De problemas de decisão decidíveis

Seja L uma linguagem recursiva. Dada uma palavra w, tem-se $w \in L$

De problemas de decisão indecidíveis

- Dada uma palavra $w \in x.y^*$, tem-se $w \in \mathrm{AutoAceite}$?
    - devido a $\mathrm{AutoAceite}$ ser não recursiva. Não existe algoritmo que decida $\mathrm{AutoAceite}$. No entanto esta linguagem é recursivamente enumeravel. Diz-se então que o problema é semi-decidível (isto significa que existe uma máquina de Turing que permite responder nos casos afirmativos, ou seja, nos casos em que w é uma palavra de $\mathrm{AutoAceite}$).
- $\mathrm{Aceita}_w(\mathcal{T})$ [Aceitação]
    - $\mathrm{AceitaTudo}(\mathcal{T})$
    - $\mathrm{Aceita}_\varepsilon(\mathcal{T})$
- $\mathrm{Para}_w(\mathcal{T})$ [Paragem]
    - $\mathrm{AceitaNada}(\mathcal{T})$
    - $\mathrm{Para}_\varepsilon(\mathcal{T})$
- $\mathrm{Equiv}(\mathcal{T}_1, \mathcal{T}_2)$: "$L(\mathcal{T}_1) = L(\mathcal{T}_2)$"
- $\mathrm{Sub}(\mathcal{T}_1, \mathcal{T}_2)$: "$L(\mathcal{T}_1) \subseteq L(\mathcal{T}_2)$"
- "$L(\mathcal{T}_1) \cap L(\mathcal{T}_2) = \emptyset$"

De linguagens não recursivamente enumeráveis

- $\mathrm{NãoAutoAceite}$

De linguagens recursivamente enumeráveis não recursivas

- $\mathrm{AutoAceite}$

De problemas indecidíveis sobre linguagens recursivamente enumeráveis

- $\varepsilon \in L$
- $L = \emptyset$
- $L = A^*$

De prova usando teorema de Rice

Diga se a afirmação é verdadeira ou falsa justificando.

O problema "Dada uma máquina de Turing $\mathcal{T}$, será que $L(\mathcal{T}) \subseteq a^*$?" e decidıvel

Seja $d = L(\mathcal{T})$

Note que $d \in D = \{L : L \text{ é uma linguagem recursivamente enumerável}\}$

Seja $P(x)$: "$x \subseteq a^*$" para $x \in D$.

Note que P não é trivial porque $a^*, b^* \in D$ mas $P(a^*)$ é verdade e $P(b^*)$ é falso.

Logo, pelo Teorema de Rice, P é indecidível.

Observações

- $\mathrm{AutoAceite}$ contém palavras que codificam máquinas de Turing que reconhecem sua codificação.

Máquinas Auxiliares

- $\mathrm{Escreve}_w$
- $\mathrm{ApagaFita}$

Definições

Função característica

Seja L uma linguagem sobre um alfabeto A. A função característica de L é a função

$$\chi_L : A^* \to \{0, 1\}$$

definida para cada $u \in A^*$, por

$$\chi_L(u) = \begin{cases} 1 \text{ se } u \in L \\ 0 \text{ se } u \notin L \end{cases}$$

Definição 1

Seja $L \subseteq A^*$ uma linguagem e seja $\mathcal{T}$ uma máquina de Turing com alfabeto de entrada A. Diz-se que

- $\mathcal{T}$ **aceita** ou **reconhece** L se $L = L(\mathcal{T})$.
- $\mathcal{T}$ **decide** L se a função característica χ_L é calculada por $\mathcal{T}$.

Definição 2

Uma linguagem L diz-se

- **recursivamente enumerável** se existe uma MT que reconhece L.
- **recursiva** (ou **decidivel**) se existe uma MT que decide L.

Proposicões e Teoremas

Proposição 1. Sejam L e K linguagens sobre um alfabeto A

- Se L e K são recursivas (resp. recursivamente enumeráveis), então $L \cup K$ e $L \cap K$ são recursivas (resp. recursivamente enumeráveis).
- Se L é recursiva, então $\overline{L}$ e recursiva.

Teorema [Post, 1943]. Uma linguagem L é recursiva se e só se L e $\overline{L}$ são recursivamente enumeráveis.

Proposicão 2. Sejam P e P' dois problemas de decisão tais que $P \leq P'$

- Se P' é decidível, então P é decidível.
- Se P é indecidível, então P' é indecidível.
- Se P' e semi-decidível, então P e semi-decidível.

Teorema [Rice, 1953]. Se P é uma propriedade não trivial sobre linguagens recursivamente enumeráveis, então P é indecidível.

Convenções

Convenção 1. Assume-se que existem dois conjuntos enumeráveis

$$\mathcal{Q} = \{q_0, q_1, \ldots\} \quad \text{e} \quad \mathcal{S} = \{s_0, s_1, \ldots\}$$

tais que, para cada máquina de Turing,

$$\mathcal{T} = (Q, A, T, \delta, i, f, \Delta)$$

se tem

- $Q \subseteq \mathcal{Q}$, com $f = q_0$
- $T \subseteq \mathcal{S}$, com $\Delta = s_0$

Diz-se que $\mathcal{T}$ é normalizada se todos os estados e todos os símbolos não brancos de $\mathcal{T}$ pertencem a alguma transição.

Função Codificadora

$$\begin{aligned} c : \mathrm{MT_N} &\to \{x, y\}^* \\ \mathcal{T} &\mapsto c(\mathcal{T}) \end{aligned}$$

- $c'(q_i) = c'(s_i) = x^{i+1}$
- $c'(C) = x$, $c'(E) = x * 2$, $c'(D) = x^3$

Note-se em particular,

- $c'(\Delta) = c'(s_0) = x$ e $c'(f) = c'(q_0) = x$

A cada transição e, descrita por $\delta(q, t) = (q', t', m)$

$$c'(e) = c'(q)yc'(t)yc'(q')yc'(t')yc'(m)y$$

Depois, codifica-se a máquina de Turing $\mathcal{T}$ pela palavra

$$c(\mathcal{T}) = c'(q_i)yc'(e_1)yc'(e_2)\cdots yc'(e_k)y$$

onde q_i é o estado inicial de $\mathcal{T}$ e $e_1, e_2, \ldots, e_k$ são as transições de $\mathcal{T}$ numa ordem fixada previamente.

Pode também codificar-se cada palavra $w = r_1r_2\cdots r_n$, onde $r_i \in \mathcal{S}$, por

$$c(w) = yyc'(r_1)yc'(r_2)\cdots yc'(r_n)y$$

Quando se considera uma sequência

$$c(\mathcal{T})c(w) = c'(q_i)yc'(e_1)yc'(e_2)\cdots yc'(e_k)yyyc'(r_1)yc'(r_2)\cdots yc'(r_n)y$$

fica claro onde $c(\mathcal{T})$ termina devido ao prefixo yy de $c(w)$.

Exemplo

$$c(\mathcal{T}) = \underbrace{x^2}_{c'(q_1)}\underbrace{yx^2yxyx^3y}_{c'(e_1)}\underbrace{yx^3yx^2yx^3yx^3yx^3y}_{c'(e_2)}y\cdots$$

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.