $3 \quad T \rightarrow x$


	x	+	\$	E	T
1	s5			g2	g3
2			a		
3	r2	s4,r2	r2		
4	s5			g6	g3
5	r3	r3	r3		
6	r1	r1	r1		

LR

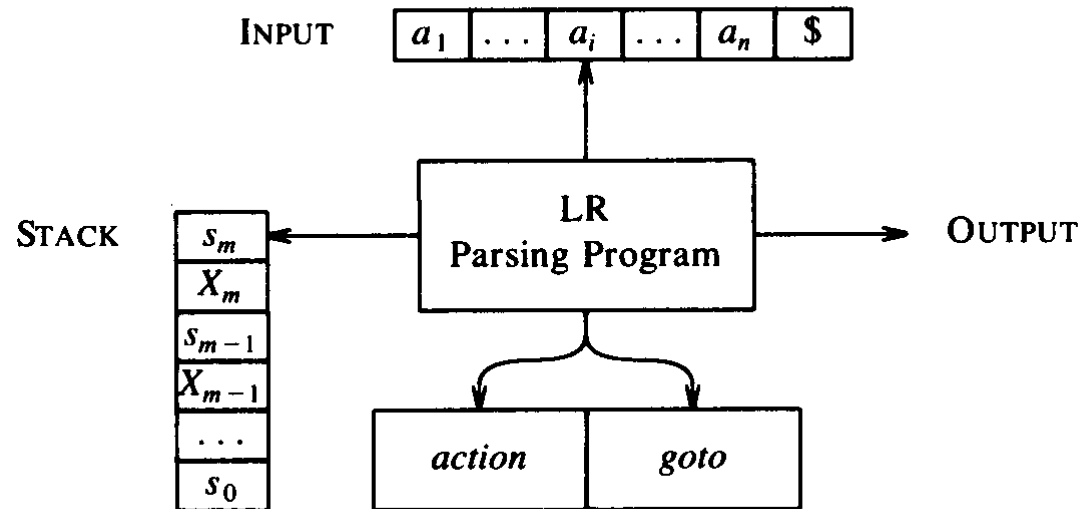
FOLLOW(E) = {\$}

	x	+	\$	E	T
1	s5			g2	g3
2			a		
3		s4	r2		
4	s5			g6	g3
5		r3	r3		
6			r1		

SLR

Analizador LR(1)

4



Analizador LR(1)

5

□ Recordando...

Stack	Input	Action
1	a := 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄	: = 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ := 6	7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ := 6 num ₁₀	; b := 'c + (d := 5 + 6 , d) \$	reduce E → num
1 id ₄ := 6 E ₁₁	; b := c + (d := 5 + 6 , d) \$	reduce S → id := E
1 S ₂	; b := c + (d := 5 + 6 , d) \$	shift
1 S ₂ ; 3	b := c + (d := 5 + 6 , d) \$	shift
1 S ₂ ; 3 id ₄	: = c + (d := 5 + 6 , d) \$	shift
1 S ₂ ; 3 id ₄ := 6	c + (d := 5 + 6 , d) \$	shift
1 S ₂ ; 3 id ₄ := 6 id ₂₀	+ (d := 5 + 6 , d) \$	reduce E → id
1 S ₂ ; 3 id ₄ := 6 E ₁₁	+ (d := 5 + 6 , d) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16	(d := 5 + 6 , d) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8	d := 5 + 6 , d) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 id ₄	: = 5 + 6 , d) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 id ₄ := 6	5 + 6 , d) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 id ₄ := 6 num ₁₀	+ 6 , d) \$	reduce E → num
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 id ₄ := 6 E ₁₁	+ 6 , d) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 id ₄ := 6 E ₁₁ + 16	6 , d) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 id ₄ := 6 E ₁₁ + 16 num ₁₀	, d) \$	reduce E → num
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 id ₄ := 6 E ₁₁ + 16 E ₁₇	, d) \$	reduce E → E + E
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 id ₄ := 6 E ₁₁	, d) \$	reduce S → id := E
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 S ₁₂	, d) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 S ₁₂ , 18	d) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 S ₁₂ , 18 id ₂₀) \$	reduce E → id
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 S ₁₂ , 18 E ₂₁) \$	shift
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 (8 S ₁₂ , 18 E ₂₁) 22	\$	reduce E → (S , E)
1 S ₂ ; 3 id ₄ := 6 E ₁₁ + 16 E ₁₇	\$	reduce E → E + E
1 S ₂ ; 3 id ₄ := 6 E ₁₁	\$	reduce S → id := E
1 S ₂ ; 3 S ₅	\$	reduce S → S ; S
1 S ₂	\$	accept

Analizador LR(1)

6

- Analizador sintático LR(1):
 - trabalha olhando para o topo da pilha, o caracter de entrada e para um caracter de *lookahead*
- Gramática:

<p>0 • $S' \rightarrow S \\$</p> <p>1 • $S \rightarrow V = E$</p> <p>2 • $S \rightarrow E$</p>	<p>3 $E \rightarrow V$</p> <p>4 $V \rightarrow x$</p> <p>5 $V \rightarrow * E$</p>
--	---
- Início
 - pilha vazia
 - frase completa à entrada, seguida de '\$'

Analizador LR(1)

7

- Ainda mais poderoso que o SLR
- Algoritmo idêntico ao LR(0), mas com itens mais complexos, com 3 componentes:
 - uma produção
 - uma posição de processamento (o ponto)
 - um símbolo de *lookahead*
- Exemplo:
 - $(A \rightarrow \alpha.\beta, x)$ indica que
 - a sequência α está no topo da pilha
 - na cabeça de leitura está uma cadeia derivável de βx
 - x é portanto o lookahead, que deverá ser encontrado depois de se processar β

Analizador LR(1)

8

□ Closure(I):

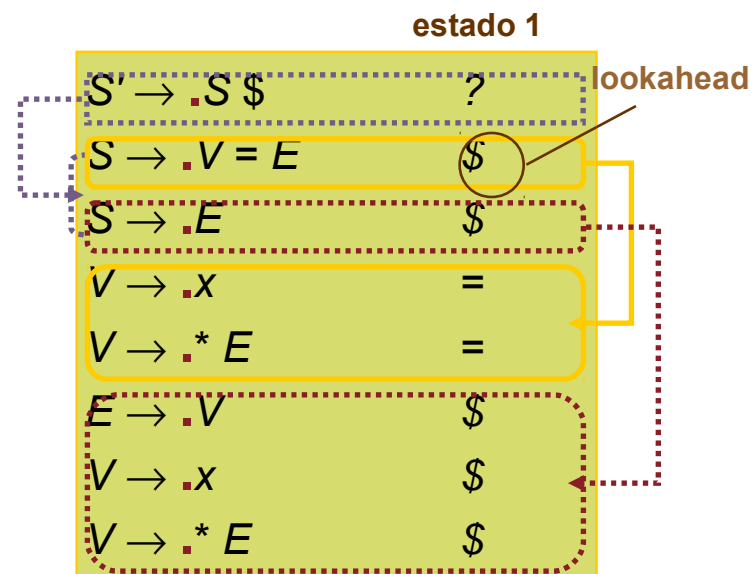
- acrescentar novos itens ao conjunto I quando o ponto está à esquerda de um não-terminal

$S' \rightarrow S \$$	$E \rightarrow V$
$S \rightarrow V = E$	$V \rightarrow x$
$S \rightarrow E$	$V \rightarrow * E$

```

Closure( $I$ ) =  $\{$ 
  repeat
    for any item ( $A \rightarrow \alpha.X\beta, z$ ) in  $I$ 
      for any production  $X \rightarrow \gamma$ 
        for any  $w \in \text{FIRST}(\beta z)$ 
           $I \leftarrow I \cup \{(X \rightarrow \cdot\gamma, w)\}$ 
  until  $I$  does not change
  return  $I$ 

```



Analizador LR(1)

9

□ goto(I, X):

- mover ponto para a direita de X em todos os itens

Goto(I, X) =

$J \leftarrow \{ \}$

for any item ($A \rightarrow \alpha.X\beta, z$) in I

 add ($A \rightarrow \alpha X.\beta, z$) to J

return Closure(J).

$S' \rightarrow S \$$	$E \rightarrow V$
$S \rightarrow V = E$	$V \rightarrow x$
$S \rightarrow E$	$V \rightarrow * E$

estado 1

$S' \rightarrow .S \$$?
$S \rightarrow .V = E$	\$
$S \rightarrow .E$	\$
$V \rightarrow .x$	=
$V \rightarrow .* E$	=
$E \rightarrow .V$	\$
$V \rightarrow .x$	\$
$V \rightarrow .* E$	\$

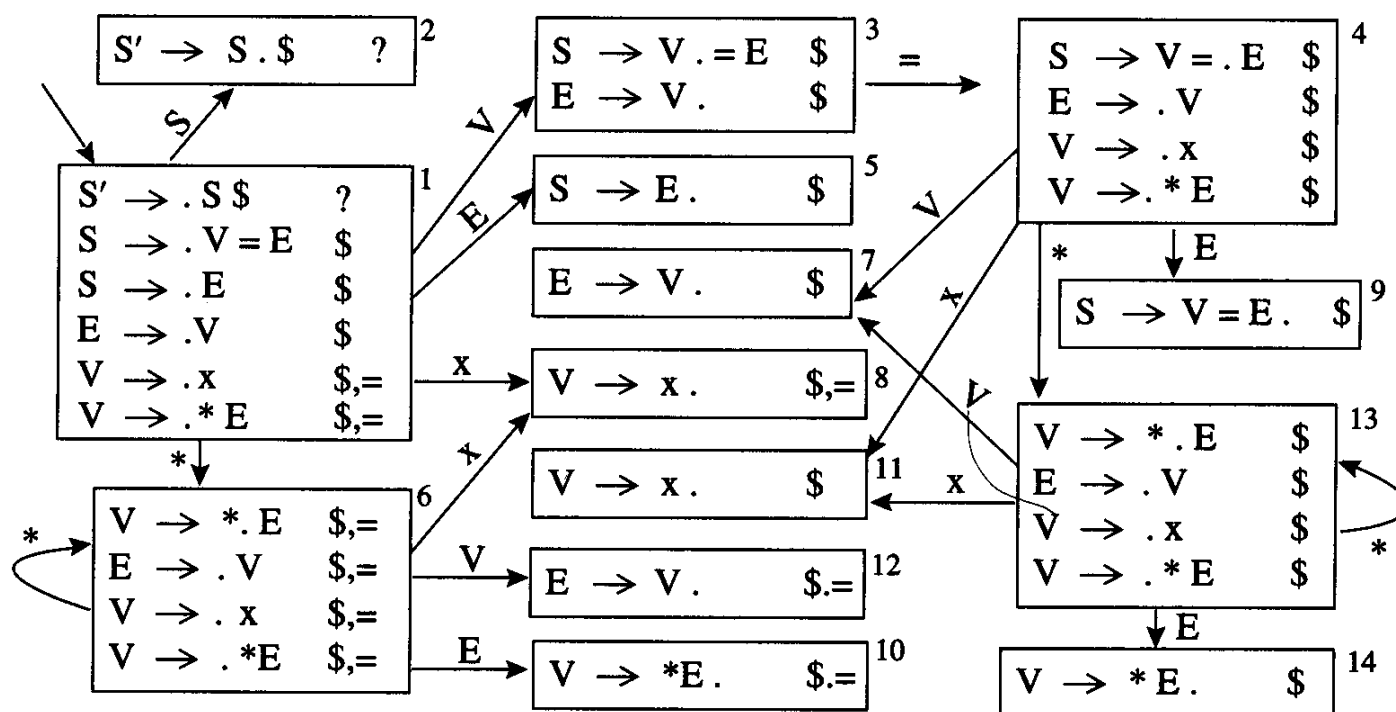
estado 3

$S \rightarrow V . = E$	\$
$E \rightarrow V .$	\$

V

Analizador LR(1) - Diagrama de Estados

10



Analizador LR(1)

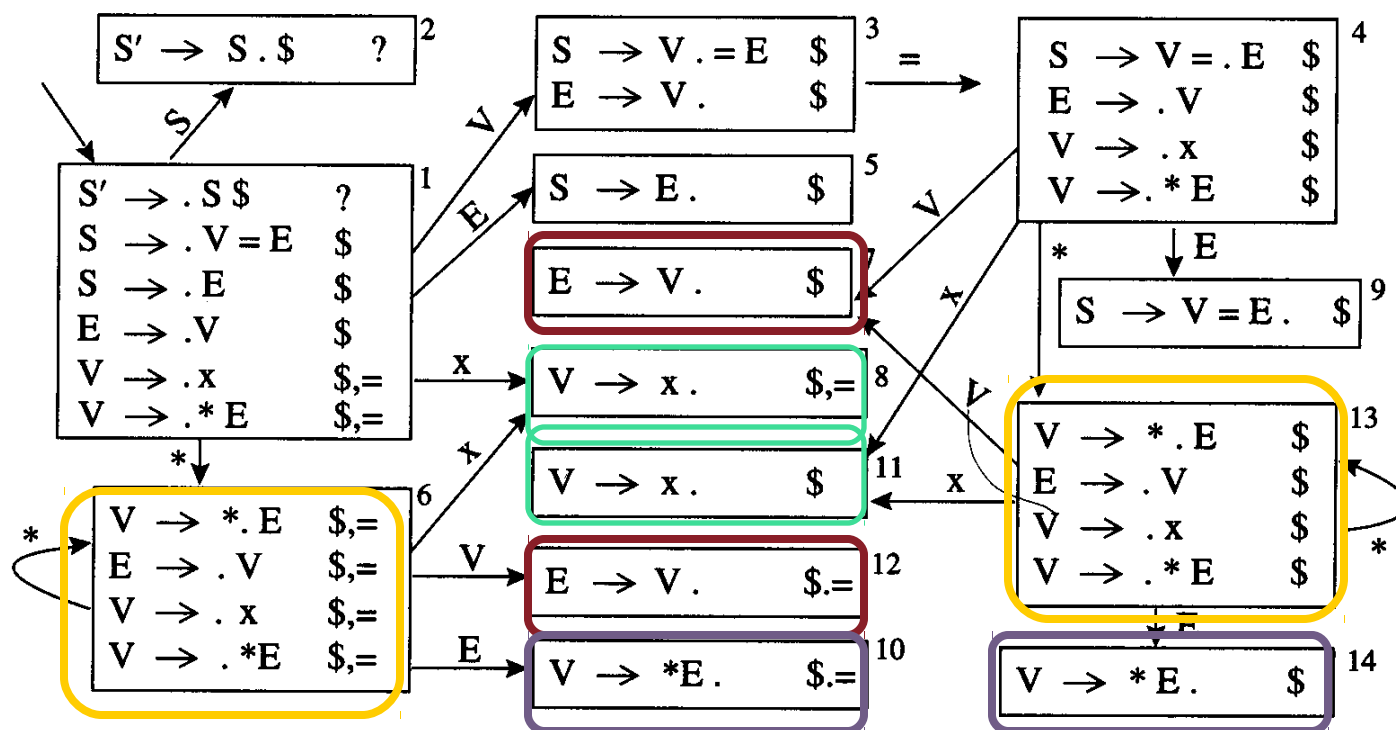
11

	x	*	=	\$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

(a) LR(1)

Tabela de Parsing LALR(1)

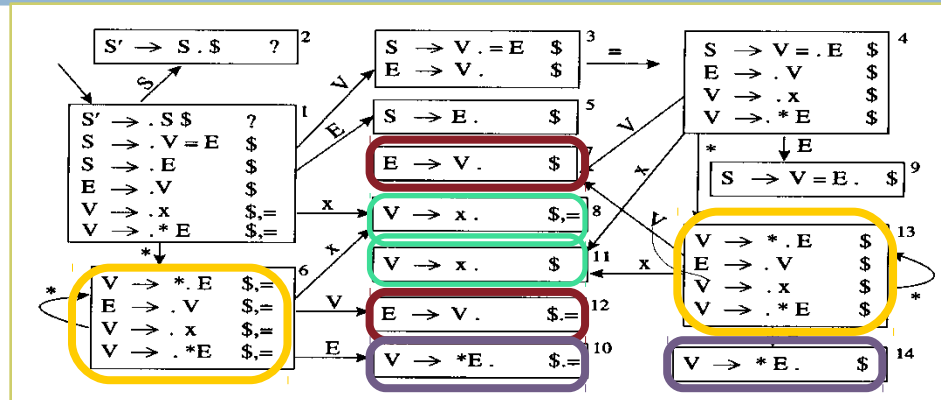
12



- Uma tabela LALR(1) obtém-se de uma LR(1) pela fusão de estados que sejam idênticos excepto no símbolo de *lookahead*

Tabela de Parsing LALR(1)

13



	x	*	=	\$	S	E	V
1	s8	s6			g2	g5	g3
2			a				
3			s4	r3			
4	s11	s13				g9	g7
5			r2				
6	s8	s6				g10	g12
7			r3				
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14			r5				

(a) LR(1)



	x	*	=	\$	S	E	V
1	s8	s6			g2	g5	g3
2			a				
3			s4	r3			
4	s8	s6				g9	g7
5			r2				
6	s8	s6				g10	g7
7			r3	r3			
8			r4	r4			
9				r1			
10			r5	r5			

ores - 2013/2014

Tabela de Parsing LALR(1)

14

- LALR(1): *lookahead LR(1)*
- Método muito utilizado:
 - tabelas de parsing muito menores que as LR(1):
 - tabela LALR(1) tem, para linguagens comuns, centenas de estados (tal como SLR)
 - tabela LR(1) tem milhares
 - uma tabela LALR(1) pode conter conflitos reduce/reduce não existentes na correspondente tabela LR(1)
 - gramáticas LALR(1) mais abrangentes que as SLR

Análise LR de Gramáticas Ambíguas

15

- Gramática:
 - $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
 - $S \rightarrow \text{if } E \text{ then } S$
 - $S \rightarrow \text{outros}$
- Exemplo:
 - $\text{if } a \text{ then if } b \text{ then } s1 \text{ else } s2$
- Duas interpretações:
 - $\text{if } a \text{ then } \{ \text{if } b \text{ then } s1 \text{ else } s2 \}$
 - $\text{if } a \text{ then } \{ \text{if } b \text{ then } s1 \} \text{ else } s2$
- *na generalidade das linguagens, a primeira é a adoptada: **else associado ao then mais próximo***

Análise LR de Gramáticas Ambíguas

16

- Duas interpretações:
 - if a then { if b then s1 else s2 }
 - if a then { if b then s1 } else s2
- Na tabela de parsing, conflito shift/reduce:

• $S \rightarrow \text{if } E \text{ then } S.$	else	(reduce)
• $S \rightarrow \text{if } E \text{ then } S. \text{ else } S$?	(shift)
- Soluções:
 - Eliminar a ambiguidade alterando a gramática
 - Tolerar o conflito, e resolvê-lo dando prioridade ao deslocamento
 - → Ambas funcionam com o YACC...

Precedência e Associatividade

17

- Tomemos como exemplo, as duas regras seguintes :

- | | |
|-----------------------------------|----------------------------|
| ■ $S \rightarrow S ; S$ | $E \rightarrow \text{id}$ |
| ■ $S \rightarrow \text{id} := E$ | $E \rightarrow \text{num}$ |
| ■ $S \rightarrow \text{print}(L)$ | $E \rightarrow E + E$ |
| ■ $L \rightarrow E$ | $E \rightarrow E * E$ |
| ■ $L \rightarrow L, E$ | |

- Se construirmos os estados LR(0), a certa altura, teremos um estado que conterá (resultante da leitura de E):

• $E \rightarrow E * E.$

...e se o próximo símbolo for um '+' ?

• $E \rightarrow E. + E$

fazemos shift ou reduce?

Precedência e Associatividade

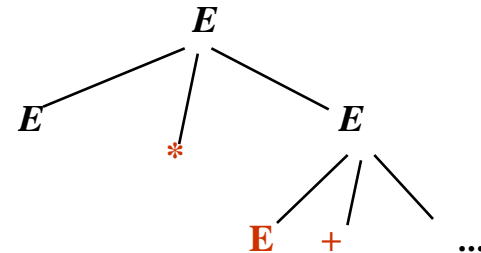
18

□ $E \rightarrow E * E.$

□ $E \rightarrow E. + E$

□ Se fizermos *shift* (deslocar), a pilha ficará com $E * E + \dots$

□ e a árvore de derivação será:



□ ...ou seja, somamos antes de multiplicar!...

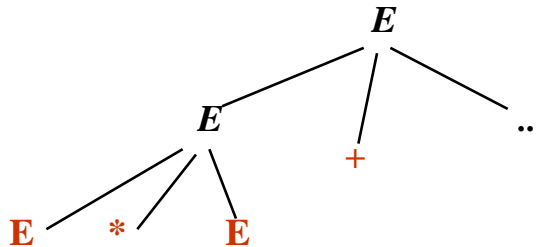
Precedência e Associatividade

19

□ $E \rightarrow E * E.$

□ $E \rightarrow E + E$

□ Se fizermos *reduce*, a árvore de derivação será:



□ ...escolher *reduce* daria o valor correcto!

Precedência e Associatividade

20

- E uma situação ainda mais simples:

$$E \rightarrow E + E.$$

$$E \rightarrow E. + E$$

- Se o próximo símbolo for um '+', o que fazer? Shift ou reduce?
- Se fizermos shift, a soma da direita é feita primeiro (associatividade à direita)
- Se fizermos reduce, será a da esquerda (associatividade à esquerda), o que está correcto!

Precedência e Associatividade

21

- Em alguns casos o conflito shift/reduce pode ser resolvido com atenção à precedência e associatividade:
 - Se o novo símbolo (a ler) tem menos prioridade que o já lido, faz-se reduce
 - $E * E . + E \rightarrow \text{reduce}$
 - Se o novo símbolo (a ler) tem maior prioridade que o já lido, faz-se shift
 - $E + E . * E \rightarrow \text{shift}$
 - Se o novo símbolo (a ler) tem a mesma prioridade que o já lido, temos duas hipóteses:
 - Associatividade à esquerda $E + E . + E \rightarrow \text{reduce}$
 - Associatividade à direita $E = E . = E \rightarrow \text{shift}$

Precedência e Associatividade

22

- O Yacc permite definir precedências e associatividade

- ...
 - `%left '+' '-'`
 - `%left '*' '/'`
 - `%right '='`
 - `%%`
 - ... Yacc rules...
- Associativo à esquerda
- Menos prioridade
- Mais prioridade
- Associativo à direita
-

Erros reduce/reduce

23

□ Tomemos como exemplo a seguinte gramática:

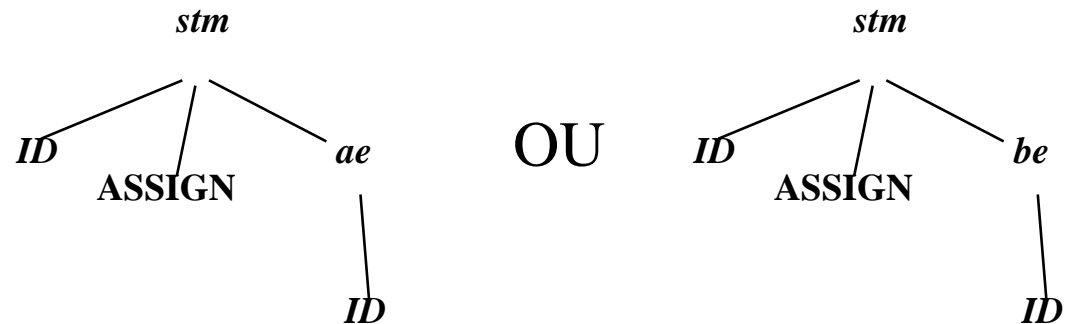
- $stm : ID \text{ ASSIGN } ae \mid ID \text{ ASSIGN } be$
- $be : be \text{ OR } be \mid be \text{ AND } be \mid ae \text{ EQUAL } ae \mid ID$
- $ae : ae \text{ PLUS } ae \mid ID$

Para a string

$x = y$

Qual é a árvore sintática?

ERRO reduce/reduce!!



Erros reduce/reduce

24

- Nestes casos, tem mesmo que se passar o problema para a fase de análise semântica.
- Ou seja, modificar a gramática:
 - `stm : ID ASSIGN be`
 - `be : be OR be | be AND be | ae EQUAL ae | ae`
 - `ae : ae PLUS ae | ID`

Recuperação de Erros

25

- É conveniente que o parser não interrompa a ação quando encontra um erro
- Ele deve levar a análise até ao fim e reportar todos os erros encontrados
- A recuperação de erros pode ser
 - local
 - global

Recuperação Local

26

- Ideia-base:
 - ajustar pilha e entrada no ponto em que o erro é descoberto
 - usar um símbolo especial de erro para ajudar na recuperação
- Símbolo **error**:
 - símbolo terminal
 - produções de recuperação consideradas na construção da tabela de parsing


Recuperação Local

27

□ Uma gramática:

- $Exp \rightarrow id$ $ExpS \rightarrow Exp$
- $Exp \rightarrow Exp + Exp$ $ExpS \rightarrow ExpS ; Exp$
- $Exp \rightarrow (ExpS)$

□ Produções de recuperação de erros:

- $Exp \rightarrow (\text{error})$
 - $ExpS \rightarrow \text{error} ; Exp$
- Tokens de sincronização
- 

□ Quando o parser LR atinge estado de erro:

- Retirar símbolos do topo da pilha até que se atinja um estado em que a acção seja *shift error*
- Fazer *shift error*
- Desprezar símbolos de entrada até se encontrar um que produza uma acção correta no estado corrente
- Retomar parsing normal

Recuperação Global

28

□ Ideia-base:

- encontrar o menor conjunto de inserções/apagamentos que tornam a entrada sintaticamente correcta
- as inserções/apagamentos poderão ocorrer em pontos afastados do local em que o parser deteta o erro

□ Método de Burke-Fisher:

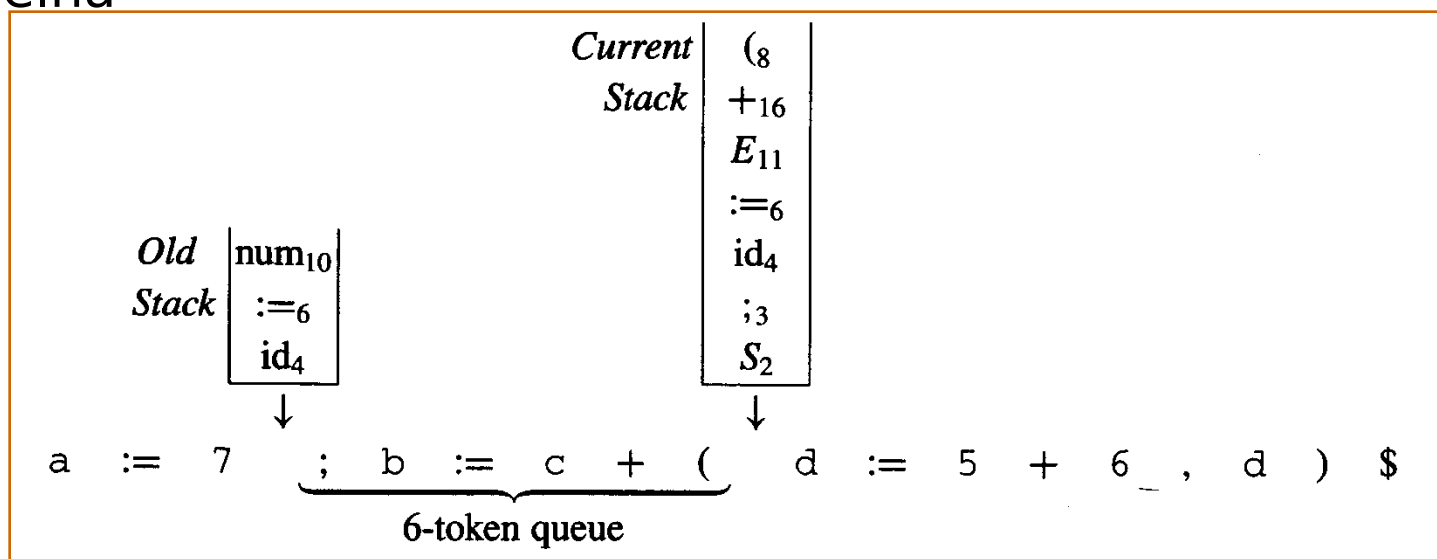
- operações de inserção, apagamento ou substituição
- operações unitárias (1 token)
- tentar todas as possibilidades até K tokens antes do ponto em que se detectou o erro
- reparação tem sucesso se permitir levar o parsing mais longe

Recuperação Global

29

□ Necessário:

- manter fila de espera com K tokens
- saber como estava a pilha K tokens atrás
- ou seja: necessário manter duas pilhas: corrente e velha



Recuperação Global

30

- Vantagem do método:
 - desnecessário alterar a gramática
 - desnecessário alterar a tabela de parsing
 - para N tokens diferentes e uma janela de comprimento K , são possíveis $K + (K+1)N + K(N-1)$ operações de apagamento, inserção e substituição; não é muito, atendendo a que só acontece quando se detecta um erro

