Parte I

Máquinas de Turing

Capítulo 1

Computações de Máquinas de Turing

1.1 Introdução

tur:mac:int:sec

O que significa dizer que uma função, digamos, de $\mathbb{N}$ para $\mathbb{N}$ é *computável*? Entre as primeiras respostas, e a mais conhecida, está a de que uma função é computável se puder ser computada por uma máquina de Turing. Essa noção foi apresentada por Alan Turing em 1936. As máquinas de Turing são um exemplo de *modelo de computação*—uma maneira matematicamente precisa de definir a ideia de um "procedimento computacional." O que exatamente isso significa é debatido, mas há amplo consenso de que as máquinas de Turing são uma forma de especificar procedimentos computacionais. Embora o termo "máquina de Turing" evoque a imagem de uma máquina física com partes móveis, estritamente falando uma máquina de Turing é um construto puramente matemático e, como tal, idealiza a ideia de um procedimento computacional. Por exemplo, não impomos restrição alguma aos requisitos de tempo ou memória de uma máquina de Turing: máquinas de Turing podem computar algo mesmo que a computação exija mais espaço de armazenamento ou mais passos do que há átomos no universo.

explanation

Talvez seja melhor pensar em uma máquina de Turing como um programa para um tipo especial de mecanismo imaginário. Esse mecanismo consiste em uma *fita* e uma *cabeça de leitura-escrita*. Na nossa versão das máquinas de Turing, a fita é infinita em uma direção (à direita) e é dividida em *quadrados*, cada um dos quais pode conter um símbolo de um *alfabeto* finito. Esses alfabetos podem conter qualquer número de símbolos diferentes, mas nos contentaremos em geral com três: $\triangleright$, 0 e 1. Quando o mecanismo é iniciado, a fita está vazia (ou seja, cada quadrado contém o símbolo 0), exceto o quadrado mais à esquerda, que contém $\triangleright$, e um número finito de quadrados que contêm a *entrada*. A qualquer momento, o mecanismo está em um de um número finito de *estados*. No início, a cabeça examina o quadrado mais à esquerda e em um *estado inicial* especificado. A cada passo da execução do mecanismo,

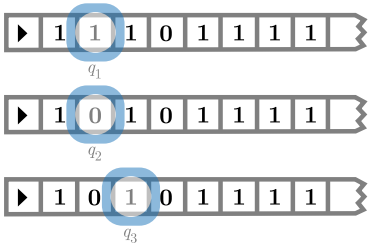


Figura 1.1: Uma máquina de Turing executando seu programa.

tur:mac:int:
fig:tm

o conteúdo do quadrado atualmente examinado, junto com o estado em que o mecanismo se encontra e o programa da máquina de Turing, determinam o que acontece em seguida. O programa da máquina de Turing é dado por uma função parcial que recebe como entrada um estado q e um símbolo σ e produz uma tripla $\langle q', \sigma', D\rangle$. Sempre que o mecanismo está no estado q e lê o símbolo σ, substitui o símbolo no quadrado atual por σ', a cabeça move-se à esquerda, à direita ou permanece no lugar conforme D seja L, R ou N, e o mecanismo passa ao estado q'.

Por exemplo, considere a situação em Figura 1.1. A parte visível da fita da máquina de Turing contém o símbolo de fim de fita ▷ no quadrado mais à esquerda, seguido de três 1's, um 0 e mais quatro 1's. A cabeça está lendo o terceiro quadrado da esquerda, que contém um 1, e está no estado q_1—dizemos que "a máquina está lendo um 1 no estado q_1." Se o programa da máquina de Turing retornar, para a entrada $\langle q_1, 1\rangle$, a tripla $\langle q_2, 0, N\rangle$, a máquina substituiria o 1 no terceiro quadrado por um 0, deixaria a cabeça de leitura/escrita onde está e passaria ao estado q_2. Se então o programa retornar $\langle q_3, 0, R\rangle$ para a entrada $\langle q_2, 0\rangle$, a máquina sobrescreveria o 0 com outro 0 (efetivamente, deixando inalterado o conteúdo da fita sob a cabeça de leitura/escrita), moveria um quadrado à direita e entraria no estado q_3. E assim por diante.

Dizemos que a máquina *para* quando encontra algum estado, q_n, e símbolo, σ, tais que não há instrução para $\langle q_n, \sigma\rangle$, ou seja, a função de transição para a entrada $\langle q_n, \sigma\rangle$ é indefinida. Em outras palavras, a máquina não tem instrução a executar e, nesse ponto, cessa a operação. A parada às vezes é representada por um estado de parada específico h. Isso será demonstrado com mais detalhes adiante.

A beleza do artigo de Turing, "On computable numbers," está em que ele apresenta não só uma definição formal, mas também um argumento de que a definição captura a noção intuitiva de computabilidade. Pela definição, deve ficar claro que toda função computável por uma máquina de Turing é computável no sentido intuitivo. Turing oferece três tipos de argumento de que o inverso é verdadeiro, ou seja, que toda função que naturalmente consideraríamos computável é computável por tal máquina. São eles (nas palavras de Turing):

digression

1. Um apelo direto à intuição.

2. Uma prova da equivalência de duas definições (caso a nova definição tenha maior apelo intuitivo).

3. Dar exemplos de grandes classes de números que são computáveis.

Nosso objetivo é tentar definir a noção de computabilidade "em princípio," ou seja, sem levar em conta limitações práticas de tempo e espaço. Claro, com a definição mais ampla de computabilidade em vigor, pode-se então considerar computação com recursos limitados; isso forma o cerne do assunto conhecido como "complexidade computacional."

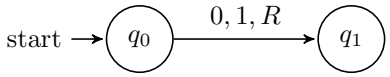
Observações históricas Alan Turing inventou as máquinas de Turing em 1936. Embora seu interesse na época fosse a decidibilidade da lógica de primeira ordem, o artigo tem sido descrito como um trabalho definitivo sobre os fundamentos do projeto de computadores. No artigo, Turing concentra-se em números reais computáveis, ou seja, números reais cujas expansões decimais são computáveis; mas observa que não é difícil adaptar suas noções a funções computáveis sobre os números naturais, e assim por diante. Note que isso foi cinco anos inteiros antes de o primeiro computador de propósito geral em funcionamento ser construído em 1941 (pelo alemão Konrad Zuse na sala de estar dos pais), sete anos antes de Turing e seus colegas em Bletchley Park construírem o Colossus que quebrava códigos (1943), nove anos antes do ENIAC americano (1945), doze anos antes do primeiro computador britânico de propósito geral—a Manchester Small-Scale Experimental Machine—ser construída em Manchester (1948), e treze anos antes dos americanos testarem o BINAC (1949). A Manchester SSEM tem a distinção de ser o primeiro computador com programa armazenado—as máquinas anteriores tinham de ser recabeada manualmente para cada nova tarefa.

1.2 Representando Máquinas de Turing

explanation

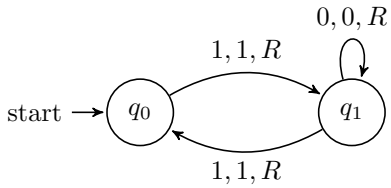
As máquinas de Turing podem ser representadas visualmente por *diagramas de estado*. Os diagramas são compostos por células de estado conectadas por setas. Sem surpresa, cada célula de estado representa um estado da máquina. Cada seta representa uma instrução que pode ser executada a partir desse estado, com as especificações da instrução escritas acima ou abaixo da seta apropriada. Considere a seguinte máquina, que tem apenas dois estados internos, q_0 e q_1, e uma instrução:

tur:mac:rep:sec



Lembre-se de que a máquina de Turing tem uma cabeça de leitura/escrita e uma fita com a entrada escrita nela. A instrução pode ser lida como *se lendo um* 0 *no estado* q_0*, escreva um* 1*, mova-se à direita, e vá para o estado* q_1. Isto é equivalente à função de transição que mapeia $\langle q_0, 0\rangle$ em $\langle q_1, 1, R\rangle$.

Exemplo 1.1. *Máquina Par*: A seguinte máquina de Turing para se, e somente se, houver um número par de 1 na fita (sob a suposição de que todos os 1 vêm antes do primeiro 0 na fita).



O diagrama de estado corresponde à seguinte função de transição:

$$\begin{aligned}\delta(q_0, 1) &= \langle q_1, 1, R\rangle,\\ \delta(q_1, 1) &= \langle q_0, 1, R\rangle,\\ \delta(q_1, 0) &= \langle q_1, 0, R\rangle\end{aligned}$$

A máquina acima para apenas quando a entrada é um número par de traços. Caso contrário, a máquina (teoricamente) continua a operar indefinidamente. Para qualquer máquina e entrada, é possível rastrear as *configurações* da máquina a fim de determinar a saída. Daremos uma definição formal de configurações mais adiante. Por enquanto, podemos pensar intuitivamente nas configurações como uma série de diagramas que mostram o estado da máquina em qualquer instante durante a operação. As configurações mostram o conteúdo da fita, o estado da máquina e a posição da cabeça de leitura/escrita. explanation

Vamos rastrear as configurações da máquina par se ela for iniciada com uma entrada de quatro 1. Nesse caso, esperamos que a máquina pare. Em seguida, executaremos a máquina com uma entrada de três 1, na qual a máquina rodará para sempre.

A máquina inicia no estado q_0, examinando o 1 mais à esquerda. Podemos representar o estado inicial da máquina da seguinte forma:

$$\triangleright 1_0 1110\ldots$$

A configuração acima é direta. Como se pode ver, a máquina inicia no estado um, examinando o 1 mais à esquerda. Isso é representado por um subscrito do nome do estado no primeiro 1. A instrução aplicável neste ponto é $\delta(q_0, 1) = \langle q_1, 1, R\rangle$, e assim a máquina move-se à direita na fita e muda para o estado q_1.

$$\triangleright 1 1_1 110\ldots$$

Como a máquina está agora no estado q_1 examinando um 1, temos que "seguir" a instrução $\delta(q_1, 1) = \langle q_0, 1, R\rangle$. Isso resulta na configuração

$$\triangleright 111_0 10\ldots$$

À medida que a máquina continua, as regras são aplicadas novamente na mesma ordem, resultando nas duas configurações a seguir:

$$\triangleright 1111_1 0 \ldots$$

$$\triangleright 11110_0 \ldots$$

A máquina está agora no estado q_0 examinando um 0. Com base no diagrama de transição, podemos ver facilmente que não há nenhuma instrução a ser executada, e portanto a máquina parou. Isso significa que a entrada foi aceita.

Suponha que, em seguida, iniciemos a máquina com uma entrada de três 1. As primeiras configurações são semelhantes, pois as mesmas instruções são executadas, com apenas uma pequena diferença na entrada da fita:

$$\triangleright 1_0 110 \ldots$$

$$\triangleright 11_1 10 \ldots$$

$$\triangleright 111_0 0 \ldots$$

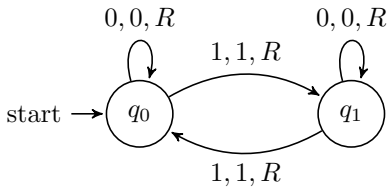
$$\triangleright 1110_1 \ldots$$

A máquina já percorreu todos os 1 e está lendo um 0 no estado q_1. Como mostrado no diagrama, há uma instrução da forma $\delta(q_1, 0) = \langle q_1, 0, R \rangle$. Como a fita está preenchida com 0 indefinidamente para a direita, a máquina continuará a executar esta instrução *para sempre*, permanecendo no estado q_1 e movendo-se cada vez mais à direita. A máquina nunca para, e não aceita a entrada.

explanation

É importante observar que nem todas as máquinas param. Se parar significa que a máquina fica sem instruções a executar, então podemos criar uma máquina que nunca para, simplesmente garantindo que haja uma seta de saída para cada símbolo em cada estado. A máquina par pode ser modificada para rodar indefinidamente adicionando uma instrução para examinar um 0 em q_0.

Exemplo 1.2.



explanation

As tabelas de máquina são outra forma de representar máquinas de Turing. As tabelas de máquina têm o alfabeto da fita exibido no eixo x, e o conjunto de estados da máquina ao longo do eixo y. Dentro da tabela, na interseção de cada estado e símbolo, está escrito o resto da instrução—o novo estado, o novo símbolo e a direção do movimento. As tabelas de máquina facilitam determinar em que estado, e para qual símbolo, a máquina para. Sempre que há uma lacuna na tabela, há um ponto possível para a máquina parar. Ao contrário

dos diagramas de estado e dos conjuntos de instruções, em que os pontos nos quais a máquina para nem sempre são imediatamente óbvios, quaisquer pontos de parada são rapidamente identificados ao encontrar as lacunas na tabela de máquina.

Exemplo 1.3. A tabela de máquina da máquina par é:

	0	1	▷
q_0		$1, q_1, R$	
q_1	$0, q_1, R$	$1, q_0, R$	

Como podemos ver, a máquina para ao examinar um 0 no estado q_0.

explanation

Até aqui consideramos apenas máquinas que leem e aceitam entradas. No entanto, as máquinas de Turing têm a capacidade de tanto ler quanto escrever. Um exemplo de tal máquina (embora haja muitos, muitos exemplos) é um *duplicador*. Um duplicador, quando iniciado com um bloco de n 1 na fita, produz como saída um bloco de $2n$ 1.

tur:mac:rep: ex:doubler

Exemplo 1.4. Antes de construir uma máquina duplicadora, é importante elaborar uma *estratégia* para resolver o problema. Como a máquina (tal como a formulamos) não consegue lembrar quantos 1 já leu, precisamos encontrar uma maneira de rastrear todos os 1 na fita. Uma dessas maneiras é separar a saída da entrada com um 0. A máquina pode então apagar o primeiro 1 da entrada, percorrer o resto da entrada, deixar um 0, e escrever dois novos 1. A máquina então voltará e encontrará o segundo 1 na entrada, e duplicará esse também. Para cada 1 de entrada, ela escreverá dois 1 de saída. Ao apagar a entrada conforme a máquina avança, podemos garantir que nenhum 1 seja perdido ou duplicado duas vezes. Quando toda a entrada é apagada, restarão $2n$ 1 na fita. O diagrama de estado da máquina de Turing resultante está representado na Figura 1.2.

Problema 1.1. Escolha uma entrada arbitrária e rastreie as configurações da máquina duplicadora em Exemplo 1.4.

Problema 1.2. Projete uma máquina de Turing com alfabeto $\{\triangleright, 0, A, B\}$ que aceite, isto é, pare, qualquer cadeia de A e B na qual o número de A seja igual ao número de B *e* todos os A precedam todos os B, e que rejeite, isto é, não pare, qualquer cadeia em que o número de A não seja igual ao número de B, ou os A não precedam todos os B. (Por exemplo, a máquina deve aceitar $AABB$, e $AAABBB$, mas rejeitar tanto AAB quanto $AABBAABB$.)

Problema 1.3. Projete uma máquina de Turing com alfabeto $\{\triangleright, 0, A, B\}$ que receba como entrada qualquer cadeia α de A e B e a duplique para produzir uma saída da forma $\alpha\alpha$. (Por exemplo, a entrada $ABBA$ deve resultar na saída $ABBAABBA$).

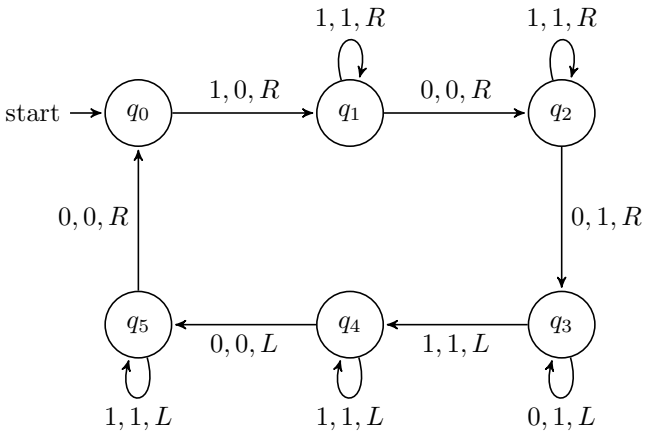


Figura 1.2: Uma máquina duplicadora

tur:mac:rep:fig:doubler

Problema 1.4. *Alfabética?:* Projete uma máquina de Turing com alfabeto $\{\triangleright, 0, A, B\}$ que, ao receber como entrada uma sequência finita de A e B, verifique se todos os A aparecem à esquerda de todos os B ou não. A máquina deve deixar a cadeia de entrada na fita, e ou parar, se a cadeia for "alfabética", ou rodar para sempre, se a cadeia não for.

Problema 1.5. *Ordenador alfabético:* Projete uma máquina de Turing com alfabeto $\{\triangleright, 0, A, B\}$ que receba como entrada uma sequência finita de A e B e os reorganize de modo que todos os A fiquem à esquerda de todos os B. (Por exemplo, a sequência $BABAA$ deve tornar-se a sequência $AAABB$, e a sequência $ABBABB$ deve tornar-se a sequência $AABBBB$).

1.3 Máquinas de Turing

explanation

A definição formal do que constitui uma máquina de Turing parece abstrata, mas é na verdade simples: apenas reúne em uma estrutura matemática toda a informação necessária para especificar o funcionamento de uma máquina de Turing. Isso inclui (1) em quais estados a máquina pode estar, (2) quais símbolos são permitidos na fita, (3) em qual estado a máquina deve começar e (4) qual é o conjunto de instruções da máquina.

tur:mac:tur:sec

Definição 1.5 (Máquina de Turing). Uma *máquina de Turing* M é uma tupla $\langle Q, \Sigma, q_0, \delta \rangle$ consistindo de

1. um conjunto finito de *estados* Q,
2. um *alfabeto* finito Σ que inclui $\triangleright$ e 0,

3. um *estado inicial* $q_0 \in Q$,

4. um *conjunto de instruções* finito $\delta\colon Q \times \Sigma \nrightarrow Q \times \Sigma \times \{L, R, N\}$.

A função parcial δ também é chamada de *função de transição* de M.

Supomos que a fita é infinita em apenas uma direção. Por essa razão é útil designar um símbolo especial $\triangleright$ como marcador da extremidade esquerda da fita. Isso facilita que programas de máquinas de Turing saibam quando estão "em perigo" de sair da fita. Poderíamos supor que esse símbolo nunca é sobrescrito, ou seja, que $\delta(q, \triangleright) = \langle q', \triangleright, x\rangle$ se $\delta(q, \triangleright)$ estiver definido. Alguns livros-texto fazem isso; nós não. Basta ter cuidado ao construir sua máquina de Turing para que ela nunca sobrescreva $\triangleright$. Além disso, há casos em que permitir tal sobrescrita oferece alguma flexibilidade conveniente. explanation

Exemplo 1.6. *Máquina dos pares:* A máquina dos pares é formalmente a quádrupla $\langle Q, \Sigma, q_0, \delta\rangle$ onde

$$\begin{aligned} Q &= \{q_0, q_1\} \\ \Sigma &= \{\triangleright, 0, 1\}, \\ \delta(q_0, 1) &= \langle q_1, 1, R\rangle, \\ \delta(q_1, 1) &= \langle q_0, 1, R\rangle, \\ \delta(q_1, 0) &= \langle q_1, 0, R\rangle. \end{aligned}$$

1.4 Configurações e Computações

cmp:tur:con:sec

Lembre-se de acompanhar as configurações da máquina dos pares anteriormente. O mecanismo imaginário formado pela fita, pela cabeça de leitura-escrita e pelo programa da máquina de Turing é, na verdade, apenas uma maneira intuitiva de visualizar o que é uma computação de máquina de Turing. Formalmente, podemos definir a computação de uma máquina de Turing sobre uma dada entrada como uma sequência de *configurações*—e uma configuração, por sua vez, é uma sequência de símbolos (correspondendo ao conteúdo da fita em um dado ponto da computação), um número que indica a posição da cabeça de leitura-escrita e um estado. Com isso, podemos definir o que a máquina de Turing M computa sobre uma dada entrada. explanation

Definição 1.7 (Configuração). Uma *configuração* da máquina de Turing $M = \langle Q, \Sigma, q_0, \delta\rangle$ é uma tripla $\langle C, m, q\rangle$ onde

1. $C \in \Sigma^*$ é uma sequência finita de símbolos de Σ,

2. $m \in \mathbb{N}$ é um número $< \mathrm{len}(C)$, e

3. $q \in Q$

Intuitivamente, a sequência C é o conteúdo da fita (símbolos de todos os quadrados desde o quadrado mais à esquerda até o último não vazio ou previamente visitado), m é o número do quadrado que a cabeça de leitura-escrita está examinando (começando com 0 como o número do quadrado mais à esquerda), e q é o estado atual da máquina.

explanation

A possível entrada para uma máquina de Turing é uma sequência de símbolos, geralmente uma sequência que codifica um número de alguma forma. A configuração inicial da máquina de Turing é aquela configuração em que iniciamos a máquina de Turing para trabalhar sobre essa entrada: a fita contém o marcador de fim de fita imediatamente seguido da entrada escrita nos quadrados à direita, a cabeça de leitura-escrita está examinando o quadrado mais à esquerda da entrada (ou seja, o quadrado à direita do marcador de fim esquerdo), e o mecanismo está no estado inicial designado q_0.

Definição 1.8 (Configuração inicial). A *configuração inicial* de M para a entrada $I \in \Sigma^*$ é

$$\langle \triangleright \frown I, 1, q_0 \rangle.$$

explanation

O símbolo $\frown$ denota *concatenação*—a cadeia de entrada começa imediatamente após o marcador de fim esquerdo.

Definição 1.9. Dizemos que uma configuração $\langle C, m, q \rangle$ *produz a configuração* $\langle C', m', q' \rangle$ *em um passo* (de acordo com M), sse

1. o m-ésimo símbolo de C é σ,
2. o conjunto de instruções de M especifica $\delta(q, \sigma) = \langle q', \sigma', D \rangle$,
3. o m-ésimo símbolo de C' é σ', e
4. a) $D = L$ e $m' = m - 1$ se $m > 0$, caso contrário $m' = 0$, ou
   b) $D = R$ e $m' = m + 1$, ou
   c) $D = N$ e $m' = m$,
5. se $m' = \text{len}(C)$, então $\text{len}(C') = \text{len}(C) + 1$ e o m'-ésimo símbolo de C' é 0. Caso contrário $\text{len}(C') = \text{len}(C)$.
6. para todo i tal que $i < \text{len}(C)$ e $i \neq m$, $C'(i) = C(i)$,

Definição 1.10. Uma *execução de M sobre a entrada I* é uma sequência C_i de configurações de M, onde C_0 é a configuração inicial de M para a entrada I, e cada C_i produz C_{i+1} em um passo.

cmp:tur:con: defn:run-output

Dizemos que M *para sobre a entrada I após k passos* se $C_k = \langle C, m, q \rangle$, o m-ésimo símbolo de C é σ, e $\delta(q, \sigma)$ é indefinido. Nesse caso, a *saída* de M para a entrada I é O, onde O é uma cadeia de símbolos que não termina em 0 tal que $C = \triangleright \frown O \frown 0^j$ para algum $j \in \mathbb{N}$. (0^j é uma sequência de j 0's.)

explanation

De acordo com esta definição, a saída O de M sempre termina em um símbolo diferente de 0, ou, se no instante k toda a fita estiver preenchida com 0 (exceto o $\triangleright$ mais à esquerda), O é a cadeia vazia.

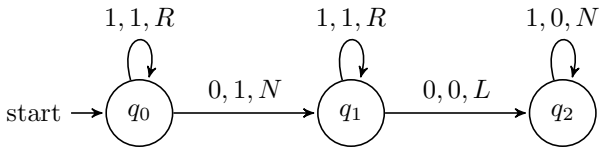


Figura 1.3: Uma máquina que computa $f(x, y) = x + y$

tur:mac:una: fig:adder

1.5 Representação Unária de Números

tur:mac:una: sec

As máquinas de Turing trabalham sobre sequências de símbolos escritos em sua fita. Dependendo do alfabeto que uma máquina de Turing usa, essas sequências de símbolos podem representar diversas entradas e saídas. De particular interesse, é claro, estão as máquinas de Turing que computam funções *aritméticas*, isto é, funções de números naturais. Uma maneira simples de representar inteiros positivos é codificá-los como sequências de um único símbolo 1. Se $n \in \mathbb{N}$, seja 1^n a sequência vazia se $n = 0$, e caso contrário a sequência composta de exatamente n 1. explanation

Definição 1.11 (Computação). Uma máquina de Turing M *computa* a função $f\colon \mathbb{N}^k \to \mathbb{N}$ sse M para com a entrada

$$1^{n_1}01^{n_2}0\ldots01^{n_k}$$

produzindo a saída $1^{f(n_1,\ldots,n_k)}$.

Problema 1.6. Dê uma definição para quando uma máquina de Turing M computa a função $f\colon \mathbb{N}^k \to \mathbb{N}^m$.

tur:mac:una: ex:adder

Exemplo 1.12. *Adição:* Vamos construir uma máquina que computa a função $f(n, m) = n+m$. Isso requer uma máquina que comece com dois blocos de 1 de comprimento n e m na fita, e pare com um único bloco composto de $n + m$ 1. Os dois blocos de entrada de 1 estão separados por um 0, então um método seria escrever um traço no quadrado que contém o 0, e apagar o último 1.

Problema 1.7. Rastreie as configurações da máquina de Exemplo 1.12 para a entrada $\langle 3, 2\rangle$. O que acontece se a máquina computa $0 + 0$?

Em Exemplo 1.4, demos um exemplo de uma máquina de Turing que recebe como entrada uma sequência de 1 e para com uma sequência de duas vezes mais 1 na fita—a máquina duplicadora. No entanto, como a saída contém 0 à esquerda do bloco duplicado de 1, ela não computa de fato a função $f(x) = 2x$, como você poderia ter suposto. Descreveremos duas maneiras de corrigir isso. explanation

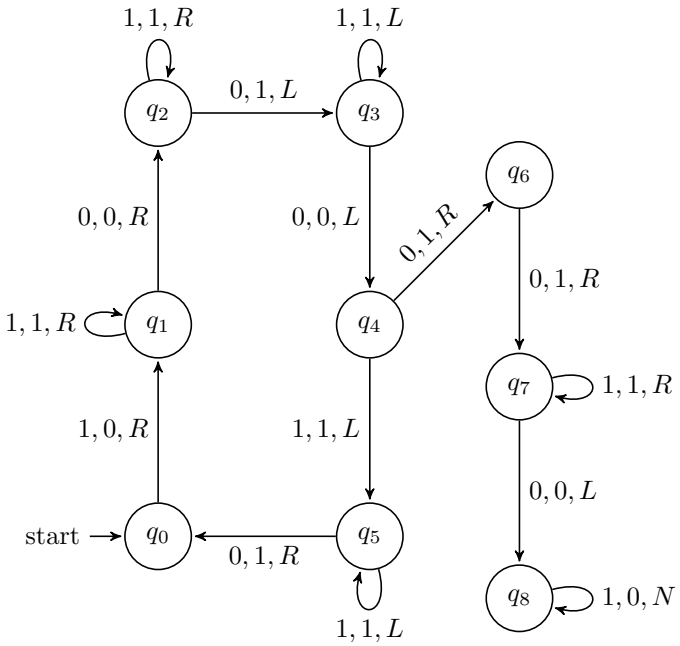


Figura 1.4: Uma máquina que computa $f(x) = 2x$

tur:mac:una: fig:doubler-disc

Exemplo 1.13. A máquina em Figura 1.4 computa a função $f(x) = 2x$. Em vez de apagar a entrada e escrever dois 1 na extremidade direita para cada 1 da entrada, como faz a máquina de Exemplo 1.4, esta máquina adiciona um único 1 à direita para cada 1 da entrada. Ela precisa rastrear onde a entrada termina, então deixa um 0 entre a entrada e os traços adicionados, que ela preenche com um 1 bem ao final. E temos que "lembrar" onde estamos na entrada, então substituímos temporariamente um 1 do bloco de entrada por um 0.

tur:mac:una: ex:mover

Exemplo 1.14. Uma segunda possibilidade para computar $f(x) = 2x$ é manter a máquina duplicadora original, mas adicionar estados e instruções ao final que movam o bloco duplicado de traços para a extremidade esquerda da fita. A máquina em Figura 1.5 faz exatamente esta última parte: iniciada em uma fita composta de um bloco de 0 seguido de um bloco de 1 (e com a cabeça posicionada em qualquer lugar do bloco de 0), ela apaga os 1 um a um e os escreve no início da fita. Para conseguir saber quando ela termina, primeiro marca o fim do bloco de 1 com um símbolo $\triangleright$, que é apagado ao final. Começamos a numerar os estados em q_6, de modo que possam ser adicionados à máquina duplicadora. Tudo de que você precisará é uma instrução adicional $\delta(q_0, 0) = \langle q_6, 0, N\rangle$, isto

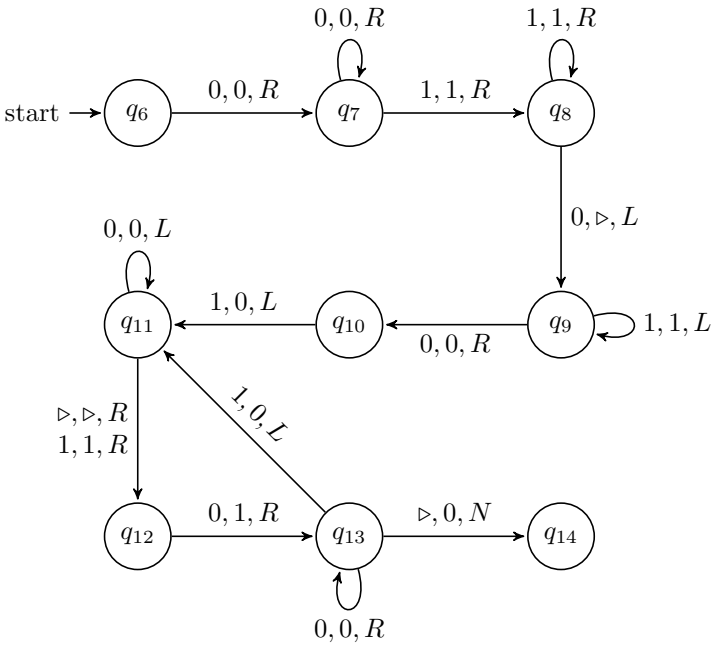


Figura 1.5: Movendo um bloco de 1 para a esquerda

tur:mac:una: fig:mover

é, uma seta de q_0 para q_6 rotulada $0, 0, N$. (Há um problema sutil: a máquina resultante não funciona para a entrada $x = 0$. Deixaremos isso como exercício.)

Problema 1.8. Em Exemplo 1.14 descrevemos uma máquina composta de uma combinação da máquina duplicadora de Figura 1.4 e da máquina movedora de Figura 1.5. O que acontece se você inicia essa máquina combinada com a entrada $x = 0$, isto é, em uma fita vazia? Como você corrigiria a máquina de modo que, nesse caso, ela parasse com saída $2x = 0$? (Você deve conseguir fazer isso adicionando um estado e uma transição.)

Problema 1.9. *Subtração:* Projete uma máquina de Turing que, ao receber uma entrada de duas cadeias não vazias de traços de comprimento n e m, em que $n > m$, compute a função $f(n, m) = n - m$.

Problema 1.10. *Igualdade:* Projete uma máquina de Turing para computar a seguinte função:

$$\text{igualdade}(n, m) = \begin{cases} 1 & \text{se } n = m \\ 0 & \text{se } n \neq m \end{cases}$$

em que n e $m \in \mathbb{Z}^+$.

Problema 1.11. Projete uma máquina de Turing para computar a função $\min(x, y)$, em que x e y são inteiros positivos representados na fita por cadeias de 1 separadas por um 0. Você pode usar símbolos adicionais no alfabeto da máquina.

A função min seleciona o menor valor entre seus argumentos, de modo que $\min(3, 5) = 3$, $\min(20, 16) = 16$, e $\min(4, 4) = 4$, e assim por diante.

Definição 1.15. Uma máquina de Turing M computa a função parcial $f\colon \mathbb{N}^k \nrightarrow \mathbb{N}$ sse,

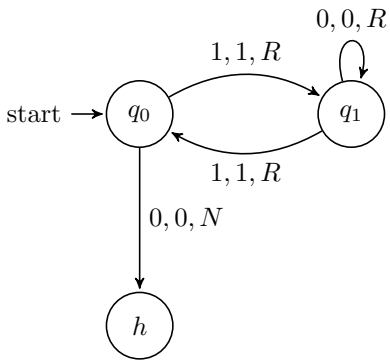
1. M para com a entrada $1^{n_1} \frown 0 \frown \ldots \frown 0 \frown 1^{n_k}$ produzindo a saída 1^m se $f(n_1, \ldots, n_k) = m$.
2. M não para de modo algum, ou para com uma saída que não é um único bloco de 1 se $f(n_1, \ldots, n_k)$ é indefinida.

1.6 Estados de Parada

explanation Embora tenhamos definido nossas máquinas para parar apenas quando não há instrução a executar, representações comuns de máquinas de Turing têm um *estado de parada* dedicado h, de modo que $h \in Q$. tur:mac:hal:sec

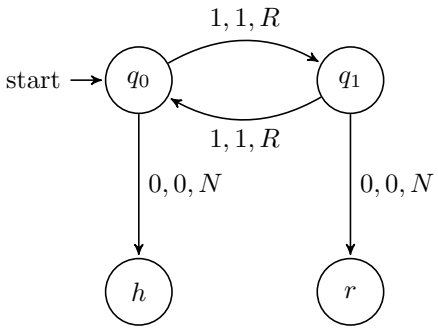
A ideia por trás de um estado de parada é simples: quando a máquina terminou a operação (está pronta para aceitar a entrada, ou terminou de escrever a saída), ela entra em um estado h onde para. Algumas máquinas têm dois estados de parada, um que aceita a entrada e outro que rejeita a entrada.

Exemplo 1.16. *Estados de parada.* Para elucidar esse conceito, comecemos com uma alteração da máquina dos pares. Em vez de fazer a máquina parar no estado q_0 se a entrada for par, podemos adicionar uma instrução para enviar a máquina a um estado de parada.



Vamos expandir ainda mais o exemplo. Quando a máquina determina que a entrada é ímpar, ela nunca para. Podemos alterar a máquina para incluir um

estado de *rejeição* substituindo a instrução em laço por uma instrução para ir a um estado de rejeição r.



Adicionar um estado de parada dedicado pode ser vantajoso em casos como este, em que deixa explícito quando a máquina aceita/rejeita certas entradas. No entanto, é importante notar que nenhum poder computacional é ganho ao adicionar um estado de parada dedicado. Da mesma forma, uma noção menos formal de parada tem suas próprias vantagens. A definição de parada usada até agora neste capítulo torna a prova do *Problema da Parada* intuitiva e fácil de demonstrar. Por essa razão, continuamos com nossa definição original. explanation

1.7 Máquinas Disciplinadas

tur:mac:dis:sec

Na Seção 1.6, consideramos máquinas de Turing que têm um único estado de parada designado h—tais máquinas são garantidas de parar, se pararem de fato, no estado h. Dessa forma, máquinas com um único estado de parada são mais "disciplinadas" do que permitimos que máquinas de Turing sejam em geral. Há outras restrições que podemos impor ao comportamento das máquinas de Turing. Por exemplo, também não proibimos máquinas de Turing de apagar o marcador de fim de fita no quadrado 0, ou de tentar mover-se à esquerda a partir do quadrado 0. (Nossa definição afirma que a cabeça simplesmente permanece no quadrado 0 nesse caso; outras definições fazem a máquina parar.) Da mesma forma, às vezes é desejável poder supor que uma máquina de Turing, se parar de fato, para no quadrado 1. explanation

tur:mac:dis:defn:disciplined

Definição 1.17. Uma máquina de Turing M é *disciplinada* sse

1. ela tem um único estado de parada designado h,
2. ela para, se parar de fato, enquanto examina o quadrado 1,
3. ela nunca apaga o símbolo ▷ no quadrado 0, e
4. ela nunca tenta mover-se à esquerda a partir do quadrado 0.

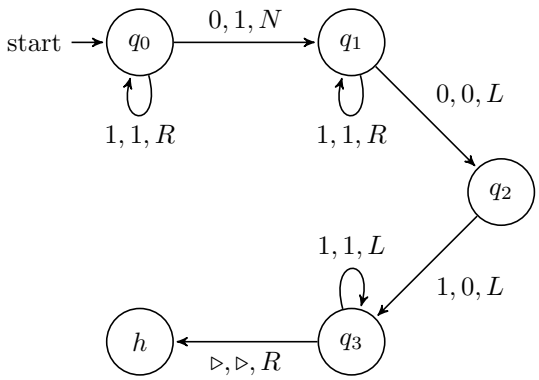


Figura 1.6: Uma máquina de adição disciplinada

tur:mac:dis:fig:adder-disc

explanation

Já discutimos que qualquer máquina de Turing pode ser transformada em uma com o mesmo comportamento, mas com um estado de parada designado. Isso é feito simplesmente adicionando um novo estado h, e adicionando uma instrução $\delta(q, \sigma) = \langle h, \sigma, N\rangle$ para qualquer par $\langle q, \sigma\rangle$ onde o δ original é indefinido. É verdade, embora tedioso de provar, que qualquer máquina de Turing M pode ser transformada em uma máquina de Turing disciplinada M' que para nas mesmas entradas e produz a mesma saída. Por exemplo, se a máquina de Turing para e não está no quadrado 1, podemos adicionar algumas instruções para fazer a cabeça mover-se à esquerda até encontrar o marcador de fim de fita, depois mover um quadrado à direita, e então parar. Deixamos a você pensar sobre como as outras condições podem ser tratadas.

Exemplo 1.18. Em Figura 1.6, transformamos a máquina de adição de Exemplo 1.12 em uma máquina disciplinada.

tur:mac:dis:prop:disciplined

Proposição 1.19. *Para toda máquina de Turing M, há uma máquina de Turing disciplinada M' que para com saída O se M para com saída O, e não para se M não para. Em particular, qualquer função $f\colon \mathbb{N}^n \to \mathbb{N}$ computável por uma máquina de Turing também é computável por uma máquina de Turing disciplinada.*

Problema 1.12. Dê uma máquina disciplinada que computa $f(x) = x + 1$.

Problema 1.13. Encontre uma máquina disciplinada que, quando iniciada com entrada 1^n produz saída $1^n \frown 0 \frown 1^n$.

1.8 Combinando Máquinas de Turing

explanation

tur:mac:cmb:sec

Os exemplos de máquinas de Turing que vimos até agora foram bastante simples por natureza. Mas, de fato, qualquer problema que possa ser resolvido com qualquer linguagem de programação moderna também pode ser resolvido com máquinas de Turing. Para construir máquinas de Turing mais complexas, é importante nos convencer de que podemos combiná-las, de modo a construir máquinas que resolvam problemas mais complexos quebrando o procedimento em partes mais simples. Se conseguirmos encontrar uma maneira natural de quebrar um problema complexo em partes constituintes, podemos atacar o problema em vários estágios, criando várias máquinas de Turing simples e combinando-as em uma única máquina capaz de resolver o problema. Esse ponto é especialmente importante ao atacar o Problema da Parada na próxima seção.

Como combinamos as máquinas de Turing $M = \langle Q, \Sigma, q_0, \delta\rangle$ e $M' = \langle Q', \Sigma', q_0', \delta'\rangle$? Usamos agora a configuração da fita após M ter parado como a configuração de entrada de uma execução da máquina M'. Para obter uma única máquina de Turing $M \frown M'$ que faz isso, faça o seguinte:

1. Renumere (ou rerrotule) todos os estados Q' de M' de modo que M e M' não tenham estados em comum ($Q \cap Q' = \emptyset$).

2. Os estados de $M \frown M'$ são $Q \cup Q'$.

3. O alfabeto da fita é $\Sigma \cup \Sigma'$.

4. O estado inicial é q_0.

5. A função de transição é a função δ'' dada por:

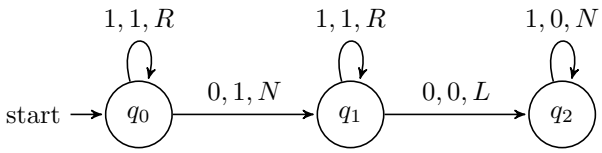
$$\delta''(q,\sigma) = \begin{cases} \delta(q,\sigma) & \text{se } q \in Q \\ \delta'(q,\sigma) & \text{se } q \in Q' \\ \langle q_0', \sigma, N\rangle & \text{se } q \in Q \text{ e } \delta(q,\sigma) \text{ é indefinida} \end{cases}$$

A máquina resultante usa as instruções de M quando está em um estado $q \in Q$, e as instruções de M' quando está em um estado $q \in Q'$. Quando está em um estado $q \in Q$ e examina um símbolo σ para o qual M não tem transição (isto é, M teria parado), ela entra no estado inicial de M' (e deixa o conteúdo da fita e a posição da cabeça como estão).

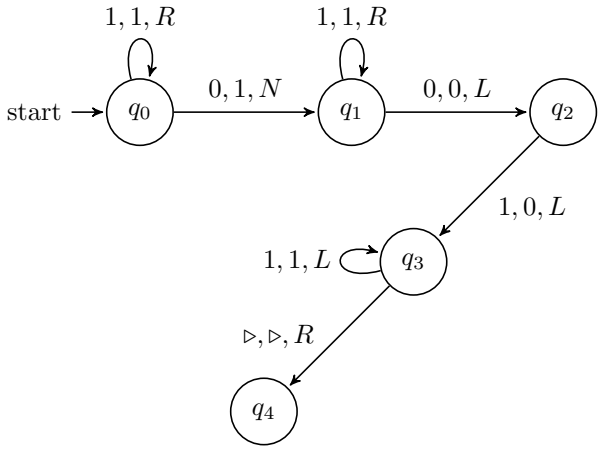
Note que, a menos que a máquina M seja disciplinada, não sabemos onde está a cabeça da fita quando M para, então a configuração de parada de M não precisa ter a cabeça examinando o quadrado 1. Ao combinar máquinas, é importante ter isso em mente.

Exemplo 1.20. *Combinando Máquinas:* Projetaremos uma máquina que, quando iniciada com uma entrada composta de dois blocos de 1 de comprimento n e m, para com um único bloco de $2(m+n)$ 1 na fita. Para construir essa máquina, podemos combinar duas máquinas que já conhecemos: a máquina de adição e a

duplicadora. Começamos desenhando um diagrama de estado para a máquina de adição.



Em vez de parar no estado q_2, queremos continuar a operação a fim de duplicar a saída. Lembre-se de que a máquina duplicadora apaga o primeiro traço da entrada e escreve dois traços em uma saída separada. Vamos adicionar uma instrução para garantir que a cabeça da fita esteja lendo o primeiro traço da saída da máquina de adição.



Agora é fácil duplicar a entrada—tudo o que temos a fazer é conectar a máquina duplicadora ao estado q_4. Isso requer renomear os estados da máquina duplicadora de modo que comecem em q_4 em vez de q_0—dessa forma não acabamos com dois estados iniciais. O diagrama final deve ter a aparência da Figura 1.7.

Proposição 1.21. *Se M e M' são disciplinadas e computam as funções $f\colon \mathbb{N}^k \to \mathbb{N}$ e $f'\colon \mathbb{N} \to \mathbb{N}$, respectivamente, então $M \frown M'$ é disciplinada e computa $f' \circ f$.*

Prova. Como M é disciplinada, quando para com saída $f(n_1, \ldots, n_k) = m$, a cabeça está examinando o quadrado 1. Se agora entrarmos no estado inicial de M', a máquina parará com saída $f'(m)$, novamente examinando o quadrado 1. As demais condições de Definição 1.17 também são satisfeitas. □

Problema 1.14. Dê uma máquina de Turing disciplinada que computa $f(x) = x + 2$ tomando a máquina M de Problema 1.12 e construindo $M \frown M$.

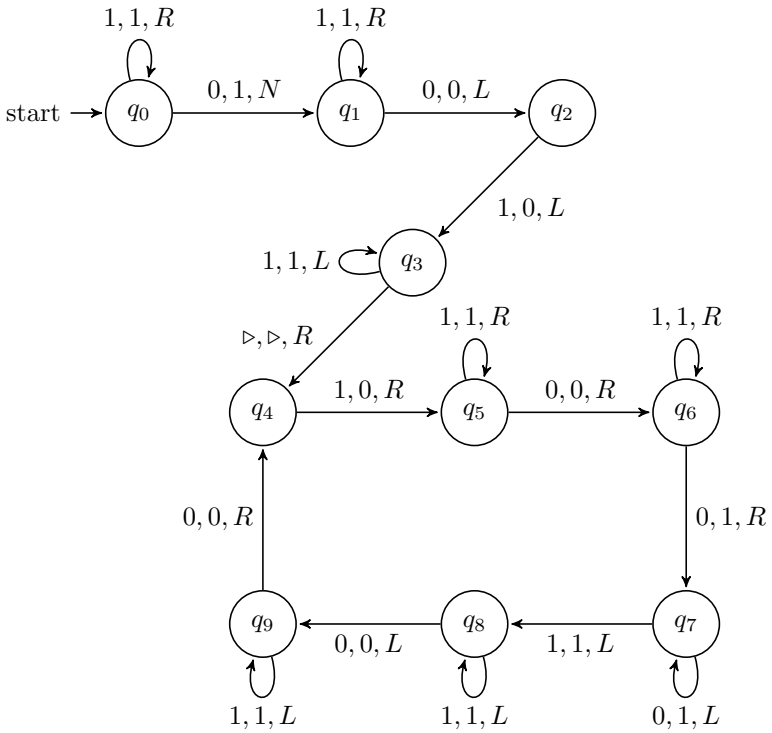


Figura 1.7: Combinando as máquinas de adição e duplicadora

tur:mac:cmb:fig:combined

1.9 Variantes de Máquinas de Turing

tur:mac:var:sec

Na verdade, há muitas maneiras possíveis de definir máquinas de Turing, das quais a nossa é apenas uma. De certas formas, nossa definição é mais liberal do que outras. Permitimos alfabetos finitos arbitrários; uma definição mais restrita poderia permitir apenas dois símbolos de fita, 1 e 0. Permitimos que a máquina escreva um símbolo na fita e se mova ao mesmo tempo; outras definições permitem escrever ou mover. Permitimos a possibilidade de escrever sem mover a cabeça da fita; outras definições omitem a "instrução" N. Em outros aspectos, nossa definição é mais restritiva. Assumimos que a fita é infinita em apenas uma direção; outras definições permitem que a fita seja infinita tanto para a esquerda quanto para a direita. Na verdade, pode-se até mesmo permitir qualquer número de fitas separadas, ou até mesmo uma grade infinita de quadrados. Representamos o conjunto de instruções da máquina de Turing por uma função de transição; outras definições usam uma relação de transição em que a máquina tem mais de uma instrução possível em qualquer

situação dada.

Essa última flexibilização da definição é particularmente interessante. Em nossa definição, quando a máquina está no estado q lendo o símbolo σ, $\delta(q, \sigma)$ determina qual é o novo símbolo, o novo estado e a nova posição da cabeça da fita. Mas se permitirmos que o conjunto de instruções seja uma relação entre pares estado-símbolo atuais $\langle q, \sigma\rangle$ e novas triplas estado-símbolo-direção $\langle q', \sigma', D\rangle$, a ação da máquina de Turing pode não ser exclusivamente determinada — a relação de instrução pode conter tanto $\langle q, \sigma, q', \sigma', D\rangle$ quanto $\langle q, \sigma, q'', \sigma'', D'\rangle$. Nesse caso, temos uma máquina de Turing *não determinística*. Estas desempenham um papel importante na teoria da complexidade computacional.

Há também diferentes convenções para quando uma máquina de Turing para: dizemos que ela para quando a função de transição é indefinida; outras definições exigem que a máquina esteja em um estado de parada especial designado. Explicamos na Seção 1.6 por que exigir um estado de parada designado não é uma restrição que impacta o que as máquinas de Turing podem calcular. Como as fitas de nossas máquinas de Turing são infinitas em apenas uma direção, há casos em que uma máquina de Turing não consegue executar adequadamente uma instrução: se ela lê o quadrado mais à esquerda e deveria se mover para a esquerda. De acordo com a nossa definição, ela simplesmente permanece no lugar em vez de "cair", mas poderíamos tê-la definido de modo que ela pare quando isso acontecer. Essa definição também é equivalente: poderíamos simular o comportamento de uma máquina de Turing que para quando tenta se mover para a esquerda do quadrado 0 excluindo cada transição $\delta(q, \triangleright) = \langle q', \sigma, L\rangle$—então, em vez de tentar mover para a esquerda em $\triangleright$, a máquina para.[1]

Há também diferentes maneiras de representar números (e, portanto, a função de entrada-saída calculada por uma máquina de Turing): usamos representação unária, mas você também pode usar representação binária. Isso requer dois símbolos além de 0 e $\triangleright$.

Agora, aqui está um fato interessante: nenhuma dessas variações importa para quais funções são Turing-computáveis. *Se uma função é Turing-computável segundo uma definição, ela é Turing-computável segundo todas elas.*

Não entraremos nos detalhes de como verificar isso. Eis apenas um exemplo: não ganhamos nenhum poder computacional adicional ao permitir uma fita que é infinita em ambas as direções, ou múltiplas fitas. A razão é, aproximadamente, que uma máquina de Turing com uma única fita infinita unidirecional pode simular múltiplas fitas ou fitas infinitas bidirecionais. Por exemplo, usando estados e instruções adicionais, podemos "traduzir" um programa para uma máquina com múltiplas fitas ou fita infinita bidirecional em um programa para uma máquina com uma única fita infinita unidirecional. A máquina traduzida pode usar os quadrados pares para os quadrados da fita 1 (ou os quadrados

[1] Isso não funciona *muito bem*, pois nada nos impede de escrever e ler $\triangleright$ em quadrados que não sejam o quadrado 0 (veja Exemplo 1.14). Podemos contornar isso adicionando um segundo símbolo $\triangleright'$ para usar em seu lugar para tal propósito.

"positivos" de uma fita infinita bidirecional) e os quadrados ímpares para os quadrados da fita 2 (ou os quadrados "negativos").

1.10 A Tese de Church–Turing

tur:mac:ctt:sec

As máquinas de Turing devem ser um substituto preciso do conceito de procedimento efetivo. Turing pensava que quem compreendesse tanto o conceito de procedimento efetivo quanto o de máquina de Turing teria a intuição de que tudo o que pudesse ser feito por meio de um procedimento efetivo poderia ser feito por uma máquina de Turing. Essa afirmação é sustentada pelo fato de que todas as outras propostas precisas de substituto do conceito de procedimento efetivo se mostram extensionalmente equivalentes ao conceito de máquina de Turing —ou seja, podem computar exatamente o mesmo conjunto de funções. Essa afirmação chama-se *tese de Church–Turing*.

Definição 1.22 (Tese de Church–Turing). A *tese de Church–Turing* afirma que tudo o que é computável por meio de um procedimento efetivo é computável por máquina de Turing.

A tese de Church–Turing é invocada de duas maneiras. O primeiro tipo de uso é uma desculpa para a preguiça. Suponha que temos uma descrição de um procedimento efetivo para computar algo, digamos, em "pseudocódigo." Então podemos invocar a tese de Church–Turing para justificar a afirmação de que a mesma função é computada por alguma máquina de Turing, mesmo que não a tenhamos de fato construído.

O outro uso da tese de Church–Turing é mais filosoficamente interessante. Pode-se mostrar que existem funções que não podem ser computadas por máquinas de Turing. Disso, usando a tese de Church–Turing, conclui-se que não podem ser computadas efetivamente, usando qualquer procedimento que seja. Pois se houvesse tal procedimento, pela tese de Church–Turing seguiria que haveria uma máquina de Turing para ele. Assim, se podemos provar que não há máquina de Turing que o compute, também não pode haver um procedimento efetivo. Em particular, a tese de Church–Turing é invocada para afirmar que o chamado problema da parada não só não pode ser resolvido por máquinas de Turing, como não pode ser resolvido efetivamente de modo algum.

Capítulo 2

Indecidibilidade

2.1 Introdução

tur:und:int:sec

Pode parecer óbvio que nem toda função, nem mesmo toda função aritmética, pode ser computável. Há simplesmente um número grande demais delas, cujo comportamento é complicado demais. Funções definidas a partir do decaimento de partículas radioativas, por exemplo, ou de outro comportamento caótico ou aleatório. Suponha que comecemos a contar intervalos de 1 segundo a partir de um dado instante e definamos a função $f(n)$ como o número de partículas no universo que decaem no n-ésimo intervalo de 1 segundo após aquele momento inicial. Esta parece ser uma candidata a função que jamais poderemos ter esperança de computar.

Mas uma coisa é não conseguir imaginar como se computaria tais funções, e outra bem diferente é de fato provar que elas são incomputáveis. Na verdade, mesmo funções que parecem irremediavelmente complicadas podem, em um sentido abstrato, ser computáveis. Por exemplo, suponha que o universo seja finito no tempo—algum dia, num futuro muito distante, o universo se contrairá em um único ponto, como preveem algumas teorias cosmológicas. Então há apenas um número finito (mas incrivelmente grande) de segundos a partir daquele momento inicial para os quais $f(n)$ está definida. E qualquer função que esteja definida para apenas finitas entradas é computável: poderíamos listar as saídas em uma grande tabela, ou codificá-la em um diagrama de transição de estados de máquina de Turing muito grande.

Frequentemente estamos interessados em casos especiais de funções cujos valores fornecem as respostas a perguntas de sim/não. Por exemplo, a pergunta "n é um número primo?" está associada à função

$$\text{primo}(n) = \begin{cases} 1 & \text{se } n \text{ é primo} \\ 0 & \text{caso contrário.} \end{cases}$$

Dizemos que uma pergunta de sim/não pode ser *efetivamente decidida* se a função associada de valores 1/0 é efetivamente computável.

Para provar matematicamente que há funções que não podem ser efetivamente computadas, ou problemas que não podem ser efetivamente decididos, é essencial fixar um modelo específico de computação e mostrar que há funções que ele não pode computar ou problemas que ele não pode decidir. Podemos mostrar, por exemplo, que nem toda função pode ser computada por máquinas de Turing, e nem todo problema pode ser decidido por máquinas de Turing. Podemos então apelar para a tese de Church-Turing para concluir que não apenas as máquinas de Turing não são poderosas o suficiente para computar toda função, como também nenhum procedimento efetivo o é.

A chave para provar tais resultados negativos é o fato de que podemos atribuir números às próprias máquinas de Turing. A maneira mais fácil de fazer isso é enumerá-las, talvez fixando um modo específico de escrever máquinas de Turing e seus programas e, em seguida, listando-os de maneira sistemática. Uma vez que vejamos que isso pode ser feito, então a existência de funções Turing-incomputáveis decorre de simples considerações de cardinalidade: o conjunto de funções de $\mathbb{N}$ a $\mathbb{N}$ (na verdade, mesmo apenas de $\mathbb{N}$ a $\{0, 1\}$) é não enumerável, mas como podemos enumerar todas as máquinas de Turing, o conjunto de funções Turing-computáveis é apenas infinito enumerável.

Também podemos definir funções e problemas *específicos* que podemos provar ser incomputáveis e indecidíveis, respectivamente. Um desses problemas é o assim chamado *Problema da Parada.* Máquinas de Turing podem ser finitamente descritas listando-se suas instruções. Tal descrição de uma máquina de Turing, isto é, um programa de máquina de Turing, pode é claro ser usada como entrada para outra máquina de Turing. Assim, podemos considerar máquinas de Turing que decidem perguntas sobre outras máquinas de Turing. Uma pergunta particularmente interessante é esta: "A máquina de Turing dada eventualmente para quando iniciada na entrada n?" Seria bom se houvesse uma máquina de Turing que pudesse decidir essa pergunta: pense nela como uma máquina de Turing de controle de qualidade que garante que as máquinas de Turing não fiquem presas em laços infinitos e coisas do tipo. O fato interessante, que Turing provou, é que não pode haver tal máquina de Turing. Não pode haver uma única máquina de Turing que, quando iniciada em uma entrada que consiste em uma descrição de uma máquina de Turing M e algum número n, sempre pare com saída 1 ou 0 conforme a máquina M teria parado ou não quando iniciada na entrada n.

Uma vez que tenhamos exemplos de problemas indecidíveis específicos, podemos usá-los para mostrar que outros problemas também são indecidíveis. Por exemplo, um célebre problema indecidível é a pergunta "A fórmula φ de primeira ordem é válida?". Não há máquina de Turing que, dada como entrada uma fórmula φ de primeira ordem, tenha garantia de parar com saída 1 ou 0 conforme φ seja válida ou não. Historicamente, a questão de encontrar um procedimento para resolver efetivamente esse problema foi chamada simplesmente de "o" problema da decisão; e assim dizemos que o problema da decisão é insolúvel. Turing e Church provaram esse resultado independentemente mais ou menos na mesma época, de modo que ele também é chamado de Teorema de Church-Turing.

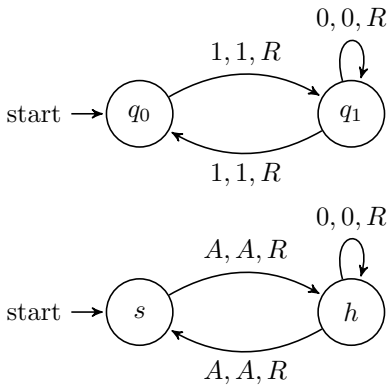


Figura 2.1: Variantes da máquina *Par*

tur:und:enu:fig:variants

2.2 Enumerando Máquinas de Turing

explanation

tur:und:enu:sec

Podemos mostrar que o conjunto de todas as máquinas de Turing é enumerável. Isso decorre do fato de que cada máquina de Turing pode ser finitamente descrita. O conjunto de estados e o vocabulário da fita são conjuntos finitos. A função de transição é uma função parcial de $Q \times \Sigma$ a $Q \times \Sigma \times \{L, R, N\}$ e, portanto, também pode ser especificada listando-se seus valores para os finitos pares de argumentos para os quais está definida.

Isso é verdade até certo ponto, mas há uma diferença sutil. A definição de máquinas de Turing não fez restrição alguma sobre quais membros o conjunto de estados e o alfabeto da fita podem ter. Assim, por exemplo, para todo número real, há tecnicamente uma máquina de Turing que usa esse número como um estado. Contudo, o *comportamento* da máquina de Turing é independente de quais objetos servem como estados e vocabulário. Considere as duas máquinas de Turing na Figura 2.1. Esses dois diagramas correspondem a duas máquinas, M com o alfabeto de fita $\Sigma = \{\triangleright, 0, 1\}$ e conjunto de estados $\{q_0, q_1\}$, e M' com alfabeto $\Sigma' = \{\triangleright, 0, A\}$ e estados $\{s, h\}$. Mas, quanto ao resto, suas instruções são as mesmas: M para em uma sequência de n 1 se, e somente se, n é par, e M' para em uma sequência de n A se, e somente se, n é par. Tudo o que fizemos foi renomear 1 para A, q_0 para s e q_1 para h. Esse exemplo se generaliza: podemos considerar máquinas de Turing como sendo a mesma desde que uma resulte da outra por tal renomeação de símbolos e estados. Na verdade, podemos simplesmente pensar nos símbolos e estados de uma máquina de Turing como inteiros positivos: em vez de σ_0 pense 1, em vez de σ_1 pense 2, etc.; $\triangleright$ é 1, 0 é 2, etc. Dessa forma, a máquina *Par* torna-se a máquina representada na Figura 2.2. Poderíamos chamar de máquina *padrão* uma máquina de Turing cujos estados e símbolos são inteiros positivos,

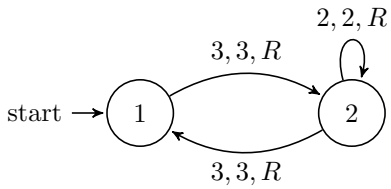


Figura 2.2: Uma máquina *Par* padrão

tur:und:enu:
fig:standard-even

e considerar apenas máquinas padrão de agora em diante.[1]

Queríamos mostrar que o conjunto de máquinas de Turing é enumerável e, tendo em mente as considerações acima, basta mostrar que o conjunto de máquinas de Turing padrão é enumerável. Suponha que nos seja dada uma máquina de Turing padrão $M = \langle Q, \Sigma, q_0, \delta\rangle$. Como poderíamos descrevê-la usando uma cadeia finita de inteiros positivos? Primeiro listaremos o número de estados, os próprios estados, o número de símbolos, os próprios símbolos e o estado inicial. (Lembre-se, todos esses são inteiros positivos, já que M é uma máquina padrão.) E quanto a δ? O conjunto de argumentos possíveis, isto é, pares $\langle q, \sigma\rangle$, é finito, já que Q e Σ são finitos. Assim, a informação em δ é simplesmente a lista finita de todas as 5-uplas $\langle q, \sigma, q', \sigma', d\rangle$ em que $\delta(q, \sigma) = \langle q', \sigma', D\rangle$, e d é um número que codifica a direção D (digamos, 1 para L, 2 para R e 3 para N).

Dessa forma, toda máquina de Turing padrão pode ser descrita por uma lista finita de inteiros positivos, isto é, como uma sequência $s_M \in (\mathbb{Z}^+)^*$. Por exemplo, a máquina *Par* padrão é codificada pela sequência

$$2, \underbrace{1, 2}_{Q}, 3, \overbrace{1, 2, 3}^{\Sigma}, 1, \underbrace{1, 3, 2, 3, 2}_{\delta(1,3)=\langle 2,3,R\rangle}, \overbrace{2, 2, 2, 2, 2}^{\delta(2,2)=\langle 2,2,R\rangle}, \underbrace{2, 3, 1, 3, 2}_{\delta(2,3)=\langle 1,3,R\rangle}.$$

Teorema 2.1. *Há funções de* $\mathbb{N}$ *a* $\mathbb{N}$ *que não são Turing-computáveis.*

Prova. Sabemos que o conjunto de sequências finitas de inteiros positivos $(\mathbb{Z}^+)^*$ é enumerável (**??**). Isso nos dá que o conjunto de descrições de máquinas de Turing padrão, como um subconjunto de $(\mathbb{Z}^+)^*$, é ele próprio enumerável. Toda função Turing-computável de $\mathbb{N}$ a $\mathbb{N}$ é computada por alguma (na verdade, muitas) máquina de Turing. Renomeando seus estados e símbolos para inteiros positivos (em particular, $\triangleright$ como 1, 0 como 2 e 1 como 3), podemos ver que toda função Turing-computável é computada por uma máquina de Turing padrão. Isso significa que o conjunto de todas as funções Turing-computáveis de $\mathbb{N}$ a $\mathbb{N}$ é também enumerável.

[1] A terminologia "máquina padrão" não é padrão.

Por outro lado, o conjunto de todas as funções de $\mathbb{N}$ a $\mathbb{N}$ não é enumerável (**??**). Se todas as funções fossem computáveis por alguma máquina de Turing, poderíamos enumerar o conjunto de todas as funções listando todas as descrições de máquinas de Turing que as computam. Logo, há algumas funções que não são Turing-computáveis. □

Problema 2.1. Você consegue pensar em uma maneira de descrever máquinas de Turing que não exija que os estados e os símbolos do alfabeto sejam listados explicitamente? Você pode definir sua própria noção de máquina "padrão", mas diga algo sobre por que toda máquina de Turing pode ser computada por uma máquina "padrão" no seu novo sentido.

2.3 Máquinas de Turing Universais

tur:und:uni:sec

Na Seção 2.2 discutimos como toda máquina de Turing pode ser descrita por uma sequência finita de inteiros. Essa sequência codifica os estados, o alfabeto, o estado inicial e as instruções da máquina de Turing. Também observamos que o conjunto de todas essas descrições é enumerável. Como o conjunto de tais descrições é infinito enumerável, isso significa que há uma função sobrejetiva de $\mathbb{N}$ nessas descrições. Uma tal função sobrejetiva pode ser obtida, por exemplo, usando o método zigue-zague de Cantor. Ele nos dá uma maneira de enumerar todas as (descrições de) máquinas de Turing. Se fixarmos uma tal enumeração, passa a fazer sentido falar da primeira, da segunda, ..., e-ésima máquina de Turing. Esses números são chamados de *índices*.

Definição 2.2. Se M é a e-ésima máquina de Turing (em nossa enumeração fixada), dizemos que e é um *índice* de M. Escrevemos M_e para a e-ésima máquina de Turing.

Uma máquina pode ter mais de um índice; por exemplo, duas descrições de M podem diferir na ordem em que listamos suas instruções, e essas diferentes descrições terão índices diferentes.

É importante notar que é possível dar a enumeração das descrições de máquinas de Turing de tal modo que possamos computar efetivamente a descrição de M a partir de seu índice, e computar efetivamente um índice de uma máquina M a partir de sua descrição. Pela tese de Church-Turing, é então possível encontrar uma máquina de Turing que recupera a descrição da máquina de Turing com índice e e escreve a descrição correspondente em sua fita como saída. A descrição seria uma sequência de blocos de 1 (representando os inteiros positivos na sequência que descreve M_e).

Diante disso, torna-se natural perguntar: que funções de índices de máquinas de Turing são elas próprias computáveis por máquinas de Turing? Que propriedades de índices de máquinas de Turing podem ser decididas por máquinas de Turing? Um exemplo: a função que leva um índice e ao número de estados que a máquina de Turing com índice e possui é computável por uma máquina

de Turing. Eis o que tal máquina de Turing faria: iniciada em uma fita contendo um único bloco de e 1, ela primeiro decodificaria e em sua descrição. A descrição passa a ser representada por uma sequência de blocos de 1 na fita. Como o primeiro membro nessa sequência é o número de estados, tudo o que resta a fazer agora é apagar tudo, exceto o primeiro bloco de 1, e então parar.

Um resultado notável é o seguinte:

tur:und:uni: thm:universal-tm

Teorema 2.3. *Há uma* máquina de Turing universal U *que, quando iniciada na entrada* $\langle e, n\rangle$*,*

1. *para se, e somente se,* M_e *para na entrada* n*,* e
2. *se* M_e *para com saída* m*, então* U *também para.*

Assim, U *computa a função* $f : \mathbb{N}\times\mathbb{N} \nrightarrow \mathbb{N}$ *dada por* $f(e, n) = m$ *se* M_e *iniciada na entrada* n *para com saída* m*, e indefinida caso contrário.*

Prova. Produzir de fato U é basicamente impossível, já que se trata de uma máquina extremamente complicada. Mas podemos descrever em linhas gerais como ela funciona, e então invocar a tese de Church-Turing. Quando ela começa, a fita de U contém um bloco de e 1 seguido de um bloco de n 1. Ela primeiro "decodifica" o índice e à direita da entrada n. Isso produz uma lista de números (isto é, blocos de 1 separados por 0) que descreve as instruções da máquina M_e. Em seguida, U escreve o número do estado inicial de M_e e o número 1 na fita, à direita da descrição de M_e. (Novamente, esses são representados em unário, como blocos de 1.) Depois, ela copia a entrada (bloco de n 1) para a direita—mas substitui cada 1 por um bloco de três 1 (lembre-se, o número do símbolo 1 é 3, sendo 1 o número de ▷ e 2 o número de 0). Na extremidade esquerda dessa sequência de blocos (separados por símbolos 0 na fita de U), ela escreve um único 1, o código de ▷.

U agora tem em sua fita: o índice e, o número n, o número de código do estado inicial (o "estado atual"), o número da posição inicial do cabeçote 1 (a "posição atual do cabeçote") e o conteúdo inicial da "fita" (uma sequência de blocos de 1 representando os números de código dos símbolos de M_e—os "símbolos"—separados por 0).

Ela agora simula o que M_e faria se iniciada na entrada n, fazendo o seguinte:

1. Encontrar o número k da "posição atual do cabeçote" (no início, esse é 1),
2. Mover-se para o k-ésimo bloco na "fita" para ver qual é o "símbolo" ali,
3. Encontrar a instrução que corresponde ao "estado" e ao "símbolo" atuais, tur:und:uni: find-inst
4. Voltar para o k-ésimo bloco na "fita" e substituir o "símbolo" ali pelo número de código do símbolo que M_e escreveria,
5. Mover o cabeçote para onde ela registra o "estado" atual e substituir o número ali pelo número do novo estado,

6. Mover-se para o lugar onde ela registra a "posição na fita" e apagar um 1 ou adicionar um 1 (se a instrução mandar mover para a esquerda ou para a direita, respectivamente).

7. Repetir.[2]

Se M_e iniciada na entrada n nunca para, então U também nunca para, de modo que sua saída é indefinida.

Se no passo (3) acontecer de a descrição de M_e não conter nenhuma instrução para o par "estado"/"símbolo" atual, então M_e pararia. Se isso acontecer, U apaga a parte de sua fita à esquerda da "fita". Para cada bloco de três 1 (representando um 1 na fita de M_e), ela escreve um 1 na extremidade esquerda de sua própria fita, e apaga sucessivamente a "fita". Quando isso está concluído, a fita de U contém um único bloco de 1 de comprimento m.

Se U encontrar algo diferente de um bloco de três 1 na "fita", ela para imediatamente. Como a fita de U nesse caso não contém um único bloco de 1, sua saída não é um número natural, isto é, $f(e, n)$ é indefinida nesse caso. □

2.4 O Problema da Parada

explanation

tur:und:hal:sec

Suponha que tenhamos fixado alguma enumeração das descrições de máquinas de Turing. Cada máquina de Turing recebe assim um *índice*: seu lugar na enumeração M_1, M_2, M_3, ... das descrições de máquinas de Turing.

Sabemos que devem existir funções não-Turing-computáveis: o conjunto de descrições de máquinas de Turing—e, portanto, o conjunto de máquinas de Turing—é enumerável, mas o conjunto de todas as funções de $\mathbb{N}$ a $\mathbb{N}$ não é. Mas também podemos encontrar exemplos específicos de funções não computáveis. Uma dessas funções é a função de parada.

Definição 2.4 (Função de parada). A *função de parada* h é definida como

$$h(e, n) = \begin{cases} 0 & \text{se a máquina } M_e \text{ não para na entrada } n \\ 1 & \text{se a máquina } M_e \text{ para na entrada } n \end{cases}$$

Definição 2.5 (Problema da parada). O *Problema da Parada* é o problema de determinar (para quaisquer e, n) se a máquina de Turing M_e para com uma entrada de n traços.

explanation

Mostramos que h não é Turing-computável mostrando que uma função relacionada, s, não é Turing-computável. Esta prova baseia-se no fato de que tudo o que pode ser computado por uma máquina de Turing pode ser computado por uma máquina de Turing disciplinada (Seção 1.7), e no fato de que duas máquinas de Turing podem ser conectadas para criar uma única máquina (Seção 1.8).

[2] Estamos passando por cima de algumas dificuldades sutis aqui. Por exemplo, U pode precisar de algum espaço extra quando incrementa o contador onde mantém o registro da "posição atual do cabeçote"—nesse caso, terá de mover toda a "fita" para a direita.

Definição 2.6. A função s é definida como

$$s(e) = \begin{cases} 0 & \text{se a máquina } M_e \text{ não para na entrada } e \\ 1 & \text{se a máquina } M_e \text{ para na entrada } e \end{cases}$$

Lema 2.7. *A função s não é Turing-computável.*

Prova. Suponha, por contradição, que a função s seja Turing-computável. Então haveria uma máquina de Turing S que computa s. Podemos supor, sem perda de generalidade, que quando S para, ela o faz examinando o primeiro quadrado (isto é, que ela é disciplinada). Esta máquina pode ser "conectada" a outra máquina J, que para se for iniciada na entrada 0 (isto é, se ela lê 0 no estado inicial enquanto examina o quadrado à direita do símbolo de fim de fita), e caso contrário vagueia para a direita, sem nunca parar. $S \frown J$, a máquina criada conectando-se S a J, é uma máquina de Turing, logo é M_e para algum e (isto é, aparece em algum lugar da enumeração). Inicie M_e em uma entrada de e 1. Há duas possibilidades: ou M_e para ou não para.

1. Suponha que M_e pare para uma entrada de e 1. Então $s(e) = 1$. Assim, S, quando iniciada em e, para com um único 1 como saída na fita. Então J começa com um 1 na fita. Nesse caso, J não para. Mas M_e é a máquina $S \frown J$, de modo que ela deveria fazer exatamente o que S seguida de J faria (isto é, neste caso, vaguear para a direita e nunca parar). Logo, M_e não pode parar para uma entrada de e 1.

2. Agora suponha que M_e não pare para uma entrada de e 1. Então $s(e) = 0$, e S, quando iniciada na entrada e, para com uma fita em branco. J, quando iniciada em uma fita em branco, para imediatamente. Novamente, M_e faz o que S seguida de J faria, de modo que M_e deve parar para uma entrada de e 1.

Em cada caso chegamos a uma contradição com nossa suposição. Isso mostra que não pode haver uma máquina de Turing S: s não é Turing-computável. □

tur:und:hal: thm:halting-problem

Teorema 2.8 (Insolubilidade do Problema da Parada). *O problema da parada é insolúvel, isto é, a função h não é Turing-computável.*

Prova. Suponha que h fosse Turing-computável, digamos, por uma máquina de Turing H. Poderíamos usar H para construir uma máquina de Turing que computa s: primeiro, fazer uma cópia da entrada (separada por um símbolo 0). Depois, voltar ao início e executar H. Claramente podemos fazer uma máquina que realiza a primeira tarefa (veja Problema 1.13), e se H existisse, poderíamos "conectá-la" a tal máquina copiadora para obter uma nova máquina que determinaria se M_e para na entrada e, isto é, computa s. Mas já mostramos que não pode existir tal máquina. Portanto, h também não é Turing-computável. □

Problema 2.2. O problema da Parada com Três (3-Parada) é o problema de dar um procedimento de decisão para determinar se uma máquina de Turing escolhida arbitrariamente para ou não para uma entrada de três 1 em uma fita que, fora isso, está em branco. Prove que o problema 3-Parada é insolúvel.

Problema 2.3. Mostre que se o problema da parada é solúvel para pares de máquina de Turing e entrada M_e e n em que $e \neq n$, então ele também é solúvel para os casos em que $e = n$.

Problema 2.4. Provamos que o problema da parada é insolúvel se a entrada é um número e, que identifica uma máquina de Turing M_e por meio de uma enumeração de todas as máquinas de Turing. E se permitirmos a descrição de máquinas de Turing de Seção 2.2 diretamente como entrada? Pode haver uma máquina de Turing que decide o problema da parada, mas que recebe como entrada descrições de máquinas de Turing em vez de índices? Explique por que sim ou por que não.

Problema 2.5. Mostre que a função *parcial* s' definida como

$$s'(e) = \begin{cases} 1 & \text{se a máquina } M_e \text{ para na entrada } e \\ \text{indefinida} & \text{se a máquina } M_e \text{ não para na entrada } e \end{cases}$$

é Turing-computável.

2.5 O Problema da Decisão

tur:und:dec:sec

Dizemos que a lógica de primeira ordem é *decidível* se, e somente se, há um método efetivo para determinar se uma dada sentença é válida ou não. Acontece que não existe tal método: o problema de decidir a validade de sentenças de primeira ordem é insolúvel.

A fim de estabelecer esse importante resultado negativo, provamos que o problema da decisão não pode ser resolvido por uma máquina de Turing. Isto é, mostramos que não existe máquina de Turing que, sempre que iniciada em uma fita que contém uma sentença de primeira ordem, eventualmente para e produz 1 ou 0 conforme a sentença seja válida ou não. Pela tese de Church-Turing, toda função que é computável é Turing-computável. Logo, se essa "função de validade" fosse de algum modo efetivamente computável, ela seria Turing-computável. Se ela não é Turing-computável, então também não pode ser efetivamente computável.

Nossa estratégia para provar que o problema da decisão é insolúvel é reduzir o problema da parada a ele. Isso significa o seguinte: provamos que a função $h(e, w)$ que para com saída 1 se a máquina de Turing descrita por e para na entrada w e produz 0 caso contrário não é Turing-computável. Mostraremos que se houvesse uma máquina de Turing que decide a validade de sentenças de primeira ordem, então haveria também uma máquina de Turing

que computa h. Como h não pode ser computada por uma máquina de Turing, também não pode haver uma máquina de Turing que decide a validade.

O primeiro passo nessa estratégia é mostrar que, para toda entrada w e máquina de Turing M, podemos descrever efetivamente a sentença $\tau(M, w)$ representando o conjunto de instruções de M e a entrada w, e a sentença $\alpha(M, w)$ exprimindo "M eventualmente para", tal que:

$$\vDash \tau(M, w) \to \alpha(M, w) \text{ se, e somente se, } M \text{ para na entrada } w.$$

A maior parte de nossa prova consistirá em descrever essas sentenças $\tau(M, w)$ e $\alpha(M, w)$ e em verificar que $\tau(M, w) \to \alpha(M, w)$ é válida se, e somente se, M para na entrada w.

2.6 Representando Máquinas de Turing

tur:und:rep:sec

A fim de representar máquinas de Turing e seu comportamento por meio de a sentença da lógica de primeira ordem, temos de definir uma linguagem adequada. A linguagem consiste em duas partes: símbolos de predicado para descrever configurações da máquina, e expressões para numerar os passos de execução ("momentos") e as posições na fita. explanation

Introduzimos dois tipos de símbolos de predicado, ambos de 2 lugares: para cada estado q, a símbolo de predicado Q_q, e para cada símbolo de fita σ, a símbolo de predicado S_σ. Os primeiros permitem-nos descrever o estado de M e a posição de seu cabeçote de fita; os segundos permitem-nos descrever o conteúdo da fita.

A fim de exprimir as posições do cabeçote de fita e o número de passos executados, precisamos de uma maneira de exprimir números. Isso é feito usando a símbolo de constante o e uma função de 1 lugar $\prime$, a função sucessor. Por convenção, ela é escrita *depois* de seu argumento (e omitimos os parênteses).

Para cada número n há um termo canônico $\overline{n}$, o *numeral* de n, que o representa em $\mathcal{L}_M$. $\overline{0}$ é o, $\overline{1}$ é o', $\overline{2}$ é o'', e assim por diante. Mais formalmente:

$$\begin{aligned} \overline{0} &= \mathsf{o} \\ \overline{n+1} &= \overline{n}' \end{aligned}$$

O termo $\overline{0}$, isto é, o, nomeia a posição mais à esquerda na fita, bem como o instante anterior ao primeiro passo de execução (a configuração inicial). O termo $\overline{1}$, isto é, o', nomeia o quadrado à direita do quadrado mais à esquerda, e o instante posterior ao primeiro passo de execução, e assim por diante.

Também introduzimos a símbolo de predicado $<$ para exprimir tanto a ordenação das posições na fita (quando significa "à esquerda de") quanto a dos passos de execução (caso em que significa "antes").

Uma vez que tenhamos a linguagem montada, listamos os "axiomas" de $\tau(M, w)$, isto é, as sentenças que, tomadas em conjunto, descrevem o comportamento de M quando executada na entrada w. Haverá sentenças que estabelecem condições sobre o, $\prime$ e $<$, sentenças que descrevem a configuração de

entrada, e sentenças que descrevem qual é a configuração de M após executar uma instrução particular.

Definição 2.9. Dada uma máquina de Turing $M = \langle Q, \Sigma, q_0, \delta \rangle$, a linguagem $\mathcal{L}_M$ consiste em: tur:und:rep:defn:tm-descr

1. Um símbolo de predicado de dois lugares $Q_q(x, y)$ para todo estado $q \in Q$. Intuitivamente, $Q_q(\overline{m}, \overline{n})$ exprime "após n passos, M está no estado q examinando o m-ésimo quadrado."

2. Um símbolo de predicado de dois lugares $S_\sigma(x, y)$ para todo símbolo $\sigma \in \Sigma$. Intuitivamente, $S_\sigma(\overline{m}, \overline{n})$ exprime "após n passos, o m-ésimo quadrado contém o símbolo σ."

3. Um símbolo de constante o

4. Um símbolo de função de um lugar $\prime$

5. Um símbolo de predicado de dois lugares $<$

As sentenças que descrevem a operação da máquina de Turing M na entrada $w = \sigma_{i_1} \dots \sigma_{i_k}$ são as seguintes:

1. Axiomas que descrevem números e $<$:

   a) A sentença que diz que todo número é menor que seu sucessor:

   $$\forall x\, x < x'$$

   b) A sentença que garante que $<$ é transitivo:

   $$\forall x \forall y \forall z\, ((x < y \wedge y < z) \to x < z)$$

2. Axiomas que descrevem a configuração de entrada:

   a) Após 0 passos—antes de a máquina começar—M está no estado inicial q_0, examinando o quadrado 1:

   $$Q_{q_0}(\overline{1}, \overline{0})$$

   b) Os primeiros $k + 1$ quadrados contêm os símbolos $\triangleright$, σ_{i_1}, $\dots$, σ_{i_k}:

   $$S_\triangleright(\overline{0}, \overline{0}) \wedge S_{\sigma_{i_1}}(\overline{1}, \overline{0}) \wedge \dots \wedge S_{\sigma_{i_k}}(\overline{k}, \overline{0})$$

   c) Caso contrário, a fita está vazia:

   $$\forall x\, (\overline{k} < x \to S_0(x, \overline{0}))$$

3. Axiomas que descrevem a transição de uma configuração para a seguinte:

   Para o que segue, seja $\varphi(x, y)$ a conjunção de todas as sentenças da forma

$$\forall z\,(((z < x \vee x < z) \wedge S_\sigma(z, y)) \to S_\sigma(z, y'))$$

   em que $\sigma \in \Sigma$. Usamos $\varphi(\overline{m}, \overline{n})$ para exprimir "exceto no quadrado m, a fita após $n + 1$ passos é a mesma que após n passos."

tur:und:rep: rep-right

   a) Para toda instrução $\delta(q_i, \sigma) = \langle q_j, \sigma', R\rangle$, a sentença:

$$\begin{aligned}\forall x\,\forall y\,((&Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \to \\ &(Q_{q_j}(x', y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y)))\end{aligned}$$

   Isto diz que se, após y passos, a máquina está no estado q_i examinando o quadrado x que contém o símbolo σ, então após $y + 1$ passos ela está examinando o quadrado $x + 1$, está no estado q_j, o quadrado x agora contém σ', e todo quadrado distinto de x contém o mesmo símbolo que continha após y passos.

tur:und:rep: rep-left

   b) Para toda instrução $\delta(q_i, \sigma) = \langle q_j, \sigma', L\rangle$, a sentença:

$$\begin{aligned}\forall x\,\forall y\,((&Q_{q_i}(x', y) \wedge S_\sigma(x', y)) \to \\ &(Q_{q_j}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\ \forall y\,((&Q_{q_i}(\overline{0}, y) \wedge S_\sigma(\overline{0}, y)) \to \\ &(Q_{q_j}(\overline{0}, y') \wedge S_{\sigma'}(\overline{0}, y') \wedge \varphi(\overline{0}, y)))\end{aligned}$$

   Reserve um momento para pensar em como isso funciona: agora não começamos com "se examinando o quadrado x ...", mas: "se examinando o quadrado $x+1$ ...". Um movimento para a esquerda significa que no próximo passo a máquina está examinando o quadrado x. Mas o quadrado em que se escreve é $x+1$. Fazemos assim porque não temos subtração nem uma função predecessor.

   Note que os números da forma $x + 1$ são 1, 2, ..., isto é, isso não cobre o caso em que a máquina está examinando o quadrado 0 e deveria mover-se para a esquerda (o que, é claro, não pode fazer—ela simplesmente permanece onde está). Esse caso especial é coberto pela segunda conjunção: ela diz que se, após y passos, a máquina está examinando o quadrado 0 no estado q_i e o quadrado 0 contém o símbolo σ, então após $y + 1$ passos ela ainda está examinando o quadrado 0, está agora no estado q_j, o símbolo no quadrado 0 é σ', e os quadrados distintos do quadrado 0 contêm os mesmos símbolos que continham após y passos.

tur:und:rep: rep-stay

   c) Para toda instrução $\delta(q_i, \sigma) = \langle q_j, \sigma', N\rangle$, a sentença:

$$\begin{aligned}\forall x\,\forall y\,((&Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \to \\ &(Q_{q_j}(x, y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y)))\end{aligned}$$

Seja $\tau(M, w)$ a conjunção de todas as sentenças acima para a máquina de Turing M e a entrada w.

A fim de exprimir que M eventualmente para, temos de encontrar uma sentença que diga "após algum número de passos, a função de transição será indefinida." Seja X o conjunto de todos os pares $\langle q, \sigma \rangle$ tais que $\delta(q, \sigma)$ é indefinida. Seja então $\alpha(M, w)$ a sentença

$$\exists x \, \exists y \, (\bigvee_{\langle q,\sigma \rangle \in X} (Q_q(x, y) \wedge S_\sigma(x, y)))$$

Se usarmos uma máquina de Turing com um estado de parada designado h, é ainda mais fácil: então a sentença $\alpha(M, w)$

$$\exists x \, \exists y \, Q_h(x, y)$$

exprime que a máquina eventualmente para.

Proposição 2.10. *Se $m < k$, então $\tau(M, w) \vDash \overline{m} < \overline{k}$* *tur:und:rep: prop:mlessk*

Prova. Exercício. □

Problema 2.6. Prove Proposição 2.10. (Dica: use indução em $k - m$).

2.7 Verificando a Representação

explanation A fim de verificar que nossa representação funciona, temos de provar duas coisas. Primeiro, temos de mostrar que se M para na entrada w, então $\tau(M, w) \to \alpha(M, w)$ é válida. Em seguida, temos de mostrar a recíproca, isto é, que se $\tau(M, w) \to \alpha(M, w)$ é válida, então M de fato eventualmente para quando executada na entrada w. *tur:und:ver: sec*

A estratégia para prová-las é bem diferente. Para o primeiro resultado, temos de mostrar que a sentença da lógica de primeira ordem (a saber, $\tau(M, w) \to \alpha(M, w)$) é válida. A maneira mais fácil de fazer isso é dar a derivação. Nossa prova, porém, deve funcionar para todos os M e w, de modo que não há realmente uma única sentença para a qual tenhamos de dar a derivação, mas infinitas. Assim, o melhor que podemos fazer é provar por indução que, sejam quais forem M e w, e por mais passos que M leve para parar na entrada w, haverá a derivação de $\tau(M, w) \to \alpha(M, w)$.

Naturalmente, nossa indução procederá sobre o número de passos que M leva antes de atingir uma configuração de parada. Em nossa prova indutiva, estabeleceremos que para cada passo n da execução de M na entrada w, $\tau(M, w) \vDash \chi(M, w, n)$, em que $\chi(M, w, n)$ descreve corretamente a configuração de M executada em w após n passos. Ora, se M para na entrada w após, digamos, n passos, $\chi(M, w, n)$ descreverá uma configuração de parada. Mostraremos também que $\chi(M, w, n) \vDash \alpha(M, w)$ sempre que $\chi(M, w, n)$ descreve uma configuração de parada. Assim, se M para na entrada w, então para algum n, M estará em uma configuração de parada após n passos. Logo,

$\tau(M,w) \vDash \chi(M,w,n)$ em que $\chi(M,w,n)$ descreve uma configuração de parada, e como nesse caso $\chi(M,w,n) \vDash \alpha(M,w)$, obtemos que $T(M,w) \vDash \alpha(M,w)$, isto é, que $\vDash \tau(M,w) \to \alpha(M,w)$.

A estratégia para a recíproca é bem diferente. Aqui supomos que $\vDash \tau(M,w) \to \alpha(M,w)$ e temos de provar que M para na entrada w. Da hipótese obtemos que $\tau(M,w) \vDash \alpha(M,w)$, isto é, $\alpha(M,w)$ é verdadeira em toda estrutura em que $\tau(M,w)$ é verdadeira. Assim, descreveremos a estrutura $\mathfrak{M}$ em que $\tau(M,w)$ é verdadeira: seu domínio será $\mathbb{N}$, e a interpretação de todos os Q_q e S_σ será dada pelas configurações de M durante uma execução na entrada w. Assim, por exemplo, $\mathfrak{M} \vDash Q_q(\overline{m},\overline{n})$ se, e somente se, T, quando executada na entrada w por n passos, está no estado q e examinando o quadrado m. Ora, como $\tau(M,w) \vDash \alpha(M,w)$ por hipótese, e como $\mathfrak{M} \vDash \tau(M,w)$ por construção, $\mathfrak{M} \vDash \alpha(M,w)$. Mas $\mathfrak{M} \vDash \alpha(M,w)$ se, e somente se, há algum $n \in |\mathfrak{M}| = \mathbb{N}$ tal que M, executada na entrada w, está em uma configuração de parada após n passos.

Definição 2.11. Seja $\chi(M,w,n)$ a sentença

$$Q_q(\overline{m},\overline{n}) \wedge S_{\sigma_0}(\overline{0},\overline{n}) \wedge \cdots \wedge S_{\sigma_k}(\overline{k},\overline{n}) \wedge \forall x\,(\overline{k} < x \to S_0(x,\overline{n}))$$

em que q é o estado de M no instante n, M está examinando o quadrado m no instante n, o quadrado i contém o símbolo σ_i no instante n para $0 \leq i \leq k$, e k é o quadrado não-vazio mais à direita da fita no instante 0, ou o quadrado mais à direita que o cabeçote de fita visitou após n passos, o que for maior.

tur:und:ver: lem:halt-config-implies-halt

Lema 2.12. *Se M executada na entrada w está em uma configuração de parada após n passos, então $\chi(M,w,n) \vDash \alpha(M,w)$.*

Prova. Suponha que M pare para a entrada w após n passos. Há algum estado q, quadrado m e símbolo σ tais que:

1. Após n passos, M está no estado q examinando o quadrado m no qual aparece σ.
2. A função de transição $\delta(q,\sigma)$ é indefinida.

$\chi(M,w,n)$ é a descrição dessa configuração e incluirá as cláusulas $Q_q(\overline{m},\overline{n})$ e $S_\sigma(\overline{m},\overline{n})$. Essas cláusulas juntas implicam $\alpha(M,w)$:

$$\exists x\,\exists y\,(\bigvee_{\langle q,\sigma\rangle \in X} (Q_q(x,y) \wedge S_\sigma(x,y)))$$

já que $Q_{q'}(\overline{m},\overline{n}) \wedge S_{\sigma'}(\overline{m},\overline{n}) \vDash \bigvee_{\langle q,\sigma\rangle \in X}(Q_q(\overline{m},\overline{n}) \wedge S_\sigma(\overline{m},\overline{n}))$, pois $\langle q',\sigma'\rangle \in X$. □

Assim, se M para na entrada w, então há algum n tal que $\chi(M,w,n) \vDash \alpha(M,w)$. Mostraremos agora que para qualquer instante n, $\tau(M,w) \vDash \chi(M,w,n)$. *explanation*

Lema 2.13. *Para cada n, se M não parou após n passos, $\tau(M,w) \vDash \chi(M,w,n)$.* tur:und:ver:lem:config

Prova. Base da indução: Se $n = 0$, então as fórmulas componentes de $\chi(M,w,0)$ são também fórmulas componentes de $\tau(M,w)$, logo implicadas por ela.

Hipótese de indução: Se M não parou antes do n-ésimo passo, então $\tau(M,w) \vDash \chi(M,w,n)$. Temos de mostrar que (a menos que $\chi(M,w,n)$ descreva uma configuração de parada), $\tau(M,w) \vDash \chi(M,w,n+1)$.

Suponha $n > 0$ e que, após n passos, M iniciada em w esteja no estado q examinando o quadrado m. Como M não para após n passos, deve haver no programa de M uma instrução de uma das três formas seguintes:

1. $\delta(q,\sigma) = \langle q', \sigma', R\rangle$ tur:und:ver:right
2. $\delta(q,\sigma) = \langle q', \sigma', L\rangle$ tur:und:ver:left
3. $\delta(q,\sigma) = \langle q', \sigma', N\rangle$ tur:und:ver:stay

Consideraremos cada um desses três casos por sua vez.

1. Suponha que haja uma instrução da forma (1). Por Definição 2.9(3a), isso significa que

$$\begin{aligned}\forall x\, \forall y\, ((Q_q(x,y) \wedge S_\sigma(x,y)) \rightarrow \\ (Q_{q'}(x',y') \wedge S_{\sigma'}(x,y') \wedge \varphi(x,y)))\end{aligned}$$

é uma fórmula componente de $\tau(M,w)$. Isso implica a seguinte sentença (instanciação universal, $\overline{m}$ por x e $\overline{n}$ por y):

$$\begin{aligned}(Q_q(\overline{m},\overline{n}) \wedge S_\sigma(\overline{m},\overline{n})) \rightarrow \\ (Q_{q'}(\overline{m}',\overline{n}') \wedge S_{\sigma'}(\overline{m},\overline{n}') \wedge \varphi(\overline{m},\overline{n})).\end{aligned}$$

Por hipótese de indução, $\tau(M,w) \vDash \chi(M,w,n)$, isto é,

$$\begin{aligned}Q_q(\overline{m},\overline{n}) \wedge S_{\sigma_0}(\overline{0},\overline{n}) \wedge \cdots \wedge S_{\sigma_k}(\overline{k},\overline{n}) \wedge \\ \forall x\, (\overline{k} < x \rightarrow S_0(x,\overline{n}))\end{aligned}$$

Como após n passos o quadrado de fita m contém σ, a fórmula componente correspondente é $S_\sigma(\overline{m},\overline{n})$, de modo que isso implica:

$$Q_q(\overline{m},\overline{n}) \wedge S_\sigma(\overline{m},\overline{n})$$

Obtemos agora

$$\begin{aligned}Q_{q'}(\overline{m}',\overline{n}') \wedge S_{\sigma'}(\overline{m},\overline{n}') \wedge \\ S_{\sigma_0}(\overline{0},\overline{n}') \wedge \cdots \wedge S_{\sigma_k}(\overline{k},\overline{n}') \wedge \\ \forall x\, (\overline{k} < x \rightarrow S_0(x,\overline{n}'))\end{aligned}$$

como segue: A primeira linha vem diretamente do consequente do condicional precedente, por modus ponens. Cada fórmula componente na linha do meio—que exclui $S_{\sigma_m}(\overline{m}, \overline{n}')$—decorre da fórmula componente correspondente em $\chi(M, w, n)$ junto com $\varphi(\overline{m}, \overline{n})$.

Se $m < k$, $\tau(M, w) \vdash \overline{m} < \overline{k}$ (Proposição 2.10) e, por transitividade de $<$, temos $\forall x\,(\overline{k} < x \to \overline{m} < x)$. Se $m = k$, então $\forall x\,(\overline{k} < x \to \overline{m} < x)$ apenas por lógica. A última linha decorre então da fórmula componente correspondente em $\chi(M, w, n)$, de $\forall x\,(\overline{k} < x \to \overline{m} < x)$ e de $\varphi(\overline{m}, \overline{n})$. Se $m < k$, isto já é $\chi(M, w, n + 1)$.

Agora suponha $m = k$. Nesse caso, após $n + 1$ passos, o cabeçote de fita também visitou o quadrado $k + 1$, que agora é o quadrado mais à direita visitado. Assim, $\chi(M, w, n + 1)$ tem uma nova fórmula componente, $S_0(\overline{k}', \overline{n}')$, e a última fórmula componente é $\forall x\,(\overline{k}' < x \to S_0(x, \overline{n}'))$. Temos de verificar que essas duas sentenças também são implicadas.

Já temos $\forall x\,(\overline{k} < x \to S_0(x, \overline{n}'))$. Em particular, isso nos dá $\overline{k} < \overline{k}' \to S_0(\overline{k}', \overline{n}')$. Do axioma $\forall x\, x < x'$ obtemos $\overline{k} < \overline{k}'$. Por modus ponens, segue-se $S_0(\overline{k}', \overline{n}')$.

Além disso, como $\tau(M, w) \vdash \overline{k} < \overline{k}'$, o axioma da transitividade de $<$ nos dá $\forall x\,(\overline{k}' < x \to S_0(x, \overline{n}'))$. (Deixamos a verificação disso como exercício.)

2. Suponha que haja uma instrução da forma (2). Então, por Definição 2.9(3b),

$$\begin{aligned}
&\forall x\,\forall y\,((Q_q(x', y) \wedge S_\sigma(x', y)) \to \\
&\qquad (Q_{q'}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\
&\forall y\,((Q_{q_i}(\overline{0}, y) \wedge S_\sigma(\overline{0}, y)) \to \\
&\qquad (Q_{q_j}(\overline{0}, y') \wedge S_{\sigma'}(\overline{0}, y') \wedge \varphi(\overline{0}, y)))
\end{aligned}$$

é uma fórmula componente de $\tau(M, w)$. Se $m > 0$, então seja $l = m - 1$ (isto é, $m = l + 1$). A primeira fórmula componente da sentença acima implica o seguinte:

$$\begin{aligned}
&(Q_q(\overline{l}', \overline{n}) \wedge S_\sigma(\overline{l}', \overline{n})) \to \\
&\qquad (Q_{q'}(\overline{l}, \overline{n}') \wedge S_{\sigma'}(\overline{l}', \overline{n}') \wedge \varphi(\overline{l}, \overline{n}))
\end{aligned}$$

Caso contrário, seja $l = m = 0$ e considere a seguinte sentença implicada pela segunda fórmula componente:

$$\begin{aligned}
&((Q_{q_i}(\overline{0}, \overline{n}) \wedge S_\sigma(\overline{0}, \overline{n})) \to \\
&\qquad (Q_{q_j}(\overline{0}, \overline{n}') \wedge S_{\sigma'}(\overline{0}, \overline{n}') \wedge \varphi(\overline{0}, \overline{n})))
\end{aligned}$$

Qualquer das sentenças implica

$$Q_{q'}(\overline{l}, \overline{n}') \wedge S_{\sigma'}(\overline{m}, \overline{n}') \wedge$$
$$S_{\sigma_0}(\overline{0}, \overline{n}') \wedge \cdots \wedge S_{\sigma_k}(\overline{k}, \overline{n}') \wedge$$
$$\forall x\, (\overline{k} < x \to S_0(x, \overline{n}'))$$

como antes. (Note que no primeiro caso, $\overline{l}' \equiv \overline{l+1} \equiv \overline{m}$ e no segundo caso $\overline{l} \equiv \overline{0}$.) Mas isto é justamente $\chi(M, w, n+1)$.

3. O caso (3) é deixado como exercício.

Mostramos que para qualquer n, $\tau(M, w) \vDash \chi(M, w, n)$. □

Problema 2.7. Complete o caso (3) da prova de Lema 2.13.

Problema 2.8. Dê a derivação de $S_{\sigma_i}(\overline{i}, \overline{n}')$ a partir de $S_{\sigma_i}(\overline{i}, \overline{n})$ e $\varphi(m, n)$ (supondo $i \neq m$, isto é, $i < m$ ou $m < i$).

Problema 2.9. Dê a derivação de $\forall x\, (\overline{k}' < x \to S_0(x, \overline{n}'))$ a partir de $\forall x\, (\overline{k} < x \to S_0(x, \overline{n}'))$, $\forall x\, x < x'$ e $\forall x \forall y \forall z\, ((x < y \wedge y < z) \to x < z)$.)

Lema 2.14. *Se M para na entrada w, então $\tau(M, w) \to \alpha(M, w)$ é válida.* tur:und:ver: lem:valid-if-halt

Prova. Por Lema 2.13, sabemos que, para qualquer instante n, a descrição $\chi(M, w, n)$ da configuração de M no instante n é implicada por $\tau(M, w)$. Suponha que M pare após k passos. Nesse ponto, ela estará examinando o quadrado m, para algum $m \in \mathbb{N}$. Então $\chi(M, w, k)$ descreve uma configuração de parada de M, isto é, contém como fórmulas componentes tanto $Q_q(\overline{m}, \overline{k})$ quanto $S_\sigma(\overline{m}, \overline{k})$ com $\delta(q, \sigma)$ indefinida. Assim, por Lema 2.12, $\chi(M, w, k) \vDash \alpha(M, w)$. Mas como $\tau(M, w) \vDash \chi(M, w, k)$, temos $\tau(M, w) \vDash \alpha(M, w)$ e portanto $\tau(M, w) \to \alpha(M, w)$ é válida. □

explanation Para completar a verificação de nossa afirmação, temos também de estabelecer a direção reversa: se $\tau(M, w) \to \alpha(M, w)$ é válida, então M de fato para quando iniciada na entrada w.

Lema 2.15. *Se $\vDash \tau(M, w) \to \alpha(M, w)$, então M para na entrada w.* tur:und:ver: lem:halt-if-valid

Prova. Considere a $\mathcal{L}_M$-estrutura $\mathfrak{M}$ com domínio $\mathbb{N}$ que interpreta o como 0, $'$ como a função sucessor e $<$ como a relação menor-que, e os predicados Q_q e S_σ como segue:

$$Q_q^{\mathfrak{M}} = \{\langle m, n\rangle : \begin{array}{l}\text{iniciada em } w, \text{ após } n \text{ passos,} \\ M \text{ está no estado } q \text{ examinando o quadrado } m\end{array}\}$$

$$S_\sigma^{\mathfrak{M}} = \{\langle m, n\rangle : \begin{array}{l}\text{iniciada em } w, \text{ após } n \text{ passos,} \\ \text{o quadrado } m \text{ de } M \text{ contém o símbolo } \sigma\end{array}\}$$

Em outras palavras, construímos a estrutura $\mathfrak{M}$ de modo que ela descreva o que M iniciada na entrada w de fato faz, passo a passo. Claramente, $\mathfrak{M} \vDash \tau(M, w)$. Se $\vDash \tau(M, w) \to \alpha(M, w)$, então também $\mathfrak{M} \vDash \alpha(M, w)$, isto é,

$$\mathfrak{M} \vDash \exists x \, \exists y \, (\bigvee_{\langle q, \sigma \rangle \in X} (Q_q(x, y) \wedge S_\sigma(x, y))).$$

Como $|\mathfrak{M}| = \mathbb{N}$, deve haver m, $n \in \mathbb{N}$ tais que $\mathfrak{M} \vDash Q_q(\overline{m}, \overline{n}) \wedge S_\sigma(\overline{m}, \overline{n})$ para algum q e σ tais que $\delta(q, \sigma)$ é indefinida. Pela definição de $\mathfrak{M}$, isso significa que M iniciada na entrada w após n passos está no estado q e lendo o símbolo σ, e a função de transição é indefinida, isto é, M parou. □

2.8 O Problema da Decisão é Insolúvel

tur:und:uns:sec
tur:und:uns:thm:decision-prob

Teorema 2.16. *O problema da decisão é insolúvel: Não há máquina de Turing D tal que, quando iniciada em uma fita que contém como entrada a sentença ψ da lógica de primeira ordem, D eventualmente para e produz 1 se, e somente se, ψ é válida, e 0 caso contrário.*

Prova. Suponha que o problema da decisão fosse solúvel, isto é, suponha que houvesse uma máquina de Turing D. Então poderíamos resolver o problema da parada como segue. Construímos uma máquina de Turing E que, dado como entrada o número e da máquina de Turing M_e e a entrada w, computa a sentença correspondente $\tau(M_e, w) \to \alpha(M_e, w)$ e para, examinando o quadrado mais à esquerda na fita. A máquina $E \frown D$ então, dada a entrada e e w, primeiro computaria $\tau(M_e, w) \to \alpha(M_e, w)$ e depois executaria a máquina do problema da decisão D nessa entrada. D para com saída 1 se, e somente se, $\tau(M_e, w) \to \alpha(M_e, w)$ é válida, e produz 0 caso contrário. Por Lema 2.15 e Lema 2.14, $\tau(M_e, w) \to \alpha(M_e, w)$ é válida se, e somente se, M_e para na entrada w. Assim, $E \frown D$, dada a entrada e e w, para com saída 1 se, e somente se, M_e para na entrada w, e para com saída 0 caso contrário. Em outras palavras, $E \frown D$ resolveria o problema da parada. Mas sabemos, por Teorema 2.8, que não pode existir tal máquina de Turing. □

tur:und:uns:cor:undecidable-sat

Corolário 2.17. *É indecidível se uma sentença arbitrária da lógica de primeira ordem é satisfatível.*

Prova. Suponha que a satisfatibilidade fosse decidível por uma máquina de Turing S. Então poderíamos resolver o problema da decisão como segue: Dada como entrada a sentença B, mova ψ um quadrado para a direita. Volte ao quadrado 1 e escreva o símbolo $\neg$.

Agora execute a máquina de Turing S. Ela eventualmente para com saída 1 (se $\neg\psi$ é satisfatível) ou 0 (se $\neg\psi$ é insatisfatível) na fita. Se há um 1 no quadrado 1, apague-o; se o quadrado 1 está vazio, escreva um 1 e então pare.

Essa máquina de Turing sempre para, e sua saída é 1 se, e somente se, $\neg\psi$ é insatisfatível, e 0 caso contrário. Como ψ é válida se, e somente se,

$\neg\psi$ é insatisfatível, a máquina produz 1 se, e somente se, ψ é válida, e 0 caso contrário, isto é, ela resolveria o problema da decisão. □

explanation

Portanto, não há máquina de Turing que sempre dê uma resposta "sim" ou "não" correta à pergunta "ψ é uma sentença válida da lógica de primeira ordem?" No entanto, *há* uma máquina de Turing que sempre dá uma resposta "sim" correta—mas que simplesmente não para se a resposta é "não". Isso decorre do teorema da correção e completude da lógica de primeira ordem, e do fato de que derivações podem ser efetivamente enumeradas.

Teorema 2.18. *A validade de sentenças de primeira ordem é semidecidível: Há uma máquina de Turing E que, quando iniciada em uma fita que contém como entrada a sentença ψ da lógica de primeira ordem, E eventualmente para e produz 1 se, e somente se, ψ é válida, mas não para caso contrário.* tur:und:uns: thm:valid-ce

Prova. Todas as derivações possíveis da lógica de primeira ordem podem ser geradas, uma após a outra, por um algoritmo efetivo. A máquina E faz isso e, quando encontra a derivação que mostra que $\vdash \psi$, ela para com saída 1. Pelo teorema da correção, se E para com saída 1, é porque $\vDash \psi$. Pelo teorema da completude, se $\vDash \psi$, há a derivação que mostra que $\vdash \psi$. Como E gera sistematicamente todas as derivações possíveis, ela eventualmente encontrará uma que mostra $\vdash \psi$, de modo que eventualmente parará com saída 1. □

2.9 Teorema de Trakhtenbrot

explanation

Na Seção 2.6 definimos as sentenças $\tau(M, w)$ e $\alpha(M, w)$ para uma máquina de Turing M e uma cadeia de entrada w. Em seguida, mostramos em Lema 2.14 e Lema 2.15 que $\tau(M, w) \to \alpha(M, w)$ é válida se, e somente se, M, iniciada na entrada w, eventualmente para. Como o Problema da Parada é indecidível, isso implica que a validade e a satisfatibilidade de sentenças da lógica de primeira ordem são indecidíveis (Teorema 2.16 e Corolário 2.17). tur:und:tra: sec

Mas a validade e a satisfatibilidade de sentenças são definidas para estruturas arbitrárias, finitas ou infinitas. Você poderia suspeitar que é mais fácil decidir se a sentença é satisfatível em uma estrutura finita (ou válida em todas as estruturas finitas). Podemos adaptar a prova da insolubilidade do problema da decisão de modo que ela mostre que não é esse o caso.

Primeiro, se você voltar à prova de Lema 2.15, verá que o que fizemos ali foi produzir um modelo $\mathfrak{M}$ de $\tau(M, w)$ que descreve exatamente o que a máquina M faz quando iniciada na entrada w. O domínio daquele modelo era $\mathbb{N}$, isto é, infinito. Mas se M de fato para na entrada w, podemos construir um modelo finito $\mathfrak{M}'$ da mesma maneira. Suponha que M iniciada na entrada w pare após k passos. Tome como domínio $|\mathfrak{M}'|$ o conjunto $\{0, \ldots, n\}$, em que n é o maior entre k e o comprimento de w, e seja

$$\prime^{\mathfrak{M}'}(x) = \begin{cases} x+1 & \text{se } x < n \\ n & \text{caso contrário,} \end{cases}$$

e $\langle x, y\rangle \in <^{\mathfrak{M}'}$ se, e somente se, $x < y$ ou $x = y = n$. Quanto ao resto, $\mathfrak{M}'$ é definida tal como $\mathfrak{M}$. Pela definição de $\mathfrak{M}'$, tal como na prova de Lema 2.15, $\mathfrak{M}' \vDash \tau(M, w)$. E como supusemos que M para na entrada w, $\mathfrak{M}' \vDash \alpha(M, w)$. Assim, $\mathfrak{M}'$ é um modelo finito de $\tau(M, w) \wedge \alpha(M, w)$ (note que substituímos $\to$ por $\wedge$).

Estamos a meio caminho de uma prova: mostramos que se M para na entrada w, então $\tau(M, w) \wedge \alpha(M, w)$ tem um modelo finito. Infelizmente, a recíproca disso não vale, isto é, há máquinas de Turing que não param em alguma entrada w, mas $\tau(M, w) \wedge \alpha(M, w)$ ainda assim tem um modelo finito. Por exemplo, considere a máquina M com o único estado q_0 e a instrução $\delta(q_0, 0) = \langle q_0, 0, N\rangle$. Iniciada na entrada vazia $w = \Lambda$, essa máquina nunca para: está em um laço infinito, mas não altera a fita nem move o cabeçote. Todas as configurações são as mesmas (mesmo estado, mesma posição do cabeçote, mesmo conteúdo da fita). Podemos definir uma estrutura finita $\mathfrak{M}''$ que satisfaz $\tau(M, \Lambda) \wedge \alpha(M, \Lambda)$ (exercício). Podemos, no entanto, modificar $\tau(M, w)$ de modo adequado para que tais estruturas sejam descartadas.

Problema 2.10. Seja M uma máquina de Turing com o único estado q_0 e a única instrução $\delta(q_0, 0) = \langle q, 0, N\rangle$. Seja $|\mathfrak{M}''| = \{0, 1, 2\}$, $\imath^{\mathfrak{M}''}(0) = \imath^{\mathfrak{M}'}(1) = 1$ e $\imath^{\mathfrak{M}''}(2) = 2$, e $<^{\mathfrak{M}''} = \{\langle 0, 1\rangle, \langle 1, 1\rangle, \langle 2, 2\rangle\}$. Defina $Q_{q_0}^{\mathfrak{M}''}$, $S_0^{\mathfrak{M}''}$ e $S_\triangleright^{\mathfrak{M}''}$ de modo que $\tau(M, \Lambda)$ e $\alpha(M, \Lambda)$ se tornem verdadeiras e explique por que o são. Dica: Observe que $\delta(q_0, \triangleright)$ é indefinida. Garanta que

$$Q_{q_0}(\overline{1}, \overline{n}) \wedge S_\triangleright(\overline{0}, \overline{n}) \wedge \forall x\, (\overline{0} < x \to S_0(x, \overline{n})) \quad \text{para todo } n \in \mathbb{N}$$
$$\exists y\, (Q_{q_0}(\overline{0}, y) \wedge S_\triangleright(\overline{0}, y))$$

são ambas verdadeiras em $\mathfrak{M}''$.

Considere as sentenças que descrevem a operação da máquina de Turing M na entrada $w = \sigma_{i_1} \dots \sigma_{i_k}$:

1. Axiomas que descrevem números e $<$ (tal como na definição de $\tau(M, w)$ na Seção 2.6).

2. Axiomas que descrevem a configuração de entrada: tal como na definição de $\tau(M, w)$.

3. Axiomas que descrevem a transição de uma configuração para a seguinte:

   Para o que segue, seja $\varphi(x, y)$ como antes, e seja

   $$\psi(y) \equiv \forall x\, (x < y \to x \neq y).$$

tur:und:tra: rep-right

   a) Para toda instrução $\delta(q_i, \sigma) = \langle q_j, \sigma', R\rangle$, a sentença:

   $$\forall x \forall y\, ((Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \to (Q_{q_j}(x', y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y) \wedge \psi(y')))$$

b) Para toda instrução $\delta(q_i, \sigma) = \langle q_j, \sigma', L\rangle$, a sentença

tur:und:tra: rep-left

$$\begin{aligned}
&\forall x\, \forall y\, ((Q_{q_i}(x', y) \wedge S_\sigma(x', y)) \to \\
&\qquad (Q_{q_j}(x, y') \wedge S_{\sigma'}(x', y') \wedge \varphi(x, y))) \wedge \\
&\forall y\, ((Q_{q_i}(\overline{0}, y) \wedge S_\sigma(\overline{0}, y)) \to \\
&\qquad (Q_{q_j}(\overline{0}, y') \wedge S_{\sigma'}(\overline{0}, y') \wedge \varphi(\overline{0}, y) \wedge \psi(y')))
\end{aligned}$$

c) Para toda instrução $\delta(q_i, \sigma) = \langle q_j, \sigma', N\rangle$, a sentença:

tur:und:tra: rep-stay

$$\begin{aligned}
&\forall x\, \forall y\, ((Q_{q_i}(x, y) \wedge S_\sigma(x, y)) \to \\
&\qquad (Q_{q_j}(x, y') \wedge S_{\sigma'}(x, y') \wedge \varphi(x, y) \wedge \psi(y')))
\end{aligned}$$

Como você pode ver, as sentenças que descrevem as transições de M são as mesmas que a sentença correspondente em $\tau(M, w)$, exceto que adicionamos $\psi(y')$ ao final. $\psi(y')$ garante que o número y' da "próxima" configuração é diferente de todos os números anteriores o, o', ...

Seja $\tau'(M, w)$ a conjunção de todas as sentenças acima para a máquina de Turing M e a entrada w.

Lema 2.19. *Se M iniciada na entrada w para, então $\tau'(M, w) \wedge \alpha(M, w)$ tem um modelo finito.*

tur:und:tra: lem:halts-sat

Prova. Seja $\mathfrak{M}'$ como na prova de Lema 2.15, exceto que

$$\begin{aligned}
|\mathfrak{M}'| &= \{0, \ldots, n\}, \\
\prime^{\mathfrak{M}'}(x) &= \begin{cases} x+1 & \text{se } x < n \\ n & \text{caso contrário,} \end{cases} \\
\langle x, y\rangle \in <^{\mathfrak{M}'} &\text{ sse } x < y \text{ ou } x = y = n,
\end{aligned}$$

em que $n = \max(k, \mathrm{len}(w))$ e k é o menor número tal que M iniciada na entrada w parou após k passos. Deixamos como exercício a verificação de que $\mathfrak{M}' \vDash \tau'(M, w) \wedge E(M, w)$. □

Problema 2.11. Complete a prova de Lema 2.19 provando que $\mathfrak{M}' \vDash \tau(M, w) \wedge E(M, w)$.

Lema 2.20. *Se $\tau'(M, w) \wedge \alpha(M, w)$ tem um modelo finito, então M iniciada na entrada w para.*

tur:und:tra: lem:sat-halts

Prova. Mostramos a contrapositiva. Suponha que M iniciada em w não pare. Se $\tau'(M, w) \wedge \alpha(M, w)$ não tem modelo algum, terminamos. Então suponha que $\mathfrak{M}$ seja um modelo de $\tau(M, w) \wedge \alpha(M, w)$. Temos de mostrar que ele não pode ser finito.

Podemos provar, tal como em Lema 2.13, que se M, iniciada na entrada w, não parou após n passos, então $\tau'(M, w) \vDash \chi(M, w, n) \wedge \psi(\overline{n})$. Como M iniciada

na entrada w não para, $\tau'(M,w) \vDash \chi(M,w,n) \wedge \psi(\overline{n})$ para todo $n \in \mathbb{N}$. Note que, por Proposição 2.10, $\tau'(M,w) \vDash \overline{k} < \overline{n}$ para todo $k < n$. Além disso, $\psi(\overline{n}) \vDash \overline{k} < \overline{n} \to \overline{k} \neq \overline{n}$. Assim, $\mathfrak{M} \vDash \overline{k} \neq \overline{n}$ para todo $k < n$, isto é, os infinitos termos $\overline{k}$ devem ter todos valores diferentes em $\mathfrak{M}$. Mas isso requer que $|\mathfrak{M}|$ seja infinito, de modo que $\mathfrak{M}$ não pode ser um modelo finito de $\tau'(M,w) \wedge \alpha(M,w)$. □

Problema 2.12. Complete a prova de Lema 2.20 provando que se M, iniciada na entrada w, não parou após n passos, então $\tau'(M,w) \vDash \psi(\overline{n})$.

tur:und:tra: thm:trakhtenbrodt

Teorema 2.21 (Teorema de Trakhtenbrot). *É indecidível se uma sentença arbitrária da lógica de primeira ordem tem um modelo finito (isto é, é finitamente satisfatível).*

Prova. Suponha que houvesse uma máquina de Turing F que decide o problema da satisfatibilidade finita. Então, dada qualquer máquina de Turing M e entrada w, poderíamos computar a sentença $\tau'(M,w) \wedge \alpha(M,w)$ e usar F para decidir se ela tem um modelo finito. Por Lemas 2.19 e 2.20, ela tem se, e somente se, M iniciada na entrada w para. Assim, poderíamos usar F para resolver o problema da parada, que sabemos ser insolúvel. □

tur:und:tra: cor:fproof-incomp

Corolário 2.22. *Não pode haver um sistema de derivação que seja correto e completo para a validade* finita*, isto é, um sistema de derivação que tenha $\vdash \psi$ se, e somente se, $\mathfrak{M} \vDash \psi$ para toda estrutura finita $\mathfrak{M}$.*

Prova. Exercício. □

Problema 2.13. Prove Corolário 2.22. Observe que ψ é satisfeita em toda estrutura finita se, e somente se, $\neg\psi$ não é finitamente satisfatível. Explique por que a satisfatibilidade finita é semidecidível no sentido de Teorema 2.18. Use isso para argumentar que se houvesse um sistema de derivação para a validade finita, então a satisfatibilidade finita seria decidível.

Créditos das Imagens

Referências Bibliográficas