Fundamentos da Computação

1ª edição: Outubro/2001

RICARDO J. MACHADO

Email: rmac@dsi.uminho.pt
URL: http://www.dsi.uminho.pt/~rmac

Universidade do Minho

Departamento de Sistemas de Informação

Sumário

1. Bases Matemáticas (1/34)

- Lógica -

- **Proposição**
  - **é uma frase que pode ser apenas ou verdadeira ou falsa**
    - **Dez é menor do que sete.**
      (proposição falsa)
    - **Como vai você?**
      (pergunta, logo, não é uma proposição)
    - **Ela é muito talentosa!**
      (frase com uma variável, logo, não é uma proposição)
    - **Existem formas de vida noutros planetas.**
      (proposição, mesmo que não saibamos qual a resposta)
- **Valores lógicos ou Booleanos**
  - **os valores que pode tomar uma proposição**
    - **verdadeiro representa-se por "V", "T" ou "1"**
    - **falso representa-se por "F" ou "0"**
    - **nota: uma proposição sempre verdadeira é designada de *tautologia***
      uma proposição sempre falsa é designada de *contradição*

1. Bases Matemáticas (2/34)

- Lógica -

- **Operadores lógicos ou Booleanos**
  - **negação de uma proposição (*não*)**
    - **representa-se por "A′ ", "Ā", "¬ A", "~ A" ou "*not* A"**
    - **lê-se "não A", "A é falso" ou "A não é verdade"**
    - **não quer dizer que "¬ A" tenha sempre um valor lógico falso, mas sim que o valor lógico de "¬ A" é o contrário de "A"**
    - **tabela de verdade de F(A) = ¬ A**

A	F
0	1
1	0

    - **exemplo:**
      - **se A = "Amanhã vai chover.", então ¬ A = "Amanhã não vai chover."**

1. Bases Matemáticas (3/34)

- Lógica -

- **Operadores lógicos ou Booleanos**
  - **conjunção de duas proposições (*e*)**
    - **representa-se por "A ∧ B", "A × B", "A · B", "A ∗ B" ou "A *and* B"**
    - **lê-se "A *e* B"**
    - **tabela de verdade de de F(A, B) = A ∧ B**

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

    - **exemplo:**
      - **"Os elefantes são grandes e as bolas são redondas."**

1. Bases Matemáticas (4/34)

- Lógica -

- **Operadores lógicos ou Booleanos**
  - **disjunção inclusiva de duas proposições (*ou*)**
    - **representa-se por "A ∨ B", "A + B" ou "A *or* B"**
    - **lê-se "A *ou* B"**
    - **tabela de verdade de F(A, B) = A ∨ B**

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

    - **exemplo:**
      - **"A Joana detesta manteiga ou adora nata."**

1. Bases Matemáticas (5/34)

- Lógica -

- **Operadores lógicos ou Booleanos**
  - **disjunção exclusiva de duas proposições (*ou ou*)**
    - **representa-se por "A ⊕ B" ou "A *xor* B"**
    - **lê-se "*ou* A *ou* B"**
    - **nota: A ⊕ B = (¬ A ∧ B) ∨ (A ∧ ¬ B)**
    - **tabela de verdade de F(A, B) = A ⊕ B**

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

    - **exemplo:**
      - **"A Joana ou detesta manteiga ou adora nata."**

1. Bases Matemáticas (6/34)

- Lógica -

- **Operadores lógicos ou Booleanos**
  - **implicação entre duas proposições (*se então*)**
    - **representa-se por "A ⇒ B "**
    - **lê-se "A *implica* B" ou "*se* A *então* B"**
    - **nota: A ⇒ B = ¬ B ∨ A**
    - **exemplo:**
      - **"Se a chuva continuar, então o rio vai transbordar."**
  - **equivalência entre duas proposições (*se e só se*)**
    - **representa-se por "A ⇔ B"**
    - **lê-se "A *se e só se* B"**
    - **nota: A ⇔ B = (A ⇒ B) ∧ (B ⇒ A)**
    - **exemplo:**
      - **"O rio vai transbordar se e só se a chuva continuar."**

1. Bases Matemáticas (7/34)

- Lógica -

- **Prioridades dos operadores**

  1ª: proposições dentro de parêntesis, dos mais internos para os mais externos
  2ª: negação ¬
  3ª: conjunção ∧
  4ª: disjunção inclusiva ∨
  5ª: disjunção exclusiva ⊕
  6ª: implicação ⇒
  7ª: equivalência ⇔

  - **exemplo:**
    ¬ A ⇒ B ∧ A ∨ C ⇔ A ⇒ D ⊕ B
    =
    ((¬ A) ⇒ ((B ∧ A) ∨ C)) ⇔ (A ⇒ (D ⊕ B))

1. Bases Matemáticas (8/34)

- Lógica -

- **Propriedades (Álgebra de Boole)**

  - **envolvimento: ¬ ¬ A = A**
  - **elemento neutro: A ∨ F = A A ∧ T = A**
  - **elemento absorvente: A ∨ T = T A ∧ F = F**
  - **idempotência: A ∨ A = A A ∧ A = A**
  - **terceiro excluído: A ∨ ¬ A = T A ∧ ¬ A = F**
  - **comutativa: A ∨ B = B ∨ A A ∧ B = B ∧ A**
  - **associativa: A ∨ B ∨ C = A ∨ (B ∨ C) = (A ∨ B) ∨ C**
  - **A ∧ B ∧ C = A ∧ (B ∧ C) = (A ∧ B) ∧ C**
  - **distributiva: A ∨ B ∧ C = (A ∨ B) ∧ (A ∨ C)**
  - **A ∧ (B ∨ C) = (A ∧ B) ∨ (A ∧ C)**
  - **absorção: A ∨ A ∧ B = A A ∧ (A ∨ B) = A**
  - **A ∨ ¬ A ∧ B = A ∨ B A ∧ (¬ A ∨ B) = A ∧ B**
  - **terceiro incluído: A ∧ B ∨ B ∧ C ∨ ¬ A ∧ C = A ∧ B ∨ ¬ A ∧ C**
  - **(A ∨ B) ∧ (B ∨ C) ∧ (¬ A ∨ C) = (A ∨ B) ∧ (B ∨ C) ∧ (¬ A ∨ C)**
  - **De Morgan: ¬ (A ∨ B) = ¬ A ∧ ¬ B ¬ (A ∧ B) = ¬ A ∨ ¬ B**

1. Bases Matemáticas (9/34)

- Lógica -

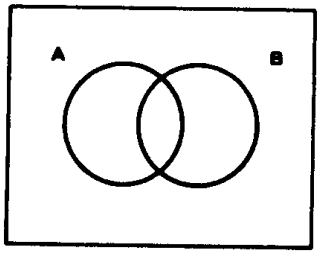
- **Demonstração de propriedades (tabela de verdade)**
  - $\neg (A \wedge B) = \neg A \vee \neg B$

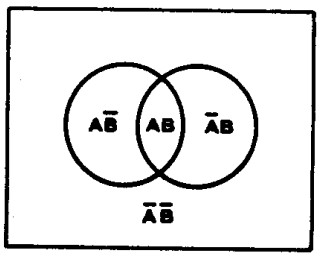
VARIÁVEIS		VALORES INTERMÉDIOS			VERIFICAÇÃO	
					1.º Membro	2.º Membro
A	B	$A \cdot B$	$\overline{A}$	$\overline{B}$	$\overline{A \cdot B}$	$\overline{A}+\overline{B}$
0	0	0	1	1	1	1
0	1	0	1	0	1	1
1	0	0	0	1	1	1
1	1	1	0	0	0	0

1. Bases Matemáticas (10/34)

- Lógica -

- **Demonstração de propriedades (diagrama de Venn)**





1. Bases Matemáticas (11/34)

- Lógica -

- **Quantificadores**
  - **universal**
    - **representa-se por "∀"**
    - **lê-se "*para todo*", "*para todos*" ou "*para qualquer*"**
  - **existencial**
    - **representa-se por "∃"**
    - **lê-se "*existe um*", "*para pelo menos um*" ou "*para algum*"**
    - **exemplo: (∀ n ∈ ℕ) (∃ m ∈ ℕ) [m > n]**

1. Bases Matemáticas (12/34)

- Conjuntos -

- **Conjunto**
  - **definição**
    - **uma colecção não ordenada de objectos não repetidos**
    - **normalmente, todos os elementos de um conjunto possuem uma mesma propriedade, para além de pertencerem ao mesmo conjunto**
    - **um conjunto vazio não possui nenhum elemento e representa-se por "∅"**
  - **representação**
    - **o conjunto representa-se com maiúsculas "A"**
    - **um elemento representa-se com minúsculas "a"**
      - **exemplo: a ∈ A**
    - **a enumeração dos elementos do conjunto exige a utilização de chavetas**
      - **exemplo: S = {1, 2, 3}**

1. Bases Matemáticas (13/34)

- Conjuntos -

- **Operadores relacionais**
  - **pertença** $\in$
    - **exemplo:** $\{1, 2\} \in \{ \{1\}, \{1, 2\}, \{1, 2, 3\} \}$
  - **não pertença** $\notin$
    - **exemplo:** $1 \notin \{ \{1\}, \{1, 2\}, \{1, 2, 3\} \}$
  - **contido** $\subset$
    - **exemplo:** $\{0\} \subset \mathbf{N}$
  - **não contido** $\not\subset$
    - **exemplo:** $\{-1\} \not\subset \mathbf{N}$
  - **contém** $\supset$
    - **exemplo:** $\mathbf{N} \supset \{1, 3, 5\}$

15

1. Bases Matemáticas (14/34)

- Conjuntos -

- **Outros exemplos**
  - **um elemento é diferente de um conjunto formado por esse elemento, ou seja:** $1 \neq \{1\}$
  - **o conjunto vazio é diferente de um conjunto que contém somente o elemento conjunto vazio, ou seja:** $\varnothing \neq \{\varnothing\}$
  - **num conjunto não há repetições a sua ordem de surgimento os elementos é irrelevante:** $\{1, 2, 3, 3, 2, 1\} = \{1, 2, 3\} = \{2, 1, 3\}$ **...**
  - **um conjunto pode ser descrito por extensão ou por compreensão** $\mathbf{N} = \{0, 1, 2, \ldots\}$ **ou** $\mathbf{N} = \{n \mid n \in \mathbf{Z} \wedge n \geq 0\}$

16

1. Bases Matemáticas (15/34)

- Conjuntos -

- **Operações em conjuntos**
  - **intersecção** $\cap$

    $A \cap B = \{x \mid x \in A \wedge x \in B\}$

    - **nota:** $A \cap B = \varnothing \Rightarrow$ **A e B *disjuntos***
  - **união** $\cup$

    $A \cup B = \{x \mid x \in A \vee x \in B\}$
  - **complemento** $/$ $'$

    $/A = A' = \{x \mid x \in U \wedge x \notin A\}$
  - **diferença** $-$

    $A - B = \{x \mid x \in A \wedge x \notin B\} = \{x \mid x \in A \wedge x \in /B\} = A \cap /B$

    - **nota:** $A - \varnothing = A$

1. Bases Matemáticas (16/34)

- Conjuntos -

- **Subconjunto de um conjunto**
  - **A é um subconjunto de B se e só se todos os elementos pertencentes a A forem elementos de B, ou seja:**

    $A \subseteq B \Leftrightarrow (\forall x)\ (x \in A \Rightarrow x \in B)$

    - **nota:** $A \subseteq B \Rightarrow (A \cap B = A) \wedge (A \cup B = B) \wedge (A - B = \varnothing)$
  - **A é um subconjunto próprio de B se e só se existe pelo menos um elemento de B que não é elemento de A, ou seja:**

    $A \subset B \Leftrightarrow (\exists y)\ (y \in B \wedge y \notin A) \wedge (\forall x \neq y)\ (x \in A \Rightarrow x \in B)$
- **Igualdade entre conjuntos**
  - **dois conjuntos são iguais se e só se contêm os mesmos elementos, ou seja:**

    $A = B \Leftrightarrow (\forall x)\ [(x \in A \Rightarrow x \in B) \wedge (x \in B \Rightarrow x \in A)] \Leftrightarrow (A \subseteq B) \wedge (B \subseteq A)$

1. Bases Matemáticas (17/34)

- Conjuntos -

- **Conjunto potência**
  - **o conjunto potência de A (ou conjunto de partes de A) é composto por todos os subconjuntos de A, pelo que conterá, pelo menos, ∅ e o próprio A, ou seja:** $P(A) = 2^A = \{x \mid x \subseteq A\}$
    - **exemplo:** $A = \{a, b\}$ $P(A) = \{\emptyset, \{a\}, \{b\}, A\}$
- **Cardinalidade e tamanho de um conjunto**
  - **o número de elementos distintos de um conjunto finito A denota-se "#(A)" e designa-se "*tamanho* de A"**
  - **quando o conjunto A é infinito, #(A) designa-se "*cardinalidade* de A"**
    - **exemplos:** $A = \{1, 1, 2, 3\}$ $\#(A) = 3$
      $\#(\emptyset) = 0$
      $\#(Z) = \#(Q) < \#(R)$
    - **nota:** $\#(2^A) = 2^{\#(A)}$

1. Bases Matemáticas (18/34)

- Conjuntos -

- **Partição de um conjunto**
  - **é uma família de subconjuntos de A, não vazios e disjuntos par a par, cuja união é igual a A, ou seja:**

    $\Pi(A) = \{S_1, S_2, \ldots\}$, com $S_i \cap S_j = \emptyset$ e $US_i = A$
- **Túplo ordenado**
  - **num túplo ordenado "(x, y)", "x" é a primeira componente e "y" é a segunda**
  - **quando o túplo possui unicamente 2 componentes designa-se de "par"**
  - **num par ordenado a ordem das componentes é relevante**
    - **exemplos:** (a, b) (a, c, d) (r, f, g, a)
      $(a, b) \neq (b, a)$ se $a \neq b$

1. Bases Matemáticas (19/34)

- Conjuntos -

- **Produto cartesiano**
  - **o produto cartesiano de dois conjuntos A e B é o conjuntos de todos os pares ordenados cujas primeiras componentes pertençam a A e as segundas a B, ou seja:**

    $A \times B = \{(x, y) \mid x \in A \wedge y \in B\}$

    - $A \times A = A^2$
    - A^n **é o conjunto de todas as** *n*-**uplas** $(x_1, x_2, \ldots, x_n)$ **de elementos de A**
    - $\mathbb{N} \times \mathbb{N} = \mathbb{N}^2$ **é o conjunto de todos os pares de inteiros**
    - $\mathbb{N} \times \{$**vermelho, amarelo, verde**$\}$ **é o conjunto de todos os pares cuja primeira componente é um número natural e a segunda uma cor (ou vermelho, ou amarelo ou verde)**

1. Bases Matemáticas (20/34)

- Conjuntos -

- **Teoremas**
  - **envolvimento:** $// A = A$
  - **identidade:** $A \cup \varnothing = A \quad A \cap U = A$
  - **absorção:** $A \cup U = U \quad A \cap \varnothing = \varnothing$
  - **idempotência:** $A \cup A = A \quad A \cap A = A$
  - **complemento:** $A \cup /A = U \quad A \cap /A = \varnothing$
  - **comutativa:** $A \cup B = B \cup A \quad A \cap B = B \cap A$
  - **associativa:** $A \cup B \cup C = A \cup (B \cup C) = (A \cup B) \cup C$
  - $A \cap B \cap C = A \cap (B \cap C) = (A \cap B) \cap C$
  - **distributiva:** $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
  - $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$

  - $(\forall A)\ (\varnothing \subseteq A)$
  - $(A \subseteq B) \wedge (B \subseteq U) \Rightarrow A \subseteq U$
  - $(A \subseteq B) \wedge (B \subset U) \Rightarrow A \subset U$
  - $(A \subset B) \wedge (B \subset U) \Rightarrow A \subset U$

1. Bases Matemáticas (21/34)

- Sequências -

- **Sequência**
  - **definição**
    - **uma lista de objectos enumerados segundo uma ordenação**
    - **uma lista pode ser definida**
      - **por enumeração** $t = \langle a, b, c \rangle$
      - **recursivamente** $l[1] = 2 \wedge l[n] = 2 \times l[n-1]$ **para** $n \geq 2$
  - **representação**
    - **a sequência representa-se com os sinais de maior e menor "<…>"**
    - **"lista[*k*]" designa o *k*-ésimo objecto da sequência**
    - **exemplos:** $\langle a, b \rangle \neq \langle b, a \rangle$
      $\langle a, b \rangle \neq \langle a, b, b \rangle$
      $l = \langle a, b, c \rangle \quad l[2] = b$

1. Bases Matemáticas (22/34)

- Sequências -

- **Operadores**
  - **comprimento** ***len***
    - **exemplos:** $\text{len}(\langle\rangle) = 0 \quad \text{len}(\langle b, a, t \rangle) = 3$
  - **índices** ***inds***
    - **exemplos:** $\text{inds}(\langle\rangle) = \varnothing \quad \text{inds}(l) = \{1, \ldots, \text{len}(l)\}$
  - **elementos** ***elems***
    - $\text{elems}(l) = \{l[i] \mid i \in \text{inds}(l)\}$
    - **exemplo:** $\text{elems}(\langle 1, 2, 1 \rangle) = \{1, 2\}$
  - **concatenação** ^
    - **exemplo:** $\langle a, b, c \rangle \hat{} \langle a, b \rangle = \langle a, b, c, a, b \rangle$

1. Bases Matemáticas (23/34)

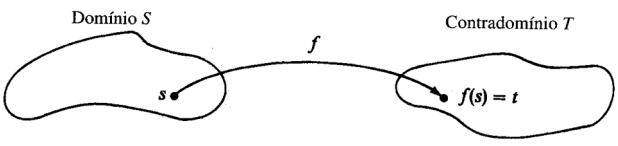
- Sequências -

- **Operadores**
  - **inserção *cons***
    - **exemplo: cons (7, <1, 2, 1>) = <7, 1, 2, 1>**
  - **cabeça *head***
    - **head (cons (a, l)) = a**
    - **exemplo: l = <1, 5, 3, 9> head (l) = 1**
  - **cauda *tail***
    - **tail (cons (a, l)) = l**
    - **exemplo: l = <1, 5, 3, 9> tail (l) = <5, 3, 9>**

1. Bases Matemáticas (24/34)

- Funções -

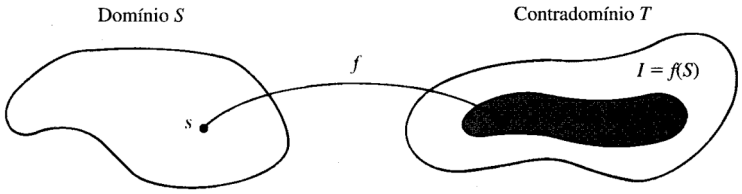
- **Função**
  - **definição**
    - **sejam *S* e *T* conjuntos**
    - **uma *função* (ou *aplicação*) *f* de *S* em *T*, "*f*: S → T", é um subconjunto de *S* × *T*, onde cada elemento de *S* aparece exactamente uma única vez como primeiro componente de um par ordenado "*(s, t)*"**
    - ***S* é o *domínio* e *T* é o *contradomínio* da função**
    - **se *(s, t)* pertence à função, então**
      - **a variável dependente *t* é a *imagem*, "t = *f* (s)", de *s* por *f***
      - **a variável independente *s* é o *objecto* de *t* por *f***



1. Bases Matemáticas (25/34)

- Funções -

- **Propriedades**
  - **sobrejectividade**
    - **o conjunto I = {f (s) | s ∈ S}, ou I = f (S), de todas as imagens de f: S → T é o *conjunto imagem* da função f**
    - **genericamente, I ⊆ T**
    - **quando I = T, o conjunto imagem é igual ao contradomínio e função f é chamada sobrejectiva**



1. Bases Matemáticas (26/34)

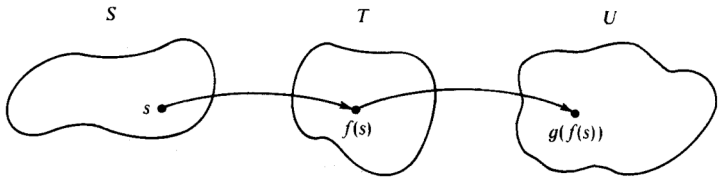
- Funções -

- **Propriedades**
  - **injectividade**
    - **uma função f: S → T é *injectiva* (ou *um-para-um*), se nenhum dos elementos de T for imagem por f de dois, ou mais, elementos distintos de *S***
  - **bijectividade**
    - **uma função f: S → T é *bijectiva*, se for, simultaneamente, sobrejectiva e injectiva**
  - **conjuntos equivalentes**
    - **o conjunto S é equivalente ao conjunto *T* se existir uma bijecção f: S → T**
    - **dois conjuntos equivalentes possuem a mesma cardinalidade**
  - **teorema de Cantor**
    - **para qualquer conjunto A, A e P(A) não são equivalentes**

1. Bases Matemáticas (27/34)

- Funções -

- **Propriedades**
  - **composição de funções**
    - **seja f: S → T e g: T → U**
    - **a função composta $g \circ f$ é uma função de *S* em *U* definida por $(g \circ f)(s) = g(f(s))$**



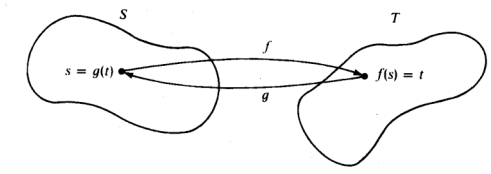
  - **teorema de composição de bijecções**
    - **a composição de duas bijecções é uma bijecção**

29

1. Bases Matemáticas (28/34)

- Funções -

- **Propriedades**
  - **função inversa**
    - **seja f uma função f: S → T**
    - **se existir uma função g: T → S, tal que $g \circ f = i_S$ e $f \circ g = i_T$, então g é chamada de função inversa de f e denotada uma função de f^{-1}**



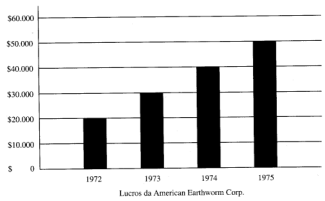
  - **teorema de sobre bijecções e funções inversas**
    - **seja f: S → T**
    - **f é uma bijecção se e só se f^{-1} existir**

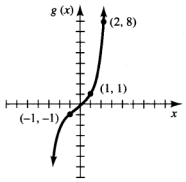
30

1. Bases Matemáticas (29/34)

- Funções -

- **Representação/descrição**





$$f = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 4 & 2 & 3 & 5 & 1 \end{pmatrix}$$

$$g(n) = \frac{\sum_{k=1}^{n}(4k-2)}{2}$$

"Seja S o conjunto de todas as cadeias de caracteres de tamanho fixo. Então a associação que relaciona a cada cadeia o número de caracteres que contém é uma função de domínio S e contradomínio $\mathbb{N}$ (é permitida a 'cadeia vazia', cujo número de caracteres é zero.)"

1. Bases Matemáticas (30/34)

- Funções -

- **Generalidades**
  - **somatório**

$$\sum_{\substack{i=1 \\ P(i)}}^{n} f(i)$$

ex.:

$$\sum_{i=1}^{5} i = 1 + 2 + 3 + 4 + 5 = 15$$

$$\sum_{\substack{i=1 \\ i\ impar}}^{5} i = 1 + 3 + 5 = 9$$

  - **produtório**

$$\prod_{\substack{i=1 \\ P(i)}}^{n} f(i)$$

ex.:

$$\prod_{i=1}^{5} i = 1 \times 2 \times 3 \times 4 \times 5 = 120$$

$$\prod_{\substack{i=1 \\ i\ par}}^{5} i = 2 \times 4 = 8$$

1. Bases Matemáticas (31/34)

- Funções -

- **Generalidades**
  - **logarítmo**

Seja $b \neq 1$ e $x \in \mathbb{R}^+$

$$\log_b x = y \Leftrightarrow b^y = x$$

Propriedades

$$\log_a (xy) = \log_a x + \log_a y$$

$$\log_a x^y = y \log_a x$$

$$\log_a x = \log_b x / \log_b a$$

  - **operador módulo**

Seja $m \geq 0$ e $n > 0$ inteiros

$m \bmod n = m - n \times (m \operatorname{div} n)$ (***div*** é a divisão inteira)

1. Bases Matemáticas (32/34)

- Funções -

- **Generalidades**
  - **valor absoluto**

$x \in \mathbb{R}$

$$|x| = \begin{cases} x \Leftarrow x \geq 0 \\ -x \Leftarrow x < 0 \end{cases}$$

  - **factorial**

Seja $n \in \mathbb{N}^+$

$$n! = \prod_{i=1}^{n} i$$

1. Bases Matemáticas (33/34)

- Funções -

- **Operadores (para funções com S e T discretos)**
  - **imagem ( )**
    - **exemplo: m = [a ↦1, c ↦3, d ↦1] m(d) = 1**
      n = [b ↦4, c ↦5] n(b) = 4
  - **conjunto dos objectos *dom***
    - **exemplo: dom (m) = {a, c, d} dom [ ] = ∅**
  - **conjunto das imagens *ran***
    - **exemplo: ran (m) = {1, 3} ran [ ] = ∅**

1. Bases Matemáticas (34/34)

- Funções -

- **Operadores (para funções com S e T discretos)**
  - **overwrite †**
    - **exemplo: m † n = [a ↦1, b ↦4, c ↦5, d ↦1]**
      n † m = [a ↦1, b ↦4, c ↦3, d ↦1]
  - **restrição |**
    - **exemplo: {a, d, e} | m = [a ↦1, d ↦1]**
  - **remoção -**
    - **exemplo: {a, d, e} - m = [c ↦3]**

exercícios

I - Lógica

1. **Negue a seguinte proposição:**
   "A Joana detesta manteiga ou adora nata."
2. **Construa a tabela de verdade do operador lógico implicação.**
3. **Construa a tabela de verdade do operador lógico equivalência.**
4. **Sejam A, B e C as seguintes proposições:**
   A: As rosas são vermelhas.
   B: As violetas são azuis.
   C: O açúcar é doce.
   Traduza as seguintes proposições compostas para notação simbólica:
   a) As rosas são vermelhas e ou as violetas são azuis ou o açúcar é doce.
   b) Sempre que as violetas são azuis, as rosas são vermelhas e o açúcar é doce.
   c) As rosas são vermelhas apenas se as violetas não forem azuis e se o açúcar for azedo.
   d) As rosas são vermelhas e se o açúcar for azedo, então as violetas não são azuis ou o açúcar é doce.

exercícios

II - Conjuntos

5. **Utilize diagramas de Venn para demonstrar as seguintes equivalências:**
   a) $A \cap (B \cap C) = (A \cap B) \cap C$
   b) $A \cup (B \cup C) = (A \cup B) \cup C$
6. **Utilize transformações para demonstrar as seguintes equivalências:**
   a) $A \cup (B \cup A) = A \cup B$
   b) $A \cup (B \cap C \cap A) = A$
7. **Descreva cada um dos conjuntos:**
   a) $A = \{x \mid x \in \mathbb{N} \wedge (\forall y)(y \in \{2, 3, 4, 5\} \Rightarrow x \geq y)\}$
   b) $B = \{x \mid (\exists y)(\exists z)(y \in \{1, 2\} \wedge z \in \{2, 3\} \wedge x = y + z)\}$
8. **Para $A = \{1, 2, 3\}$, defina P(A).**

exercícios

III - Sequências

9. Sejam l e k duas sequências, com l = <t, c, a> e k = <33, 2, 50, 9, 9>. Complete as seguintes alíneas de modo a transformá-las em proposições verdadeiras:

a) len (k) = ...
b) inds (l) ... inds (k)
c) l[...] = t
d) elems (k) = ...
e) l ^ k = ...
f) cons (a, <cons (t, <t, c, a>)>) = ...
g) head (k) = ...
h) tail (l) = ...

exercícios

III - Funções

10. Sejam *f* e *g* duas funções, com *f* = [1 ↦abc, 2 ↦bcd, 3 ↦cde] e *g* = [5 ↦1, 2 ↦20, 9 ↦bcd]. Complete as seguintes alíneas de modo a transformá-las em proposições verdadeiras:

a) *f* (...) = *g* (...)
b) dom (*f*) = ...
c) ran (*g*) = ...
d) *f* † *g* = ...
e) (*g* † *f*) † *g* = ...
f) dom (*g*) ⏐ *f* = ...
g) ∅ - *g* = ...

2. Modelos de Computação (1/24)

- autómato de estados finitos -

- **Máquinas computacionais**
  - **uma descrição formal de uma máquina de processamento de informação deve ser independente da implementação física (hardware)**
  - **deve existir somente uma descrição do tipo de operações internas realizadas pela máquina quando esta processa várias classes de informação recebida**
  - **a abstracção das características de operação das máquinas de processamento permite obter um modelo que é, genericamente, chamado de *modelo computacional* ou *modelo de computação***

2. Modelos de Computação (2/24)

- autómato de estados finitos -

- **O exemplo da máquina de selos**
  - **uma máquina distribuidora de selos de $20 aceita moedas de**
    - **$5 (designada simbolicamente por "c")**
    - **$10 ("d")**
    - **$20 ("v")**
  - **a máquina dispõe de uma portinhola que, quando activada (aberta) , entrega o selo desejado ao utilizador**
  - **a portinhola abre quando, pelo menos, $20 são inseridos na máquina**
  - **as moedas são inseridas na máquina pelo utilizador segundo uma certa sequência (m_i é a *i*-ésima moeda inserida)**

2. Modelos de Computação (3/24)

- autómato de estados finitos -

- **Algoritmo da máquina de selos**

  1. **ler m_1; se m_1 = c, então ir para 2, senão**
     se m_1 = d, então ir para 3, senão
     se m_1 = v, então ir para 5
  2. **ler m_2; se m_2 = c, então ir para 3, senão**
     se m_2 = d, então ir para 4, senão
     se m_2 = v, então ir para 5
  3. **ler m_3; se m_3 = c, então ir para 4, senão**
     se m_3 = d ou se m_3 = v, então ir para 5
  4. **ler m_4; se m_4 = c ou se m_4 = d ou se m_4 = v, então ir para 5**
  5. **se portinhola fechada, então abrir portinhola (fornecer selo)**

2. Modelos de Computação (4/24)

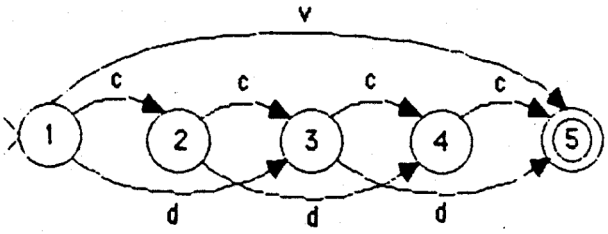
- autómato de estados finitos -

- **Algoritmo da máquina de selos (cont.)**
  - **cada passo do algoritmo representa uma dada configuração interna da máquina e que determina o tipo de operação a efectuar:**
    - **no passo 2, a máquina está à espera de receber mais $15**
    - **no passo 5, a máquina fornece, finalmente, um selo**
  - **o algoritmo apresentado não está completo, ou seja, não prevê todas as situações**
    - **não verifica a utilização de moedas diferentes de $5, $15 e $20**
    - **não procede ao reinício da máquina para realizar a venda de um novo selo**

2. Modelos de Computação (5/24)

- autómato de estados finitos -

- **Diagrama de estados**
  - **cada passo do algoritmo corresponde a um *estado de computação* ou *estado da máquina***
  - **o algoritmo como um todo (e, portanto, o funcionamento da máquina) pode ser descrito por uma sequência de estados computacionais representada por um diagrama de estados**



45

2. Modelos de Computação (6/24)

- autómato de estados finitos -

- **Características da máquina de selos**
  - **o algoritmo da máquina de selos possui leitura e saída de dados, teste de condições e saltos**
  - **os dados de entrada são usados imediatamente depois de lidos**
  - **a máquina não possui**
    - **memória, pelo que não dispõe de uma operação de atribuição (no passo 2, por exemplo, não é possível reutilizar o valor de m_1)**
    - **operação de paragem**
  - **este modelo de computação é designado de *autómato de estados finitos* ou *autómato finito determinístico (finite-state machine: FSM)***

46

2. Modelos de Computação (7/24)

- autómato de estados finitos -

- **Definição de FSM**
  - ***M* = [*S*, *I*, *O*, f_s, f_o] é uma FSM se**
    - ***S* for um conjunto finito de estados (s_o denota o estado inicial)**
    - ***I* for um conjunto finito de símbolos de entrada (o *alfabeto de entrada*)**
    - ***O* for o conjunto finito de símbolos de saída (o *alfabeto de saída*)**
    - **f_s e f_o forem funções onde**
      - **f_s: S × I → S é a função do próximo estado**
      - **f_o: S → O é a função de saída**
    - **exemplo da máquina de selos:**
      - **S = {1, 2, 3, 4, 5}**
      - **I = {c, d, v}**
      - **O = {abre_portinhola}**
      - **f_s = [(1, c) → 2, (1, d) → 3, (1, v) → 5, (2, c) → 3, …]**
      - **f_o = [5 → abre_portinhola]**

2. Modelos de Computação (8/24)

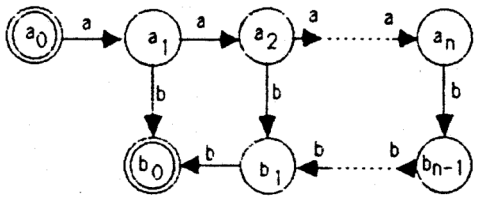
- autómato de estados finitos -

- **Reconhecimento por FSMs**
  - **a sequência de operações de uma FSM corresponde a um *processo computável***
  - **a computação de uma FSM tanto pode ser vista como o resultado (valor) obtido no estado final, como a classe de sequências de entrada aceites pela máquina**
  - **uma FSM *M* com alfabeto de entrada *I* reconhece ou aceita um subconjunto *S* de *I** se *M*, começando no estado s_o e processando uma sequência de entrada α, terminar num estado final em que $\alpha \in S$**
    - **nota: *I** denota o conjunto de todas as cadeias de comprimento finito sobre o alfabeto de entrada *I:***
      - ***a sequência vazia (sequência sem símbolos), λ, pertence a I****
      - ***qualquer elemento de I pertence a I****
      - ***x^y pertence a I*, se x e y forem sequências em I****

2. Modelos de Computação (9/24)

- autómato de estados finitos -

- **O exemplo do reconhecedor de $a^i b^i$, com $i \leq n$**
  - **o diagrama seguinte retracta a FSM que aceita expressões regulares do tipo $a^i b^i$, com $i \leq n$, onde a^i denota a repetição i vezes do símbolo a**
  - **o comprimento das sequências é limitado ($i \leq n$) pelo número de estados da máquina ($2.n+1$)**



2. Modelos de Computação (10/24)

- autómato de estados finitos -

- **Expressão regular**
  - **expressões regulares sobre I são**
    - **o símbolo ∅ (define a linguagem vazia)**
    - **o símbolo λ (frase nula {λ})**
    - **o símbolo i, ∀ i ∈ I**
    - **se A e B forem expressões regulares, então também**
      - **a sequência AB (o mesmo que A^B)**
      - **a sequência A+B**
      - **a sequência A***
    - **Exemplos de expressões regulares para I= {0, 1}:**
      - **1*0(01)* representa qualquer número (incluindo nenhum) de 1s, seguido por um único 0, seguido por qualquer número (incluindo nenhum) de pares 01**
      - **0+1* representa um único 0 ou qualquer número de 1s**
      - **11((10)*11)*(00*) representa uma sequência não vazia de pares de 1s intercalados por qualquer número de pares 10, seguido por, pelo menos, um 0**

2. Modelos de Computação (11/24)

- autómato de estados finitos -

- **Conjunto Regular**
  - **qualquer conjunto representado por uma expressão regular de acordo com as regras seguintes é chamado de conjunto regular:**
    - **$\varnothing$ representa o conjunto vazio**
    - **λ representa o conjunto $\{\lambda\}$ contendo a sequência vazia**
    - **i representa $\{i\}$**
    - **para as expressões regulares A e B**
      - **AB representa o conjunto de todos os elementos da forma concatenada $\alpha\beta$, com $\alpha \in A$ e $\beta \in B$**
      - **A+B representa a união dos conjuntos A e B**
      - **A* representa o conjunto de todas as concatenações dos elementos do conjunto A**

2. Modelos de Computação (12/24)

- autómato de estados finitos -

- **Teorema de Kleene**
  - **qualquer conjunto reconhecido por uma FSM é regular e qualquer conjunto regular pode ser reconhecido por uma FSM**
  - **Notas:**
    - **o teorema de Kleene estabelece as limitações, bem como as capacidades das FSMs, pois nem todos os conjuntos são regulares e esses outros possuem elementos não reconhecíveis por FSMs**
    - **por exemplo, $S = \{ 0^i 1^i \mid i \geq 0 \}$ não é regular, pelo que, pelo teorema de Kleene, não existe nenhuma FSM capaz de reconhecer o conjunto S**

2. Modelos de Computação (13/24)

- autómato de pilha -

- **O PDA como extensão das FSMs**
  - **as capacidades das FSMs podem ser estendidas, se for lhes for adicionada memória que permita guardar leituras para operar num estado posterior**
  - **esta extensão possui uma capacidade ilimitada de memória, gerida segundo o princípio FILO (*first in, last out*), ou seja, o primeiro símbolo guardado na memória é o último a sair**
  - **esta memória é designada de *pilha* (*stack*) e é operada por duas instruções:**
    - **PUSH, que guarda um símbolo no topo da pilha**
    - **POP, que retira um símbolo do topo da pilha**
  - **este modelo de computação é designado de *autómato de pilha determinístico* ou *máquina de stack (push-down automaton: PDA)***

2. Modelos de Computação (14/24)

- autómato de pilha -

- **Definição de PDA**
  - ***M* = [*S*, *I*, Γ, *O*, f_s, f_o] é um PDA se**
    - ***S* for um conjunto finito de estados (s_o denota o estado inicial)**
    - ***I* for o alfabeto de entrada**
    - **Γ ⊂ *I* for o conjunto finito de símbolos da pilha (alfabeto da pilha)**
    - ***O* for o alfabeto de saída**
    - **f_s e f_o forem funções onde**
      - **f_s: S × I × Γ → S × {↓, ↑, ▪} é a função do próximo estado**
      - **f_o: S → O é a função de saída**
    - **a computação de um PDA começa no estado inicial com a pilha vazia**
    - **uma sequência de entrada é aceite, se a máquina termina num estado final com a pilha vazia**
    - **se a pilha está vazia, o seu topo está ocupado com λ**

2. Modelos de Computação (15/24)

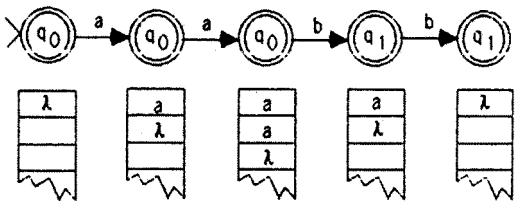
- autómato de pilha -

- **Actuação sobre a pilha**
  - **com $(q_i \wedge q_j) \in S \wedge x \in I \wedge (y \wedge z) \in \Gamma$:**
    - **$f_s (q_i, x, y) = (q_j, z\downarrow)$ significa que no estado q_i, com a entrada x e com o topo de pilha y, o PDA transita para o estado q_j e faz PUSH de z**
    - **$f_s (q_i, x, y) = (q_j, y\uparrow)$ significa que no estado q_i, com a entrada x e com o topo de pilha y, o PDA transita para o estado q_j e faz POP de y**
    - **$f_s (q_i, x, y) = (q_j, =)$ significa que no estado q_i, com a entrada x e com o topo de pilha y, o PDA transita para o estado q_j e deixa a pilha inalterada**

2. Modelos de Computação (16/24)

- autómato de pilha -

- **O exemplo do reconhecedor de $a^i b^i$, com $i \geq 0$**
  - **$S = \{q_0, q_1, q_2\}$ $I = \{a, b\}$ $\Gamma = \{a\}$ $O = \{a\}$**
  - **$f_s = [(q_0, a, \lambda) \mapsto (q_0, a\downarrow), (q_0, a, a) \mapsto (q_0, a\downarrow),$**
    $(q_0, b, \lambda) \mapsto (q_2, =), (q_0, b, a) \mapsto (q_1, a\uparrow),$
    $(q_1, a, \lambda) \mapsto (q_2, =), (q_1, a, a) \mapsto (q_2, a\uparrow),$
    $(q_1, b, \lambda) \mapsto (q_2, =), (q_1, b, a) \mapsto (q_1, a\uparrow)]$



2. Modelos de Computação (17/24)

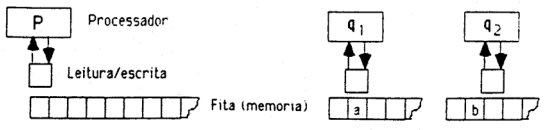
- máquina de Turing -

- **O modelo computacional MT *(proposto por Alan Turing em 1936)***
  - **para representar computacionalmente procedimentos algorítmicos mais gerais do que aqueles que podem ser modelados por FSMs e PDAs deve recorrer-se ao modelo computacional das máquinas de Turing (*Turing machines: MT*)**
  - **uma MT é essencialmente uma FSM com**
    - **a habilidade de ler as suas entradas mais do que uma vez**
    - **a habilidade de apagar ou substituir os valores das suas entradas**
    - **uma memória auxiliar ilimitada**
  - **uma MT com estas características consegue superar as limitações dos modelos computacionais precedentes (FSM e PDA), tornando-o no modelo que esteve por trás do projecto e desenvolvimento do "computador digital de programa armazenado" ainda utilizado actualmente**

2. Modelos de Computação (18/24)

- máquina de Turing -

- **O modelo computacional MT (cont.)**
  - **uma MT pode ser vista como um processador *P* acoplado a uma cabeça de leitura/escrita de símbolos numa fita que se estende infinitamente numa direcção (por exemplo, para a direita, tal como surge na figura)**
  - **a memória é representada pela fita**
  - **a deslocação da cabeça sobre a fita é efectuada mediante a indicação do sentido do movimento L (de *left*) ou R (de *right*)**



2. Modelos de Computação (19/24)

- máquina de Turing -

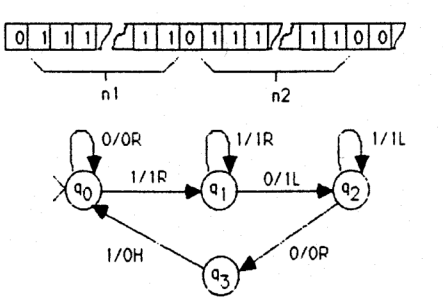
- **Definição de MT**
  - ***M* = [*S*, *I*, Γ, f_s] é uma MT se**
    - ***S* for um conjunto finito de estados (s_o denota o estado inicial)**
    - ***I* ⊂ Γ – {λ} for o alfabeto de entrada**
    - **Γ for um conjunto finito de símbolos da fita (alfabeto da memória)**
    - **f_s for uma função f_s: S × Γ → S × Γ × {L, R, H}**
    - **o funcionamento de uma MT (transição da MT) é especificável fornecendo indicações para 3 acções**
      - **a passagem ao novo estado**
      - **a escrita de um símbolo**
      - **o deslocamento da cabeça**
    - **a transição de uma MT é caracterizada por instanciando valores a f_s (q_i, x) = (q_j, z, op) com (q_i ∧ q_j) ∈ *S* ∧ (x ∧ z) ∈ Γ ∧ op ∈ {L, R, H}**
    - **exemplo:**
      - **na figura b) anterior efectuou-se f_s (q_1, a) = (q_2, b, R), ou simplesmente (q_1, a, q_2, b, R)**

2. Modelos de Computação (20/24)

- máquina de Turing -

- **O exemplo do somador de dois números**

q_i	x	q_j	z	op	Comentário
q_0	0	q_0	0	R	move →
q_0	1	q_1	1	R	sobre n1
q_1	1	q_1	1	R	até 0;
q_1	0	q_2	1	L	substitui por 1
q_2	1	q_2	1	L	e move ←
q_2	0	q_3	0	R	até início.
q_3	1	q_0	0	H	Remove 1
					e pára



2. Modelos de Computação (21/24)

- máquina de Turing -

- **O modelo computacional MT revisitado**
  - **é possível demonstrar que as várias extensões possíveis de uma MT (não-determinismo, transições expontâneas, vároas fitas, etc.) não aumentam as suas capacidades computacionais**
  - **foram desenvolvidos procedimentos específicos com vista a "construir" máquinas complexas à custa de MTs elementares**
  - **tais máquinas, capazes de realizar computações já longe da trivialidade, têm contudo um equivalente MT satisfazendo a definição inicial**
  - **neste contexto e perante um vasto conjunto de resultados teóricos, a comunidade científica aceita actualmente a chamada *tese de Church-Turing* (Church, 1936)**

2. Modelos de Computação (22/24)

- máquina de Turing -

- **Tese de Church-Turing**
  - **qualquer problema é resolúvel por um processo algorítmico se e só se for também resolúvel por uma máquina de Turing**
- **Computabilidade**
  - **uma função diz-se *computável* se puder ser avaliada numa MT para quaisquer dados válidos com um número finito de passos**
  - **é possível mostrar que uma vastíssima classe de funções numéricas (designadas de recursivas) são computáveis**
  - **a computacionalidade destas funções permite, na prática, a computação de qualquer problema algoritmizável, desde que previamente convertido num problema numérico através de codificação conveniente**

2. Modelos de Computação (23/24)

- máquina de Turing -

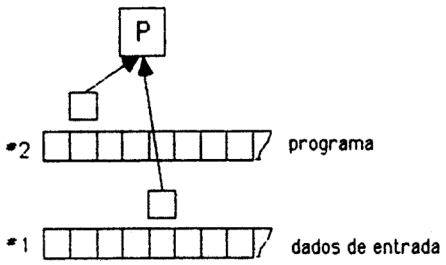
- **MT universal**
  - **a definição anteriormente apresentada para uma MT pressupõe uma computação fixa definida através da função f_s**
  - **contudo, a tabela de transição de estados pode ser codificada e o código resultante (programa) constituir uma fita suplementar (fita #2) de uma MT**
  - **neste cenário, num dado estado q_i a MT**
    - **lê os dados *x* da fita #1 (memória de dados)**
    - **lê o próximo estado q_j da fita #2 (memória de programa)**
  - **esta MT programável e com duas fitas é designada por *máquina de Turing universal* e pode ser convertida numa MT convencional de uma só fita**

63

2. Modelos de Computação (24/24)

- máquina de Turing -

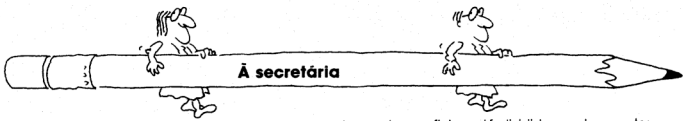
- **MT universal (cont.)**
  - **a MT universal constitui o modelo abstracto de computação que vai suportar toda a abordagem aos algoritmos e à programação seguida na disciplina de Programação Estruturada**

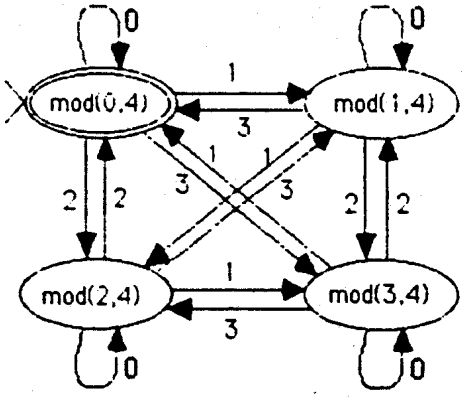


64

exercícios

1. **Considere uma FSM que realiza a soma de módulo 4:**
   - **a) Arbitre os elementos pertencentes a *I***
   - **b) Caracterize o conjunto *S***
   - **c) Caracterize a função f_s**
   - **d) Desenhe o diagrama de estados da FSM**
2. **Construa uma FSM que calcule x+1, onde x é dado na forma binária, com o bit LBS primeiro.**
3. **Diga se as sequências pertencem aos conjuntos regulares apresentados.**
   - **a) 01110111 ∈ (1*01)*(11+0*) ?**
   - **b) 11100111 ∈ [(1*0)*+0*11]* ?**
   - **c) 011100101 ∈ 01*10*(11*0)* ?**
   - **d) 1000011 ∈ (10*+11)*(0*1)* ?**
4. **Caracterize uma máquina de Turing que substitua, na fita, por 1s todos os 0s e por 0s todos os 1s, de uma sequência constituída por 0s e 1s.**





3. Linguagens e Gramáticas (1/25)

- **Considerações sobre linguagens**
  - **uma linguagem não admite todas as combinações possíveis dos vocábulos (símbolos do seu alfabeto)**
  - **apenas certas combinações é que dão origem a frases válidas**
  - **a linguagem é vulgarmente um conjunto infinito, o que torna a sua enumeração impossível**
  - **para definir uma linguagem é necessário**
    - **estabelecer qual o alfabeto a usar**
    - **indicar as regras que restringem as combinações possíveis àquelas que, de facto, serão frases correctas e que se dividem, tipicamente, em dois conjuntos: as regras *sintácticas* e as *semânticas***

3. Linguagens e Gramáticas (2/25)

- **Regras sintácticas**
  - **definem as formas correctas, estabelecendo as combinações, ou agrupamentos, de símbolos possíveis**
  - **preocupam-se com a estrutura das frases, actuando ao nível das intenções**
  - **indicam, por exemplo, que símbolos devem ser usados e porque ordem devem ser escritos, quando, num programa para computador, se quer**
    - **atribuir o valor de uma expressão a uma variável**
    - **ordenar a repetição de um bloco de instruções**

3. Linguagens e Gramáticas (3/25)

- **Regras semânticas**
  - **definem as condições que têm que ser respeitadas pelos símbolos para que as frases sintacticamente correctas façam sentido, i.e., para que seja possível interpretá-las (compreender e executar a sua mensagem)**
  - **preocupam-se com o significado das frases (o seu conteúdo semântico), indicando como é que essas frases serão interpretadas, trabalhando ao nível dos valores intrínsecos dos símbolos ou aferíveis a partir deles**
  - **o significado é a informação contida numa frase e é aquilo que realmente interessa conhecer para que a comunicação entre dois agentes tenha algum efeito prático**

3. Linguagens e Gramáticas (4/25)

- **Sintaxe vs. semântica**
  - **"Os carneiros falam muito alto."**
    - **o significado pode surpreender, uma vez que a sua *semântica* parece pouco adequada (os carneiros não falam!)**
    - **a forma é perfeitamente aceitável, uma vez que a sua *sintaxe* é válida na língua portuguesa, ou seja, as várias partes que a compõem (substantivos, verbos, etc.) estão encadeados de maneira correcta**
  - **"Alto carneiro os falam."**
    - **o significado é incompreensível, uma vez que a forma como a frase foi escrita viola as regras sintácticas**
    - **a violação de sintaxe, por combinação "ilegal" das palavras que a compõem (mas também poderia ter sido por utilização de palavras inexistentes), põe em causa a compreensão do seu significado (semântica)**

3. Linguagens e Gramáticas (5/25)

- **Linguagens naturais**
  - **nas linguagens naturais (línguas faladas no dia-a-dia pelos povos) as frases pertencem à linguagem por razões *de facto*, i.e., porque as pessoas as usam assim mesmo na sua comunicação quotidiana**
  - **as regras surgem, então, posteriormente com o intuito de sistematizar e ensinar futuramente a linguagem e organizar (estruturar) essas frases**
  - **é frequente ter de se recorrer à enumeração de excepções para cobrir toda a linguagem natural**

3. Linguagens e Gramáticas (6/25)

- **Linguagens artificiais**
  - **nas linguagens artificiais (aquelas que são criadas com o propósito de suportar a comunicação homem/máquina) só começam a ser usadas depois de o vocabulário ter sido escolhido e de as regras sintácticas e semânticas terem sido estabelecidas**
  - **quando essas regras são apresentadas rigorosamente, através do recurso a formalismos apropriados, a linguagem artificial diz-se *formal***
  - **as regras que definem as linguagens artificiais são pensadas de modo a garantir a *não ambiguidade* dessas linguagens, ou seja, a existência de uma única interpretação possível para cada frase válida**

3. Linguagens e Gramáticas (7/25)

- **Definição de alfabeto**
  - **um *alfabeto A* é um conjunto finito não vazio de símbolos**
    - **nota: uma *palavra* sobre *A* é uma sequência de comprimento finito de símbolos de *A*, vulgarmente designada por *cadeia de caracteres* (*string*)**
  - **exemplos:**
    - **o conjunto de palavras da língua portuguesa listadas num dicionário constitui o alfabeto da língua portuguesa**
    - **o conjunto $A = \{0, 1\}$ constitui o alfabeto binário**

3. Linguagens e Gramáticas (8/25)

- **Definição de frase**
  - **uma *frase* é uma sequência finita de palavras de um alfabeto *A*, ou seja, é uma cadeia de caracteres composta**
  - **exemplos:**
    - **os palavras "processadores", "de" e "linguagens" fazem parte do alfabeto da língua portuguesa**
    - **"processadores de linguagens" é uma frase à luz do alfabeto da língua portuguesa**
- **Definição de linguagem**
  - **uma *linguagem L* sobre *A* é um qualquer subconjunto de *A****
  - ***L* ⊂ *A****
    - **nota: *A** denota o conjunto de todas as cadeias de caracteres compostas (frases) que se podem construir com o alfabeto *A***

3. Linguagens e Gramáticas (9/25)

- **Gramática**
  - **as cadeias de caracteres compostas (frases) da linguagem *L* obedecem a um certo conjunto de regras que constituem a *gramática* da linguagem**
    - **Exemplo:**

<frase> → <sujeito> <predicado>
<sujeito> → <determinante> <substantivo>
<predicado> → <verbo> <advérbio>

<frase> → <sujeito> <predicado>
→ <determinante> <substantivo> <predicado>
→ *Os* <substantivo> <predicado>
→ *Os carneiros* <predicado>
→ *Os carneiros* <verbo> <advérbio>
→ *Os carneiros falam* <advérbio>
→ *Os carneiros falam alto*

3. Linguagens e Gramáticas (10/25)

- **Definição de gramática**
  - ***G* = [V_T, V_N, *S*, *P*] é uma gramática se**
    - **V_T for um conjunto finito, não vazio, de símbolos designados de *terminais* (ou *palavras)* que constituem o alfabeto da linguagem (V_T também é designado de *léxico* da linguagem)**
    - **V_N for um conjunto finito, não vazio, de símbolos designados de *não-terminais* que representam classes de elementos de V_T (classes sintácticas) não pertencentes a V_T, ou seja, $V_T \cap V_N = \varnothing$**
      - **os não-terminais são, tipicamente, representados ou em maiúsculas, ou em negrito (*bold*), ou entre "<" e ">"**
      - **$V = V_T \cup V_N$ é chamado *vocabulário* da linguagem *L***
    - **$S \in V_N$, sendo designado de *símbolo inicial***

3. Linguagens e Gramáticas (11/25)

- **Definição de gramática (cont.)**
  - ***G* = [V_T, V_N, *S*, *P*] é uma gramática se**
    - ***P* for um conjunto finito de pares (α, β) designados de *regras de produção* (ou *regras de reescrita*, ou *regras sintácticas*) representados na forma $\alpha \rightarrow \beta$, com $\alpha \in V^+ \wedge \beta \in V^*$ e significando que numa sequência de símbolos em que apareça α pode substituir-se α por β**
      - **$\alpha \rightarrow \beta$ lê-se "α produz β", ou "α pode ser substituído por β", ou " β deriva de α"**
    - **Notas:**
      - **um elemento $\nu \in V^*$ derivável de *S* por aplicação das regras de produção *P* chama-se *forma sentencial***
      - **uma linguagem *L* com gramática *G* denota-se *L(G)* e constitui o conjunto de todas as formas sentenciais deriváveis por *G***

3. Linguagens e Gramáticas (12/25)

- **Exemplo da gramática *G1***
  - V_T = {o, a, Luís, Carlos, cão, canção, canta, segura, bem, mal}
  - V_N = {<frase>, <sujeito>, <predicado>, <determinante>, <substantivo>, <verbo>, <advérbio>}
  - *S* = <frase>
  - *P* = {<frase> → <sujeito> <predicado>,
    <sujeito> → <determinante> <substantivo>,
    <sujeito> → <substantivo>,
    <predicado> → <verbo> <advérbio>,
    <determinante> → o, <determinante> → a,
    <substantivo> → Luís, <substantivo> → Carlos,
    <substantivo> → cão, <substantivo> → canção,
    <verbo> → canta, <verbo> → segura,
    <advérbio> → bem, <advérbio> → mal}
  - **"cão segura o canção bem" ∈ *L(G1)*, mas semanticamente inaceitável (sem sentido)**

3. Linguagens e Gramáticas (13/25)

- **Exemplo da gramática *G2***
  - $V_T = \{(,), i, j, +, *\}$
  - $V_N = \{E, T\}$
  - $S = E$
  - $P = \{E \rightarrow T \mid E + T \mid E * T,\ T \rightarrow (E) \mid i \mid j\}$
  - nota: "$\alpha \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta n$" significa "$\alpha \rightarrow \beta_1$, $\alpha \rightarrow \beta_2$, ..., $\alpha \rightarrow \beta_n$" o símbolo "$\mid$" lê-se "ou"
  - G2 é *recursiva*, porque possui um mesmo não-terminal em ambos os membros de uma produção
  - uma gramática com produções recursivas dá origem a uma linguagem *infinita*, i.e., possui um número ilimitado de frases
  - $(i * j + i) * i \in L(G2)$
  - $i * (j + (+ i + j)) \notin L(G2)$

3. Linguagens e Gramáticas (14/25)

- **Exemplo da gramática *G3***
  - $V_T = \{a, b, c\}$
  - $V_N = \{S, A, B\}$
  - $S = S$
  - $P = \{S \rightarrow Ac \mid aB,\ A \rightarrow ab,\ B \rightarrow bc\}$
  - a cadeia de caracteres "abc" pode ser gerada de dois modos distintos:

$S \rightarrow Ac$
$\rightarrow abc$

ou

$S \rightarrow aB$
$\rightarrow abc$

  - *G3* é uma gramática *ambígua*

3. Linguagens e Gramáticas (15/25)

- **Linguagem gerada por uma gramática**
  - **seja *G* uma gramática $G = [V_T, V_N, S, P]$ e sejam w_1 e w_2 palavras sobre *V***
    - **se $\alpha \rightarrow \beta$ for uma produção de *G*, se w_1 contiver uma cópia de α e se w_2 for obtido de w_1 através da substituição de α por β, então w_1 gera (deriva) directamente w_2, denotando-se $w_1 \Rightarrow w_2$**
    - **se $w_1, w_2, \ldots, w_n$ forem palavras sobre *V* e se $w_1 \Rightarrow w_2 \wedge w_2 \Rightarrow w_3 \wedge \ldots \wedge w_{n-1} \Rightarrow w_n$, então w_1 gera (deriva) w_n, denotando-se $w_1 *\Rightarrow w_n$**
    - **por convenção, $w_1 *\Rightarrow w_1$**
  - **dada uma gramática *G*, a linguagem *L* gerada por *G*, denotada *L(G)*, é o conjunto $L = \{ w \in V_T^* \mid S *\Rightarrow w \}$, ou seja, *L* é o conjunto de todas as cadeias de terminais geradas a partir do símbolo inicial**

3. Linguagens e Gramáticas (16/25)

- **Hierarquia de Chomsky**
  - **é possível obter vários tipos de gramáticas tendo em conta as restrições impostas às regras de produção**
  - **as linguagens podem ser igualmente catalogadas, tendo em conta o tipo de gramática que as geram**

3. Linguagens e Gramáticas (17/25)

- **Tipo 0: gramáticas irrestritas**
  - **as gramáticas do tipo 0 correspondem à definição de gramática apresentada anteriormente**
  - **as linguagens geradas por gramáticas do tipo 0 correspondem a classes de sequências aceites por máquinas de Turing**
  - **para cada gramática de tipo 0 é possível encontrar uma MT que implementa a gramática, no sentido em que aceita as frases da linguagem por ela gerada**

3. Linguagens e Gramáticas (18/25)

- **Tipo 1: gramáticas sensíveis ao contexto**
  - **as gramáticas do tipo 1, relativamente às tipo 0, possuem a seguinte restrição $\alpha \rightarrow \beta$, com $\alpha, \beta \in V^+ \wedge |\alpha| \leq |\beta|$**
  - **as produções de uma gramática do tipo 1 geram sequências de comprimento não decrescente**
  - **as regras de produção são da forma $\alpha a \beta \rightarrow \alpha \nu \beta$, com $a \in V_N \wedge \alpha, \beta \in V^* \wedge \nu \in V^+$**
  - ***a* é substituível por ν no contexto $\alpha\beta$**
  - **as linguagens geradas por gramáticas do tipo 1 são reconhecidas por *autómatos linearmente limitados* (*linear bounded automaton: LBA*) que são máquinas de Turing que aceitam o conjunto de todas as entradas para as quais existe alguma sequência de movimentos que as façam parar em algum estado**

3. Linguagens e Gramáticas (19/25)

- **Tipo 2: gramáticas independentes de contexto**
  - **as gramáticas do tipo 2, relativamente às tipo 0, possuem a seguinte restrição $\alpha \rightarrow \beta$, com $\alpha \in V_N \wedge \beta \in V^*$**
  - **no lado esquerdo das regras de produção só aparecem não-terminais, ou seja, a substituição de um não-terminal não depende do contexto**
  - **as linguagens geradas por gramáticas do tipo 2 são reconhecidas por autómatos de pilha**
  - **as linguagens do tipo 2 são importantes por 3 razões**
    - **são de relativa simplicidade, uma vez que só permitem a substituição de um símbolo de cada vez**
    - **diversas linguagens de programação podem ser descritas através de gramáticas do tipo 2**
    - **as sequências de produções podem ser representadas graficamente recorrendo a uma *árvore sintáctica* ou *de análise* (*parsing tree*)**

3. Linguagens e Gramáticas (20/25)

- **Exemplo de gramática do tipo 2**
  - **a geração de identificadores numa linguagem de programação pode ser representada por uma gramática do tipo 2**

<identificador> → <letra>
<identificador> → <identificador> <letra>
<identificador> → <identificador> <dígito>
<letra> → *a*
<letra> → *b*
...
<letra> → *z*
<dígito> → *0*
<dígito> → *1*
...
<dígito> → *9*

3. Linguagens e Gramáticas (21/25)

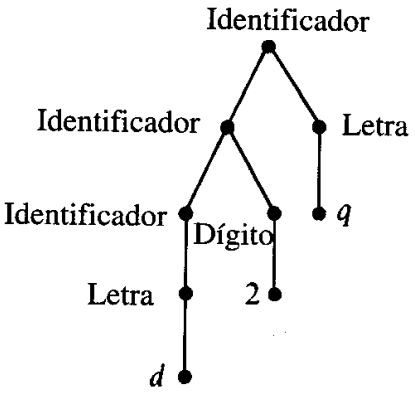
- **Exemplo de gramática do tipo 2 (cont.)**
  - **a palavra "d2q" pode ser derivada da seguinte maneira**

<identificador> → <identificador> <letra>
→ <identificador> <dígito> <letra>
→ <letra> <dígito> <letra>
→ *d* <dígito> <letra>
→ *d 2* <letra>
→ *d 2 q*

  - **esta derivação pode ser representada recorrendo a uma *parsing tree***
    - **é um gráfico orientado em que cada nodo representa uma forma sentencial a partir da qual se obtém, de cima para baixo, as formas sentenciais derivadas**
    - **a raiz é o símbolo inicial**

3. Linguagens e Gramáticas (22/25)

- **Exemplo de gramática do tipo 2 (cont.)**



3. Linguagens e Gramáticas (23/25)

- **Tipo 3: gramáticas regulares**
  - **as gramáticas do tipo 3, relativamente às tipo 0, possuem a seguinte restrição $\alpha \rightarrow a\beta \mid a \mid \lambda$, com $\alpha, \beta \in V_N \wedge a \in V_T$**
  - **as formas sentenciais nas gramáticas do tipo 3 contêm, no máximo, um só não-terminal do lado direito da produção**
  - **cada regra de produção contribui para juntar um terminal**
  - **as linguagens geradas por gramáticas do tipo 3 são reconhecidas por autómatos de estados finitos**

3. Linguagens e Gramáticas (24/25)

- **A meta-linguagem BNF (*Backus Naur Form*)**
  - **nas descrições anteriores de gramáticas, utilizaram-se vários símbolos e algumas convenções/regras que constituem uma linguagem para descrever linguagens, ou seja, uma *metalinguagem***
  - **a descrição da linguagem *Algol60* foi realizada por *John Bakus* e por *Peter Naur*, em 1959, recorrendo a notações especiais de descrição que, mais tarde, vieram a designar-se *formas de Backus Naur*, ou *formas normais de Bakus*, ou simplesmente *notação BNF***
  - **Exemplo:**

<identificador> ::= <letra> | <identificador> <letra> |
<identificador> <dígito>
<letra> ::= *a* | *b* | ... | *z*
<dígito> ::= *0* | *1* | ... | *9*

3. Linguagens e Gramáticas (25/25)

- **Regras BNF**
  - **::= é o símbolo de produção**
  - **| é o símbolo de alternativa**
  - **<*string*> designa um não-terminal**
- **Regras EBNF (*extended* BNF) ou ABNF (*augmented* BNF)**
  - **( ) permite a factorização**
    - **<identificador> ::= <letra> | <identificador> (<letra> | <dígito>)**
  - **{ } permite a iteração de zero ou mais vezes**
    - **<identificador> ::= <letra> { <letra> | <dígito> }**
  - **[ ] permite especificar opcionalidade**
    - **<a> ::= ST | SBT | SZT é equivalente a <a> ::= S [B | Z] T**

exercícios

I - Gramáticas

1. **Considere um gramática *G* com $V_T = \{0, 1\}$, $V_N = \{S\}$ e $P = \{S \rightarrow 0S, S \rightarrow 1\}$. Mostre que $00S \overset{*}{\Rightarrow} 00000S$.**
2. **Seja $L = \{a^n b^n c^n;\ n \geq 1\}$. Uma possível gramática *G* que gera *L* possui $V_T = \{a, b, c\}$, $V_N = \{S, B, C\}$ e $P = \{S \rightarrow aSBC, S \rightarrow aBC, CB \rightarrow BC, aB \rightarrow ab, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\}$. Derive $a^2b^2c^2$.**
3. **Considere as seguintes três gramáticas:**
   - ***G1* com $V_T = \{0\}$, $V_N = \{S, A, B\}$ e $P = \{S \rightarrow \lambda, S \rightarrow ABA, AB \rightarrow 00, 0A \rightarrow 000A, A \rightarrow 0\}$**
   - ***G2* com $V_T = \{0\}$, $V_N = \{S, A\}$ e $P = \{S \rightarrow \lambda, S \rightarrow 00A, A \rightarrow 00A, A \rightarrow 0\}$**
   - ***G3* com $V_T = \{0\}$, $V_N = \{S, A, B, C\}$ e $P = \{S \rightarrow \lambda, S \rightarrow 0A, A \rightarrow 0B, B \rightarrow 0, B \rightarrow 0C, B \rightarrow 0B\}$**

   a) Mostre que as três gramáticas são capazes de gerar a linguagem L que consiste numa cadeia vazia λ, juntamente com o conjunto de todas as cadeias compostas por um número ímpar *n* de 0s, com $n \geq 3$.

   b) Mostre que, apesar das três linguagens serem equivalentes, pertencem a classes distintas, uma vez que *G1* é do tipo 1, *G2* é do tipo 2 e *G3* é do tipo 3.

exercícios

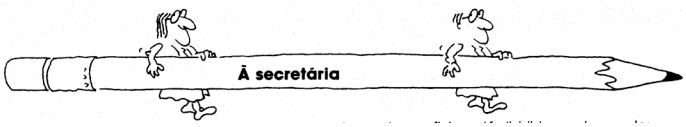
II - BNF

4. Considere o seguinte conjunto de produções em BNF:

<expr> ::= <expr> + <expr> | <expr> – <expr> | <expr> * <expr> | (<expr>) | <constante>
<constante> ::= <dígito> | <constante> <dígito>
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Derive (1 + 3) * 4.

5. Recorrendo ao operador de iteração do BNF estendido escreva a produção de
a) constantes inteiras.
b) expressões aritméticas simples.

6. Recorrendo ao operador de opcionalidade do BNF estendido escreva a produção de um endereço postal.



5. a) <constante> ::= <dígito> { <dígito> }
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

5. b) <expr> ::= <termo> {+ <termo> | – <termo> }
<termo> ::= <factor> {* <factor> }
<factor> ::= <constante> | (<expr>)
<constante> ::= <dígito> { <dígito> }
<dígito> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

6. <endereco_postal> ::= <nome> <rua> <codigo_postal> [<país>]
<nome> ::= <nome_proprio> <apelido> | <nome_proprio> <nome>
<nome_proprio> ::= <nome> | <inicial>
<rua> ::= <nome_da_rua> <numero_de_policia> [<andar>]
<codigo_postal> ::= <codigo> <localidade>

4. Processamento de Linguagens (1/15)

- **Processador para uma linguagem**
  - **seja *L*(*G*) a linguagem gerada por uma gramática G; um processador para essa linguagem *P*[*L*(*G*)] é um programa que, tendo conhecimento da gramática *G***
    - **lê um texto (sequência de caracteres)**
    - **verifica se esse texto é uma frase válida de *L*(*G*)**
    - **executa uma acção qualquer em função do significado da frase reconhecida**
  - **esta definição é muito genérica, no entanto, todos os programas que sejam considerados processadores de linguagens são constituídos por dois módulos:**
    - **o módulo de *análise* que executa o reconhecimento do significado do texto fonte**
    - **o módulo de *síntese* que reage ao significado identificado, produzindo um determinado resultado**

4. Processamento de Linguagens (2/15)

- **Módulo de análise**
  - ***análise léxica*, responsável pela leitura sequencial dos caracteres que formam o texto fonte, pela sua separação em palavras e pelo reconhecimento dos vocábulos (símbolos terminais) representados por cada palavra**
  - ***análise sintáctica*, encarregue de agrupar os símbolos terminais, verificando se formam uma frase sintacticamente correcta, i.e., composta de acordo com as regras sintácticas da linguagem**
  - ***análise semântica*, destinada a verificar se as regras semânticas da linguagem são satisfeitas e a calcular os valores aos símbolos, de modo a poder conhecer-se o significado completo da frase**
  - **nota: os erros lexicais e sintácticos ("*bohr-errors*") são imediatamente detectados durante a tradução (*compile time*), enquanto que os erros semânticos ("*heisen-errors*") só são conhecidos durante a execução do programa (*run time*)**

4. Processamento de Linguagens (3/15)

- **Exemplos de processadores de linguagens**
  - **assembladores (*assemblers*), que traduzem linguagens de programação de baixo nível formadas por menmónicas (linguagem *assembly*) para código máquina (binário)**
  - **compiladores (*compilers*), que traduzem linguagens de programação de alto nível para código máquina**
  - **interpretadores (*interpreters*), que executam os programas logo após o seu reconhecimento, i.e., em vez traduzirem os programas para uma linguagem de baixo nível, realizam acções**
  - **tradutores em geral, que transformam textos escritos numa linguagem qualquer para outra linguagem qualquer**

4. Processamento de Linguagens (4/15)

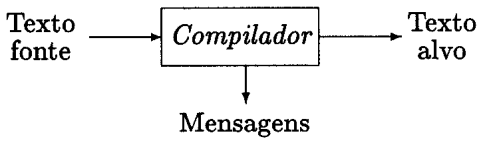
- **Exemplos de processadores de linguagens (cont.)**
  - **carregadores (*loaders*), que reconhecem descrições de dados e carregam essa informação para bases de dados, ou para estruturas de dados em memória central**
  - **pesquisadores, que reconhecem questões e pesquisam em bases de dados para mostrarem as respostas encontradas**
  - **filtros, que reproduzem à saída o texto que receberam à entrada, depois de lhe retirarem, expandirem ou transformarem certas palavras (ou blocos)**
  - **processadores de documentos, usados para diversos tipos de manipulações, tais como a formatação ou a extracção de conhecimento em documentos anotados**

4. Processamento de Linguagens (5/15)

- **Compiladores**
  - **processadores de linguagens construídos propositadamente para reconhecer programas escritos numa linguagem de alto nível para os traduzir para código máquina (ou código binário), que é uma linguagem de baixo nível directamente reconhecida e executada por um determinado processador**
  - **no caso particular em que o compilador aceita um texto escrito numa linguagem de alto nível e produz um texto escrito noutra linguagem de alto nível, o compilador é designado de *transcompilador* (*cross-compiler*)**
  - **tipicamente, os primeiros compiladores de linguagens de alto nível costumam ser implementados através de tradutores (transcompiladores) para outras linguagens de alto nível para os quais já existem compiladores**
    - **o primeiro compilador de *C++* era um transcompilador para *C***

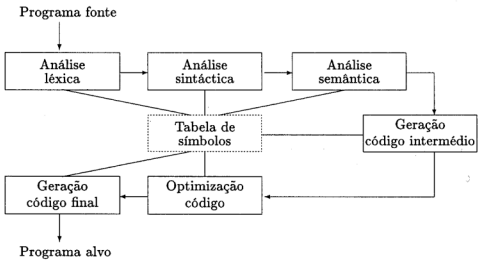
4. Processamento de Linguagens (6/15)

- **Compiladores (cont.)**
  - **um compilador é específico de um determinado processador, ou seja, produz código máquina que varia com a marca do processador em que se pretende correr o programa depois de compilado**
  - **as linguagens de programação de alto nível (texto fonte) são independentes da máquina alvo (processador em que o programa vai correr), enquanto que a linguagem de baixo nível (texto alvo) é específica para cada máquina**



4. Processamento de Linguagens (7/15)

- **Compiladores (cont.)**
  - **existe um grande desnível entre a complexidade das instruções na linguagem de alto nível e de baixo nível que "obriga" os compiladores a realizarem a geração de código em duas fases, recorrendo a código intermédio para facilitar a tarefa de compilação**

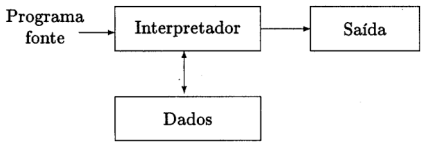


4. Processamento de Linguagens (8/15)

- **Assembladores**
  - **são funcionalmente muito parecidos com os compiladores, uma vez que traduzem um programa fonte para código máquina**
  - **a diferença reside no facto de que o texto fonte, no caso dos assembladores (*assemblers*), não é escrito numa linguagem de alto nível, mas sim numa linguagem de menmónicas (*assembly*) estruturalmente tão simples quanto um programa em código máquina**
  - **desta forma, o reconhecimento das frases é muito fácil, os esquemas de tradução são simples e o processo de tradução é directo**
  - **os assembladores são praticamente iguais aos módulos de geração de código final dos compiladores**

4. Processamento de Linguagens (9/15)

- **Interpretadores**
  - **os interpretadores são processadores de linguagens que não geram texto alvo, uma vez que executam as instruções do programa fonte, logo que o reconhecem**
  - **a execução é feita sobre a representação intermédia**
  - **os interpretadores processam, geralmente, linguagens menos ricas estruturalmente do que os compiladores**
    - **os exemplos típicos são as linguagens *Basic* e *Prolog***



4. Processamento de Linguagens (10/15)

- **Linguagem de alto nível**

```
VOID EXPORTAPI DrawBox(HDC hdc)
{
// hdc is a value used by Windows to identify a
// window to draw in.
// The numbers represent coordinates in the window.
   MoveTo(hdc, 50,50);
   LineTo(hdc, 100,50);
   LineTo(hdc, 100,100);
   LineTo(hdc, 50,100);
   LineTo(hdc, 50,50);
  }
```

4. Processamento de Linguagens (11/15)

- **Linguagem de baixo nível: *assembly***

```
DrawBox:
   mov   ax,SEG con0
   enter OFFSET L08041,OFFSET 0
   push  si
   push  di
   push  ds
   mov   ds,ax
   push  WORD PTR 6[bp]
   push  OFFSET 50
   push  OFFSET 50
   call  FAR PTR MoveTo
   push  WORD PTR 6[bp]
   push  OFFSET 100
   push  OFFSET 50
   call  FAR PTR LineTo
   push  WORD PTR 6[bp]
   push   OFFSET 100
   push   OFFSET 100
   call   FAR PTR LineTo
   push   WORD PTR 6[bp]
   push   OFFSET 50
   push   OFFSET 100
   call   FAR PTR LineTo
   push   WORD PTR 6[bp]
   push   OFFSET 50
   push   OFFSET 50
   call   FAR PTR LineTo
   pop    ds
   pop    di
   pop    si
   leave
   ret    OFFSET 2
```


4. Processamento de Linguagens (12/15)

- **Linguagem de baixo nível: código máquina**

```
B8 87 55
C8 02 00 00
56
57
1E
8E D8
FF 76 06
6A 32
6A 32
9A AA 0E 0F 05
FF 76 06
6A 64
6A 32
9A 96 0E 0F 05
FF 76 06
6A 64
6A 64
9A 96 0E 0F 05
FF 76 06
6A 32
6A 64
9A 96 0E 0F 05
FF 76 06
6A 32
6A 32
9A 96 0E 0F 05
E9 00 00
1F
5F
5E
C9
CA 02 00
```

4. Processamento de Linguagens (13/15)

- **Tratamento de erros**
  - **no contexto de processamento de linguagens, entende-se por tratamento de erros o processo que é desencadeado pelo reconhecedor logo após a *detecção de um erro* na frase que está a ser analisada**
  - **a reacção compreende duas grandes tarefas**
    - **a *sinalização do erro* (ou notificação) a enviar ao utilizador (programador) para o alertar para o facto ter sido encontrada uma violação a uma das regras que fazem da sequência de símbolos uma frase da linguagem**
    - **a *correcção/recuperação* que permite superar a falta detectada e prosseguir a análise até à aceitação da frase ou até à detecção de um novo erro**

4. Processamento de Linguagens (14/15)

- **Detecção de erros**
  - **a detecção de um erro é uma tarefa inerente à análise, ou seja, indissociável do reconhecimento**
  - **quando um processador está a analisar uma frase, tentando verificar se ela foi correctamente escrita, de acordo com a gramática da linguagem em causa, são duas as razões que podem levar à detecção de um erro**
    - ***erro léxico*, quando surgem caracteres inválidos que impedem a identificação de qualquer símbolo terminal que pertence ao alfabeto da linguagem**
    - ***erro sintáctico*, quando existe a combinação inválida de símbolos terminais válidos, impedindo o reconhecimento de uma subfrase aceitável**

4. Processamento de Linguagens (15/15)

- **Sinalização de erros**
  - **a sinalização do erro é de importância crucial para que o programador possa localizar o foco de problemas, interpretá-los e corrigi-los definitivamente**
  - **a mensagem enviada ao programador para assinalar um erro deveria, no mínimo, indicar**
    - **a posição (linha e coluna do programa fonte) onde o símbolo de erro foi encontrado**
    - **o símbolo de erro**
    - **a causa provável que justifica esse erro (diagnóstico), podendo indicar-se os símbolos de que o processador de linguagens estava à espera, ou a concordância que era pretendida**