Licenciatura em Engenharia Informática – DEI/ISEP
**Linguagens de Programação 2006/07**

# Ficha 5

# Gramáticas

**Objectivos:**

- Introdução aos conceitos e notações relacionados com gramáticas;
- Árvores de derivação;
- O problema da ambiguidade;
- Conversão entre autómatos finitos e gramáticas;
- *Parser* em descida recursiva;
- Aprendizagem dos conceitos através da realização de exercícios.

## 5.1 Introdução

Uma gramática é uma ferramenta poderosa para a descrição e análise de linguagens. Uma gramática é constituída por um conjunto de regras segundo as quais as frases válidas da linguagem são construídas. Considere-se o seguinte excerto de uma gramática:

**frase** → <**sintagma-nominal**><**sintagma-verbal**>
**sintagma-nominal** → <**artigo**><**nome**> | <**nome**>
**sintagma-verbal** → <**verbo**><**sintagma-nominal**>
**artigo** → *o* | *a* | *os* | *as*
**nome** → *Pedro* | *Maria* | *crianças* | *rapazes* | *cartas* | *futebol*
**verbo** → *conhece* | *conhecem* | *é* | *são* | *joga* | *jogam*

Com estas regras, também designadas por produções, é possível analisar frases como as seguintes:

*os rapazes jogam futebol.*
*o Pedro conhece a Maria.*
*a Maria é criança.*

Considere-se a seguinte derivação à esquerda:

$$
\begin{aligned}
\textbf{frase} &\Rightarrow \langle\textbf{sintagma-nominal}\rangle\langle\textbf{sintagma-verbal}\rangle \\
&\Rightarrow \langle\textbf{artigo}\rangle\langle\textbf{nome}\rangle\langle\textbf{sintagma-verbal}\rangle \\
&\Rightarrow \textit{os}\ \langle\textbf{nome}\rangle\langle\textbf{sintagma-verbal}\rangle \\
&\Rightarrow \textit{os rapazes}\ \langle\textbf{sintagma-verbal}\rangle \\
&\Rightarrow \textit{os rapazes}\ \langle\textbf{verbo}\rangle\langle\textbf{sintagma-nominal}\rangle \\
&\Rightarrow \textit{os rapazes jogam}\ \langle\textbf{sintagma-nominal}\rangle \\
&\Rightarrow \textit{os rapazes jogam}\ \langle\textbf{nome}\rangle \\
&\Rightarrow \textit{os rapazes jogam futebol}
\end{aligned}
$$

O processo de derivação pode ser representado graficamente através de uma árvore (de derivação) conforme a figura 5.1.

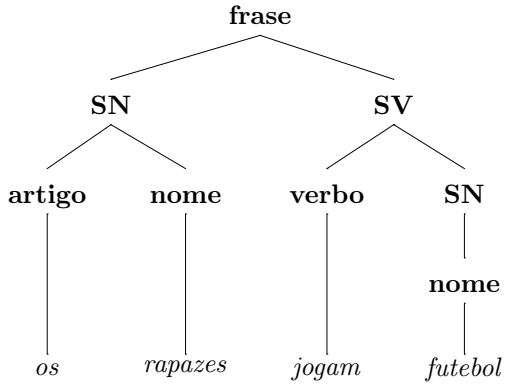



Figura 5.1: Árvore sintáctica

## 5.2 Definições e notação

Uma **gramática** é um conjunto de regras segundo o qual as frases válidas de uma linguagem são construídas. Formalmente, uma gramática é definida por um tuplo:

$$G = (V, \Sigma, P, S)$$

no qual:

- $V$ – é um conjunto finito, não vazio, de **variáveis** (símbolos **não terminais**). Estes são os símbolos da gramática que podem ser substituídos por uma sequência de símbolos;
- $\Sigma$ – é um conjunto finito, não vazio, dito **alfabeto** ou conjunto de símbolos **terminais**. Estes são os símbolos da gramática que não podem ser substituídos, correspondem às palavras válidas na linguagem;
- **Produção** – é uma regra da gramática que define a forma como os símbolos não terminais podem ser substituídos. A sua forma geral é a seguinte:

  $$X \rightarrow Y_1 Y_2 Y_3 \dots Y_n$$

  Neste exemplo o símbolo não terminal $X$ é definido como equivalente à concatenação dos símbolos $Y_1 Y_2 Y_3 \dots Y_n$. Note-se que a parte esquerda tem de conter, obrigatoriamente, um símbolo não terminal, enquanto que a parte direita é composta por uma sequência de terminais e não-terminais;
- $S$ – uma gramática contém, obrigatoriamente, um símbolo não-terminal (**símbolo inicial**) a partir do qual todas as frases são derivadas.

Uma **derivação** é uma sequência de aplicações de regras das gramática que produzem um cadeia de símbolos terminais. A processo de substituição pode implicar zero ou mais passos, sendo que, após todas as substituição é obtida uma frase contendo apenas símbolos terminais. Assim, diz-se que $u \Rightarrow v$, se $u = x\alpha y \wedge v = x\beta y$ sendo que $x, y \in (V \cup \Sigma)^* \wedge \alpha \rightarrow \beta \in P$

O número de passos utilizados na derivação é notado da seguinte forma:

$u \Rightarrow v$ – diz-se que $u$ deriva em $v$ num passo;
$u \Rightarrow^* v$ – diz-se que $u$ deriva em $v$ em zero ou mais passos;
$u \Rightarrow^+ v$ – diz-se que $u$ deriva em $v$ em um ou mais passos;

A linguagem gerada por uma gramática $L(G)$ é dada pelo conjunto de todas as derivações que, partindo do estado inicial $S$, originam uma sequência de símbolos do alfabeto:

$$L(G) = \{u \in \Sigma^* : S \Rightarrow^* u\}$$

### 5.2.1 Hierarquia de Chomsky

Noam Chomsky, um linguista americano, criou uma hierarquia para as gramáticas formais, dividindo-as em quatro tipos. Estas vão desde as gramáticas do **tipo 0**, as mais abrangentes, até às do **tipo 3**, as mais restritivas (ver diagrama da figura 5.2).

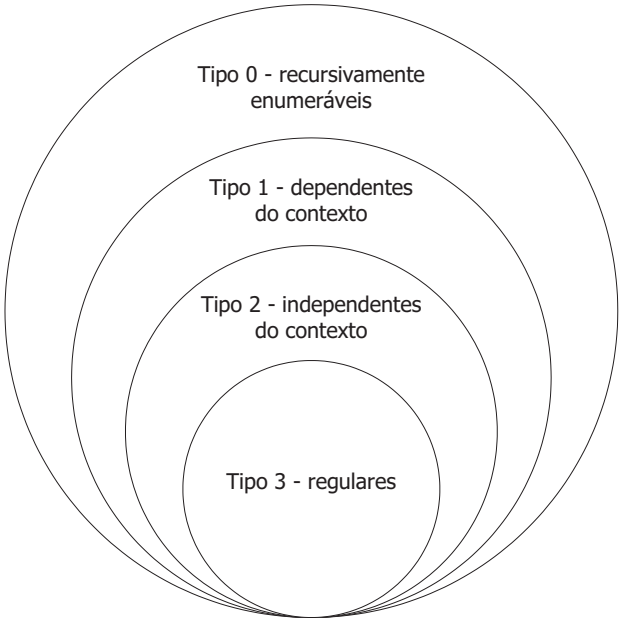



Figura 5.2: Hierarquia de Chomsky

- **tipo 0** – gramáticas livres ou sem restrições, são as mais abrangentes possíveis. As produções são da forma $\alpha \rightarrow \beta$ na qual tanto $\alpha$ como $\beta$ são sequências arbitrárias de símbolos terminais e não terminais. O lado esquerdo da produção não pode ser vazio;

- **tipo 1** – gramáticas dependentes do contexto. A produções são da forma $\alpha A \beta \rightarrow \alpha \gamma \beta$ na qual $\alpha$,$\beta$ e $\gamma$ são sequências arbitrárias de símbolos terminais e não terminais, sendo que $\gamma$ não é nulo e $A$ é um não terminal singular. Por outras palavras, $A$ pode ser substituído por $\gamma$ quando rodeados por $\alpha$ e $\beta$, isto é, num determinado contexto;

- **tipo 2** – gramáticas independentes do contexto. A produções são da forma $A \rightarrow \alpha$ na qual $\alpha$ é sequência arbitrária de símbolos terminais e não terminais, sendo $A$ um não terminal singular. Tal significa que qualquer ocorrência de $A$ pode ser substituído por $\alpha$ independentemente do contexto;

- **tipo 3** – gramáticas regulares. A produções são da forma $A \rightarrow a$, $A \rightarrow aB$ ou $A \rightarrow \varepsilon$ na qual $A$ e $B$ são não terminais singulares e $a$ um terminal. Estas são as formas de gramáticas mais restritas em termos de poder de representação.

As gramáticas do **tipo 3** dizem-se **lineares à direita** se as produções são da forma $A \rightarrow a$, $A \rightarrow aB$ ou $A \rightarrow \varepsilon$. Enquanto que nas **lineares à esquerda** as produções são da forma $A \rightarrow a$, $A \rightarrow Ba$ ou $A \rightarrow \varepsilon$. Note-ser que apenas

as gramáticas do **tipo 3** são lineares. Por outro lado toda a gramática linear, à esquerda ou direita, é regular.

### 5.2.2 Notação BNF e EBNF (Extended BNF)

A notação BNF (Backus Naur Form ou Backus Normal Form) foi originalmente criada por John Backus e Peter Naur, no final dos anos 50, para descrever a linguagem ALGOL. Desde então a sua utilização generalizou-se para a especificação de linguagens de programação.

Os meta-símbolos utilizados na notação BNF são:

::= – representa "definido como";
| – indica uma alternativa;
< > – indicam uma regra

Considere-se o seguinte exemplo no qual é especificada a gramática relativa à instrução *if-then-else* e à definição de um identificador, vulgarmente utilizados em linguagens de programação, em notação BNF.

<**if**> ::= *if* <**condição**> *then* <**instruções**> *endif*
| *if* <**condição**> *then* <**instruções**> *else* <**instruções**> *endif*
<**identificador**> ::= <**letra**><**alfanum**> | _<**alfanum**>
<**alfanum**> ::= <**letra**> | <**algarismo**> | _
<**letra**> ::= *a* | *A* | *b* | ... | *z* | *Z*
<**algarismo**> ::= *0* | *1* | *2* | ... | *8* | *9*

A notação EBNF extende a notação BNF com os seguinte meta-símbolos:

[ ] – indica uma parte opcional;
{ } – indica uma parte que se pode repetir 0 ou mais vezes;
( ) – indica precedências dentro da regra ;
" " – indica um carácter a tratar como terminal *e.g.*, "<".

Considere-se o seguinte especificação da gramática relativa à instrução *if-then-else* e à definição de um identificador, com recurso à notação EBNF.

<**if**> ::= *if* <**condição**> *then* <**instruções**>[ *else* <**instruções**>] *endif*
<**identificador**> ::= (<**letra**> | _){(<**letra**> | <**algarismo**> | _)}
<**letra**> ::= *a* | *A* | *b* | ... | *z* | *Z*
<**algarismo**> ::= *0* | *1* | *2* | ... | *8* | *9*

Mais recentemente nas bibliografias tornou-se comum a indicação dos não terminais a **negrito** sem os símbolos "<" e ">" e os terminais em texto normal.

## 5.3 Árvores de Derivação e Ambiguidade

Dada uma gramática $G = (V, \Sigma, P, S)$ independente de contexto, e dada uma derivação em G, $S \Rightarrow^* u$, define-se a sua árvore de derivação da seguinte forma:

- o nó inicial é o símbolo inicial S;
- em cada passo da derivação, para cada variável $A$ que seja uma folha da árvore, faz-se corresponder a produção $A \rightarrow \alpha_1\alpha_2 \ldots \alpha_n$ a uma sub-árvore, em que $A$ é o pai e $\alpha_1$, $\alpha_1$, ..., $\alpha_n$ são os filhos;

No fim, a árvore de derivação terá a seguinte configuração:

- o nó principal é o símbolo inicial S;
- os não terminais são nós interiores;
- os terminais são folhas.

Diz-se que uma gramática é ambígua, se é possível derivar uma mesma frase a partir de duas sequências de derivação distintas.

Considere-se a seguinte gramática.

$$\begin{aligned} \mathbf{E} &\rightarrow \langle\mathbf{E}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \mid \langle\mathbf{ID}\rangle \\ \mathbf{OP} &\rightarrow + \mid - \mid \times \mid \div \\ \mathbf{ID} &\rightarrow x \mid y \mid z \end{aligned}$$

Esta gramática é ambígua, pois conforme apresentada na figura 5.3 existem pelo menos duas sequências de derivação distintas para a frase "$x + y \times z$". De seguida apresentam-se as duas sequências de derivação para esta frase, usando a derivação mais à esquerda.

$$\begin{aligned} \langle\mathbf{E}\rangle &\Rightarrow \langle\mathbf{E}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow \langle\mathbf{E}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow \\ &\Rightarrow \langle\mathbf{ID}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow x\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow \\ &\Rightarrow x+\langle\mathbf{E}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow x+\langle\mathbf{ID}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow x+y\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow \\ &\Rightarrow x+y\times\langle\mathbf{E}\rangle \Rightarrow x+y\times\langle\mathbf{ID}\rangle \Rightarrow x+y\times z \end{aligned}$$

$$\begin{aligned} \langle\mathbf{E}\rangle &\Rightarrow \langle\mathbf{E}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow \langle\mathbf{ID}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow x\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow x+\langle\mathbf{E}\rangle \Rightarrow \\ &\Rightarrow x+\langle\mathbf{E}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow x+\langle\mathbf{ID}\rangle\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow x+y\langle\mathbf{OP}\rangle\langle\mathbf{E}\rangle \Rightarrow \\ &\Rightarrow x+y\times\langle\mathbf{E}\rangle \Rightarrow x+y\times\langle\mathbf{ID}\rangle \Rightarrow x+y\times z \end{aligned}$$

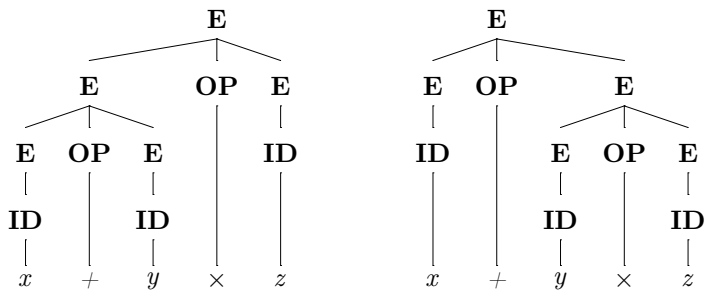



Figura 5.3: Árvores sintácticas para expressão ambígua

## 5.4 Conversão de autómatos finitos em gramáticas

Para a conversão dum AF numa gramática é necessário criar uma produção para cada estado. Após o que, para cada transição é necessário criar uma alternativa na produção. Se a transição puder ter várias alternativas, estas podem desdobradas ou criada uma nova produção com essas alternativas. Todos os estados finais podem receber a cadeia vazia ($\varepsilon$).

Considere-se o autómato finito não determinístico apresentado na figura 5.4.

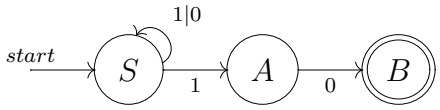



Figura 5.4: AFN a converter numa gramática regular

A gramática correspondente é a seguinte:

$$\mathbf{S} \rightarrow \mathit{0}\mathbf{S} \mid \mathit{1}\mathbf{S} \mid \mathit{1}\mathbf{A}$$
$$\mathbf{A} \rightarrow \mathit{0}\mathbf{B}$$
$$\mathbf{B} \rightarrow \varepsilon$$

Considere-se ainda o autómato finito determinístico capaz de reconhecer a mesma linguagem, apresentado na figura 5.5.

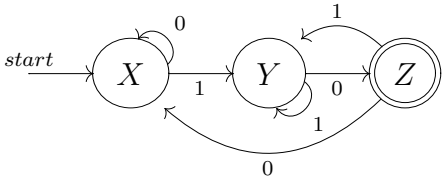



Figura 5.5: AFD a converter numa gramática regular

A gramática correspondente a este autómato é a seguinte:

$$\mathbf{X} \rightarrow \mathit{0}\mathbf{X} \mid \mathit{1}\mathbf{Y}$$
$$\mathbf{Y} \rightarrow \mathit{0}\mathbf{Z} \mid \mathit{1}\mathbf{Y}$$
$$\mathbf{Z} \rightarrow \mathit{0}\mathbf{X} \mid \mathit{1}\mathbf{Y} \mid \varepsilon$$

## 5.5 Conversão de gramáticas em autómatos finitos

Somente as gramáticas do tipo 3 podem ser convertidas em AF. Para a conversão a gramática deve ser linear à direita. Considere uma gramática capaz de reconhecer números binários pares:

$$L(G) = \{u \in \Sigma^* : \text{u é um número binário par}\}$$
$$G = (V, \Sigma, P, S) = (\{A, B\}, \{0, 1\}, P, A)$$

em que as produções ($P$) são:

$$\mathbf{A} \rightarrow \mathit{1}\mathbf{A} \mid \mathit{0}\mathbf{B}$$
$$\mathbf{B} \rightarrow \mathit{0}\mathbf{B} \mid \mathit{1}\mathbf{A} \mid \varepsilon$$

A primeira parte da conversão consiste na criação de um estado para cada produção, neste exemplo são necessários 2 estados, $A$ e $B$. Depois é necessários substituir as produções por transições da seguinte forma:

- As produções do tipo $X \rightarrow \alpha Y$ geram no autómato a transições do tipo $\delta(X, \alpha) = Y$.
- As produções do tipo $X \rightarrow \varepsilon$ implicam que o estado correspondente à produção seja um estado final;
- As transições do tipo $X \rightarrow \alpha$, implicam a criação um novo estado final extra, e essas produções são substituídas por transições para esse estado.

Na figura 5.6 está representado o autómato resultante da conversão desta gramática, neste caso determinístico.

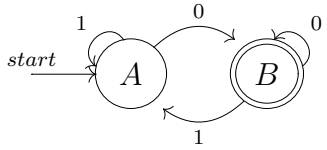



Figura 5.6: Gramática convertida numa AFD

## 5.6 *Parsers* em descida recursiva

A descida recursiva pode ser usada, para implementar *parsers* preditivos. Um *parser* preditivo é capaz de escolher a produção a aplicar simplesmente sabendo o não terminal actual e o terminal a ser processado. Este tipo de gramáticas são chamadas de LL(1), onde:

- o primeiro "L" indica que a frase é processada da esquerda para a direita;
- o segundo "L" indica que se usa primeiro a derivação mais à esquerda;
- o (1) indica o número de terminais de avanço necessários para escolher entre produções alternativas;

Todas as gramáticas do tipo 3 podem ser convertidas em LL(1). Considere a seguinte gramática capaz de a linguagem com o alfabeto $\Sigma = \{a, b, c, d\}$ em que o primeiro "b" é precedido de um "a":

$$\begin{aligned}
\mathbf{S} &\rightarrow a\mathbf{B} \mid c\mathbf{S} \mid d\mathbf{S} \\
\mathbf{A} &\rightarrow a\mathbf{A} \mid b\mathbf{B} \mid c\mathbf{S} \mid d\mathbf{S} \\
\mathbf{B} &\rightarrow a\mathbf{B} \mid b\mathbf{B} \mid c\mathbf{B} \mid d\mathbf{B} \mid \varepsilon
\end{aligned}$$

Para construir um *parser* em descida recursiva é necessário criar uma função para processar cada um dos não terminais. De seguida é necessário para cada não terminal, processar os *starters* e chamar as funções relativas aos não terminais que se lhes seguem. De seguida apresenta-se um *parser* em descida recursiva implementado em *FLEX* que reconhece a gramática apresentada. Este parser pode ser obtido em `http://www.dei.isep.ipp.pt/~matos/lprog/parserDR.flex`

```
%{
  enum{TOKEN_A,TOKEN_B,TOKEN_C,TOKEN_D,OUTRO,FIM};

  int token, nerros=0;

  void s(); /* protótipos */
  void a();
  void b();
%}

%%

a|A         return TOKEN_A;
b|B         return TOKEN_B;
c|C         return TOKEN_C;
d|D         return TOKEN_D;
.           return OUTRO;
\n          return FIM;
<<EOF>>     return FIM;
```

```
%%

void erro(char *s)
{
  nerros++;
  printf("%s<-Erro %d: %s\n",yytext,nerros,s);
  exit(1);
}

void getToken()
{
  printf("%s",yytext);
  token=yylex();
}

void s() /* S-> aA | cS | dS */
{
  switch (token) {
    case TOKEN_A : getToken(); a(); break;
    case TOKEN_C : getToken(); s(); break;
    case TOKEN_D : getToken(); s(); break;
    default    :
         erro("simbolo não reconhecido (esperava a,c ou d)");
  }
}

void a() /* A-> aA | bB | cS | dS  */
{
  switch (token) {
    case TOKEN_A : getToken(); a(); break;
    case TOKEN_B : getToken(); b(); break;
    case TOKEN_C : getToken(); s(); break;
    case TOKEN_D : getToken(); s(); break;
    default    :
         erro("simbolo não reconhecido (esperava a,b,c ou d)");
  }
}

void b() /* B-> aB | bB | cB | dB | vazio */
{
  switch (token) {
    case TOKEN_A : getToken(); b(); break;
    case TOKEN_B : getToken(); b(); break;
    case TOKEN_C : getToken(); b(); break;
    case TOKEN_D : getToken(); b(); break;
  }
  /* como pode ser vazio não dá erro */
}
```

```
int main()
{
  token=yylex(); /* vai buscar o 1º token */
  s();           /* chama a produção inicial */

  if (nerros==0 && token==FIM)
    printf("\nExpressao válida\n");
  else
    erro("simbolo não reconhecido (esperava a,b,c,d ou fim)");
  return 0;
}
```

## 5.7 Propostas de exercícios

1. Considere a seguinte gramática:

$$\mathbf{S} \rightarrow (\mathbf{L}) \mid a$$
$$\mathbf{L} \rightarrow \mathbf{L},\mathbf{S} \mid \mathbf{S}$$

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

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

   (c) Crie um autómato equivalente a esta gramática;

   (d) Caracterize formalmente a linguagem representada por esta gramática.

2. Considere a seguinte gramática:

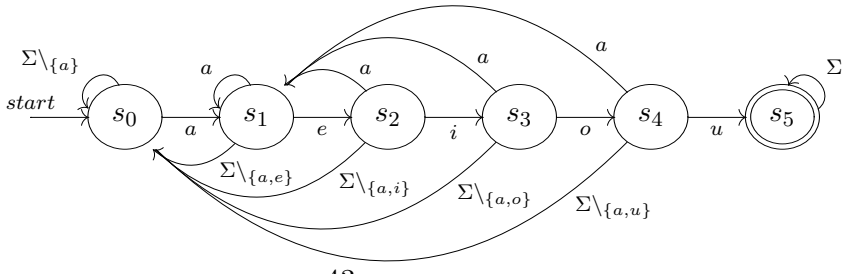
$$\mathbf{S} \rightarrow \textit{if } b \textit{ then } \mathbf{S} \textit{ else } \mathbf{S}$$
$$\mathbf{S} \rightarrow \textit{if } b \textit{ then } \mathbf{S}$$
$$\mathbf{S} \rightarrow a$$

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

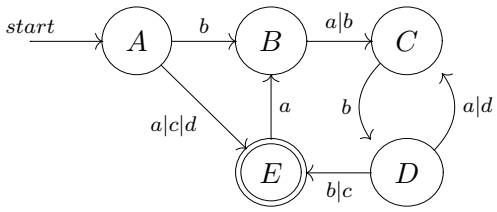
   (b) Escreva uma gramática equivalente não ambígua.

3. converta os seguintes autómatos em gramáticas:

(a)



(b)




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

   (a) Palavras no alfabeto $\Sigma = \{a, b\}$ que terminam em "b" e começam em "a";

   (b) Palavras no alfabeto $\Sigma = \{a, b, c, d\}$ em que um "b" é sempre precedido de um "a";

   (c) Palavras no alfabeto $\Sigma = \{a, b, c, d\}$ começam e terminam em "a";

   (d) Palavras no alfabeto $\Sigma = \{a, b, c, d\}$ que são capicuas e têm um comprimento maior que 1;

   (e) Números romanos menores que 50;

5. Suponha uma gramática com as seguintes produções (**S** é a produção inicial):

$$\begin{aligned}
&\mathbf{S} \rightarrow \mathbf{AB} \\
&\mathbf{A} \rightarrow a\mathbf{B} \mid \varepsilon \\
&\mathbf{B} \rightarrow b\mathbf{AC} \mid \varepsilon \\
&\mathbf{C} \rightarrow c(\mathbf{A}+\mathbf{B})
\end{aligned}$$

   (a) Escreva em *FLEX* um *parser* em descida recursiva que implemente a gramática anterior;

   (b) Diga se as seguintes frases pertencem à gramática, justificando.

      i. `abc(+ba)`
      ii. `abc(a+)`
      iii. `abc(a+b)`
      iv. `ac(a+a)a`

6. Considere uma gramática G tal que:

$$\mathbf{S} \rightarrow a\mathbf{S} \mid \mathbf{S}b \mid ab \mid \mathbf{SS}$$

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

(b) Escreva uma sequência de derivação para `aaabb`;

(c) Classifique a gramática G quanto à ambiguidade.

7. Considere uma calculadora básica capaz de processar as quatro operações algébricas fundamentais representadas pelos símbolos $+, -, *, /$, parêntesis e o '-' unário.

   (a) Escreva uma sintaxe para expressões algébricas válidas nesta calculadora;

   (b) Escreva uma árvore de derivação para a expressão: $2 + 3 * 4$. A árvore de derivação que obteve é única? Que problema está aqui em causa? Explique.

   (c) Adicione as operações: ^ e % à gramática.

8. Considere uma gramática capaz de identificar definições de variáveis na linguagem C. 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).

   (a) 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*".

   (b) Implemente esta gramática em *FLEX* através de um *parser* em descida recursiva. crie expressões regulares para reconhecer as diversas palavras reservadas, identificadores de variáveis, identificadores de variáveis, inteiros, reais, caracteres e os símbolos '=' e ';'.

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

| **Moeda** | **Exemplo** |
|---|---|
| Euro | €12,23; €1,00; €2,35; 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 |

10. 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.

$$\mathbf{E} \rightarrow \mathbf{E}\ \textit{or}\ \mathbf{E} \mid \mathbf{E}\ \textit{and}\ \mathbf{E} \mid \mathbf{E}\ \textit{xor}\ \mathbf{E} \mid \textit{not}\ \mathbf{E} \mid \textit{(}\mathbf{E}\textit{)} \mid \mathbf{ID} \mid \mathbf{INT} \mid \mathbf{REAL}$$

**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.

11. Implemente um analisador léxico para a seguinte gramática:

$$\mathbf{S} \rightarrow \mathbf{ID} = \mathbf{E} \mid \mathbf{E}$$
$$\mathbf{E} \rightarrow \mathbf{E} + \mathbf{E} \mid \mathbf{E} - \mathbf{E} \mid \mathbf{E} * \mathbf{E} \mid \mathbf{E} / \mathbf{E} \mid -\mathbf{E} \mid (\mathbf{E}) \mid \mathbf{ID} \mid \mathbf{INT} \mid \mathbf{REAL}$$