1) Defina derivação mais à esquerda e derivação mais à direita.

Derivação mais à esquerda: ocorre quando o não-terminal substituído é sempre mais à esquerda.

Por exemplo, se a cadeia "1 + 1 + a" é derivada de acordo com a derivação mais à esquerda:

$S \rightarrow S + S$ (1)

$\rightarrow 1 + S$ (2)

$\rightarrow 1 + S + S$ (1)

$\rightarrow 1 + 1 + S$ (2)

$\rightarrow 1 + 1 + a$ (3)

a estrutura da cadeia seria: $\{ \{ 1 \}_S + \{ \{ 1 \}_S + \{ a \}_S \}_S \}_S$ onde $\{ ... \}_S$ indica uma sub-cadeia reconhecida como parte de S.

Derivação mais à direita: ocorre quando o não-terminal substituído é sempre mais à direita.

$S \rightarrow S + S$ (1)
$\rightarrow S + a$ (3)
$\rightarrow S + S + a$ (1)
$\rightarrow S + 1 + a$ (2)
$\rightarrow 1 + 1 + a$ (2)

2) Elabore a derivação mais à esquerda e a derivação mais à direita da cadeia (nome+nome) +nome*nome usando a gramática abaixo. Expr é o símbolo inícial.

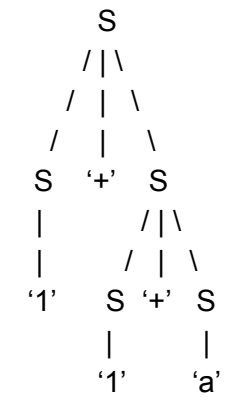
Expr -> (Expr)
Expr -> Expr Op Id | Id
Id -> Nome
Op -> + | - | X | /

Derivação mais a esquerda:
Expr -> Expr Op id -> Expr Op Id Op id -> (Expr) Op Id Op Id (Expr Op id) Op id Op id -> (id op id) op id op id -> (nome op id) op id op id -> (nome + id) op id op id -> (nome + nome) op id op id -> (nome + nome) + id op id -> (nome + nome) + nome op id -> (nome + nome) + nome * id -> (nome + nome) + nome * nome

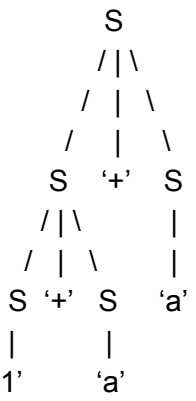
Derivação mais a direita:
Expr -> Expr Op id -> Expr op nome -> Expr * nome -> Expr op id * nome -> Expr op nome * id -> Expr + nome * nome -> (Expr) + nome*nome -> (Expr Op id)+ nome*nome -> (Expr Op nome)+ nome*nome -> (Expr + nome) + nome*nome -> (id + nome) + nome*nome -> (nome + nome) +nome*nome

3) Ainda sobre a questão 1, construa as árvores sintáticas das duas derivações.

Derivação à esquerda:



Derivação à direita:

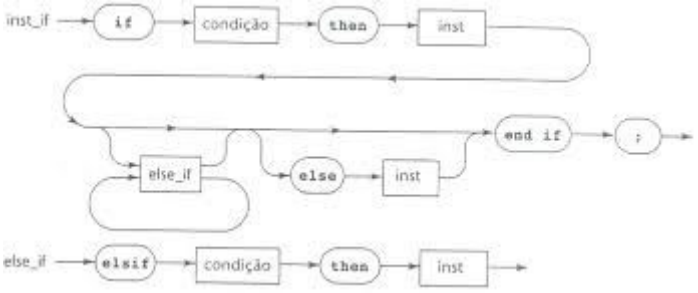


4) Se w é uma cadeia pertencente a uma linguagem livre de contexto arbitrária cuja gramática G. As únicas derivações possíveis são a derivação à esquerda e a derivação à direita? Por que?

Essas derivações são as únicas possíveis porque, na maioria dos parsers a transformação da entrada é definida como dar um pedaço de código para cada regra gramatical. Por isso, é importante saber se o parser determina o tipo de derivação mais a esquerda ou à direita, porque isso determina a ordem em que as partes de código vão ser executadas. A derivação também impõe um certo sentido, uma estrutura hierárquica na cadeia que vai ser derivada.

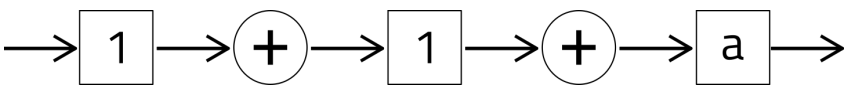
5) O que é um grafo sintático? Construa o grafo sintático da gramática da questão 1.

Grafos sintáticos são um outro tipo de notação usada para representar gramáticas. Esta notação tem o mesmo poder de expressão de BNF, porém define uma representação visual para as regras de uma gramática livre de contexto.



Na notação de grafos sintáticos, símbolos terminais são representados por círculos e símbolos não-terminais por retângulos. Setas são utilizadas para indicar a seqüência de expansão de um símbolo não-terminal.

Cadeia "1 + 1 + a":



6) Enumere os símbolos não terminais da linguagem EWE descrita abaixo com a notação EBNF. O símbolo inicial é ramprog.

```
1.<ramprog> ::= <executable> <equates> EOF
2.
3.<executable> ::= <labeled instruction> | <labeled instruction> <executable
4.
5.<labeled instruction> ::= Identifier ":" <labeled instruction> | <instr>
6.
7.<instr> ::=
8.       <memref> ":=" Integer
9.   |   <memref> ":=" "PC" "+" Integer
10.  |   "PC" ":=" <memref>
11.  |   <memref> ":=" <memref>
12.  |   <memref> ":=" <memref> "+" <memref>
13.  |   <memref> ":=" <memref> "-" <memref>
14.  |   <memref> ":=" <memref> "*" <memref>
15.  |   <memref> ":=" <memref> "/" <memref>
16.  |   <memref> ":=" <memref> "%" <memref>
17.  |   <memref> ":=" "M" "[" <memref> "+" Integer "]"
18.  |   "M" "[" <memref> "+" Integer "]" ":=" <memref>
19.  |   "read" "("<memref> ")"
20.  |   "writeln" "(" <memref> ")"
21.  |   "goto" Integer
22.  |   "goto" Identifier
23.  |   "if" <memref> <condition> <memref> "then" "goto" Integer
24.  |   "if" <memref> <condition> <memref> "then" "goto" Identifier
25.  |   "halt"
26.  |   "break"
27.
28.<equates> ::= null | "equ" Identifier "M" "[" Integer "]" <equates>
29.
30.<memref> ::= "M" "[" Integer "]" | Identifier
31.
32.<condition> ::= ">=" | ">" | "<=" | "<" | "=" | "<>"
```

1 2 3 4 5 6 7 8

<ramprog> <executable> <equates> <labeled instruction> <instr> <menref> <condition> <nemref>