

Compiladores (CC3001)

Aula 5: Análise sintática *bottom-up*

Mário Florido

DCC/FCUP

2024



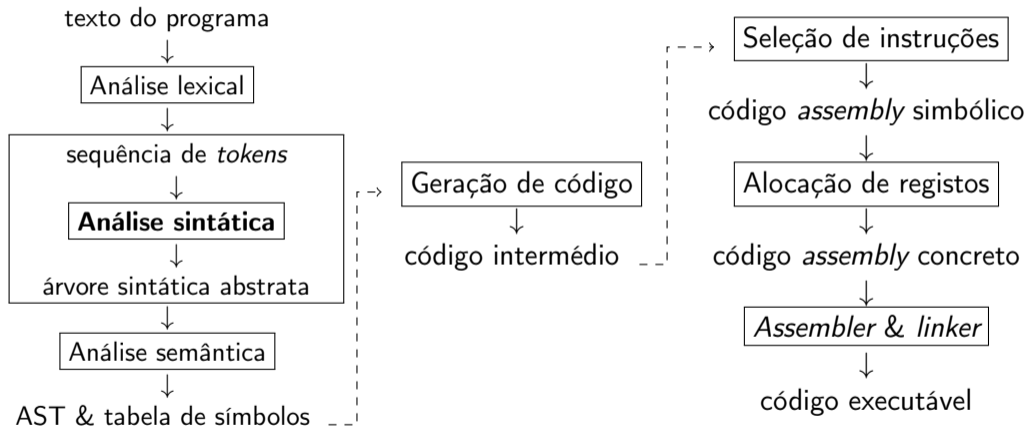
Análise sintática LR

Análise $LR(0)$

Análise $SLR(1)$

Resolução de conflitos

Extras



Análise sintática LR

Análise $LR(0)$

Análise $SLR(1)$

Resolução de conflitos

Extras

- ▶ Vamos hoje ver métodos de análise sintática *LR* (*Left-right parse, Rightmost derivation*)
- ▶ Principais vantagens:
 - ▶ análise *LR* permitem reconhecer mais linguagens
 - ▶ é mais fácil re-escrever uma gramática para análise *LR* do que *LL* (top-down)
 - ▶ análise *LR* permite resolver ambiguidades definindo *prioridades* e *associatividades* de símbolos sem alterar a gramática

Um autómato de pilha:

- ▶ Uma sequência de símbolos terminais (**input**)
- ▶ Uma **pilha** de símbolos (terminais e não-terminais)
- ▶ Inicialmente a pilha está vazia a sequência de *input* está no início
- ▶ Em cada passo escolhe uma de duas ações:
 - Shift* Mover o próximo terminal da entrada para o topo da pilha;
 - Reduce* Escolher uma produção $X \rightarrow \gamma$ tal que os símbolos γ estão no topo da pilha; remover esses símbolos e empilhar X .

Um autómato de pilha:

- ▶ Uma sequência de símbolos terminais (*input*)
- ▶ Uma *pilha* de símbolos (terminais e não-terminais)
- ▶ Inicialmente a pilha está vazia a sequência de *input* está no início
- ▶ Em cada passo escolhe uma de duas ações:
 - Shift* Mover o próximo terminal da entrada para o topo da pilha;
 - Reduce* Escolher uma produção $X \rightarrow \gamma$ tal que os símbolos γ estão no topo da pilha; remover esses símbolos e empilhar X .

Por razão técnicas vamos acrescentar:

- ▶ um novo símbolo inicial S' e um terminal $\$$ (marcador de fim);
- ▶ uma produção $S' \rightarrow S \$$

Uma gramática para a linguagem de parêntesis equilibrados:

$$\begin{aligned} S' &\rightarrow S \$ \\ S &\rightarrow (S) S \quad | \quad \varepsilon \end{aligned}$$

Sequência de passos de um analisador LR:

<i>pilha</i>	<i>input</i>	<i>ação</i>
ε	$()\$$	shift
$($	$)\$$	reduce $S \rightarrow \varepsilon$
$(S$	$)\$$	shift
(S)	$\$$	reduce $S \rightarrow \varepsilon$
$(S)S$	$\$$	reduce $S \rightarrow (S)S$
S	$\$$	accept

Lendo de baixo para cima obtemos a derivação:

$$S \Rightarrow (S)S \Rightarrow (S) \Rightarrow ()$$

Uma gramática para expressões simples:

$$\begin{aligned} E' &\rightarrow E \$ \\ E &\rightarrow E+n \quad | \quad n \end{aligned}$$

Sequência de passos:

<i>pilha</i>	<i>input</i>	<i>ação</i>
ε	$n+n\$$	shift
n	$+n\$$	reduce $E \rightarrow n$
E	$+n\$$	shift
$E+$	$n\$$	shift
$E+n$	$\$$	reduce $E \rightarrow E+n$
E	$\$$	accept

Lendo de baixo para cima obtemos a derivação:

$$E \Rightarrow E+n \Rightarrow n+n$$

Como escolhe o autómato a próxima ação?

- ▶ pela **configuração da pilha** (não apenas o símbolo do topo)
- ▶ e possivelmente pelos **próximos símbolos da entrada** (*look-ahead*)

Como escolhe o autómato a próxima ação?

- ▶ pela **configuração da pilha** (não apenas o símbolo do topo)
- ▶ e possivelmente pelos **próximos símbolos da entrada** (*look-ahead*)

Assim:

- ▶ Vamos usar inteiros para representar os **estados** do autómato (configurações da pilha)
- ▶ O autómato **muda de estado** quando empilhamos ou removemos símbolos da pilha
- ▶ Estas ações são descritas numa **tabela de parsing LR**

- ▶ Cada linha corresponde a um **estado** (número inteiro)
- ▶ Cada coluna corresponde a um **símbolo** (terminal ou não-terminal)
- ▶ Cada entrada contém uma **ação**

shift q mover um terminal para a pilha e transitar para o estado q
reduce k seja $X \rightarrow \alpha_1 \dots \alpha_n$ a produção número k :

1. remover da pilha $\alpha_n, \dots, \alpha_1$ (i.e. os símbolos do lado direito)
2. retroceder para o estado associado à nova pilha e colocar X no topo;
3. procurar na tabela “*go* q ” para a entrada X nesse estado;
4. transitar para o estado q

go q mudar para o estado q (depois de um *reduce*)

accept terminar aceitando a sequência

Exemplo de uma tabela

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

$$(0) \quad T' \rightarrow T \$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \varepsilon$$

$$(4) \quad R \rightarrow bR$$

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

$$(0) \quad T' \rightarrow T \$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \varepsilon$$

$$(4) \quad R \rightarrow bR$$

Algumas transições:

- ▶ no estado 0, com próximo símbolo *a*, faz *shift* e muda para o estado 3;

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

$$(0) \quad T' \rightarrow T \$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \varepsilon$$

$$(4) \quad R \rightarrow bR$$

Algumas transições:

- ▶ no estado 0, com próximo símbolo *a*, faz *shift* e muda para o estado 3;
- ▶ no estado 3, com próximo símbolo *c*, faz *reduce* pela regra 3 ($R \rightarrow \varepsilon$);

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

$$(0) \quad T' \rightarrow T \$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \varepsilon$$

$$(4) \quad R \rightarrow bR$$

Algumas transições:

- ▶ no estado 0, com próximo símbolo *a*, faz *shift* e muda para o estado 3;
- ▶ no estado 3, com próximo símbolo *c*, faz *reduce* pela regra 3 ($R \rightarrow \varepsilon$);
- ▶ no estado 0, depois de um *reduce* $T \rightarrow \dots$ vai para o estado 1;

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

$$(0) \quad T' \rightarrow T \$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \varepsilon$$

$$(4) \quad R \rightarrow bR$$

Algumas transições:

- ▶ no estado 0, com próximo símbolo *a*, faz *shift* e muda para o estado 3;
- ▶ no estado 3, com próximo símbolo *c*, faz *reduce* pela regra 3 ($R \rightarrow \varepsilon$);
- ▶ no estado 0, depois de um *reduce* $T \rightarrow \dots$ vai para o estado 1;
- ▶ no estado 1, com próximo símbolo $\$$ termina (*accept*).

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	
1				a						
2			r1	r1						
3	s3	s4	r3	r3	g5	g2				
4		s4	r3	r3		g6				
5			s7							
6			r4	r4						
7			r2	r2						

$$(0) \quad T' \rightarrow T\$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \epsilon$$

$$(4) \quad R \rightarrow bR$$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	
2			r1	r1						
3	s3	s4	r3	r3	g5	g2				
4		s4	r3	r3		g6				
5			s7							
6			r4	r4						
7			r2	r2						

$$(0) \quad T' \rightarrow T\$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \epsilon$$

$$(4) \quad R \rightarrow bR$$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	
3	s3	s4	r3	r3	g5	g2				
4		s4	r3	r3		g6				
5			s7							
6			r4	r4						
7			r2	r2						

$$(0) \quad T' \rightarrow T\$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \epsilon$$

$$(4) \quad R \rightarrow bR$$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	
4		s4	r3	r3		g6				
5			s7							
6			r4	r4						
7			r2	r2						

$$(0) \quad T' \rightarrow T\$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \epsilon$$

$$(4) \quad R \rightarrow bR$$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	
5			s7							
6			r4	r4						
7			r2	r2						

$$(0) \quad T' \rightarrow T\$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \epsilon$$

$$(4) \quad R \rightarrow bR$$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	
6			r4	r4						
7			r2	r2						

$$(0) \quad T' \rightarrow T\$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \epsilon$$

$$(4) \quad R \rightarrow bR$$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	reduce $R \rightarrow \epsilon$; go 6
6			r4	r4			6	aabbbR	cc\$	
7			r2	r2						

$$(0) \quad T' \rightarrow T\$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \epsilon$$

$$(4) \quad R \rightarrow bR$$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	reduce $R \rightarrow \epsilon$; go 6
6			r4	r4			6	aabbbR	cc\$	reduce $R \rightarrow bR$; go 6
7			r2	r2			6	aabbR	cc\$	

(0) $T' \rightarrow T\$$

(1) $T \rightarrow R$

(2) $T \rightarrow aTc$

(3) $R \rightarrow \epsilon$

(4) $R \rightarrow bR$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	reduce $R \rightarrow \epsilon$; go 6
6			r4	r4			6	aabbbR	cc\$	reduce $R \rightarrow bR$; go 6
7			r2	r2			6	aabbR	cc\$	reduce $R \rightarrow bR$; go 6
							6	aabR	cc\$	

$$(0) \quad T' \rightarrow T\$$$

$$(1) \quad T \rightarrow R$$

$$(2) \quad T \rightarrow aTc$$

$$(3) \quad R \rightarrow \epsilon$$

$$(4) \quad R \rightarrow bR$$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R
0	s3	s4	r3	r3	g1	g2
1				a		
2			r1	r1		
3	s3	s4	r3	r3	g5	g2
4		s4	r3	r3		g6
5			s7			
6			r4	r4		
7			r2	r2		

- (0) $T' \rightarrow T\$$
- (1) $T \rightarrow R$
- (2) $T \rightarrow aTc$
- (3) $R \rightarrow \varepsilon$
- (4) $R \rightarrow bR$

estado	pilha	input	ação
0	ε	aabbbcc\$	shift 3
3	a	abbbcc\$	shift 3
3	aa	bbbcc\$	shift 4
4	aab	bbcc\$	shift 4
4	aabb	bcc\$	shift 4
4	aabbb	cc\$	reduce $R \rightarrow \varepsilon$; go 6
6	aabbbR	cc\$	reduce $R \rightarrow bR$; go 6
6	aabbR	cc\$	reduce $R \rightarrow bR$; go 6
6	aabR	cc\$	reduce $R \rightarrow bR$; go 2
2	aaR	cc\$	

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	reduce $R \rightarrow \epsilon$; go 6
6			r4	r4			6	aabbbR	cc\$	reduce $R \rightarrow bR$; go 6
7			r2	r2			6	aabbR	cc\$	reduce $R \rightarrow bR$; go 6
							6	aabR	cc\$	reduce $R \rightarrow bR$; go 2
							2	aaR	cc\$	reduce $T \rightarrow R$; go 5
							5	aaT	cc\$	

(0) $T' \rightarrow T\$$

(1) $T \rightarrow R$

(2) $T \rightarrow aTc$

(3) $R \rightarrow \epsilon$

(4) $R \rightarrow bR$

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	reduce $R \rightarrow \epsilon$; go 6
6			r4	r4			6	aabbbR	cc\$	reduce $R \rightarrow bR$; go 6
7			r2	r2			6	aabbR	cc\$	reduce $R \rightarrow bR$; go 6
							6	aabR	cc\$	reduce $R \rightarrow bR$; go 2
(0) $T' \rightarrow T\$$							2	aaR	cc\$	reduce $T \rightarrow R$; go 5
(1) $T \rightarrow R$							5	aaT	cc\$	shift 7
(2) $T \rightarrow aTc$							7	aaTc	c\$	
(3) $R \rightarrow \epsilon$										
(4) $R \rightarrow bR$										

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	reduce $R \rightarrow \epsilon$; go 6
6			r4	r4			6	aabbbR	cc\$	reduce $R \rightarrow bR$; go 6
7			r2	r2			6	aabbR	cc\$	reduce $R \rightarrow bR$; go 6
(0)	$T' \rightarrow T\$$						6	aabR	cc\$	reduce $R \rightarrow bR$; go 2
(1)	$T \rightarrow R$						2	aaR	cc\$	reduce $T \rightarrow R$; go 5
(2)	$T \rightarrow aTc$						5	aaT	cc\$	shift 7
(3)	$R \rightarrow \epsilon$						7	aaTc	c\$	reduce $T \rightarrow aTc$; go 5
(4)	$R \rightarrow bR$						5	aT	c\$	

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	reduce $R \rightarrow \epsilon$; go 6
6			r4	r4			6	aabbbR	cc\$	reduce $R \rightarrow bR$; go 6
7			r2	r2			6	aabbR	cc\$	reduce $R \rightarrow bR$; go 6
							6	aabR	cc\$	reduce $R \rightarrow bR$; go 2
(0) $T' \rightarrow T\$$							2	aaR	cc\$	reduce $T \rightarrow R$; go 5
(1) $T \rightarrow R$							5	aaT	cc\$	shift 7
(2) $T \rightarrow aTc$							7	aaTc	c\$	reduce $T \rightarrow aTc$; go 5
(3) $R \rightarrow \epsilon$							5	aT	c\$	shift 7
(4) $R \rightarrow bR$							7	aTc	\$	

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	reduce $R \rightarrow \epsilon$; go 6
6			r4	r4			6	aabbbR	cc\$	reduce $R \rightarrow bR$; go 6
7			r2	r2			6	aabbR	cc\$	reduce $R \rightarrow bR$; go 6
							6	aabR	cc\$	reduce $R \rightarrow bR$; go 2
(0)	$T' \rightarrow T\$$						2	aaR	cc\$	reduce $T \rightarrow R$; go 5
(1)	$T \rightarrow R$						5	aaT	cc\$	shift 7
(2)	$T \rightarrow aTc$						7	aaTc	c\$	reduce $T \rightarrow aTc$; go 5
(3)	$R \rightarrow \epsilon$						5	aT	c\$	shift 7
(4)	$R \rightarrow bR$						7	aTc	\$	reduce $T \rightarrow aTc$; go 1
							1	T	\$	

Exemplo de *parsing* seguindo a tabela

	a	b	c	\$	T	R	estado	pilha	input	ação
0	s3	s4	r3	r3	g1	g2	0	ϵ	aabbbcc\$	shift 3
1				a			3	a	abbbcc\$	shift 3
2			r1	r1			3	aa	bbbcc\$	shift 4
3	s3	s4	r3	r3	g5	g2	4	aab	bbcc\$	shift 4
4		s4	r3	r3		g6	4	aabb	bcc\$	shift 4
5			s7				4	aabbb	cc\$	reduce $R \rightarrow \epsilon$; go 6
6			r4	r4			6	aabbbR	cc\$	reduce $R \rightarrow bR$; go 6
7			r2	r2			6	aabbR	cc\$	reduce $R \rightarrow bR$; go 6
(0)	$T' \rightarrow T\$$						6	aabR	cc\$	reduce $R \rightarrow bR$; go 2
(1)	$T \rightarrow R$						2	aaR	cc\$	reduce $T \rightarrow R$; go 5
(2)	$T \rightarrow aTc$						5	aaT	cc\$	shift 7
(3)	$R \rightarrow \epsilon$						7	aaTc	c\$	reduce $T \rightarrow aTc$; go 5
(4)	$R \rightarrow bR$						5	aT	c\$	shift 7
							7	aTc	\$	reduce $T \rightarrow aTc$; go 1
							1	T	\$	accept

```
stack= empty; push(0, stack); next=getToken()
loop
  case table[top(stack),next] of
    shift s: push(s, stack); next=getToken()

    reduce p: let X = left-hand-side of production p
               n = length(right-hand-side of production p)
               pop n elements of stack;
               lookup table[top(stack),X] and find "go s";
               push(s,stack)

    accept:  terminate with sucess

    empty:   report error
```

- ▶ Em cada passo o algoritmo usa a tabela para decidir a ação a tomar
- ▶ É suficiente guardar apenas os **estados** na pilha
 - ▶ cada estado representa uma configuração particular de símbolos na pilha
 - ▶ não é necessário guardar também os símbolos (mas ajudam a perceber a derivação)
- ▶ A parte crucial é a **construção da tabela** de *parsing*
- ▶ Vamos construir a tabela a partir da gramática (independente do *input*)

- ▶ Para implementar um *parser* LR é necessário construir um tabela de *parsing*
- ▶ Podemos decidir as ações usando a pilha e os próximos símbolos terminais (*look-ahead*):
 - $LR(0)$ apenas a pilha (0 símbolos de *look-ahead*)
 - $LR(1)$ 1 símbolo de *look-ahead*
 - $LR(k)$ k símbolos de *look-ahead* (mais geral)
- ▶ A construção da tabela $LR(0)$ é mais simples mas não permite reconhecer muitas linguagens
- ▶ $LR(k)$ com $k \geq 2$ requer tabelas muito grandes
- ▶ $LR(1)$ é suficiente para a maior parte das linguagens de programação

Análise sintática LR

Análise $LR(0)$

Análise $SLR(1)$

Resolução de conflitos

Extras

- ▶ Os *items* são produções com uma posição marcada no lado-direito (com um sinal de ponto final)
- ▶ Os *estados* do autómato vão ser conjuntos destes items

Exemplo: a gramática dos parêntesis (à esquerda) tem 3 produções a que correspondem 9 items (à direita).

$$\begin{aligned} S' &\rightarrow S \$ \\ S &\rightarrow (S) S \\ S &\rightarrow \varepsilon \end{aligned}$$

$$\begin{aligned} S' &\rightarrow .S \$ \\ S' &\rightarrow S. \$ \\ S' &\rightarrow S \$. \\ S &\rightarrow .(S) S \\ S &\rightarrow (.S) S \\ S &\rightarrow (S.) S \\ S &\rightarrow (S). S \\ S &\rightarrow (S) S . \\ S &\rightarrow . \end{aligned}$$

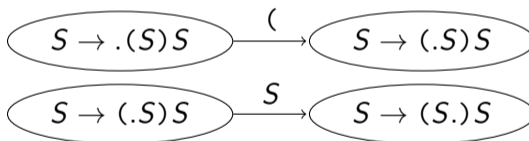
- ▶ Items representam um estado intermédio no reconhecimento de uma produção
- ▶ Exemplo: $S \rightarrow (S).S$
 - ▶ no topo da pilha está a sequência (S)
 - ▶ autómato já reconheceu (S) e poderá continuar com S
- ▶ Mais geralmente

$$A \rightarrow \beta.\gamma$$

significa que β está no topo da pilha e o autómato poderá continuar com γ

- ▶ $A \rightarrow .\gamma$ é um **item inicial**: podemos começar com γ
- ▶ $A \rightarrow \gamma.$ é um **item completo**: γ está no topo da pilha e podemos reconhecer A (*reduce*)

- ▶ Os estados do autómato LR vão ser **items**
- ▶ Para cada item, vamos considerar as **transições** por símbolos terminais e não-terminais
- ▶ Exemplos:



- ▶ Uma transição por um terminal ocorre depois de um **shift**
- ▶ Uma transição por não-terminal ocorre depois de um **reduce**

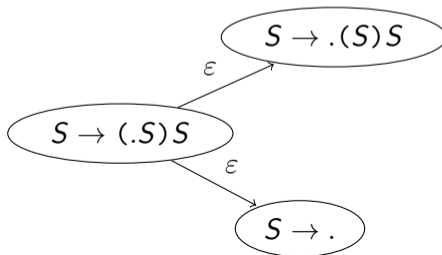
- Sempre que temos um item com um próximo símbolo não-terminal B

$$A \rightarrow \alpha.B\gamma$$

acrescentamos transições- ϵ para *todos* os itens iniciais de B

$$B \rightarrow .\beta$$

- Exemplo:

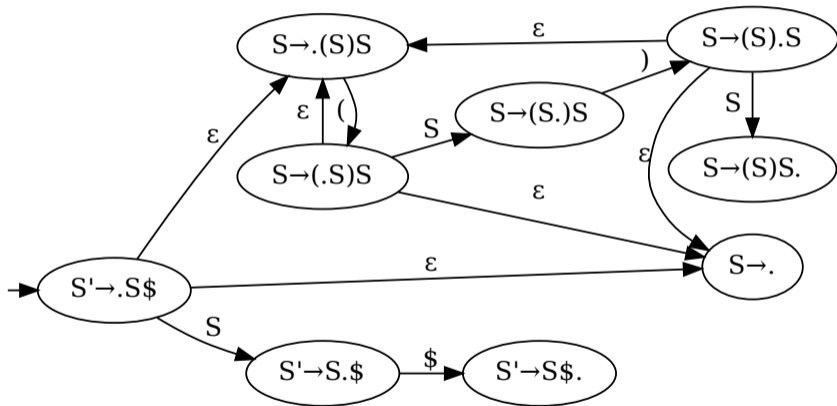


- ▶ O estado inicial do automáto LR é o **item inicial**

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

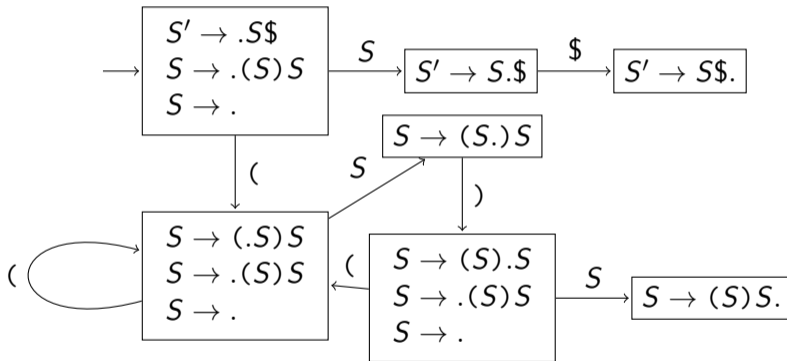
(em que S' é o não-terminal que acrescentamos à gramática)

- ▶ O autómato **não tem estados finais**
- ▶ A aceitação ocorre quando fazemos *shift* do terminador \$
(será uma das ações na tabela de *parsing*)



- ▶ Por causa das transições- ϵ o autómato obtido é não-determinístico (NFA)
- ▶ Vamos usar a construção dos sub-conjuntos para transformar num DFA
- ▶ Os estados do DFA são **conjuntos de items**

Exemplo: DFA para linguagem de parêntesis



		terminais	não-terminais
estados	0
	1
	\vdots		
	n

- ▶ Numeramos os estados do DFA
- ▶ Cada transição com um terminal: colocar uma ação **shift**
- ▶ Cada transição por um não-terminal: colocar uma ação **go**
- ▶ Em cada estado com um **item completo** ($A \rightarrow \gamma.$): colocar uma ação **reduce**
- ▶ No estado $\{S' \rightarrow S.\}$ e próximo símbolo $\$$ colocar a ação **accept**

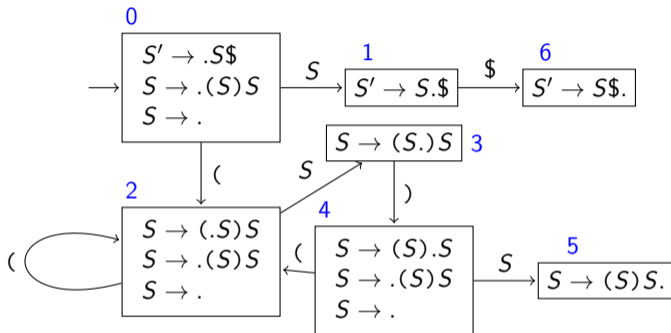
A gramática será $LR(0)$ se **cada entrada tiver no máximo uma ação**.

Tabela $LR(0)$ para a gramática de parêntesis

	()	\$	S
0				
1				
2				
3				
4				
5				

Tabela $LR(0)$ para a gramática de parêntesis

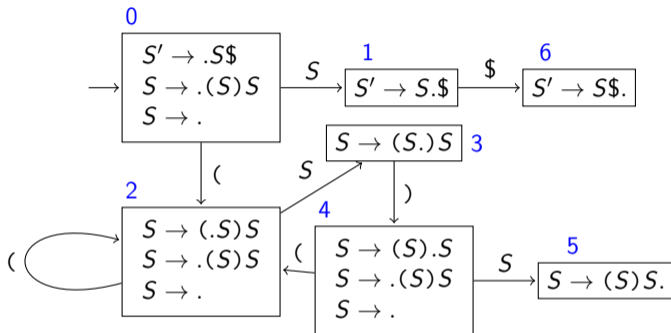
	()	\$	S
0	s2			
1				
2	s2			
3		s4		
4	s2			
5				



► Transições por terminais (shift)

Tabela $LR(0)$ para a gramática de parêntesis

	()	\$	S
0	s2			g1
1				
2	s2			g3
3			s4	
4	s2			g5
5				

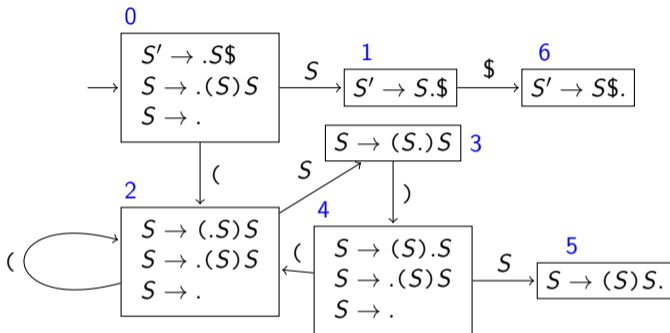


- Transições por terminais (shift)
- Transições por não-terminais (goto)

Tabela $LR(0)$ para a gramática de parêntesis

	()	\$	S
0	s2,r2	r2	r2	g1
1			a	
2	s2,r2	r2	r2	g3
3		s4		
4	s2,r2	r2	r2	g5
5	r1	r1	r1	

0 $S' \rightarrow S\$$
 1 $S \rightarrow (S)S$
 2 $S \rightarrow \epsilon$

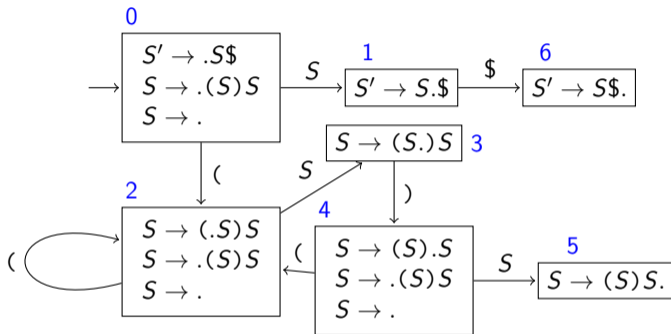


- ▶ Transições por terminais (shift)
- ▶ Transições por não-terminais (goto)
- ▶ Itens completos (reduce) e shift por \$ (accept)

Tabela $LR(0)$ para a gramática de parêntesis

	()	\$	S
0	s2,r2	r2	r2	g1
1			a	
2	s2,r2	r2	r2	g3
3		s4		
4	s2,r2	r2	r2	g5
5	r1	r1	r1	

0	$S' \rightarrow S\$$
1	$S \rightarrow (S)S$
2	$S \rightarrow \epsilon$



- ▶ Transições por terminais (shift)
- ▶ Transições por não-terminais (goto)
- ▶ Items completos (reduce) e shift por \$ (accept)
- ▶ Existem **conflitos shift/reduce**: a gramática não é $LR(0)$

Mostre que a gramática de parêntesis simples

$$A \rightarrow (A) \mid a$$

é $LR(0)$ construindo o autómato e tabela.

Análise sintática LR

Análise $LR(0)$

Análise $SLR(1)$

Resolução de conflitos

Extras

- ▶ O autómato $LR(0)$ usa apenas a informação na pilha para escolher as ações *reduce*
- ▶ Isto frequentemente gera conflitos na tabela — muitas gramáticas úteis não são $LR(0)$
- ▶ Podemos reconhecer muito mais linguagens se usarmos também o próximo símbolo terminal (*look-ahead*)
- ▶ Vamos ver uma extensão simples de $LR(0)$: análise $SLR(1)$
(*Simple Left-right parse, Rightmost derivation 1 symbol look-ahead*)

- ▶ O algoritmo de *parsing* mantém-se
- ▶ Construimos o autómato como no caso $LR(0)$
- ▶ Construção da tabela de *parsing*:
 - ▶ colocamos ações de *shift* e *goto* como anteriormente;
 - ▶ colocamos cada ações *reduce* para estados $A \rightarrow \gamma$. apenas nas colunas do símbolos terminais em $FOLLOW(A)$

Construir a tabela para a gramática de parêntesis:

	()	\$	S
0	s2			g1
1			a	
2	s2			g3
3		s4		
4	s2			g5
5				

- ▶ As transições por terminais (shift) e não-terminais (goto) são como anteriormente

Construir a tabela para a gramática de parêntesis:

	()	\$	S
0	s2	r2	r2	g1
1			a	
2	s2	r2	r2	g3
3		s4		
4	s2	r2	r2	g5
5		r1	r1	

$$0 \quad S' \rightarrow S \$$$

$$1 \quad S \rightarrow (S) S$$

$$2 \quad S \rightarrow \epsilon$$

$$\text{FOLLOW}(S) = \{), \$\}$$

- ▶ As transições por terminais (shift) e não-terminais (goto) são como anteriormente
- ▶ Items completos (reduce) usam *look-ahead*
- ▶ Já não existem conflitos *shift* e *reduce* — a gramática é *SLR(1)*.

Construindo o autómato e a tabela, mostre que a gramática do exemplo inicial

$$T \rightarrow R$$

$$T \rightarrow aTc$$

$$R \rightarrow \varepsilon$$

$$R \rightarrow bR$$

é $SLR(1)$.

(Vai obter uma tabela diferente do exemplo mas mesmo assim sem conflitos.)

Análise sintática LR

Análise $LR(0)$

Análise $SLR(1)$

Resolução de conflitos

Extras

- ▶ Quando obtemos duas ações na mesma entrada da tabela de *parsing* diz-se que temos um **conflito**
- ▶ O conflito significa que o autómato LR tem mais do que uma ação possível:
 - conflito shift/reduce**: uma ação de *shift* e outra(s) de *reduce* no mesmo estado
 - conflito reduce/reduce**: duas (ou mais) ações *reduce* no mesmo estado
- ▶ Os conflitos pode resultar de ambiguidade na gramática¹
- ▶ Os **geradores de parsers LR** (e.g. Bison ou Happy) avisam-nos todos os conflitos na gramática (próxima aula)
- ▶ Podemos eliminar conflitos:
 - ▶ re-escrevendo a gramática (se for ambígua)
 - ▶ definindo associatividades e precedências para *tokens*
 - ▶ escolhendo *shift* em vez de *reduce* (por omissão)

¹Mas nem sempre: a gramática dos parêntesis não é ambígua e tinha um conflito *shift/reduce* na tabela *LR(0)*.

Uma gramática para comandos com “else” opcional.

$$S \rightarrow \text{if cond then } S \text{ else } S$$
$$S \rightarrow \text{if cond then } S$$
$$S \rightarrow \text{skip}$$

Esta gramática é ambígua: a frase

if cond then if cond then skip else skip

admite duas árvores de derivação

if cond then {if cond then skip else skip} (1)

if cond then {if cond then skip} else skip (2)

Ao construir autómato $SLR(1)$ obtemos um estado

$$\begin{aligned} S &\rightarrow \text{if cond then } S . \\ S &\rightarrow \text{if cond then } S . \text{else } S \end{aligned}$$

Quando o próximo *token* é `else` podemos:

- ▶ fazer *shift* pela 2ª produção; ou
- ▶ fazer *reduce* pela 1ª produção (porque $\text{FOLLOW}(S) = \{\text{else}, \$\}$)

Logo: vamos ter um **conflito shift/reduce**.

Se optarmos por **shift** e obtemos a árvore correspondente à interpretação usual

if cond then {if cond then skip else skip}

em linguagens de programação (Pascal, C, Java, etc.)

Análise sintática LR

Análise $LR(0)$

Análise $SLR(1)$

Resolução de conflitos

Extras

- ▶ Apesar da análise $SLR(1)$ reconhecer mais do que $LR(0)$ não é suficiente para algumas construções das linguagens de programação
- ▶ A análise $LR(1)$ é uma generalização de $SLR(1)$ que resolve essas limitações
- ▶ Contudo: o método $LR(1)$ pode produzir automáto muito mais estados que os $SLR(1)$
- ▶ Na prática: os geradores de analisadores LR usam uma variante mais “compacta” designada $LALR(1)$ — *Look-ahead* $LR(1)$
- ▶ As diferenças entre $SLR(1)$, $LR(1)$ e $LALR(1)$ são bastante técnicas
- ▶ Mas **conseguem compreender e usar os geradores de analisadores se compreenderem análise $SLR(1)$**

- ▶ Os estados do autómato $LR(0)$ representam apenas posições no lado direito das produções
- ▶ Generalização $LR(1)$: usar também os símbolos de *look-ahead* para a definição dos próprios estados do autómato
- ▶ Isto vai permitir usar o símbolo de *look-ahead* para distinguir estados e assim evitar conflitos

- ▶ Os *items* são pares de
 - ▶ produções com uma posição no lado-direito (i.e. um item $LR(0)$);
 - ▶ e um símbolo terminal (*look-ahead*)

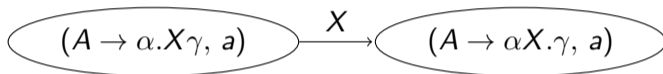
$$\left(\underbrace{A \rightarrow \alpha.\beta}_{\text{item } LR(0)}, \underbrace{a}_{\text{look-ahead}} \right)$$

- ▶ Tal como antes: os estados do autómato são conjuntos destes items
- ▶ Quando o autómato está num estado com um item

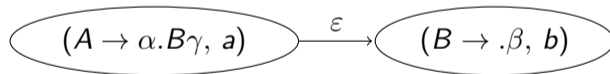
$$(A \rightarrow \alpha.\beta, a)$$

a sequência α está no topo da pilha e o resto da entrada é derivável a partir de βa

Transições por símbolos (terminais e não-terminais):



Transições- ϵ :



Quando B é um não-terminal e para **todas** as produções $B \rightarrow \beta$ e $b \in \text{FIRST}(\gamma a)$.

- ▶ Acrescentamos ações *shift* e *go* como no caso $LR(0)$ e $SLR(1)$
- ▶ Acrescentamos ações *reduce* $A \rightarrow \alpha$ quando um estado contém um item completo

$$(A \rightarrow \alpha., a)$$

e o *próximo terminal* é a .