

## 1COP020 - Lista de Exercícios 11

1.  **Exercício Teórico:** Considere a gramática apresentada a seguir:


$rexpr \rightarrow rexpr + rterm$   
 $rexpr \rightarrow rterm$   
 $rterm \rightarrow rterm rfactor$   
 $rterm \rightarrow rfactor$   
 $rfactor \rightarrow rfactor *$   
 $rfactor \rightarrow rprimary$   
 $rprimary \rightarrow \mathbf{a}$   
 $rprimary \rightarrow \mathbf{b}$

- (a) Modifique a gramática criando um novo símbolo inicial e acrescentando o símbolo de fim de arquivo e eliminando recursões à esquerda.  
(b) Mostre se a gramática resultante é ou não LL(1) construindo a tabela de análise sintática.

2.  **Exercício Teórico:** Considere a gramática apresentada a seguir:

$S \rightarrow u B D z$   
 $B \rightarrow B v$   
 $B \rightarrow w$   
 $D \rightarrow E F$   
 $E \rightarrow y$   
 $E \rightarrow$   
 $F \rightarrow x$   
 $F \rightarrow$

- (a) Compute os conjuntos FIRST, FOLLOW e Nullable para a gramática.  
(b) Construa a tabela de análise sintática LL(1).  
(c) Mostre evidências de que a gramática não é LL(1).  
(d) Modifique a gramática de forma que ela seja LL(1) e aceite a mesma linguagem. Mostre que a linguagem que você construiu é LL(1) construindo as tabelas e conjuntos necessários.

3.  **Exercício Teórico:** Modifique a gramática a seguir, adicionando o símbolo de fim de arquivo e removendo quaisquer recursões à esquerda e, se necessário, realizando a fatoração à esquerda. Após as modificações, diga se a gramática é ou não LL(1) através da construção da tabela de análise sintática.

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

4.  **Exercício Teórico:** Considere a gramática de expressões apresentada abaixo:

$_0 E \rightarrow E + E$   
 $_1 E \rightarrow E * E$   
 $_2 E \rightarrow ( E )$   
 $_3 E \rightarrow num$

Tal gramática é ambígua, pois para a cadeia **num+num\*num** existem duas árvores de derivação. Se por exemplo o *token* **num** for substituído pelo número **3**, a expressão **3+3\*3** poderia ser avaliada de duas formas:

$$3 + 3 * 3 = 3 + 9 = 12 \text{ (a)}$$

$$3 + 3 * 3 = 6 * 3 = 18 \text{ (b)}$$

Isso acontece porque a gramática não leva em consideração a precedência dos operadores. Como o operador  $*$  (multiplicação) possui maior precedência, ele deve ser avaliado primeiro, o que implica que o resultado correto da expressão é o apresentado em **(a)**. De modo a evitar a ambiguidade na gramática e de modo a exprimir a correta precedência dos operadores, uma gramática como a apresentada acima seria modificada, de modo que a árvore de derivação da cadeia **3+3\*3** seja construída de forma a fazer com que o operador de multiplicação seja avaliado primeiro que o operador de adição.

Considere agora os seguintes símbolos terminais:

**id num + \* ++ ( )**

Considere agora a seguinte ordem de precedência dos operadores:

$++$   
 $*$   
 $+$

isto é, primeiro se avalia o operador  $++$  (pós-incremento), depois se avalia o operador  $*$  (multiplicação) e por último, o operador  $+$  (adição). Desta forma, a expressão **1+3\*5++**, tem a seguinte ordem de avaliação:

$$\begin{aligned}1 + 3 * 5 + + \\1 + 3 * 6 \\1 + 18 \\19\end{aligned}$$

Escreva uma gramática cuja linguagem gerada sejam todas as expressões válidas que se podem formar com os *tokens* **id num + \* ++ ( )**, sendo que a mesma deve respeitar a ordem de precedência dos operadores. Todos os operadores podem ser aplicados a números e identificadores. Desta forma as seguintes cadeias (mostradas sem o símbolo de fim de arquivo) são válidas:

$$\begin{aligned}num + + \\id + + \\num + id \\id * id + + \\(id * id) + +\end{aligned}$$

Construa a sua gramática de modo que ela seja LL(1), e com a devida tabela de análise, determine se as seguintes cadeias pertencem ou não a linguagem gerada pela gramática. **IMPORTANTE:** Lembre-se que \$ corresponde ao símbolo de fim de arquivo.

- (a) **num+id++\$**
- (b) **num++\*num\$**
- (c) **num++\*id+num\$**
- (d) **num++\*(id+num)++\$**
- (e) **id\*(id++num)\$**