

**Construção de Compiladores 1 - 2018.1 - Profs. Mário César San Felice
(e Helena Caseli, Murilo Naldi, Daniel Lucrédio)
Tópico 04 - Análise Sintática Descendente - Lista de Exercícios Resolvida**

1. Qual a diferença conceitual entre análise sintática ascendente e descendente?

R: Ambas constroem uma árvore de análise sintática completa (ainda que não de forma explícita). Porém, a análise sintática ascendente utiliza inferência recursiva (em cada passo, substitui-se o corpo pela cabeça das regras), enquanto a descendente utiliza derivação (em cada passo, substitui-se a cabeça pelo corpo das regras).

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. Construa os conjuntos Primeiros e Seguidores para as seguintes gramáticas:

a) $S : bAb$
 $A : CB \mid a$
 $B : Aa$
 $C : c \mid \varepsilon$

R:

$\text{primeiros}(S) = \{b\}$
 $\text{primeiros}(A) = \{a, c\}$
 $\text{primeiros}(B) = \{a, c\}$
 $\text{primeiros}(C) = \{c, \varepsilon\}$
 $\text{primeiros}(bAb) = \{b\}$
 $\text{primeiros}(CB) = \{a, c\}$
 $\text{primeiros}(a) = \{a\}$
 $\text{primeiros}(Aa) = \{a, c\}$
 $\text{primeiros}(c) = \{c\}$

$\text{seguidores}(S) = \{\$ \}$
 $\text{seguidores}(A) = \{a, b\}$
 $\text{seguidores}(B) = \{a, b\}$
 $\text{seguidores}(C) = \{a, c\}$

b) $S : A \mid B \mid \varepsilon$
 $A : A + B \mid A - B \mid 1 \mid 2 \mid 3 \mid \varepsilon$

B : A | C
C : (A)

R:

primeiros(S) = { ϵ , 1, 2, 3, +, -, (
primeiros(A) = { ϵ , 1, 2, 3, +, -}
primeiros(B) = { ϵ , 1, 2, 3, +, -, (
primeiros(C) = { (
primeiros(A+B) = {1, 2, 3, +, -}
primeiros(A-B) = {1, 2, 3, +, -}
primeiros(1) = {1}
primeiros(2) = {2}
primeiros(3) = {3}
primeiros("(" A ")") = { (

seguidores(S) = { $\$$ }
seguidores(A) = { $\$, +, -,)$ }
seguidores(B) = { $\$, +, -,)$ }
seguidores(C) = { $\$, +, -,)$ }

c) S : ϵ | abA | abB | abC
A : aSaa | b
B := bSbb | c
C := cScC | d

R:

primeiros(S) = { ϵ , a}
primeiros(A) = {a, b}
primeiros(B) = {b, c}
primeiros(C) = {c, d}
primeiros(abA) = {a}
primeiros(abB) = {a}
primeiros(abC) = {a}
primeiros(aSaa) = {a}
primeiros(b) = {b}
primeiros(bSbb) = {b}
primeiros(c) = {c}
primeiros(cScC) = {c}
primeiros(d) = {d}

seguidores(S) = { $\$, a, b, c$ }
seguidores(A) = { $\$, a, b, c$ }
seguidores(B) = { $\$, a, b, c$ }
seguidores(C) = { $\$, a, b, c$ }

d) E : TE'
E' : +TE' | ϵ
T : FT'
T' : *FT' | ϵ
F : (E) | id

R:

```
primeiros(E) = {(, id}
primeiros(E') = {+, ε}
primeiros(T) = {(, id}
primeiros(T') = {*, ε}
primeiros(F) = {(, id}
primeiros(TE') = {(, id}
primeiros(+TE') = {+}
primeiros(FT') = {(, id}
primeiros(*FT') = {*}
primeiros("(" E ")") = {(}
primeiros(id) = {id}
```

```
seguidores(E) = {$, )}
seguidores(E') = {$, )}
seguidores(T) = {+, $, )}
seguidores(T') = {+, $, )}
seguidores(F) = {*, +, $, )}
```

- e) declaração : ifdecl | "outra"
ifdecl : "if" "(" exp ")" declaração elseparte
elseparte : "else" declaração | ε
exp : "0" | "1"

R:

```
primeiros(declaração) = {"outra", "if"}
primeiros(ifdecl) = {"if"}
primeiros(elseparte) = {"else", ε}
primeiros(exp) = {"0", "1"}
primeiros("outra") = {"outra"}
primeiros("if" "(" exp ")" declaração elseparte) = {"if"}
primeiros("else" declaração) = {"else"}
primeiros("0") = {"0"}
primeiros("1") = {"1"}
```

```
seguidores(declaração) = {$, "else"}
seguidores(ifdecl) = {$, "else"}
seguidores(elseparte) = {$, "else"}
seguidores(exp) = {}}
```

5. Dada a gramática a seguir:

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

- a) 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

```

b) Construa a tabela sintática LL(1) correspondente à gramática (alterada na letra a) se for o caso).

R:

```

primeiros(Expr) = {'NÃO', id}
primeiros(Expr2) = {'OU', ε}
primeiros(Termo) = {'NÃO', id}
primeiros(Termo2) = {'E', ε}
primeiros(Fator) = {'NÃO', id}
primeiros(Termo Expr2) = {'NÃO', id}
primeiros('OU' Termo Expr2) = {'OU'}
primeiros(ε) = {ε}
primeiros(Fator Termo2) = {'NÃO', id}
primeiros('E' Fator Termo2) = {'E'}
primeiros('NÃO' Fator) = {'NÃO'}
primeiros(id) = {id}

```

```

seguidores(Expr) = {$}
seguidores(Expr2) = {$}
seguidores(Termo) = {'OU', $}
seguidores(Termo2) = {'OU', $}
seguidores(Fator) = {'E', 'OU', $}

```

	'OU'	'E'	'NÃO'	id	\$
Expr			Expr → Termo Expr2	Expr → Termo Expr2	
Expr2	Expr2 → 'OU' Termo Expr2				Expr2 → ε
Termo			Termo → Fator Termo2	Termo → Fator Termo2	
Termo2	Termo2 → ε	Termo2 → 'E' Fator Termo2			Termo2 → ε
Fator			Fator → 'NÃO' Fator	Fator → id	

6. Considere a gramática

```

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

```

a) Remova a recursão à esquerda.

R:

```

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

```

b) Construa os conjuntos Primeiros e Seguidores para os não-terminais da gramática resultante (letra a).

R:

```

primeiros(lexp) = {numero, identificador, (}
primeiros(atomo) = {numero, identificador}
primeiros(lista) = {(}
primeiros(lexpseq) = {numero, identificador, (}
primeiros(lexpseq2) = {numero, identificador, (, ε}

seguidores(lexp) = {$, numero, identificador, (, )}
seguidores(atomo) = {$, numero, identificador, (, )}
seguidores(lista) = {$, numero, identificador, (, )}
seguidores(lexpseq) = {}
seguidores(lexpseq2) = {}

```

c) Construa a tabela de análise sintática a ser usada por um método de ASD preditiva não recursiva, a partir da gramática resultante (letra a).

R:

	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	

d) Mostre as ações do analisador preditivo não recursivo correspondente (de acordo com a tabela criada na letra c) dada a cadeia de entrada (x (y (2)) (z)). OBS.: x, y e z são identificadores e 2, número.

Casamento	Pilha	Entrada	Ação
	<u>lexp</u>	<u>(</u> (x(y(2))(z))\$	lexp → lista
	<u>lista</u>	<u>(</u> (x(y(2))(z))\$	lista → (lexpseq)
	<u>(</u> lexpseq)	<u>(</u> (x(y(2))(z))\$	casamento
<u>(</u>	<u>lexpseq</u>	<u>x</u> (y(2))(z))\$	lexpseq → lexp lexpseq2
	<u>lexp</u> lexpseq2)	<u>x</u> (y(2))(z))\$	lexp → atomo

	<u>atomo</u> lexpseq2) \$	<u>x</u> (y(2))(z))\$	atomo → identificador
	<u>identificador</u> lexpseq2) \$	<u>x</u> (y(2))(z))\$	casamento
(<u>x</u>	<u>lexpseq2</u>) \$	(y(2))(z))\$	lexpseq2 → lexp lexpseq2
	<u>lexp</u> lexpseq2) \$	(y(2))(z))\$	lexp → lista
	<u>lista</u> lexpseq2) \$	(y(2))(z))\$	lista → (lexpseq)
	(lexpseq) lexpseq2) \$	(y(2))(z))\$	casamento
(x(<u>(</u>	<u>lexpseq</u>) lexpseq2) \$	y(2))(z))\$	lexpseq → lexp lexpseq2
	<u>lexp</u> lexpseq2) lexpseq2) \$	y(2))(z))\$	lexp → atomo
	<u>atomo</u> lexpseq2) lexpseq2) \$	y(2))(z))\$	atomo → identificador
	<u>identificador</u> lexpseq2) lexpseq2) \$	y(2))(z))\$	casamento
(x(<u>y</u>	<u>lexpseq2</u>) lexpseq2) \$	(2))(z))\$	lexpseq2 → lexp lexpseq2
	<u>lexp</u> lexpseq2) lexpseq2) \$	(2))(z))\$	lexp → lista
	<u>lista</u> lexpseq2) lexpseq2) \$	(2))(z))\$	lista → (lexpseq)
	(lexpseq) lexpseq2) lexpseq2) \$	(2))(z))\$	casamento
(x(y(<u>(</u>	<u>lexpseq</u>) lexpseq2) lexpseq2) \$	2))(z))\$	lexpseq → lexp lexpseq2
	<u>lexp</u> lexpseq2) lexpseq2) lexpseq2) \$	2))(z))\$	lexp → atomo
	<u>atomo</u> lexpseq2) lexpseq2) lexpseq2) \$	2))(z))\$	atomo → numero
	<u>numero</u> lexpseq2) lexpseq2) lexpseq2) \$	2))(z))\$	casamento
(x(y(2	<u>lexpseq2</u>) lexpseq2) lexpseq2) \$) (z))\$	lexpseq2 → ε
) lexpseq2) lexpseq2) \$) (z))\$	casamento
(x(y(2)	<u>lexpseq2</u>) lexpseq2) \$) (z))\$	lexpseq2 → ε
) lexpseq2) \$) (z))\$	casamento
(x(y(2))	<u>lexpseq2</u>) \$	(z))\$	lexpseq2 → lexp lexpseq2
	<u>lexp</u> lexpseq2) \$	(z))\$	lexp → lista
	<u>lista</u> lexpseq2) \$	(z))\$	lista → (lexpseq)
	(lexpseq) lexpseq2) \$	(z))\$	casamento
(x(y(2))	<u>lexpseq</u>) lexpseq2) \$	z))\$	lexpseq → lexp lexpseq2
	<u>lexp</u> lexpseq2) lexpseq2) \$	z))\$	lexp → atomo
	<u>atomo</u> lexpseq2) lexpseq2) \$	z))\$	atomo → identificador
	<u>identificador</u> lexpseq2) lexpseq2) \$	z))\$	casamento
(x(y(2))	<u>lexpseq2</u>) lexpseq2) \$)\$	lexpseq2 → ε
) lexpseq2) \$)\$	casamento
(x(y(2))(z)	<u>lexpseq2</u>) \$)\$	lexpseq2 → ε
) \$)\$	casamento
(x(y(2))(z)	<u>lexpseq2</u>) \$)\$	lexpseq2 → ε
) \$)\$	casamento
(x(y(2))(z)	<u>lexpseq2</u>) \$)\$	fim

e) Repita o exercício d) para a cadeia de entrada (x y 2) (z)). OBS.: x, y e z são identificadores e 2, número.

Casamento	Pilha	Entrada	Ação
	<u>lexp</u>	\$(x y 2) (z))\$	lexp → lista
	<u>lista</u>	\$(x y 2) (z))\$	lista → (lexpseq)
	(lexpseq)	\$(x y 2) (z))\$	casamento
(<u>lexpseq</u>	\$(x y 2) (z))\$	lexpseq → lexp lexpseq2

	<u>lexp</u> lexpseq2) \$ x y 2)) (z))\$	lexp → atomo
	<u>atomo</u> lexpseq2) \$ x y 2)) (z))\$	atomo → identificador
	<u>identificador</u> lexpseq2) \$ x y 2)) (z))\$	casamento
(<u>x</u>	<u>lexpseq2</u>) \$ y 2)) (z))\$	lexpseq2 → lexp lexpseq2
	<u>lexp</u> lexpseq2) \$ y 2)) (z))\$	lexp → atomo
	<u>atomo</u> lexpseq2) \$ y 2)) (z))\$	atomo → identificador
	<u>identificador</u> lexpseq2) \$ y 2)) (z))\$	casamento
(x <u>y</u>	lexpseq2) \$ 2)) (z))\$	lexpseq2 → lexp lexpseq2
	<u>lexp</u> lexpseq2) \$ 2)) (z))\$	lexp → atomo
	<u>atomo</u> lexpseq2) \$ 2)) (z))\$	atomo → numero
	<u>numero</u> lexpseq2) \$ 2)) (z))\$	casamento
(x y <u>2</u>	<u>lexpseq2</u>) \$) (z))\$	lexpseq2 → ε
	<u>lexpseq2</u>) \$) (z))\$	casamento
(x y 2 <u>l</u>	<u>lexpseq2</u>) \$) (z))\$	erro!

f) Repita o exercício d) para a cadeia de entrada (x y 2. OBS.: x, y e z são identificadores e 2, número.

Casamento	Pilha	Entrada	Ação
	<u>lexp</u>	\$(x y 2\$	lexp → lista
	<u>lista</u>	\$(x y 2\$	lista → (lexpseq)
	(\$(x y 2\$	casamento
(<u>lexpseq</u>)	\$x y 2\$	lexpseq → lexp lexpseq2
	<u>lexp</u> lexpseq2)	\$x y 2\$	lexp → atomo
	<u>atomo</u> lexpseq2)	\$x y 2\$	atomo → identificador
	<u>identificador</u> lexpseq2)	\$x y 2\$	casamento
(<u>x</u>	<u>lexpseq2</u>)	\$y 2\$	lexpseq2 → lexp lexpseq2
	<u>lexp</u> lexpseq2)	\$y 2\$	lexp → atomo
	<u>atomo</u> lexpseq2)	\$y 2\$	atomo → identificador
	<u>identificador</u> lexpseq2)	\$y 2\$	casamento
(x <u>y</u>	lexpseq2)	\$2\$	lexpseq2 → lexp lexpseq2
	<u>lexp</u> lexpseq2)	\$2\$	lexp → atomo
	<u>atomo</u> lexpseq2)	\$2\$	atomo → numero
	<u>numero</u> lexpseq2)	\$2\$	casamento
(x y 2 <u>l</u>	<u>lexpseq2</u>)	\$l\$	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.

9. O que é um DFA de decisão?

R: É um autômato finito determinístico capaz de determinar qual produção utilizar durante a análise sintática descendente preditiva. O DFA de decisão consegue, analisando a cadeia de entrada, olhar um número qualquer de símbolos à frente, até encontrar um símbolo que permita que a decisão seja tomada.

10. Qual a diferença entre DFAs de decisão para gramáticas LL(k) e para gramáticas LL($*$)?

R: Em uma gramática LL(k), um DFA acíclico é suficiente, ou seja, não é necessário passar por um mesmo estado mais de uma vez. Isso porque tem-se a garantia de que existe um número finito k que garante a tomada de decisão. Já em uma gramática LL($*$), é necessário um DFA que pode conter ciclos, já que não existe um número k finito que garante a tomada de decisão.

11. Porque a gramática do exercício 8 não é LL($*$)? Como fazer com que ela seja LL($*$) sem modificar a linguagem?

R: Porque a regra “nomeQualificado” apresenta recursividade, o que faz com que não seja possível implementar um DFA de decisão. Em outras palavras, a linguagem de decisão não é regular (devido à existência de recursividade). Para resolver, é necessário transformar a linguagem de decisão em uma linguagem regular, eliminando a recursividade. Nesse exemplo, é necessário o uso de operadores EBNF (+, *, ?) para indicar a repetição sem precisar de recursividade:

```
declaracao : nomeQualificado ':' ID ';'
           | nomeQualificado ':' ID '=' expressao;
nomeQualificado : ID ('.' ID)*;
```

12. Classifique as seguintes gramáticas como sendo não-LL, LL(1), LL(k) ou LL($*$)

a)

```
S : A | B | ε
A : A + B | A - B | 1 | 2 | 3 | ε
B : A | C
C : (A)
```

R: não-LL (recursividade à esquerda)

b)

```
S : A | B | ε
A : + A B | - A B | 1 | 2 | 3 | ε
B : A | C
C : (A)
```

R: não-LL (ambiguidade)

- c) S : + A B | + A C | + A d
 A : (-|/)*
 B : [S]
 C : (S)

R: LL(*) (DFA com ciclo na regra A)

- d) S : + A + | + A - | + A / | + A *
 A : id | num | (S)

R: não-LL (ambiguidade)