Licenciatura em Engenharia Informática - DEI/ISEP

Linguagens de Programação 2019/2020

Resolução da Ficha PL 2

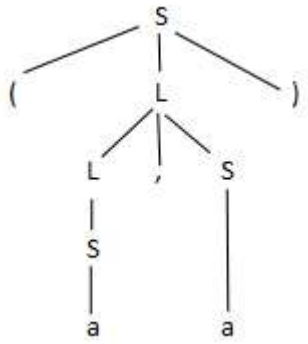
Gramáticas

1.

a) Identifique os símbolos terminais e não terminais desta gramática;

R: Terminais: ( ) , a Não-terminais: L S

b) Determine uma árvore de derivação desta gramática para (a,a);



c) Crie um autómato equivalente a esta gramática;

Não é possível porque a gramática não é regular (tipo 3 da hierarquia de Chomsky). A gramática tem que ser do tipo 3 e ser linear à direita.

d) Caracterize formalmente a linguagem representada por esta gramática.

G=({S,L},{a,(,),",",},{S→(L)|a, L→L,S|S},S)

2. Escreva uma gramática capaz de reconhecer cada uma das seguintes linguagens:

a) Palavras no alfabeto Σ = {a, b} que terminam em "b" e começam em "a"

S→aTb

T→aT|bT|ε

b) Palavras no alfabeto Σ = {a, b, c, d} em que um "b" é sempre precedido de um "a"

S→aS|abS|cS|dS|ε

c) Palavras no alfabeto Σ= {a, b, c, d} que são palíndromas e têm um comprimento maior que 1;

S→aTa|bTb|cTc|dTd

T→S|a|b|c|d|ε

3.Defina uma gramática capaz de representar uma quantia monetária nas moedas apresentadas na tabela seguinte.

Tabela 4.1: Representação de moedas

Moeda	**Exemplo**
Euro	`e12,23; e1,00; e2,35; 23,50EUR`
Libra	`£12.50; £22.12; £22.99`
Dólar	`$25.13; $5.00; $0.30`
Escudo	`12$50; 25$00; 150$00; 0$50`

<quantias> → <euro> | <libra> | <dolar> | <escudo>
<euro> → €<real> | <real>EUR
<libra> → £<real>
<dolar> → $<real>
<escudo> → <int>$<int>
<real> → <int>,<int> | <int>.<int> | int
<int> → <digito><int> | <digito>
<digito> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

4. Considere a seguinte gramática G = (V, Σ,P,S):

S→ TyBT
T→ xT | x
B→ zB | ε

a) Defina formalmente a gramática G.

G=({S,T,B},{x,y,z},{S→TyBT,T→xT|x,B→zB|ε},S)

b) Classifique a gramática G, segundo a hierarquia de Chomsky. Justifique.
Trata-se de uma gramática de nível 2 (Gramática Independente do Contexto- GIC), definida formalmente por G = (V,∑, P, S) onde:

$X \rightarrow \alpha$ onde $X \in V$ e α uma sequência de símbolos terminais e não terminais

Note-se que a parte esquerda das produções tem de conter, obrigatoriamente, um único símbolo não terminal, enquanto que a parte direita é composta por uma sequência de símbolos terminais e não-terminais;

Para ser de nível 3 (gramática regular) além das restrições do lado esquerdo das produções, devido ao facto de ser uma GIC, têm também restrições do lado direito das produções:

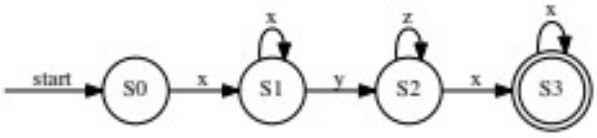
- Todas as produções devem ter apenas um símbolo não terminal do lado direito, que irá preceder ou suceder qualquer sub-palavra de terminais($\alpha \in \Sigma^*$)
- Todas as produções serem lineares à esquerda ou à direita

O que não acontece nesta gramática acima,

- A regra associada a S→TyBT tem mais do que 1 símbolo não terminal do lado direito

c) Caso seja possível, represente um Autómato Finito que reconheça a linguagem gerada pela gramática G. Justifique.

$S_0 \rightarrow xS_1$
$S_1 \rightarrow xS_1 | yS_2$
$S_2 \rightarrow xS_3 | zS_2$
$S_3 \rightarrow xS3 | \varepsilon$



5. Considere a expressão regular ($x^+yz^*x^+$):
a) Representa a mesma linguagem que a gramática G do exercício 4? Justifique.
Sim, aceita pelo menos um x, seguido de exatamente um y, seguido de zero ou mais z e terminando em pelo menos um x.

b) Verifique se palavra xyyzzxx é válida no âmbito da expressão regular anterior. Justifique.
Não é válida. A gramática aceita apenas um y.

6. Considere a gramática G=({S,L,A},{(,),a,+,b},{S→ (L)|a, L→A+S|b, A→a|L}, S):
a) Considere a palavra (b+(a+a)+a), valide se pertence à linguagem gerada pela gramática. Apresente a sequência de derivação mais à direita para a frase.

S→ (L)|a
L→A+S|b
A→a|L

S⇒ (L) ⇒(A+S) ⇒(A+a)→(L+a) ⇒(A+S+a) ⇒(A+(L)+a) ⇒ (A+(A+S)+a) ⇒ (A+(A+a)+a) ⇒
(A+(a+a)+a) ⇒ (L+(a+a)+a) ⇒ (b+(a+a)+a)

b) Classifique a gramática G segundo a hierarquia de Chomsky. Justifique.

Trata-se de uma gramática de nível 2 (Gramática Independente do Contexto- GIC), definida formalmente por G = (V,∑, P, S) onde:

X →α onde X ∈ V e α uma sequência de símbolos terminais e não terminais

Note-se que a parte esquerda das produções tem de conter, obrigatoriamente, um único símbolo não terminal, enquanto que a parte direita é composta por uma sequência de símbolos terminais e não-terminais;

Para ser de nível 3 (gramática regular) além das restrições do lado esquerdo das produções, devido ao facto de ser uma GIC, têm também restrições do lado direito das produções:

- Todas as produções devem ter apenas um símbolo não terminal do lado direito, que irá preceder ou suceder qualquer sub-palavra de terminais($\alpha \in \Sigma^*$)
- Todas as produções serem lineares à esquerda ou à direita

O que não acontece na gramática acima,

- Uma das regras associadas ao símbolo não terminal S é precedido e sucedido por um símbolo terminal (auto-contenção) $S \rightarrow (L)|a$
- Uma das regras associadas ao símbolo não terminal L tem 2 símbolos não terminais do lado direito $L \rightarrow A+S$

c) Defina uma expressão regular equivalente à gramática G.

Não é possível porque a gramática não é regular (tipo 3 da hierarquia de Chomsky). A gramática tem que ser do tipo 3 e ser linear à direita;

d) Apresente uma gramática equivalente à Expressão Regular (#|@)*(@|#)#*

$S \rightarrow \#S|@S|@T|\#T$

$T \rightarrow \#T|\varepsilon$

OU

$S \longrightarrow \#S|@S|\#|@$

7. Considere uma gramática G tal que:

S → aS | Sb | ab | SS

a) Escreva uma expressão regular para reconhecer a linguagem definida pela gramática G, ou seja, L(G).

L(G) = a(a|b)*b

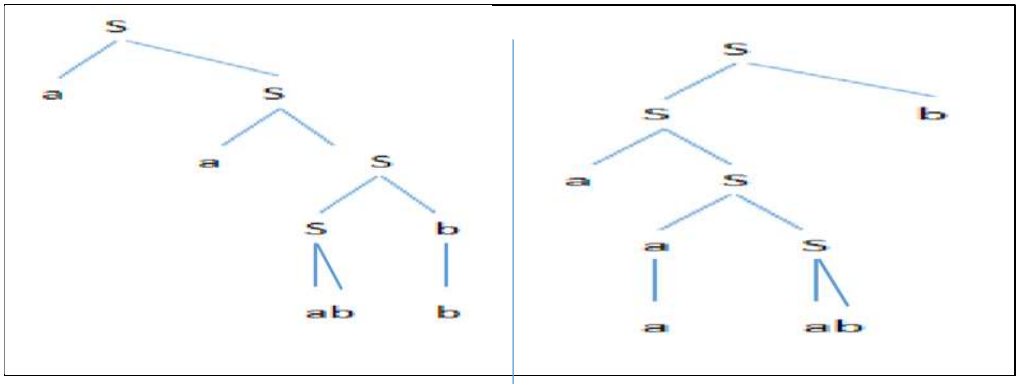
b) Escreva uma sequência de derivação para aaabb.

$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaSb \Rightarrow aaabb$

c) Classifique a gramática G quanto à ambiguidade.

Existe uma outra sequência de derivação possível para **aaabb** à qual corresponde outra árvore de parse ou derivação.

S⇒ Sb ⇒ aSb ⇒ aaSb ⇒ aaabb



A gramática é ambígua porque existe pelo menos uma frase da linguagem (aaabb), gerada pela gramática, **com mais do que uma árvore de parse.**

8. Considere a seguinte gramática:

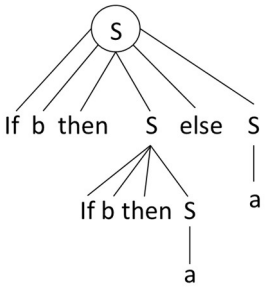
S → `if b then` **S** `else` **S**
S → `if b then` **S**
S → `a`

a) Mostre que a gramática em questão é ambígua (por exemplo, encontre uma frase que tenha duas árvores sintáticas).

Frase: if b then if b then a else a

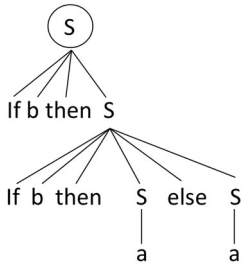
Sequência de derivação 1:

S ⇒ if b then S else S ⇒ if b then if b then S else S ⇒ if b then if b then a else S ⇒ if b then if b then a else a



Sequência de derivação 2:

S ⇒ if b then S⇒ if b then if b then S else S ⇒ if b then if b then a else S ⇒ if b then if b then a else a

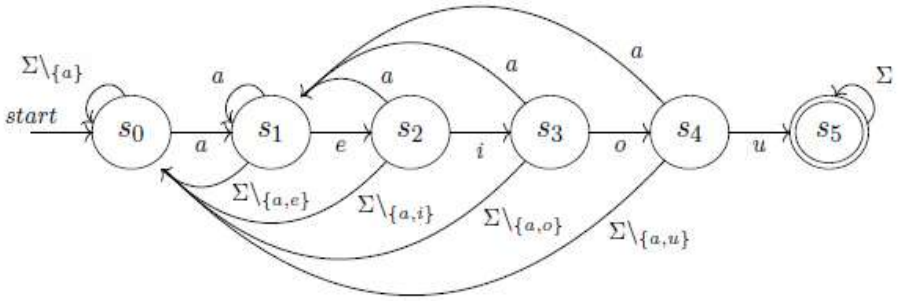


A gramática é ambígua porque existe pelo menos uma frase da linguagem (if b then if b then a else a), gerada pela gramática, com mais do que uma árvore de parse.

b) Escreva uma gramática equivalente não ambígua.

S⟶ A|B
A⟶ if b then A else A | a
B⟶ if b then S|if b then A else B

9. Converta o seguinte autómato numa gramática:



$S_0 \rightarrow aS_1 \mid eS_0 \mid iS_0 \mid oS_0 \mid uS_0$
$S_1 \rightarrow aS_1 \mid eS_2 \mid iS_0 \mid oS_0 \mid uS_0$
$S_2 \rightarrow iS_3 \mid aS_1 \mid eS_0 \mid oS_0 \mid uS_0$
$S_3 \rightarrow oS_4 \mid aS_1 \mid eS_0 \mid iS_0 \mid uS_0$
$S_4 \rightarrow uS_5 \mid aS_1 \mid eS_0 \mid iS_0 \mid uS_0$
$S_5 \rightarrow aS_5 \mid eS_5 \mid iS_5 \mid oS_5 \mid uS_5 \mid \varepsilon$

10. De seguida apresentam-se exemplos de declarações válidas.

- **int a, x1=10, y=x1;**
- **long int numero;**
- **unsigned char c='a';**
- **long double real=1.234e-5, pi=3.14159265358979, num1, num2;**

Os tipos válidos são int, char, float e double, os modifiers que podem aparecer antes do tipo são short, long e unsigned. Os identificadores são uma letra seguida de zero ou mais letras e algarismos. Opcionalmente pode ser efectuada uma atribuição de um valor (constante ou variável).
Crie uma gramática capaz de reconhecer este tipo de declarações de variáveis. Não é necessário validar a coerência de tipos nas atribuições nem combinações inválidas tipo short, char. Use como ponto de partida a seguinte gramática:

<declaracoes> → <declaracao><declaracoes>|ε
<declaracao> → <tipo><lista_variaveis>;
<tipo> → …
<lista_variaveis> → …

<declaracoes> → <declaracao><declaracoes> | ε
<declaracao> → <tipo><lista_variaveis>;
<tipo> → <modificador><tipovar>|<tipovar>
<lista_variaveis> → <var>, <lista_variaveis> | <var>
<modificador> → short | long | unsigned
<tipovar> → int | char | float | double
<var> → <id> | <id>=<valor> | <id>=<id>
<id> → <letra><alfas>
<valor> → <id> | <inteiro> | <char> | <real>
<letra> → a | A | ... | z | Z
<alfas> → <alfa><alfas>|<alfa>
<inteiro> → <digito><inteiro> | <digito>
<char> → '<letra>'
<real> → <inteiro>.<inteiro> | <inteiro>.<inteiro><expoente>
<alfa> → <letra> | <digito>
<digito> → 0 | .. | 9
<expoente> → e-<inteiro> | e+<inteiro>

11. Crie um programa em FLEX, que identifique os tokens presentes em expressões lógicas válidas na gramática a seguir descrita, sendo que quando um token for identificado, a função yylex deverá devolver o identificador respectivo.

E → E or E|E and E|E xor E| not E| (E) | ID |INT|REAL
NOTA: Um ID representa um identificador (uma letra seguida de letras ou algarismos), INT um número inteiro e REAL um número real. Os espaços, tabs e mudanças de linha devem ser ignorados. Qualquer outro carácter deve originar um erro.

```
%{
        enum{OR,XOR,AND,NOT,ABRIR_P,FECHAR_P,INT,REAL,ID,FIM};
%}
 /* definição de dígito */
digito [0-9]
letra [a-zA-Z]
```

```
%%
or                                                  return OR;
xor                                                 return XOR;
and                                                 return AND;
not                                                 return NOT;
"("                                                 return ABRIR_P;
")"                                                 return FECHAR_P;
{digito}+                                           return INT;
({digito}*,)?{digito}+([Ee][-+]?{digito}+)?         return REAL;
{letra}({letra}|{digito})*                          return ID;
[ \t\n]                                             ;
<<EOF>>                                             return FIM;
.                                                   printf ("Erro
léxico:%s\n",yytext);
%%

int main(){
  int token;
  printf("Insira dados:\n");
  do{
    token=yylex();
    switch(token){
      case OR        : printf("token OR\n");
                   break;
      case XOR       : printf("token XOR\n");
                   break;
      case AND       : printf("token AND\n");
                   break;
      case ABRIR_P   : printf("token '('\n");
                   break;
      case FECHAR_P  : printf("token ')'\n");
                   break;
      case INT       : printf("token INT %s\n",yytext);
                   break;
      case REAL      : printf("token REAL %s\n",yytext);
                   break;
      case ID        : printf("token ID %s\n",yytext);
                   break;
      case FIM       : printf("token FIM %s\n",yytext);
                   break;
      default        : printf("token desconhecido %s\n",yytext);
    }
  }while (token!=FIM);
}

/* se definir esta função não necessita de compilar com o parâmetro -lfl
*/
int yywrap(){
       return(1);
}
```