Computabilidade para Complexidade...

Março de 2007

Armando B. Matos

Índice

Introdução

Breves notas...

Inicialmente descrevemos dois formalismos muito diferentes com vista a tentar caracterizar exactamente o conjunto de *todas as funções computáveis e totais*: a linguagem FOR e a definição clássica das funções ditas "primitivas recursivas". Estes formalismos são equivalentes, isto é, definem a mesma (e muito vasta!) classe de funções computáveis e totais, mas não incluem todas essas funções. Este insucesso é inevitável como resulta do Teorema 4, página 13, que é demonstrado com uma técnica de diagonalização.

Se não exirgirmos que as funções caracterizadas pelo modelo sejam totais, já é possível incluir todas as funções computáveis! Na realidade, usando diversos formalismos equivalentes – como as máquinas de Turing, a linguagem WHILE, processos de redução baseados no λ–calculus... – é possível caracterizar exactamente a classe de *todas* as funções computáveis (totais ou não!) por um qualquer "processo algorítmico", ver também a Tese de Church-Turing, página 18. Chamemos *universais* a estes formalismos.

Há problemas de decisão que não são decidíveis por nenhum processo algorítmico; esses problemas dizem-se indecidíveis. O primeiro problema que demonstraremos (por contradição) ser indecidível é o da "auto-paragem", Teorema 6, página 20. Mais tarde usaremos técnicas de redução entre problemas (página 6) para mostrar que há outros problemas indecidíveis; o Teorema de Rice, página 26, mostra que, qualquer que seja a propriedade das funções que consideremos (desde que haja pelo menos uma função com e outra sem essa propriedade), é indecidível saber-se se uma função dada tem essa propriedade! Na realidade, e talvez ao contrário do que se esperaria, muitos problemas naturais são indecidíveis, ver páginas 27 e 31.

Alguns conceitos que serão posteriormente estudados com mais pormenor
Visão de conjunto de alguns aspectos da computabilidade

O aluno deverá ler esta parte (páginas 3-6) no início e relê-la no fim...

Definição de função. Na teoria da computabilidade uma função implementada por uma máquina (instância) de modelo de computação está definida num ponto sse a "máquina" pára, fornecendo um resultado (o valor da função); e não está definida se a máquina não pára. Esta definição é diferente da definição matemática usual!

Funções PR e linguagem FOR. Uma classe computável de funções totais: funções "primitivas recursivas" (ou melhor, definidas por recursão primitiva). Podem também ser definidas através da linguagem FOR: equivalência. Nenhum modelo de computação caracteriza exactamente a classe das funções totais (demonstração pela técnica da diagonalização).

Sobre problemas de decisão. Um problema diz-se *de decisão* quando a resposta a cada caso particular dos dados ("instância") é SIM ou NÃO. Por exemplo: são dadas as expressões aritméticas E e F as quais podem envolver a variável x, adições e multiplicações; pergunta-se "E e F são equivalentes[1]?". Neste trabalho trataremos essencialmente de problemas de decisão[2].

Sobre problemas de decisão e linguagens. A cada problema de decisão Π está naturalmente associada a linguagem L_Π formada por todas as instâncias para as quais a resposta é SIM. Um problema de decisão é *decidível* sse a correspondente linguagem é recursiva. Um problema de decisão é *semi-decidível* sse a correspondente linguagem é recursivamente enumerável.

Alguns conceitos que serão posteriormente estudados com mais pormenor
Visão de conjunto de alguns aspectos da computabilidade
– CONTINUAÇÃO –

Sobre funções e linguagens. Nomenclatura normalmente usada:

funções recursivas, classe Rec	$\leftrightarrow$	linguagens recursivamente enumeráveis, classe RE
funções recursivas totais	$\leftrightarrow$	linguagens recursivas, classe R

Esta nomenclatura é um pouco confusa: uma função *recursiva* é (em geral) de uma função parcial, isto é, uma MT qua a define (em geral) não pára alguns dados, mas uma linguagem L diz-se *recursiva* quando existe uma MT que decide (e portanto pára sempre) se um dado elemento pertence a L.

Redução entre problemas. Notação: $A \leq B$: a linguagem (ou problema) A reduz-se à linguagem (ou problema) B; por definição isso acontece quando existe uma função computável e total $f : \Sigma^\star \to \Sigma^\star$ tal que $x \in A$ sse $f(x) \in B$.
Nota. Em geral é $f(A) \neq \Sigma^\star$, isto é, em geral f não é sobrejectiva.

Redução polinomial. Quando a redução se faz através de um algoritmo *de tempo polinomial em* $|x|$ escrevemos[3] "$A \leq_p B$.

Linguagens completas numa classe. Uma linguagem L diz-se *completa* numa classe $\mathcal{C}$ relativamente a uma redução "$\leq$" quando (i) $L \in \mathcal{C}$ e (ii) para toda a linguagem L' de $\mathcal{C}$ é $L' \leq L$. Assim, as linguagens completas são de algum modo as *mais difíceis* da classe.

Notação

Definições correspondentes a um determinado modelo: máquina de Turing, funções recursivas, programas WHILE, etc.

- $f(n)\uparrow$: a computação $f(n)$ não termina.
  Ou o valor $f(n)$ não está definido.
- $f(n)\downarrow$: a computação $f(n)$ termina com um determinado valor que não se explicita.
  Ou o valor $f(n)$ está definido (não se diz qual é).
- $f(n) = x$: a computação $f(n)$ termina com o resultado x.
  Ou o valor $f(n)$ está definido e tem o valor x.
- $\{i\}$ A função correspondente à instância do modelo que tem o número i.
- $\{i\}_m$ A função correspondente à instância do modelo que tem o número i interpretada como uma função de $\mathbb{N}^m$ em $\mathbb{N}$.

"Muitas vezes tanto faz" ou a "importância dos conceitos"

- Sabemos que a máquina de Turing é um modelo de computação universal. Mas há muitos outros: algumas linguagens de registos (como o WHILE), modelos derivados de λ-calculus, etc. Qual devemos usar? Tanto faz, são equivalentes.
- Devemos usar um ou mais que um argumentos de entrada? Tanto faz, existem isomorfismos computáveis[4] (aliás eficientes[5]) entre $\mathbb{N}$ e $\mathbb{N}^2$, etc.
- Devemos usar inteiros ou palavras do alfabeto $\{0,1\}$? Tanto faz. Toda a teoria das funções $\mathbb{N}^k \to \mathbb{N}$ pode ser reformulada em termos de funções $(\Sigma^\star)^k \to \Sigma^\star$ pois como sabemos existe uma bijecção computável entre $\mathbb{N}$ e $\Sigma^\star$. **Os resultados são essencialmente os mesmos**.
- Qual o alfabeto a usar, $\{0,1\}$ ou outro? Tanto faz, podem facilmente ser "simulados" uns nos outros.
- Se usarmos a máquina de Turing como modelo de computação, devemos usar uma fita de trabalho infinita nas duas direcções ou só infinita para a direita? Tanto faz, uma computação num destes tipos de máquina pode ser simulado no outro tipo (aliás de modo eficiente).

[4] Também se diz "efectivos" ou "algorítmicos".
[5] Isto é, computáveis em tempo polinomial no tamanho dos dados.

Sobre linguagens

Como sabemos, uma linguagem definida num alfabeto é um conjunto de palavras desse alfabeto. Geralmente consideraremos o alfabeto $\{0,1\}$. Uma linguagem L diz-se *recursiva* se existe uma máquina de Turing M tal que $M(x) = 1$ se $x \in L$ e $M(x) = 0$ se $x \notin L$ ($M(x)$ pára sempre), ver página 23 e a tese de Church-Turing, pág 18. Uma linguagem L diz-se *recursivamente enumerável* (RE) se existe uma máquina de Turing M tal que $M(x) = 1$ se $x \in L$ e $M(x) \uparrow$ se $x \notin L$.

Sobre problemas de decisão

Num problema de decisão faz-se uma pergunta sobre uma instância (dados). Por exemplo para o PGP (problema geral da paragem)
Dados: Um par de inteiros $\langle i, j\rangle$.
Pergunta: A máquina de Turing caracterizada pelo índice i pára com os dados (conteúdo inicial da fita) j? Em símbolos: $\boxed{\{i\}(j)\downarrow}$?
A cada problema π de decisão está associada uma linguagem L_Π que é o conjunto de todas as instâncias[6] para as quais a resposta é Sim. Por exemplo

$$L_{\mathsf{PGP}} = \{\langle i, j\rangle \ : \ \{i\}(j)\downarrow\}$$

Um problema π diz-se *decidível* se existe uma função computável (total) $f(x)$ tal que para cada instância x é $f(x) = 1$ se a resposta a x é Sim e $f(x) = 0$ se a resposta é Não e diz-se *semi-decidível* se existe se existe uma função computável (em geral parcial) $f(x)$ tal que para cada instância x é $f(x) = 1$ se a resposta a x é Sim e $f(x) \uparrow$ se a resposta é Não. Assim, *um problema* Π *é decidível sse* L_Π *é uma linguagem recursiva e é semi-decidível sse* L_Π *é uma linguagem RE*

Outro problema de decisão e uma redução

Para o PAP (problema da auto-paragem) temos
Dados: Um inteiro i.
Pergunta: A máquina de Turing caracterizada pelo índice i pára quando os dados (conteúdo inicial da fita) são esse mesmo índice i? Em símbolos: $\boxed{\{i\}(i)\downarrow}$?

Para mostrar que L_{PAP} se reduz a L_{PGP} ou, simbolicamente $L_{\mathsf{PAP}} \leq L_{\mathsf{PGP}}$, temos que transformar algoritmicamente cada instância i de PAP numa instância $\langle a, b\rangle$ de PGP

$$f : \ i \to \langle a, b\rangle$$

tal que a resposta a i (em PAP) seja Sim sse a resposta a $f(i)$ (em PGP) for Sim. Esta redução é trivial (porquê?) e é a seguinte

$$f(i) = \langle i, i\rangle$$

Observação: como acontece frequentemente, o contra-domínio é apenas uma parte do conjunto das instâncias possíveis do problema (neste caso é constituído pelas instâncias $\langle i, j\rangle$ de PGP em que $i = j$).

[6] Na realidade as instâncias devem ser codificadas, isto é, representadas por palavras do alfabeto. Todavia a codificação não é importante uma vez que as linguagens que correspondem a diferentes codificações são equivalentes no sentido em que existe um algoritmo que transforma uma codificação noutra.

Funções PR e linguagem FOR

Funções definidas por recursão primitiva também conhecidas (inglesismo) por "primitivas recursivas" ou PR:
Tentativa de englobar todas as funções totais computáveis numa mesma definição. Falhou!
Funções $\mathbb{N}^m \to \mathbb{N}$ com $m \geq 1$.
Funções de base As seguintes funções são PR.

1. *Função 0*. $0(x) = 0$
$$0 \ : \ \mathbb{N} \to \mathbb{N}$$
2. *Função sucessor*, $s(x) = x + 1$
$$s \ : \ \mathbb{N} \to \mathbb{N}$$
3. *Projecção*. As funções $\pi^n_m \ : \ \mathbb{N}^n \to \mathbb{N}$ onde $1 \leq m \leq n$, sendo
$$\pi^n_m(x_1, x_2, \ldots, x_n) = x_m$$

Funções PR / Regras para a formação de novas funções PR

1. *Composição*. Dada a função PR $f \ : \ \mathbb{N}^n \to \mathbb{N}$ e as n funções PR $g_i \ : \ \mathbb{N}^m \to \mathbb{N}$ para $i = 1, \ldots, n$ então a função
$$\text{comp}^n_m(f, g_1, \cdots, g_n) \ : \ \mathbb{N}^m \to \mathbb{N}$$
definida por
$$\text{comp}^n_m(\mathbf{x}) = f(g_1(\mathbf{x}), \cdots, g_n(\mathbf{x}))$$
onde $\mathbf{x}$ designa o vector de variáveis $(x_1, \cdots, x_m)$, também é PR.
2. *Recursão primitiva*. Sejam 2 funções PR $f \ : \ \mathbb{N}^n \to \mathbb{N}$ e $g \ : \ \mathbb{N}^{n+2} \to \mathbb{N}$ onde $n \geq 1$. Então a função $h \ : \ \mathbb{N}^{n+1} \to \mathbb{N}$ definida abaixo (onde $\mathbf{x}$ representa $x_1, \cdots, x_m$) também é PR.
$$\begin{cases} h(\mathbf{x}, 0) = f(\mathbf{x}) \\ h(\mathbf{x}, y+1) = g(\mathbf{x}, y, h(\mathbf{x}, y)) \end{cases}$$

Exercícios PR

Exercício 1 *Mostre que as seguintes funções são PR.*

(a) $f(x) = x$
(b) $s(s(x))$
(c) $f(x) = 2$
(d) $f(x, y) = g(y, x)$ *onde* g *é PR. Sug. Usar a composição.*
(e) $f(x, y) = x + y$
(f) $f(x, y) = x \times y$
(g) $sig = \begin{cases} 1 & \text{se } x = 0 \\ 0 & \text{se } x \geq 1 \end{cases}$
(h) A função $f(x, y) = g(x)$ *(independente de* y*) supondo que* g *é PR.*

A linguagem FOR

Trata-se de uma linguagem de registos. Um registo pode conter qualquer inteiro não negativo. Há uma infinidade de registos x_0, x_1, x_2, x_3, $\cdots$ Convenciona-se que quando um programa calcula uma função $\mathbb{N}^k \to \mathbb{N}$, os valores iniciais dos k argumentos estão por ordem em x_1, x_2,..., x_k e que o resultado fica em x_0; Todos os restantes argumenos contêm inicialmente 0.
As instruções possíveis são

- `x++`: o conteúdo do registo `x` é incrementado de uma unidade.
- `x--`: o conteúdo do registo `x` é decrementado de uma unidade, convencionando-se que $0-1=0$.
- `FOR x{P}` em que `P` é um programa FOR arbitrário: o programa `P` é executado um número de vezes que é o valor *inicial* de `x`. No fim `x` fica com o valor 0. Dentro do ciclo o registo `x` não pode ser referido.

Um programa FOR é uma sequência de 0 ou mais instruções (o programa "nulo" é pois permitido)
O seguinte programa

```
FOR x1{x3++;  x4++}
FOR x3{
  FOR x2{x0++}
}
x4--; x4--; x4--;
FOR x4 {FOR x0 {}}
```

calcula a função

$$f(x,y) = \begin{cases} 0 & \text{se } x = 0 \text{ ou } x \geq 4 \\ y & \text{se } 1 \leq x \leq 3 \end{cases}$$

FOR=PR

FOR: classe das funções $\mathbb{N}^k \to \mathbb{N}$ implementáveis em linguagem FOR.
PR: classe das funções definidas por recursão primitiva

Teorema 1 *FOR=PR*

Demonstração Damos só uma ideia da demonstração de que

$$\text{PR} \subseteq \text{FOR}$$

Seja $f \in$PR. A prova é por indução no comprimento da definição de f.

- Funções base.
  1. $0(x) \ : \ \mathbb{N} \to \mathbb{N}$. **Exercício**
  2. $s(x) \ : \ \mathbb{N} \to \mathbb{N}$. **Exercício**
- Projecções. $\pi_m^n(x_1, x_2, \ldots, x_n) = x_m$. **Exercício**
- Composição. Ideia através de um exemplo.

  $$\text{comp}_1^2(f, g_1, g_2) = f(g_1(x), g_2(x))$$

  Um programa FOR que implementa a função composta pode ter a seguinte estrutura.
  1. *Cálculo de* $g_1(x)$
     Inicialmente x está no registo x1. Este valor é copiado para x2. Existe por hipótese um programa FOR que calcula $g_1(x)$ (sem usar x2). O resultado, x0 é copiado para x3.
  2. *Cálculo de* $g_1(x)$
     x2 é copiado para x1 (que fica com o valor inicial, x) e corre-se um programa FOR que calcula $g_2(x)$; supomos que este programa não usa x3 (o que é sempre possível).
  3. *Cálculo de* $f(\cdot, \cdot)$
     Neste momento x3 tem $g_1(x)$ e x0 tem $g_2(x)$. Copia-se x3 para x1 e x0 para x2. Corre-se um programa FOR que calcula $f(x_1, x_2)$, isto é, $f(g_1(x), g_2(x))$.
- Recursão primitiva.
  Exercício. Sugestão: usar um ciclo em y.

O modelo FOR é incompleto

Vamos ver que existem funções totais computáveis não implementáveis em FOR. Mais tarde veremos que isto não é um defeito do FOR: isto acontece sempre para qualquer modelo de funções totais computáveis.
Um exemplo de uma função total computável mas não implementável em FOR é a função de Ackermann.

$$\begin{array}{lcl} A(0,y) & = & y+1 \\ A(x+1,0) & = & A(x,1) \\ A(x+1,y+1) & = & A(x,A(x+1,y)) \end{array}$$

Exercício 2 *Determine os valores de* $A(i,j)$ *para* $0 \leq i,\ j, \leq 3$.

Exercício 3 *Escreva uma função em linguagem C que implemente* A*, supondo que a precisão dos inteiros é arbitrária.*

Exercício 4 *Determine formas fechadas para* $A(0,j)$, $A(1,j)$ *e* $A(2,j)$.

Porque é que $A(i,j)$ não é implementável em FOR? Ideia: cresce mais rapidamente que qualquer função PR. Os parâmetros especificam o grau de imbricamento necessário das instruções de ciclo (FOR). Mas este número é fixo, o programa é fixo!

Generalização!

Seja $\mathcal{M}$ um *qualquer modelo de computação que inclua apenas funções computáveis totais*. Então $\mathcal{M}$ é incompleto no sentido em que existem funções computáveis totais não implementáveis (representáveis) em $\mathcal{M}$. Vamos ver que é assim no Teorema 4, página 13.
A hipótese seguinte é muito geral. Poderia ser de outra maneira???

Hipótese. As instâncias (casos particulares) de qualquer modelo de computação $\mathcal{M}$ são codificáveis, isto é, existe um alfabeto Σ e uma função injectiva computável (codificação)

$$c \ : \ \mathcal{M} \to \Sigma^\star$$

Existe também uma função computável (sintaxe)

$$\text{teste} \ : \ \Sigma^\star \to \{\text{F}, \text{V}\}$$

tal que teste$(x) =$V sse x é uma codificação de alguma instância do modelo.

O aluno deve comprender esta hipótese!

Cada programa FOR é representado - numa certa codificação - por uma palavra de um certo alfabeto. O mesmo se passa para todos os modelos de computação.
Vamos, usando esta hipótese, tirar conclusões muito gerais.

Codificações, índices, bijecções, diagonalização

Codificação de MTs

Usamos o alfabeto

$$\{\texttt{,}, \texttt{(}, \texttt{)}, \texttt{\{}, \texttt{\}}, \texttt{s}, \texttt{0}, \texttt{1}, \texttt{b}, \texttt{+}, \texttt{-}, \texttt{>}\}$$

representando, por exemplo o estado s_6 por `s110`, etc.
Por exemplo, uma MT particular é representada pela seguinte palavra; usamos

$$M = (\Sigma, \Gamma, b, S, s_0, F, \delta)$$

`"({0,1},{0,1,b},b,{s0,s1},s0,{s1},{(s0,0)->(s0,0,+1),(s0,1)->(s1,0,+1)})"`

ou, usando $\Sigma = \{0, 1\}$ e representando "," por 0000, "(" por 0001,..., "-" por 1010

`"000100110110...0010"`

Exercício 5 *Completar a palavra anterior*

Resultados gerais - $\Sigma^\star \leftrightarrow \mathbb{N}$

"Tanto faz usar palavras como inteiros"

Teorema 2 *Para qualquer alfabeto Σ e inteiro $n \geq 1$ existe uma função computável e bijectiva $f : \Sigma^\star \leftrightarrow \mathbb{N}^n$*

Exercício 6 *Defina uma função bijectiva computável entre $\mathbb{N}^2$, e $\mathbb{N}$. Sugestão: a cada par (l, c) corresponde biunívocamente um inteiro não negativo conforme indicado*

l c :	0	1	2	3	4	···
0	0	1	3	6	10	···
1	2	4	7	11	···	···
2	5	8	12	···	···	···
3	9	13	···	···	···	···
4	14	···	···	···	···	···
···	···	···	···	···	···	···

Exercício 7 Codificação de Cantor. *Usando o resultado do exercício anterior defina funções bijectivas computáveis entre $\mathbb{N}^k$, e $\mathbb{N}$ para $k \geq 3$. Sugestão: Use composição de funções.*

Exercício 8 *Defina uma função injectiva computável entre $\{0, 1\}^\star$, e $\mathbb{N}$ (cuidado com os "zeros à esquerda")*

Inteiro que descreve um programa

Teorema 3 *Seja um modelo de computação $\mathcal{M}$ e uma codificação c. Existe uma função bijectiva $f : c(\mathcal{M}) \leftrightarrow \mathbb{N}$ onde $c(\mathcal{M})$ representa o domínio da codificação, isto é, o conjunto das palavras que representam instâncias do modelo.*

Demonstração baseada num programa. Seja "teste" a função de teste da codificação. A ordem ("order" nos algoritmos em baixo) considerada para as palavras é: ordem crescente de comprimentos e, dentro de cada comprimento, ordem lexicográfica, isto é

$$\varepsilon, 0, 1, 00, 01, 10, 11, 000, \cdots$$

```
f(word x){
/* Dá o índice da codificação x  */
   i=0;
   for all y in order
     if teste(y){
        if x==y return i;
        i=i+1;
     }
```

Para calcular f^{-1}

```
f1(int i){
/* Dá a codificação de Índice i  */
   j=0;
   for all y in order
     if teste(y){
        if i==j return y;
        j=j+1;
     }
```

Assim podemos falar da instância número i
MT número i
Programa número i
Definição número i
...
Resulta...

Corolário 1 *Seja qual for o modelo de computação $\mathcal{M}$ existe uma função bijectiva $f : \mathcal{M} \leftrightarrow \Sigma^\star$*

A função (eventualmente) calculada pela instância do modelo com o número i designa-se por $\{i\}$ (ou por ϕ_i); por exemplo, se essa função é o sucessor temos por exemplo $\boxed{\{i\}(5) = 6}$ (ou $\boxed{\phi_i(5) = 6}$).

O método da diagonalização

Usado por Cantor para provar que o conjunto dos inteiros não é isomorfo ao dos reais.
Vamos usá-lo para mostrar o resultado já enunciado

Teorema 4 *Seja $\mathcal{M}$ um qualquer modelo de computação (ou de representação) que inclua apenas funções computáveis totais. Então $\mathcal{M}$ é incompleto no sentido em que existem funções computáveis totais não implementáveis (representáveis) em $\mathcal{M}$.*

Demonstração. Seja a função g assim definida onde f é a função que associa a cada inteiro a respectiva instância (f depende de $\mathcal{M}$; $f(i)$ é basicamente a função $\{i\}$)

$$g(i) = f(i)(i) + 1$$

isto é, a instância número i é aplicada ao inteiro i e soma-se 1.
A função g é computável. Se g fosse implementada (definida, etc.) por alguma instância do modelo - seja k o seu número - tínhamos

$$\forall i \geq 0 \; g(i) = f(k)(i)$$

Mas em particular, poderíamos tomar $i = k$ donde resulta

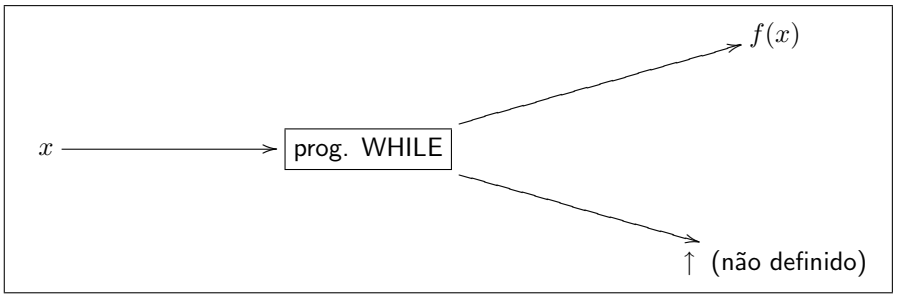
$$g(k) = f(k)(k)$$

Mas isto contradiz a definição de g. ◇
Para o Teorema "funcionar" é importante que $\mathcal{M}$ só caracterize funções totais!

Não definido = não termina

Há modelos de computação que (aparentemente) caracterizam todas as funções totais computáveis. Mas caracterizam também obrigatoriamente funções parciais.
Princípio da equivalência entre indefinibilidade e não terminação

não termina = não definido



Linguagem WHILE e funções recursivas

A linguagem WHILE

A linguagem WHILE caracteriza exactamente as funções recursivas – isto é as funções (em geral parciais) que são implementáveis por um qualquer processo algorítmico. Veremos outro método de caracterizar a mesma classe de funções.

As convenções sobre o resultado e argumentos das funções, valores iniciais, etc. são semelhantes às da linguagem FOR.
Um programa é uma sequência de instruções. Uma instrução pode ser de um dos tipos seguintes.

- `x++`: como na linguagem FOR.
- `x--`: como na linguagem FOR.
- `WHILE x{P}`: O programa `P` é executado enquanto for $\mathtt{x} > 0$. Nota: ao contrário do que passa com a linguagem FOR, a variável `x` pode ser (e muitas vezes é) modificada dentro de `P`.

Exercícios sobre a linguagem WHILE

Exercício 9 *Mostre que todo a função implementável em linguagem* FOR *é também implementável em linguagem* WHILE.

Exercício 10 *Mostre que existem funções implementáveis em linguagem* WHILE *que não são implementáveis em linguagem* FOR.

Exercício 11 *Como implementaria em linguagem* WHILE *uma pilha ("stack")? Note que em qualquer programa o número de registos utilizado é finito e que não existe indexação.*
Sugestão. Use um inteiro, fazendo o "push" com multiplicação+adição.

Exercício 12 *Mostre que uma instrução do tipo*

$$\mathtt{xi} \leftarrow \mathtt{xj+k}$$

onde "`k`*" é uma constante, pode ser implementada em WHILE.*

Exercício 13 *Implemente em* WHILE *a seguinte função*

$$f(n) = \begin{cases} 0 & \text{se } n \text{ é par} \\ \uparrow & \text{se } n \text{ é ímpar} \end{cases}$$

Minimização

Todas as funções PR são totais. Vamos introduzir um novo operador que permite a definição de funções parciais.
Notação. Se A e B são conjuntos, $f : A \longrightarrow_p B$ designa uma função não necessariamente total de A em B.

Operador de minimização

Seja $f : \mathbb{N}^{n+1} \longrightarrow_p \mathbb{N}$. Define-se $g : \mathbb{N}^n \longrightarrow_p \mathbb{N}$ por *minimização* como:

$$g(\mathbf{x}) = \text{menor } y \text{ tal que } f(\mathbf{x}, y) = 0$$

Designa-se $g = \mu^n(f)$.
Mesmo que f seja total, a função $\mu^n(f)$ pode não o ser.

Exercício 14 *Dê um exemplo do que acabamos de afirmar.*

Mais rigorosamente

$$\begin{aligned} \mu^n(f)(\mathbf{x}) = y \qquad & \text{se } f(\mathbf{x}, y) = 0 \text{ e} \\ & \forall z,\, 0 \leq z < y,\, f(\mathbf{x}, z) \downarrow \text{ e } f(\mathbf{x}, z) \neq 0 \\ \mu^n(f)(\mathbf{x}) \uparrow \qquad & \text{caso tal } y \text{ não exista} \end{aligned}$$

Classe Rec e linguagem WHILE

Funções recursivas – definição clássica

Funções recursivas - em geral parciais
Definidas de modo análogo às funções PR (ver página 7) usando

- As mesmas funções de base.
- Os mesmos métodos de definição
- A minimização.

Por outras palavras, a classe das funções (parciais) recursivas Rec é o menor conjunto de funções de $\mathbb{N}^n$ em $\mathbb{N}$ que inclui as funções de base $0(x)$, $s(x)$ e π e que é fechado relativamente à composição, à recursão primitiva e à minimização. Uma função diz-se recursiva total se pertence a Rec e é total.

WHILE=Rec

Seja WHILE a classe das funções implementáveis na linguagem WHILE (com algum "overloading"...).
Temos o seguinte resultado

Teorema 5 *WHILE=Rec*

Demonstração Não incluída.

Funções recursivas - exercícios

Exercício 15 *(O exercício mais fácil jamais visto...)*
Mostre que a classe de funções PR está incluído na classe Rec

Exercício 16 *Mostre que a classe de funções PR está incluído propriamente na classe Rec*

Exercício 17 *Mostre que as seguintes funções são recursivas. Pressupomos que se sabe que determinadas funções são PR.*

1. *A função totalmente indefinida* $\uparrow : \mathbb{N} \to \mathbb{N}$
2. *A função* $f(n) = \begin{cases} 0 & \text{se } n \leq 5 \\ \uparrow & \text{se } n \geq 6 \end{cases}$

Funções recursivas totais

A classe PR está propriamente incluída na classe das funções recursivas totais, RecT. Por exemplo, a função de Ackermann é recursiva total. *Note que não existe uma caracterização "sintáctica" de RecT, ver Teorema 4, página 13.*
Aparentemente, qualquer função parcial computável, em particular, qualquer função total computável parece ser recursiva.

Exercício 18 *Porque é que o método da diagonalização não se aplica às funções recursivas? Em particular considere o seguinte esquema diagonalização onde* $\{i\}$ *representa o modelo (definição de função recursiva) com o índice* i *e se pretende mostrar que existem sempre funções computáveis não representadas no modelo.*

$$f(i) = \begin{cases} 0 & \text{se } \{i\}(i) \uparrow \\ \uparrow & \text{se } \{i\}(i) \downarrow \end{cases}$$

A função f *é alguma das funções* $\{j\}$*? A função* f *é recursiva (implementável)?*

Modelos universais e tese de Church-Turing

Modelo universal: qualquer modelo de computação equivalente às máquinas de Turing. Há muitos.

Tese de Church-Turing (versão para funções): Se uma função (parcial, em geral) é computável por um qualquer processo algorítmico, então também é computável por uma máquina de Turing determinística.
Diz-se que uma função (parcial, em geral) $f(x)$ é computável por um processo algorítmico quando se verificam as seguintes condições.

- Se $f(x)\downarrow$ e $f(x)=y$, o "processo algorítmico" termina com resultado y.
- Se $f(x)\uparrow$ ($f(x)$ não está definida) o "processo algorítmico" não termina.

Assim

não terminação da computação do processo algorítmico $\iff$ não definibilidade da função

Tese de Church-Turing (versão para linguagens): Se uma linguagem é aceite por um qualquer processo algorítmico, então também é aceite por uma máquina de Turing determinística.

Diz-se que uma linguagem L é *aceite* por um modelo de computação algorítmico se, a computação termina com resultado "SIM" quando $x \in L$ e não termina caso contrário.

Em termos de **classes de linguagens equivalentes**:

$$
\begin{array}{rl}
 & \text{Linguagens recursivamente enumeráveis} \\
= & \text{Linguagens reconhecidas por máquinas de Turing} \\
= & \text{Linguagens do tipo 0} \\
= & \text{Linguagens de Post, } L(\mathrm{PS}, T) \\
= & \text{Linguagens definidas por sistemas de Markov} \\
= & \text{Linguagens definidas por sistemas } \mathbf{EIL}
\end{array}
$$

Classes de funções e linguagens

Seja

- FOR - Classe das funções que podem ser implementadas em linguagem FOR.
- PR - Classe das funções definidas por recursão primitiva.
- RT - Classe das funções recursivas totais - computáveis e totais (não é uma classe que possa ser definida através de um modelo de computação).
- Rec - Classe das funções recursivas (parciais em geral) - computáveis.

Temos FOR $=$ PR $\subset$ RT $\subset$ Rec onde "$\subset$" representa a inclusão própria de conjuntos.
Uma linguagem é recursiva (classe r) sse tiver uma função característica recursiva total.
Uma linguagem é recursivamente enumerável (classe re) sse tiver uma função semi-característica recursiva.

Definindo classes recursivas...

Algumas classes de funções de $\mathbb{N}^k$ em $\mathbb{N}$ computáveis (parciais, em geral), isto é, sub-conjuntos de Rec, classe das funções recursivas.

- Funções implementáveis numa máquina de Turing $=$ para as quais existe uma MT que dá o mesmo valor da função quando ele está definida e não pára quando não está.
- Funções implementáveis em linguagem WHILE.
- Funções implementáveis em linguagem C (sem limites no valor dos inteiros).
- Funções definíveis em λ-calculus puro.

Todas estas classes são iguais e, pela tese de Church-Turing, iguais a Rec.

Funções não computáveis / Problemas indecidíveis

Exemplo 1 Uma diagonalização

$$f(n) = \begin{cases} \{n\}(n)+1 & \textit{se } \{n\}(n)\downarrow \\ 0 & \textit{caso contrário} \end{cases}$$

Teorema 6 Indecidibilidade do problema da auto-paragem, "self-halting problem"
O problema $\boxed{\{n\}(n)\downarrow}$ *é indecidível. Não existe nenhuma função recursiva total, seja* f *tal que*

$$f(n) = \begin{cases} 1 & \textit{se } \{n\}(n)\downarrow \\ 0 & \textit{caso contrário} \end{cases}$$

Demonstração. Por contradição. Supomos que existe f e seja k um seu índice.
Podemos definir a seguinte função computável (recursiva)

$$f'(n) = \begin{cases} \uparrow & \text{se } \{n\}(n)\downarrow \\ 0 & \text{caso contrário} \end{cases}$$

Seja M' uma MT que calcula f' e p um seu índice. Considere-se a computação

$$\{p\}(p)$$

Chega-se a uma contradição. ◇

Exercício 19 *PGP, Problema geral da paragem, "halting problem"*
Considere o problema "dados n e m, $\boxed{\{n\}(m)\downarrow}$". Mostre que este problema, o "problema geral da paragem" é indecidível, usando uma redução $PAP \leq PGP$.

Exercício 20 *Paragem com fita vazia*
Mostre que o seguinte problema é indecidível:

Dado n, $\boxed{\{n\}(0)\downarrow}$

Sugestão Defina uma redução

$$PAP \leq PFV$$

Linguagens não recursivas

As seguintes linguagens não são recursivas.

1. $\{n \mid \{n\}(n)\downarrow\}$
2. $\{p(n,m) \mid \{n\}(m)\downarrow\}$ onde $p(-,-)$ é uma bijecção de Cantor de $\mathbb{N}^2$ em $\mathbb{N}$.
3. $\{n \mid \{n\}(0)\downarrow\}$

Exercício 21 Mostre que todas as linguagens referidas são RE

Problemas indecidíveis

Demonstração de indecidibilidade

¡Como se mostra que o problema P é indecidível? Há vários processos.

1. Directamente: Supor que P é decidível e chegar a uma contradição. Foi o que fizemos para o problema da auto-paragem.
2. Fazer uma redução de Turing $Q \leq P$ onde Q é um problema que já sabemos ser indecidível.
3. Usar outros resultados, Teorema de Rice.

Definição 1 *Diz-se que se faz uma redução de Turing do problema P ao problema Q e escrevemos $P \leq Q$ se definirmos uma função computável (total) das instâncias de P nas instâncias de Q*

$$f : P \to Q$$

tal que, qualquer que seja a instância x de P, a resposta a x é SIM sse a resposta a $f(x)$ é SIM.

Exercício 22 Mostre que se $P \leq Q$ e P é indecidível então Q também é indecidível.

Versões para linguagens

Exercícios de revisão

Exercício 23 Considere uma variedade $\mathcal{V}$ de máquinas de Turing em que o alfabeto da fita é $\Gamma = \{b, a\}$ (b é o "branco").

1. Em que sentido é que se pode dizer que o tipo $\mathcal{V}$ de MTs tem a capacidade computacional de calcular todas as funções recursivas.
2. Considere uma máquina de Turing em que $\Gamma' = \{0, 1, 2\}$ (0 é o "branco"). Mostre como poderia simular uma computação desta máquina numa do tipo $\mathcal{V}$. Considere por exemplo a tradução de uma transição do tipo $\delta(1, s) = (0, s', \rightarrow)$.

Exercício 24 Mostre que a seguinte função onde k é uma constante é primitiva recursiva; pode usar a definição ou implementá-la em linguagem FOR (sem acrescentos...).

$$f(n) = \begin{cases} 1 & \text{se } n = k \\ 0 & \text{caso contrário} \end{cases}$$

Exercício 25 Usando o resultado do exercício anterior mostre que se $f(x)$ é PR e $g(x)$ só difere de $f(x)$ num conjunto finito de pontos, então g também é PR; sabe-se à partida que a multiplicação e adição são PR.

Exercício 26 Mostre directamente que as seguintes funções são PR:

1. $f(x) = x - 1$, sendo $f(0) = 0$.
2. $f(x, y) = x - y$,

Exercício 27 Suponha que $f(x, y)$ é recursiva. Mostre que o seguinte programa calcula também uma função recursiva.

```
g(int x){ int t;
  y=0;
  while((t=f(x,y))!=0)
  /* O cálculo de f(x,y) pode não terminar! */
    y=y+1;
  return(y); }
```

Exercício 28 Mostre que a seguinte linguagem é recursivamente enumerável mas não recursiva $\{n \mid \{n\}(n) \downarrow\}$ Sugestão Resolva uma questão equivalente relativa à decidibilidade de problemas.

Exercício 29 Defina modelo universal de computação de funções.

Exercício 30 Mostre que não existe nenhum modelo de computação (com as características que geralmente lhes são atribuídas) que inclua exactamente as funções recursivas totais.

Computabilidade

Função característica de $A \subseteq \mathbb{N}$ é a função (total) (verde ou vermelho)

$$ch_A(n) = \begin{cases} 1 & n \in A \\ 0 & n \notin A \end{cases}$$

Função semi-característica s_A é a função (parcial) (verde ou nada)

$$s_A(n) = \begin{cases} 1 & n \in A \\ \uparrow & n \notin A \end{cases}$$

$A \subseteq \mathbb{N}$ é *recursivo*: se a sua função característica é recursiva.
$A \subseteq \mathbb{N}$ é *recursivamente enumerável (RE)* se a sua função semi-característica s_A é recursiva.

Máquinas de Turing universais

MT universal U. Simula qualquer MT. Seja qual for $n, m \in \mathbb{N}$ é

$$\{u\}\langle n, m\rangle = \{n\}(m)$$

onde u é um índice de U e $\langle n, m\rangle$ é um inteiro que representa o par (ex: bijecção de Cantor).

O teorema de Post

Teorema 7 Teorema de Post
Um conjunto $A \subseteq \mathbb{N}$ é recursivo se e só se ele e o seu complementar forem recursivamente enumeráveis.

Exercício 31 Demonstre o Teorema de Post. Pode usar processos informais de cálculo e usar a tese de Church-Turing.

Exercício 32 Suponha que os conjuntos $A, \subseteq \mathbb{N}$ são recursivos (RE). Mostre que $A \cup B$ e $A \cap B$ são recursivos (respectivamente RE).

Exercício 33 Seja Mostre que o conjunto $\{n \mid \{n\}(n) \uparrow\}$ não é recursivo nem RE

O Teorema S-m-n

O seguinte teorema não é profundo mas facilita demonstrações.

Teorema 8 Teorema S-m-n
Sejam $m, n \geq 1$. Existe uma função primitiva recursiva S_m^n tal que para qualquer índice e, $\mathbf{x} \in \mathbb{N}^m$ e $\mathbf{y} \in \mathbb{N}^n$,

$$\{e\}_{m+n}(\mathbf{x}, \mathbf{y}) \quad = \quad \{S_n^m(e, \mathbf{x})\}(\mathbf{y})$$

Demonstração. Ideia (sem mostrar que é PR). Dado $\mathbf{x}$ e e, a função S_n^m calcula o índice de uma máquina de Turing que, perante qualquer dado $\mathbf{y}$, começa por gerar o par $(\mathbf{x}, \mathbf{y})$ e depois se comporta como a máquina de índice e. ◇

O nome "recursivamente enumerável". Conceito de MT enumeradora.

Teorema 9 *Um conjunto A é RE sse existir uma MT enumeradora que enumera os seus elementos.*

Teoremas dos domínios e contradomínios

O que significa aqui domínio e contradomínio?

Teorema 10 *O conjunto $A \subseteq \mathbb{N}$ é RE se e só se A é o domínio duma função recursiva parcial.*

Demonstração. Sentido $\rightarrow$: Se A é RE existe uma função recursiva parcial $f(x)$ tal que $f(x) = 1$ se $x \in A$ e $f(x) \uparrow$ se $x \notin A$. Obviamente o domínio de f é A.
Sentido $\leftarrow$: Suponhamos que A é o domínio duma função recursiva parcial f. A função f' tal que $f'(x) = 1$ se $f(x) \downarrow$ e $f'(x) \uparrow$ se $f(x) \uparrow$ é também recursiva parcial (porquê?), o que mostra que A é RE. $\diamond$

Teorema 11 *Um conjunto infinito A é RE se e só se é o contradomínio de uma função recursiva total.*

Demonstração. Sentido $\rightarrow$: Se A é RE existe uma função recursiva parcial $f(x)$ tal que $f(x) = 1$ se $x \in A$ e $f(x) \uparrow$ se $x \notin A$. Consideremos uma MT M' que tem como dados um inteiro n e simula simultaneamente as computações $M(\varepsilon)$, $M(0)$, $M(1)$, $M(00)\ldots$ usando a técnica "dovetailing"; M' pára quando ocorrer a paragem da n-ésima MT simulada, seja $M(x)$; o resultado é x. A MT M' corresponde a uma função recursiva total (é total devido ao facto de A ser infinito) cujo contradoínio é A.
Sentido $\leftarrow$: Seja A o contradomínio de uma função recursiva total f. Seja M uma MT que implementa f. Considere-se a função recursiva parcial f' que corresponde a uma MT M' que, dado x, simula simultaneamente as computações $M(\varepsilon)$, $M(0)$, $M(1)$, $M(00)\ldots$ usando a técnica "dovetailing" e pára se e quando uma das máquinas simuladas parar com o resultado x. Se M' parar, o resultado é 1. Então, se $x \in A$, $M'(x)$ tem o resultado 1 e, se $x \notin A$, a computação $M'(x)$ não termina oque mostra que A é RE. $\diamond$

Teorema de Rice

Seja A' o complementar do conjunto A, $A' = \mathbb{N} \setminus A$.
O Teorema de Rice diz-nos que os conjuntos de inteiros que são os índices de MTs a que correspondem funções unárias com certas propriedades são (quase sempre) não recursivos.
Por outras palavras, o seguinte problema é em geral indecídivel: Dado n, a função implementada pela MT com índice n é...?
Seja C_1 o conjunto das funções recursivas de aridade 1.

Teorema 12 Teorema de Rice
Seja $B \subseteq C_1$, $B \neq \emptyset$ e $B \neq C_1$. Então, o conjunto $\{n \in \mathbb{N} \mid \{n\} \in B\}$ não é recursivo.

Demonstração do teorema de Rice

Demonstração. Pelo enunciado $B \neq \emptyset$ e $B' \neq \emptyset$. A função totalmente indefinida $\uparrow$ ou está em B ou em B'. Seja h uma função que não está no mesmo conjunto que $\uparrow$. Sem falta de generalidade suponhamos que $\uparrow \notin B$. Defina-se

$$f(x,y) = \begin{cases} h(y) & \{x\}(x) \downarrow \\ \uparrow & \text{caso contrário} \end{cases}$$

A função f é computável. Pelo Teorema $S-n-m$, existe uma função primitiva recursiva g tal que $f(x,y) = \{g(x)\}(y)$.

$$\begin{aligned} \forall y\ \{g(x)\}(y) = h(y) \quad & se \quad \{x\}(x) \downarrow \\ \forall y\ \{g(x)\}(y) \uparrow \quad & se \quad \{x\}(x) \uparrow \end{aligned}$$

então

$$\{x\}(x) \downarrow \quad sse \quad \{g(x)\} \in B$$

Isto é reduzimos o problema da auto-paragem ao problema de saber se uma dada função está em B. Como o primeiro é indecidível o problema $\boxed{\{x\} \in B}$ é indecidível. ◇

O problema "Uma função é total" é indecidível

3 demonstrações

Mostramos que o conjunto seguinte não é recursivo

$$T = \{n \mid \{n\} \text{ é total}\}$$

1. *Não há nenhum modelo que inclua todas as funções totais computáveis.*
   Se T fosse recursivo tínhamos um modelo nessas condições - existia f total (teste sintático?) tal que $f(n) = 1$ se $x \in \Sigma^\star$ representa uma MT que implementa uma função total e $= 0$ caso contrário.

2. *Pelo teorema de Rice* - exercício.

3. *Directamente*
   $$f(x,y) = \begin{cases} 1 & \text{se } \{x\}(x) \text{ não pára em} \leq y \text{ passos} \\ \uparrow & \text{caso contrário} \end{cases}$$
   A função f é computável (**Exercício**: Explique porquê e como). Pelo Teorema S-m-n, $f(x,y) = \{g(x)\}(y)$.
   Se $\{x\}(x) \uparrow$ então $\{g(x)\}$ é total.
   Se $\{x\}(x) \downarrow$ então $\{g(x)\}$ não está definida a partir dum dado valor.

   $$\{g(x)\} \text{ é total} \quad sse \quad \{x\}(x) \uparrow$$

   Então se $\boxed{\{n\} \text{ é total}}$ é semi-decidível também o é o problema complementar da auto-paragem.

Outro problema indecidível

Equações diofantinas: é dado um conjunto de variáveis

$$\{x, y, \cdots, z\}$$

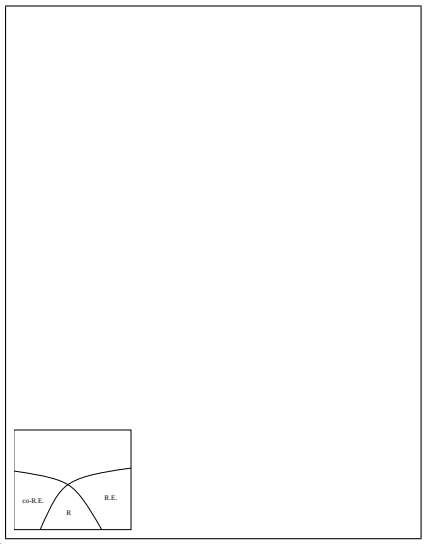
e uma equação de coeficientes inteiros nestas variáveis, p. ex.

$$1782x^3z^7 - 332x^{11}y^9 + 997z^{10} + 12331 = 0$$

Pergunta-se: A equação tem solução inteira - isto é, existem $x,\ldots, z$ que a satisfazem?
Este problema é indecidível (1971). Essencialmente porque se pode "programar" a paragem de uma qualquer MT numa equação diofantina.

Classes de linguagens; uma linguagem completa em RE

Classes de linguagens (e problemas)



Classificando o conjunto $A \subseteq \mathbb{N}$:

- R - A e A' são recursivos.
  Problema correspondente: decidível.
- RE - A é RE e não recursivo
  Problema correspondente: semi-decidível.
- co-RE - A' é RE e não recursivo
  Problema correspondente: complementar de semi-decidível.
- Nada disso - Nem A nem A' são RE

Exercícios

Exercício 34 *Classifique os seguintes conjuntos ou problemas numa das 4 classes referidas. Com excepção do c), pode usar o teorema de Rice.*

1. *Dado x, $\{x\}(x) \downarrow$ (auto-paragem)*
2. *Dado x, $\{x\}(x) \uparrow$ (co-auto-paragem)*
3. *Dado x, $\{x\}(0) \downarrow$ (paragem com dados 0)*
4. *Dado x, $\{x\}(5) = 7$.*
5. *Dado y (x fixo), $\{x\}(y) \downarrow$ (paragem com dados y).*
6. *Dado y (u é um índice de uma MT universal), $\{x\}(y) \downarrow$.*
7. *Dado x, $\{x\}(0) \uparrow$ (não pára com dados 0)*

8. *Dado x, $\{x\}$ não é a função $\uparrow$?*
9. *Dado x, $\{x\}$ é a função $\uparrow$?*
10. *Dado x, $\{x\}$ é uma função total?*

Uma linguagem completa em RE

Nota. Designaremos por RE tanto uma abreviatura de "recursivamente enumerável" ("a linguagem L é RE") como a classe das linguagens recursivamente enumeráveis.
Como sabemos, a linguagem L_{PGP} associada ao problema geral da paragem é

$$\{\langle n, m\rangle \ : \ \{n\}(m)\downarrow\}$$

Teorema 13 *A linguagem L_{PGP} é completa na classe das linguagens RE, isto é, para qualquer linguagem L de RE temos $L \leq L_{PGP}$.*

Demonstração. Seja L uma linguagem de RE. Por definição existe uma MT M_L tal que $M_L(x) = 1$ se $x \in L$ e $M_L(x)\uparrow$ caso contrário. Seja i um índice de M_L e x uma palavra qualquer. Consideremos a transformação de instâncias

$$f_i : \ x \to \langle i, x\rangle$$

Note-se que para cada linguagem L, o inteiro i é fixo. A função f_i é computável (aliás de forma trivial) e, por definição do PGP,

$$x \in L \ \longleftrightarrow \ M_L(x) = 1 \ \longleftrightarrow \ M_L(x)\downarrow \ \longleftrightarrow \ \langle i, x\rangle = f(x) \in L_{\mathsf{PGP}}$$

onde o sinal $\longleftrightarrow$ significa "sse" (se e só se). As duas primeiras equivalências resultam da definição da máquina M_L e a última das definições do problema PGP e da função f. ◇

Exercício 35 *Mostre que L_{PAP} é também um problema completo na classe RE.*
Sugestão. *Para resolver este problema siga o seguinte caminho*

- *Mostre que PGP$\leq$PAP.*
- *Mostre que a redutibilidade entre linguagens é transitiva.*
- *Use os resultados que provou para mostrar que para qualquer linguagem L de RE é $L \leq$PAP.*

Exercício 36 *Mostre que uma linguagem recursiva não pode ser completa em RE.*

Mais alguns problemas decidíveis e indecidíveis

Já vimos diversos problemas de decisão decidíveis e indecidíveis. Vamos agora mencionar mais alguns. Pensamos que o leitor saberá em cada caso caracterizar mais formalmente o problema e, em particular, de ver quais são os dados ("instância"). Usamos as seguintes abreviaturas: AFD (autómato finito determinístico), GIC (gramática independente de contexto ou "de contexto livre") e MT (máquina de Turing).

- *Problema da correspondência de Post – indecidível.*
  Damos uma ideia deste problema com um exemplo (de Rozenberg e Salomaa). Sejam as seguintes sequências de palavras de alfabeto $\{a, b\}$: $\langle aa, bb, abb\rangle$ e $\langle aa, bb, abb\rangle$. Se concatenarmos a primeira, a segunda, a primeira e a terceira palavras de qualquer uma das duas listas obtemos a mesma palavra: $aabbaaabb$; para esta instância (as duas listas) o problema da correspondência de Post tem solução.

- *A linguagem caracterizada por um AFD é recursiva.*
  Por outras palavras dado o AFD A e a palavra x, o problema "$x \in L_A$?" é decidível. O mesmo se passa se em vez de AFD's usarmos GIC's (uma dada palavra é gerada por uma dada GIC?). Muitos outros problemas relativos a linguagens regulares também são decidíveis. Um exemplo: a linguagem caracterizada por um AFD é vazia?

- *A linguagem caracterizada por uma GIC é recursiva.*
  "Uma dada palavra é gerada por uma dada GIC?".

- *Equivalência de dois DFA's – decidível.*
  Dois AFD's dados são equivalentes (caracterizam a mesma linguagem)?

- *Equivalência de duas GIC's – indecidível.*

- *Equivalência de duas MT's – indecidível.*

- *Equivalência de duas expressões em* $(\mathbb{N}, +)$ *– decidível.*
  É dada uma expressão lógica fechada (sem variáveis livres) do seguinte tipo: $Q_1 x_1 \ \cdots Q_k x_k \ E = F$ em que E e F são expressões aritméticas que podem envolver variáveis, constantes parêntesis e o operador "+". Um exemplo: $\exists x \ 4 + x = 5$. Pergunta-se: a expressão lógica é verdadeira?

- *Equivalência de duas expressões em* $(\mathbb{N}, +, \times)$ *– indecidível.*
  Já referimos um caso particular, o das equações diofantinas (pág. 27).

Pequena bibliografia

[1] Referência suficiente para esta disciplina
I. C. C. Phillips, *Recursion theory*, Handbook of Logic in Computer Science (S. Abramsky D. M. Gabbay and T.S.E. Maibaum, eds.), vol. 2, Oxford University Press, 1992, pp. 79–187. páginas 109–139, especialmente 122-136.

[2] Referência complementar
Bernard M. Moret, *The Theory of Computation*, Addison-Wesley, 1997.
Nota. Inclui também a teoria das classes de complexidade.

[3] Referência avançada
Piergiorgio Odifreddi, *Classical Recursion Theory*, North-Holland, 1999.