Análise Sintática LL(K)

Left-to-right parsing, Leftmost derivation, K-symbol lookahead

NON-TERMINAL	INPUT SYMBOL					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to$	$E' \to$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to$	$T' \to *FT'$		$T' \to$	$T' \to$
F	$F \to \mathbf{id}$			$F \to (E)$		
S	$S \to E\$$			$S \to E\$$		

MATCHED	STACK	INPUT	ACTION
	$E\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	
	$TE'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $E \to TE'$
	$FT'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $T \to FT'$
	$\mathbf{id}\ T'E'\$$	$\mathbf{id} + \mathbf{id} * \mathbf{id}\$$	output $F \to \mathbf{id}$
$\mathbf{id}$	$T'E'\$$	$+\ \mathbf{id} * \mathbf{id}\$$	match $\mathbf{id}$
$\mathbf{id}$	$E'\$$	$+\ \mathbf{id} * \mathbf{id}\$$	output $T' \to \epsilon$
$\mathbf{id}$	$+\ TE'\$$	$+\ \mathbf{id} * \mathbf{id}\$$	output $E' \to +\ TE'$
$\mathbf{id} +$	$TE'\$$	$\mathbf{id} * \mathbf{id}\$$	match $+$
$\mathbf{id} +$	$FT'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $T \to FT'$
$\mathbf{id} +$	$\mathbf{id}\ T'E'\$$	$\mathbf{id} * \mathbf{id}\$$	output $F \to \mathbf{id}$
$\mathbf{id} + \mathbf{id}$	$T'E'\$$	$*\ \mathbf{id}\$$	match $\mathbf{id}$
$\mathbf{id} + \mathbf{id}$	$*\ FT'E'\$$	$*\ \mathbf{id}\$$	output $T' \to *\ FT'$
$\mathbf{id} + \mathbf{id} *$	$FT'E'\$$	$\mathbf{id}\$$	match $*$
$\mathbf{id} + \mathbf{id} *$	$\mathbf{id}\ T'E'\$$	$\mathbf{id}\$$	output $F \to \mathbf{id}$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$T'E'\$$	$\$$	match $\mathbf{id}$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$E'\$$	$\$$	output $T' \to \epsilon$
$\mathbf{id} + \mathbf{id} * \mathbf{id}$	$\$$	$\$$	output $E' \to \epsilon$

Análise Sintática LR(K)

O ponto fraco da técnica LL(K) é precisar prever que produção usar com base nos primeiros K *tokens* do lado direito da produção.

A técnica LR(K) posterga a decisão até ter visto todo o lado direito de uma produção, mais os k próximos *tokens* da entrada.

Left-to-right parsing, Rightmost-derivation, K-symbol lookahead

O *parser* tem uma pilha e a entrada.

Os primeiros k *tokens* da entrada formam o *lookahead*

LR Parsing

O *parser* possui uma pilha e a entrada

O primeiros K *tokens* da entrada formam o lookahead

Possui ações a serem executadas:

- SHIFT: move o primeiro *token* para o topo da pilha
- REDUCE:
  - Escolhe uma produção X → ABC;
  - Desempilha C, B e A
  - Empilha X (GOTO)

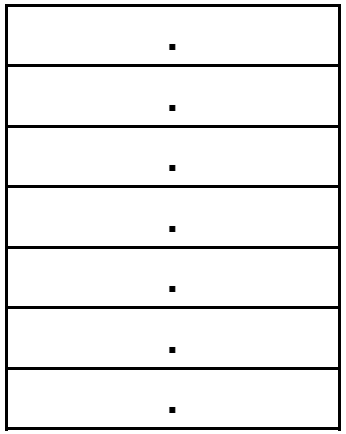
LR Parsing

$X \rightarrow A\ B\ C$

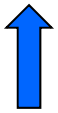
$A \rightarrow$ **a**

$B \rightarrow$ **b**

$C \rightarrow$ **c**



entrada: a b c



LR Parsing

$X \rightarrow .A\ B\ C$

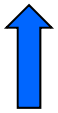
$A \rightarrow$ **a**

$B \rightarrow$ **b**

$C \rightarrow$ **c**

.
.
.
.
.
.
.

entrada: a b c



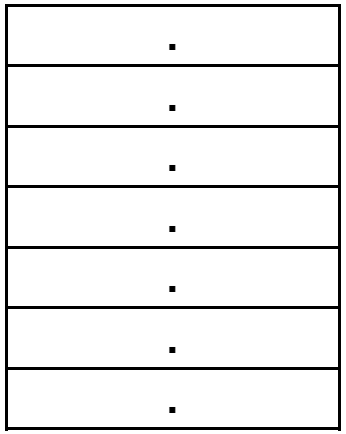
LR Parsing

$X \rightarrow .A\ B\ C$

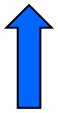
$A \rightarrow .a$

$B \rightarrow b$

$C \rightarrow c$



entrada: a b c



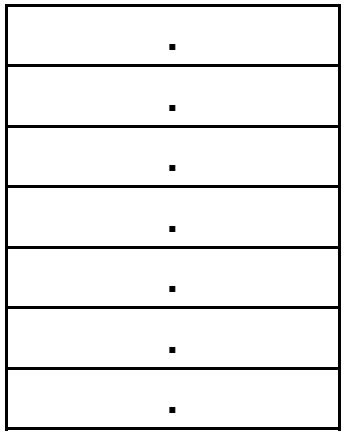
LR Parsing

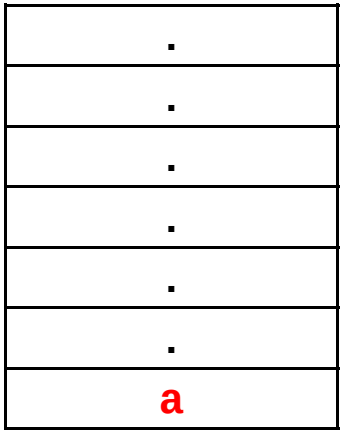
$X \rightarrow .A\ B\ C$

$A \rightarrow a.$

$B \rightarrow b$

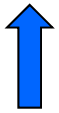
$C \rightarrow c$





push a (SHIFT)

entrada: a b c



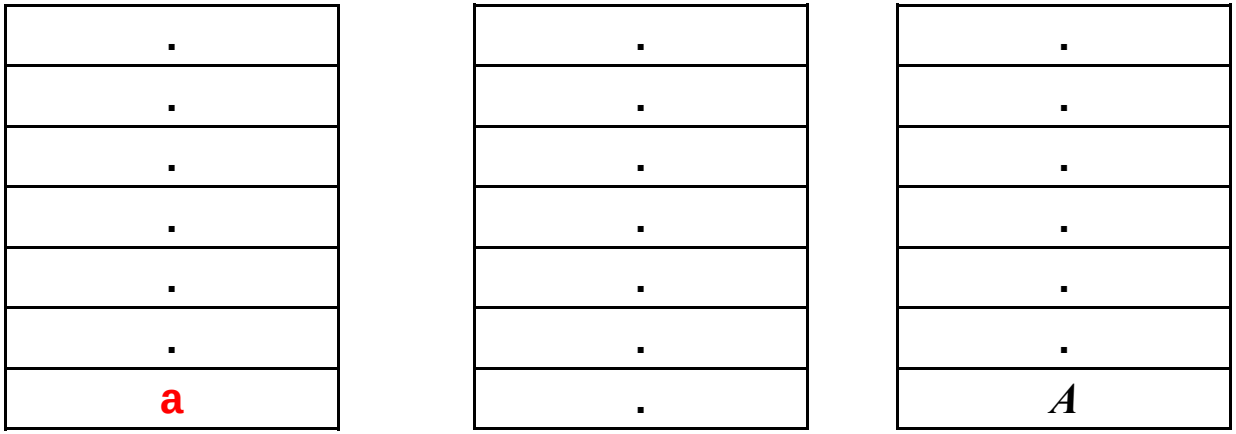
LR Parsing

$X \rightarrow A \bullet B\ C$

$A \rightarrow a \bullet$

$B \rightarrow b$

$C \rightarrow c$



entrada: a b c

$A \rightarrow a$

LR Parsing

$X \rightarrow A \,.\, B\ C$

$A \rightarrow$ **a**

$B \rightarrow \,.\,$ **b**

$C \rightarrow$ **c**

.
.
.
.
.
.
A

entrada: a b c

$A \rightarrow$ **a**

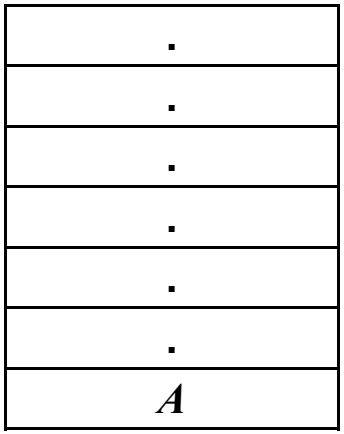
LR Parsing

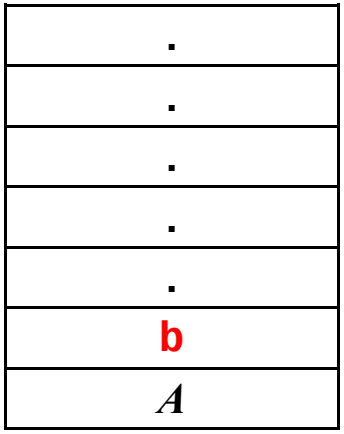
$X \rightarrow A \bullet B\ C$

$A \rightarrow$ **a**

$B \rightarrow$ **b** $\bullet$

$C \rightarrow$ **c**





push b (SHIFT)

entrada: a b c

$A \rightarrow$ **a**

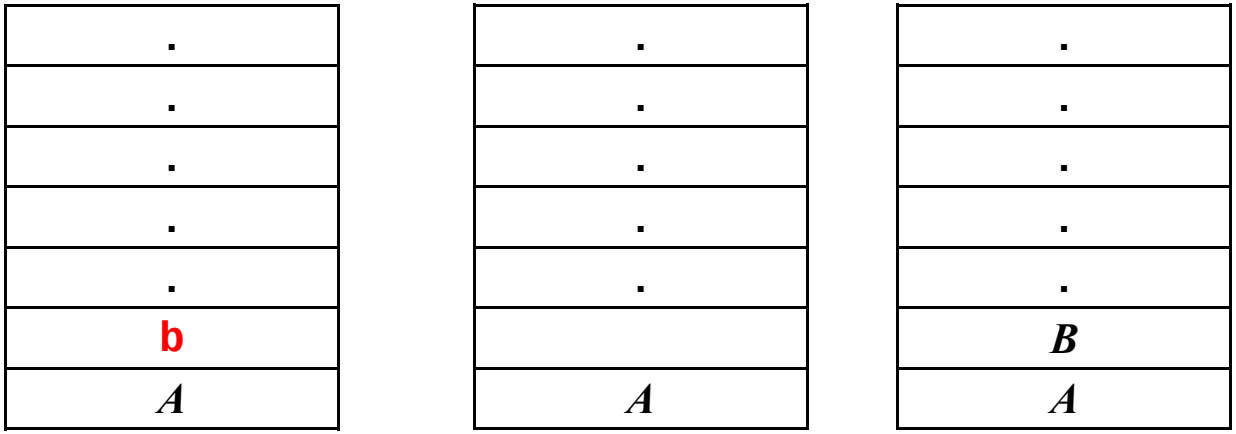
LR Parsing

$X \rightarrow A\ B\ .\ C$

$A \rightarrow a$

$B \rightarrow b\ .$

$C \rightarrow c$



pop b (REDUCE)

push B (GOTO)

entrada: a b c

$B \rightarrow b$

$A \rightarrow a$

LR Parsing

$X \rightarrow A\ B\ .\ C$

$A \rightarrow$ **a**

$B \rightarrow$ **b**

$C \rightarrow .$ **c**

.
.
.
.
.
B
A

entrada: a b c

$B \rightarrow$ **b**

$A \rightarrow$ **a**

LR Parsing

$X \rightarrow A\ B\ .\ C$

$A \rightarrow$ a

$B \rightarrow$ b

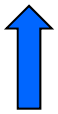
$C \rightarrow$ c .

.
.
.
.
.
B
A

.
.
.
.
c
B
A

push c (SHIFT)

entrada: a b c



$B \rightarrow$ b

$A \rightarrow$ a

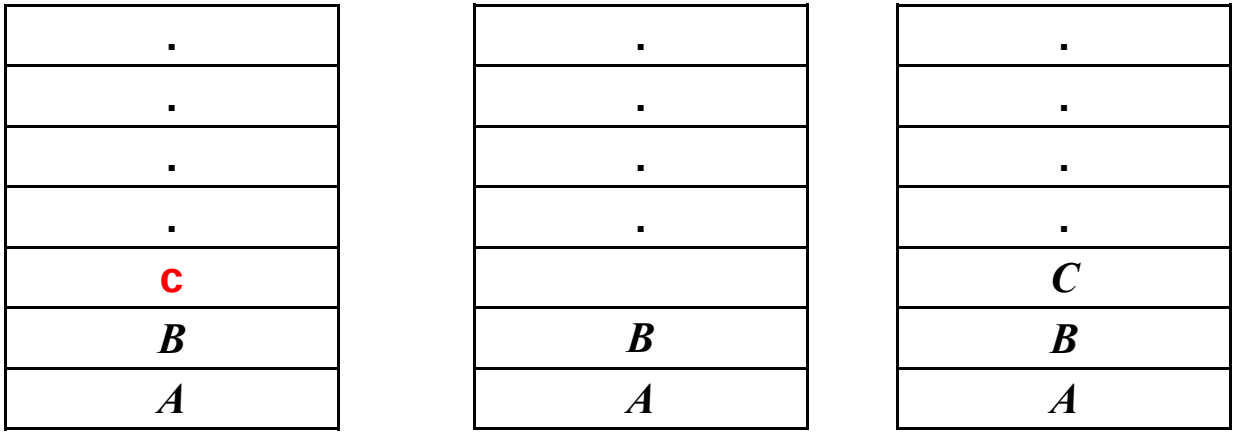
LR Parsing

$X \rightarrow A\ B\ C$.

$A \rightarrow$ a

$B \rightarrow$ b

$C \rightarrow$ c .



pop c (REDUCE)

push C (GOTO)

entrada: a b c

$C \rightarrow$ c

$B \rightarrow$ b

$A \rightarrow$ a

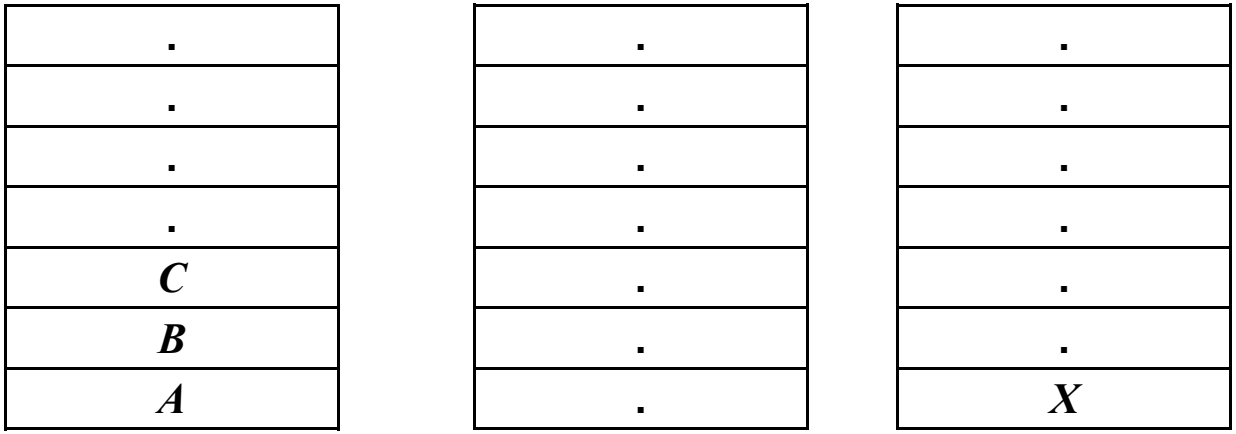
LR Parsing

$X \rightarrow A\ B\ C$.

$A \rightarrow$ **a**

$B \rightarrow$ **b**

$C \rightarrow$ **c**



entrada: a b c

$X \rightarrow A\ B\ C$

$C \rightarrow$ **c**

$B \rightarrow$ **b**

$A \rightarrow$ **a**

LR Parsing

$X \rightarrow A\ B\ C$

$C \rightarrow$ **c**

$B \rightarrow$ **b**

$A \rightarrow$ **a**

X

LR Parsing

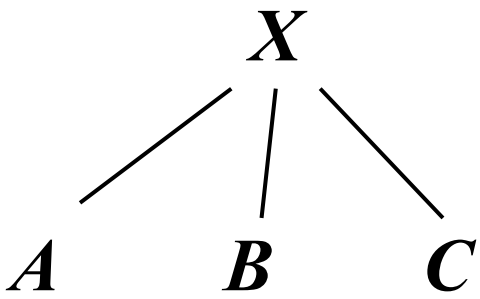
$X \rightarrow A\ B\ C$ 

$C \rightarrow$ **c**

$B \rightarrow$ **b**

$A \rightarrow$ **a**

X



$X \rightarrow A\ B\ C$

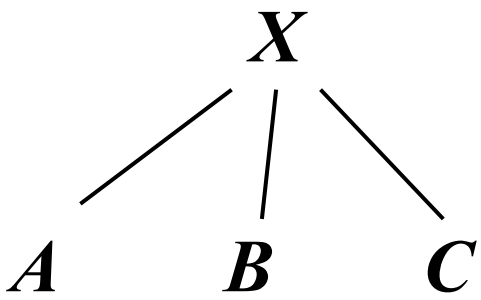
LR Parsing

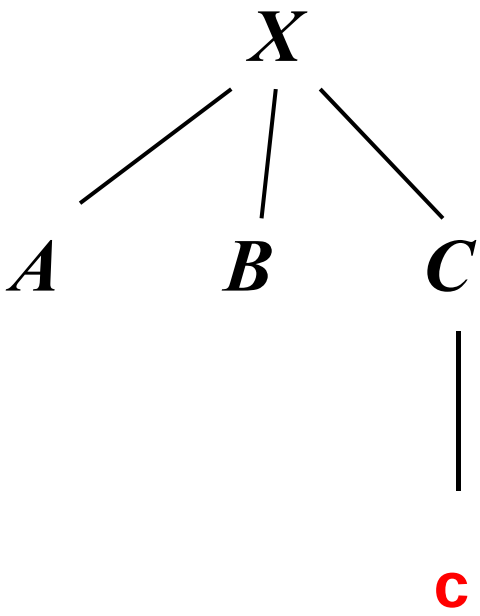
$X \rightarrow A\ B\ C$

$C \rightarrow$ c

$B \rightarrow$ b

$A \rightarrow$ a





$X \rightarrow A\ B\ C \rightarrow A\ B$ c

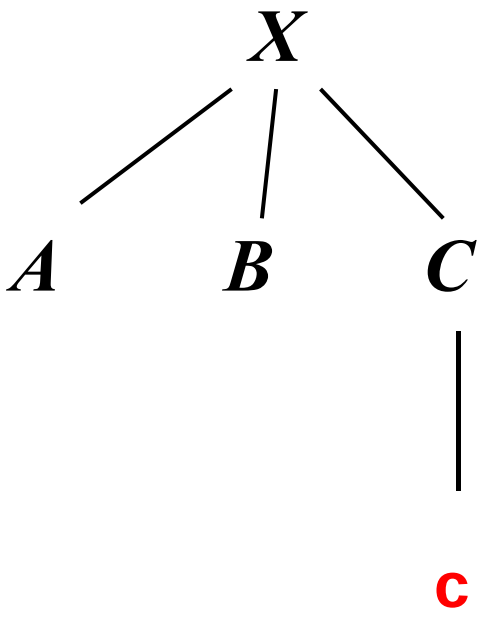
LR Parsing

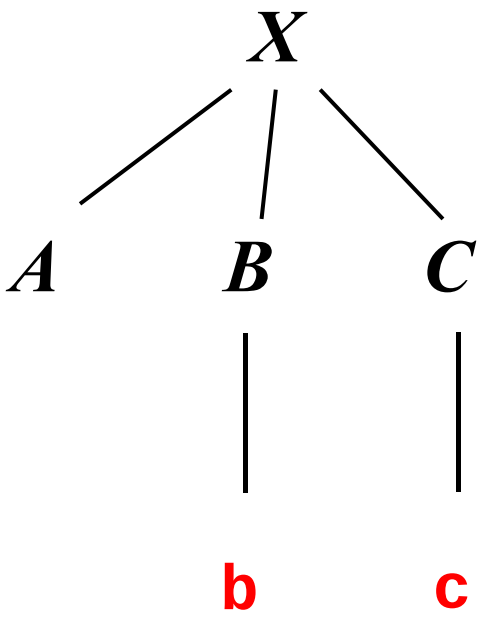
$X \rightarrow A\ B\ C$

$C \rightarrow c$

$B \rightarrow b$ ⟸

$A \rightarrow a$





$X \rightarrow A\ B\ C \rightarrow A\ B$ **c** $\rightarrow A$ **bc**

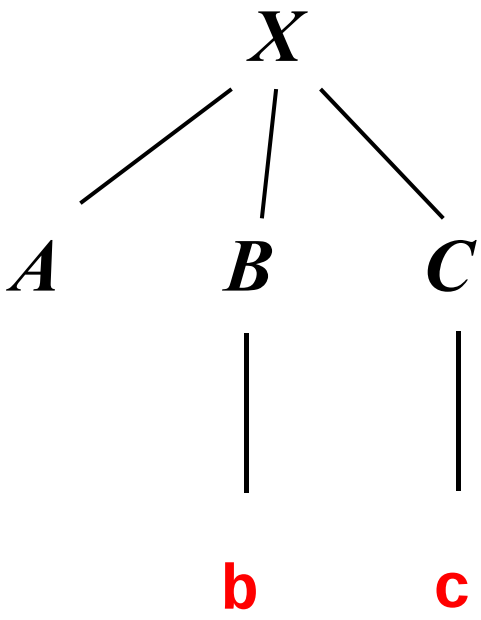
LR Parsing

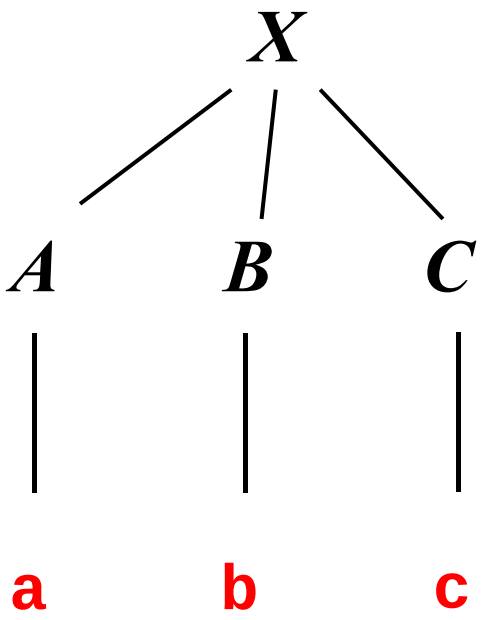
$X \rightarrow A\ B\ C$

$C \rightarrow$ c

$B \rightarrow$ b

$A \rightarrow$ a





$X \rightarrow A\ B\ C \rightarrow A\ B$ c $\rightarrow A$ bc $\rightarrow$ abc

LR Parsing - Exemplo

0. $S' \rightarrow S\$$
1. $S \rightarrow S\ ;\ S$
2. $S \rightarrow \texttt{id} := E$
3. $S \rightarrow \texttt{print}(L)$
4. $E \rightarrow \texttt{id}$
5. $E \rightarrow \texttt{num}$
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) $

LR Parsing - Exemplo

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

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* $E \rightarrow$ num
1 id$_4$:=$_6$ E_{11}	; b := c + (d := 5 + 6 , d) $	*reduce* $S \rightarrow$ 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* $E \rightarrow$ 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* $E \rightarrow$ 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* $E \rightarrow$ 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* $S \rightarrow$ 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* $E \rightarrow$ 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* $S \rightarrow$ id:=E
1 S_2 ;$_3$ S_5	$	*reduce* $S \rightarrow S; S$
1 S_2	$	*accept*

LR Parsing Engine

Como o *parser* sabe quando fazer um shift ou um reduce?

LR Parsing Engine

Como o *parser* sabe quando fazer um shift ou um reduce?

Usando um autômato de pilha!

As arestas são nomeadas com os símbolos que podem aparecer na pilha

4 tipos de ações:

- **s***n:* Shift para o estado *n*;
- **g***n:* Vá para o estado *n*;
- **r***k:* Reduza pela regra *k*;
- **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

LR Parsing Engine

	id	num	print	;	,	+	:=	(	)	$	*S*	*E*	*L*
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				

Geração de Parsers LR(0)

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

- $S' \rightarrow S\$$

1. $S \rightarrow (L)$
2. $S \rightarrow x$
3. $L \rightarrow S$
4. $L \rightarrow L, S$

Estados

•	$S' \rightarrow S\$$
1.	$S \rightarrow (L)$
2.	$S \rightarrow x$
3.	$L \rightarrow S$
4.	$L \rightarrow L, S$

$$S' \rightarrow .S\$$$
$$S \rightarrow .x$$
$$S \rightarrow .(L)$$

1

- *Estado Inicial*

$$S \rightarrow x.$$

2

- *Shift de "x" e "("*
- Estado 2 permite reduce

$$S \rightarrow (.L)$$
$$L \rightarrow .L, S$$
$$L \rightarrow .S$$
$$S \rightarrow .(L)$$
$$S \rightarrow .x$$

3

$$S' \rightarrow S.\$$$

4

Estados

Goto Action:

- Imagine um shift de x ou "$($" 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)

$$S' \rightarrow S.\$ \quad 4$$

Algoritmos

- 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 → α.Xβ in I
    for any production X → γ
       I ← I ∪ {X → .γ}
 until I does not change.
 return I
```

```
Goto(I, X) =
     set J to the empty set
for any item A → α.Xβ in I
         add A → αX.β to J
   return Closure(J)
```

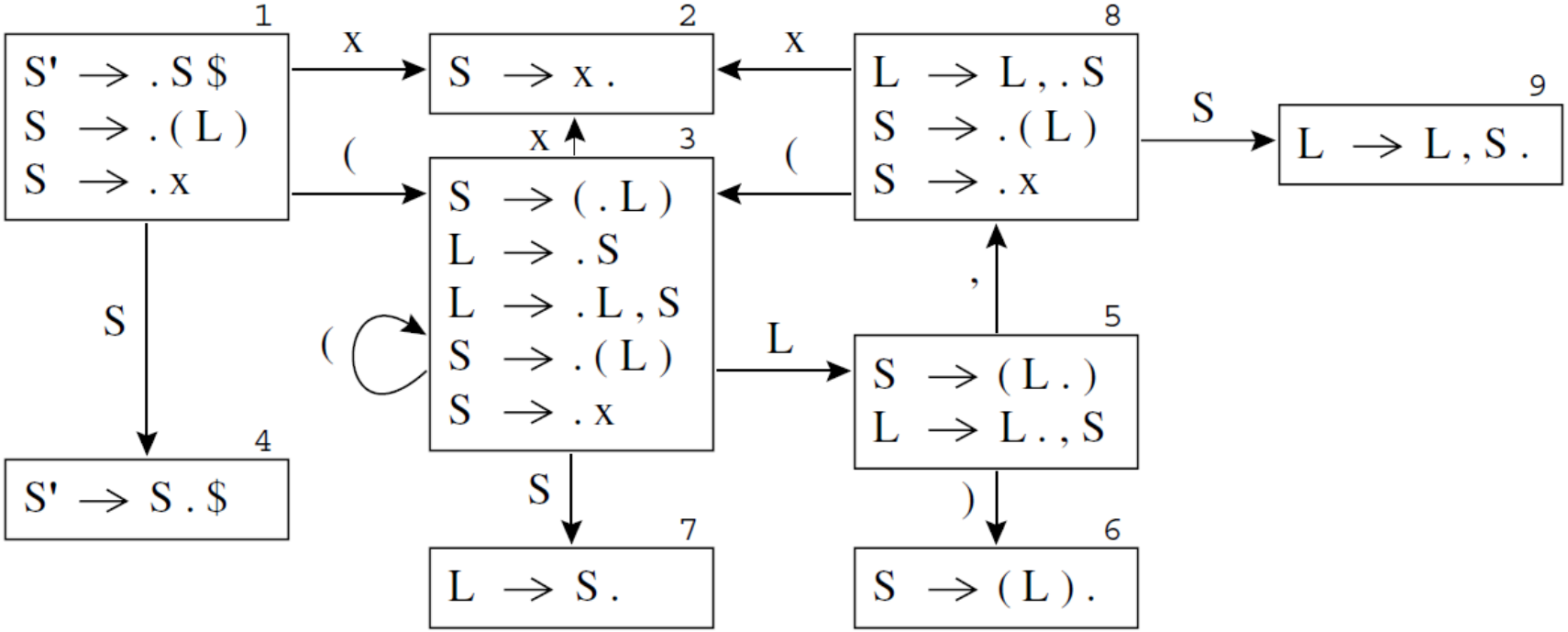
Algoritmos

Construção do parser LR(0)

```
Initialize T to {Closure({S' → .S$})}
Initialize E to empty.
repeat
    for each state I in T
        for each item A → α.Xβ in I
            let J be Goto(I, X)
            T ← T U {J}
            E ← E U {I --X--> J}
until E and T did not change in this iteration
```

```
R ← {}
for each state I in T
    for each item A →α.
        R ← R U{I, A → α}
```

Exemplo

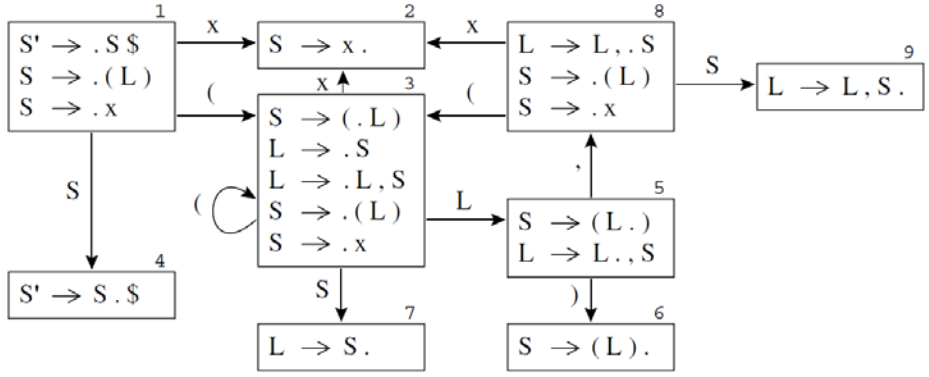


•	$S' \rightarrow S\$$
1.	$S \rightarrow (L)$
2.	$S \rightarrow x$
3.	$L \rightarrow S$
4.	$L \rightarrow L, S$

Exemplo

	(	)	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		

•	*S′ → S$*
1.	*S → (L)*
2.	*S → x*
3.	*L → S*
4.	*L → L, S*



Exemplo

	(	)	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		

•	$S' \rightarrow S\$$
1.	$S \rightarrow (L)$
2.	$S \rightarrow x$
3.	$L \rightarrow S$
4.	$L \rightarrow L, S$

A cadeia abaixo pertence a linguagem gerada pela gramática?

(x)$

Exemplo

•	$S' \rightarrow S\$$
1.	$S \rightarrow (L)$
2.	$S \rightarrow x$
3.	$L \rightarrow S$
4.	$L \rightarrow L, S$

	(	)	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		

Pilha	**Entrada | Ação**
$_1$	(x)$ | shift
$_1(_3$	x)$ | shift
$_1(_3x_2$	)$ | Reduce S-> x
$_1(_3S_7$	)$ | Reduce L-> S
$_1(_3L_5$	)$ | shift
$_1(_3L_5)_6$	$ | Reduce S->(L)
$_1S_4$	$ | accept

Tente construir um *parser* LR(0)

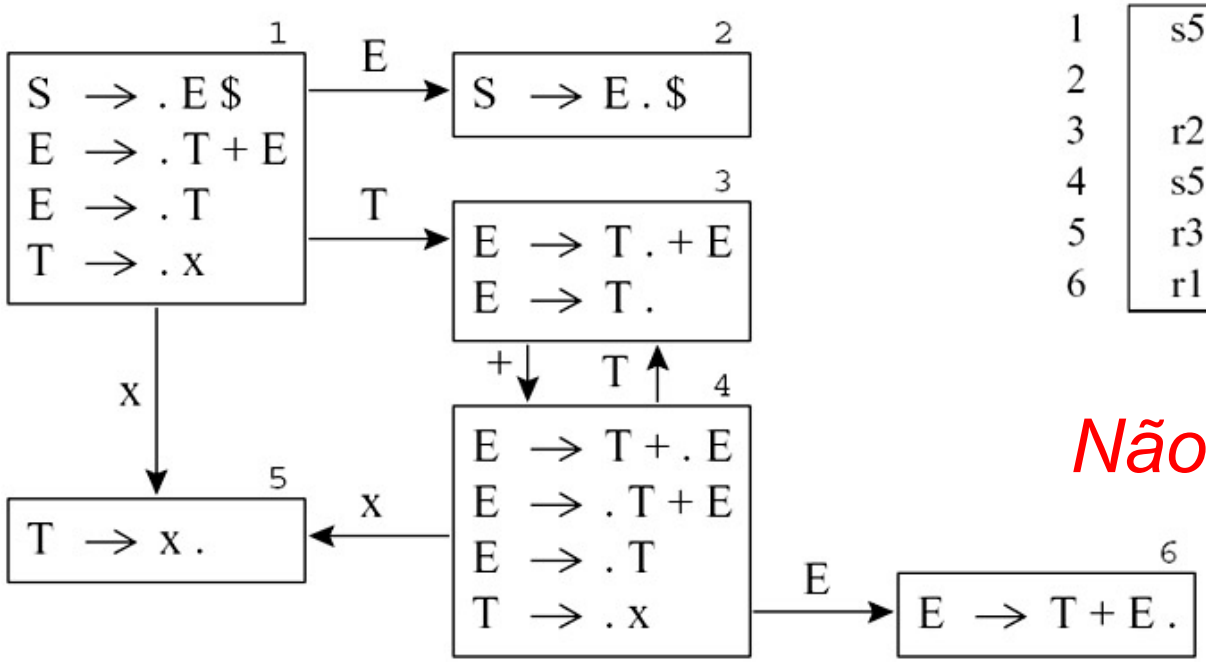
- $S \rightarrow E\ \$$

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

Tente construir um *parser* LR(0)

- $S \rightarrow E\ \$$

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

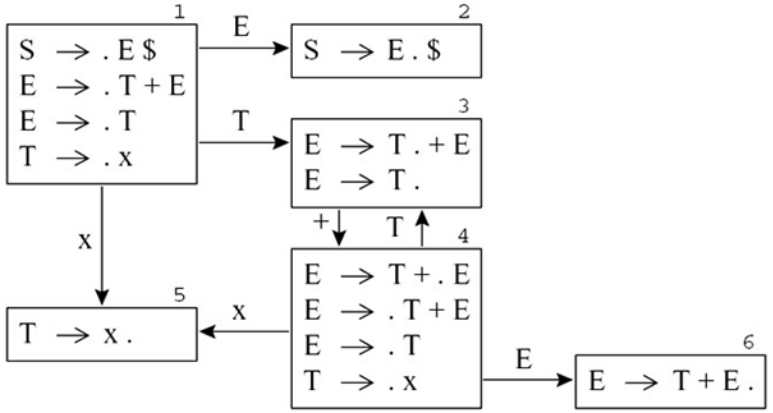


	x	+	$	*E*	*T*
1	s5			g2	g3
2			a		
3	r2	s4,r2	r2		
4	s5			g6	g3
5	r3	r3	r3		
6	r1	r1	r1		

Não é LR(0)!!!

SLR Parser (Simple LR)

Colocar reduções somente onde indicado pelo conjunto FOLLOW



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

Follow(E) = { $ }
Follow(T) = { +, $}

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

É SLR!!!