

# Teoria da Computação

## Problemas, Programas, Linguagens

Simão Melo de Sousa

Computer Science Department  
University of Beira Interior, Portugal

# Plano

- 1 Introdução
- 2 Problemas e Procedimentos Efectivos
- 3 A Formalização dos Problemas
- 4 Representação dos Problemas
- 5 Linguagens
- 6 Técnicas de Demonstração

# Plano

- 1 Introdução
- 2 Problemas e Procedimentos Efectivos
- 3 A Formalização dos Problemas
- 4 Representação dos Problemas
- 5 Linguagens
- 6 Técnicas de Demonstração



# Motivação

- Perceber os limites da informática
- Distinguir os problemas resolúveis algoritmicamente dos que não o são
- Obter resultados independentes da tecnologia utilizada para construir computadores

# Plano

- 1 Introdução
- 2 Problemas e Procedimentos Efectivos**
- 3 A Formalização dos Problemas
- 4 Representação dos Problemas
- 5 Linguagens
- 6 Técnicas de Demonstração

# Problemas

- Que problemas podem ser resolvidos por um programa executados num computador?

Para responder a tal pergunta é preciso formalizar e definir

- a noção de problema
- a noção de programa e da sua execução por um computador

Poderemos assim determinar com toda a precisão (i.e. matematicamente) o que é um algoritmo e qual é o seu poder expressivo. Ou seja: o que um algoritmo pode resolver e o que não pode.



# A noção de Problema

**O que é um problema?** é uma questão **genérica**

- Dado um vector, podemos ordenar os seus elementos de forma crescente?
- Qualquer que seja um grafo e dois nodos deste grafo, é possível determinar o caminho mais curto entre estes dois nodos?
- Existe uma forma de saber se um programa qualquer termina (problema da paragem)?
- Dada uma equação polinomial de coeficientes inteiros, é possível determinar as suas soluções inteiras (décimo problema de Hilbert)?

**O que é uma instância de um problema?** Um caso particular dum problema. Por exemplo a ordenação do vector  $[[1; 7; 2; 6; 4; 5]]$  é uma instância do problema da ordenação.



# A noção de Programa

**Programa, algoritmo, procedimento efectivo:** uma descrição rigorosa, completa, finita (em tempo e em tamanho) e determinística dum processo de resolução que quando submetido a um mecanismo de execução permite determinar sistematicamente a solução dum instancia dum problema.

- um programa Java é um procedimento efectivo
- o algoritmo de euclides, o crivo de erastostene, o quicksort, são procedimentos efectivos



## Problemas e procedimentos efectivos

Como já vimos, nem todos os problemas tem procedimentos efectivos associados. Isto é, nem todos tem solução algorítmica.

- Os dois primeiros problemas tem solução algorítmica, os dois últimos não. Diz-se, neste caso, que são **indecidíveis**.
- As funções estruturalmente recursivas terminam? Sim. Mas será possível determinar a terminação de qualquer programa (não necessariamente as funções estruturalmente recursiva). Resposta: não. Veja só:
- O problema de Syracuse (ou de Collatz): A função seguinte termina?

```
let rec syracuse (n:int) =  
  if n=1 then 1 else  
    if (n mod 2 = 0) then syracuse (n/2)  
    else syracuse (3*n + 1)
```

até a data de hoje, não se sabe.



## O problema da paragem é indecidível

Vimos na aula de introdução que o problema da paragem foi demonstrado indecidível em 1936 pelo **Church** e pelo **Turing**. Repetimos aqui a demonstração informal:

- Imaginemos que exista uma função/programa termina que aceita um programa, digamos  $p$ , e que responderia em tempo finito **true** se o programa  $p$  termina e **false** se o programa  $p$  não termina.
- Consideremos agora o programa seguinte:

```
let rec sem_fim x = if x then (sem_fim x) else false
termina(sem_fim) devolve então falso.
```



## O problema da paragem é indecidível

- Consideremos também o programa seguinte:

```
let rec teste () =  
    if (termina teste) then teste () else true
```

- Que devolverá então (termina teste)?
  - Se teste não termina então (termina teste) devolve **false** e o valor de teste é **true**, logo teste termina.
  - Da mesma forma, se teste termina então (termina teste) devolve **true** e teste é de novo avaliado (devido a recursão) e entra em ciclo.

Temos aqui uma situação contraditória. A função termina não pode, infelizmente, existir: **o problema da terminação é indecidível.**

# Plano

- 1 Introdução
- 2 Problemas e Procedimentos Efectivos
- 3 A Formalização dos Problemas**
- 4 Representação dos Problemas
- 5 Linguagens
- 6 Técnicas de Demonstração

Como representar problemas, as suas instâncias, os seus parâmetros? através de algo que as possa descrever: a noção de linguagem e das suas palavras.

# Alfabeto

Alfabeto: Conjunto finito de símbolos

- $\{a, b, c\}$
- $\{\alpha, \beta, \gamma\}$
- $\{1, 2, 3\}$
- $\{\diamond, \heartsuit, \clubsuit, \spadesuit\}$
- etc...

# Palavras

- Palavras: sequência finita (eventualmente vazia) de elementos dum alfabeto.
- Monoíde livremente gerado por um alfabeto  $A$  (designado por  $A^*$ ): conjuntos de todas as palavras geradas a partir do alfabeto  $A$ . Origem do nome: algébrica.  $A^*$  é um conjunto que tem um operador  $.$  (a concatenação) associativo com um elemento neutro: a palavra vazia  $\epsilon$  (a palavra feita com 0 letras de  $A$ , i.e. de comprimento 0).
- Admite-se, por razões de conveniência, que a concatenação  $.$ , por exemplo  $a.b.c.d.e.f.g$  seja notada  $abcdefg$ .



# Palavras e Linguagens

- Linguagem  $L$  sobre um alfabeto  $A$ : Conjunto de palavras, ou seja um subconjunto de  $A^*$ .
- Define-se sobre as linguagens e sobre as palavras as seguintes operações:
  - O comprimento de uma palavra  $w$ , notada  $|w|$ , devolve o comprimento da sequência de símbolos que compõe a palavra.
  - Seja  $w$  uma palavra,  $i$  um inteiro natural tal que  $1 \leq i \leq |w|$ , então  $w.(i)$  designa o  $i$ -ésimo símbolo (letra) da sequência  $w$ .
- exemplos:  $abb3bwk217m$  é uma palavra sobre o alfabeto  $\{0, \dots, 9, a, \dots, z\}$ . Designemos por  $w$  esta palavra.  
 $w.(5) = b$ ,  $|w| = 11$ . No que diz respeito a palavra vazia,  $|\epsilon| = 0$





# Plano

- 1 Introdução
- 2 Problemas e Procedimentos Efectivos
- 3 A Formalização dos Problemas
- 4 Representação dos Problemas**
- 5 Linguagens
- 6 Técnicas de Demonstração

## Codificação dos Problemas

Consideremos um problema binário (problema cuja resposta é "sim" ou "não"), cujas instâncias estão codificadas por palavras definidas sobre um alfabeto  $\Sigma$ . O conjunto  $\Sigma^*$  de todas as palavras definidas sobre  $\Sigma$  pode ser particionado em 3 sub-conjuntos:

- as instâncias positivas: palavras que representam instancias do problema e para as quais a resposta ao problema é positiva (sim)
- as instâncias negativas: palavras que representam instancias do problema e para as quais a resposta ao problema é negativa (não)
- palavras que não são nem instâncias positivas nem instâncias negativas (não são instâncias do problema)



## Um exemplo

Por exemplo, o problema da satisfação de fórmulas lógicas proposicionais ( $\mathcal{V}$  = conjunto de variáveis proposicionais).

- Alfabeto:  $A = \mathcal{V} \cup \{\top, \perp, \wedge, \vee, \rightarrow, \leftrightarrow, \neg, (, )\}$
- Linguagem: o conjunto indutivo  $\mathcal{Prop}$  das fórmulas proposicionais definido sobre  $A^*$  por
  - (B)  $\forall v \in \mathcal{V}, v \in \mathcal{Prop}$
  - (B)  $\top \in \mathcal{Prop}, \perp \in \mathcal{Prop}$
  - (I)  $\forall F, G \in \mathcal{Prop}, \neg F, (F \vee G), (F \wedge G), (F \rightarrow G), (F \leftrightarrow G) \in \mathcal{Prop}$



## Um exemplo

- O problema: Dado uma fórmula, será esta uma tautologia?
- Conjunto de todas as palavras:  $A^*$
- As instâncias:  $\mathcal{P}rop$
- As instâncias positivas: as fórmulas de  $\mathcal{P}rop$  que são tautologias
- As instâncias negativas: as fórmulas de  $\mathcal{P}rop$  que não são tautologias
- $A^* - \mathcal{P}rop$  é o conjunto das palavras que não são instâncias do problema.

Por exemplo " $((\forall A$ " é uma palavra da linguagem, isto é: " $((\forall A \in A^*$ , mas não é uma instancia do problema da satisfação (porque nem sequer é uma formula lógica)

# Plano

- 1 Introdução
- 2 Problemas e Procedimentos Efectivos
- 3 A Formalização dos Problemas
- 4 Representação dos Problemas
- 5 Linguagens**
- 6 Técnicas de Demonstração

## Operações sobre Linguagens

- A linguagem vazia (notação  $\emptyset$ ), não é igual a linguagem que só tem a palavra vazia (ou seja  $\{\epsilon\}$ ).
- Sejam  $L_1$  e  $L_2$  duas linguagens.
  - $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$
  - $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$
  - $L_1.L_2 = \{w \mid w = x.y, x \in L_1 \wedge y \in L_2\}$
  - $L_1^* = \{w \mid \exists k \geq 0, w_1 \dots w_k \in L_1, w = w_1.w_2 \dots w_k\}$
  - $L_1^+ = \{w \mid \exists k > 0, w_1 \dots w_k \in L_1, w = w_1.w_2 \dots w_k\}$
  - $\overline{L_1} = \{w \mid w \notin L_1\}$



## Linguagens regulares

$\mathcal{R}$ , o conjunto das linguagens regulares sobre um alfabeto  $\Sigma$ , é definido por indução, isto é, como o menor conjunto de linguagens tais que

- (B)  $\emptyset \in \mathcal{R}$ ,  $\{\epsilon\} \in \mathcal{R}$ ,  $\forall a \in \Sigma, \{a\} \in \mathcal{R}$
- (I)  $\forall A, B \in \mathcal{R}, A \cup B \in \mathcal{R}, A.B \in \mathcal{R}, A^* \in \mathcal{R}$



## Expressões Regulares

Nem é sempre cómodo tratar as linguagens como conjuntos de palavras. Uma outra abordagem consiste em agrupar palavras consoante os padrões que essas apresentam: as expressões regulares. Definição indutiva da noção de expressão regular:

- Definido sobre o monoíde livremente gerado por o alfabeto  $\Sigma$  seguinte:  $(\Sigma \cup \{\}, (, \emptyset, +, *, \epsilon)^*$
- (B) elementos de base:  $\emptyset, \epsilon, \forall x \in \Sigma$
- (I) se  $a, b$  são expressões regulares, então  $(ab)$ ,  $(a + b)$ ,  $(a)^*$  são igualmente expressões regulares.

Ou seja: Linguagens são conjuntos, Expressões regulares são uma notação. Veremos que são uma boa notação para as linguagens regulares.



## Expressões regulares em OCaml

```
type 'a expreg =  
  Vazia           (* Linguagem vazia *)  
  Epsilon        (* Palavra vazia *)  
  Caracter       of 'a      (* Caracter c *)  
  Uniao          of expreg * expreg (* r1 + r2 *)  
  Produto        of expreg * expreg (* r1r2 *)  
  Estrela        of expreg      (* r* *)
```



## Expressões regulares e linguagens

Convém agora relacionar a notação com os conjuntos.

Seja  $r$  uma expressão regular, designamos por  $L(r)$  a linguagem definida por

- se  $r = \emptyset$ ,  $L(r) = \emptyset$
- se  $r = \epsilon$ ,  $L(r) = \{\epsilon\}$
- se  $r = a$  (com  $a \in \Sigma$ ),  $L(r) = \{a\}$
- se  $r = (a + b)$ ,  $L(r) = L(a) \cup L(b)$
- se  $r = (ab)$ ,  $L(r) = L(a).L(b)$
- se  $r = (a)^*$ ,  $L(r) = L(a)^*$

Esta linguagem é dita **gerada pela expressão regular**  $r$ .



# Teorema

**Uma linguagem é regular (LR) se e só se pode ser representado por uma expressão regular (ER)**

Demonstração?

- $(LR) \implies (ER)$  Por indução estrutural sobre a definição duma linguagem regular
- $(ER) \implies (LR)$  Por indução estrutural sobre a definição duma expressão regular



# Teorema

## Uma linguagem é regular se e só se pode ser representado por uma expressão regular

Demonstração de  $(ER) \implies (LR)$ ? (esqueleto)

Por indução estrutural sobre a definição duma expressão regular

- Demonstrar que as linguagens geradas pelas expressões regulares de base são regulares
  - 1  $L(\emptyset)$  é regular por definição, QED (do latim *Quod Erat Demonstrandum*).
  - 2  $L(\epsilon)$  é regular por definição, QED.
  - 3  $\forall a \in A, L(a) = \{a\}$  é regular, por definição, QED.
- Sejam  $a$  e  $b$  são duas expressões regulares. Admitimos que  $L(a)$  e  $L(b)$  são regulares. Serão  $L(a^*)$ ,  $L(a + b)$   $L(a.b)$  regulares? Sim (por definição de linguagem regular....trivial). QED.

QED. Demonstração concluída



## Linguagens regulares

- Várias expressões regulares podem estar associadas à mesma linguagem regular.
- Seja  $\Sigma = \{a_1, \dots, a_n\}$ .  $\Sigma^*$  pode ser gerada pela expressão regular  $(a_1 + a_2 + \dots + a_n)^*$
- o conjunto das palavras não vazias geradas a partir do alfabeto  $\Sigma$ , notado  $\Sigma^+$ , é denotado por  $(a_1 + a_2 + \dots + a_n)(a_1 + a_2 + \dots + a_n)^*$  (ou seja  $\Sigma^+ = \Sigma\Sigma^*$ ).

# Linguagens regulares

- Por conveniência nota-se por
  - $r^+$  (ou  $r^+$ ) a expressão  $rr^*$ .
  - $r^?$  a expressão  $(r + \epsilon)$  ("eventualmente  $r$ ")
  - $r^n$  a expressão  $\underbrace{r \cdots r}_n$  por  $r^n$
- $(a + b)^* a (a + b)^*$  representa a linguagem das palavras que contém pelo menos um  $a$ .
- Em sintaxe BNF a expressão regular representando os números flutuantes (os "floats") define-se por:  
 $\langle \text{float} \rangle ::= -? \langle \text{digit} \rangle^+ (. \langle \text{digit} \rangle^?)? ((e | E) (+ | -)? \langle \text{digit} \rangle^+)?$



# Uma equivalência

$$(a^*b)^* + (b^*a)^* = (a + b)^*$$

Demonstração:

- $(a^*b)^* + (b^*a)^* \subseteq (a + b)^*$ . Porque  $(a + b)^*$  designa o conjunto das palavras constituídas de  $a$  e de  $b$ .



# Uma equivalência

$$(a^*b)^* + (b^*a)^* = (a+b)^*$$

Demonstração:

- Consideramos uma palavra qualquer  $w$  de  $(a+b)^*$ . Digamos  $w = w_1w_2w_3 \dots w_n$ . Podemos distinguir os 4 casos seguintes:
  - 1  $w = a^n$ , logo  $w \subseteq (\epsilon a)^* \subseteq (b^*a)^*$
  - 2  $w = b^n$ , logo  $w \subseteq (\epsilon b)^* \subseteq (a^*b)^*$
  - 3  $w$  contém  $a$  e  $b$  em quantidade arbitrária e termina por um  $b$ . Logo podemos descrever  $w$  da seguinte forma:

$$w = \underbrace{a \dots ab}_{a^*b} \underbrace{\dots b}_{(a^*b)^*} \underbrace{a \dots ab}_{a^*b} \underbrace{\dots b}_{(a^*b)^*}$$

$$\implies w \in (a^*b)^* + (b^*a)^*$$

- 4  $w$  contém  $a$  e  $b$  em quantidade arbitrária e termina por um  $a$ .  $w$  pode se descompor de forma semelhante ao ponto anterior.



# Linguagens não regulares

Questão: serão todas as linguagens, linguagens regulares?

Resposta: não. Há mais linguagens do que as linguagens regulares.

## Considerações sobre conjuntos

- Diz-se de dois conjuntos que tem a mesma cardinalidade quando existe uma bijecção entre eles. Por exemplo  $\{1, 2, 3, 4\}$  e  $\{a, b, c, d\}$  via, por exemplo, a bijecção  $\{(1, a), (2, b), (3, c), (4, d)\}$
- um conjunto  $C$  é designado de numerável quando existe uma bijecção entre  $C$  e  $\mathbb{N}$ , ou seja quando tem a cardinalidade de  $\mathbb{N}$ .
- Por exemplo, o conjunto das palavras sobre o alfabeto  $\{a, b\}$  é numerável. Porque podemos definir a bijecção seguinte:  $\{(\epsilon, 0), (a, 1), (b, 2), (aa, 3), (ab, 4), (ba, 5), (bb, 6), (aaa, 7), \dots\}$
- As expressões regulares são numeráveis. De facto as expressões regulares são palavras formadas a partir dum alfabeto finito. Logo, como no exemplo anterior, o conjunto das expressões regulares é numerável.

## O conjunto das linguagens

- Seja  $A$  um conjunto, o conjunto dos subconjuntos de  $A$  (também designado como o conjunto das partes de  $A$ ) não é numerável (ver a discussão sobre a *técnica da diagonal*).
- O conjunto das linguagens sobre um alfabeto  $\Sigma$  é o conjunto dos subconjuntos de  $\Sigma^*$ . Logo não é numerável.
- O conjunto das linguagens regulares é numerável, visto que cada linguagem está associada a pelo menos uma expressão regular (logo existe uma bijecção entre estes dois conjuntos).
- Conclusão: há mais linguagens do que linguagens regulares.



# Plano

- 1 Introdução
- 2 Problemas e Procedimentos Efectivos
- 3 A Formalização dos Problemas
- 4 Representação dos Problemas
- 5 Linguagens
- 6 Técnicas de Demonstração

## Demonstrações por contradição

Imagine que pretendemos demonstrar uma propriedade  $P$ . Podemos construir uma argumentação dedutiva que exhibe que  $P$  é válida a partir de resultados previamente demonstrado, axiomas ou até mesmo do cálculo. Ou podemos demonstrar que  $P$  não pode inválida. Este processo é designado de **demonstração por contradição**, ou ainda de **demonstração por absurdo** ou de **demonstração por redução ao absurdo** (do latim *reductio ad absurdum*)



## Demonstrações por contradição

Genericamente, o processo de demonstração por contradição duma propriedade  $P$  baseia-se na percepção de que **ou temos  $P$  ou temos  $\neg P$**  (ou exclusivo, ou seja nunca os dois simultaneamente) e conduz-se da seguinte forma:

- 1 Admitir (como hipótese) que o contrário do que pretendemos provar é válido (ou seja admitir  $\neg P$ )
- 2 condizir uma dedução que nos leva desta hipótese a uma situação contraditória (dito de outra forma absurda ou ainda impossível).
- 3 logo a hipótese inicial,  $\neg P$ , não pode ser válida. Podemos assim concluir que  $P$  é válida.



## Demonstrações por contradição - Um exemplo

$$\sqrt{2} \in (\mathbb{R} - \mathbb{Q})?$$

Demonstração por absurdo de que  $\sqrt{2}$  é irracional:

- Assumimos que  $\exists n, m \in \mathbb{N}$ , tais que  $\sqrt{2} = \frac{n}{m}$ , sendo  $\frac{n}{m}$  uma fracção irreductível ( $n$  e  $m$  não tem divisores comuns);
- ou seja  $2m^2 = n^2$ .
- logo  $n^2$  é par, ou seja  $n$  igualmente. Seja  $k \in \mathbb{N}$  tal que  $n = 2k$ , então  $2m^2 = 4k^2$ .
- assim sendo  $m^2 = 2k^2$ . O valor  $m$  é assim e igualmente par.
- Contradição.  $m$  e  $n$  são ambos divisíveis por 2 embora a hipótese inicial afirmasse que não houvesse divisores comuns.  
 **$\sqrt{2}$  só pode ser irracional.**



## Demonstrações por contradição

Um ponto subtil: nunca foi provado que  $P$  (no exemplo " $\sqrt{2}$  é irracional") seja válida, mas sim que  $\neg P$  ( $\sqrt{2}$  é racional) é inválida (ou seja, não se argumentou a validade de  $P$  mas sim que não podia ser de outra forma). Diz-se dessas demonstrações que *não são constructivas* (não se constrói uma prova de  $P$  mas sim constata-se formalmente que esta tem de ser válida).

Para os curiosos, ver a discussão sobre a matemática constructiva, o intuicionismo de Brouwer, Heyting e Kolmogorov (os pais desta escola matemática) e as ligações do constructivismo com a algoritmia.





## Demonstrações por contradição

outro exemplo de demonstração não constructiva:

$$\exists \alpha, \beta \in (\mathbb{R} - \mathbb{Q}), \alpha^\beta \in \mathbb{Q}?$$

### Demonstração:

Temos:  $\sqrt{2} \notin \mathbb{Q}$ . Sejam  $\alpha = \beta = \sqrt{2}$

- Ou temos  $\alpha^\beta = \sqrt{2}^{\sqrt{2}} \in \mathbb{Q}$ , *QED*
- Ou temos  $\sqrt{2}^{\sqrt{2}} \notin \mathbb{Q}$ . Sejam então  $\alpha' = \sqrt{2}^{\sqrt{2}}$  e  $\beta' = \sqrt{2}$  então  $\alpha'^{\beta'} = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^2 = 2 = \frac{2}{1} \in \mathbb{Q}$ , ou seja *QED*

mas  $\sqrt{2}^{\sqrt{2}}$  é ou não é racional? Esta demonstração não o diz.



## Demonstrações pelo princípio da gaiola de pombos

- se  $A$  e  $B$  são dois conjuntos finitos tais que  $|A| > |B|$  então não existe nenhuma bijecção entre  $A$  e  $B$ .
- Por outras palavras (que justificam o nome), se tiver mais pombos do que gaiolas então necessariamente pelo menos uma das gaiolas vai ter de ficar com mais do que um pombo.
- Este princípio simples tem, surpreendentemente, muitas aplicações na matemática, em ciência da computação e em informática.

## Demonstrações pelo princípios da gaiola de pombos - Um exemplo

Retirado de [http://en.wikipedia.org/wiki/Pigeonhole\\_principle](http://en.wikipedia.org/wiki/Pigeonhole_principle):

*Although the pigeonhole principle may seem to be a trivial observation, it can be used to demonstrate possibly unexpected results. For example, there must be at least two people in London with the same number of hairs on their heads. Demonstration: a typical head of hair has around 150,000 hairs. It is reasonable to assume that no one has more than 1,000,000 hairs on their head. There are more than 1,000,000 people in London. If we assign a pigeonhole for each number of hairs on a head, and assign people to the pigeonhole with their number of hairs on it, there must be at least two people with the same number of hairs on their heads.*



## Demonstrações pelo princípios da gaiola de pombos - Outro exemplo

**Seja  $R$  uma relação binária sobre um conjunto  $A$ . Sejam  $a$  e  $b$  dois elementos de  $A$ . Se existe um caminho entre  $a$  e  $b$  por  $R$  então existe um caminho de comprimento máximo  $|A|$ .**

**Demonstração:** Seja  $a = a_1 a_2, a_3, \dots, a_n = b$  o caminho mais curto entre  $a$  e  $b$ . Suponha agora que  $n > |A|$ . Isto significa que há mais elementos no caminho do que elementos em  $|A|$ . Pelo princípios da gaiola de pombos sabemos que pelo menos um elemento de  $A$  está duas vezes no caminho (se mapeamos os pontos do caminho para elementos do conjunto  $A$ , então pelo menos dois pontos do caminho vão ser mapeados para um elemento comum). Digamos  $a_i = a_j$  para  $1 \leq i < j \leq n$ . Mas então  $a_1, a_2, \dots, a_i, a_{j+1}, \dots, a_n$  é igualmente um caminho, e mais curto. Contradição. QED.



# Demonstrações por diagonalização

## Princípio da Diagonalização:

Seja  $R$  uma relação binária sobre um conjunto  $A$ . Seja  $D$  o conjunto, designado de *diagonal*, definido por  $\{a \mid a \in A \wedge (a, a) \notin R\}$ . Para cada  $a \in A$ , seja  $R_a = \{b \mid b \in A \wedge (a, b) \in R\}$ . Então  $D$  distingue-se de cada  $R_a$ .

Este princípio simple é no entanto surpreendentemente poderoso.



## Demonstrações por diagonalização

Imagine a seguinte relação  $R$ :

$\{(a, b), (a, d), (b, b), (b, c), (c, c), (d, b), (d, c), (d, f), (e, e), (e, f), (f, a), (f, c), (f, d), (f, e)\}$

	$a$	$b$	$c$	$d$	$e$	$f$
$a$		×		×		
$b$		×	×			
$c$			×			
$d$		×	×		×	×
$e$					×	×
$f$	×		×	×	×	

a diagonal é então

$a$	$b$	$c$	$d$	$e$	$f$
	×	×		×	

e o seu complemento

$a$	$b$	$c$	$d$	$e$	$f$
×			×		×

é o **conjunto diagonal**,  $D$  de  $R$ , ou seja  $\{a, d, f\}$ . Repare que é diferente de qualquer linha da matriz.

## Demonstrações por diagonalização

Um exemplo : o Teorema de Cantor sobre a enumerabilidade do conjunto dos subconjuntos.

**O conjunto dos subconjuntos dum conjunto enumerável não é numerável**

**Demonstração:** Vamos proceder por uma redução ao absurdo e utilizar o princípio da diagonal. Seja  $A = \{a_0, a_1, a_2, \dots\}$ . e  $S = \{s_0, s_1, s_2\}$ . Supomos que  $S$  seja numerável. Consideremos então a relação  $R$  seguinte:  $R \triangleq \{(a, s) \mid a \in A, s \in S, a \in s\}$ .



## Demonstrações por diagonalização

$R \triangleq \{(a, s) \mid a \in A, s \in S, a \in s\}$  induz a matriz (eventualmente infinita) seguinte:

	$a_0$	$a_1$	$a_2$	$a_3$	$a_4 \dots$
$s_0$	×		×		×
$s_1$	×	□	×		
$s_2$		×	□	×	×
$s_3$			×	×	×
$s_4$	×	×		×	□
$\vdots$					

O conjunto diagonal  $D$  é assim  $\{a_i \mid a_i \notin s_i\}$  (os elementos assinalados por □).  $D$  é um subconjunto de  $S$  já que é composto de elementos de  $A$ .



## Demonstrações por diagonalização

- $A = \{a_0, a_1, a_2, \dots\}$
- $S = \{s_0, s_1, s_2\}$ .
- $R \triangleq \{(a, s) \mid a \in A, s \in S, a \in s\}$
- $D \triangleq \{a_i \mid a_i \notin s_i\}$ .

Pelo princípio da diagonal,  $\forall i \in \mathbb{N}, D \neq s_i$ . Logo  $D$  não é um subconjunto de  $A$  ( $D \notin S$ ). Contradição. QED.

Vejam esta argumentação com mais cuidado: Porque  $D$  não pode ser um dos  $s_i$ ? Imaginemos que  $\exists k \in \mathbb{N}, s_k = D$ . Então  $a_k \in D$  se e só se  $a_k \notin s_k$  (por definição de  $D$ ).

**O conjunto dos subconjuntos não é numerável.**



## Demonstração por indução estrutural

Esta técnica, muito útil no contexto desta disciplina, foi abordada no capítulo anterior e aplica-se às propriedades sobre conjuntos definidos por indução estrutural.  
Damos a seguir dois exemplos.

## Demonstração por indução estrutural

Seja  $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ , a função recursiva definida por:

$$f(m, n) \triangleq \begin{cases} n + 1 & \text{se } m = 0 \\ f(m - 1, 1) & \text{se } n = 0 \wedge m \neq 0 \\ f(m - 1, f(m, n - 1)) & \text{se } n > 0 \wedge m > 0 \end{cases}$$

Vamos demonstrar por indução que  $\forall k \in \mathbb{N}, f(1, k) = k + 2$ .



## Demonstração por indução estrutural

Vamos aqui utilizar a definição indutiva standard do inteiros natural (da aritmética de Peano). Neste caso a demonstração por indução deverá seguir o princípio:

$$((P\ 0) \wedge (\forall n \in \mathbb{N}. P\ n \rightarrow P\ (n + 1))) \rightarrow (\forall n \in \mathbb{N}. P\ n).$$

Aqui  $P\ x \triangleq f(1, x) = x + 2$

- Caso de Base: Demonstrar que temos  $P\ 0$  ou seja  $f(1, 0) = 2$ . Vejamos.  $f(1, 0) = (\text{regra 2})\ f(0, 1) = (\text{regra 1})\ 1 + 1 = 2$
- Passo indutivo: Seja  $x$  um valor inteiro. Admitindo que temos  $P\ x$  (Hipótese de indução, ou *HI*), temos de verificar se temos necessariamente  $P\ (x + 1)$ . Vejamos.

$$f(1, x + 1) = (\text{regra3})\ f(0, f(1, x)) = (\text{regra1})\ f(1, x) + 1 = (\text{HI})\ x + 2 + 1 \text{ ou seja } (x + 1) + 2. \text{ Qed.}$$

Conclusão:  $\forall k \in \mathbb{N}, f(1, k) = k + 2$



## Demonstração por indução estrutural

- 1 Defina de forma indutiva o conjunto  $bin_A$  das árvores binárias *não vazias* de elementos dum conjunto  $A$ . Por árvores não vazias, entendemos que as mais pequenas árvores deste conjunto são folhas (árvores com um só elemento do conjunto  $A$ );
- 2 Dê o princípio de indução associada a esta definição indutiva;
- 3 Defina a função *arestas* que calcula o número de vértice da árvore em parâmetro;
- 4 Defina a função *nodos* que calcula o número de nodos da árvore em parâmetro;
- 5 Demonstre que  $\forall a \in bin_A, nodos(a) = arestas(a) + 1$ .

## Demonstração por indução estrutural

- Seja  $A$  um conjunto. O conjunto das árvores binárias não vazias de elementos de  $A$  é o conjunto  $bin_A$  definido de forma indutiva a partir do alfabeto  $A_A \triangleq A \cup \{ "(", ")", ",", " " \}$  e das regras (B) e (I). De forma equivalente,  $bin_A$  é o mais pequeno conjunto  $X$ , dos subconjuntos do monóide livre gerado por  $A_A$  (ou seja:  $A_A^*$ ) que verifica:

$$(B): \forall a \in A, a \in X$$

$$(I): \forall e, d \in X, \forall a \in A, (e, a, d) \in X$$



## Demonstração por indução estrutural

- O princípio de indução, para uma propriedade  $P$  sobre árvores de  $bin_A$ , é o seguinte:

$$\begin{aligned} & (\forall x \in A, P(x)) \wedge (\forall e, d \in bin_A, \forall a \in A, P(e) \wedge P(d) \rightarrow P((e, a, d))) \\ & \rightarrow (\forall ab \in bin_A, P(ab)) \end{aligned}$$



## Demonstração por indução estrutural



$$\text{arestas } n = \begin{cases} 0 & \text{se } n \in A \text{ (} n \text{ folha)} \\ 2 + \text{arestas}(e) + \text{arestas}(d) & \text{se } n = (e, a, d) \end{cases}$$



$$\text{nodos } n = \begin{cases} 1 & \text{se } n \in A \text{ (} n \text{ folha)} \\ 1 + \text{nodos}(e) + \text{nodos}(d) & \text{se } n = (e, a, d) \end{cases}$$





## Demonstração por indução estrutural

- Vamos demonstrar por indução que  
 $\forall ab \in bin_A, nodos(ab) = arestas(ab) + 1$ . Temos assim de considerar o caso de base e o passo indutivo.

**Base:** Demonstrar que para toda a folha  
 $a \in A, nodos(a) = arestas(a) + 1$ . Esta demonstração é trivial. Seja  $a$  uma folha  
( $a \in A$ )  $nodos(a) = 1$  e  $arestas(a) = 0$ , logo  
 $nodos(a) = arestas(a) + 1$ .

## Demonstração por indução estrutural

**Indutivo:** Sejam  $e$  e  $d$  duas árvores de  $bin_A$  e  $a$  um elemento de  $A$ . As hipóteses de indução são as seguintes: (HI1)  $nodos(e) = 1 + arestas(e)$  e (HI2)  $nodos(d) = 1 + arestas(d)$ . Vamos a seguir demonstrar que (HI1) e (HI2) implicam que  $nodos((e, a, d)) = 1 + arestas((e, a, d))$ .

$$arestas((e, a, d)) = 2 + arestas(e) + arestas(d) \quad (*)$$

$$\begin{aligned} nodos((e, a, d)) &= 1 + nodos(e) + nodos(d) \\ (\text{por HI1}) &= 1 + 1 + arestas(e) + nodos(d) \\ (\text{por HI2}) &= 1 + 1 + 1 + arestas(e) + arestas(d) \\ &= 1 + 2 + arestas(e) + arestas(d) \\ (\text{por } (*)) &= 1 + arestas((e, a, d)) \end{aligned}$$

QED.

Temos assim  $\forall ab \in bin_A, nodos(ab) = arestas(ab) + 1$



## Demonstração por indução estrutural

```
type 'a abin =  
| Folha of 'a  
| Nodo of ('a abin)*'a*('a abin)  
;;
```

```
let rec arestas a =  
match a with  
| Folha _ → 0  
| Nodo (c, x, d) → (arestas c) + (arestas d) + 2  
;;
```

```
let rec nodos a =  
match a with  
| Folha _ → 1  
| Nodo (c, x, d) → (nodos c) + (nodos d) + 1  
;;
```



## Demonstração por indução bem fundada

- Pretende-se demonstrar  $\forall x \in A, P(x)$ .
- Uma ordem  $\leq$  bem fundada sobre  $A$  é uma relação binária sobre  $A$  tal que não exista para nenhum  $x$  de  $A$  uma sequência  $\dots \leq x_n \leq \dots \leq x_2 \leq x_1 \leq x$  **estritamente** decrescente infinita.

Por exemplo

- $\leq$  é bem fundada sobre  $\mathbb{N}$  (qualquer que seja o inteiro  $n$ , as sequências estritamente decrescentes acabam no pior caso em 0)
- $|$  a relação de divisão inteira ( $a|b$  significa que  $a$  divide  $b$ ) é bem fundada em  $\mathbb{N}$ : qualquer que seja o inteiro natural  $n$ , a maior sequência de divisores acaba em 1.
- Se o conjunto  $A$  dispõe de uma relação de ordem  $\leq$  bem fundada então o conjunto  $A$  dispõe dum princípio de indução designado de **bem fundado** definido da seguinte forma:


$$(\forall x \in A, (\forall y \in A, ((y < x) \wedge P(y)) \implies P(x))) \implies \forall x \in A, P(x)$$

(onde  $y < x \equiv y \leq x \wedge y \neq x$ )

## Demonstração por indução bem fundada

### Cada inteiro natural tem um decomposição única em números primos (modulo permutação dos elementos da decomposição)

**Demonstração da existência:** Por indução bem fundada sobre o inteiro  $n$  e a ordem  $|$  (onde  $a|b \triangleq a$  divide  $b$ ). Esta ordem é bem fundada (todas as cadeias estritamente decrescentes acabam no máximo em 1).

- 1 Caso em que  $n$  é primo. Este caso é trivial (não há decomposição).
  - 2 Caso contrário, existe pelo menos um  $d \in \mathbb{N}$  tal que  $1 < d < n$  e  $d|n$ . Seja  $p_1$  o menor destes  $d$ . Se  $p_1$  não for primo então e da mesma forma existe um  $q$  tal que  $1 < q < p_1$  e  $q|p_1$  mas neste caso  $q$  divide igualmente  $n$ . Contradição (porque  $q$  é um divisor menor do que  $p_1$ ). Logo  $p_1$  é primo. Neste caso  $n = p_1 \cdot n_1$ . Se  $n_1$  for primo, então já temos a nossa decomposição. Senão procedemos da mesma forma sobre  $n_1$  e obtemos uma decomposição  $n_1 = p_2 \cdot n_3$ . (etc...).
- Agora, podemos reparar que  $\dots n_i | n_{i-1} | \dots | n_2 | n_1 | n$ . Como sabemos que esta ordem é bem fundada, esta sequência tem de ser finita:  $\exists k$  tal que  $n_{k-1}$  primo (que designamos por  $p_k$ ). Ou seja,  $n = p_1 \cdot p_2 \cdot \dots \cdot p_k$ : QED. 

## Demonstração por indução bem fundada

**Cada inteiro natural tem um decomposição única em números primos (modulo permutação dos elementos da decomposição)**

**Demonstração da unicidade:** Por contradicção.

$$n = p_1 p_2 \dots p_r = q_1 q_2 \dots q_s$$

Assumimos, sem perda de generalidade que  $r \leq s$  e que os factores primos estão ordenados de forma crescente. Ou seja  $p_1 \leq p_2 \leq p_3 \leq \dots \leq p_r$  e  $q_1 \leq q_2 \leq q_3 \leq \dots \leq q_s$ . Sabemos que  $p_1$  divide  $n$ , ou seja, divide igualmente  $q_1 q_2 \dots q_s$ . Como todos os  $q_i$ s são primos, existe um  $k$  tal que  $p_1 = q_k$ . Logo  $p_1 \geq q_1$ . Da mesma forma, se olharmos desta vez para  $q_1$  conseguimos demonstrar que  $q_1 \geq p_1$ . Logo  $p_1 = q_1$ . Ou seja  $p_1 p_2 \dots p_r = p_1 q_2 \dots q_s$ . Ou ainda  $p_2 \dots p_r = q_2 \dots q_s$ .

Podemos repetir este raciocínio até  $p_r$ . Logo  $1 = q_{r+1} q_{r+2} \dots q_s$ . Esta situação só é possível se  $s = r$  (nenhum produto de números primos (que são todos  $> 1$ ) é igual a 1). Logo  $\forall i, p_i = q_i$ . QED.

## Demonstração por indução bem fundada

Seja  $A^*$  o monoíde livre gerado pelo alfabeto  $A$ . Vamos demonstrar que

$$\forall u, v \in A^*. (u.v = v.u \leftrightarrow \exists w \in A^*, \exists p, q \in \mathbb{N}. u = w^p \text{ e } v = w^q)$$

→ . Fácil. se  $u = w^p$  e  $v = w^q$  então  
 $u.v = w^p.w^q = w^{p+q} = w^q.w^p = v.u$



## Demonstração por indução bem fundada

←. Demonstração por indução bem fundada sobre  $|u| + |v|$ . Ou seja a propriedade por demonstrar é:

$$P(n) \triangleq \forall u.v \in A^*. |u| + |v| = n, u.v = v.u \rightarrow \exists w \in A^*, \exists p, q \in \mathbb{N}. u = w^p \text{ e } v = w^q$$

. A ordem por utilizar aqui é a ordem natural  $\leq$  sobre os inteiros. Esta ordem é bem fundada (não podemos definir cadeias infinitas estritamente decrescentes).

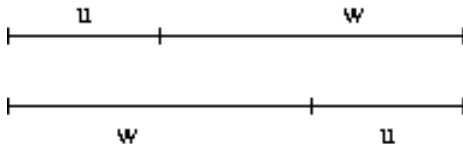
Assim sendo, Por hipótese de indução bem fundada temos:  $\forall k < n, P(k)$ .  
Verifiquemos agora que temos então necessariamente  $P(n)$ .

Sem perder generalidade supomos que  $|u| \leq |v|$ . Neste caso  $u$  é prefixo de  $v$ .





## Demonstração por indução bem fundada



- Se  $u = \epsilon$  ou  $u = v$  então basta escolher  $w = v$
- Nos outros casos ( $u$  é designado de prefixo próprio de  $v$ ), então existe um  $v'$  tal que  $v = uv'$  e verifica-se que  $v'u = v = uv'$ . Visto que  $|v'| < |v|$  podemos aplicar a hipótese de indução ao par  $(u, v')$  (ou seja  $\exists w, p, q. u = w^p \wedge v' = w^q$ ). Como  $v = u.v'$  deduz-se que  $v = w^{p+q}$ . QED.



## Demonstração por indução bem fundada

Outro caso de utilização privilegiada desta técnica de demonstração é a demonstração de terminação de funções recursivas.

Por exemplo:

Seja  $div\_euclides : \mathbb{N} \rightarrow \mathbb{N} \rightarrow \mathbb{N}$  a função seguinte:

$$div\_euclidesxy = \begin{cases} 0 & \text{if } x < y \\ x & \text{if } y = 0 \\ (div\_euclides (x - y) y) + 1 & \text{otherwise} \end{cases}$$

Demonstremos, por indução bem fundada, que a função  $div\_euclides$  termina.



## Demonstração por indução bem fundada

- Para demonstrar que a função  $d$  termina, basta conseguir ligar as chamadas recursivas à uma relação de ordem bem fundada. A ideia é ver que para determinados  $x$  e  $y$ ,  $x - y$  é mais pequeno, quando comparado com essa ordem, do que  $x$ . Como esta ordem não tem cadeias descendentes infinitas, a função tem de terminar.

Várias ordens bem fundamentadas são possíveis candidatas. Escolhamos a mais natural de todas:  $\leq$  ( $\subseteq \mathbb{N} \times \mathbb{N}$ ). (Podíamos ter escolhido a ordem  $\leq_{\mathcal{L}}$  sobre os pares de inteiros. Neste caso teríamos de nos debruçar sobre o par  $(x, y)$  e não só sobre  $x$ )

Começemos por verificar que todos os casos de base terminam. Todos eles se determinam em relação ao valor de  $y$  (que não mude durante as diferentes chamadas recursivas).

- 1 se  $y = 0$  o cálculo de  $x$  termina obviamente;
- 2 se  $x < y$  o cálculo de 0 termina obviamente;



## Demonstração por indução bem fundada

Desbruscemo-nos agora sobre o passo indutivo. Temos de verificar:

- que a chamada recursiva faz com um argumento menor estritamente (estritamente menor em relação a ordem bem fundamentada escolhida);
- que se  $(d(x - y) y)$  termina então  $(d \times y)$  também termina.

Estes dois pontos são triviais.  $(x - y) < x$  quando  $y > 0$  (que é o caso, visto  $y = 0$  ser um dos casos de base). Somar um a um cálculo por hipótese finito é feito em tempo finito. **QED**

