Capítulo 4

Decidibilidade

No Capítulo 3 introduzimos a máquina de Turing como um modelo de um computador de propósito geral e definimos a noção de algoritmo em termos de máquinas de Turing por meio da tese de Church–Turing.

Neste capítulo começamos a investigar o poder de algoritmos para resolver problemas. Demonstramos certos problemas que podem ser resolvidos algoritmicamente e outros que não podem. Nosso objetivo é explorar os limites da solubilidade algorítmica. Você está provavelmente familiarizado com solubilidade por algoritmos porque muito da ciência da computação é dedicado a resolver problemas. A insolubilidade de certos problemas pode vir como uma surpresa.

Por que você deveria estudar insolubilidade? Afinal de contas, mostrar que um problema é insolúvel não parece ser de qualquer utilidade se você tem que resolvê-lo. Você precisa estudar esse fenômeno por duas razões. Primeiro, saber quando um problema é algoritmicamente insolúvel *é* útil porque então você se dá conta de que o problema deve ser simplificado ou alterado antes que você possa encontrar uma solução algorítmica. Como qualquer ferramenta, computadores têm capacidades e limitações que têm que ser apreciadas se elas são para serem bem usadas. A segunda razão é cultural. Mesmo se você lida com problemas que claramente são solúveis, uma olhadela no insolúvel pode estimular sua imaginação e ajudá-lo a ganhar uma importante perspectiva sobre computação.

4.1 Linguagens decidíveis

Nesta seção damos alguns exemplos de linguagens que são decidíveis por algoritmos. Por exemplo, apresentamos um algoritmo que testa se uma cadeia é um membro de uma linguagem livre-do-contexto. Esse problema está relacionado ao problema de reconhecer e compilar programas em uma linguagem de programação. Ver algoritmos resolvendo vários problemas concernentes a autômatos é útil, porque mais adiante você encontrará outros problemas concernentes a autômatos que não podem ser resolvidos por algoritmos.

Problemas decidíveis concernentes a linguagens regulares

Começamos com certos problemas computacionais concernentes a autômatos finitos. Damos algoritmos para testar se um autômato finito aceita uma cadeia, se a linguagem de um autômato é vazia, e se dois autômatos finitos são equivalentes.

Por conveniência usamos linguagens para representar vários problemas computacionais porque já estabelecemos terminologia para lidar com linguagens. Por exemplo, o ***problema da aceitação*** para AFD's de testar se um autômato finito específico aceita uma dada cadeia pode ser expresso como uma linguagem A_{AFD}. Essa linguagem contém as codificações de todos os AFD's juntamente com cadeias que os AFD's aceitam. Seja

$$A_{\mathsf{AFD}} = \{\langle B, w\rangle \mid B \text{ é um AFD que aceita a cadeia de entrada } w\}.$$

O problema de testar se um AFD B aceita uma entrada w é o mesmo que o problema de testar se $\langle B, w\rangle$ é um membro da linguagem A_{AFD}. Igualmente, podemos formular outros problemas computacionais em termos de testar pertinência em uma linguagem. Mostrar que uma linguagem é decidível é o mesmo que mostrar que o problema computacional é decidível.

No teorema a seguir mostrarmos que A_{AFD} é decidível. Portanto esse teorema mostra que o problema de testar se um dado autômato finito aceita uma dada cadeia é decidível.

Teorema 4.1 .....
A_{AFD} é uma linguagem decidível.

.....

Idéia da prova. A idéia da prova é muito simples. Precisamos somente de apresentar uma MT M que decide A_{AFD}.

M = "Sobre a entrada $\langle B, w\rangle$, onde B é um AFD e w é uma cadeia:

1. Simule B sobre a entrada w.
2. Se a simulação termina em um estado de aceitação, *aceite*. Se ela termina em um estado de não-aceitação, *rejeite*."

Prova. Mencionamos somente alguns poucos detalhes de implementação desta prova. Para aqueles de vocês familiares com escrever programas em qualquer linguagem de programação padrão, imagine como você escreveria um programa para levar adiante a simulação.

Primeiro, vamos examinar a entrada $\langle B, w\rangle$. Ela é uma representação de um AFD B juntamente com uma cadeia w. Uma representação razoável de B é simplesmente uma lista de seus cinco componentes, Q, Σ, δ, q_0, e F. Quando M recebe sua entrada, M primeiro verifica se ela representa propriamente um AFD B e uma cadeia w. Se não, M rejeita.

Então M leva adiante a simulação de uma maneira direta. Ela mantém registro do estado atual de B e da posição atual de B sobre a entrada w escrevendo essa informação na sua fita. Inicialmente, o estado atual de B é q_0 e a posição atual de B sobre a entrada é o símbolo mais à esquerda de w. Os estados e a posição são atualizados conforme a função de transição especificada δ. Quando M termina de processar o último símbolo de w, M aceita a entrada se B está num estado de aceitação; M rejeita a entrada se B está em um estado de não-aceitação.

.....

Podemos provar um teorema semelhante para autômatos finitos não-determinísticos. Seja

$$A_{\mathsf{AFN}} = \{\langle B, w\rangle \mid B \text{ é um AFN que aceita a cadeia de entrada } w\}.$$

Teorema 4.2 .....
A_{AFN} é uma linguagem decidível.

Prova. Apresentamos uma MT N que decide A_{AFN}. Poderíamos projetar N para operar como M, simulando um AFN ao invés de um AFD. Ao invés disso, faremos diferente para ilustrar uma nova idéia: fazer N usar M como uma subrotina. Devido ao fato de que M é projetado para funcionar com AFD's, N primeiro converte o AFN que ele recebe como entrada para um AFD antes de passá-lo para M.

$N =$ "Sobre a entrada $\langle B, w\rangle$ onde B é um AFN, e w é uma cadeia:

1. Converta o AFN B para um AFD C equivalente usando o procedimento para conversão dado no Teorema 1.19.
2. Rode a MT M do Teorema 4.1 sobre a entrada $\langle C, w\rangle$.
3. Se M aceita, *aceite*; caso contrário *rejeite*."

Rodar a MT M no estágio 2 significar incorporar M no projeto de N como um subprocedimento.

.....

Igualmente, podemos testar se uma expressão regular gera uma dada cadeia. Seja $A_{\mathsf{EXR}} = \{\langle R, w\rangle \mid R \text{ é uma expressão regular que gera a cadeia } w\}$.

Teorema 4.3 .....
A_{EXR} é uma linguagem decidível.

Prova. A seguinte MT P decide A_{EXR}.

$P =$ "Sobre a entrada $\langle R, w\rangle$ onde R é uma expressão regular e w é uma cadeia:

1. Converta a expressão regular R para um AFD equivalente A usando o procedimento para essa conversão dado no Teorema 1.28.
2. Rode a MT M sobre a entrada $\langle A, w\rangle$.
3. Se M aceita, *aceite*; se M rejeita, *rejeite*."

.....

Os Teoremas 4.1, 4.2, e 4.3 ilustram que para propósitos de decidibilidade, apresentar uma máquina de Turing com um AFD, um AFN, ou uma expressão regular são todos equivalentes porque a máquina é capaz de converter uma forma de codificação na outra.

Agora nos voltamos para um tipo diferente de problema concernente a autômatos finitos: ***testar vacuidade*** para a linguagem de um autômato finito. Nos teoremas precedentes tínhamos que testar se um autômato finito aceita uma cadeia específica. Na próxima prova temos que testar se um autômato finito aceita alguma cadeia. Seja

$$V_{\mathsf{AFD}} = \{\langle A\rangle \mid A \text{ é um AFD e } L(A) = \emptyset\}.$$

Teorema 4.4 .....
V_{AFD} é uma linguagem decidível.

Prova. Um AFD aceita alguma cadeia se e somente se atingir um estado de aceitação a partir do estado inicial viajando por setas do AFD é possível. Para testar essa condição podemos projetar uma MT T que usa um algoritmo de marcação semelhante àquele usado no Exemplo 3.14.

T = "Sobre a entrada $\langle A \rangle$ onde A é um AFD:

1. Marque o estado inicial de A.
2. Repita até que nenhum estado novo venha a ser marcado:
3. Marque qualquer estado que tem uma transição chegando nele a partir de qualquer estado que já esteja marcado.
4. Se nenhum estado de aceitação está marcado, *aceite*; caso contrário *rejeite*."

.....

O próximo teorema enuncia que testar se dois AFD's reconhecem a mesma linguagem é decidível. Seja

$$EQ_{\mathsf{AFD}} = \{\langle A, B \rangle \mid A \text{ e } B \text{ são AFD's e } L(A) = L(B)\}.$$

Teorema 4.5 .....
EQ_{AFD} é uma linguagem decidível.

Prova. Para provar esse teorema usamos o Teorema 4.4. Construimos um novo AFD C a partir de A e B, onde C aceita apenas aquelas cadeias que são aceitas por A ou B mas não por ambas. Por conseguinte, se A e B reconhecem a mesma linguagem C não aceitará nada. A linguagem de C é

$$L(C) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right).$$

Essa expressão é às vezes chamada a **diferença simétrica** de $L(A)$ e $L(B)$ e é ilustrada na Figura 4.1. Aqui $\overline{L(A)}$ é o complemento de $L(A)$. A diferença simétrica é util aqui porque $L(C) = \emptyset$ se e somente se $L(A) = L(B)$. Podemos construir C a partir de A e B com as construções para provar que a classe das linguagens regulares é fechada sob complementação, união, e interseção. Essas construções são algoritmos que podem ser levados adiante por máquinas de Turing. Uma vez que construimos C podemos usar o Teorema 4.4 para testar se $L(C)$ é vazia. Se ela é vazia, $L(A)$ e $L(B)$ têm que ser iguais.

F = "Sobre a entrada $\langle A, B \rangle$, onde A e B são AFD's:

1. Construa o AFD C como descrito.
2. Rode a MT T do Teorema 4.4 sobre a entrada $\langle C \rangle$.
3. Se T aceita, *aceite*; caso contrário *rejeite*.

Figura 4.1: A diferença simétrica de $L(A)$ e $L(B)$

Problemas decidíveis concernentes a linguagens livres-do-contexto

Aqui, descrevemos algoritmos para testar se uma GLC gera uma cadeia específica e testar se a linguagem de uma GLC é vazia. Seja

$$A_{\mathsf{GLC}} = \{\langle G, w\rangle \mid G \text{ é uma GLC que gera a cadeia } w\}.$$

Teorema 4.6 . . . . . .
A_{GLC} é uma linguagem decidível.

. . . . . .
Idéia da prova. Para a GLC G e a cadeia w desejamos testar se G gera w. Uma idéia é usar G para passar por todas as derivações para determinar se alguma delas é uma derivação de w. Essa idéia não funciona, pois uma quantidade infinita de derivações pode ter que ser tentada. Se G não gera w, esse algoritmo nunca pararia. Essa idéia dá uma máquina de Turing que um reconhecedor, mas não um decisor, para A_{GLC}.

Para fazer dessa máquina de Turing um decisor precisamos de assegurar que o algoritmo tenta somente uma quantidade finita de derivações. No Problema 2.19 na página 121 mostramos que, se G estivesse na forma normal de Chomsky, qualquer derivação de w tem $2n-1$ passos, onde n é o comprimento de w. Naquele caso verificar somente derivações com $2n - 1$ passos para determinar se G gera w seria suficiente. Somente uma quantidade finita de tais derivações existem. Podemos converter G para a forma normal de Chomsky usando o procedimento dado na Seção 2.1.

Prova. A MT S para A_{GLC} segue.
$S =$ "Sobre a entrada $\langle G, w\rangle$, onde G é uma GLC e w é uma cadeia:

1. Converta G para uma gramática equivalente na forma normal de Chomsky.
2. Liste todas as derivações com $2n - 1$ passos, onde n é o comprimento de w, exceto se $n = 0$, então nesse caso liste todas as derivações com 1 passo.
3. Se qualquer dessas derivações gera w, *aceite*; se não, *rejeite*."

. . . . . .

O problema de testar se uma GLC gera um cadeia específica está relacionado ao problema de compilar linguagens de programação. O algoritmo na MT S é muito ineficiente e nunca seria usado na prática, mas ele é fácil de descrever e não estamos preocupados com eficiência aqui. Na Parte Três deste livro lidamos com questões concernentes ao tempo de execução e o uso de memória de algoritmos. Na prova do Teorema 7.14, descrevemos um algoritmo mais eficiente para reconhecer linguagens livres-do-contexto.

Lembre-se que demos procedimentos para converter de um lado para o outro entre GLC's e AP's no Teorema 2.12. Portanto tudo que dizemos sobre a decidibilidade de problemas concernentes a GLC's aplica-se igualmente a AP's.

Vamos nos voltar agora para o problema de testar vacuidade para a linguagem de uma GLC. Como fizemos para AFD's, podemos mostrar que o problema de testar se uma GLC gera alguma cadeia é decidível. Seja

$$V_{\mathsf{GLC}} = \{\langle G\rangle \mid G \text{ é uma GLC e } L(G) = \emptyset\}.$$

Teorema 4.7
V_{GLC} é uma linguagem decidível.

Idéia da prova. Para encontrar um algoritmo para esse problema podemos tentar usar a MT S do Teorema 4.6. Ele enuncia que podemos testar se uma GLC gera uma cadeia w específica. Para determinar se $L(G) = \emptyset$ o algoritmo poderia tentar passar por todas as possíveis w's, uma por uma. Mas existe uma quantidade infinita de w's para tentar, portanto esse método poderia terminar rodando para sempre. Precisamos de adotar uma abordagem diferente.

De modo a testar se a linguagem de uma gramática é vazia, precisamos testar se a variável inicial pode gerar uma cadeia de terminais. O algoritmo faz isso resolvendo um problema mais geral. Ele determina *para cada variável* se aquela variável é capaz de gerar uma cadeia de terminais. Quando o algoritmo determinou que uma variável pode gerar alguma cadeia de terminais, o algoritmo mantém registro dessa informação colocando uma marca sobre aquela variável.

Primeiro, o algoritmo marca todos os símbolos terminais na gramática. Então, ele varre todas as regras da gramática. Se ele por acaso encontra uma regra que permite alguma variável ser substituída por alguma cadeia de símbolos dos quais todos já estejam marcados, o algoritmo sabe que essa variável pode ser marcada também. O algoritmo continua dessa forma até que ele não possa marcar quaisquer variáveis adicionais. A MT R implementa esse algoritmo.

Prova.
$R =$ "Sobre a entrada $\langle G \rangle$, onde G é uma GLC:

1. Marque todos os símbolos terminais de G.
2. Repita até que nenhuma variável nova venha a ser marcada.
3. Marque qualquer variável A onde G tem a regra $A \to U_1 U_2 \cdots U_k$ e cada símbolo $U_1, \ldots, U_k$ já tenha sido marcado.
4. Se o símbolo inicial não está marcado, *aceite*; caso contrário *rejeite*."

A seguir consideramos o problema de testar se duas gramáticas livres-do-contexto geram a mesma linguagem. Seja

$$EQ_{\mathsf{GLC}} = \{\langle G, H \rangle \mid G \text{ e } H \text{ são GLC's e } L(G) = L(H)\}.$$

O Teorema 4.5 deu um algoritmo que decide a linguagem análoga EQ_{AFD} para autômatos finitos. Usamos o procedimento de decisão para V_{AFD} para provar que EQ_{AFD} é decidível. Devido ao fato de que V_{GLC} também é decidível, você poderia pensar que podemos usar uma estratégia semelhante para provar que EQ_{GLC} é decidível. Mas algo dá errado com essa idéia! A classe de linguagens livres-do-contexto *não* é fechada sob complementação ou interseção como você provou no Exercício 2.2. Na verdade, EQ_{GLC} não é decidível, e você verá a técnica para provar isso no Capítulo 5.

Agora mostramos que toda linguagem livre-do-contexto é decidível por uma máquina de Turing.

Teorema 4.8 .....
Toda linguagem livre-do-contexto é decidível.

.....

Idéia da prova. Seja A uma GLC. Nosso objetivo é mostrar que A é decidível. Uma (má) idéia é converter um AP para A diretamente numa MT. Isso não é difícil de fazer porque simular uma pilha com as fitas mais versáteis de MT's é fácil. O AP para A pode ser não-determinístico, mas isso parece legítimo porque você pode convertê-lo numa MT não-determinística e sabemos que qualquer MT não-determinística pode ser convertida numa MT determinística equivalente. Mesmo assim, há uma dificuldade. Alguns ramos da computação do AP pode seguir para sempre, lendo e escrevendo na pilha sem chegar numa parada. A MT simuladora então também teria alguns ramos não-terminantes na sua computação, e portanto a MT não seria um decisor. Uma idéia diferente é necessária. Ao invés, provamos esse teorema com a MT S que projetamos no Teorema 4.6 para decidir A_{GLC}.

Prova. Seja G uma GLC para A e projete uma MT M_G que decide A. Construimos uma cópia de G dentro de M_G. Ela funciona da seguinte maneira:

$M =$ "Sobre a entrada w:

1. Rode a MT S sobre a entrada $\langle G, w\rangle$.
2. Se essa máquina aceita, *aceite*; se ela rejeita, *rejeite*."

.....

O Teorema 4.8 provê a ligação final no relacionamento entre as quatro principais classes de linguagens que descrevemos até agora neste curso: regulares, livres-do-contexto, decidíveis, e Turing-reconhecíveis. A Figura 4.2 ilustra esse relacionamento.

Figura 4.2: O relacionamento entre classes de linguagens

4.2 O problema da parada .....

Nesta seção provamos um dos teoremas mais filosificamente importantes da teoria da computação: existe um problema específico que é algoritmicamente insolúvel. Computadores parecem ser tão poderosos que você pode acreditar que todos os problemas irão eventualmente se render a eles. O teorema apresentado aqui demonstra que computadores são limitados de uma maneira muito fundamental.

Que tipo de problemas são insolúveis por computador? Eles são esotéricos, residindo apenas nas mentes dos teóricos? Não! Até alguns problemas comuns que as pessoas desejam resolver acontecem de ser computacionalmente insolúveis.

Em um tipo de problemas insolúveis, você é apresentado a um programa de computador e uma especificação precisa do que aquele programa supostamente faz (e.g., ordenar uma lista de números). Devido ao fato de que ambos programa e especificação são objetos matematicamente precisos, você espera automatizar o processo de verificação alimentando esses objetos num computador programado apropriadamente. Entretanto,

você vai se desapontar. O problema geral de verificação de software não é solúvel por computador.

Nesta seção e no Capítulo 5 você vai encontrar problemas computacionalmente insolúveis. Nossos objetivos são ajudar você a desenvolver um sentimento para os tipos de problemas que são insolúveis e aprender técnicas para provar insolubilidade.

Agora nos voltamos para nosso primeiro teorema que estabelece a indecidibilidade de uma linguagem específica: o problema de testar se uma máquina de Turing aceita uma dada cadeia de entrada. Chamamo-lo A_{MT} por analogia com A_{AFD} e A_{GLC}. Mas, enquanto que A_{AFD} e A_{GLC} eram decidíveis, A_{MT} não o é. Seja

$$A_{\mathsf{MT}} = \{\langle M, w\rangle \mid M \text{ é uma } \mathsf{MT} \text{ e } M \text{ aceita } w\}.$$

Teorema 4.9 .....
A_{MT} é indecidível.

Antes de chegar na prova, vamos primeiro observar que A_{MT} é Turing-reconhecível. Por conseguinte o Teorema 4.9 mostra que reconhecedores *são* mais poderosos que decisores. Exigir de uma MT que páre sobre todas as entradas restringe os tipos de linguagens que ela pode reconhecer. A seguinte máquina de Turing U reconhece A_{MT}.

U = "Sobre a entrada $\langle M, w\rangle$, onde M é uma MT e w é uma cadeia:

1. Simule M sobre a entrada w.
2. Se M alguma vez entra no seu estado de aceitação, *aceite*; se M alguma vez entra no seu estado de rejeição, *rejeite*."

Note que essa máquina entra em loop sobre a entrada $\langle M, w\rangle$ se M entra em loop sobre w, e é por isso que essa máquina não decide A_{MT}. Se o algoritmo tivesse alguma maneira de determinar que M não estava parando sobre w, ele poderia *rejeitar*. Portanto A_{MT} é às vezes chamado o ***problema da parada***. Como demonstramos, um algoritmo não tem forma de fazer essa determinação.

A máquina de Turing U é interessante em si própria. Ela é um exemplo da *máquina de Turing universal* originalmente proposta por Turing. Essa máquina é chamada universal porque ela é capaz de simular qualquer outra máquina de Turing a partir da descrição daquela máquina. A máquina de Turing universal desempenhou um importante papel inicial no estímulo ao desenvolvimento de computadores de programa-armazenado.

O método da diagonalização

A prova da indecidibilidade do problema da parada usa uma técnica chamada ***diagonalização***, descoberta pelo matemático Georg Cantor em 1873. Cantor estava preocupado com o problema de medir os tamanhos de conjuntos infinitos. Se temos dois conjuntos infinitos, como dizer se um é maior que o outro ou se eles são de mesmo tamanho? Para conjuntos finitos, é claro, responder a essas perguntas é fácil. Simplesmente contamos os elementos em um conjunto finito, e o número resultante é o seu tamanho. Mas, se tentarmos contar os elementos de um conjunto infinito, nunca terminaremos! Portanto não podemos usar o método de contagem para determinar os tamanhos relativos de conjuntos infinitos.

Por exemplo, tome o conjunto de inteiros pares e o conjunto de todos as cadeias sobre $\{\mathtt{0}, \mathtt{1}\}$. Ambos os conjuntos são infinitos e por conseguinte maiores que qualquer

conjunto finito, mas um dos dois é maior que o outro? Como podemos comparar seu tamanho relativo?

Cantor propôs uma solução um tanto bela para esse problema. Ele observou que dois conjuntos finitos têm o mesmo tamanho se os elementos de um conjunto podem ser emparelhados com os elementos do outro conjunto. Esse método compara os tamanhos sem recorrer à contagem. Podemos estender essa idéia para conjuntos infinitos. Vamos ver o que ele significa mais precisamente.

Definição 4.10 . . . . . .

Assuma que temos dois conjuntos A e B e uma função f de A para B. Digamos que f é ***um-para-um*** se ela nunca mapeia dois elementos diferentes ao mesmo lugar, isto é, se $f(a) \neq f(b)$ sempre que $a \neq b$. Digamos que f é ***sobrejetora*** se ela chegar em todo elemento de B, ou seja, se para todo $b \in B$ existe um $a \in A$ tal que $f(a) = b$. Digamos que A e B têm o ***mesmo tamanho*** se existe um função um-para-um, sobrejetora $f : A \longrightarrow B$. Uma função que é tanto um-para-um quanto sobrejetora é chamada uma ***correspondência***. Em uma correspondência todo elemento de A mapeia para um único elemento de B e cada elemento de B tem um único elemento de A mapeando para ele. Uma correspondência é simplesmente uma maneira de emparelhar os elementos de A com os elementos de B.

Exemplo 4.11 . . . . . .

Seja $\mathcal{N}$ o conjunto de números naturais $\{1, 2, 3, \ldots\}$ e seja $\mathcal{E}$ o conjunto dos números naturais pares $\{2, 4, 6, \ldots\}$. Usando a definição de Cantor de tamanho podemos ver que $\mathcal{N}$ e $\mathcal{E}$ têm o mesmo tamanho. A correspondência f mapeando $\mathcal{N}$ para $\mathcal{E}$ é simplesmente $f(n) = 2n$. Podemos visualizar f mais facilmente com a ajuda de uma tabela.

n	$f(n)$
1	2
2	4
3	6
$\vdots$	$\vdots$

É claro que esse exemplo parece bizarro. Intuitivamente, $\mathcal{E}$ é menor que $\mathcal{N}$ porque $\mathcal{E}$ é um subconjunto próprio de $\mathcal{N}$. Mas emparelhar cada membro de $\mathcal{N}$ com seu próprio membro de $\mathcal{E}$ é possível, portanto declaramos esses dois conjuntos como sendo de mesmo tamanho.

Definição 4.12 . . . . . .

Um conjunto A é ***contável*** se ele é finito ou tem o mesmo tamanho que $\mathcal{N}$.

Exemplo 4.13 . . . . . .

Agora nos voltamos para um exemplo ainda mais estranho. Se fizermos $\mathcal{Q}$ ser o conjunto dos números racionais positivos, ou seja, $\mathcal{Q} = \{\frac{m}{n} \mid m, n \in \mathcal{N}\}$, $\mathcal{Q}$ parece ser muito maior que $\mathcal{N}$. Mesmo assim esses dois conjuntos são do mesmo tamanho. Demonstramos essa conclusão dando uma correspondência com $\mathcal{N}$ para mostrar que $\mathcal{Q}$ é contável. Uma maneira fácil de dar uma correspondência com $\mathcal{N}$ é listar todos os elementos de $\mathcal{Q}$. Então emparelhamos o primeiro elemento na lista com o número 1 de

$\mathcal{N}$, o segundo elemento na lista com o número 2 de $\mathcal{N}$, e assim por diante. Temos que verificar para ter certeza de que todo membro de $\mathcal{Q}$ aparece apenas uma vez na lista.

Para obter essa lista fazemos uma matriz infinita contendo todos os membros dos racionais positivos, como mostrado na Figura 4.3. A i-ésima linha contém todos os números com numerador i e a j-ésima coluna tem todos os números com denominador j. Portanto o número $\frac{i}{j}$ ocorre na i-ésima linha e j-ésima coluna.

Agora transformamos essa matriz numa lista. Uma (má) forma de tentá-lo seria começar a lista com todos os elementos na primeira linha. Essa não é uma boa abordagem porque a primeira linha é infinita, portanto a lista nunca chegaria na segunda linha. Ao invés disso, listamos os elementos nas diagonais, começando do canto, que estão superimpostas no diagrama. A primeira diagonal contém um único elemento $\frac{1}{1}$, e a segunda diagonal contém os dois elementos $\frac{2}{1}$ e $\frac{1}{2}$. Portanto os primeiros três elementos na lista são $\frac{1}{1}$, $\frac{2}{1}$, e $\frac{1}{2}$. Na terceira diagonal uma complicação aparece. Ela conteém $\frac{3}{1}$, $\frac{2}{2}$, e $\frac{1}{3}$. Se simplesmente adicionarmos esses à lista, repeteríamos $\frac{1}{1} = \frac{2}{2}$. Evitamos fazer isso pulando um elemento quando ele causaria uma repetição. Portanto adicionamos somente os dois novos elementos $\frac{3}{1}$ e $\frac{1}{3}$. Continuando dessa maneira obtemos uma lista de todos os elementos de $\mathcal{Q}$.

Figura 4.3: Uma correspondência de $\mathcal{N}$ e $\mathcal{Q}$

Após ver a correspondência de $\mathcal{N}$ e $\mathcal{Q}$, você poderia pensar que quaisquer dois conjuntos infinitos podem ser mostrados ter o mesmo tamanho. Afinal de contas, você precisa somente demonstrar uma correspondência, e esse exemplo mostra que correspondências surpreendentes realmente existem. Entretanto, para alguns conjuntos infinitos nenhuma correspondência com $\mathcal{N}$ existe. Esses conjuntos são simplesmente grandes demais. Tais conjuntos são chamados ***incontáveis***.

O conjunto dos números reais é um exemplo de um conjunto incontável. Um ***número real*** é aquele que tem uma representação decimal. Os números $\pi = 3,14159265...$ e $\sqrt{2} = 1,4142135...$ são exemplos de números reais. Seja $\mathcal{R}$ o conjunto de números reais. Cantor provou que $\mathcal{R}$ é incontável. Fazendo isso ele introduziu o método da diagonalização.

Teorema 4.14 .....
$\mathcal{R}$ é incontável.

Prova. Para mostrar que $\mathcal{R}$ é incontável, mostramos que nenhuma correspondência existe entre $\mathcal{N}$ e $\mathcal{R}$. A prova é por contradição. Suponha que uma correspondência f existisse entre $\mathcal{N}$ e $\mathcal{R}$. Nossa tarefa é mostrar que f falha em funcionar como deveria. Para ela ser uma correspondência, f tem que emparelhar todos os membros de $\mathcal{N}$ com todos os membros de $\mathcal{R}$. Mas encontraremos um x em $\mathcal{R}$ que não está emparelhado com nada em $\mathcal{N}$, o que será nossa contradição.

A maneira pela qual encontramos esse x é verdadeiramente construindo-o. Escolhemos cada dígito de x para fazer x diferente de um dos números reais que está emparelhado com um elemento de $\mathcal{N}$. No final temos certeza de que x é diferente de qualquer número real que está emparelhado.

Podemos ilustrar essa idéia dando um exemplo. Suponha que a correspondência f existe. Faça $f(1) = 3,14159...$, $f(2) = 55,55555...$, $f(3) = ...$, e assim por diante,

só para mencionar alguns valores de f. Então f emparelha o número 1 com $3,14159...$, o número 2 com $55,55555...$, e assim por diante. A seguinte tabela mostra uns poucos valores de uma correspondência hipotética f entre $\mathcal{N}$ e $\mathcal{R}$.

n	$f(n)$
1	$3,14159...$
2	$55,55555...$
3	$0,12345...$
4	$0,50000...$
$\vdots$	$\vdots$

Construimos o x desejado dando sua representação decimal. É um número entre 0 e 1, de modo que todos os dígitos significativos são dígitos fracionários seguintes à vírgula decimal. Nosso objetivo é assegurar que $x \neq f(n)$ para qualquer n. Para assegurar que $x \neq f(1)$ fazemos com que o primeiro dígito de x seja qualquer coisa diferente do primeiro dígito fracionário 1 de $f(1) = 3,14159...$. Arbitrariamente , fazemos com que ele seja 4. Para assegurar que $x \neq f(2)$ fazemos o com que o segundo dígito de x seja qualquer coisa diferente do segundo dígito fracionário 5 de $f(2) = 55,55555...$. Arbitrariamente, fazemos com que ele seja 6. O terceiro dígito fracionário de $f(3) = 0,12345...$ é 3, portanto fazemos com que x seja qualquer coisa diferente, digamos, 4. Continuando dessa maneira descendo na diagonal da tabela para f, obtemos todos os dígitos de x, como mostrado na tabela abaixo. Sabemos que x não é $f(n)$ para qualquer que seja n porque ele difere de $f(n)$ no n-ésimo dígito fracionário. (Um pequeno problema surge porque certos números, tais como $0,1999...$ e $0,2000...$, são iguais muito embora suas representações decimais sejam diferentes. Evitamos esse problema nunca selecionando os dígitos 0 ou 9 quando construimos x.)

n	$f(n)$
1	$3,\underline{1}4159...$
2	$55,5\underline{5}555...$
3	$0,12\underline{3}45...$
4	$0,500\underline{0}0...$
$\vdots$	$\vdots$

$x = 0,4641...$

. . . . . .

O teorema precedente tem uma importante aplicação à teoria da computação. Ele mostra que algumas linguagens não são decidíveis ou mesmo Turing-reconhecíveis, pela razão que existe uma quantidade incontável de linguagens e mesmo assim somente uma quantidade contável de máquinas de Turing. Devido ao fato de que cada máquina de Turing pode reconhecer uma única linguagem e que existem mais linguagens que máquinas de Turing, algumas linguagens não são reconhecíveis por nenhuma máquina de Turing. Tais linguagens não são Turing-reconhecíveis, como enunciamos no corolário a seguir.

Corolário 4.15 . . . . . .
Algumas linguagens não são Turing-reconhecíveis.

Prova. Para mostrar que o conjunto de todas as máquinas de Turing é contável primeiro observamos que o conjunto de todas as cadeias Σ^* é contável, para qualquer alfabeto Σ. Com apenas uma quantidade finita de cadeias de cada comprimento, podemos

formar uma lista de Σ^* relacionando todas as cadeias de comprimento 0, comprimento 1, comprimento 2, e assim por diante.

O conjunto de todas as máquinas de Turing é contável porque cada máquina de Turing M tem uma codificação numa cadeia $\langle M \rangle$. Se simplesmente omitirmos aquelas cadeias que não são codificações legítimas de máquinas de Turing, podemos obter uma lista de todas as máquinas de Turing.

Para mostrar que o conjunto de todas as linguagens é incontável primeiro observamos que o conjunto de todas as seqüências binárias infinitas é incontável. Uma *seqüência binária infinita* é uma seqüência sem-fim de `0`'s e `1`'s. Suponha que $\mathcal{B}$ seja o conjunto de todas as seqüências binárias infinitas. Podemos mostrar que $\mathcal{B}$ é incontável usando uma prova por diagonalização semelhante àquela usada na Teorema 4.14 para mostrar que $\mathcal{R}$ é incontável.

Seja $\mathcal{L}$ o conjunto de todas as linguagens sobre o alfabeto Σ. Mostramos que $\mathcal{L}$ é incontável dando uma correspondência com $\mathcal{B}$, portanto mostrando que os dois conjuntos são do mesmo tamanho. Seja $\Sigma^* = \{s_1, s_2, s_3, \ldots\}$. Cada linguagem $A \in \mathcal{L}$ tem uma única seqüência em $\mathcal{B}$. O i-ésimo bit daquela seqüência é um `1` se $s_i \in A$ e é um `0` se $s_i \notin A$, o que é chamado a ***seqüência característica*** de A. Por exemplo, se A fosse a linguagem de todas as cadeias começando com um `0` sobre o alfabeto $\{\mathtt{0}, \mathtt{1}\}$, sua seqüência característica χ_A seria

$$\begin{array}{rccccccccccccccccccc}
\Sigma^* = \{ & \varepsilon & , & \mathtt{0} & , & \mathtt{1} & , & \mathtt{00} & , & \mathtt{01} & , & \mathtt{10} & , & \mathtt{11} & , & \mathtt{000} & , & \mathtt{001} & , & \cdots & \} \; ; \\
A = \{ & & , & \mathtt{0} & , & & , & \mathtt{00} & , & \mathtt{01} & , & & , & & , & \mathtt{000} & , & \mathtt{001} & , & \cdots & \} \; ; \\
\chi_A = & 0 & & 1 & & 0 & & 1 & & 1 & & 0 & & 0 & & 1 & & 1 & & \cdots & .
\end{array}$$

A função $f : \mathcal{L} \to \mathcal{B}$, onde $f(A)$ é igual à seqüência característica de A, é injetora e sobrejetora e portanto uma correspondência. Por conseguinte, como $\mathcal{B}$ é incontável, $\mathcal{L}$ também é incontável.

Portanto mostramos que o conjunto de todas as linguagens não pode ser posto numa correspondência com o conjunto de todas as máquinas de Turing. Concluimos que algumas linguagens não são reconhecidas por nenhuma máquina de Turing.

. . . . . .

O problema da parada é indecidível

Agora estamos prontos para provar o Teorema 4.9, a indecidibilidade da linguagem

$$A_{\mathsf{MT}} = \{\langle M, w \rangle \mid M \text{ é uma } \mathsf{MT} \text{ e } M \text{ aceita } w\}.$$

Prova. Assumimos que A_{MT} é decidível e obtemos uma contradição. Suponha que H é um decisor para A_{MT}. Sobre a entrada $\langle M, w \rangle$, onde M é uma MT e w é uma cadeia, H pára e aceita se M aceita w. Além disso, H pára e rejeita se M falha em aceita w. Em outras palavras, assumimos que H é uma MT, onde

$$H(\langle M, w \rangle) = \begin{cases} \textit{aceita} & \text{se } M \text{ aceita } w \\ \textit{rejeita} & \text{se } M \text{ não aceita } w. \end{cases}$$

Agora construimos uma nova máquina de Turing D com H como uma subrotina. Essa nova MT chama H para determinar o que M faz quando a entrada para M é sua própria descrição $\langle M \rangle$. Uma vez que D tenha determinado essa informação, ela faz o oposto. Ou seja, ela rejeita se M aceita e aceita se M não aceita. Abaixo está uma descrição de D:

$D =$ "Sobre a entrada $\langle M \rangle$, onde M é uma MT

1. Rode H sobre a entrada $\langle M, \langle M \rangle \rangle$.
2. Dê como saída o oposto do que H dá como saída, ou seja, se H aceita, *rejeite* e se H rejeita, *aceite*."

Não se confunda com a idéia de rodar uma máquina sobre sua própria descrição! Isso é semelhante a rodar um programa com ele próprio como entrada, algo que realmente ocorre eventualmente na prática. Por exemplo, um compilador é um programa que traduz outros programas. Um compilador para a linguagem Pascal pode ele próprio ser escrito em Pascal, de modo que rodar aquele programa sobre si próprio faria sentido. Em resumo,

$$D(\langle M \rangle) = \begin{cases} \textit{aceita} & \text{se } M \text{ não aceita } \langle M \rangle \\ \textit{rejeita} & \text{se } M \text{ aceita } \langle M \rangle. \end{cases}$$

O que acontece quando rodamos D com sua própria descrição $\langle D \rangle$ como entrada? Nesse caso obtemos

$$D(\langle D \rangle) = \begin{cases} \textit{aceita} & \text{se } D \text{ não aceita } \langle D \rangle \\ \textit{rejeita} & \text{se } D \text{ aceita } \langle D \rangle. \end{cases}$$

Independentemente do que D faz, ela é forçada a fazer o oposto, o que é obviamente uma contradição. Por conseguinte nem a MT D nem a MT M podem existir.

. . . . . .

Vamos revisar os passos dessa prova. Assuma que uma MT H decide A_{MT}. Então use H para construir uma MT D que quando recebe a entrada $\langle M \rangle$ aceita exatamente quando M não aceita a entrada $\langle M \rangle$. Finalmente, rode D sobre si própria. As máquinas toma as seguintes ações, com a última linha sendo a contradição.

- H aceita $\langle M, w \rangle$ exatamente quando M aceita w.
- D rejeita $\langle M \rangle$ exatamente quando M aceita $\langle M \rangle$.
- D rejeita $\langle D \rangle$ exatamente quando D aceita $\langle D \rangle$.

Onde está a diagonalização na prova do Teorema 4.9? Ele se torna aparente quando você examina as tabelas de comportamento para as MT's H e D. Nessas tabelas listamos todas as MT's nas linhas, M_1, M_2, ... e todas as suas descrições nas colunas, $\langle M_1 \rangle$, $\langle M_2 \rangle$, ... As entradas dizem se a máquina em uma dada linha aceita a entrada em uma dada coluna. A entrada é *aceita* se a máquina aceita a entrada mas está em branco se ela rejeita ou entra em loop sobre aquela entrada. Montamos as entradas na figura abaixo para ilustrar a idéia.

Na figura seguinte as entradas são os resultados de rodar H sobre as entradas correspondentes na Figura 4.4. Portanto se M_3 não aceita a entrada $\langle M_2 \rangle$, a entrada para linha M_3 e coluna $\langle M_2 \rangle$ é *rejeita* porque H rejeita a entrada $\langle M_3, \langle M_2 \rangle \rangle$.

Na figura a seguir adicionamos D à Figura 4.5. Pela nossa hipótese, H é uma MT e portanto D também o é. Por conseguinte ela tem que ocorrer na lista M_1, M_2, ... de todas as MT's. Note que D computa o oposto das entradas da diagonal. A contradição ocorre no ponto onde está a interrogação pois a entrada aí tem que ser o oposto de si próprio.

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*aceita*		*aceita*		
M_2	*aceita*	*aceita*	*aceita*	*aceita*	
M_3					$\cdots$
M_4	*aceita*	*aceita*			
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figura 4.4: Entrada i, j é aceita se M_i aceita $\langle M_j \rangle$

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$
M_1	*aceita*	*rejeita*	*aceita*	*rejeita*	
M_2	*aceita*	*aceita*	*aceita*	*aceita*	
M_3	*rejeita*	*rejeita*	*rejeita*	*rejeita*	$\cdots$
M_4	*aceita*	*aceita*	*rejeita*	*rejeita*	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figura 4.5: A entrada i, j é o valor de H sobre a entrada $\langle M_i, \langle M_j \rangle \rangle$

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\cdots$	$\langle D \rangle$	$\cdots$
M_1	*aceita*	*rejeita*	*aceita*	*rejeita*		*aceita*	
M_2	*aceita*	*aceita*	*aceita*	*aceita*		*aceita*	
M_3	*rejeita*	*rejeita*	*rejeita*	*rejeita*	$\cdots$	*rejeita*	$\cdots$
M_4	*aceita*	*aceita*	*rejeita*	*rejeita*		*aceita*	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
D	*rejeita*	*rejeita*	*aceita*	*aceita*		?	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Figura 4.6: Se D está na figura, uma contradição ocorre em "?"

Uma linguagem Turing-irreconhecível

Na seção precedente exibimos uma linguagem, a saber, A_{MT}, que é indecidível. Agora exibimos uma linguagem que não é sequer Turing-reconhecível. Note que A_{MT} não bastará para esse propósito porque mostramos que A_{MT} é Turing-reconhecível na página 160. O teorema a seguir mostra que, se tanto uma linguagem quanto seu complemento são Turing-reconhecíveis, a linguagem é decidível. Portanto, para qualquer linguagem indecidível, ou ela ou seu complemento não é Turing-reconhecível. Lembre-se que o complemento de uma linguagem é a linguagem consistindo de todas as cadeias que não estão na linguagem. Dizemos que uma linguagem é ***co-Turing-reconhecível*** se ela é o complemento de uma linguagem Turing-reconhecível.

Teorema 4.16 . . . . . .
Uma linguagem é decidível se e somente se ela é tanto Turing-reconhecível quanto co-Turing-reconhecível.

Em outras palavras, uma linguagem é decidível se e somente se tanto ela quanto seu complemento são Turing-reconhecíveis.

Prova. Temos duas direções para provar. Primeiro, se A é decidível, podemos facilmente ver que tanto A quanto seu complemento $\overline{A}$ são Turing-reconhecíveis. Qualquer linguagem decidível é Turing-reconhecível, e o complemento de uma linguagem decidível também é decidível.

Para a outra direção, se tanto A quanto $\overline{A}$ são Turing-reconhecíveis, tomamos M_1 como sendo o reconhecedor para A e M_2 como o reconhecedor para $\overline{A}$. A seguinte máquina de Turing M é um decisor para A.

M = "Sobre a entrada w:

1. Rode ambas M_1 e M_2 sobre a entrada w em paralelo.

2. Se M_1 aceita, *aceite*; se M_2 aceita, *rejeite*."

Rodar as duas máquinas em paralelo significa que M tem duas fitas, uma para simular M_1 e outra para simular M_2. Nesse caso M alterna simulando um passo de cada máquina, o que continua até que uma delas páre.

Agora mostramos que M decide A. Toda cadeia w está em A ou em $\overline{A}$. Por conseguinte ou M_1 ou M_2 tem que aceitar w. Devido ao fato de que M pára sempre que M_1 ou M_2 aceita, M sempre pára e portanto ela é um decisor. Além do mais, ela aceita todas as cadeias em A e rejeita todas as cadeias que não estão em A. Portanto M é um decisor para A, e por conseguinte A é decidível.
. . . . . .

Corolário 4.17 . . . . . .
$\overline{A_{\mathsf{MT}}}$ não é Turing-reconhecível.

Prova. Sabemos que A_{MT} é Turing-reconhecível. Se $\overline{A_{\mathsf{MT}}}$ também fosse Turing-reconhecível, A_{MT} seria decidível. O Teorema 4.9 nos diz que A_{MT} não é decidível, portanto $\overline{A_{\mathsf{MT}}}$ tem que não-Turing-reconhecível.
. . . . . .