Por $\mathcal{A}lef\ \mathcal{K}euffer$

Primitive Recursion

$$h = \mathrm{Rec}(f, g) \iff \begin{cases} h(\vec{x}, 0) & = f(\vec{x}) \\ h(\vec{x}, y+1) & = g(\vec{x}, y, h(\vec{x}, y)) \end{cases}$$

where $\vec{x} = x_1, x_2, \ldots, x_k$

$$h = M_f \quad \iff \quad h(\vec{x}) = \min\{y \in \mathbb{N}_0 : f(\vec{x}, y) = 0\}$$

where $\vec{x} = x_1, x_2, \ldots, x_k$

Definições:

1. As funções recursivas primitivas são as funções iniciais e todas aquelas que podem ser obtidas das funções iniciais pela aplicação de um número finito de vezes das operações de composição e de recursão primitiva.

Teoremas

1. Todas as funções recursivas primitivas são computáveis.
2. Todas as funções recursivas primitivas são funções totais.
3. Existem funções totais computáveis que não são recursivas primitivas.
4. Uma função diz-se parcial μ-recursiva (ou simplesmente parcial recursiva) se é uma função inicial ou pode ser obtida destas pela aplicação de um número finito de vezes das operações de composição, recursão primitiva e minimização. Uma função parcial recursiva que seja total diz-se recursiva.
5. Uma função $f : \mathbb{N}_0^k \to \mathbb{N}_0$ é parcial recursiva se e só se é computável

Funções primitivas recursivas (provadas em exercícios)

- $\mathrm{mult}(x, y) = x \cdot y$
- $\exp(x, y) = x^y$
- $\mathrm{fat}(x) = \begin{cases} 1 & \text{se } x = 0 \\ x \cdot (x-1) \cdot \ldots \cdot 2 \cdot 1 & \text{se } x > 0 \end{cases}$
- $\mathrm{ad}^{(k)}(x_1, \ldots, x_k) = x_1 + \cdots + x_k$

Complexity

Ordem

$g(n) \in \mathcal{O}(f(n)) \implies$
$\exists (c \in \mathbb{R}^+). \exists (n_0 \in \mathbb{N}). \forall (n > n_0). 0 \leq g(n) \leq cf(n).$

$\mathcal{O}(f(n)) =$
$\{g(n) : \exists (c \in \mathbb{R}^+). \exists (n_0 \in \mathbb{N}). \forall (n > n_0). 0 \leq g(n) \leq cf($

Complexidade determinista

Seja $\mathcal{T}$ uma máquina de Turing que pára sempre (ou seja, $\mathcal{T}$ é um algoritmo). A complexidade temporal de $\mathcal{T}$ é a função $tc_{\mathcal{T}} : \mathbb{N}_0 \to \mathbb{N}_0$ tal que, para cada $n \in \mathbb{N}_0$,

$$tc_{\mathcal{T}}(n) = \max \left\{ m_u \,\middle|\, \begin{array}{l} u \text{ é uma palavra de} \\ \text{comprimento } n \text{ e } m_u \text{ é o} \\ \text{número de passos que } \mathcal{T} \\ \text{executa (até parar) quando} \\ \text{é iniciada com } u. \end{array} \right\}$$

.

Complexidade não-determinista

Seja $\mathcal{T}$ uma MT não-determinista que pára sempre. A **complexidade temporal** de $\mathcal{T}$ é a função $tc_{\mathcal{T}} : \mathbb{N}_0 \to \mathbb{N}_0$ definida, para cada $n \in \mathbb{N}_0$, por

$$tc_{\mathcal{T}}(n) = \max \left\{ m_u \,\middle|\, \begin{array}{l} m_u \text{ é o maior número} \\ \text{de computações que podem} \\ \text{ser efetuadas por } \mathcal{T} \\ \text{quando iniciada com} \\ \text{uma palavra } u \\ \text{de comprimento } n. \end{array} \right\}$$

Complexidade de linguagens

Sejam $f : \mathbb{N}_0 \to \mathbb{R}$ uma função (total) e L uma linguagem. Diz-se que L é **aceite em tempo determinista (resp. não-determinista)** $f(n)$ se existe um algoritmo determinista (resp. não-determinista) $\mathcal{T}$ tal que:

- $\mathcal{T}$ aceita L
- $tc_{\mathcal{T}}(n) \in \mathcal{O}(f(n))$

A classe destas linguagens é denotada por $\mathrm{DTIME}(f(n))$ (resp.) $\mathrm{NTIME}(f(n))$. Note-se que $\mathrm{DTIME}(f(n)) \subseteq \mathrm{NTIME}(f(n))$.

Podemos agora definir duas classes de complexidade importantes:

$$\mathrm{P} = \bigcup_{k \geq 0} \mathrm{DTIME}(n^k) \qquad \text{e} \qquad \mathrm{NP} = \bigcup_{k \geq 0} \mathrm{NTIME}(n^k)$$

Redução

Consideremos linguagens $L_1 \subseteq A_1^*$ e $L_2 \subseteq A_2^*$. Diz-se que L_1 é **polinomialmente reduzível** a L_2 (ou que L_1 **se reduz a** L_2 **em tempo polinomial**), e escreve-se $L_1 \leq_p L_2$, se existe uma função $f : A_1^* \to A_2^*$ tal que:

- $\forall u \in A_1^* . \, u \in L_1 \iff f(u) \in L_2$
- a função f é computável em tempo polinomial, ou seja, f é calculada por um algoritmo $\mathcal{T}$ tal que $\exists k \in \mathbb{N} . \, tc_{\mathcal{T}}(n) \in \mathcal{O}(n^k)$

Teoremas:

Sejam L_1, L_2, L_3 linguagens.

1. Se $L_1 \leq_p L_2$ e $L_2 \leq_p L_3$, então $L_1 \leq_p L_3$
2. Se $L_1 \leq_p L_2$ e $L_2 \in \mathrm{P}$, então $L_1 \in \mathrm{P}$

Uma linguagem L diz-se:

- NP-difícil se $L' \leq_p L$ para toda linguagem $L' \in \mathrm{NP}$.
- NP-completa se L é NP-difícil e $L \in \mathrm{NP}$.

Teoremas:

Sejam L e K linguagens:

- Se L é NP-difícil e $L \leq_p K$, então K é NP-difícil
- Se L é NP-completa, então $L \in \mathrm{P}$ se e só se $\mathrm{P} = \mathrm{NP}$.

O problema SAT, de decidir se uma fórmula lógica em forma normal conjuntiva admite alguma valoração das variáveis que a satisfaça é NP-completo.

Teste 2017-2018

2.b)

Suppose $(*) \quad A(3, y) = 2^{y+3}$

Prove $A(4, y) = 2 \uparrow\uparrow (y + 3) - 3$.

Using induction on y:

- $y = 0$:
  $A(4, y) = A(4, 0) = A(3, 1) \overset{*}{=} 2^4 - 3 = 2 \uparrow\uparrow 3 - 3$
  which proves the property for $y = 0$.
- Let $y \in \mathbb{N}_0$ and suppose, as the induction hypothesis,

  $$A(4, y) = 2 \uparrow\uparrow (y + 3) - 3$$

  We want to prove that

  $$A(4, y + 1) = 2 \uparrow\uparrow (y + 4) - 3$$

  Well,

  $$A(4, y + 1) \overset{2.ii}{=} A(3, A(y, 4)) \overset{*}{=} 2^{A(4,y)+3} - 3 \overset{\text{hip.}}{=} 2^{2 \uparrow\uparrow (}$$