Construção de Compiladores
Daniel Lucrédio, Helena Caseli, Mário César San Felice e Murilo Naldi
Tópico 04 - Análise Sintática Descendente - Lista de Exercícios Resolvida
(Última revisão: fev/2020)

1. Qual o processo usado pela análise sintática descendente ao associar uma gramática a uma entrada?

R: É um processo que encontra derivações, a partir do símbolo inicial da gramática, até encontrar uma cadeia que corresponde à entrada. Caso não consiga, é detectado um erro sintático.

2. Quais as vantagens do analisador preditivo com relação ao analisador com retrocesso? Existe alguma desvantagem?

R: A principal vantagem é a eficiência, pois um analisador preditivo consegue determinar a regra de substituição a ser aplicada em cada passo da derivação. A desvantagem é que a gramática precisa ser "predizível", ou seja, ela precisa ter características especiais que permitem que a predição possa ocorrer. Já um analisador com retrocesso funciona mesmo sem essas características especiais. Ou seja, um analisador preditivo reconhece uma classe menor de gramáticas do que o analisador com retrocesso.

3. Qual o principal desafio do analisador sintático preditivo?

R: Encontrar a regra de substituição correta a ser aplicada, olhando-se apenas um certo número de símbolos terminais à frente.

4. Dada a gramática a seguir:

```
Expr : Expr 'OU' Termo | Termo
Termo : Termo 'E' Fator | Fator
Fator : 'NÃO' Fator | id
```

Ela é LL(1)? Se não, aplique as transformações necessárias para convertê-la para LL(1).

R: Não é LL(1), pois, há recursividade à esquerda. Removendo:

```
Expr: Termo Expr2
Expr2: 'OU' Termo Expr2 | ε
Termo: Fator Termo2
Termo2: 'E' Fator Termo2 | ε
Fator: 'NÃO' Fator | id
```

5. Considere a gramática a seguir:

```
lexp : atomo | lista
atomo : numero | identificador
lista : ( lexpseq )
lexpseq : lexpseq lexp | lexp
```

Ela é LL(1)? Se não, aplique as transformações necessárias para convertê-la para LL(1).

R: Não é LL(1), pois, há recursividade à esquerda. Removendo:

```
lexp : atomo | lista
atomo : numero | identificador
lista : ( lexpseq )
lexpseq : lexp lexpseq2
lexpseq2 : lexp lexpseq2 | ε
```

6. Dada a seguinte tabela LL(1):

	numero	**identificador**	**(**	**)**	**$**
lexp	lexp → atomo	lexp → atomo	lexp → lista		
atomo	atomo → numero	atomo → identificador			
lista			lista → (lexpseq)		
lexpseq	lexpseq → lexp lexpseq2	lexpseq → lexp lexpseq2	lexpseq → lexp lexpseq2		
lexpseq2	lexpseq2 → lexp lexpseq2	lexpseq2 → lexp lexpseq2	lexpseq2 → lexp lexpseq2	lexpseq2 → ε	

a) Mostre as ações do analisador preditivo não recursivo correspondente dada a cadeia de entrada:

(x (y (2)) (z))

OBS.: x, y e z são identificadores e 2, número.

R:

Casamento	**Pilha**	**Entrada**	**Ação**
	lexp $	(x(y(2))(z))$	lexp → lista
	lista $	(x(y(2))(z))$	lista → (lexpseq)
	(lexpseq) $	(x(y(2))(z))$	casamento
(	**lexpseq**) $	x(y(2))(z))$	lexpseq → lexp lexpseq2
	lexp lexpseq2) $	x(y(2))(z))$	lexp → atomo
	atomo lexpseq2) $	x(y(2))(z))$	atomo → identificador
	identificador lexpseq2) $	x(y(2))(z))$	casamento
(**x**	**lexpseq2**) $	(y(2))(z))$	lexpseq2 → lexp lexpseq2
	lexp lexpseq2) $	(y(2))(z))$	lexp → lista
	lista lexpseq2) $	(y(2))(z))$	lista → (lexpseq)
	(lexpseq) lexpseq2) $	(y(2))(z))$	casamento
(x**(**	**lexpseq**) lexpseq2) $	y(2))(z))$	lexpseq → lexp lexpseq2
	lexp lexpseq2) lexpseq2) $	y(2))(z))$	lexp → atomo
	atomo lexpseq2) lexpseq2) $	y(2))(z))$	atomo → identificador
	identificador lexpseq2) lexpseq2) $	y(2))(z))$	casamento
(x(**y**	**lexpseq2**) lexpseq2) $	(2))(z))$	lexpseq2 → lexp

			lexpseq2
	lexp lexpseq2) lexpseq2) $	**(**2))(z))$	lexp → lista
	lista lexpseq2) lexpseq2) $	**(**2))(z))$	lista → (lexpseq)
	(lexpseq) lexpseq2) lexpseq2) $	**(**2))(z))$	casamento
(x(y**(**	**lexpseq**) lexpseq2) lexpseq2) $	**2**))(z))$	lexpseq → lexp lexpseq2
	lexp lexpseq2) lexpseq2) lexpseq2) $	**2**))(z))$	lexp → atomo
	atomo lexpseq2) lexpseq2) lexpseq2) $	**2**))(z))$	atomo → numero
	numero lexpseq2) lexpseq2) lexpseq2) $	**2**))(z))$	casamento
(x(y(**2**	**lexpseq2**) lexpseq2) lexpseq2) $	**)**)(z))$	lexpseq2 → ε
	) lexpseq2) lexpseq2) $	**)**)(z))$	casamento
(x(y(2**)**	**lexpseq2**) lexpseq2) $	**)**(z))$	lexpseq2 → ε
	) lexpseq2) $	**)**(z))$	casamento
(x(y(2)**)**	**lexpseq2**) $	**(**z))$	lexpseq2 → lexp lexpseq2
	lexp lexpseq2) $	**(**z))$	lexp → lista
	lista lexpseq2) $	**(**z))$	lista → (lexpseq)
	(lexpseq) lexpseq2) $	**(**z))$	casamento
(x(y(2))**(**	**lexpseq**) lexpseq2) $	**z**))$	lexpseq → lexp lexpseq2
	lexp lexpseq2) lexpseq2) $	**z**))$	lexp → atomo
	atomo lexpseq2) lexpseq2) $	**z**))$	atomo → identificador
	identificador lexpseq2) lexpseq2) $	**z**))$	casamento
(x(y(2))(**z**	**lexpseq2**) lexpseq2) $	**)**)$	lexpseq2 → ε
	) lexpseq2) $	**)**)$	casamento
(x(y(2))(z**)**	**lexpseq2**) $	**)**$	lexpseq2 → ε
	) $	**)**$	casamento
(x(y(2))(z)**)**	**$**	**$**	fim

b) Repita o item a) para a entrada:

(x y 2)) (z))

R:

Casamento	**Pilha**	**Entrada**	**Ação**
	lexp $	**(**x y 2))(z))$	lexp → lista
	lista $	**(**x y 2))(z))$	lista → (lexpseq)
	(lexpseq) $	**(**x y 2))(z))$	casamento
(	**lexpseq**) $	**x** y 2))(z))$	lexpseq → lexp lexpseq2
	lexp lexpseq2) $	**x** y 2))(z))$	lexp → atomo
	atomo lexpseq2) $	**x** y 2))(z))$	atomo → identificador
	identificador lexpseq2) $	**x** y 2))(z))$	casamento
(**x**	**lexpseq2**) $	**y** 2))(z))$	lexpseq2 → lexp lexpseq2
	lexp lexpseq2) $	**y** 2))(z))$	lexp → atomo
	atomo lexpseq2) $	**y** 2))(z))$	atomo → identificador
	identificador lexpseq2) $	**y** 2))(z))$	casamento
(x **y**	lexpseq2) $	**2**))(z))$	lexpseq → lexp lexpseq2
	lexp lexpseq2) $	**2**))(z))$	lexp → atomo
	atomo lexpseq2) $	**2**))(z))$	atomo → numero

	numero lexpseq2) $	**2**))(z))$	casamento
(x y **2**	**lexpseq2**) $	**)**)(z))$	lexpseq2 → ε
	) $	**)**)(z))$	casamento
(x y 2**)**	**$**	**)**(z))$	erro!

c) Repita o item a) para a entrada:

(x y 2

R:

Casamento	Pilha	Entrada	Ação
	lexp $	**(**x y 2$	lexp → lista
	lista $	**(**x y 2$	lista → (lexpseq)
	(lexpseq) $	**(**x y 2$	casamento
(	**lexpseq**) $	**x** y 2$	lexpseq → lexp lexpseq2
	lexp lexpseq2) $	**x** y 2$	lexp → atomo
	atomo lexpseq2) $	**x** y 2$	atomo → identificador
	identificador lexpseq2) $	**x** y 2$	casamento
(**x**	**lexpseq2**) $	**y** 2$	lexpseq2 → lexp lexpseq2
	lexp lexpseq2) $	**y** 2$	lexp → atomo
	atomo lexpseq2) $	**y** 2$	atomo → identificador
	identificador lexpseq2) $	**y** 2$	casamento
(x **y**	lexpseq2) $	**2**$	lexpseq → lexp lexpseq2
	lexp lexpseq2) $	**2**$	lexp → atomo
	atomo lexpseq2) $	**2**$	atomo → numero
	numero lexpseq2) $	**2**$	casamento
(x y **2**	**lexpseq2**) $	**$**	erro!

7. A gramática a seguir é LL(k).

```
S : id ':' id | id ':' id '{' S '}';
```

Qual o valor de k?

R: k == 4, pois é preciso olhar 4 símbolos à frente para decidir qual das duas produções de S utilizar.

8. A gramática a seguir é LL(k)? Justifique sua resposta

```
declaracao : nomeQualificado ':' ID ';'
           | nomeQualificado ':' ID '=' expressao;
nomeQualificado : ID | ID '.' nomeQualificado;
```

R: Não, pois não há nenhum valor mínimo de k tal que garantidamente seja possível determinar qual das duas produções da regra "declaração" utilizar. Isso porque a regra nomeQualificado tem recursividade, e portanto pode gerar nomes infinitamente grandes.