Capítulo I : Noções Gerais

1.1 Conceitos Básicos de Linguagens Formais

Alfabeto: Qualquer conjunto finito, não vazio, de **símbolos** (ou **letras**). Um alfabeto é geralmente designado por Σ.

Exemplos:

$\Sigma = \{0, 1\}$, o alfabeto binário.

$\Sigma = \{a, b, c, ..., z\}$, o conjunto das letras minúsculas.

Palavra ou **cadeia**, é qualquer sequência finita de símbolos de um alfabeto Σ. A **palavra vazia** contém zero ocorrências de símbolos e é habitualmente denotada por ε.
Designa-se por $|w|$ o **comprimento** da palavra w e $|\varepsilon| = 0$.

Potências de um alfabeto: Denotamos por Σ^k o conjunto de todas as palavras de comprimento k, sobre um alfabeto Σ.

Exemplos, para o alfabeto $\Sigma = \{0, 1\}$:

$\Sigma^0 = \{\varepsilon\}$ (contém só a palavra vazia)

$\Sigma^1 = \{0, 1\}$ (não confundir com Σ)

$\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

O conjunto de <u>todas</u> as palavras sobre o alfabeto Σ denota-se habitualmente por Σ^*,

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$$

excluindo a palavra vazia obtemos Σ^+, ou seja,

$$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$$

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$$

Para o alfabeto $\Sigma = \{0, 1\}$:

$\{0, 1\}^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, ...\}$

Concatenação: Justapondo duas palavras *v* e *w* obtemos *vw*. A concatenação de palavras é **associativa** mas, não necessariamente, comutativa. A **palavra vazia** é o **elemento identidade** da operação, isto é, $w\varepsilon = \varepsilon w = w$, para qualquer palavra *w*.
O comprimento é **aditivo** relativamente à concatenação, $|vw| = |v| + |w|$.

Duas palavras são **iguais** quando uma for a cópia, letra a letra, da outra.
Diz-se que a palavra *v* é **parte** da palavra *w* quando existirem palavras *x* e *y*, tais que $w = xvy$.

Para qualquer número natural <u>n</u> e qualquer palavra *w*, designamos por w^n a **potência** de ordem <u>n</u> ou a **concatenação iterada** <u>n</u> vezes de *w*.

Para o alfabeto $\Sigma = \{a, b, c, ..., z\}$, seja $w = ab$

$w^1 = ab$, $w^2 = abab$, $w^3 = ababab$, …

Note-se que:
$(ab)^3 = ababab$ não é o mesmo que $ab^3 = abbb$

Por convenção, $w^0 = \varepsilon$ para qualquer palavra.

Designa-se por w^R a palavra **inversa** de w,
ou seja, se seja $w = ab$ então $w^R = ba$.
Naturalmente que $(w^R)^R = w$
e que $(w^R)^n = (w^n)^R$ para qualquer n.

Linguagem sobre um alfabeto Σ é um conjunto de palavras de Σ^*.

$$L \subseteq \Sigma^*$$

Naturalmente que L pode não incluir todas as palavras de Σ^*,
nem sequer as suas palavras incluírem todos os símbolos de Σ.

A **linguagem vazia** é a que não contém palavras.
Designa-se por $\emptyset$ e não é a mesma que a linguagem $\{\varepsilon\}$.

A **linguagem completa** é todo o conjunto Σ^*.

Exemplos de Linguagens:

- Português
- C (o conjunto de todos os programas em C, sintacticamente correctos)

Para o alfabeto $\Sigma = \{0, 1\}$:

- $\{w \mid w$ contém o mesmo número de 0's e 1's$\}$
- $\{w \mid w$ representa um número primo em binário$\}$
- $\{w \mid w = xx^R\}$

  todos os palíndromos pares
- $\{0^n1^n \mid n \geq 0\}$

  todas a palavras formadas por sequências de 0's, seguidas de um número igual de 1's, isto é,

  $\{\varepsilon, 01, 0011, 000111, ...\}$
- $\{0^n1^n \mid n \geq 1\}$

  $\{01, 0011, 000111, ...\}$
- $\{0^i1^j \mid 0 \leq i \leq j\}$

  todas a palavras formadas por sequências de 0's, seguidas de um número igual ou superior de 1's.

No contexto da Teoria dos Autómatos e Linguagens Formais, define-se:

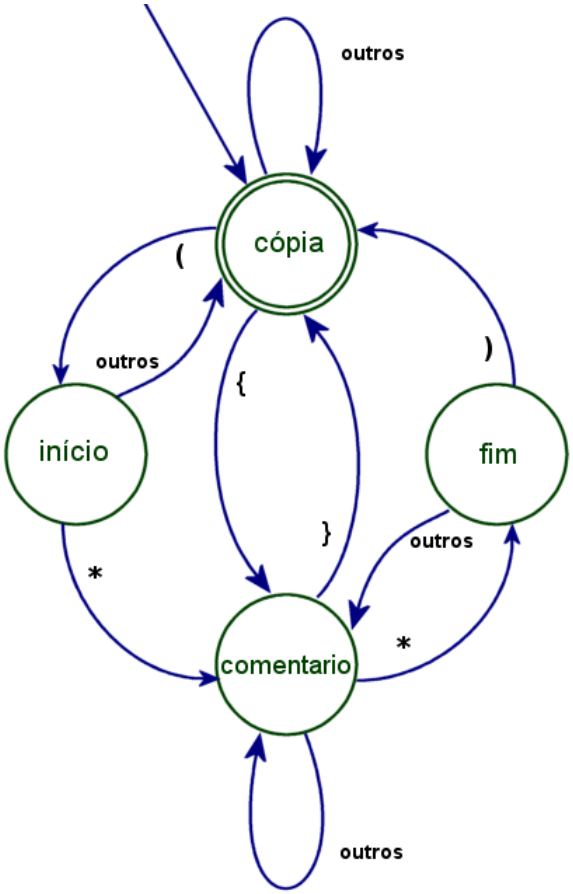
Problema: Dados um alfabeto Σ e uma linguagem L sobre Σ, **decidir** se uma dada palavra $w \in \Sigma^*$ pertence, ou não, a L.

1.2 Uma introdução informal à noção de Autómato

Um exemplo em Pascal: O Ficheiro origem contém um programa Pascal. Pretendemos **remover todos os comentários** e registar a nova versão do programa no Ficheiro destino. Os comentários podem estar nas duas formas possíveis, **(* … *)** ou **{ … }**.

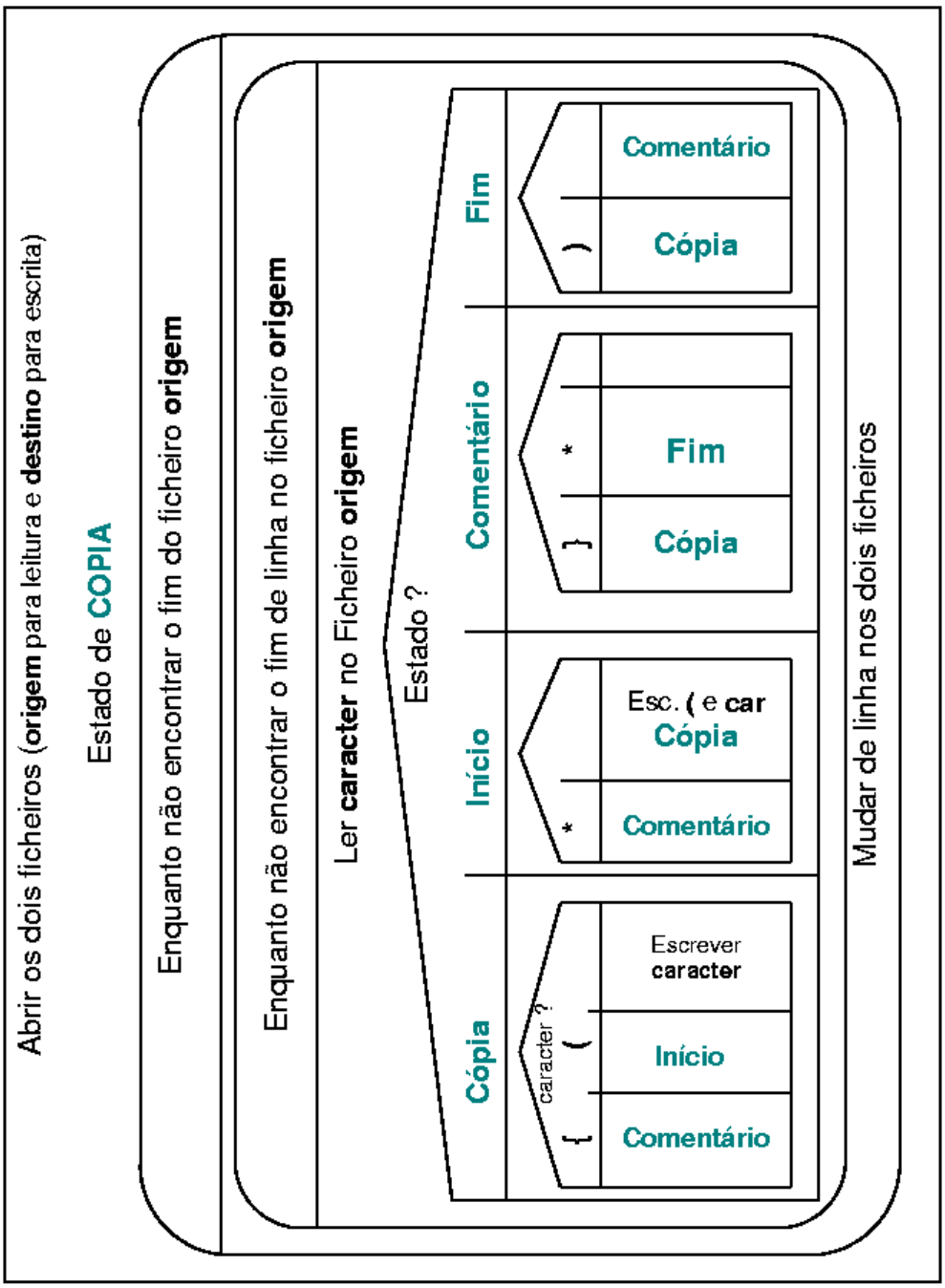
Resolução por um Autómato:

Consideremos um Autómato com quatro Estados: **Cópia**, **Comentário**, **Início** e **Fim**. A leitura de cada caracter provoca uma **Transição de Estado** e também uma operação associada.



Os comentários estão **Sintacticamente Correctos** se e só se, partindo do Estado de **Cópia**, for antingido o mesmo Estado após a leitura do último caracter.

Diagrama de Estrutura do Algoritmo:



Programa Pascal:

```
program DesComentar (origem, destino);
var  origem, destino : text;
     c : char;
     estado : (copia, inicio, comentario, fim);

begin reset(origem);
      rewrite(destino);

      estado:= copia;

      while  not eof(origem) do
            begin while  not eoln(origem) do

                        begin read(origem, c);
                              case estado of
                                    copia : case  c of
                                                '{' : estado:= comentario;
                                                '(' : estado:= inicio
                                          otherwise write(destino, c)
                                          end;
                                    inicio : if c = '*'
                                          then estado:= comentario
                                          else  begin write(destino, '(', c);
                                                      estado:= copia
                                                end;
                                    comentario : case  c of
                                                      '}' : estado:= copia;
                                                      '*' : estado:= fim
                                                otherwise (* nada *);
                                                end;
                                    fim :  if c = ')'
                                          then  estado:= copia
                                          else  estado:= comentario
                              end; (* Fim do case *)
                        end; (* Fim de linha *)

                  readln(origem);
                  writeln(destino)
            end (* Fim do texto *)
end. (* Fim do programa *)
```

1.3 O Princípio da Indução Matemática

Estabelecer a veracidade de uma Proposição P(X) definida sobre os elementos de um conjunto numerável X, em **duas** etapas:

1. **Caso Base**: Provar directamente para um, ou vários, elementos "pequenos" de X;
2. **Passo Indutivo**: Assumindo que P é verdadeira para todos os elementos "menores" que X e, utilizando esse facto, provar P(X).

Exemplo 1:

Definição Indutiva da **potência** de ordem n de uma palavra:

$$w^n = \begin{cases} \varepsilon & \text{se } n = 0 \\ ww^{n-1} & \text{se } n > 0 \end{cases}$$

Exemplo 2:

Definição Indutiva de palavra **inversa**:

$$w^R = \begin{cases} w & \text{se } |w| = 0 \\ a(v^R) & \text{se } w = va, \text{ com } a \in \Sigma, v \in \Sigma^* \end{cases}$$

para w = abc, teremos:

$$\begin{aligned}(abc)^R &= ((ab)c)^R \\ &= c(ab)^R \\ &= c((a)b)^R \\ &= c(b(a)^R) \\ &= c(b((\varepsilon)a)^R) \\ &= c(b(a(\varepsilon)^R)) \\ &= c(b(a(\varepsilon))) \\ &= cba\end{aligned}$$

Exemplo 3:

Mostrar que, $\forall\, x, y \in \Sigma^*$, $(xy)^R = y^R x^R$.

Demonstração por Indução sobre $|y|$:

1. **Caso Base**:

   Se $|y| = 0$ então y = ε

   $(xy)^R = (x\varepsilon)^R = x^R = \varepsilon\, x^R = y^R x^R$.

2. **Passo Indutivo:**

   Hipótese de Indução: $\forall\, y$, $|y| \leq n$, $(xy)^R = y^R x^R$.

   Tese: $\forall\, y$, $|y| = n+1$, $(xy)^R = y^R x^R$.

Seja $y = wa$, com $|w| = n$ e $a \in \Sigma$

$(xy)^R$	=	$(x(wa))^R$	definição de y
	=	$((xw)a)^R$	associatividade da concatenação
	=	$a(xw)^R$	definição de palavra inversa
	=	$a(w^R x^R)$	hipótese de indução
	=	$(aw^R)\, x^R$	associatividade da concatenação
	=	$(wa)^R\, x^R$	definição de palavra inversa
	=	$y^R\, x^R$	definição de y

Exemplo 4:

Definição Indutiva de **Expressão Aritmética**.

Consideremos o alfabeto:

$$\Sigma = \{a, b, c, ..., z\} \cup \mathbb{R} \cup \{+, *, (,)\}$$

Uma palavra de Σ^* é uma **Expressão**, se e só se:

1. **Caso Base**: Qualquer número (real) ou qualquer variável (letra) é uma Expressão.
2. **Passo Indutivo**: Se E e F são Expressões, então também são Expressões E+F, E∗F e (E).

Assim, são Expressões as palavras:

2	x
x+2	(x+2)
3∗(x+2)	...

Exemplo 5:

Mostrar que em toda a Expressão é igual o número de parêntesis esquerdos e direitos, isto é,

Para qualquer Expressão G, construída segundo a definição anterior, provar:

P(G) = {G possui o mesmo número de parêntesis esquerdos e direitos}

Demonstração por Indução:

1. **Caso Base**: Se G for apenas um número ou uma variável, o número de parêntesis esquerdos e direitos é zero.
2. **Passo Indutivo**: Senão, G foi construída segundo alguma das 3 regras:
   a) G = E + F
   b) G = E ∗ F
   c) G = (E)

Hipótese de Indução: P(E) ∧ P(F)
Seja m = número de parêntesis esquerdos de E
= número de parêntesis direitos de E
e seja n = número de parêntesis esquerdos de F
= número de parêntesis direitos de F

Tese: P(G)

Demonstração:

a) Se G = E + F então
m + n = número de parêntesis esquerdos de G
= número de parêntesis direitos de G

b) O caso G = E ∗ F é análogo

c) Se G = (E) então
m + 1 = número de parêntesis esquerdos de G
= número de parêntesis direitos de G

Exemplo 6: Conversão **Binário ↦ Decimal**

Consideremos o alfabeto binário $\Sigma = \{0, 1\}$.
O conjunto das palavras de Σ^+ pode ser interpretado como a **Representação Binária** dos números naturais, como por exemplo:

$1001 \mapsto 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9$

$11001010 \mapsto 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^3 + 1 \times 2^1 = 202$

Definição: $\forall\, w \in \{0, 1\}^+$

$$w = b_k \ldots b_1\, b_0 \mapsto \sum_{i=0}^{k} b_i\, 2^i = n \in \mathbb{N}$$

Teorema: (Conversão Binário ↦ Decimal **da esquerda para a direita**)

1. **Caso Base**:

   $0 \in \{0, 1\} \mapsto 0 \in \mathbb{N}$

   $1 \in \{0, 1\} \mapsto 1 \in \mathbb{N}$

2. **Passo Indutivo:**

   Se $w \in \{0, 1\}^+ \mapsto n \in \mathbb{N}$ então $w0 \mapsto 2 \times n$

   $w1 \mapsto 2 \times n + 1$

Exemplo: $11001 \mapsto 25$

1	1	= 1
1	$2 \times 1 + 1$	= 3
0	2×3	= 6
0	2×6	= 12
1	$2 \times 12 + 1$	= 25

Demonstração:

1. **Caso Base**:

$0 \in \{0, 1\} \mapsto 0 \times 2^0 = \mathbf{0}$
$1 \in \{0, 1\} \mapsto 1 \times 2^0 = \mathbf{1}$

2. **Passo Indutivo:**

Se $w = b_k \ldots b_1\, b_0 \mapsto \sum_{i=0}^{k} b_i\, 2^i = n$

então $\boldsymbol{w0} = b_k \ldots b_1\, b_0\, 0 \mapsto \sum_{i=0}^{k} b_i\, 2^{i+1} + 0 \times 2^0 =$

$$2 \times \sum_{i=0}^{k} b_i\, 2^i = \mathbf{2 \times n}$$

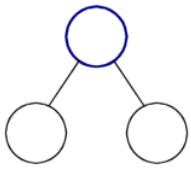
$\boldsymbol{w1} = b_k \ldots b_1\, b_0\, 1 \mapsto \sum_{i=0}^{k} b_i\, 2^{i+1} + 1 \times 2^0 =$

$$2 \times \sum_{i=0}^{k} b_i\, 2^i + 1 = \mathbf{2 \times n + 1}$$

Exemplo 7:

Uma **Árvore Binária** é um conjunto finito de nós, tal que:

- ou é o conjunto vazio
- ou é constituída por uma raiz e duas Árvores Binárias disjuntas



A formalização Indutiva da definição de uma Estrutura de Dados é um instrumento muito útil para a construção de algoritmos recorrentes.

Seja:

```
type arvore_binaria  = ^vertice;
     vertice         = record  elemento : tipo;
                               esquerda, direita : arvore_binaria
                       end;
```

```
(* Calcular o número de vértices de uma Árvore Binária *)

function   numvert(T : arvore_binaria) : integer;

           begin  if T = nil
                  then   numvert := 0
                  else   numvert := 1 + numvert(T^.esquerda) +
                                       numvert(T^.direita)
           end (* numvert *);
```

```
(* Calcular a altura de uma Árvore Binária *)

function   altura(T : arvore_binaria) : integer;
           var e, d : integer;

           begin  if T = nil
                  then   altura := 0
                  else   begin  e := altura(T^.esquerda);
                                d := altura(T^.direita);
                                if e > d
                                then   altura := e + 1
                                else   altura := d + 1
                         end
           end (* altura *);
```

```
(* Verificar se duas Árvores Binárias são iguais *)

function   iguais(R, T : arvore_binaria) : boolean;
var i : boolean;

begin  i := (R = nil) and (T = nil);
       if (R <> nil) and (T <> nil)
       then   begin  i := R^.elemento = T^.elemento;
                     if i
                     then  begin  i := iguais(R^esquerda,T^esquerda);
                                  if i
                                  then i := iguais(R^direita, T^direita)
                           end
              end;
       iguais := i
end (* iguais *);
```

Uma árvore binária diz-se **completa** (ou **total**) se, em cada nível k ($k \geq 1$) existirem 2^k vértices.

Exemplo 8:

Uma árvore binária completa com n folhas tem **2n-1 vértices**.

- **Especificação**: P(T) ≡ {Se T for uma árvore binária com n folhas, então T tem 2n-1 vértices}.
- **Indução**: sobre #(T), o número de vértices de T.

Demonstração por Indução:

1. **Caso Base**: Uma árvore binária completa com uma só folha (n=1) possui só um vértice, portanto #(T) = 2×1-1 = 1;

2. **Passo Indutivo:**

   Hipótese de Indução: P(U) é verdadeira para todas as árvores binárias completas U, tais que #(U) < #(T), em particular para todas as sub-árvores de T.

   Tese: P(T)

   Demonstração:

   Se T tem mais de um vértice, então tem uma raiz e duas sub-árvores U e V. Sejam u e v o número de folhas de U e V, respectivamente, e t o número de folhas de T. Então t = u+v.
   Por hipótese de indução, U e V têm 2u-1 e 2v-1 vértices.
   Então, o número de vértices de T,

   $= 1 + (2u - 1) + (2v - 1)$
   $= 2 (u + v) - 1$
   $= 2t - 1$ □