

Geração de Parser

LR(1)

Hervé Yviquel

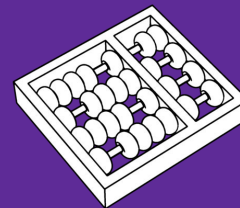
herve@ic.unicamp.br

Universidade Estadual de Campinas (Unicamp)

Instituto de Computação (IC)

Laboratório de Sistemas de Computação (LSC)

MC921 • Projeto e Construção de Compiladores • 2022 S2



UNICAMP

Aula Anterior

Resumo

- Visão Geral do Front-end
- Analisador Semântica
- Symbol table
- Type Checking
- Visitor
- Erros semânticos

Aula de Hoje

Plano

- Parser LR(1)
- LALR(1)
- Ambiguidade
- GLR
- Recuperação de Erros

LR(1)



- Mais poderosa do que SLR
- Maioria das linguagens de programação são LR(1)
 - Exceção notável : C++!!!
- Funcionamento similar a LR(0)
 - Autômato associado a uma pilha
- Item LR(1): $(A \rightarrow \alpha.\beta, x)$
 - α está no topo da pilha
 - x representa o símbolo de lookahead

- Closure(I)
 - Adiciona itens a um estado quando um "." precede um não terminal
- Goto(I,X)
 - Movimenta o "." para depois de X em todos os itens

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

```
Goto(I, X) =  
  set  $J$  to the empty set  
  for any item  $A \rightarrow (\alpha.X\beta, z)$  in  $I$   
    add  $A \rightarrow (\alpha X.\beta, z)$  to  $J$   
return Closure( $J$ )
```

- Construção do parser LR(1)

Initialize T to $\{\mathbf{Closure}(\{S' \rightarrow .S\})\}$

Initialize E to empty

repeat

for each state I in T

for each item $A \rightarrow \alpha.X\beta$ in I

let J be $\mathbf{Goto}(I, X)$

$T \leftarrow T \cup \{J\}$

$E \leftarrow E \cup \{I \xrightarrow{X} J\}$

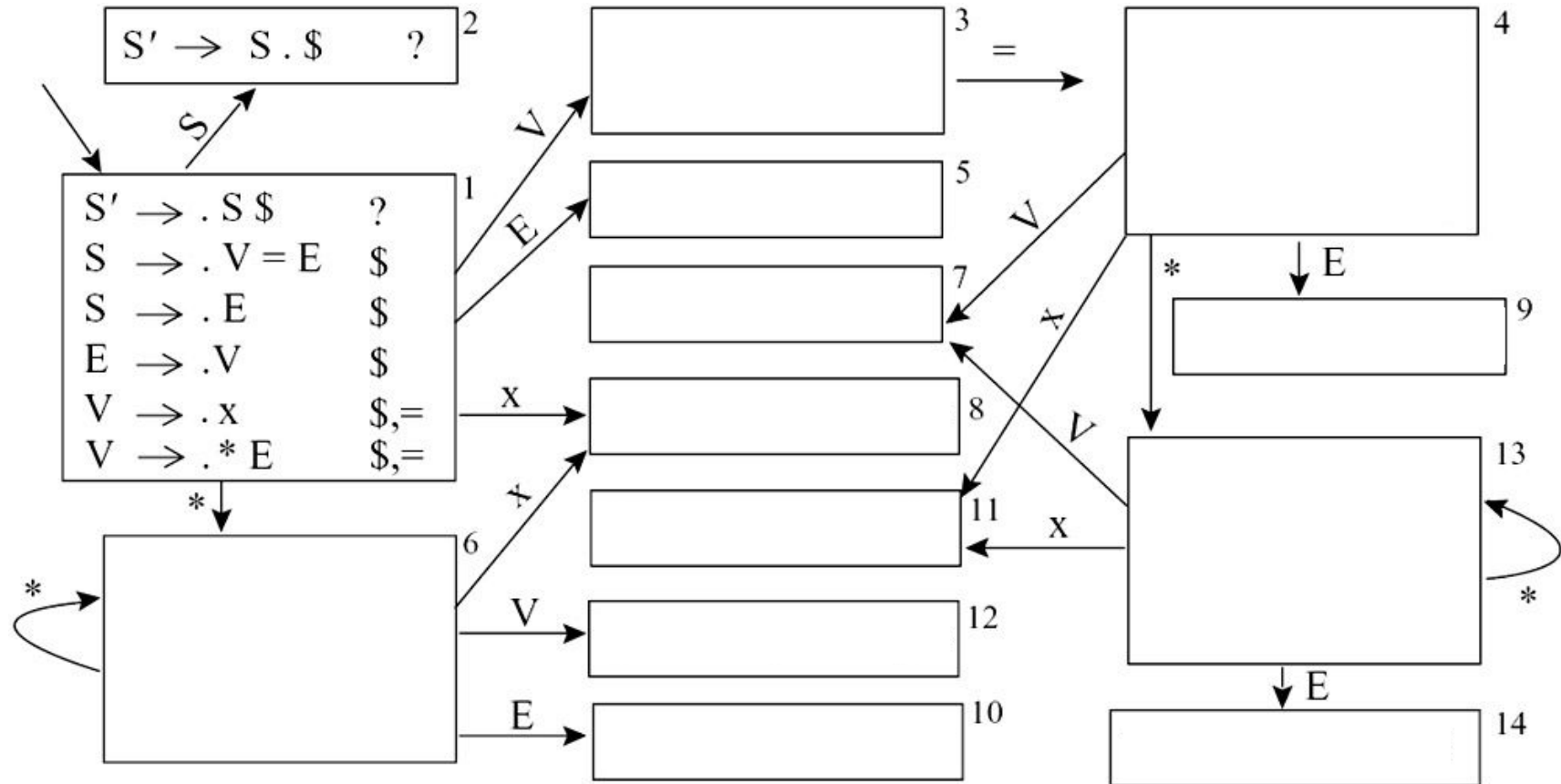
until E and T did not change in this iteration

Exemplo



- 0. $S' \rightarrow S \$$
- 1. $S \rightarrow V = E$
- 2. $S \rightarrow E$
- 3. $E \rightarrow V$
- 4. $V \rightarrow x$
- 5. $V \rightarrow * E$

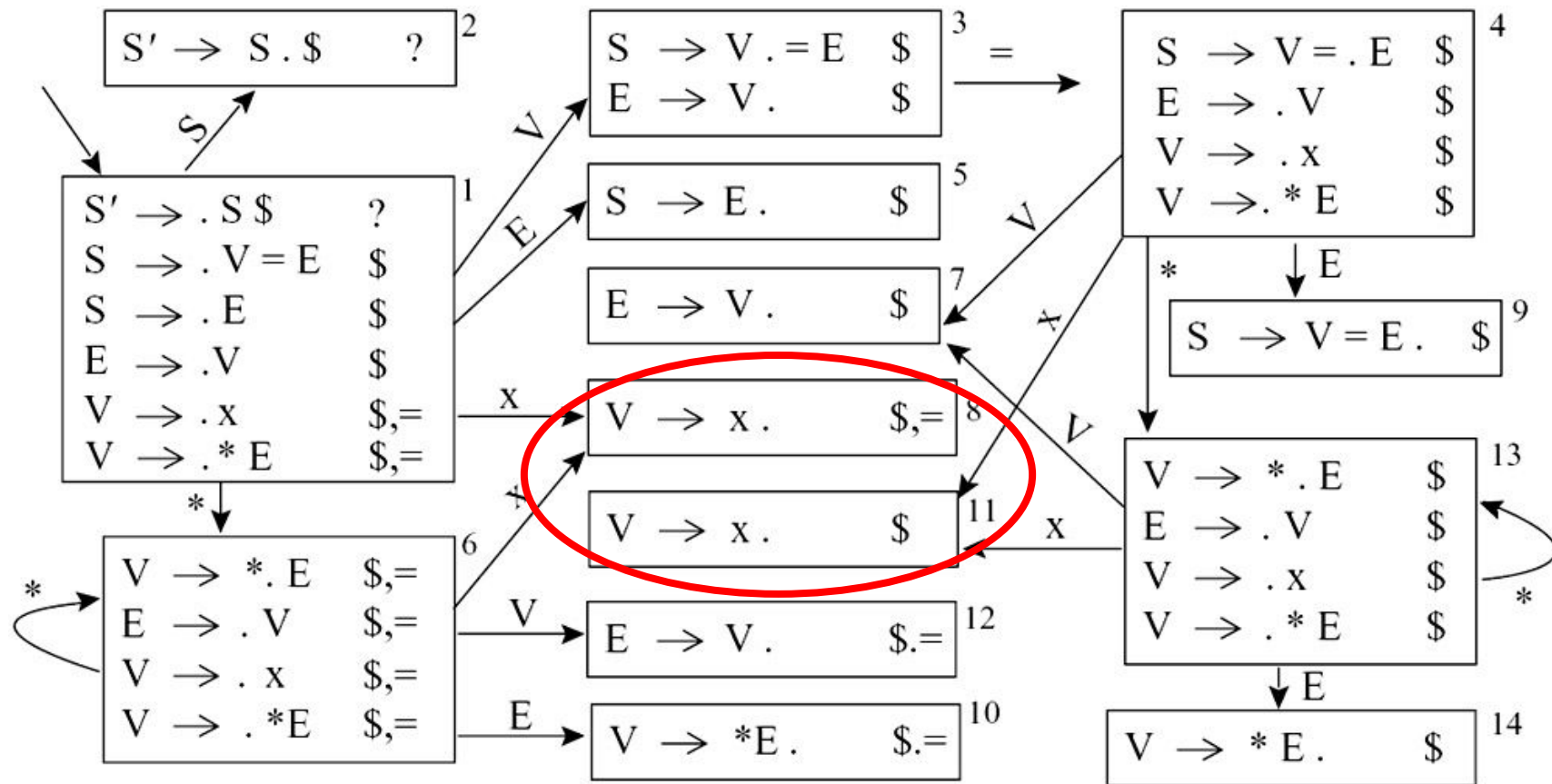
$S' \rightarrow . S \$$?
-------------------------	---



	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)

- O tamanho das tabelas LR(1) pode ser muito grande
- É possível reduzir unindo estados do DFA
 - Junte os estados que possuam os itens idênticos, exceto pelo lookahead
- Vejamos o exemplo anterior

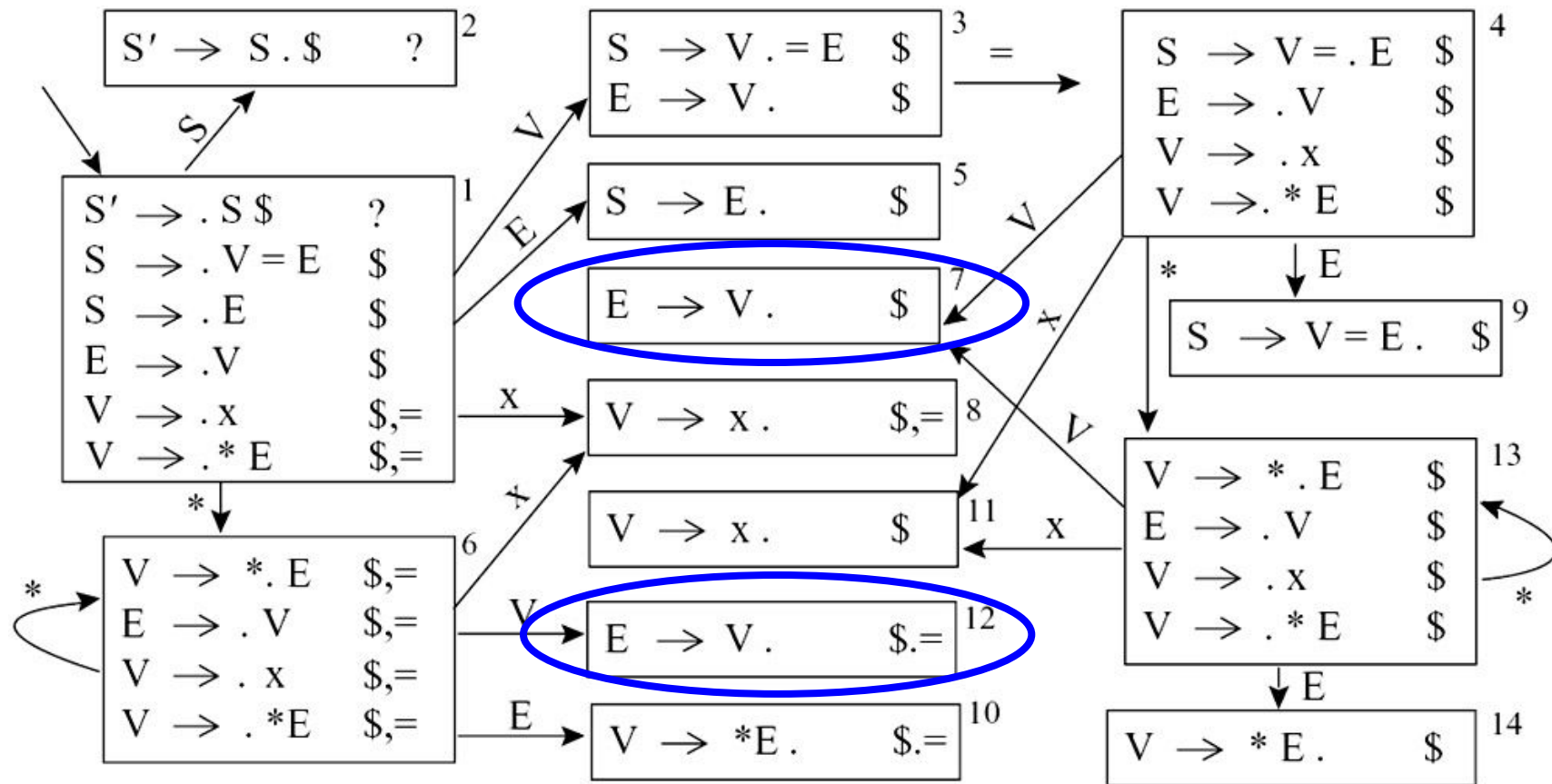


	x	*	=	\$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
7				r3			
→ 8			r4	r4			
9				r1			
10			r5	r5			
→ 11				r4			
12			r3	r3			
13	s11	s13				g14	g7
14				r5			

(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			

(b) LALR(1)

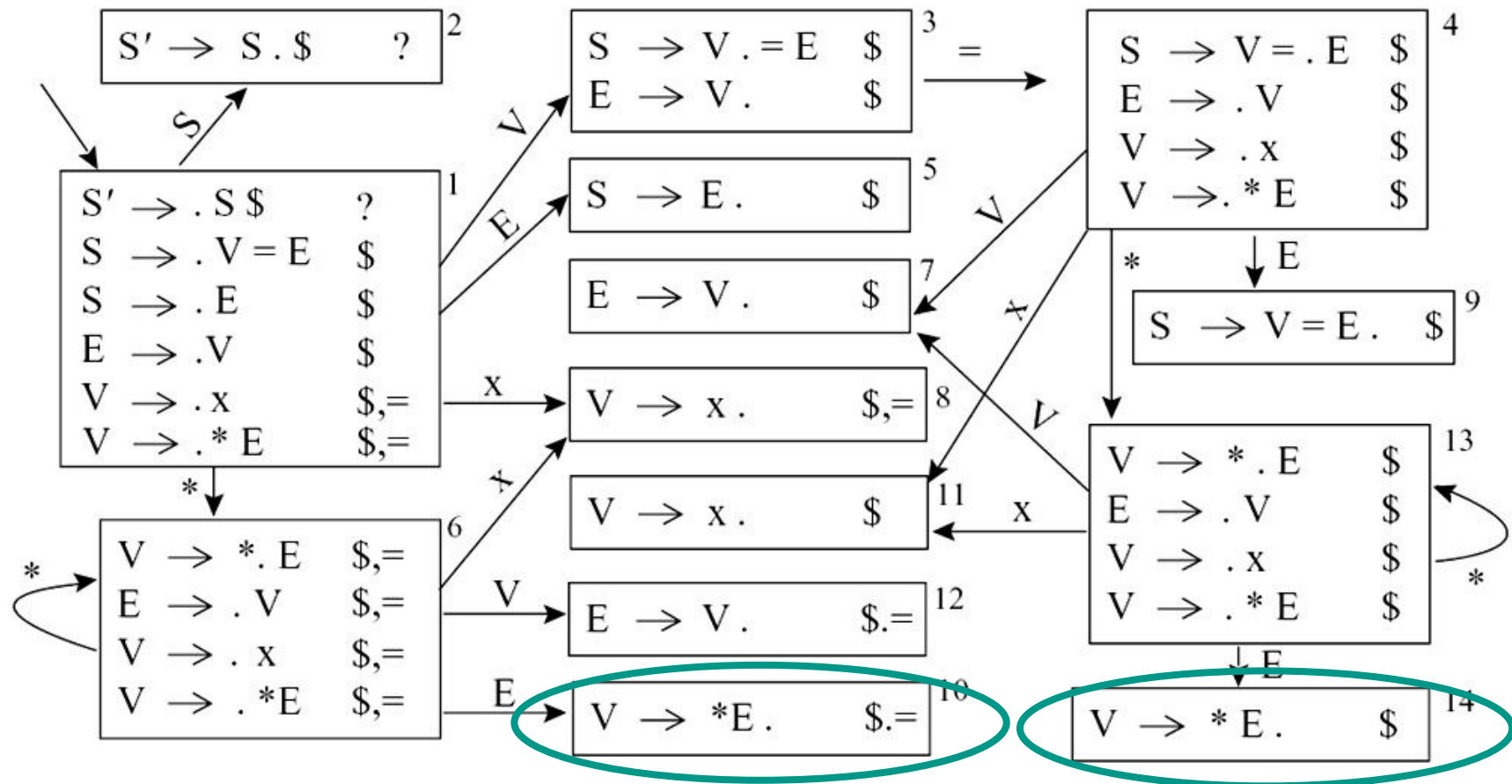


	x	*	=	\$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s11	s13				g9	g7
5				r2			
6	s8	s6				g10	g12
→ 7				r3			
8			r4	r4			
9				r1			
10			r5	r5			
11				r4			
→ 12			r3	r3			
13	s11	s13				g14	g7
14				r5			

(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			

(b) LALR(1)



	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)

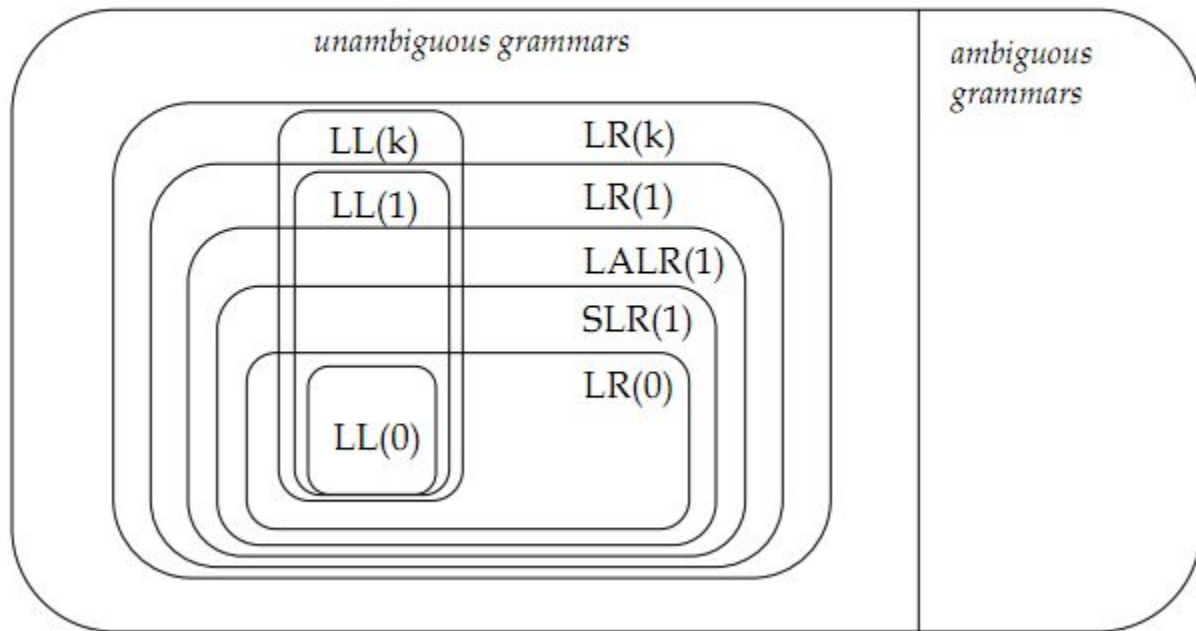
	x	*	=	\$	S	E	V
1	s8	s6			g2	g5	g3
2				a			
3			s4	r3			
4	s8	s6				g9	g7
5				r2			
6	s8	s6				g10	g7
7			r3	r3			
8			r4	r4			
9				r1			
→ 10			r5	r5			

(b) LALR(1)

- Pode gerar uma tabela com conflitos, onde a LR(1) não possuía
 - Na prática, o efeito de redução no uso de memória é bastante desejável
 - Em C cai de 10.000 para 350 estados
- A maioria das linguagens de programação é LALR(1)
- É o tipo mais usado em geradores automáticos de parser
 - Usado no PLY

LL(1) versus LR(k)

A picture is worth a thousand words:



Ambiguidade



- $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
- $S \rightarrow \text{if } E \text{ then } S$
- $S \rightarrow \text{other}$

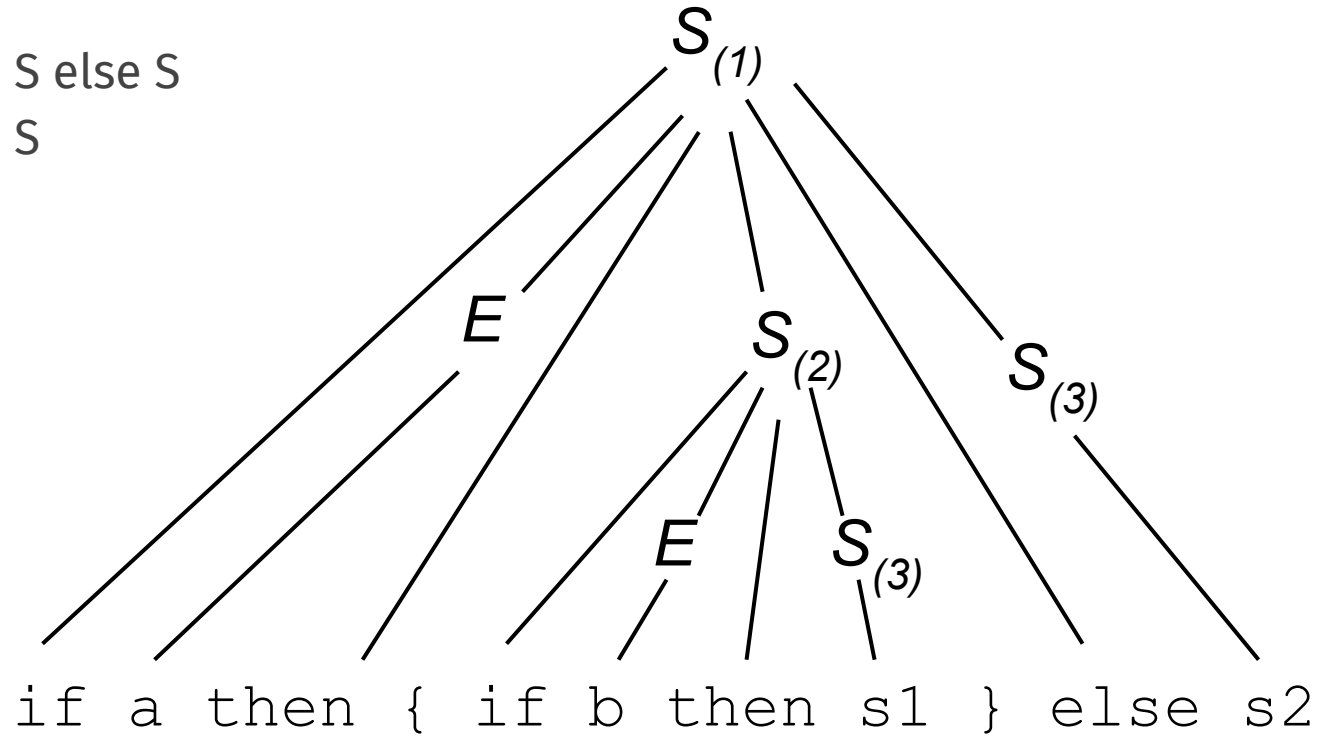
Como seria a parser tree para a sentença abaixo?
O “else” deve estar associado a qual “if”?

`if a then if b then s1 else s2`

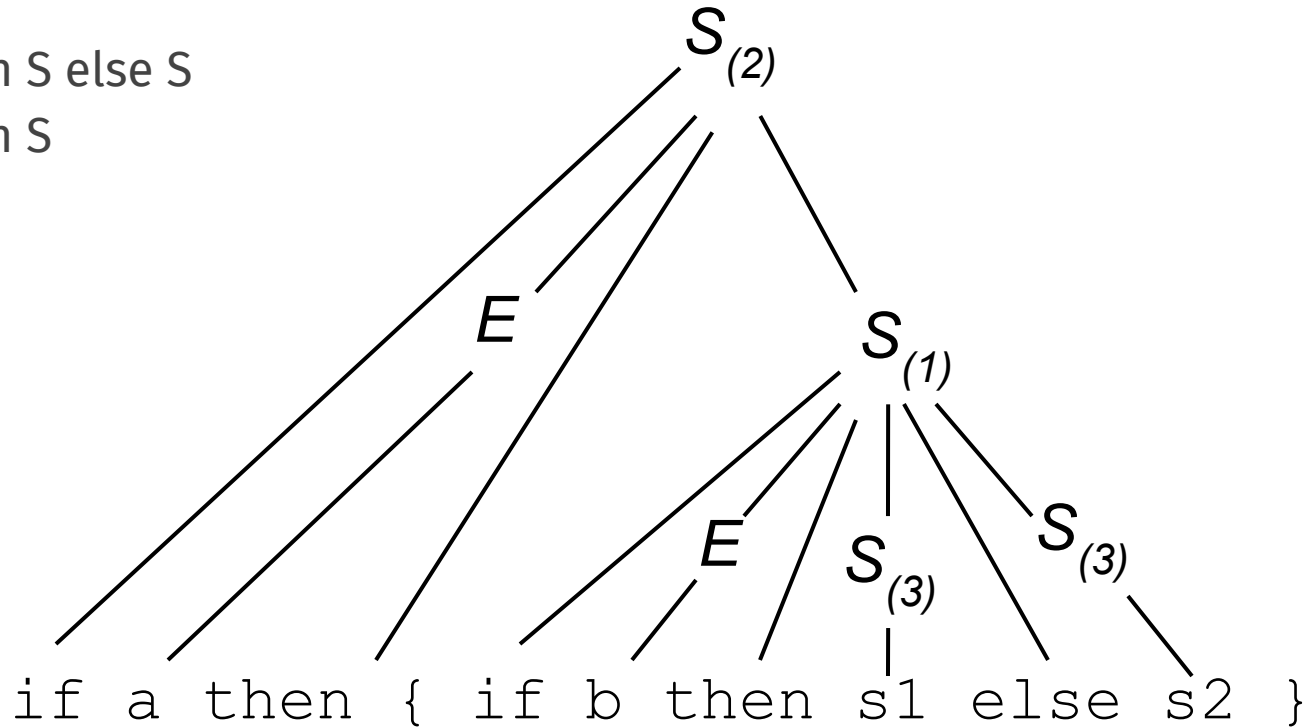
(1) `if a then { if b then s1 else s2 }`

(2) `if a then { if b then s1 } else s2`

- $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
- $S \rightarrow \text{if } E \text{ then } S$
- $S \rightarrow \text{other}$



- $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
- $S \rightarrow \text{if } E \text{ then } S$
- $S \rightarrow \text{other}$



- $S \rightarrow \text{if } E \text{ then } S \text{ else } S$
- $S \rightarrow \text{if } E \text{ then } S$
- $S \rightarrow \text{other}$

(1) `if a then { if b then s1 else s2 }`

(2) `if a then { if b then s1 } else s2`

$S \rightarrow \text{if } E \text{ then } S .$	else
$S \rightarrow \text{if } E \text{ then } S . \text{ else } S$	(any)

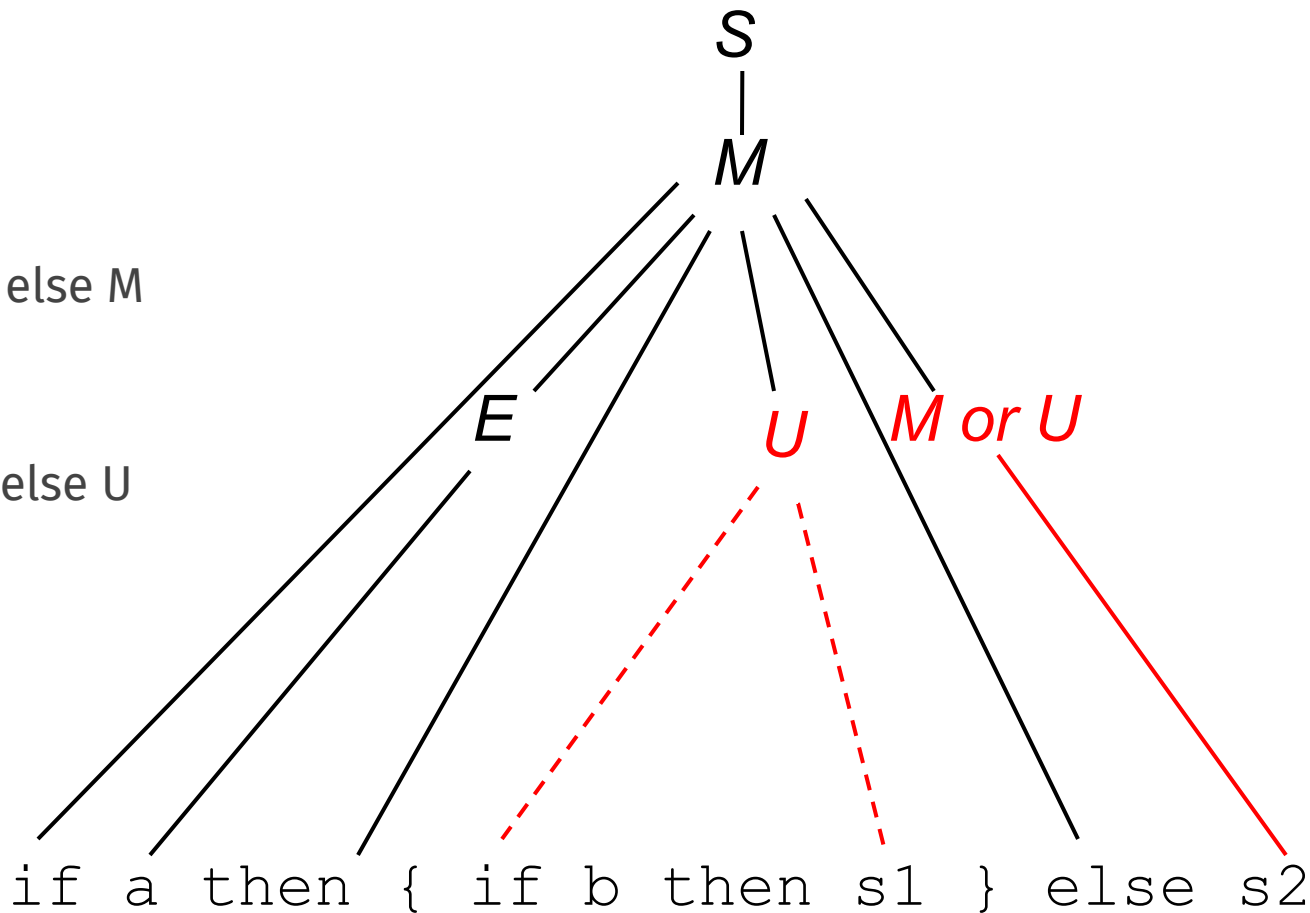
Conflito shift-reduce !

- $S \rightarrow M$
- $S \rightarrow U$
- $M \rightarrow \text{if } E \text{ then } M \text{ else } M$
- $M \rightarrow \text{other}$
- $U \rightarrow \text{if } E \text{ then } S$
- $U \rightarrow \text{if } E \text{ then } M \text{ else } U$

Como seria a parser tree para a sentença abaixo?

if a then if b then s1 else s2

- $S \rightarrow M$
- $S \rightarrow U$
- $M \rightarrow \text{if } E \text{ then } M \text{ else } M$
- $M \rightarrow \text{other}$
- $U \rightarrow \text{if } E \text{ then } S$
- $U \rightarrow \text{if } E \text{ then } M \text{ else } U$



- Às vezes, re-escrever gramática pode complicar muito
- Pode-se usar a gramática ambígua, decidindo os conflitos sempre por shift em casos desse tipo
- Somente aconselhável em casos bem conhecidos

- Nenhuma gramática ambígua é LR(k), para nenhum k
- Podemos usá-las se encontrarmos uma maneira de resolver os conflitos
- Relembrando um exemplo anterior...

- $E \rightarrow \text{id}$
- $E \rightarrow \text{num}$
- $E \rightarrow E * E$
- $E \rightarrow E / E$
- $E \rightarrow E + E$
- $E \rightarrow E - E$
- $E \rightarrow (E)$



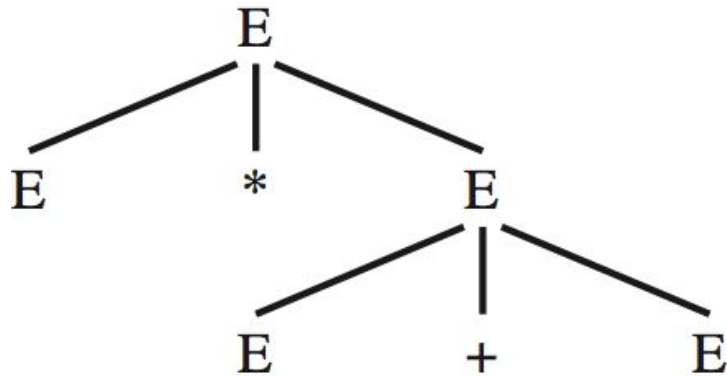
- $E \rightarrow E + T$
- $E \rightarrow E - T$
- $E \rightarrow T$
- $T \rightarrow T * F$
- $T \rightarrow T / F$
- $T \rightarrow F$
- $F \rightarrow \text{id}$
- $F \rightarrow \text{num}$
- $F \rightarrow (E)$

Tabela LR(1): muitos conflitos

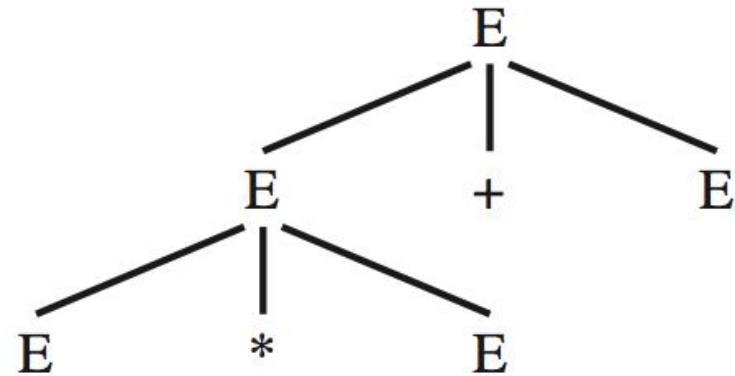
	id	num	+	-	*	/	()	\$	<i>E</i>
1	s2	s3					s4			<i>g</i> ⁷
2			r1	r1	r1	r1		r1	r1	
3			r2	r2	r2	r2		r2	r2	
4	s2	s3					s4			<i>g</i> ⁵
5								s6		
6			r7	r7	r7	r7		r7	r7	
7			s8	s10	s12	s14			a	
8	s2	s3					s4			<i>g</i> ⁹
9			s8,r5	s10,r5	s12,r5	s14,r5		r5	r5	
10	s2	s3					s4			<i>g</i> ¹¹
11			s8,r6	s10,r6	s12,r6	s14,r6		r6	r6	
12	s2	s3					s4			<i>g</i> ¹³
13			s8,r3	s10,r3	s12,r3	s14,r3		r3	r3	
14	s2	s3					s4			<i>g</i> ¹⁵
15			s8,r4	s10,r4	s12,r4	s14,r4		r4	r4	

Vejamos o estado 13:


$E \rightarrow E * E .$	$+$
$E \rightarrow E . + E$	(any)



Shift



Reduce



**precedence nonassoc EQ, NEQ;
precedence left PLUS, MINUS;
precedence left TIMES, DIV;
precedence right EXP;**

Indicaria que:

- + e - têm igual precedência e são associativos à esquerda
- * e / são associativos à esquerda e têm maior precedência que + e -

$E \rightarrow E * E .$	+
$E \rightarrow E . \textcircled{+} E$	(any)

Como seria resolvido?

- A precedência de uma regra é a mesma que a de seu terminal mais à direita
 - Precedência da regra 1 é a de *
 - Precedência da regra 2 é a de +
- * tem precedência sobre +
 - então reduz

1	$E \rightarrow E * E .$	+
2	$E \rightarrow E . + E$	(any)

- Podem ajudar
- Não devem ser abusivamente utilizadas
- Se não conseguir explicar ou achar um uso de precedências que resolva o seu problema, reescreva a gramática!

LR Error Recovery



- Tentativa de reportar o maior número de erros
- Mecanismos existentes variam de gerador para gerador
 - Verifique a documentação do gerador em uso

- Ajusta a pilha do parser no ponto onde o erro é detectado
- O parser continua a partir deste ponto
- Exemplo: YACC
 - Usa um símbolo especial: error
 - Controla o processo de recuperação
 - Onde ele aparecer na gramática, podem existir símbolos errôneos sendo consumidos

- $\text{exp} \rightarrow \text{ID}$
- $\text{exp} \rightarrow \text{exp} + \text{exp}$
- $\text{exp} \rightarrow (\text{exps})$
- $\text{exps} \rightarrow \text{exp}$
- $\text{exps} \rightarrow \text{exps} ; \text{exp}$
- $\text{exp} \rightarrow (\text{error})$
- $\text{exps} \rightarrow \text{error} ; \text{exp}$



Regras adicionadas
com token de sincronização

- A idéia é definir o fecho parênteses e o ponto-e-vírgula como tokens de sincronização
- Erros no meio de uma expressão causam avanço até o próximo token de sincronização
- error é um terminal: na tabela aparecem ações de shift

- Quando um erro é encontrado:
 1. Desempilha, se necessário, até atingir um estado onde haja a ação de shift para o token error. Pilha: (. error
 2. Shift para o token error. Pilha: (error .
 3. Descarta símbolos da entrada até que um símbolo com ação diferente de erro para o estado corrente seja alcançado. Pilha: (error .)
 4. No nosso exemplo, a ação do passo 3 sempre será shift. Pilha: (error) .
 5. Retoma procedimento normal.
- Cuidado com reduce
 - Não consome a entrada, podendo o mesmo símbolo continuar causando erros
 - Evitar regras do tipo $\text{exp} \rightarrow \text{error}$

Show that the following grammar is LALR(1) but not SLR(1).

$$S \rightarrow Aa \mid bAc \mid dc \mid bda$$
$$A \rightarrow d$$

Show that the following grammar is LR(1) but not LALR(1).

$$S \rightarrow Aa \mid bAc \mid Bc \mid bBa$$
$$A \rightarrow d$$
$$B \rightarrow d$$

Construct an SLR parsing table for the following grammar:

$$R \rightarrow R \mid R$$
$$R \rightarrow RR$$
$$R \rightarrow R^*$$
$$R \rightarrow (R)$$
$$R \rightarrow a$$
$$R \rightarrow b$$

Resolve the parsing action conflicts in such a way that regular expression will be parsed normally.

Resumo

- **Análise Ascendente**
 - Derivações/Reduções
 - **Parser LR**
 - Usa pilha
 - Ações Shift e Reduce
 - **Construção do LR(0)**
 - Autômato e tabela
 - **Conflitos**
 - Shift-Reduce
 - Reduce-Reduce
 - **SLR**
 - FOLLOW do reduce
-

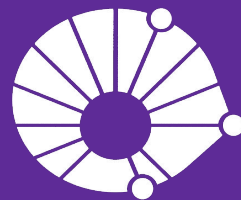
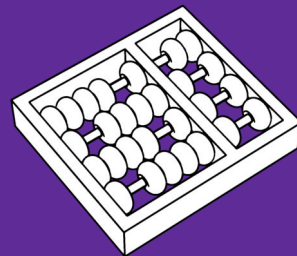
Leitura Recomendada

- Capítulo 3.4 do livro do Cooper.
- Capítulo 3.3 do livro do Appel.

Próxima Aula

- Geração do Parser
 - LR(1)
 - LALR

Obrigado!
Merci!



UNICAMP

Pallette



BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

BUBBLE

Lorem ipsum dolor
sit amet, consectetur
adipiscing elit.

DRACULA

Table Title	
Column 1	Column 2
One	Two
Three	Four

Table Title	
Column 1	Column 2
One	Two
Three	Four

Table Title	
Column 1	Column 2
One	Two
Three	Four

Table Title	
Column 1	Column 2
One	Two
Three	Four



	+	-	*	/
			⋮	
9				
11	...		s12	s14 ...
13	s8,r3	s10,r3	s12,r3	s14,r3
15			⋮	

Vejamos o estado 13:

$E \rightarrow E * E .$	$+$
$E \rightarrow E . + E$	(any)

Pilha (shift):

... $E * E$ (+)

Shift:

... $E * E +$

Chegando a:

... $E * E + E$

Pilha (reduce):

... $E * E$ (+)

Reduce:

... E (+)

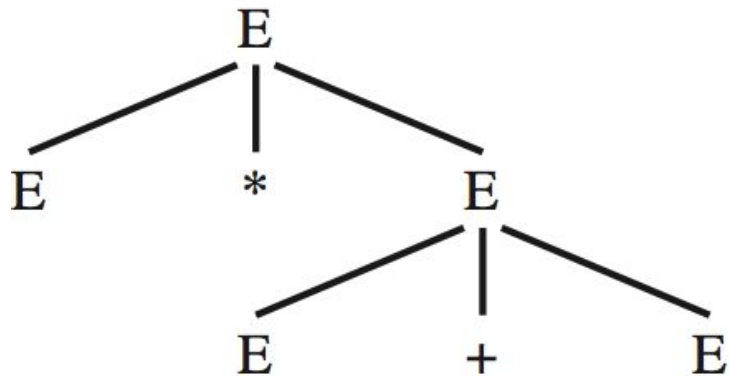
Chegando a:

... $E +$

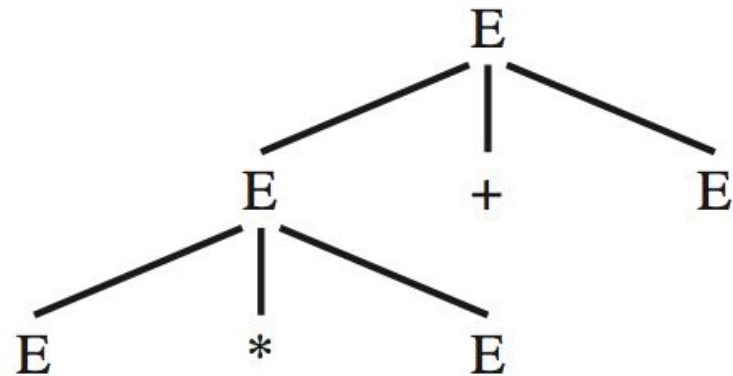
		+	-	*	/	
				⋮		
9		s8,r5	s10,r5	s12,r5	s14,r5	
11
13		r3	r3	r3	r3	
15		r4	r4			
				⋮		

Vejamos o estado 9:

$E \rightarrow E + E .$	$+$
$E \rightarrow E . + E$	(any)



Shift



Reduce

	+	-	*	/	
			⋮		
9	r5	r5	s12	s14	
11	...		s12	s14	...
13	r3	r3	r3	r3	
15	r4	r4			
			⋮		

```
%{ declarations of yylex and yyerror %}  
%token INT PLUS MINUS TIMES UMINUS  
%start exp
```

```
%left PLUS MINUS  
%left TIMES  
%left UMINUS  
%%
```

```
exp : INT  
    | exp PLUS exp  
    | exp MINUS exp  
    | exp TIMES exp  
    | MINUS exp %prec UMINUS
```


- Cuidado com ações semânticas:

```
statements: statements exp SEMICOLON
           | statements error SEMICOLON
           | /* empty */
```

```
exp : increment exp decrement
    | ID
```

```
increment: LPAREN { : nest=nest+1; : }
```

```
decrement: RPAREN { : nest=nest-1; : }
```

Aonde está o problema?

- Duas pilhas defasadas
 - Fila com K tokens antigos, N tokens na linguagem
 - $K.N$ (inserções) + K (remoções) + $K.N$ (alterações)
 - Reduções realizadas somente pela Old stack

