

Geração de Parser

Análise Ascendente

Hervé Yviquel

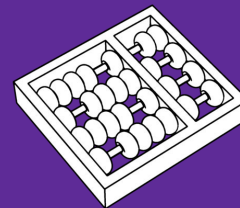
herve@ic.unicamp.br

Universidade Estadual de Campinas (Unicamp)

Instituto de Computação (IC)

Laboratório de Sistemas de Computação (LSC)

MC921 • Projeto e Construção de Compiladores • 2023 S2



UNICAMP

Aula Anterior

Resumo

- Gerar um Parser
- Parser Preditivo
- Conjuntos FIRST e FOLLOW
- Construção do Parser Preditivo
- Ambiguidade
- Recuperação de Erros

Aula de Hoje

Plano

- Análise Ascendente
- Parser LR
- Construção do LR(0)
- Conflitos

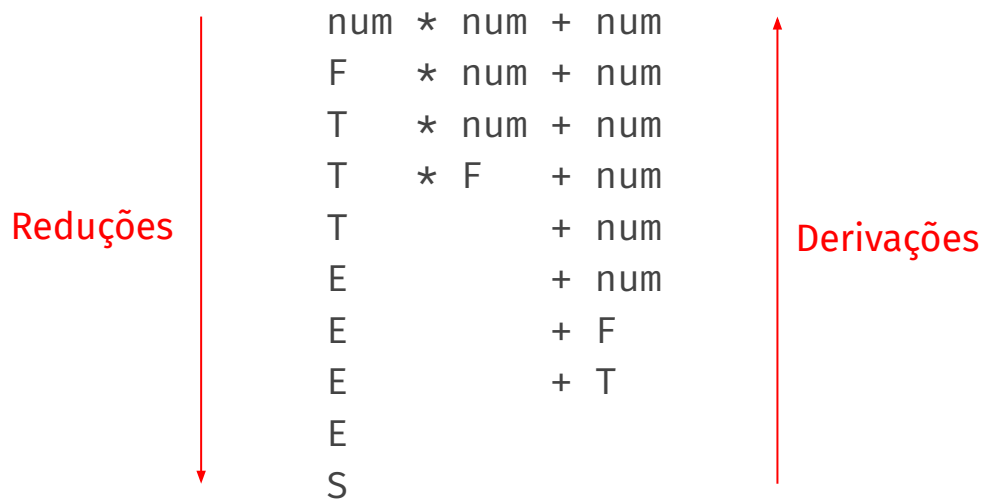
Análise Ascendente



- Parser LL(k) que vimos usam a análise descendente (ou *top-down*)
 - O ponto fraco da técnica LL(k) é precisar prever que produção usar com base nos primeiros k tokens do lado direito da produção
- Vamos ver agora uma outra estratégia de análise chamado de ascendente (ou *bottom-up*)
 - Usado para implementar parsers LR(k): Left-to-right parser, rightmost-derivation, k-token lookahead
 - LR(k) posterga a decisão até ter visto todo o lado direito de uma produção, mais o k próximos tokens da entrada
- A diferença mais visível entre as duas é a forma de construção da árvore
 - na análise descendente construímos a árvore de cima para baixo, começando pela raiz
 - na ascendente de baixo para cima, começando pelas folhas

- A análise ascendente é mais complicada de implementar
 - tanto para um analisador escrito à mão (o que é muito raro) quanto para geradores
- Mas é mais geral, o que quer dizer que impõe menos restrições à gramática
 - Por exemplo, recursão à esquerda e prefixos em comum não são problemas para as técnicas de análise ascendente
- Vamos usar um exemplo que deixa essas vantagens bem claras

- A análise ascendente analisa uma cadeia através de **reduções**
 - aplicando as regras da gramática ao contrário



- A sequência de reduções da análise ascendente equivale a uma derivação mais à direita, lida de trás pra frente
 - Para uma gramática não ambígua, cada entrada só pode ter uma única derivação mais à direita
 - Ou seja, a sequência de reduções também é única!
 - O trabalho do analisador é então achar qual a próxima redução que tem que ser feita a cada passo

Parser LR



- Para funcionar, o parser usa
 - a entrada
 - uma pilha
 - um autômato
- Dois tipos de ações:
 - **SHIFT**
 - Move o primeiro token para o topo da pilha (**Push**)
 - **REDUCE**
 - Escolhe uma regra de produção $X \rightarrow A B C$
 - Desempilha C, B, e A (**Pop**)
 - Empilha X (**Push** chamado de “**Goto**”)

input: a j d f **p** ...



X \rightarrow AB.C

C \rightarrow .p

...

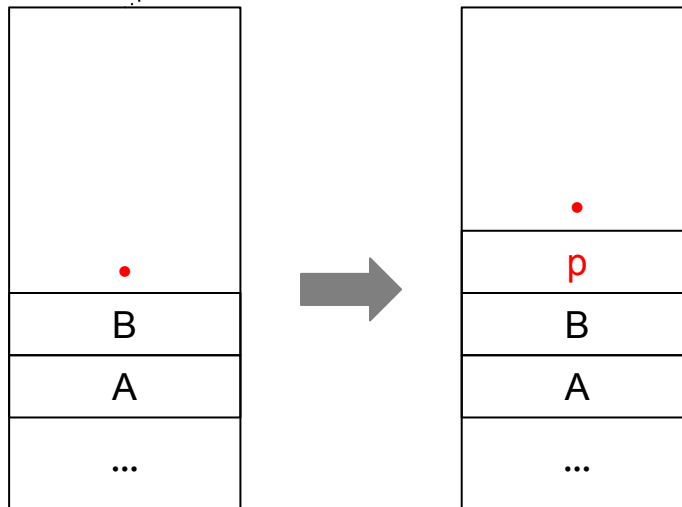


X \rightarrow AB.C

C \rightarrow p.

...

Pilha



push p

REDUCE = Pop + Goto

12

input: a j d f **p** ...



X → AB.C

C → p.

...

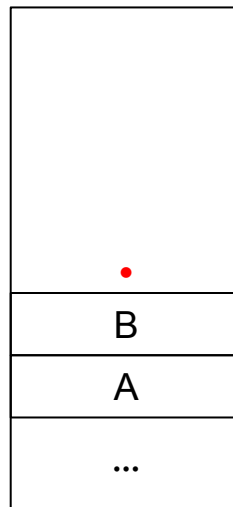
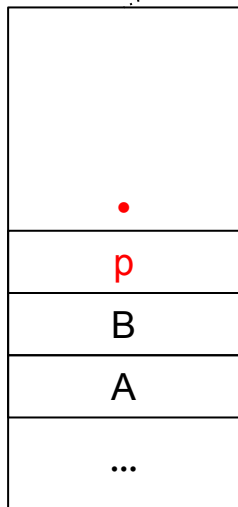


X → ABC.

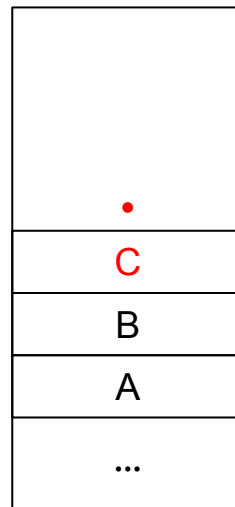
C → p.

...

Pilha



pop p



push C
(goto)

REDUCE = Pop + Goto

13

input: a j d f **p** ...

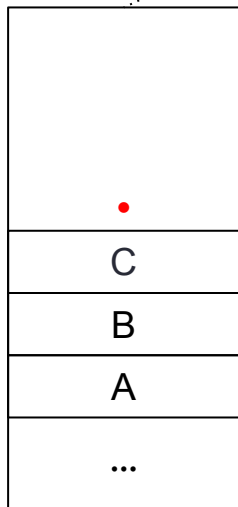


X → ABC.

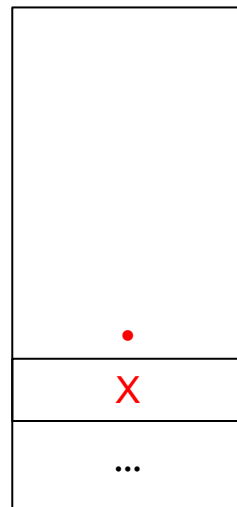
C → p.

...

Pilha



pop C, B, A



push X
(goto)

Exemplo de Parser LR



0. $S' \rightarrow S\$$
1. $S \rightarrow S; S$
2. $S \rightarrow \text{id} := E$
3. $S \rightarrow \text{print}(L)$
4. $E \rightarrow \text{id}$
5. $E \rightarrow \text{num}$
6. $E \rightarrow E + E$
7. $E \rightarrow (S, E)$
8. $L \rightarrow E$
9. $L \rightarrow L, E$

Derivação para:

$a := 7;$

$b := c + (d := 5 + 6, d)$

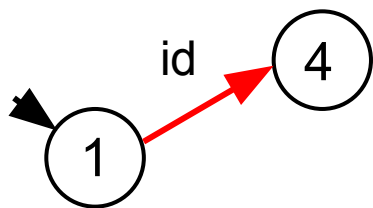
Stack	Input	Action
1	a := 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄	:= 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ := ₆	7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ := ₆ num ₁₀	; b := c + (d := 5 + 6 , d) \$	reduce $E \rightarrow \text{num}$
1 id ₄ := ₆ E ₁₁	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$

a := 7;

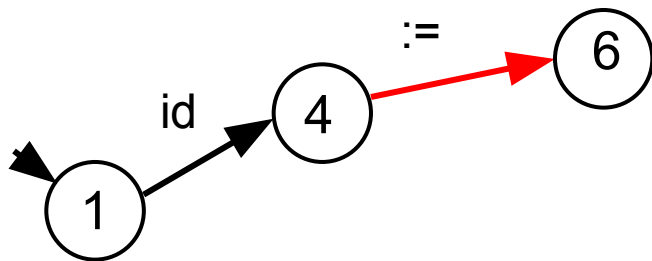
b := c + (d := 5 + 6, d)

- O parser sabe quando fazer um *shift* ou um *reduce* usando um DFA aplicado a pilha!
 - Implementado com uma tabela de transições
 - Vamos ver depois como construir este autômato
- As arestas são nomeadas com os símbolos que podem aparecer na pilha
- Vamos ver o funcionamento no exemplo anterior

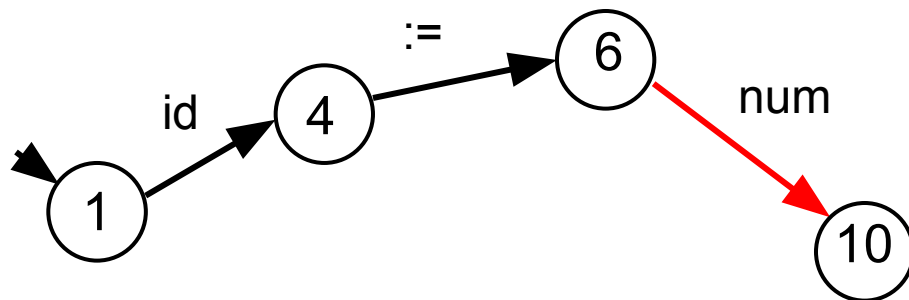
Stack	Input	Action
1	a := 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ ←	:= 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6	7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6 num ₁₀	; b := c + (d := 5 + 6 , d) \$	reduce $E \rightarrow \text{num}$
1 id ₄ :=6 E_{11}	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S_2	; b := c + (d := 5 + 6 , d) \$	shift



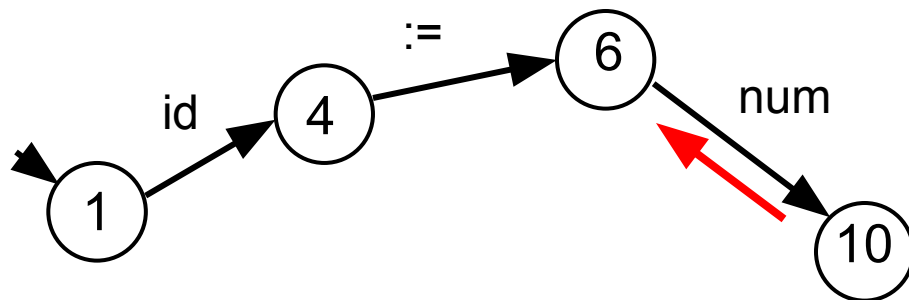
Stack	Input	Action
1	a := 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄	:= 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6 ←	7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6 num ₁₀	; b := c + (d := 5 + 6 , d) \$	reduce $E \rightarrow \text{num}$
1 id ₄ :=6 E_{11}	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S_2	; b := c + (d := 5 + 6 , d) \$	shift




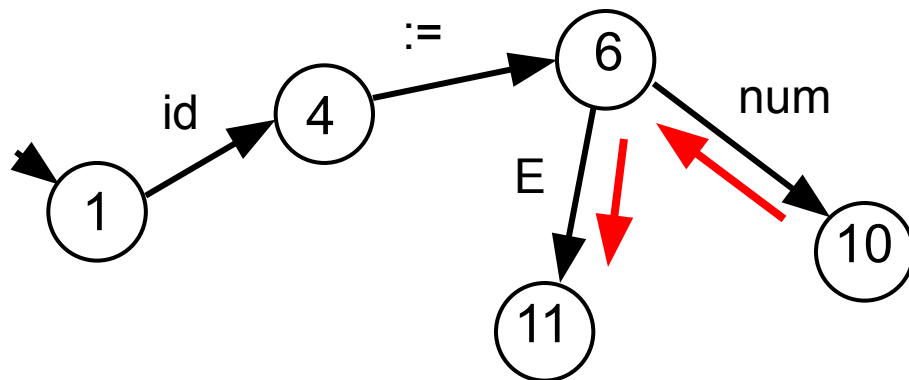
Stack	Input	Action
1	a := 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄	:= 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6	7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6 num ₁₀ ←	; b := c + (d := 5 + 6 , d) \$	reduce $E \rightarrow \text{num}$
1 id ₄ :=6 E ₁₁	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S ₂	; b := c + (d := 5 + 6 , d) \$	shift




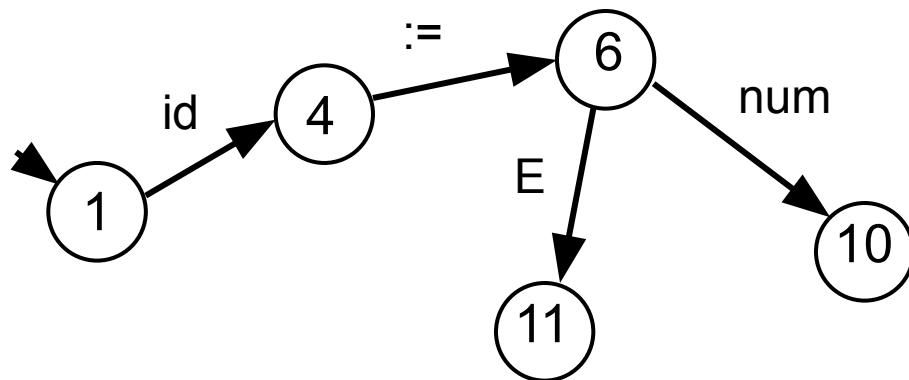
Stack		Input	Action
1		a := 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄		:= 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6		7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6	←	; b := c + (d := 5 + 6 , d) \$	reduce $E \rightarrow \text{num}$
1 id ₄ :=6 E ₁₁		; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S ₂		; b := c + (d := 5 + 6 , d) \$	shift



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 $E \rightarrow \text{num}$
1 id ₄ :=6 E ₁₁ 	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S ₂	; b := c + (d := 5 + 6 , d) \$	shift



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 $E \rightarrow \text{num}$
1 id ₄ :=6 E ₁₁ 	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S ₂	; b := c + (d := 5 + 6 , d) \$	shift



Stack

1
1 id₄
1 id₄ := 6
1 id₄ := 6 num₁₀
1
1 S₂

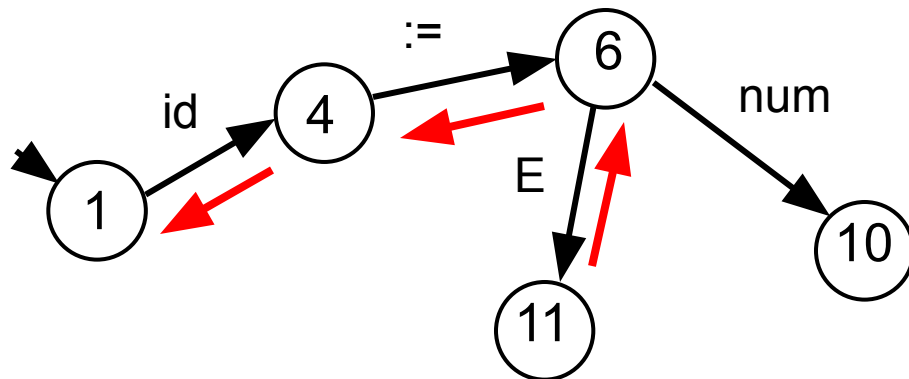


Input

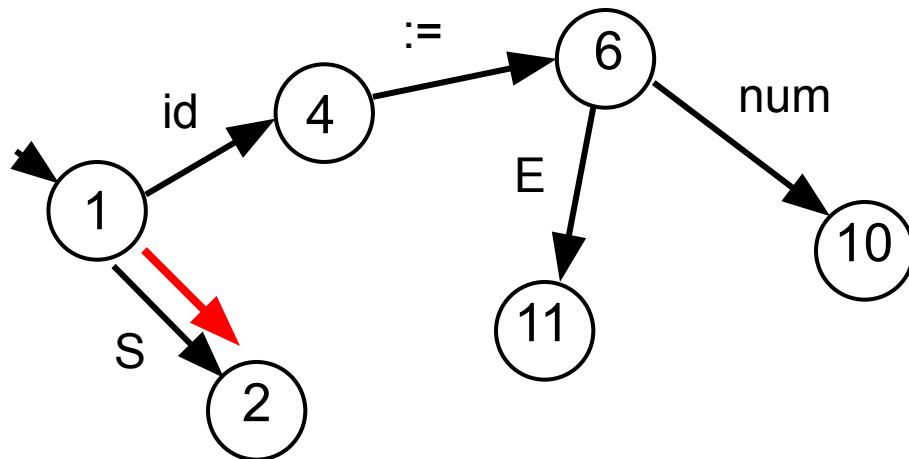
a := 7 ; b := c + (d := 5 + 6 , d) \$
:= 7 ; b := c + (d := 5 + 6 , d) \$
7 ; b := c + (d := 5 + 6 , d) \$
; b := c + (d := 5 + 6 , d) \$
; b := c + (d := 5 + 6 , d) \$
; b := c + (d := 5 + 6 , d) \$

Action

shift
shift
shift
reduce $E \rightarrow \text{num}$
reduce $S \rightarrow \text{id} := E$
shift



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 $E \rightarrow \text{num}$
1 id ₄ :=6 E ₁₁	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S ₂ ←	; b := c + (d := 5 + 6 , d) \$	shift




Exemplo inteiro

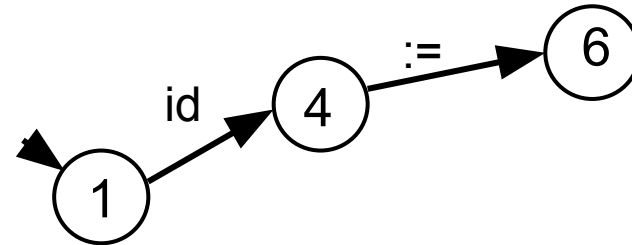
Stack	Input	Action
1	a := 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄	: = 7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6	7 ; b := c + (d := 5 + 6 , d) \$	shift
1 id ₄ :=6 num ₁₀	; b := c + (d := 5 + 6 , d) \$	reduce $E \rightarrow \text{num}$
1 id ₄ :=6 E ₁₁	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
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 $E \rightarrow \text{id}$
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 $E \rightarrow \text{num}$
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 $E \rightarrow \text{num}$
1 S ₂ ;3 id ₄ :=6 E ₁₁ +16 (8 id ₄ :=6 E ₁₁ +16 E ₁₇	, d) \$	reduce $E \rightarrow E + E$
1 S ₂ ;3 id ₄ :=6 E ₁₁ +16 (8 id ₄ :=6 E ₁₁	, d) \$	reduce $S \rightarrow \text{id} := E$
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 $E \rightarrow \text{id}$
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 $E \rightarrow (S, E)$
1 S ₂ ;3 id ₄ :=6 E ₁₁ +16 E ₁₇	\$	reduce $E \rightarrow E + E$
1 S ₂ ;3 id ₄ :=6 E ₁₁	\$	reduce $S \rightarrow \text{id} := E$
1 S ₂ ;3 S ₅	\$	reduce $S \rightarrow S; S$
1 S ₂	\$	accept

- Linhas correspondem aos estados
- Coluna correspondem aos terminais (*tokens*) e não terminais
- 4 tipos de ações
 - sn : Shift para o estado n
 - gn : Vá para o estado n
 - rk : Reduza pela regra k (cuidado o número não é o estado!)
 - a : Accept
 - : Error (entrada em branco)
- As arestas do DFA são as ações shift e goto
- No exemplo anterior, cada número indica o estado destino

Tabela do Exemplo

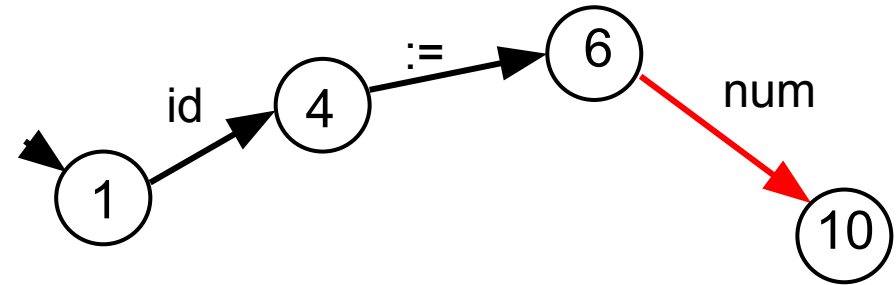
	id	num	print	;	,	+	:=	()	\$	<i>S</i>	<i>E</i>	<i>L</i>
1	s4		s7								g2		
2				s3						a			
3	s4		s7								g5		
4						s6							
5				r1	r1					r1			
6	s20	s10						s8				g11	
7								s9					
8	s4		s7								g12		
9	s20	s10						s8				g15	g14
10				r5	r5	r5			r5	r5			
11				r2	r2	s16				r2			
12				s3	s18								
13				r3	r3					r3			
14					s19				s13				
15					r8				r8				
16	s20	s10						s8				g17	
17				r6	r6	s16			r6	r6			
18	s20	s10						s8				g21	
19	s20	s10						s8				g23	
20				r4	r4	r4			r4	r4			
21									s22				
22				r7	r7	r7			r7	r7			
23					r9	s16			r9				

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 $E \rightarrow \text{num}$
1 id ₄ :=6 E ₁₁	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S ₂	; b := c + (d := 5 + 6 , d) \$	shift



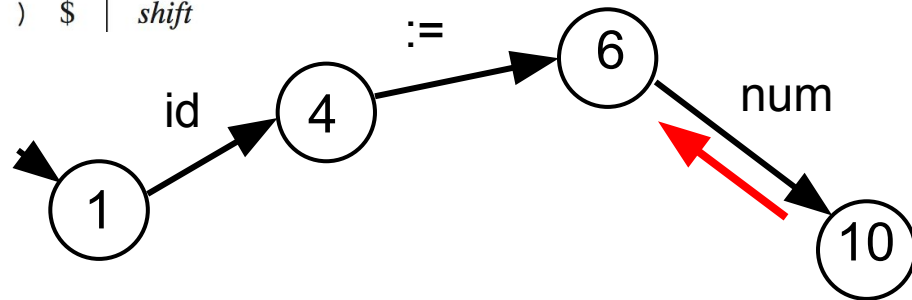
	id	num	print	;	,	+	:=	()	\$	S	E	L
6	s20	s10						s8				g11	
7								s9					
8	s4		s7								g12		
9	s20	s10						s8				g15	g14
10				r5	r5	r5			r5	r5			

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 $E \rightarrow \text{num}$
1 id ₄ :=6 E ₁₁	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S ₂	; b := c + (d := 5 + 6 , d) \$	shift



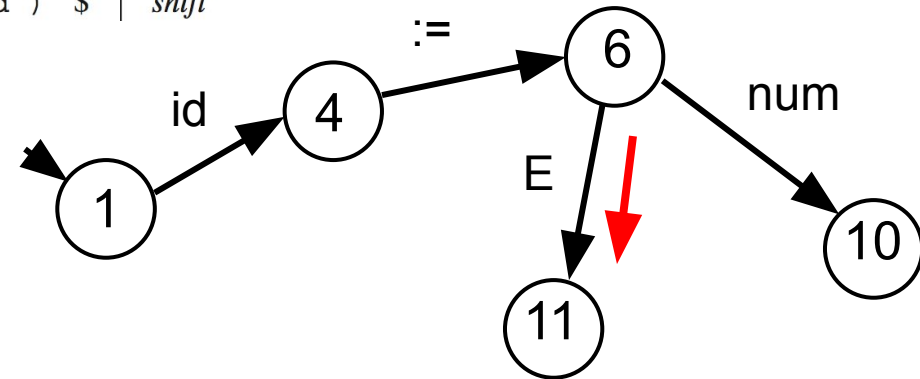
	id	num	print	;	,	+	:=	()	\$	S	E	L
6	s20	s10						s8				g11	
7								s9					
8	s4		s7								g12		
9	s20	s10						s8				g15	g14
10				r5	r5	r5			r5	r5			

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 $E \rightarrow \text{num}$
1 id ₄ :=6	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S ₂	; b := c + (d := 5 + 6 , d) \$	shift



	id	num	print	;	,	+	:=	()	\$	S	E	L
6	s20	s10						s8				g11	
7								s9					
8	s4		s7								g12		
9	s20	s10						s8				g15	g14
10				r5	r5	r5			r5	r5			

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 $E \rightarrow \text{num}$
1 id ₄ :=6 E ₁₁ ←	; b := c + (d := 5 + 6 , d) \$	reduce $S \rightarrow \text{id} := E$
1 S ₂	; b := c + (d := 5 + 6 , d) \$	shift



	id	num	print	;	,	+	:=	()	\$	S	E	L
6	s20	s10						s8				g11	
7								s9					
8	s4		s7								g12		
9	s20	s10						s8				g15	g14
10				r5	r5	r5			r5	r5			

Olha o estado da pilha superior e o símbolo de entrada para obter a ação; Se a ação for

- Shift(n):
 - Leia um token na entrada; empurre n na pilha
- Reduce(k):
 - Pop a pilha tantas vezes quanto o número de símbolos no lado direito da regra k
 - Seja X o símbolo do lado esquerdo da regra k
 - No estado agora no topo da pilha, procure X para obter "goto n"
 - Empurre n no topo da pilha
- Accept: Pare de analisar, informe o sucesso
- Error: Pare de analisar, informe a falha

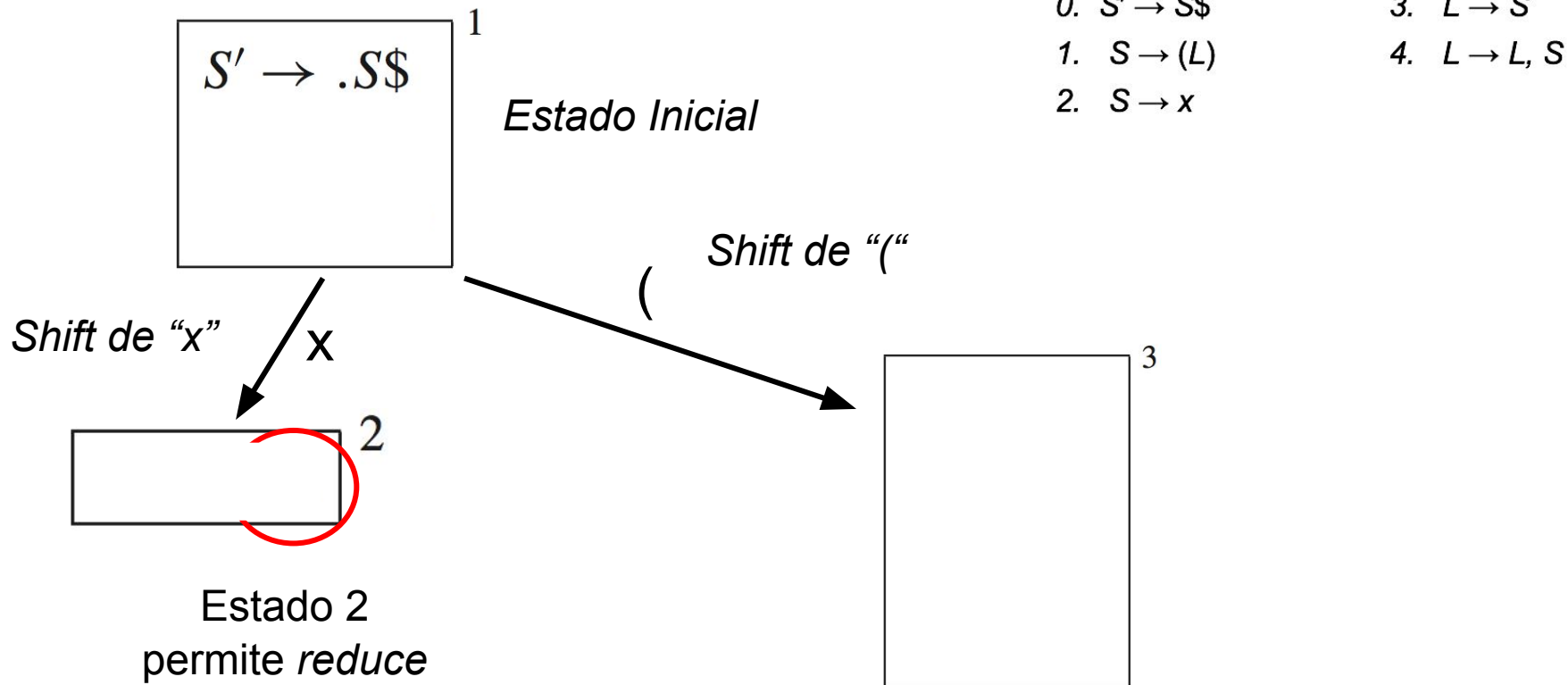
- O exemplo anterior mostrou o uso de 1 símbolo de lookahead
- Para k , a tabela terá colunas para todas as sequências de k tokens
 - $k > 1$ praticamente não é usado para compilação
- Maioria das linguagens de programação podem ser descritas por gramáticas LR(1)
- Agora, vamos ver como construir o autômato

Construção LR(0)



LR(0) são as gramáticas que podem ser analisadas olhando somente a pilha

- Para construir o autômato usamos o conceito de itens LR(0)
- Um item LR(0) de uma gramática é uma produção da gramática com uma marca (o ponto) no seu lado direito
 - a marca indica o estado da pilha
- Por exemplo, os itens para a produção $F \rightarrow (E)$ são:
 - $F \rightarrow . (E)$
 - $F \rightarrow (. E)$
 - $F \rightarrow (E .)$
 - $F \rightarrow (E) .$
- Uma produção vazia tem um único item LR(0)
- Itens com a marca no final são itens de redução



- Goto Action:
 - Imagine um shift de x no estado 1 seguido de redução pela produção de S correspondente
 - Todos os símbolos do lado direito da produção serão desempilhados e o parser vai executar um goto para S no estado 1
 - Isso se representa movendo-se o ponto para após o S e colocando este item em um novo estado (4)

$$\boxed{S' \rightarrow S.\4$

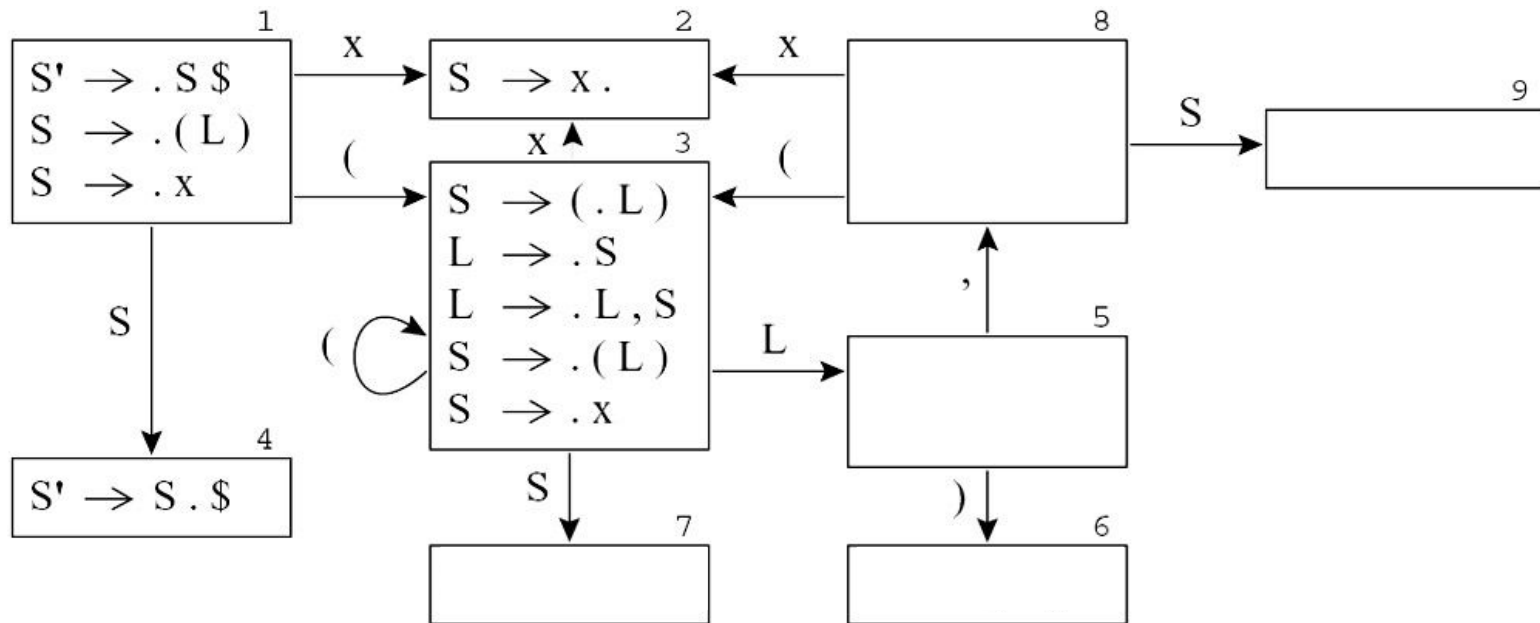
0. $S' \rightarrow S\$$

1. $S \rightarrow (L)$

2. $S \rightarrow x$

3. $L \rightarrow S$

4. $L \rightarrow L, S$



- 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  $A \rightarrow \alpha.X\beta$  in  $I$   
      for any production  $X \rightarrow \gamma$   
         $I \leftarrow I \cup \{X \rightarrow .\gamma\}$   
  until  $I$  does not change  
return  $I$ 
```

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


- Construção do parser LR(0)

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 $\mathbf{Goto}(I, X)$

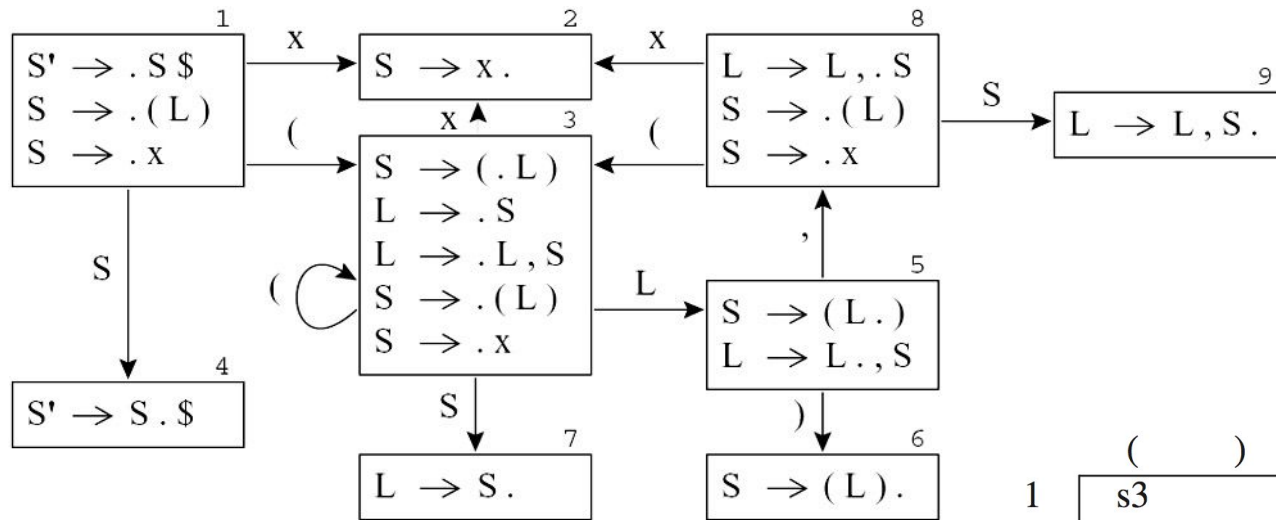
$T \leftarrow T \cup \{J\}$

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

until E and T did not change in this iteration

Tabela de Transição

42



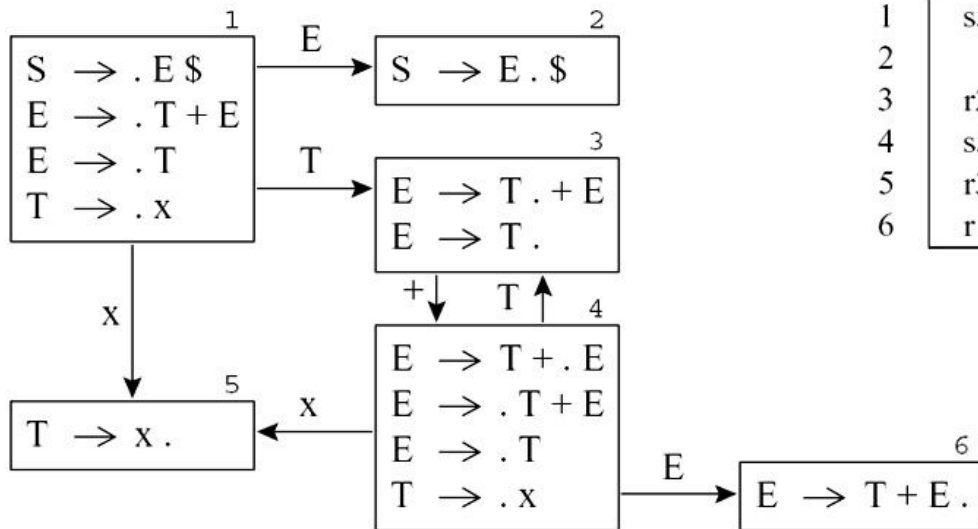
	()	x	,	\$	S	L
1	s3		s2			g4	
2	r2	r2	r2	r2	r2		
3	s3		s2			g7	g5
4					a		
5		s6		s8			
6	r1	r1	r1	r1	r1		
7	r3	r3	r3	r3	r3		
8	s3		s2			g9	
9	r4	r4	r4	r4	r4		

Conflitos



- Análise ascendente usa a análise shift-reduce como base
- Problemas na gramática (como ambiguidade), ou limitações da técnica específica adotada, pode levar a conflitos
 1. Conflito **shift-reduce** é quando o analisador não tem como decidir entre uma (ou mais) ações de shift e uma ação reduce, o que normalmente acontece por limitações da técnica escolhida
 2. Conflito **reduce-reduce** é quando o analisador não tem como decidir entre duas ou mais ações de reduce, o que normalmente é um bug na gramática

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



Não é LR(0)

	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		

- Colocar reduções somente onde indicado pelo conjunto FOLLOW
 - Ex: $\text{FOLLOW}(E) = \{\$, a\}$

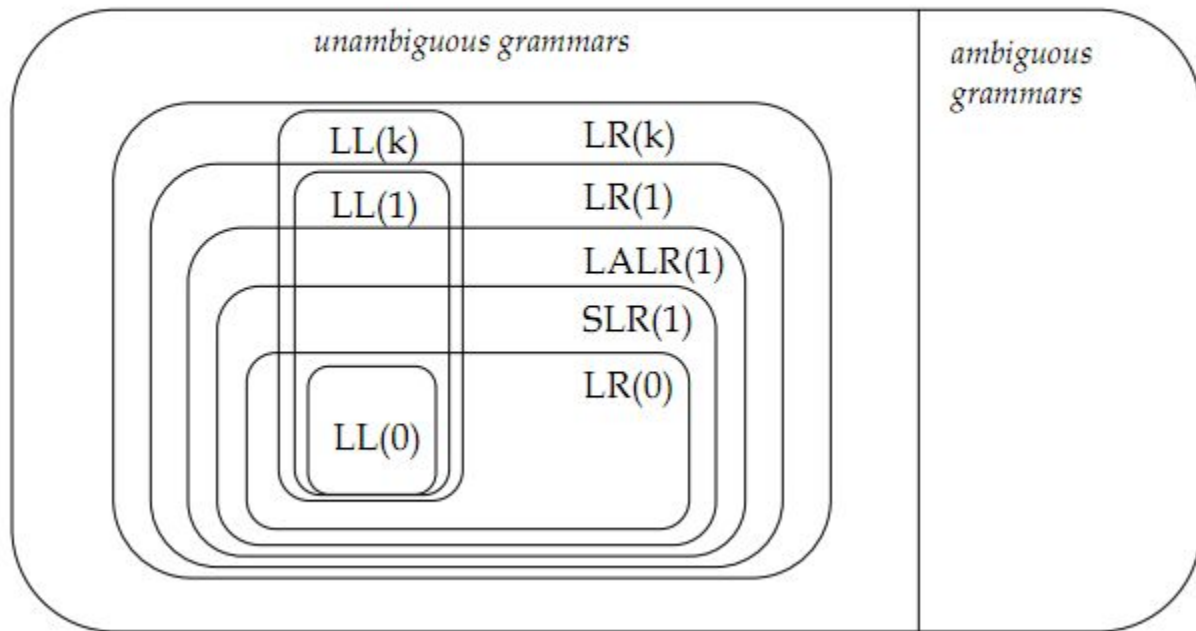
0. $S \rightarrow E \$$
1. $E \rightarrow T + E$
2. $E \rightarrow T$
3. $T \rightarrow x$

É SLR!!!

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

LL(1) versus LR(k)

A picture is worth a thousand words:



Exercícios



Construa os estados LR(0) para essa gramática e, em seguida, determine se ela é uma gramática SLR

$S \rightarrow B \$$

$B \rightarrow id P$

$B \rightarrow id (E]$

$P \rightarrow$

$P \rightarrow (E)$

$E \rightarrow B$

$E \rightarrow B, E$

State 0:

$S \rightarrow \bullet B \$$
 $B \rightarrow \bullet id P$
 $B \rightarrow \bullet id^*(E)$

State 1:

$S \rightarrow B \bullet \$$

State 2:

$B \rightarrow id \bullet P$
 $B \rightarrow id \bullet^*(E)$
 $P \rightarrow \bullet$
 $P \rightarrow \bullet(E)$

State 3:

$B \rightarrow id P \bullet$

State 4:

$B \rightarrow id^* \bullet(E)$

State 5:

$P \rightarrow (\bullet E)$
 $E \rightarrow \bullet B$
 $E \rightarrow \bullet B, E$
 $B \rightarrow \bullet id P$
 $B \rightarrow \bullet id^*(E)$

State 6:

$B \rightarrow i^* (\bullet E)$
 $E \rightarrow \bullet B$
 $E \rightarrow \bullet B, E$
 $B \rightarrow \bullet i P$
 $B \rightarrow \bullet i^*(E)$

State 7:

$P \rightarrow (E \bullet)$

State 8:

$E \rightarrow B \bullet$
 $E \rightarrow B \bullet, E$

State 9:

$B \rightarrow i^* (E \bullet)$

State 10:

$P \rightarrow (E) \bullet$

State 11:

$E \rightarrow B, \bullet E$
 $E \rightarrow \bullet B$
 $E \rightarrow \bullet B, E$
 $B \rightarrow \bullet i P$
 $B \rightarrow \bullet i^*(E)$

State 12:

$B \rightarrow i^* (E) \bullet$

State 13:

$E \rightarrow B, E \bullet$

$S \rightarrow B \$$
 $B \rightarrow id P$
 $B \rightarrow id (E)$
 $P \rightarrow$
 $P \rightarrow (E)$
 $E \rightarrow B$
 $E \rightarrow B, E$

Tem conflito no estado 2 e 8.

$\text{Follow}(E) = \{ \textcolor{red}{[}, \textcolor{red}{)} \}$

$\text{Follow}(P) = \{ \}$ U $\text{Follow}(B) = \{ \$, , \}$ U $\text{Follow}(E) = \{ \$, , , \textcolor{red}{[}, \textcolor{red}{)} \}$

Se $A \rightarrow pB$ é uma produção, então tudo em $\text{FOLLOW}(A)$ está em $\text{FOLLOW}(B)$

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
-

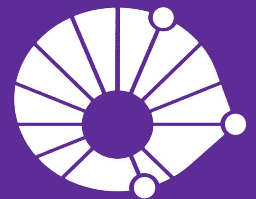
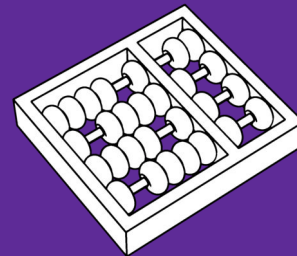
Leitura Recomendada

- 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

