UNIVERSIDADE BEIRA INTERIOR

# Teoria da Computação
# Aula 2 - Problemas, Programas, Linguagens

Simão Melo de Sousa

(re)lease
REliable And SEcure Computation Group

# Introdução

## prelúdio

- A redacção dos apontamentos da disciplina documento baseou-se fortemente na bibliografia indicada. Parece-nos então óbvio que a leitura e a aprendizagem directa pelas obras originais é recomendada, e mesmo essencial à compreensão profunda das noções aqui apresentadas;
- O português não é a língua materna do autor e o presente documento encontra-se em fase (constante) de elaboração/melhoramento pelo que se agradece e até se incentiva qualquer sugestão ou correcção;

# Referencias bibliográficas


- (Principal) C. H. Papadimitriou, H. R. Lewis. **Elements of the Theory of Computation** por Prentice Hall, 1997. Tradução brasileira: **Elementos de Teoria da Computação**, 2a Edição. Bookman, Porto Alegre, 2000.
- (introdutório e de leitura agradável) P. Linz. **An introduction to formal languages and automata**. Jones and Bartlett Publisher, 2006.
- (Uma obra de referência e muito completo... um "must") John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman. **Introduction to Automata Theory, Languages, and Computation** (3nd Edition). Addison Wesley, 2006 (existe em português do Brasil).
- (Completo, rigoroso e muito bem organizado) Dexter Kozen. **Automata and Computability**. Undergraduate Texts in Computer Science, Springer, 1997.
- (Completo e também um "must") M. Sipser. **Introducton to the Theory of Computation**. PWS Publishing, 2006.

- Perceber os limites da informática
- Distinguir os problemas resolúveis algoritmicamente dos que não o são
- Obter resultados independentes da tecnologia utilizada para construir computadores
- Perceber que tudo na programação é questão de linguagem e de computação sobre palavras

# Problemas e Procedimentos Efectivos

- 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.

**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.

**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 duma instancia dum problema.

- um programa Java é um procedimento efectivo (pensado para ser executado por um computador)
- o algoritmo de euclides, o crivo de eratostene, 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 algoritmica.

- 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**. Repitimos 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`.

- 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**.

# Problemas e Linguagens

## A Formalização dos Problemas

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: Conjunto finito de símbolos

por exemplo

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

- 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 binário** . (a concatenação) **associativo** com um **elemento neutro**: a palavra vazia $\epsilon$ (a palavra feita com 0 letras de $A$, i.e. de cumprimento 0).
- Admite-se, por razões de conveniência, que a concatenação ., por exemplo $a.b.c.d.e.f.g$ seja notada $abcdefg$.

- Linguagem $L$ sobre um alfabeto $A$: Conjunto de palavras, ou seja um subconjunto de $A^*$.
- Define-se sobre as linguagens e sobres as palavras as seguintes operações:
  - O cumprimento de uma palavra $w$, notada $|w|$, devolve o cumprimento da sequência de simbolos 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 simbolo (letra) da sequência $w$.
- exemplos: $abb3bwk217m$ é uma palavra sobre o alfabeto $\{0, \cdots, 9, a, \cdots, z\}$. Designemos por $w$ esta palavra. $w.(5) = b$, $|w| = 11$. No que diz respeito a palavra vazia, $|\epsilon| = 0$

# Problemas e Linguagens

## Representação 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)

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

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

- O problema: Dado uma fórmula, será esta uma tautologia?
- Conjunto de todas as palavras: $A^*$
- As instâncias: $\mathcal{P}rop \quad (\in A^*)$
- 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 "$((\vee A$" é uma palavra da linguagem, isto é: "$((\vee A$" $\in A^*$, mas não é uma instância do problema da satisfação (porque nem sequer é uma formula lógica)

# Linguagens Formais

- Primeiro, uma nota: A linguagem vazia (notação $\emptyset$), não é igual a linguagem que só tem a palavra vazia (ou seja $\{\epsilon\}$).
- Sejam $L$, $L_1$ e $L_2$ linguagens.
  - $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$
  - $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$
  - $L_1.L_2 = \{w \mid w = x.y, x \in L_1 \wedge y \in L_2\}$
  - $L^0 = \{\epsilon\}$
  - $L^n = L.L^{n-1}$ (com $n > 0$),

    alternativamente temos
    $L^n = \{w \mid \exists w_1 \ldots w_n \in L, w = w_1.w_2.\cdots w_n\}$

    logo, $L^1 = L$
  - $L^* = \{w \mid \exists k \geq 0, w_1 \ldots w_k \in L, w = w_1.w_2.\cdots w_k\} = \bigcup_{n=0}^{+\infty} L^n$
  - $L^+ = \{w \mid \exists k > 0, w_1 \ldots w_k \in L, w = w_1.w_2.\cdots w_k\} = \bigcup_{n=1}^{+\infty} L^n$
  - $\overline{L} = \{w \mid w \notin L\}$

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

- (B) elementos de base:
  - $\emptyset \in \mathcal{R}$,
  - $\{\epsilon\} \in \mathcal{R}$,
  - $\forall a \in \Sigma, \{a\} \in \mathcal{R}$
- (I) elementos produzidos por indução:
  $\forall A, B \in \mathcal{R}$,
  - (Fecho por União) $A \cup B \in \mathcal{R}$
  - (Fecho por Concatenação) $A.B \in \mathcal{R}$
  - (Fecho pela operação de Kleene)$A^* \in \mathcal{R}$

- Assim, linguagens regulares são linguagens construídas a partir de linguagens "atómicas" e operações simples.
- mais uma vez: $\emptyset \neq \{\epsilon\}$
- Um exemplo interessante:

  Seja $C$ um alfabeto qualquer e $A = \{a, b\} \cup C$.

  A linguagem $\mathcal{A} = A^* - (A^*\{ab\}A^*)$ é a linguagem de todas as palavras geradas a partir de letras de $A$ mas que não contém a sequência $ab$.

  Esta linguagem é regular, apesar da operação de diferença $-$ não ser **por definição** regular.

  De facto $\mathcal{A} = (\{b\} \cup C)^*.(\{a\}^+.C.(\{b\} \cup C)^*)^*.\{a\}^*$.

Nem sempre é cómodo tratar as linguagens como conjuntos de palavras. Uma outra abordagem consiste em agrupar palavras consoante os padrões que essas apresentam: as **expressões regulares**.

Ou seja: Linguagens são conjuntos de palavras, **Expressões regulares são uma notação**.

Definição indutiva da noção de expressão regular sobre um alfabeto $\Sigma$ (notação $RegExp(\Sigma)$).

Definido sobre o monoíde livremente gerado por o alfabeto $\Sigma$ seguinte: $(\Sigma \bigcup \{), (, \emptyset, +, *, \epsilon\})^*$:

- (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.

Veremos que são uma boa notação para as linguagens regulares.

# Expressões regulares são uma notação

para insistir sobre o facto que as expressões regulares formam uma notação cómoda para falar de linguagens:

$\{x \in \mathbb{N} \mid x \ mod \ 2 = 0\}$ representa comodamente o conjunto infinito $\{0, 2, 4, 6, 8, 10, \cdots\}$

$(a + b)^* bb$ é uma notação finita que representa comodamente o conjunto de palavras (i.e. linguagem) $\{bb, abb, bbb, aabb, abbb, babb, bbbb, \cdots\}$

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

Convém agora relacionar a notação com os conjuntos.

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

$$L(r) = \begin{cases} \emptyset & se\ \ r = \emptyset \\ \{\epsilon\} & se\ \ r = \epsilon \\ \{a\} & se\ \ r = a\ (com\ a \in \Sigma) \\ L(a) \cup L(b) & se\ \ r = (a + b) \\ L(a).L(b) & se\ \ r = (a.b) \\ L(a)^* & se\ \ r = (a)* \end{cases}$$

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

de certa forma a função $L$ é um compilador de expressões regulares para linguagens (que veremos serem regulares)

exemplo:

$$L(ab + (c + \epsilon)) = L(a).L(b) \cup (L(c) \cup L(\epsilon)) = \{ab\} \cup \{c, \epsilon\} = \{ab, c, \epsilon\}$$

Assumindo que uma linguagem é uma lista de string
(ou seja `type linguagem = string list`)

```
let rec language_of_expreg = function
| Vazia          -> []
| Epsilon        -> [empty_string]
| Caracter a     -> [(string_of_char a)]
| Uniao    (a,b) -> uniao ((language_of_expreg a) (language_of_expreg b))
| Produto  (a,b) -> let la = language_of_expreg a in
                    let lb = language_of_expreg b in
                    flatten (map (fun x -> (map (fun y -> x^y) lb)) la)
| Estrela  a     -> repeat max_n a
```

(assume-se a definição prévia das funções auxiliares e do limite `max_n`)

esta função é uma aproximação (até `max_n`) da função $L(.)$ visto esta gerar conjuntos infinitos

- Duas expressões regulares $E$ e $F$ são **equivalentes** se descrevem a mesma linguagem ( i.e. $\mathcal{L}(E) = \mathcal{L}(F)$, notação $E \sim F$).
- O problema de saber se $\forall a, b \in RegExp(A), a \sim b$ é um problema **central**, complexo mas **solúvel**.

  Veremos assim que existam métodos baseados em autómatos (com a ajuda do teorema de Kleene) ou baseados numa manipulação algébrica, mas directa, das expressões regulares.

**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

**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
  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.

**Quod Erat Demonstrandum**.

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

- Por conveniência nota-se por
  - $ab$ a expressão $a.b$.
  - (ocasionalmente) $a|b$ a expressão $a + b$.
  - $r^+$ (ou $r+$) a expressão $r.r^*$ (ou $rr^*$).
  - $r?$ a expressão $(r + \epsilon)$ ("eventualmente $r$")
  - $r^n$ a expressão $\underbrace{r \cdots r}_{n}$

- Para facilitar a escrita de expressões regulares:
  - Prioridade implícita dos operadores (por ordem decrescente): $* \cdot +$
  - Associatividade de . e de $+$:
    - Porque $\mathcal{L}(((E+F)+G)) = \mathcal{L}((E+(F+G)))$, então utilizaremos a notação $E+F+G$
      (**exercício**: demonstrar esta afirmação).
    - Porque $\mathcal{L}(((E.F).G)) = \mathcal{L}((E.(F.G)))$, então utilizaremos a notação $E.F.G$
      (**exercício**: demonstrar esta afirmação).
- Finalmente, confundiremos expressão regular $e$ e a linguagem gerada por $e$ (isto é, $\mathcal{L}(e)$).

  Falaremos assim da ***linguagem*** $aa^*(b+c)^*$ em vez de $\mathcal{L}(aa^*(b+c)^*)$.

- Exemplos: Seja o alfabeto $A = \{a, b\}$
  - $(a + b)^*$ descreve a linguagem $(\{a\} \cup \{b\})^*$
  - $(a^*).b.(a)^*$ descreve a linguagem das palavras que contêm exactamente uma ocorrência de $b$
  - $\mathcal{L}(a(ab)^*b) = \{a(ab)^n b \mid n \geq 0\}$
- Para o alfabeto $A' = \{0, 1\}$, a expressão regular $(1(0 + 1)^*)^*10$ descreve a linguagem de todos os números binários $m$ tais que $\exists k \in \mathbb{N}, m = 4k + 2$.
  (**exercício**: porquê?)
- teremos
  - $(ab)^* \sim a^*b^*$?
  - $(a + b)^* \sim a^* + b^*$?
  - $a(b + c) \sim ab + ac$?

- $(a+b)^* a (a+b)^*$ 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:

```
<float> ::= -? <digit>+ (. <digit>*)? ((e | E) (+ | -)? <digit>+)?
```

um conceito interessante que envolve expressões regulares e linguagens é o de **linguagens quocientes esquerdas**

sejam $\Sigma$ um alfabeto, $L$ uma linguagem sobre este alfabeto e $w$ uma palavra de $\Sigma^*$
a **linguagem quociente esquerda** de $L$ em relação a $w$, denotada por $D_w(L)$ é definida por

$$D_w(L) \triangleq \{v \mid v \in \Sigma^* \wedge wv \in L\}$$

ou seja, é o conjunto das palavras $v$ de $\Sigma^*$ que sufixando $w$ formam uma palavra de $L$
alternativamente, se considerarmos $L'$ como o subconjunto de $L$ das palavras que têm $w$ como prefixo, $D_w(L)$ é o conjunto dos sufixos de $L'$

um caso particular de $D_w(L)$ ocorre quando $w$ é uma letra de $\Sigma$ (digamos $a \in \Sigma$)
assim, parafraseando,

$$D_a(L) \triangleq \{v \mid v \in \Sigma^* \wedge av \in L\}$$

uma nota interessante é a de que

$$L = \bigcup_{a \in \Sigma} a.D_a(L)$$

$$(a^*b)^* + (b^*a)^* \sim (a+b)^*$$

Demonstração:
Em dois passos. Primeiro, provar que $(a^*b)^* + (b^*a)^* \subseteq (a+b)^*$. Provar, em seguida, que $(a+b)^* \subseteq (a^*b)^* + (b^*a)^*$.

**passo 1.**

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

$$(a^*b)^* + (b^*a)^* \sim (a+b)^*$$

Demonstração:

**passo 2.**

- $(a+b)^* \subseteq (a^*b)^* + (b^*a)^*$.

  Consideramos uma palavra qualquer $w$ de $(a+b)^*$. Digamos $w = w_1w_2w_3 \ldots w_n$. 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:

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

     $\implies w \in (a^*b)^* + (b^*a)^*$
  4. $w$ contém $a$ e $b$ em quantidade arbitrária e termina por um $a$. $w$ pode se descompor de forma semelhante ao ponto anterior.

**Situação:** Acabamos de ver como provar a equivalência de duas expressões regulares.

**Argumento usado:** Reflexão e Demonstração Matemática.

**Problema:** A equivalência de expressões regulares é um problema decidível (para o qual existe um procedimento efectivo de resolução)?

**Resposta:** Sim. a equivalência de expressões regulares é um problema decidível. Veremos um dos algoritmos possíveis no capítulo dedicado às linguagens regulares e aos autómatos finitos. Veremos assim pelo menos uma forma mais cómoda e automática de comprovar (ou não) a equivalência de expressões regulares.

Para terminar esta aula,

**Questão:** Serão todas as linguagens, linguagens regulares?

**Resposta:** Não. Há mais linguagens do que as linguagens regulares.

- Diz-se de dois conjuntos que têm 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 um subconjunto de $\mathbb{N}$, ou seja, quando tem no máximo a cardinalidade de $\mathbb{N}$.
- Por exemplo, o conjunto das palavras sobre o alfabeto $\{a, b\}$ é numerável. Porque podemos definir a bijecção seguinte: $\{(\epsilon, 0), (a, 1), (b, 2), (aa, 3), (ab4), (ba, 5), (bb, 6), (aaa, 7), \ldots)\}$
- 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.

- 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**.