Construção de Compiladores 1 - 2018.1 - Profs. Mário César San Felice (e Helena Caseli, Murilo Naldi, Daniel Lucrédio)
Tópico 06 - Análise Sintática Ascendente LR - Lista de Exercícios Resolvida

1. Dada a gramática

```
S → 'if' E 'then' C | C
E → a
C → b
```

a) Construa a tabela SLR

R:

```
Coleção canônica de itens LR(0):
   C = {
      I0 = {[S' → . S], [S → . 'if' E 'then' C], [S → . C], [C → . b]},
      I1 = {[S' → S .]},
      I2 = {[S → 'if' . E 'then' C], [E → . a]},
      I3 = {[S → C .]},
      I4 = {[C → b .]},
      I5 = {[S → 'if' E . 'then' C]},
      I6 = {[E → a .]},
      I7 = {[S → 'if' E 'then' . C], [C → . b]},
      I8 = {[S → 'if' E 'then' C .]}
   }
```

Função GOTO:

	'if'	'then'	a	b	S	E	C
I0	I2			I4	I1		I3
I1							
I2			I6			I5	
I3							
I4							
I5		I7					
I6							
I7				I4			I8
I8							

Conjuntos primeiros e seguidores:

```
   primeiros(S') = {'if',b}
   primeiros(S) = {'if',b}
   primeiros(E) = {a}
   primeiros(C) = {b}

   seguidores(S') = {$}
   seguidores(S) = {$}
   seguidores(E) = {'then'}
   seguidores(C) = {$}
```

Gramática enumerada:

S′ → S
(1) S → 'if' E 'then' C
(2) S → C
(3) E → a
(4) C → b

	Ação	Ação	Ação	Ação	Ação	Trans.	Trans.	Trans.
Estados	**'if'**	**'then'**	**a**	**b**	**$**	**S**	**E**	**C**
0	s2			s4		1		3
1					OK			
2			s6				5	
3					r2			
4					r4			
5		s7						
6		r3						
7				s4				8
8					r1			

b) Faça a análise sintática SLR para a cadeia **`if a then b`**, preenchendo os valores da pilha, símbolos, cadeia e ação a cada passo

R:

Pilha	Símbolos	Entrada	Ação
0		if a then b $	s2
0 2	if	a then b $	s6
0 2 6	if a	then b $	r3
0 2 5	if E	then b $	s7
0 2 5 7	if E then	b $	s4
0 2 5 7 4	if E then b	$	r4
0 2 5 7 8	if E then C	$	r1
0 1	S	$	OK

2. Ordene os três tipos de análise sintática ascendente LR do tipo mais simples e menos poderoso para o mais complexo e mais poderoso, descrevendo brevemente as características de cada um

R:

1. SLR(1): análise sintática LR (Left-to-right, Rightmost derivation) simples, que realiza um processo de inferência recursiva para obter derivações mais à direita, lendo a entrada da esquerda para a direita e olhando um caractere à frente durante a análise. Utiliza uma coleção canônica de itens LR(0), ou seja, os itens não consideram um caractere à frente. Por esse motivo, a tabela pode conter um alto número de conflitos empilha-reduz e reduz-reduz, já que alguns itens vão se sobrepor em seus possíveis movimentos.
2. LALR(1): similar à SLR(1), porém utiliza uma coleção de itens LR(1), ou seja, os itens consideram um caractere à frente. Sua construção é mais parecida com a técnica LR(1) canônica (descrita a seguir), porém a tabela é otimizada de forma a não ser necessário a construção de todos os itens LR(1) (coleção canônica de itens LR(1)). Assim, ela apresenta menos conflitos que a técnica SLR, ao mesmo tempo que a

tabela criada não é tão grande. Além disso, os analisadores LALR(1) se comportam exatamente da mesma forma que os analisadores LR(1) (descritos a seguir), quando a entrada é correta. Quando a entrada possui erro sintático, o analisador LALR apenas faz alguns movimentos a mais do que o analisador LR(1). Dessa forma, é a abordagem mais indicada, pois: não gera tantos conflitos quanto a SLR(1); não gera uma tabela grande como a LR(1); e seu comportamento é quase perfeito em termos de reconhecimento de cadeias certas e erradas;

3. LR(1) canônica: similar a SLR(1), porém utiliza uma coleção canônica de itens LR(1), que são itens que consideram um caractere à frente. Dessa forma, a tabela de análise não irá prever alguns movimentos impossíveis conforme os próximos caracteres, o que reduz drasticamente a ocorrência de conflitos. O preço pago é o tamanho da tabela, que fica muito grande. No entanto, esse é o analisador LR mais poderoso de todos, capaz de reconhecer mais gramáticas e gerar menos conflitos.

3. Dada a gramática e tabela sintática LR a seguir

(1) E → E + T
(2) E → T
(3) T → T * F
(4) T → F
(5) F → (E)
(6) F → id

Estados	Ações						Transições		
	id	+	*	(	)	$	E	T	F
0	s5			s4			1	2	3
1		s6				OK			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

Faça a análise sintática para as seguintes cadeias, preenchendo os valores da pilha, símbolos, cadeia e ação a cada passo:

a) (id)

Pilha	Símbolos	Entrada	Ação
0		(id)$	s4
0 4	(	id)$	s5
0 4 5	(id	)$	r6
0 4 3	(F	)$	r4
0 4 2	(T	)$	r2
0 4 8	(E	)$	s11
0 4 8 11	(E)	$	r5
0 3	F	$	r4
0 2	T	$	r2

0 1	E	$	OK

b) ((id))

Pilha	Símbolos	Entrada	Ação
0		((id))$	s4
0 4	(	(id))$	s4
0 4 4	((	id))$	s5
0 4 4 5	((id	))$	r6
0 4 4 3	((F	))$	r4
0 4 4 2	((T	))$	r2
0 4 4 8	((E	))$	s11
0 4 4 8 11	((E)	)$	r5
0 4 3	(F	)$	r4
0 4 2	(T	)$	r2
0 4 8	(E	)$	s11
0 4 8 11	(E)	$	r5
0 3	F	$	r4
0 2	T	$	r2
0 1	E	$	OK

c) (id * id (id))

Pilha	Símbolos	Entrada	Ação
0		(id * id (id))$	s4
0 4	(	id * id (id))$	s5
0 4 5	(id	* id (id))$	r6
0 4 3	(F	* id (id))$	r4
0 4 2	(T	* id (id))$	s7
0 4 2 7	(T *	id (id))$	s5
0 4 2 7 5	(T * id	(id))$	erro! Esperando: +, *,), $

d) ()

Pilha	Símbolos	Entrada	Ação
0		()$	s4
0 4	(	)$	erro! Esperando: id, (

e) (id *)id)

Pilha	Símbolos	Entrada	Ação
0		(id *)id)$	s4
0 4	(	id *)id)$	s5
0 4 5	(id	*)id)$	r6
0 4 3	(F	*)id)$	r4
0 4 2	(T	*)id)$	s7
0 4 2 7	(T *	)id)$	erro! Esperando: id, (

f) (id) * id)

Pilha	Símbolos	Entrada	Ação
0		(id) * id)$	s4
0 4	(	id) * id)$	s5
0 4 5	(id	) * id)$	r6
0 4 3	(F	) * id)$	r4
0 4 2	(T	) * id)$	r2
0 4 8	(E	) * id)$	s11
0 4 8 11	(E)	* id)$	r5
0 3	F	* id)$	r4
0 2	T	* id)$	s7
0 2 7	T *	id)$	s5
0 2 7 5	T * id	)$	r6
0 2 7 10	T * F	)$	r3
0 2	T	)$	r2
0 1	E	)$	erro! Esperando: +, $

g) `id + id * id`

Pilha	Símbolos	Entrada	Ação
0		id + id * id$	s5
0 5	id	+ id * id$	r6
0 3	F	+ id * id$	r4
0 2	T	+ id * id$	r2
0 1	E	+ id * id$	s6
0 1 6	E +	id * id$	s5
0 1 6 5	E + id	* id$	r6
0 1 6 3	E + F	* id$	r4
0 1 6 9	E + T	* id$	s7
0 1 6 9 7	E + T *	id$	s5
0 1 6 9 7 5	E + T * id	$	r6
0 1 6 9 7 10	E + T * F	$	r3
0 1 6 9	E + T	$	r1
0 1	E	$	OK

h) `id + id + id`

Pilha	Símbolos	Entrada	Ação
0		id + id + id$	s5
0 5	id	+ id + id$	r6
0 3	F	+ id + id$	r4
0 2	T	+ id + id$	r2
0 1	E	+ id + id$	s6
0 1 6	E +	id + id$	s5
0 1 6 5	E + id	+ id$	r6
0 1 6 3	E + F	+ id$	r4
0 1 6 9	E + T	+ id$	r1
0 1	E	+ id$	s6
0 1 6	E +	id$	s5
0 1 6 5	E + id	$	r6
0 1 6 3	E + F	$	r4
0 1 6 9	E + T	$	r1
0 1	E	$	OK

4. Dada a gramática

E → E+E | E*E | id

a) Construa a tabela SLR

R:

Gramática enumerada:

(1) E → E + E
(2) E → E * E
(3) E → id

	+	*	id	$	E
0			s2		1
1	s3	s4		OK	
2	r3	r3		r3	
3			s2		5
4			s2		6
5	r1/s3	r1/s4		r1	
6	r2/s3	r2/s4		r2	

b) Identifique os conflitos e classifique-os como shift/shift e shift/reduce

R: Temos 4 conflitos do tipo shift/reduce.

Temos dois conflitos shift/reduce relativos ao estado 5:
I5 = {[E → E+E.],[E → E.+E], [E → E.*E]}

No estado 5, acabou de ocorrer a leitura de uma soma (indicado pelo item [E → E+E.]), e o analisador está pronto para fazer a redução E → E+E. Qualquer símbolo que vier, exceto id, pode causar uma redução válida, já que seguidores(E) = {+,*,$}.

No entanto, caso o próximo símbolo seja + ou *, também é possível fazer o empilhamento, conforme indicado pelos itens [E → E.+E] e [E → E.*E]. Nesses dois casos, haverá conflito entre reduzir ou empilhar
De forma análoga, temos dois conflitos shift/reduce relativos ao estado 6:
I6 = {[E → E*E.],[E → E.+E], [E → E.*E]}

A situação do estado 6 é idêntica, com exceção que aqui acabou de ocorrer a leitura de uma multiplicação (indicado pelo item [E → E*E.]). Assim, pode ocorrer redução E → E*E ou um empilhamento, caso os próximos símbolos sejam + ou *.

c) Resolva os conflitos considerando a precedência e associatividade convencionais dos operadores aritméticos

	+	*	id	$	E
0			s2		1
1	s3	s4		OK	
2	r3	r3		r3	

3			s2		5
4			s2		6
5	**r1 (+ é associativo à esquerda)**	**s4 (* tem precedência sobre +)**		r1	
6	**r2 (* tem precedência sobre +)**	**r2 (* é associativo à esquerda)**		r2	

d) Faça a análise sintática da cadeia **`id + id * id * id`** conforme a resolução de conflitos do item anterior

Pilha	Símbolos	Entrada	Ação
0		id + id * id * id$	s2
0 2	id	+ id * id * id$	r3
0 1	E	+ id * id * id$	s3
0 1 3	E +	id * id * id$	s2
0 1 3 2	E + id	* id * id$	r3
0 1 3 5	E + E	* id * id$	s4
0 1 3 5 4	E + E *	id * id$	s2
0 1 3 5 4 2	E + E * id	* id$	r3
0 1 3 5 4 6	E + E * E	* id$	r2
0 1 3 5	E + E	* id$	s4
0 1 3 5 4	E + E *	id$	s2
0 1 3 5 4 2	E + E * id	$	r3
0 1 3 5 4 6	E + E * E	$	r2
0 1 3 5	E + E	$	r1
0 1	E	$	OK