CAPÍTULO II

PROBLEMAS INDECIDÍVEIS

1. Introdução

Em 1900, no Segundo Congresso Internacional de Matemática realizado em Paris, David Hilbert propôs uma lista de problemas de envergadura em Matemática, cuja solução "desafiaria futuras gerações de matemáticos". Vários problemas desta famosa lista foram desde então resolvidos. Alguns, porém, resistiram até hoje a todas as tentativas de solução, e permanecem ainda em aberto.

O décimo problema da lista de Hilbert era de enunciado bastante simples: descreva um algoritmo que determina se uma dada equação diofantina, a diversas variáveis, isto é, uma equação do tipo $P(u_1, u_2, \ldots, u_n) = 0$, onde P é um polinômio com coeficientes inteiros, tem uma solução no anel dos inteiros. Em linguagem ligada à computação: escreva um programa que tem por entrada uma equação diofantina, e que dá por saída *sim* ou *não*, conforme a equação dada tenha ou não soluções inteiras. O programa não precisa encontrar a solução da equação. Precisa apenas determinar se tal solução existe. Por exemplo, se a entrada for $x^2 - y^2 + xy - 11 = 0$, então o programa, após um número finito de passos, deve responder *sim*, pois esta equação tem solução inteira ($x = 3$, $y = 2$ é uma solução). Já se a entrada ao programa fosse $x^2 + y^2 + xy - 11 = 0$, então a resposta deveria ser *não*, pois esta equação não possui soluções inteiras.

A simplicidade do enunciado do décimo problema de Hilbert é ilusória, pois apesar do intenso esforço para resolvê-lo, não foi senão 70 anos depois que a solução foi finalmente encontrada, por Matijasevič, um matemático russo de apenas 22 anos na época. A solução de Matijasevič é bastante complexa, dependendo tanto de resultados gerais de Teoria dos Números, conhecidos há centenas de anos, como do trabalho anterior, específico sobre o problema de Hilbert, de três americanos, Martin Davis, Julia Robinson e Hilary Putnam, que por sua vez baseia-se em certos resultados fundamentais sobre lógica e algoritmos descobertos na década de 30 deste século por Kurt Gödel, Alan Turing, Emil Post, Alonzo Church e Stephen Kleene.

Embora os pormenores da solução sejam complexos, o resultado de todo este trabalho pode ser descrito com mais facilidade ainda do que o próprio problema: tal algoritmo não existe. O décimo problema de Hilbert é, portanto, um exemplo de um problema indecidível por meio de um algoritmo. Tais problemas de tipo *sim* ou *não* são denominados problemas *recursivamente* (isto é, por meio de algoritmos) *indecidíveis*, ou simplesmente *problemas indecidíveis*.

Neste capítulo exibiremos alguns problemas naturais de computação que são indecidíveis. Na verdade, as técnicas que exporemos podem ser usadas para demonstrar limitações inerentes a qualquer sistema formal mecanizável. E uma vez demonstrado que um problema é indecidível, pode-se freqüentemente demonstrar que outros problemas também são indecidíveis, usando uma técnica de redução de um problema ao outro que estudaremos na Seção 5 deste capítulo. Foi por uma redução deste tipo, embora muito mais complexa do que os exemplos que veremos, que se provou que o décimo problema de Hilbert é indecidível. No fim do capítulo daremos uma lista de outros problemas indecidíveis em lógica, álgebra e computação.

2. Um problema indecidível de computação

Na Parte A deste livro foram apresentados os conceitos de procedimento efetivo, ou simplesmente procedimento, e de algoritmo. Um algoritmo, como vimos, é um procedimento efetivo que pára após um número finito de passos, para todos os valores possíveis de seus argumentos. Vimos, por exemplo, uma demonstração de que o Algoritmo de Euclides sempre pára, isto é, que este procedimento é de fato um algoritmo.

Obviamente, a questão de determinar se um dado procedimento é ou não um algoritmo é importante. Em geral, porém, esta questão é difícil de ser respondida como indica o seguinte exemplo, apresentado na linguagem *LP* definida na Parte A do livro, e que é a solução do Exercício A.I.3.

procedimento *Fermat*( ):
  início
    $x, y, z, n \leftarrow 1, 1, 1, 3$;
    repita $x, y, z, n \leftarrow$ *sucessor* (x, y, z, n) **até que** $x^n + y^n = z^n$;
    devolva 1
  fim

```
procedimento sucessor(x, y, z, n) : {seleciona a próxima quádrupla}
  início
    se n > 3 então devolva x, y, z + 1, n − 1
             senão {n = 3}
    se z > 1 então devolva x, y + 1, 1, z + 1
             senão {z = 1, n = 3}
    se y > 1 então devolva x + 1, 1, 1, y + 1
             senão {y = 1, z = 1, n = 3}
    devolva 1, 1, 1, x + 3
  fim
```

Não é difícil verificar que este procedimento pára se e somente se existirem inteiros positivos x, y, z e um inteiro $n \geq 3$ tais que $x^n + y^n = z^n$. (Verifique. Sugestão: verifique primeiro que sucessor enumera todas as quádruplas (x, y, z, n) válidas.) Não se sabe se tais números existem, embora Fermat tivesse anotado na margem de seu exemplar do livro de Diofanto que tinha achado "uma prova maravilhosa" de que tais números não existem, mas a prova infelizmente era demasiadamente longa para caber no reduzido espaço da margem de seu livro. A prova de que Fermat falava nunca foi encontrada, e se de fato era uma prova correta, deve ter sido realmente maravilhosa, pois em 300 anos ninguém conseguiu demonstrar esta proposição, geralmente conhecida como o Último "Teorema" de Fermat. De qualquer modo, deve ser difícil decidir se o procedimento acima é ou não um algoritmo, pois decidir esta questão, como vimos, é equivalente a resolver um problema de Matemática, em aberto há três centenas de anos!

Seria ótimo se pudessemos desenvolver um algoritmo que decidisse se um procedimento arbitrário dado é ou não um algoritmo. Se tivéssemos um tal algoritmo poderíamos, pelo menos em princípio, programar um computador que após um tempo finito nos daria a resposta se o procedimento *Fermat* acima é um algoritmo e, portanto, se o Último "Teorema" de Fermat é ou não verdadeiro. Na Seção 5, no entanto, mostraremos que esse problema também é indecidível, e portanto tal algoritmo não existe.

3. Conceitos básicos

Para poder provar que um problema é indecidível precisaremos formalizar algumas noções.

Um *predicado* θ sobre um conjunto X é uma função $\theta: X \to \{0, 1\}$. Se $\theta(x) = 1$ para $x \in X$, então dizemos que x *satisfaz* θ, ou que θ é *verdadeiro* em x. Caso contrário, isto é, se $\theta(x) = 0$, então dizemos que x *não satisfaz* θ, ou que θ é *falso* em x.

Um predicado θ sobre X é *decidível* se existir um algoritmo F que calcula θ, isto é, se F com argumento $x \in X$ pára, e $F(x) = \theta(x)$. Neste caso dizemos que F é um *processo efetivo de decisão* para θ. O predicado θ é *indecidível* se não for decidível.

Certamente, qualquer coleção de perguntas que admitem por resposta *sim* ou *não*, isto é, qualquer problema *sim* ou *não*, pode ser representado por meio de um predicado. Um membro desta coleção é uma *instância* do problema correspondente. Um problema é *decidível* se o predicado que o representa for decidível.

Um programa na linguagem *LP* é uma palavra sobre um alfabeto Γ conveniente, que inclui todos os símbolos necessários para representar programas em *LP*. O alfabeto Γ portanto inclui, entre outros, as letras maiúsculas e minúsculas A, a, B, b, ..., Z, z, numerais 0, 1, ..., 9, sinais de pontuação „ :, ;„ sinais que denotam operações e símbolos $+$, $-$, $\times$, $\div$, $>$, $=$, $\leftarrow$, **procedimento**, **início**, **fim**, **repita**, **até que**, **faça**, **devolva**, etc. Naturalmente nem todas as seqüências de símbolos de Γ, isto é, nem todas as palavras de Γ^*, descrevem programas válidos. Existe porém um algoritmo que decide se uma palavra de Γ^* representa ou não um programa válido em *LP*. (Vimos parte deste algoritmo descrito em *LP* no Exemplo A.I.7.)

Cada programa em *LP* tem um *nome* que o identifica. Como sabemos, este nome é a cadeia de caracteres que aparece logo após o símbolo **procedimento**.

Um programa, quando executado, pode aceitar dados de entrada de vários tipos, como números inteiros, matrizes, textos, etc. Cada um destes objetos pode ser descrito por uma palavra sobre Γ. Assim, podemos supor que os dados d para um programa são codificados numa única palavra de Γ^*, que chamaremos de *texto de dados* do programa e que denotaremos por d_t. Deve-se especificar precisamente, o que por motivos didáticos não foi feito no Capítulo A.I, o que acontece se o texto de dados for inadequado, por conter um número incorreto de argumentos, ou argumentos de tipo errado, ou argumentos codificados incorretamente. Os pormenores destas especificações não nos interessam tampouco, pois qualquer convenção razoável é adequada para os nossos propósitos. O que suporemos simplesmente é que esteja precisamente especificada a seqüência de ações de um pro-

grama qualquer quando executado com qualquer texto de dados. Por exemplo se o programa *sucessor* descrito na Seção 2 for executado como um texto de dados que codifica cinco inteiros em vez dos quatro necessários, então o último pode ser ignorado; se o texto de dados não codifica pelo menos quatro inteiros então a execução do programa pode ser interrompida imediatamente tendo por resultado um símbolo especial de Γ denotando dados incorretos, e assim por diante.

4. O resultado principal

Estamos agora em condições de enunciar o principal teorema deste capítulo.

TEOREMA 1. *Seja θ o seguinte predicado sobre o conjunto $\Gamma^* \times \Gamma^*$ de pares de palavras de Γ^*:*

(i) $\theta(P_t, d_t) = 1$ *se P_t for um programa em LP que, executado com texto de dados d_t, pára.*

(ii) $\theta(P_t, d_t) = 0$ *em caso contrário.*

Nestas condições não existe um programa que calcula o predicado θ. Em outras palavras, θ é indecidível[1].

É importante entender que o teorema não afirma que dado um particular programa e seu texto de dados de entrada, seja impossível determinar se o programa pára ou não com estes dados. Na Parte A já vimos uma prova de que o Algoritmo de Euclides sempre pára para quaisquer entradas m e n de inteiros positivos, e portanto, às vezes é perfeitamente possível determinar o valor de $\theta(P_t, d_t)$. O que o teorema afirma é que não existe um algoritmo que decida esta questão para qualquer programa P_t e texto de dados de entrada d_t.

Podemos agora passar à prova do Teorema 1.

Demonstração. Por contradição, suponhamos que existe um programa, de nome *Decide*, que calcula o predicado θ. Considere então o programa seguinte:

---

(1) O problema de decidir se um dado procedimento P pára com dados de entrada d é às vezes chamado de "o problema da parada". Uma outra forma de enunciar este teorema é, portanto, dizer que o problema da parada é indecidível. Note que o teorema afirma que não existe um programa em *LP* com as características desejadas. Pela Tese de Church isto "significa" que não existe um algoritmo para decidir o problema da parada.

procedimento *Confunde* (A, B):
se *Decide* $(A, B) = 1$ **então repita** $x \leftarrow x$ **até que** $1 = 0$
{$A(B)$ pára – *Confunde* não pára}
senão devolva 1
{$A(B)$ não pára – *Confunde* pára}

Analisemos a execução de *Confunde* com dados de entrada P_t e d_t, supondo que P_t é um programa de nome P. Claramente, se *Decide* $(P_t, d_t) = 1$, isto é, pela condição (i), se P pára com dados d, então *Confunde* (P_t, d_t) nunca pára, pois repetirá incessantemente a execução do comando $x \leftarrow x$, uma vez que a condição $1 = 0$ é sempre falsa. Por outro lado, se *Decide* $(P_t, d_t) = 0$, isto é se P não pára com dados d, então *Confunde* (P_t, d_t) pára com resposta 1. Portanto, para um programa P_t, de nome P, com texto de dados d_t:

$$\textit{Confunde}\ (P_t, d_t) = \begin{cases} 1 \text{ se } P \text{ não pára com os dados } d \text{ descritos por } d_t \\ \\ \text{não pára, se } P \text{ pára com estes dados.} \end{cases}$$

Com o auxílio de *Confunde* definimos o seguinte programa:

procedimento *Diagonal* (A): **devolva** *Confunde* (A, A).

Analisemos agora a execução de *Diagonal*, supondo que A seja um programa. *Diagonal* chama *Confunde* (A, A), e portanto *Diagonal* pára com resposta 1 se o programa A não pára com dados de entrada que é o seu próprio texto. Caso contrário *Diagonal* não pára.

Agora estamos em condições de provar a contradição que é acarretada pela hipótese da existência de *Decide*. De fato, qual é o resultado de *Diagonal* $(Diagonal_t)$? Se *Diagonal* tendo seu próprio texto por dados de entrada parar, então pela discussão acima de como *Diagonal* funciona, sabemos que *Diagonal* $(Diagonal_t)$ não pára. Se, por outro lado, *Diagonal* tendo seu próprio texto por dados de entrada não parar, então da mesma discussão sabemos que *Diagonal* $(Diagonal_t)$ pára. Em suma:

Diagonal $(Diagonal_t)$ pára sse *Diagonal* $(Diagonal_t)$ não pára.

Isto é uma contradição e portanto somos forçados a concluir que *Decide* não existe. ■

A prova deste teorema é relativamente simples e curta, mas é pouco intuitiva. Vale a pena analisarmos um pouco a estratégia da demonstração. Queremos demonstrar que nenhum programa com as

características de *Decide* pode existir. Como não sabemos como tal programa poderia ser, agiremos contra um "adversário" que alega (falsamente) estar de posse de um tal programa. Nós então lhe solicitamos o texto descrevendo *Decide*, e de sua posse construímos *Confunde* e *Diagonal*. *Diagonal* opera sobre um programa P_t, de nome P, e consultando *Decide* sobre o que P faz quando apresentado com um texto P_t, faz exatamente o contrário. A contradição surge então se *Diagonal* é apresentado com o seu próprio texto $Diagonal_t$, pois neste caso se *Decide* diz que *Diagonal* pára então *Diagonal* não pára, e se *Decide* diz que *Diagonal* não pára então *Diagonal* pára. Deste modo mostramos ao nosso adversário que *Decide* não pode dar a resposta correta com entrada $(Diagonal_t, Diagonal_t)$[2]. Em outras palavras, *Diagonal* é o procedimento que, tendo seu texto por entrada, faz exatamente o contrário do que *Decide* afirma que *Diagonal* deveria fazer.

5. Redutibilidades

Existe uma grande variedade de problemas indecidíveis. Embora para muitos destes problemas uma prova direta, como a que vimos na seção anterior, seja possível, é muitas vezes mais conveniente utilizar-se uma prova indireta empregando o conceito de redutibilidade. Intuitivamente, um predicado θ_1 é redutível a um predicado θ_2, se tivermos um método efetivo para decidir θ_1 desde que nos seja fornecido o valor de θ_2 sempre que for necessário. Este conceito pode ser formalizado de diversas maneiras, com diferentes graus de generalidade. Daremos abaixo uma destas formalizações.

Um predicado $\theta_i : X_i \to \{0, 1\}$ é *m-redutível* a um predicado $\theta_2 : X_2 \to \{0, 1\}$ sse existir um algoritmo f que para qualquer entrada $x_1 \in X_1$ produz uma saída $x_2 \in X_2$ tal que $\theta_1(x_1) = 1$ sse $\theta_2(x_2) = 1$ (e $\theta_1(x_1) = 0$ sse $\theta_2(x_2) = 0$). Se θ_1 é m-redutível a θ_2 então escrevemos $\theta_1 \leq_m \theta_2$.

Uma das aplicações de m-redutibilidade é dada pelo seguinte teorema:

TEOREMA 2. *Se* θ_1 *for um predicado indecidível e* $\theta_1 \leq_m \theta_2$ *então* θ_2 *é também indecidível.*

---

(2) Este método de demonstração é essencialmente a técnica de *diagonalização* concebida por Cantor em suas investigações sobre a Teoria dos Conjuntos.

Demonstração. Suponha que θ_2 é decidível e $\theta_1 \leq_m \theta_2$ via f. Seja g o algoritmo que calcula θ_2, e seja h o procedimento:

procedimento $h(x)$: **devolva** $g(f(x))$

Este procedimento sempre pára, pois tanto f como g sempre param. Mas por hipótese $\theta_1(x) = 1$ sse $\theta_2(f(x)) = 1$ e como g calcula θ_2, $\theta_2(f(x)) = 1$ sse $g(f(x)) = 1$. Portanto h é um algoritmo que calcula θ_1 o que é uma contradição, pois θ_1 é indecidível. ■

A seguir aplicaremos este teorema para provar que alguns outros problemas são indecidíveis.

COROLÁRIO 1 (Problema da parada uniforme). *O problema de determinar se um procedimento é ou não um algoritmo é indecidível. Mais precisamente, não existe um programa Pára que tem por entrada um programa P_t, de nome P, e que pare para qualquer entrada P_t, produzindo saída* 1 *se P é um algoritmo (isto é, parar com quaisquer dados de entrada), e* 0 *em caso contrário.*

Demonstração. Mostraremos que o problema da parada é m-redutível ao problema da parada uniforme. Portanto, como o problema da parada é indecidível (Teorema 1), pelo Teorema 2 o problema da parada uniforme também o é.

Queremos, então, construir um algoritmo f que, tendo por entrada um programa P_t e seu texto de dados d_t, dê por saída um programa Q_t, de nome Q, tal que Q com quaisquer dados pára se e somente se P pára em d. A saída de f será então o texto

procedimento $Q(\ )$: $P(d)$
P_t

onde P é o nome de P_t e d são os dados correspondentes a d_t. É óbvio que tal agoritmo f pode ser construído. Pela Tese de Church, portanto, podemos construir um programa para $f(P_t, d_t)$. Note que f manipula o texto P_t dele extraindo P, e d_t extraindo d, e em seguida escreve o texto Q_t acima.

Analisemos agora o funcionamento de Q. O texto P_t é incluído para definir P dentro de Q. A execução de Q consiste simplesmente em chamar P com dados d, e portanto é óbvio que Q pára com quaisquer dados sse P pára com d. ■

COROLÁRIO 2 (Problema da equivalência de programas). *Não existe um algoritmo Eq que decide se dois procedimentos dados P_1 e P_2 são equivalentes; mais precisamente, não existe um programa Eq (P_t, Q_t) tal que Eq pára com quaisquer dados de entrada, e Eq $(P_t, Q_t) = 1$ se os procedimentos P e Q calculam a mesma função e Eq $(P_t, Q_t) = 0$ em caso contrário. Note que P e Q calculam a mesma função se para qualquer entrada ou ambos não param, ou ambos param com a mesma resposta.*

Demonstração.

Eq (P_t, **procedimento** *Diverge* ( ): **repita** $x \leftarrow x$ **até que** $1 = 0$) $= 1$

sse P não pára com nenhuma entrada. Isto mostra que o problema do Exercício 4 é m-redutível ao problema de equivalência de programas, e portanto, pelo mesmo exercício, este problema é indecidível. ■

6. Outros problemas indecidíveis em matemática

Nesta seção daremos alguns exemplos de problemas indecidíveis em vários ramos da Matemática, sem dar a demonstração destes resultados, limitando-nos simplesmente a enunciá-los. No fim do capítulo damos indicações bibliográficas onde as demonstrações podem ser encontradas.

Em geral, as demonstrações destes resultados são por meio de reduções do tipo que vimos na seção anterior, às vezes muito mais complicadas. Já mencionamos um destes problemas na introdução deste capítulo: é o décimo problema de Hilbert. Passemos agora a outros exemplos de problemas indecidíveis.

(a) — O Problema da Correspondência de Post

Dado um alfabeto Σ e um conjunto finito de pares de palavras sobre Σ, $\{(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)\}$, é indecidível se existe uma seqüência finita de índices $i_1, i_2, \ldots, i_n$ tal que $1 \leq i_j \leq k$ e $x_{i_1} x_{i_2} \ldots x_{i_n} = y_{i_1} y_{i_2} \ldots y_{i_n}$.

(b) — Teorias de Primeira Ordem

É indecidível se uma expressão formada com os símbolos 0, 1, $+$, $\cdot$, $=$ conectivos lógicos $\wedge$, $\vee$, $\neg$, variáveis, quantificadores lógicos $\exists$, $\forall$, é um Teorema da Aritmética.

(c) – O Problema das Palavras em Grupos

É indecidível se duas palavras dadas representam o mesmo elemento de um grupo finitamente apresentado.

Estes exemplos mostram a diversidade dos problemas que já foram demonstrados serem indecidíveis. Existem muitos outros problemas em áreas que vão de Topologia a Linguagens Formais, de Biologia Matemática a Álgebra.

Em muitos destes casos, até que o problema foi demonstrado ser indecidível, acreditava-se ser possível construir um algoritmo para decidir o problema. Deste modo uma das "vantagens" de se ter uma demonstração de indecidibilidade, é exatamente em mostrar a futilidade de se continuar a tentar desenvolver um algoritmo para o problema em questão. Note, no entanto, que mesmo que um problema seja indecidível, uma subclasse menos geral do problema pode ser decidível. Pode, portanto, ser interessante identificar uma tal subclasse e desenvolver algoritmos de decisão para a mesma.

EXERCÍCIOS

1. Verifique que a contradição a que chegamos, supondo a existência do algoritmo *Decide* no Teorema 1, desaparece se supusermos apenas que *Decide* é um procedimento, isto é se supusermos que se *Decide* (P_t, d_t) pára, então *Decide* $(P_t, d_t) = 1$ se $P(d)$ pára, e *Decide* $(P_t, d_t) = 0$ se $P(d)$ não pára. *Decide*, porém, pode nunca parar para alguns argumentos.
2. Esboce um procedimento efetivo *Decide* que com dados P_t e d_t pára sempre que P em d parar, e se parar, então dá o resultado 1 se P em d parar, e 0 em caso contrário.
3. Verifique que dado um conjunto X, a relação $\leq_m$ é reflexiva e transitiva sobre o conjunto de predicados sobre X.
4. Prove que o problema de determinar se um procedimento não pára com nenhuma entrada é indecidível.
5. Prove que o problema de determinar se um programa P com dados de entrada d executa um dado comando do programa é indecidível.
6. Prove que o problema de determinar se o valor da função inteira calculada por um programa P é maior do que 5 para alguma entrada é indecidível.

7. Considere a função $f: \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$ de pares ordenados de naturais não-nulos em naturais não-nulos:

$$f(x, y) = \frac{(x+y-1)\,(x+y-2)}{2} + y.$$

(a) Mostre que f é bijetora.

(b) Ache as funções "inversas" $g: \mathbb{N}^+ \to \mathbb{N}^+$ e $h: \mathbb{N}^+ \to \mathbb{N}^+$ tais que $g(f(x, y)) = x$ e $h(f(x, y)) = y$.

(c) Observe que f enumera os pontos de $\mathbb{N}^+ \times \mathbb{N}^+$; o j-ésimo ponto de $\mathbb{N}^+ \times \mathbb{N}^+$ é o único par ordenado (x, y) tal que $f(x, y) = j$, com $x = g(j)$ e $y = h(j)$. Mostre graficamente esta enumeração, colocando nos pontos (x, y) do plano cartesiano o valor de $f(x, y)$. Descreva em termos geométricos a regra pela qual f enumera $\mathbb{N}^+ \times \mathbb{N}^+$.

(d) Generalizando a regra geométrica acima, obtenha uma função t que enumera $\mathbb{N}^{+3}$, isto é, uma função bijetora $t: \mathbb{N}^+ \times \times \mathbb{N}^+ \times \mathbb{N}^+ \to \mathbb{N}^+$.

(e) Sejam

$$t_1(x, y, z) = f(x, f(y, z))$$

e

$$t_2(x, y, z) = f(f(x, y), z)$$

Mostre que as funções t_1, t_2 e a função t, dada em (d), são três bijeções distintas de $\mathbb{N}^{+3}$ em $\mathbb{N}^+$.

(f) Construa uma bijeção de $\mathbb{N}^{+k}$ em $\mathbb{N}^+$, para k fixo.

(g) Construa uma bijeção de $\bigcup_{k=1}^{\infty} \mathbb{N}^{+k}$ em $\mathbb{N}^+$.

8. Um conjunto A se diz *recursivamente enumerável* sse ou $A = \varnothing$ ou existe um algoritmo F que calcula uma função sobrejetora $f: \mathbb{N} \to A$.

(a) Mostre que a união e a intersecção de conjuntos recursivamente enumeráveis é recursivamente enumerável.

(b) Mostre que um subconjunto A de X é recursivamente enumerável sse existe um programa P que com uma entrada $x \in X$ pára sse $x \in A$.

9. Um subconjunto A de X se diz *recursivo* sse existe um algoritmo que calcula sua função característica, isto é, quando o problema "$x \in A$?" é decidível.

(a) Mostre que se A é recursivo, então A é recursivamente enumerável.
(b) Mostre que existe um conjunto recursivamente enumerável que não é recursivo.
(c) Seja A um subconjunto dos números naturais. Mostre que se A é infinito e recursivamente enumerável em ordem não decrescente, isto é, se $A = \{f(i) \mid i \in \mathbb{N}\}$ onde f é computável, e $f(i) \leq f(j)$ se $i \leq j$, então A é recursivo.
(d) Mostre que um subconjunto A de um conjunto X é recursivo sse A e $X \backslash A$ são recursivamente enumeráveis.
(e) Exiba um conjunto que não é recursivamente enumerável.

10. Prove que nem o conjunto T de todos os programas em LP que são algoritmos, nem seu complemento em relação a Γ^* (ou em relação ao conjunto de todos os programas em LP) são recursivamente enumeráveis.

11. Sejam A e B subconjuntos de Γ^*. Dizemos que A é *m-redutível* a B, e escrevemos $A \leq_m B$, se o predicado "$x \in A$" é m-redutível ao predicado "$y \in B$".
(a) Mostre que se $A \leq_m B$ e B é recursivo (recursivamente enumerável) então A é recursivo (recursivamente enumerável).
(b) Seja $K = \{P_t \mid P_t \in \Gamma^*, P_t$ é um programa que com dados P_t pára$\}$. Mostre que K é recursivamente enumerável, mas não é recursivo.
(c) Mostre que um subconjunto A de Γ^* é recursivamente enumerável sse $A \leq_m K$.

12. Sejam A e B subconjuntos de Γ^*. Dizemos que A é *m-equivalente* a B e escrevemos $A \equiv_m B$ sse $A \leq_m B$ e $B \leq_m A$.
(a) Mostre que $\equiv_m$ é uma relação de equivalência sobre 2^{Γ^*} e que a relação $\leq_m$ se mantém entre as classes de equivalência (chamadas *graus de indecidibilidade*).
(b) O que você pode dizer sobre os conjuntos T e K dos Exercícios 10 e 11, quanto à relação $\leq_m$?

13. Quais dos subconjuntos de Γ^* abaixo são (i) recursivos, (ii) recursivamente enumeráveis ou (iii) tem complemento recursivamente enumerável?
(a) $\{P_t \mid P_t$ é um programa que pára num conjunto infinito de entradas$\}$
(b) $\{P_t \mid P$ é equivalente a um programa Q, dado$\}$

(c) $\{P_t \mid P$ calcula a função que é identicamente $0\}$
(d) $\{P_t \mid$ o conjunto de entradas para os quais P pára é um conjunto recursivo$\}$.

Em todos os casos, a entrada dos programas é uma palavra de Γ^*.

14. Mostre que dado um programa P_t que calcula uma função não decrescente, o problema de decidir se a imagem de P é finita é indecidível.

15. Mostre que o problema da parada para uma máquina de Turing que dispõe de apenas k células em sua fita de trabalho é decidível.

16. Dizemos que um algoritmo P é *limitado em tempo* por $f(n)$, se para toda entrada x de comprimento n, $P(x)$ pára em no máximo $f(n)$ passos (veja Capítulo B.I).
    (a) Mostre que o problema "dado o algoritmo P, de tempo limitado por n^3, P é de tempo limitado por n^2?" é indecidível.
    (b) Exiba um algoritmo P que calcula uma função f, tal que f é computável em tempo limitado por n^3 mas que não pode ser calculada por nenhum algoritmo de tempo limitado por n^2.
    (c) Mostre que o problema "dados os algoritmos P_1 e P_2, ambos de tempo limitado por n^2, e uma entrada x, $P_1(x) = P_2(x)$?" é decidível, mas o problema "dados os algoritmos P_1 e P_2, ambos de tempo limitado por n^2, P_1 e P_2 calculam a mesma função?" é indecidível.

NOTAS BIBLIOGRÁFICAS

Existem vários textos excelentes que discutem os conceitos tratados neste capítulo. O artigo original de Turing [119] contém uma análise excepcionalmente lúcida do modelo computacional de Turing a partir da noção intuitiva de procedimento efetivo. A mesma discussão pode ser encontrada no livro de Minsky [83]. Davis [18] é um texto clássico em computabilidade e Rogers [101] é um tratado avançado da Teoria das Funções Recursivas, recomendado para um estudo mais aprofundado. Os livros de Minsky [83], Hopcroft e Ullman [56], Manna [77], Simon [113] e Machtey e Young [71] são bons livros texto em que os conceitos deste capítulo são examinados, e o artigo de Borodin [7] é um "survey" bastante completo da área.

A solução do décimo problema de Hilbert é apresentada em [20] e [21]. O problema da correspondência de Post está no artigo [95] e também nos textos [83], [56]. A indecidibilidade da aritmética foi o

primeiro resultado na área e deve-se a Gödel [38] (veja também o texto de Shoenfield[111]). O problema das palavras em grupos é um resultado de Novikov [89] e também está no texto [111]. [101] e [112] contêm uma exposição clara da teoria dos graus de indecidibilidade. Uma coleção dos artigos pioneiros mais importantes nesta área contendo os artigos de Gödel, Post e Turing mencionados acima, além de outros trabalhos pioneiros, pode ser encontrada em [19]. O Teorema 1 é baseado no trabalho de Turing [119].