Capítulo 1. Introdução e definições básicas

1.1. Linguagens

1.2. Gramáticas

1.3. Autómatos

1.4. Os três paradigmas da computação

1.1. Linguagens

1. Alfabeto, Σ

Σ={símbolos}, conjunto não vazio de símbolos (letras, algarismos, ...)

- {*a*,*b*},
- {0, 1, 2, ..., 9 }, conjunto dos algarismos árabes
- {0,1}, alfabeto binário
- { [, (,) ,]} ,
- {#,$,§,&}
- {*a*, *b*,*c*, ..., *z*}, o alfabeto romano
- { copos, facas, garfos, pratos, colheres, tachos}

Qualquer objecto pode pertencer a um alfabeto.

2. Cadeia (*string*), *w*

w={sequência finita de símbolos do alfabeto; pode ser vazia}

$\Sigma=\{a,b\}$,

$w= a$, $w=ab$, $w= abbaba$

3. Concatenação de cadeias *w*, *v*

$w= a_1a_2a_3 \ldots a_n \quad v= b_1b_2b_3\ldots b_m$

concatenação de w e v $\quad wv=a_1a_2a_3\ldots a_nb_1b_2b_3\ldots b_m$

concatenação de v e w, $\quad vw=b_1b_2b_3\ldots b_ma_1a_2a_3\ldots a_n$

4. Cadeia **reversa** , $w^R = a_n ... a_3 a_2 a_1$

5. **Comprimento** de uma cadeia, $|w|$:

número de símbolos da cadeia

6. Cadeia **vazia**, λ

cadeia sem qualquer símbolo, $|\lambda|=0$

$\lambda w = w\lambda = w$, para todo o w

77 . Palíndromos (Capicuas) (*palindromes*) em Σ *:* cadeias tais que $w = w^R$

8. **Subcadeia** de uma cadeia *w*

qualquer cadeia composta por caracteres sucessivos de *w*

w=*uv*, *u* é o **prefixo** e *v* o **sufixo** de *w*

Exemplo: *w*=*abbab*

Subcadeias de *w* : *a*, *b*, *bb*, *bba*, ...

Prefixos de *w*: {λ, *a*, *ab*, *abb*, *abba*, *abbab*}

Sufixos de *w*: {*abbab*, *bbab*, *bab*, *ab*, *b*, λ}

9. **Potências** de uma cadeia w^n

cadeia obtida pela (auto)concatenação n vezes da cadeia w

$w^0 = \lambda$, para todo o w.

10. Σ^* **:** conjunto de todas as cadeias que se podem obter pela concatenação de zero ou mais símbolos de Σ..
Contém sempre a cadeia nula λ.

11. $\Sigma^+ = \Sigma^* - \{\lambda\}$, i.e., exclui a cadeia vazia

quer Σ^* quer Σ^+ são sempre infinitas, mesmo quando Σ é finito.

12. **Linguagem** L: qualquer subconjunto de Σ^*

13. **Frase (sentença)** : qualquer cadeia na linguagem L.

Exemplo 1:

$\Sigma = \{a,b\}$

$\Sigma^* = \{\lambda,a,b,aa,ab,ba,bb,aaa,aab,aba,abb,baa,bab,bba,bbb, ...\}$,
todas as combinações possíveis dos símbolos do alfabeto, incluindo a cadeia vazia.

O conjunto $L=\{a, ab, abb\}$ é uma linguagem em Σ.

É uma linguagem **finita.**

Exemplo 2.

O conjunto $L=\{a^n b^n : n \geq 0\}$ é uma linguagem em Σ.

É uma linguagem **infinita.**

ab, *aabb*, *aaabbb*, ... pertencem *a L*

aba, *abb*, *aaababbbab*, não pertencem *a L*.

Uma linguagem é um **conjunto** !

Portanto aplicam-se-lhe todas as operações sobre conjuntos:

14. União: $L_1 \cup L_2 = \{w: w \in L_1 \text{ ou } w \in L_2\}$

15. Intersecção: $L_1 \cap L_2 = \{w: w \in L_1 \text{ e } w \in L_2\}$

16. Diferença de duas linguagens: $L_1 - L_2 = \{w: w \in L_1 \text{ e } w \notin L_2\}$

17. Complemento de uma linguagem: $Compl(L) = \Sigma^* - L$

18. Reversa de uma linguagem: $L^R = \{w^R : w \in L\}$

19. Concatenação de duas linguagens, $L_1 \bullet L_2$, $L_1 L_2$

$$L_1 \bullet L_2 = L_1 L_2 = \{uv : u \in L_1, v \in L_2\}$$

$L_1 \bullet (L_2 \cup L_3) = L_1 \bullet L_2 \cup L_1 \bullet L_3$

(distributividade da concatenação em ordem à união)

$L_1 \cup (L_2 \bullet L_3) \neq L_1 \cup L_2 \bullet L_1 \cup L_3$

em geral a união não é distributiva sobre a concatenação.

$L \bullet \varnothing = \varnothing$

$L \bullet \lambda = L$

20. Potência de uma linguagem: concatenação com ela própria:

$$L^0 = \{\lambda\},\ L^1 = L,\ L^2 = LL\ ,\ L^3 = LLL\ ,\ \ldots$$

21. Fecho-estrela (*star-closure, Kleene star, Kleene closure*) L^* : todas as cadeias que se podem obter por concatenação da linguagem com ela própria, incluindo sempre a cadeia vazia:

$$L^* : L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots..$$

Fecho-positivo *(positive closure)* L^+ : todas as cadeias que se podem obter por concatenação da linguagem com ela própria; não contém a cadeia vazia se L não contém λ ,

$$L^+ : L^1 \cup L^2 \cup L^3 \cup \ldots... = L^* - \{\lambda\} \qquad (\text{se } \lambda \notin L)$$

Exemplo : $L=\{a^n b^n, n \geq 0\}$

$L=\{\lambda, ab, aabb, aaabbb, aaaabbbb, ...\}$ (note-se que $\lambda \in L$)

$L^2 = LL=\{\lambda, ab, aabb, aaabbb, aaaabbbb, ...$
$abab, abaabb, abaaabbb, abaaaabbbb, ...$
$aabbab, aabbaabb, aabbaaabbb, ...\}$

$= \{a^n b^n a^m b^m, n \geq 0, m \geq 0\}$, n e m são independentes.

Reversa de L: $L^R=\{b^n a^n, n \geq 0\}$
Complemento de L : ???
Fecho estrela L^*: ??? L^+ : ???
Será a notação de conjuntos adequada para especificar linguagens ?

1.2. Gramáticas

Uma ferramenta para estudar matematicamente linguagens, muito poderosa, que ultrapassa as limitações da notação de conjuntos anterior.

Exemplo da gramática da língua portuguesa:

Regras de produção de uma frase (ou oração) simples, em versão simplificada e incompleta, para fins ilustrativos :

⟨frase⟩ → ⟨sintagma nominal⟩ ⟨sintagma verbal⟩

⟨sintagma nominal⟩ → ⟨determinante⟩ ⟨nome⟩ | ⟨vazio⟩

⟨sintagma verbal⟩ → ⟨ verbo⟩ ⟨sintagma nominal⟩

⟨determinante⟩ → ⟨artigo⟩ | ⟨deícticos⟩ | ⟨vazio⟩

⟨verbo⟩ → estuda | ama | compra | dorme|chove

⟨artigo⟩ → o | a |um | uma|<vazio>

⟨deícticos⟩ → este | esse | aquele | meu | teu | seu |<vazio>

⟨nome⟩ → Luís | Antónia | Isabel | livro | gelado |

O Luís compra um livro : como derivar ? (*parsing*)

(ver tb Gramática da Língua Portuguesa, Mateus, M,, A. M.Brito, I. Duarte, I. H Faria, Caminho Série Linguística, 1989, p. 210

Verificar se as seguintes frases obedecem a essas regras de produção:

(i) O João vai ao cinema
(ii) Chove
(iii) Um por todos, todos por um !

Este exemplo mostra como uma frase (ou oração), conceito geral e complexo, pode ser definida à custa de elementos simples (decomposição da complexidade).

Começa-se no conceito mais complexo (frase), e reduz-se sucessivamente até se obterem os elementos irredutíveis com que se constrói a linguagem.

A generalização deste princípio leva-nos à definição de gramáticas formais.

Definição 1.1.

Uma gramática G é definida por um quadrupleto

$$G = (V, T, S, P)$$

V : **variáveis,** conjunto finito de objectos, não vazio

T : **símbolos Terminais,** conjunto finito de objectos, não vazio

$S \in V$, variável de ***inicialização (Semente, Start, Axioma)***

P : um conjunto finito de ***Produções***

V e T são disjuntos

As regras de produção

- o cerne da gramática, pois
- é através delas que se forma uma cadeia
- e que uma cadeia se transforma noutra cadeia

Por isso elas definem uma linguagem associada à gramática .

Uma linguagem pode ser produzida por várias gramáticas, mas uma gramática produz uma e uma só linguagem.

Todas as regras de produção são da forma

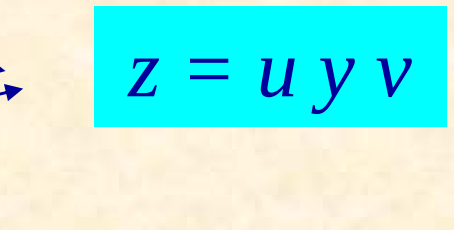
$$x \rightarrow y$$

$x \in (V \cup T)^{+}$,
cadeia de variáveis e símbolos terminais, excluindo a cadeia vazia

$y \in (V \cup T)^{*}$,
cadeia de variáveis e símbolos terminais, podendo incluir a cadeia vazia.

Modo de produção:

- dada $w = u\ x\ v$
- dada $x \rightarrow y$



por substituição de *x* *por* *y* em *w*, obtém-se a cadeia *z*

- *w* **produz** *z* , ou *z* **é produzida** *por w*.

$$w \Rightarrow z$$

As regras de produção podem aplicar-se por **qualquer ordem** e **tantas vezes quantas as necessárias** !!!

Se

$$w_1 \Rightarrow w_2 \Rightarrow \ldots \Rightarrow w_n$$

diz-se que w_1 *produz* w_n ; em escrita mais compacta

$$w_1 \overset{*}{\Rightarrow} w_n$$

em que * indica que para derivar w_n de w_1 são necessários um número não especificado de passos (incluindo zero passos, como em $w_1 \overset{*}{\Rightarrow} w_1$).

- As regras de produção podem ser aplicadas numa **ordem qualquer** e um **número qualquer** de vezes.
- Por cada ordem e por cada número, produzem uma cadeia terminal.

O conjunto de todas as cadeias terminais que se podem obter pela gramática, forma a **linguagem gerada pela gramática**, $L(G)$

$$G = (V, T, S, P)$$

$$L(G) = \{ w \in T^* : S \overset{*}{\Rightarrow} w \}$$

Se $w \in L(G)$, então a sequência

$$S \Rightarrow w_1 \Rightarrow w_2 \Rightarrow \ ... \Rightarrow w_n \Rightarrow w$$

é uma **derivação** da frase w.

Formas sentenciais da derivação de w são as cadeias

$$S, w_1, w_2, ..., w_n$$

As formas sentenciais contêm variáveis e símbolos terminais (nalguns casos só contêm variáveis).

Exemplo: $G=(\{S\},\{a,b\}, S, P)$

Variáveis: S
Símbolos terminais: a,b.
Variável de inicialização: S

Produções:

$P_1: \quad S \rightarrow a\ S\ b$
$P_2: \quad S \rightarrow \lambda$

$$S \Rightarrow a\ S\ b \Rightarrow a\ a\ S\ b\ b \Rightarrow a\ a\ a\ S\ b\ b\ b \Rightarrow \ldots$$

Em qualquer altura se pode aplicar a produção P_2. Depois da

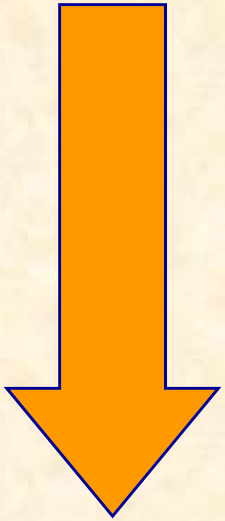
- 1ª produção obtém-se $a\ b$
- 2ª produção obtém-se $a\ a\ b\ b$
- n-èsima produção $a\ a\ a....a\ b\ b\ b...b = a^n\ b^n$

Se aplicarmos primeiro a produção P_2 obtém-se a cadeia vazia

$$G=(\{S\},\{a,b\}, S, P)$$

$$P_1: S \rightarrow aSb$$

$$P_2: S \rightarrow \lambda$$



$$L(G) = \{a^n b^n, n \geq 0\}$$

Exemplo: Encontrar uma gramática que gere a linguagem

$$L = \{ a^{n+1} b^{n}, n \geq 0 \}$$

Relativamente ao exemplo anterior, é necessário gerar um *a* adicional. Pode-se gerá-lo em primeiro lugar, aplicando depois regras de produção como no caso anterior

P_1: $S \rightarrow a\ S$
P_2: $S \rightarrow a\ S\ b$
P_3: $S \rightarrow \lambda$

Antes
P_1: $S \rightarrow a\ S\ b$
P_2: $S \rightarrow \lambda$

$G = (\{S\}, \{a,b\}, S, P)$

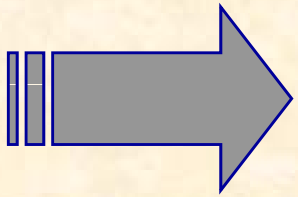
Aplicando P_1, P_1, P_2, P_3,

$$S \Rightarrow a\ S \Rightarrow a\ a\ S \Rightarrow a\ a\ a\ S\ b \Rightarrow a\ a\ a\ b$$

i.e. a^3b, que não faz parte da linguagem (porquê ?)

Para que isto não aconteça, P_1 só se pode aplicar uma vez. Para isso introduz-se uma variável adicional, A, e definem-se as produções de modo que S só possa produzir uma vez :

$$P_1: \quad S \to a\,S$$
$$P_2: \quad S \to a\,S\,b$$
$$P_3: \quad S \to \lambda$$



$$P_1: \quad S \to a\,A$$
$$P_2: \quad A \to a\,A\,b$$
$$P_3: \quad A \to \lambda$$

$$G = (\{A, S\}, \{a, b\}, S, P)$$

em que S é a variável de inicialização. Resulta a gramática G.

Para demonstrar que uma dada linguagem L é gerada por uma certa gramática G, tem que se mostrar que:

(i) toda a cadeia $w \in L$ pode ser derivada a partir de S usando as produções P da gramática G,

(ii) qualquer cadeia gerada pela gramática G pertence à linguagem L.

Uma gramática de palíndromos (PAL) sobre um alfabeto Σ

Pode-se definir um palíndromo pelas três regras seguintes:

1. $\lambda \in$ PAL
2. Para todo o carácter $a \in \Sigma$, $a \in$ PAL
3. Para toda a $w \in$ PAL , e todo o $a \in \Sigma$, $awa \in$ PAL

Uma *w* só pertence a PAL se puder ser gerada por estas 3 regras

Exemplo $\Sigma = \{ a,b \}$

1. $\lambda, a, b \in PAL$;
2. Para qualquer $w \in$ PAL, awa e $bwb \in$ PAL

Por exemplo:

$b \in$ PAL ,	pela regra 1
$aba \in$ PAL,	pela regra 2
$babab \in$ PAL	pela regra 2
$abababa \in$ PAL	pela regra 2

Uma gramática para esta linguagem

1. $S \rightarrow \lambda$ ou a ou b (regra 1)
2. $S \rightarrow aSa$ ou bSb (regra 2)

Por exemplo: $S \Rightarrow aSa \Rightarrow abSba \Rightarrow ab\lambda ba = abba$

Temos assim as produções da Gramática dos palíndromos

$$S \rightarrow a \mid b \mid \lambda \,,$$

$$S \rightarrow aSa \mid bSb$$

ou

$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow a$$
$$S \rightarrow b$$
$$S \rightarrow \lambda$$

em que '|' significa "ou" e S é o símbolo inicial.

$$G=(\{S\},\{a,b\}, S, P)$$

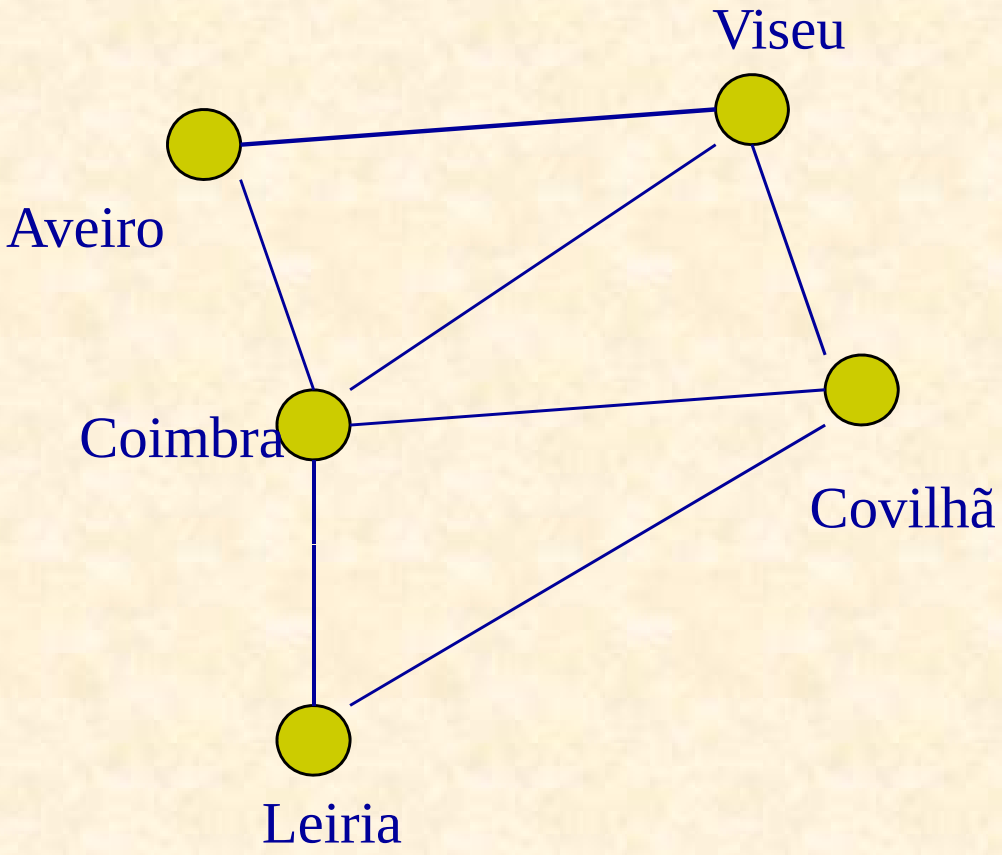
Notas sobre grafos e árvores

1. Grafos

Grafos são estruturas discretas compostas por

- **vértices**, ou nós
- **arestas** , ligando os nós

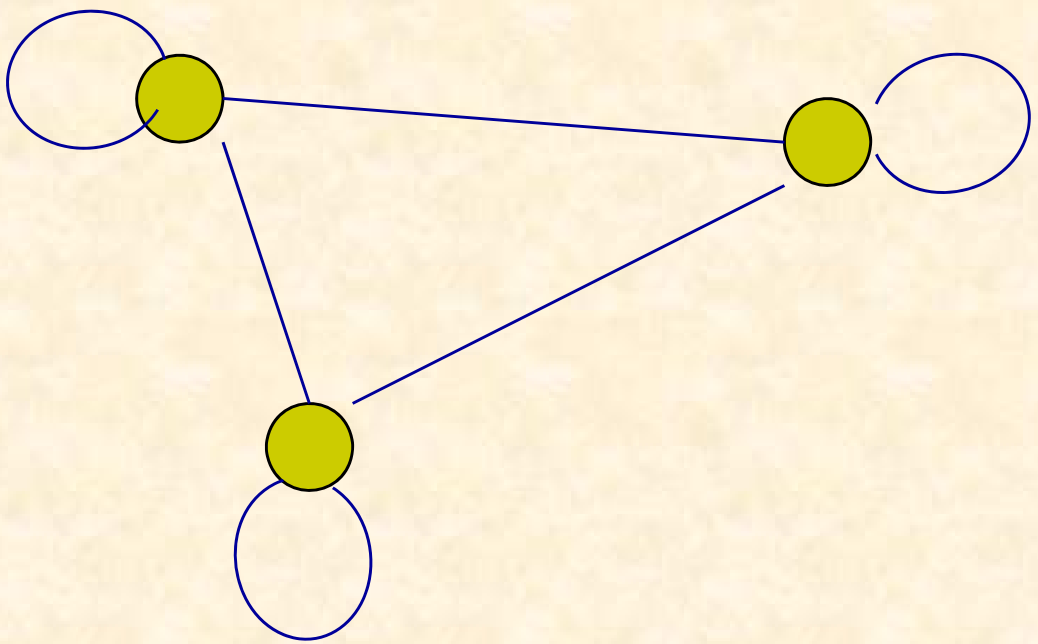
Ver Caps7 e 8 , *Discrete Mathematics & Its Applications*, Kenneth Rosen



Por hipótese: rede de fibra óptica na região centro.

Os grafos definem uma relação binária (entre dois argumentos) e podem ser de muitos tipos.

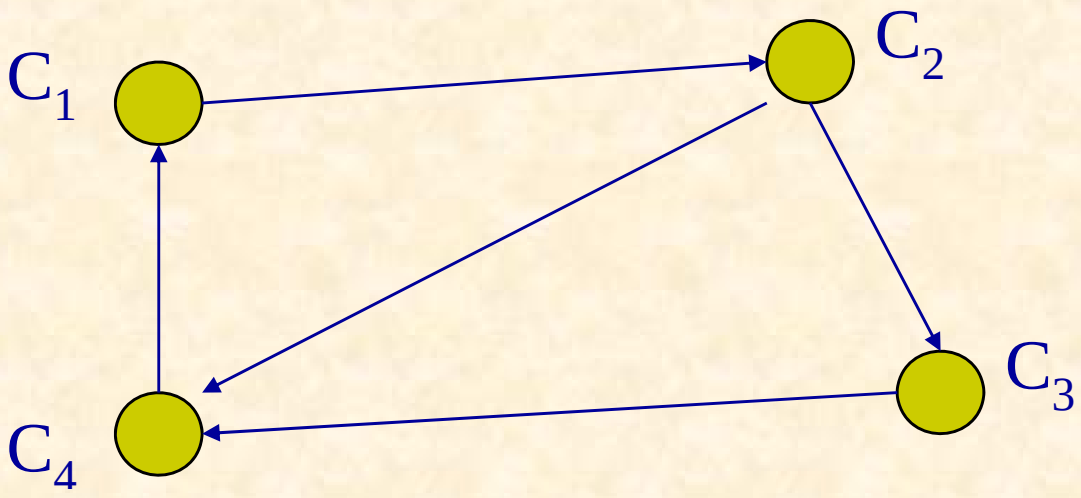
Grafos com anéis (loops):



3 computadores ligados por linhas telefónicas. Cada um tem uma linha ligada a si mesmo, para diagnóstico, por exemplo.

Grafos orientados:

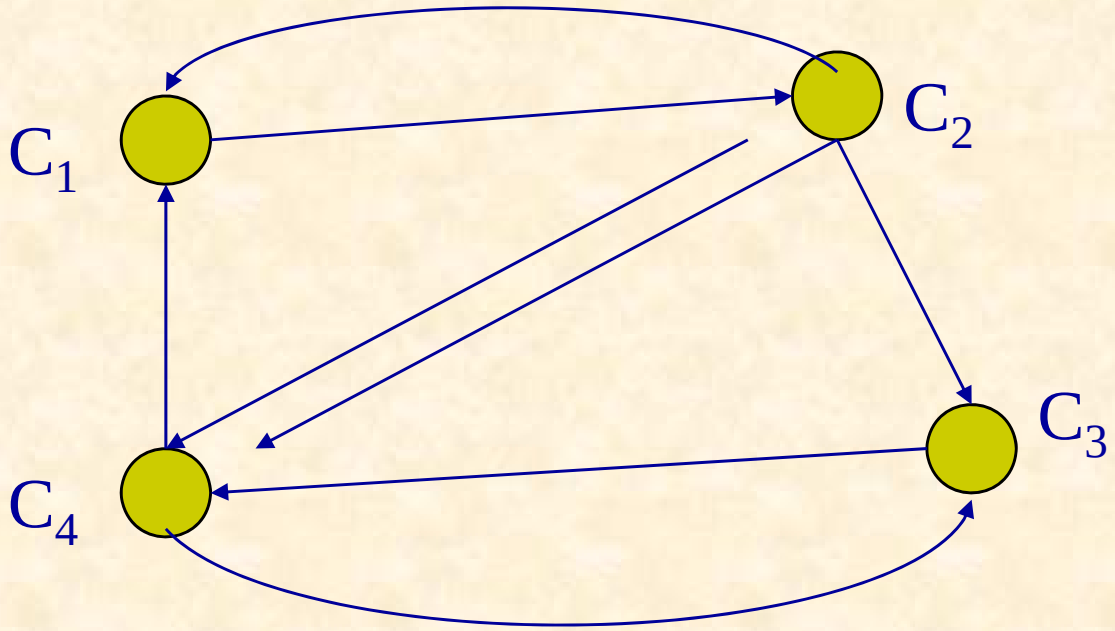
as arestas têm um sentido indicado por uma seta



Rede de computadores com comunicação unidireccional

Multigrafos:

Grafos com várias arestas entre cada par de vértices. Podem ser ou não orientados.



Terminologia básica de grafos:

$G = (V, E)$: Grafo

V : conjunto não vazio de vértices

E : conjunto de pares de elementos distintos de V, chamados vértices (ex. $\{v_1, v_2\}$)

Vértices adjacentes (ou vizinhos): ligados por uma aresta.
Os vértices *u* e *v* são adjacentes se existir $\{u, v\}$.

A aresta $\{u, v\}$ diz-se **incidente** com os vértices *u* e *v*, que são os seus **pontos terminais.**

Se o grafo é orientado de *u* para *v* , *u* é adjacente a *v*, e *v* é adjacente a partir de *u*.

Grau de um vértice num grafo não orientado: número de arestas incidentes com ele

Vértice isolado: vértice de grau zero.

Vértice pendente: vértice de grau 1

Grafo completo : contém exactamente uma aresta entre cada par de vértices (grafo não orientado).

Ciclo (*cycle*) , consiste num conjunto de três ou mais vértices e arestas formando um caminho fechado (parte e chega ao mesmo vértice)

2. Árvores

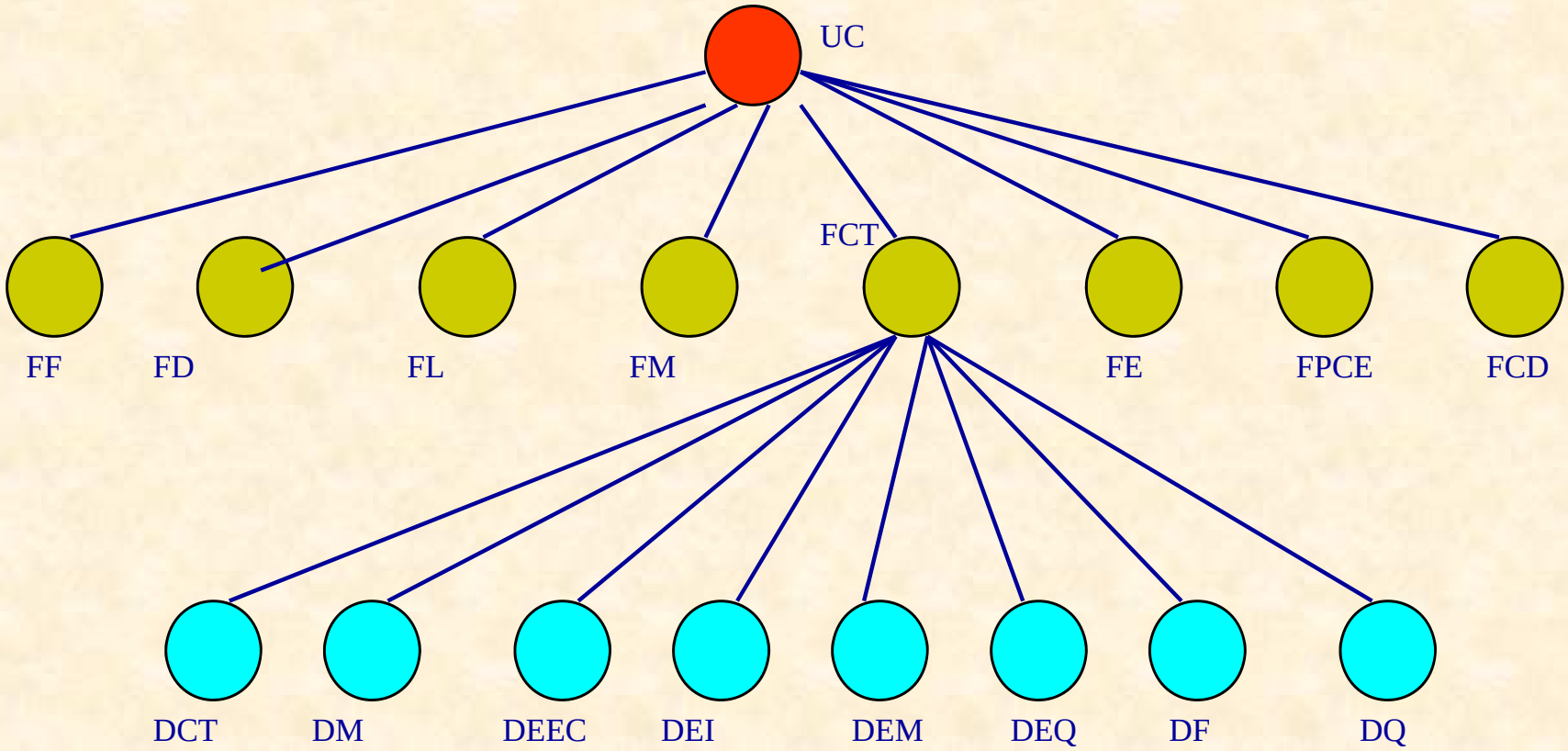
Tipos especiais de grafos não orientados sem ciclos

Exemplos:

Árvore genealógica

Representação da estrutura de ficheiros de um computador (directórios, sub-directórios, etc.).

Representação orgânica da Universidade

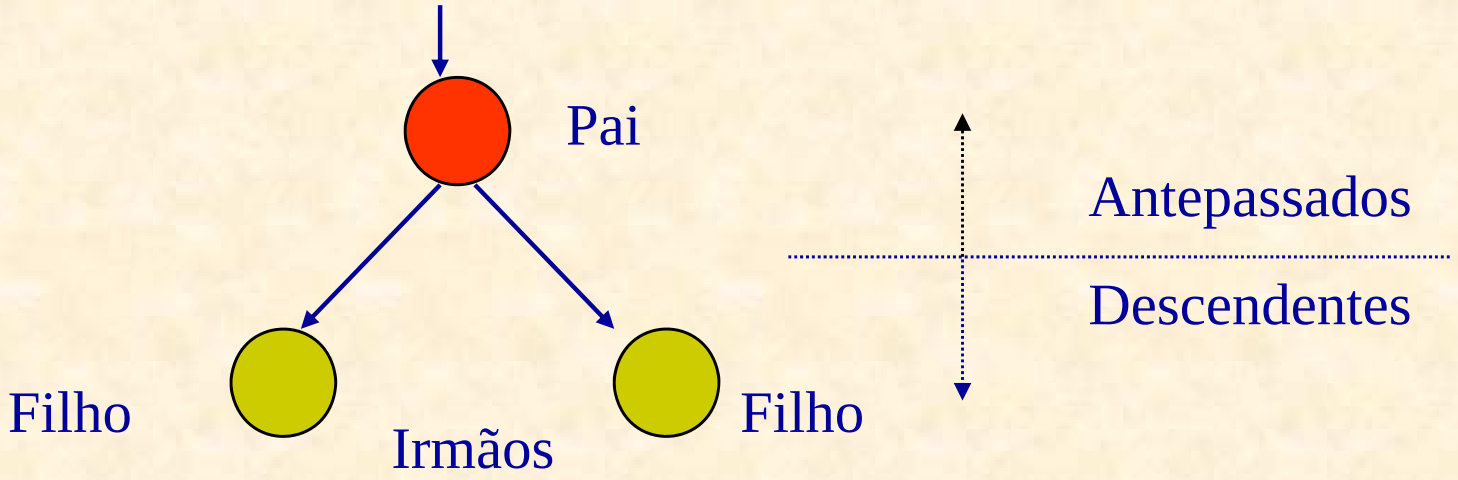


Entre quaisquer dois vértices de um árvore existe um e um só caminho.

Se o grafo da árvore é orientado, a árvore é orientada.

Raiz da árvore: vértice especial de árvores orientadas. Dele parte uma aresta orientada, produzindo uma **árvore com raiz.**

Normalmente desenha-se a árvore com a raiz no ponto mais alto.



Antepassados ou antecedentes de um vértice: todos os que o antecedem até à raiz (incluindo esta).

Descendentes de um vértice: todos os que o têm como antepassado.

Folha da árvore: vértice sem filhos.

Vértices internos: os que têm filhos

Sub-árvore de um vértice: a árvore que tem esse vértice como raiz e contém todos os seus descendentes e todas as arestas incidentes nesses descendentes.

Árvore *n* – ária: todos os vértices internos não têm mais de *n* filhos. É *n* –ária **completa** se todos os vértices internos tiverem exactamente *n* filhos.

Árvore binária: se $n = 2$

Árvore ordenada com raiz: os filhos de cada vértice interno são ordenados da esquerda para a direita. Se é binária, chamam-se **filho esquerdo** e **filho direito.**

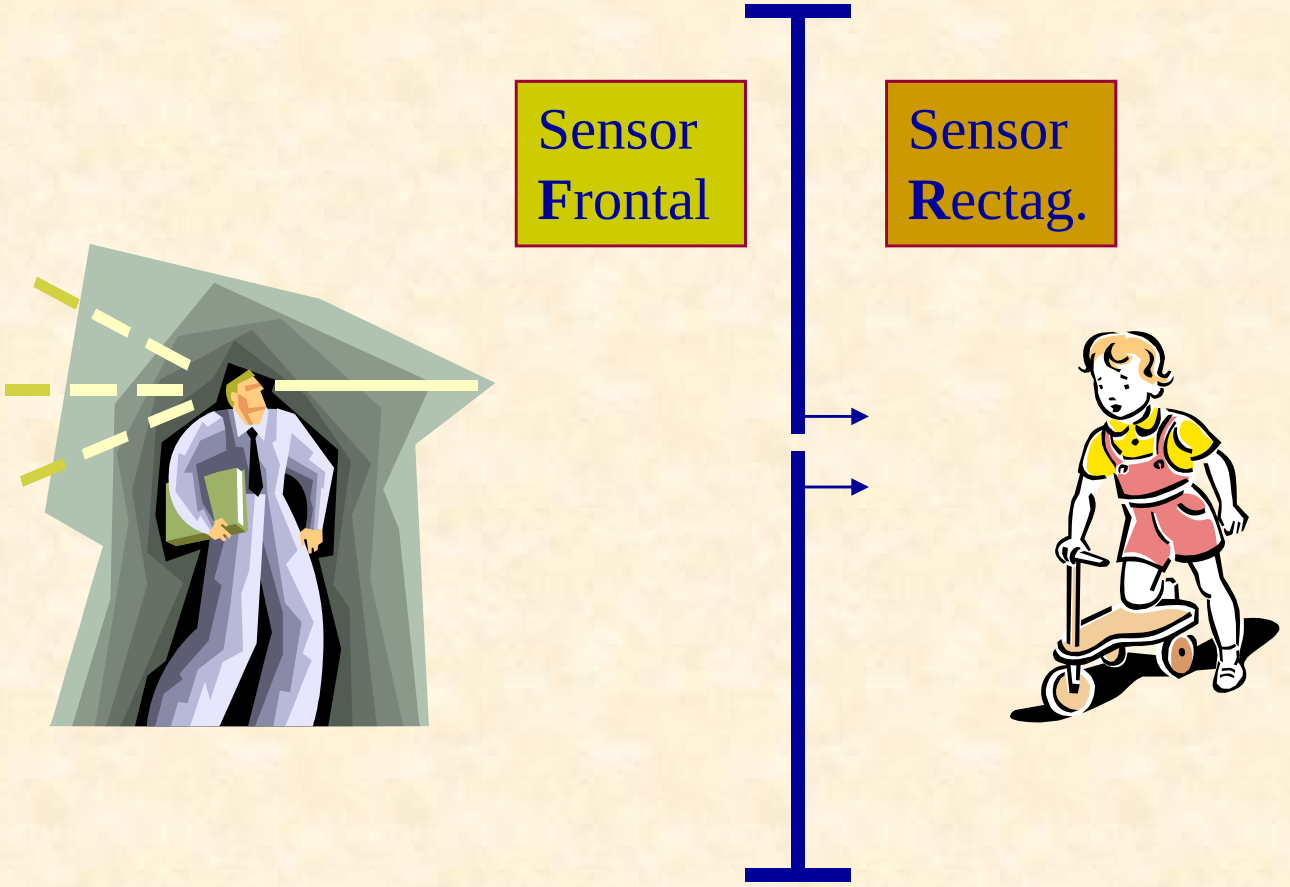
O filho da esquerda origina a **sub-árvore esquerda** e o da direita a **sub-árvore direita.**

Nível de um vértice: número de vértices existentes no caminho desde a raiz até ao vértice.

Altura uma árvore: o maior nível de todos os seus vértices.

1.3. Autómatos

Porta automática (só para entrar)



Adaptado de Sipser, p.32

Estados do controlador: aberta (**A**), fechada (**F**)

Sinais de controlo: presença/ausência frontal (**PF**) 1 ou 0
presença/ausência rectaguarda (**PR**) 1 ou 0

Alfabeto do autómato

PF	PR		Carácter
0	**0**	**00**	**a**
0	**1**	**01**	**b**
1	**0**	**10**	**c**
1	**1**	**11**	**d**

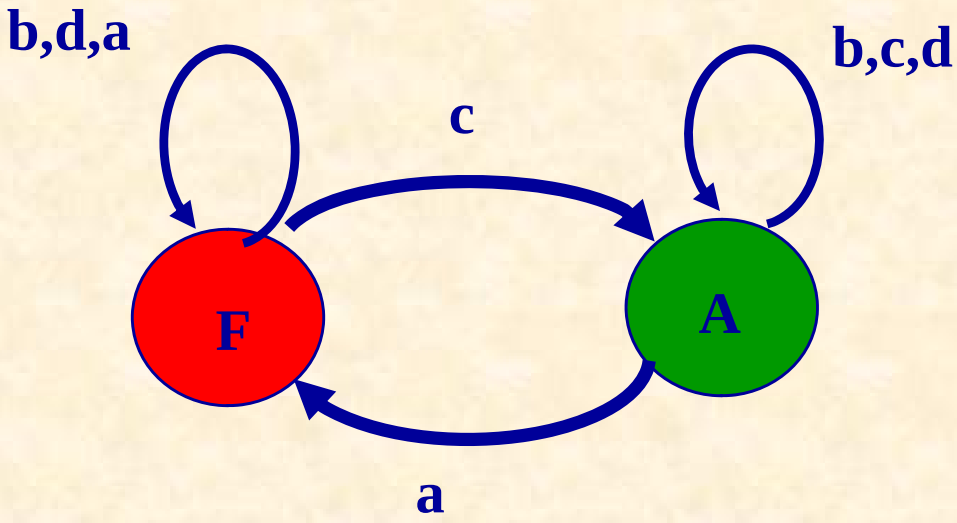
Estados

Estado	Símbolo
Aberta	**A**
Fechada	**F**

Funcionamento

Entrada → / Estado ↓	a 00	b 01	c 10	d 11
F	F	F	A	F
A	F	A	A	A

Diagrama de estados (grafo)



Autómato finito: um só bit de memória (F,A)

Autómatos

- modelo abstracto de um computador digital.

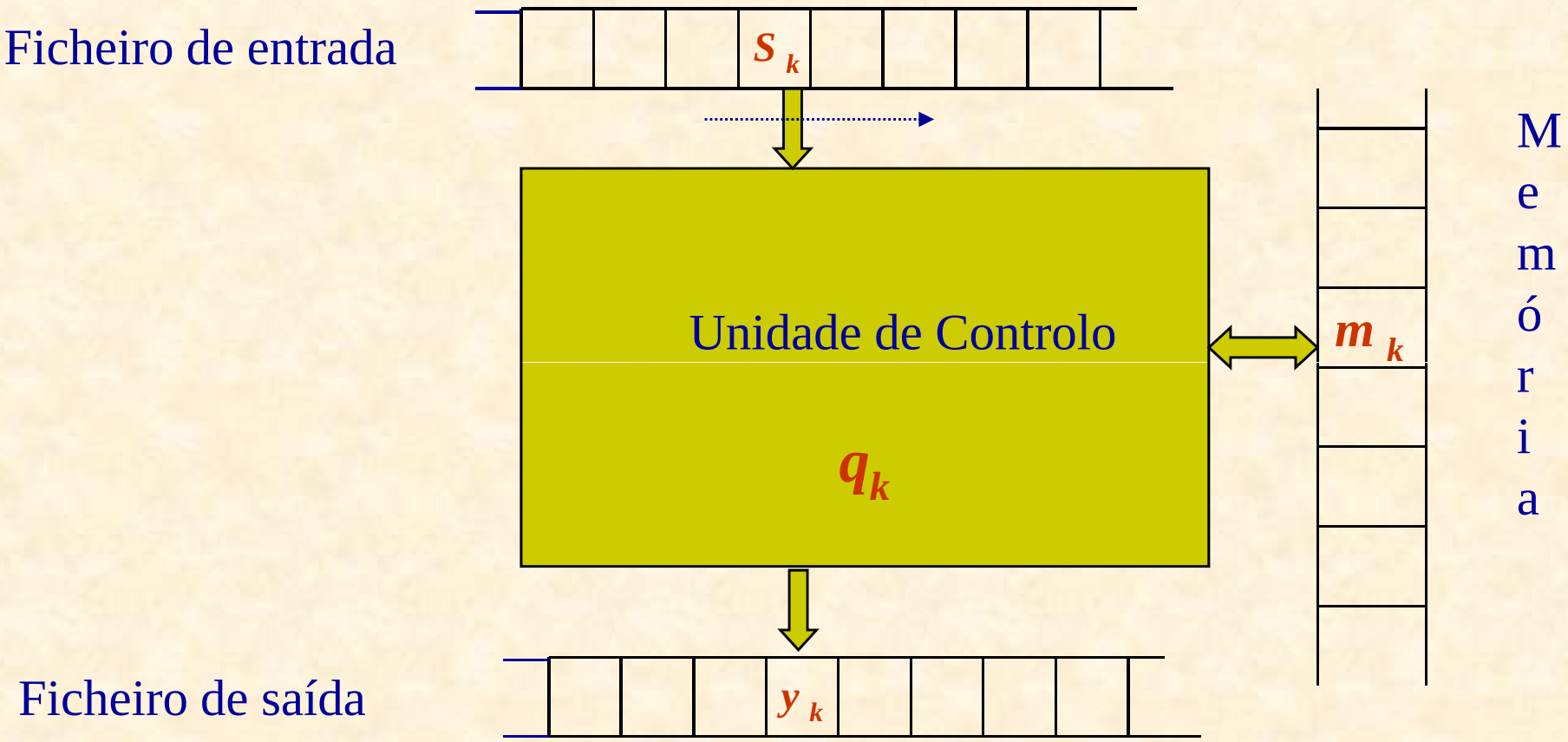
- componentes:

  - um **ficheiro de entrada**, cadeia num alfabeto
  - um mecanismo para leitura de entradas
  - um dispositivo de **memória temporária**. Pode ler e alterar o conteúdo dos registos de memória.
  - uma **unidade de controlo**, que pode estar num entre um certo número finito de **estados internos**
  - um **ficheiro de saída**

- pode mudar de estado segundo alguma regra.

Configuração **geral** de um autómato

Cada tipo tem a sua configuração própria



Função de transição de estado ***f***

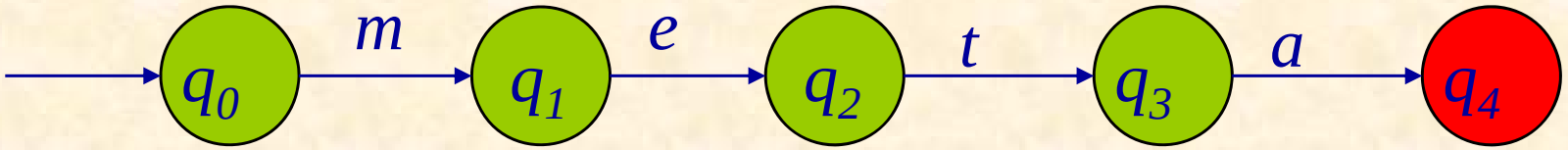
$$q_{k+1} = f\,(q_k, s_k, m_k)$$

Função de transição da saída ***g***

$$y_{k+1} = g\,(q_k, s_k)$$

Exemplo

Alfabeto de entrada : {*a*, *b*, *c*, ..., *z* }



Configuração do autómato

conjunto composto por

- o estado interno da unidade de controlo
- o ficheiro de entrada
- o conteúdo da memória
- o conteúdo da saída (em alguns tipos)

Chama-se um **passo ou movida** à transição do autómato de uma configuração para a configuração seguinte.

Os tipos de autómatos distinguem-se por

A. Forma de produzir a saída

- **aceitadores** : reconhecem ou não a cadeia de entrada - saída respectivamente *sim* ou *não*.
- **transdutores** : produzem cadeias de caracteres à saída.

B. Natureza da memória temporária (o factor mais importante)

- **autómatos finitos**, sem memória temporária
- **autómatos de pilha** (*pushdown automata*) de memória em pilha
- **máquinas de Turing**: de memória em fita infinita podendo ser lida e alterada em qualquer ordem.

C. Natureza da função de transição

- **determinísticos**, dada uma configuração, existe um e um só comportamento futuro possível;

- **não determinísticos** : dada uma configuração, o próximo passo pode ter várias alternativas.

1.4. Os três paradigmas da computação

1.4.1. Computação (do valor) de funções numéricas

$f(n)$ = o n-ésimo número primo (função unária)

$g(n,m) = n \times m$ (produto, função binária)

$h(a_1,\ldots, a_n)=a_1+\ldots+a_n$ (somatório, função n-ária)

1.4.2. Reconhecimento de linguagens (ou classificação de cadeias de símbolos)

- *ababbaba* é um palíndromo ?

- em 10011000111
  o número de 0's é igual ao número de 1's ?

Qualquer problema de decisão se pode transformar num problema de reconhecimento de linguagens.

Qualquer problema de decisão se pode transformar num problema de reconhecimento de linguagens.

Exemplo:

- O número natural *n* é primo ? (problema de decisão)

Linguagem $L = \{1^n, n \text{ primo}\}$ $1^n = 111\ldots1$ *n* vezes

$L = \{11, 111, 11111, 1111111, \ldots\}$

- A cadeia 1^n pertence à linguagem L ? (problema de reconhecimento de linguagens)

1.4.3. Transdução (transformação) de cadeias de símbolos

reverter

atrasar (*delay*)

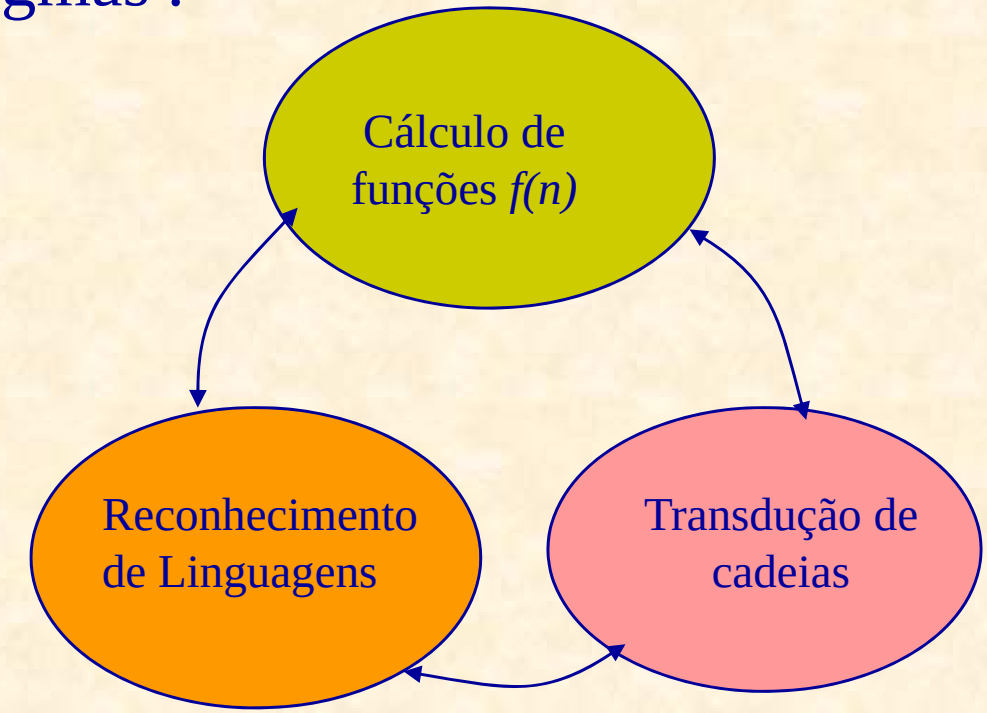
combinar símbolos
identificar símbolos

etc …

Não há um paradigma mais importante !!!

Princípio da inter-redutibilidade:

- qualquer instância de um dos três paradigmas pode ser olhada como uma instância de um dos outros dois paradigmas .



Qualquer caso de computação de uma função pode ser reduzido a uma instância do reconhecimento de uma linguagem.

Exemplo:

$f(n) = m$, m é o n-ésimo número primo

$f(3) = ??$

n	$f(n)$
1	2
2	3
3	5
4	7
..	..

Nota: números primos

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149
151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241 251 257 263 269 271 277 281 283 293
307 311 313 317 331 337 347 349 353 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461
463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587 593 599 601 607 613 617 619 631 641
643 647 653 659 661 673 677 683 691 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823
827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947 953 967 971 977 983 991 997 1009 1013
1019 1021 1031 1033 1039 1049 1051 1061 1063 1069 1087 1091 1093 1097 1103 1109 1117 1123 1129 1151 1153
1163 1171 1181 1187 1193 1201 1213 1217 1223 1229 1231 1237 1249 1259 1277 1279 1283 1289 1291 1297 1301
1303 1307 1319 1321 1327 1361 1367 1373 1381 1399 1409 1423 1427 1429 1433 1439 1447 1451 1453 1459 1471
1481 1483 1487 1489 1493 1499 1511 1523 1531 1543 1549 1553 1559 1567 1571 1579 1583 1597 1601 1607 1609
1613 1619 1621 1627 1637 1657 1663 1667 1669 1693 1697 1699 1709 1721 1723 1733 1741 1747 1753 1759 1777
1783 1787 1789 1801 1811 1823 1831 1847 1861 1867 1871 1873 1877 1879 1889 1901 1907 1913 1931 1933 1949
1951 1973 1979 1987 1993 1997 1999 2003 2011 2017 2027 2029 2039 2053 2063 2069 2081 2083 2087 2089 2099
2111 2113 2129 2131 2137 2141 2143 2153 2161 2179 2203 2207 2213 2221 2237 2239 2243 2251 2267 2269 2273
2281 2287 2293 2297 2309 2311 2333 2339 2341 2347 2351 2357 2371 2377 2381 2383 2389 2393 2399 2411 2417
2423 2437 2441 2447 2459 2467 2473 2477 2503 2521 2531 2539 2543 2549 2551 2557 2579 2591 2593 2609 2617
2621 2633 2647 2657 2659 2663 2671 2677 2683 2687 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741 2749
2753 2767 2777 2789 2791 2797 2801 2803 2819 2833 2837 2843 2851 2857 2861 2879 2887 2897 2903 2909
… … …

Usando a representação binária de números naturais, constroem-se as cadeias que resultam da concatenação de 11 (3) com os números naturais (1, 2, 3, 4, 5, 6, …), usando por exemplo # como separador:

11#1
11#10
11#11
11#100
11#101
11#110
…

Define-se agora a linguagem L associada aos números primos

L ={ Binário(*n*)#Binário(*m*)}

A computação do valor de f(3) é equivalente à determinação de qual a única cadeia daquele conjunto que pertence a L .

Como se pode reduzir uma classificação de cadeias a uma computação numérica de uma função ?

Tomemos o caso do reconhecimento de palíndromos:

- Em primeiro lugar faz-se uma codificação dos símbolos do alfabeto, de modo que cada símbolo fique associado a um e um só número natural. Por exemplo se a = 01 e b = 10, então *aba* resulta no número natural 011001=25

- Em segundo lugar constrói-se a linguagem dos palíndromos, $L(Pal)$, e os seus códigos.

A função característica de L será

$\chi(n) = 1$, se n é o código de um palíndromo
$= 0$, se não o é.

- Agora para se saber se 25 é o código de um palíndromo, basta calcular o valor da função característica $\chi(25)= 1$, e assim se conclui que *aba* é um palíndromo.

O princípio da inter-redutibilidade permite que se use o problema da identificação de linguagens em teoria da computação como representativo dos três paradigmas.

Utilidade da teoria dos autómatos finitos

Modelos para...

- Software de projecto e implementação de circuitos digitais (os transdutores)
- Analisador lexical em compiladores (o componente do compilador que divide o texto em unidades lógicas, tais como identificadores, palavras-chave e pontuação)
- Software para analisar grandes quantidades de texto (páginas *web*, procurar ocorrências de palavras, de frases, etc.)
- Software para a verificação de protocolos de comunicação, protocolos de segurança, ou qualquer tipo sistemas com um número finito de estados

Bibliografia

An Introduction to Formal Languages and Automata, Peter Linz, 3rd Ed., Jones and Bartelett Computer Science, 2001
(livro mais seguido na cadeira)

Models of Computation and Formal Languages, R. Gregory Taylor, Oxford University Press, 1998.

Introduction to Automata Theory, Languages and Computation, 2nd Ed., John Hopcroft, Rajeev Motwani, Jeffrey Ullman, Addison Wesley, 2001.

Elements for the Theory of Computation, Harry Lewis and Christos Papadimitriou, 2nd Ed., Prentice Hall, 1998.

Introduction to the Theory of Computation, Michael Sipser, PWS Publishing Co, 1997.