Teoria da Computação

Linguagens

versão 1.3

Prof. D.Sc. Fabiano Oliveira
fabiano.oliveira@ime.uerj.br

Linguagens

- Um ***alfabeto*** é um conjunto finito não-vazio cujos elementos são chamados de símbolos.

  Ex.:
  {a, b, c, d}
  {1, 2, 3, 4, 5}
  $\{x \in \mathbb{N} \mid x \leq 1000\}$

  Usualmente, denotaremos um alfabeto por $\sum$

Linguagens

- Uma ***cadeia*** s de comprimento n sobre o alfabeto Σ é ε se n = 0 (cadeia nula) ou uma função s: {1,...,n} $\rightarrow \Sigma$ se n > 0
  - n é denotado por |s|
  - para n > 0, usualmente s é representado por s(1)s(2)...s(n-1)s(n)

  Ex.:
  cada é uma cadeia sobre {a, b, c, d ,e}
  123 é uma cadeia sobre $\{x \in \mathbb{N} \mid x \leq 9\}$

Linguagens

- A ***concatenação*** de duas cadeias x e y, denotado por x º y, é:
  - x, se y = ε
  - y, se x = ε
  - a cadeia x(1)...x(|x|)y(1)...y(|y|), caso contrário

  Ex.:

  ε º ε = ε

  cada º ε = ε º cada = cada

  cada º abc = cadaabc

Linguagens

- Uma ***linguagem*** L sobre o alfabeto $\sum$ é um conjunto de cadeias sobre $\sum$

  Ex.: $\sum = \{a, b, c, d, e\}$

  $\varnothing$, {ε}, {abc, bcd, cde}, {ε, a, b, c, d, e}

  $\{a^i : i \in \mathbb{N}\}$

  $\{a^i bc^i de^i : i \in \mathbb{N} \mid i \text{ é ímpar}\}$

Linguagens

- Se $\sum$ é um alfabeto, $\sum^*$ é a linguagem que contém todas as cadeias sobre $\sum$

  Ex.: $\sum = \{a, b\}$

  $\sum^* = \{ \varepsilon , a, b, aa, ab, ba, bb, aaa, aab, aba, abb,$

  $baa, bab, bba, bbb, aaaa, \ldots\}$

- Portanto, se L é uma linguagem sobre $\sum$, necessariamente $L \subseteq \sum^*$

Linguagens

- Durante TODO O RESTANTE DO CURSO, estaremos interessados em computar:

  Seja L uma linguagem sobre $\sum$ e x $\in \sum^*$.
  Pergunta: x $\in$ L ?

- Esta pergunta, é o "átomo" da computação. O que é possível computar, depende de sabermos a resposta a esta pergunta!

Linguagens

- Em geral, cada problema computacional pode ser apresentado em três versões:
  - Dado uma instância E de um problema, seja Resp (E) o conjunto de possíveis respostas para E. Seja v (R) uma função que atribui a cada $R \in Resp(E)$ um valor positivo. Considere os problemas:
    - **Decisão**: Dado uma constante K, existe $R \in Resp(E)$ tal que $v(R) \leq K$?
    - **Busca**: Dado uma constante K, encontre $R \in Resp(E)$ tal que $v(R) \leq K$
    - **Otimização**: Encontre $R \in Resp(E)$ tal que $v(R) = \min \{ v(R') \mid R' \in Resp(E) \}$

Linguagens

- **Ex.**: Seja M um mapa rodoviário. Considere os problemas:
  - **Decisão**: Dado uma constante K, e duas cidades x e y, existe um caminho entre x e y com comprimento ≤ K?
  - **Busca**: Dado uma constante K, e duas cidades x e y, encontre um caminho entre x e y com comprimento ≤ K
  -
  - **Otimização**: Determine a distância mínima entre x e y, i. e., min { K ∈ $\mathbb{R}$ | existe um caminho entre x e y com comprimento ≤ K }

(E = mapa, x e y, R ∈ Resp(E) = um caminho entre x e y,
v(R) = comprimento de R)

Linguagens

- Portanto, a versão mais fundamental de qualquer problema é este problema na sua versão de decisão

- Por outro lado, dado problema de decisão P, podemos definir o seguinte conjunto:

P_{SIM} = { (E, K) | P tem resposta SIM com instância E, constante K }

Linguagens

- Logo, dada uma instância E e um natural K, resolver o problema de decisão é saber se (E, K) pertence ou não a P_{SIM}

- Para entrar para um computador, E e K devem ser codificados sobre algum alfabeto $\sum$. Sendo assim, fazendo $L = P_{SIM}$, queremos resolver

  Dado $x \in \sum^*$, $x \in L$?

- Conforme queríamos demonstrar, esta é uma pergunta fundamental da Ciência da Computação

Linguagens

- Operação com linguagens:
  - **união, interseção, diferença, complemento:**
    Como linguagens são conjuntos, tais operações são aquelas de conjuntos
    - Para o complemento, qual seria o conjunto universo natural?

Linguagens

- Operação com linguagens:
  - **concatenação:**

$$L \circ M = \{ x \circ y : x \in L, y \in M \}$$

Ex.: $L = \{\varepsilon, a, ab\}$, $M = \{\varepsilon, bc, bcd, e\}$

$L \circ M = \{\varepsilon, bc, bcd, e,$
$a, abc, abcd, ae,$
$ab, abbc, abbcd, abe\}$

Linguagens

- Operação com linguagens:
  - **potenciação:**

$L^0 = \{ \varepsilon \}$
$L^k = L \circ L^{k-1}$, para todo $k > 0$

Ex.: $\{0, 1\}^3 = \{0, 1\} \circ (\{0, 1\} \circ (\{0, 1\} \circ \{ \varepsilon \})) =$
$= \{0, 1\} \circ (\{0, 1\} \circ \{0, 1\}) =$
$= \{0, 1\} \circ \{00, 01, 10, 11\} =$
$= \{000, 001, 010, 011, 100, 101, 110, 111\}$

Linguagens

- Operação com linguagens:
  - **fechamento:**

$L^* = L^0 \cup L^1 \cup L^2 \cup ...$

Ex.: $\{\varepsilon, ab, bc\}^* = \{ \varepsilon \} \cup \{\varepsilon, ab, bc\} \cup$

$\cup \{\varepsilon, ab, bc, abab, abbc, bcab, bcbc\} \cup$

...

Linguagens

Teorema:
Para qualquer alfabeto $\sum$, $\sum^*$ é enumerável.

- Dem(???): Ordenar as cadeias de $\sum^*$ por ordem alfabética. Seja f: $\mathbb{N} \rightarrow \sum^*$ tal que f(i) = s $\Leftrightarrow$ s é o (i+1)-ésimo da ordenação feita. (Qual o problema com esta prova?)

Linguagens

Teorema:
Para qualquer alfabeto $\sum$, $\sum^*$ é enumerável.

- Dem.: Ordenar as cadeias de $\sum^*$ por ordem de tamanho, e, dentre as de mesmo tamanho, em ordem alfabética. Seja f: $\mathbb{N} \rightarrow \sum^*$ tal que f(i) = s $\Leftrightarrow$ s é o (i+1)-ésimo da ordenação feita.

Linguagens

Teorema:
Qualquer linguagem é enumerável.

- Dem.: Uma linguagem sobre $\sum$ é um subconjunto de $\sum^*$. Pelo teorema anterior, $\sum^*$ é enumerável. Como um subconjunto de um conjunto enumerável deve ser enumerável (prove!), qualquer linguagem é enumerável.

Linguagens

Teorema:
O conjunto de todas as linguagens sobre um alfabeto não é enumerável

- Dem.: Se L é uma linguagem sobre $\sum$, então $L \subseteq \sum^*$. Logo, o conjunto de todas as possíveis linguagens sobre $\sum$ é precisamente todos os subconjuntos de $\sum^*$, isto é, $P(\sum^*)$. Pelo processo de diagonalização, $P(\sum^*)$ não é enumerável.

Linguagens

Teorema:
Existem problemas de decisão que não admitem um algoritmo para resolvê-los.

Linguagens

- Dem.: Uma vez definido um alfabeto $\sum_{ALG}$ para escrever algoritmos, o conjunto de todos os possíveis algoritmos (cadeias sobre $\sum_{ALG}$) é um subconjunto de $\sum_{ALG}^*$, que é enumerável por teorema anterior. Uma vez definido um alfabeto $\sum_{ENT}$ de entrada de problemas (cadeias sobre $\sum_{ENT}$), cada problema de decisão pode ser visto como um conjunto de entradas para as quais a resposta é SIM. Ou seja, um problema é um subconjunto de $\sum_{ENT}^*$. O conjunto de todos os problemas é, portanto, todos os subconjuntos de $\sum_{ENT}^*$, que é não-enumerável por teorema anterior. Logo, não se pode fazer uma correspondência entre o conjunto de algoritmos e de problemas, pois existem mais problemas que algoritmos.

Linguagens

- Uma linguagem L é ***recursivamente enumerável*** (r.e.) se existe um algoritmo que escreve cada x ∈ L em tempo finito.

  Ex.: os racionais são r.e.

```
procedimento Enumera()   // ∑ = {0, …, 9, /, -}
        para i ← 1 até ∞ faça
                para j ← 1 até i faça
                        escrever (i- j, "/", j); escrever ("-", i- j, "/", j)
                fim-para
        fim-para
fim-procedimento
```

Linguagens

- Uma linguagem L é ***recursiva*** se existe um algoritmo que determina, para cada $x \in \sum^*$, se $x \in L$ ou se $x \notin L$ em tempo finito.
  Ex.: os racionais são recursivos

```
função Recursivo(x ∈ {0, …, 9, /, -}*): Lógico
        i ← 0
        se x(i) = "-" então i ← i + 1
        enquanto s(i) ≠ "/" faça
                p ← p ° s(i);  i ← i + 1
        fim-enquanto
        para i ← i + 1 até |x| faça
                q ← q ° s(i)
        fim-para
        retornar (p não contém "-") E (q não contém "/" nem "-") E (q ≠ "0")
fim-função
```

Linguagens

Teorema:
Toda linguagem recursiva é recursivamente enumerável.

Linguagens

Dem.: Seja L uma linguagem recursiva. Seja A(x) o algoritmo que decide se $x \in L$, para cada $x \in \Sigma^*$. Como Σ^* é enumerável, existe sobrejeção $f: \mathbb{N} \rightarrow \Sigma^*$. Considere o seguinte algoritmo que lista todos os elementos de L:

```
procedimento Enumera()
    para i ← 0 até ∞ faça
        x ← f(i)
        se (A(x) = V) então
            escrever (x)
        fim-se
    fim-para
fim-procedimento
```