Capítulo 10: Modelos Computacionais

Você já se perguntou, enquanto aprendia como programar computadores a realizar tarefas que deseja, sobre o que não pode ser feito com uma máquina? E o que pode ser feito, em princípio (algo que é resolvível), mas não pode ser feito de forma viável (é intratável)?

Neste capítulo, vamos começar a considerar a própria natureza da computação. Isto foi intensamente estudado durante um século, portanto, teremos apenas uma pequena amostra de tudo o que é conhecido.

Veremos alguns tipos de estruturas utilizadas para modelar a computação: gramáticas, máquinas de estado finitas, e finalmente a Máquina de Turing (TM). Gramáticas são utilizadas para gerar as palavras válidas de uma linguagem bem como para determinar se uma palavra é válida em uma linguagem. Elas são utilizadas tanto em linguagens formais, como *linguagens de programação* ou *lógica de predicados*, bem como em linguagens naturais, como a *língua portuguesa*. Na computação, as linguagens são de extrema importância para a construção de compiladores e interpretadores. Os tipos de gramática que veremos foram introduzidas pelo linguista estadunidense Noam Chomsky, nos anos 1950, hoje muito conhecido também pelo seu trabalho como ativista político.

Várias tipos de máquinas de estados podem ser utilizadas como modelo computacional. Algumas, mais simples, como as máquinas de estados finitos, não são suficientemente poderosas para modelar tudo que pode ser computado por um computador convencional. Entretanto, tais máquinas simples já podem ser utilizadas para modelar muitos tipos de máquina, como máquinas de venda automáticas. Outras, mais complexas, como as Máquinas de Turing, podem ser utilizadas como modelo computacional geral, de tal maneira que tudo que é computável pode ser realizado por uma Máquina de Turing. Embora seja um modelo teórico, a máquina de Turing é extremamente importante para estudar a dificuldade intrínseca de resolver problemas computacionalmente, permitindo a classificação destes como resolvíveis ou não, tratáveis, ou não.

10.1. Máquinas de Turing

Apesar de, na nossa reflexão, desejarmos minimizar a importância de detalhes de implementação, seguiremos a abordagem de Alan Turing de que o primeiro passo para definir o conjunto de funções computáveis é refletir sobre os detalhes do que os mecanismos computacionais podem fazer.

O contexto do pensamento de Turing foi o *Entscheidungsproblem*[1], proposto em 1928 pelos matemáticos alemães David Hilbert e Wilhelm Ackermann, que solicita um algoritmo que decide, após receber como entrada uma afirmação matemática, se essa afirmação é verdadeira ou falsa[2]. Assim, ele considerou o tipo de cálculo de manipula-

1 Problema de decisão, em Alemão.

2 Ao terminar a computação, ele pode acender uma luz para “verdadeiro”, ou imprimir o símbolo 1.

ção de símbolos familiar em matemática, como quando fatoramos um polinômio ou damos um passo numa demonstração de geometria plana.

Depois de refletir sobre o assunto durante algum tempo, um dia, depois de uma corrida, Turing deitou-se na grama e imaginou um funcionário fazendo multiplicação à mão com uma folha de papel que gradualmente se torna cheia de colunas de números. Com esta imagem como ponto de partida, Turing apresentou condições para o agente computacional[3].

Em primeiro lugar, o agente computacional tem uma **unidade de memória**, tal como o papel do funcionário, para armazenar e recuperar informação.

Em segundo lugar, o agente computacional deve seguir um procedimento definido, um **conjunto preciso de instruções**, sem margem para pulos criativos. Parte do que torna o procedimento definido é que as instruções não envolvem métodos aleatórios, tais como a contagem de partículas de uma deterioração radioativa, para determinar qual das alternativas executar.

A outra coisa que torna o procedimento definido é que o agente não utiliza métodos contínuos ou dispositivos analógicos. Portanto, não há dúvida sobre a precisão das operações, como poderia haver, por exemplo, quando se leem os resultados de uma régua ou de um ponteiro num mostrador analógico de um instrumento. Em vez disso, o agente trabalha de uma forma *discreta*, passo-a-passo. Por exemplo, se necessário, ele poderia fazer uma pausa entre os passos, anotar onde se encontra (ex.: “prestes a transportar um 1”), e mais tarde retomar o processo. Dizemos que em cada momento o funcionário está num dentre os possíveis elementos de um **conjunto finito de estados possíveis**, que denotamos q_0, q_1, ...

A terceira condição de Turing surgiu porque ele queria investigar o que é computável em princípio. Por conseguinte, ele não impôs qualquer limite superior à quantidade de memória disponível. Mais precisamente, ele não impôs nenhum limite superior finito — se um cálculo ameaçar esgotar o espaço de armazenamento, então lhe é fornecido mais espaço. Isto inclui a não imposição de limite superior à quantidade de memória disponível para entradas ou saídas, e a não imposição de limite superior à quantidade de armazenamento extra, memória temporária, necessária para além daquela para entradas e saídas.[4] Da mesma forma, ele não impôs limite superior ao número de instruções. E, deixou sem limite o número de passos que uma computação executa antes de terminar.[5]

3 Da próxima vez que você terminar de fazer exercícios, tente resolver um problema fundamental da computação.

4 É verdade que um computador físico como o seu celular tem um espaço de memória limitado (sem levar em consideração o armazenamento de coisas na nuvem). No entanto, esse espaço é extremamente grande. Neste sentido, quando trabalhamos com os dispositivos modelo, descobrimos que impor um limite à memória é irrelevante, ou mesmo um impedimento.

5 Alguns autores descrevem a disponibilidade de recursos tais como a quantidade de memória como “infinita”. O próprio Turing faz isto. Um leitor pode opor-se porque isto viola o objetivo da definição, de modelar computações fisicamente realizáveis, e por isso o presente texto diz, em vez disso, que os recursos não têm limite superior finito. Mas, na verdade, não importa. Se mostrarmos que algo não pode ser computado quando não há limites, então mostramos que não pode ser computado em qualquer dispositivo do mundo real.

A questão final que Turing enfrentou é: quão inteligente é o agente computacional? Por exemplo, ele pode multiplicar? Não precisamos incluir uma unidade especial de multiplicação porque podemos, em princípio, multiplicar através de adições repetidas. Nem sequer precisamos de adição porque podemos iterar a operação de sucessor, a operação de somar um. Desta forma, ele reduziu o agente computacional até que fosse bastante básico, bastante fácil de compreender, até que as operações sejam tão elementares que não podemos facilmente imaginá-las ainda mais subdivididas, enquanto que ainda manteve o agente suficientemente poderoso para fazer qualquer coisa que possa, em princípio, ser feita.

Com base nestas reflexões, Turing desenhou uma caixa contendo um mecanismo e equipada com uma fita.

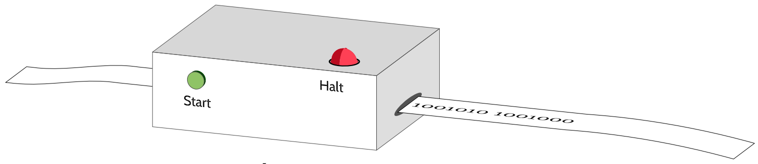


Figura 10.1: Máquina de Turing

A fita é a memória, por vezes chamada de repositório. A caixa pode ler e escrever nela, um caractere de cada vez, bem como mover uma cabeça de leitura/escrita em relação à fita em qualquer direção. Por exemplo, para multiplicar, o agente computacional pode obter os dois multiplicandos de entrada da fita (a figura mostra 74 e 72, representados em binário – sem sinal – e separados por um espaço em branco), pode utilizar a fita para o trabalho de rascunho, e pode escrever a saída na fita.

A caixa é o agente de computação, a CPU, por vezes chamada de controle. O botão *Start* inicia o cálculo e quando este termina, a luz *Halt* acende. A engenharia dentro da caixa não é importante — talvez tenha circuitos integrados como as máquinas a que estamos habituados, ou talvez tenha engrenagens e alavancas, ou talvez LEGO's - o que importa é que cada um dos seus componentes (em quantidade finita) pode estar em apenas uma quantidade finita de estados. Se tiver chips, então cada registrador tem um número finito de valores possíveis e se for feito com engrenagens ou blocos, então cada um deles estabelece-se em apenas um número finito de posições possíveis. Assim, seja como for construída, no total a caixa tem apenas um número finito de estados.

Ao executar um cálculo, o mecanismo passa de estado para estado. Por exemplo, um agente que faz multiplicação pode determinar, devido ao estado em que se encontra agora e ao que está lendo na fita, que precisa agora transportar 1. O agente transiciona para um novo estado, aquele cujo significado intuitivo é que o transporte de 1 ocorre lá.

Consequentemente, as etapas da máquina envolvem quatro pedaços de informação. Denominamos o estado atual como q_p e o próximo estado como q_n. Os outros dois pedaços, T_p e T_n, descrevem o símbolo da fita para o qual a cabeça de leitura/escrita está atualmente apontando e o que acontece a seguir com a fita. Quanto ao conjunto de caracteres que vão na fita, escolheremos o que for conveniente, mas com exceção de finitamente muitos lugares, cada fita é preenchida com espaços em branco, pelo que este deve ser um dos símbolos (denotamos o branco por B onde deixar um espaço vazio poderia causar confusão). As coisas que podem acontecer a seguir com a fita são: escrever um símbolo na fita sem mover a cabeça, que denotamos com esse símbolo, por exemplo por $T_n = 1$ (para dizer que será escrito 1), ou mover a cabeça da fita para a esquerda ou direita sem escrever, que denotamos por $T_n = \mathsf{L}$ ou $T_n = \mathsf{R}$.[6]

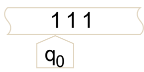
A 4-upla $q_p T_p T_n q_n$ é uma instrução. Por exemplo, a instrução $q_3 1\mathsf{B} q_5$ só é executada se a máquina estiver agora no estado q_3 e estiver lendo um 1 na fita. Se assim for, a máquina escreve um branco na fita, substituindo o 1, e passa para o estado q_5.

Exemplo 10.1.1

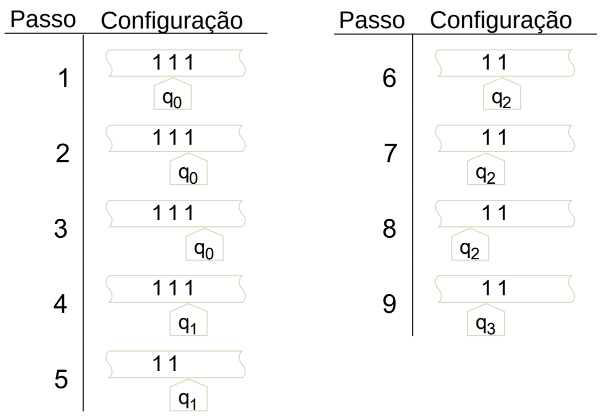
Esta máquina de Turing com o conjunto de caracteres $\Sigma = \{\mathsf{B}, 1\}$ tem seis instruções.

$$\mathcal{P}_{\text{pred}} = \{q_0\mathsf{BL}q_1,\ q_0 1\mathsf{R}q_0,\ q_1\mathsf{BL}q_2,\ q_1 1\mathsf{B}q_1,\ q_2\mathsf{BR}q_3,\ q_2 1\mathsf{L}q_2\}$$

Para rastrear a sua execução, abaixo representamos esta máquina numa configuração inicial. Isto mostra um trecho da fita, incluindo todo o seu conteúdo não em branco, juntamente com o estado da máquina e a posição da sua cabeça de leitura-escrita.



Adotamos a convenção de que quando pressionamos Start a máquina está no estado q_0. A figura mostra-a lendo 1, portanto a instrução $q_0 1\mathsf{R} q_0$ aplica-se. Assim, o primeiro passo é que a máquina move a sua cabeça da fita para a direita e permanece no estado q_0. Abaixo, a primeira linha mostra isto e as linhas posteriores mostram a configuração da máquina após os passos posteriores. Grosso modo, a computação desliza para a direita, apaga o 1 final, e desliza de volta para o início.



A seguir, por não haver estado q_3, não se aplica nenhuma instrução e a máquina pára. Podemos pensar nesta máquina como computando a função predecessora[7]

$$\text{pred}(x) = \begin{cases} x - 1 & \text{– se } x > 0 \\ 0 & \text{– caso contrário} \end{cases}$$

---

6 Se movemos a fita ou a cabeça não importa, o que importa é o seu movimento relativo. Assim, $T_n = \mathsf{L}$ significa que uma ou outra se move de tal forma que a cabeça agora aponta para o local um lugar à esquerda. Nas figuras, mantemos a fita parada e movemos a cabeça porque depois é mais fácil comparar as figuras passo a passo.

7 Note que **não** estamos utilizando representação binária, mas simplesmente a quantidade de 1's para representar um valor.

porque se inicializarmos a fita para que contenha somente uma sequência de 1's de comprimento n e a cabeça da máquina apontar para o primeiro deles, então no final a fita terá $n - 1$ 1's (exceto para $n = 0$, quando a fita terminará sem nenhum 1).

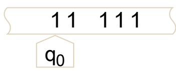
Exemplo 10.1.2

Esta máquina soma dois números naturais.

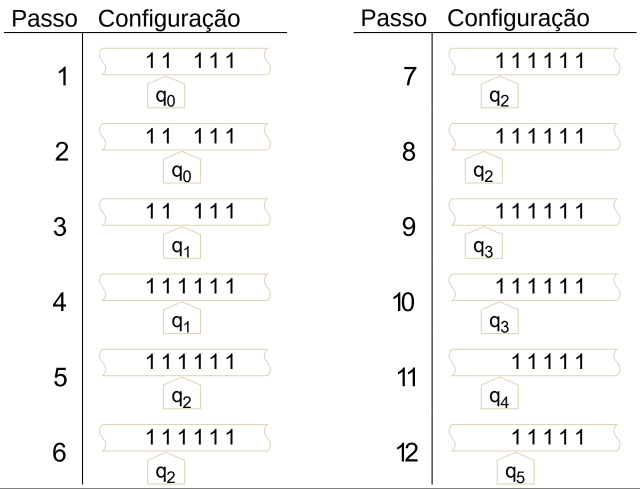
$$\mathcal{P}_{\text{add}} = \{\ q_0\mathsf{BB}q_1,\ q_0\mathsf{1R}q_0,\ q_1\mathsf{B1}q_1,\ q_1\mathsf{11}q_2,\ q_2\mathsf{BB}q_3,\ q_2\mathsf{1L}q_2,$$

$$q_3\mathsf{BR}q_3,\ q_3\mathsf{1B}q_4,\ q_4\mathsf{BR}q_5,\ q_4\mathsf{11}q_5\ \}$$

Os números de entrada são representados por sequências de 1's que são separadas por um espaço em branco. A cabeça de leitura/escrita começa sob o primeiro símbolo do primeiro número. Eis a máquina pronta a calcular 2 + 3:

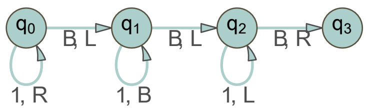


A máquina varre para a direita, procurando o separador em branco. Muda este para 1, depois varre para a esquerda até encontrar o início. Finalmente, ela corta um 1 e pára com a cabeça de leitura/escrita no início da sequência. Aqui estão os passos:



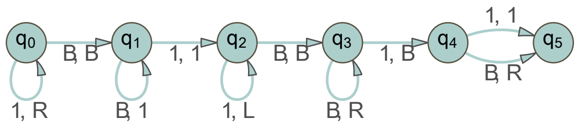
Em vez de fornecer as instruções de uma máquina como uma lista, podemos usar uma tabela ou um diagrama. Aqui está a **tabela de transições** para a $\mathcal{P}_{\text{pred}}$ e o seu **grafo de transições**.

Δ	B	1
q_0	Lq_1	Rq_0
q_1	Lq_2	Bq_1
q_2	Rq_3	Lq_2
q_3	–	–



E aqui está a tabela e o grafo correspondentes para $\mathcal{P}_{\text{add}}$.

Δ	B	1
q_0	Bq_1	Rq_0
q_1	1q_1	1q_2
q_2	Bq_3	Lq_2
q_3	Rq_3	Bq_4
q_4	Rq_5	1q_5
q_5	–	–

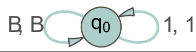


O grafo é como apresentaremos mais frequentemente máquinas que são pequenas, mas se houver muitos estados então ele pode ser visualmente confuso.

A seguir, uma observação crucial. Algumas máquinas de Turing, pelo menos para algumas configurações iniciais, nunca param.

Exemplo 10.1.3

A máquina $\mathcal{P}_{\text{inf loop}} = \{\ q_0\mathsf{BB}q_0,\ q_0\mathsf{11}q_0\ \}$ nunca pára, independentemente da entrada.



Tente conceber exemplos de máquinas de Turing que param em algumas entradas e não em outras.

Chegou o momento das definições. Tomamos um **símbolo** como algo que o dispositivo pode escrever e ler, para armazenamento e recuperação.[8]

Definição: Máquina de Turing

Uma **máquina de Turing** $\mathcal{P}$ é um conjunto finito de 4-uplas de **instruções** $q_pT_pT_nq_n$. Numa instrução, o **estado atual** q_p e o **próximo estado** q_n são elementos de um **conjunto de estados** Q. O **símbolo de entrada** ou **símbolo atual** T_p é um elemento do **conjunto alfabeto** da fita Σ, que contém pelo menos dois membros, incluindo um chamado **em branco** (e não contém L ou R). O **símbolo de ação** ou **próximo símbolo** T_n é um elemento do **conjunto de ações** $\Sigma \cup \{\mathsf{L}, \mathsf{R}\}$.

O conjunto $\mathcal{P}$ deve ser **determinístico**: diferentes 4-uplas não podem começar com o mesmo q_pT_p. Assim, sobre o conjunto de instruções $q_pT_pT_nq_n \in \mathcal{P}$, a associação do par atual q_pT_p com o próximo par T_nq_n define uma função, a **função de transição** ou **função do próximo estado** $\Delta : Q \times \Sigma \to (\Sigma \cup \{\mathsf{L}, \mathsf{R}\}) \times Q$.

Denominamos uma máquina de Turing por $\mathcal{P}$ porque na nossa experiência quotidiana o que mais se assemelha a uma máquina de Turing é um programa.

8 Como o dispositivo faz isto depende dos seus detalhes de construção. Por exemplo, para ter uma máquina com dois símbolos, em branco e 1, podemos ler e escrever marcas numa fita de papel, ou alinhar partículas magnéticas numa fita plástica, ou bits num chip, ou podemos empurrar blocos LEGO para o lado esquerdo ou direito de uma fenda. A discretização assegura que a máquina possa distinguir claramente os símbolos, em contraste com o problema que um instrumento pode ter em distinguir dois valores próximos do seu limite de resolução.

Evidentemente, o sentido de uma máquina é o que ela faz. Uma máquina de Turing é um plano para uma computação — é como um programa — e assim para terminar a formalização iniciada pela definição, damos uma descrição completa de como estas máquinas atuam.

Vimos no rastreamento do Exemplo 10.1.1 e do Exemplo 10.1.2 que uma máquina atua fazendo a transição de uma configuração para a seguinte. Uma **configuração** de uma máquina de Turing é uma 4-upla $C = \langle q, s, \tau_L, \tau_R \rangle$, onde q é um estado, um membro de Q, s é um caractere do alfabeto da fita Σ, e τ_L e τ_R são sequências de elementos do alfabeto da fita, incluindo possivelmente a sequência vazia ε. Estes significam o estado atual, o caractere sob a cabeça de leitura/escrita, e o conteúdo da fita à esquerda e à direita da cabeça. Por exemplo, a linha 2 da tabela de rastreamento do Exemplo 10.1.2, onde o estado é $q = q_0$, o caractere sob a cabeça s é o em branco, e à esquerda da cabeça temos $\tau_L = 11$, enquanto que à direita temos $\tau_R = 111$, representa graficamente a configuração $\langle q, s, \tau_L, \tau_R \rangle$. Ou seja, uma configuração é um retrato instantâneo, um instante da computação.

Escrevemos $C(t)$ para a configuração da máquina após a t-ésima transição, e dizemos que esta é a configuração no passo t. Estendemos isso ao passo 0, e dizemos que a configuração inicial $C(0)$ é a configuração da máquina antes de apertarmos Start.

Suponha que no passo t uma máquina $\mathcal{P}$ esteja na configuração $C(t) = \langle q, s, \tau_L, \tau_R \rangle$. Para fazer a próxima transição, encontre uma instrução $q_p T_p T_n q_n \in \mathcal{P}$ com $q_p = q$ e T_p = s. Se não houver tal instrução então no passo $t + 1$ a máquina $\mathcal{P}$ **pára**.

Caso contrário, haverá apenas uma instrução deste tipo, por determinismo. Há três possibilidades. (1) Se T_n for um símbolo no alfabeto da fita Σ então a máquina escreve esse símbolo na fita, de modo que a configuração seguinte seja $C(t + 1) = \langle q_n, T_n, \tau_L, \tau_R \rangle$. (2) Se $T_n = \mathsf{L}$, então a máquina move a cabeça da fita para a esquerda. Ou seja, a configuração seguinte é $C(t + 1) = \langle q_n, \tilde{s}, \tilde{\tau}_L, \tilde{\tau}_R \rangle$, onde $\tilde{s}$ é o caractere mais à direita da cadeia τ_L (se $\tau_L = \varepsilon$ então $\tilde{s}$ é o caractere em branco), onde $\tilde{\tau}_L$ é τ_L com o seu caractere mais à direita removido (se $\tau_L = \varepsilon$ então $\tilde{\tau}_L = \varepsilon$ também), e onde $\tilde{\tau}_R$ é a concatenação de $\langle s \rangle$ e τ_R. (3) Se $T_n = \mathsf{R}$ então a máquina move a cabeça da fita para a direita. Isto é análogo a (2) – mas na outra direção, por isso omitimos os detalhes.

Se duas configurações estiverem relacionadas por estarem separadas por um passo, então escrevemos $C(i) \vdash C(i + 1)$.[9] Uma computação é uma sequência $C(0) \vdash C(1) \vdash C(2) \vdash \ldots$. Abreviamos essa sequência por $\vdash^*$.[10] Se a computação parar, então a sequência tem uma configuração final $C(h)$, por isso podemos escrever uma computação de parada como $C(0) \vdash* C(h)$.

Exemplo 10.1.4

No Exemplo 10.1.1, as imagens que traçam a execução da máquina mostram as sucessivas configurações. Portanto, o cálculo é o seguinte:

9 Leia o símbolo da catraca $\vdash$ em voz alta como “deriva em um passo”. Poderíamos, onde I é a instrução aplicável, escrever $\vdash_I$, mas nunca precisaremos dessa construção.

10 Leia isso em voz alta como “deriva eventualmente”, ou simplesmente “deriva”.

$\langle q_0, 1, \varepsilon, 11\rangle \vdash \langle q_0, 1, 1, 1\rangle \vdash \langle q_0, 1, 11, \varepsilon\rangle \vdash \langle q_0, \mathtt{B}, 111, \varepsilon\rangle \vdash \langle q_1, 1, 11, \varepsilon\rangle \vdash$
$\langle q_1, \mathtt{B}, 11, \varepsilon\rangle \vdash \langle q_2, 1, 1, \varepsilon\rangle \vdash \langle q_2, 1, \varepsilon, 1\rangle \vdash \langle q_2, \mathtt{B}, \varepsilon, 11\rangle \vdash \langle q_3, 1, \varepsilon, 1\rangle$

Essa descrição da ação de uma máquina de Turing enfatiza que ela é uma máquina de estados — a computação da máquina de Turing é a respeito das transições, os passos discretos que levam de uma configuração para outra.

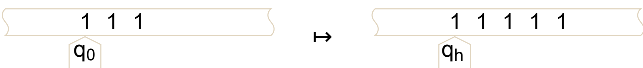
Funções efetivas. No início deste capítulo, declaramos que o nosso interesse não é tanto nas máquinas, mas nas coisas que elas computam. Encerraremos essa seção com uma definição do conjunto de funções que são mecanicamente computáveis.

Uma função é uma associação de entradas com saídas. A candidata mais simples para uma definição de função computada por uma máquina é a associação da sequência na fita quando a máquina começa com a sequência na fita quando esta pára, se ela de fato parar. (Para a definição seguinte, note que onde X é um conjunto de caracteres, utilizamos $X*$ para denotar o conjunto de sequências de comprimento finito desses caracteres).

Definição. Função computada por uma máquina de Turing

Seja $\mathcal{P}$ uma máquina de Turing com alfabeto de fita Σ, e seja $\Sigma_0 = \Sigma - \{\mathtt{B}\}$. A função $\boldsymbol{\phi_{\mathcal{P}} : \Sigma_0* \to \Sigma_0*}$ **computada por** $\boldsymbol{\mathcal{P}}$ é: para uma entrada $\sigma \in \Sigma_0*$, a saída é a cadeia de caracteres que resulta da colocação de σ numa fita até então em branco, apontando a cabeça de leitura/escrita para o seu símbolo mais à esquerda, e executando a máquina. Se $\mathcal{P}$ parar, e os caracteres não em branco forem consecutivos, e o primeiro caractere estiver debaixo da cabeça, então $\phi_{\mathcal{P}}(\sigma)$ é tal cadeia de caracteres.

Eis a ilustração do cálculo de uma função onde $\phi(111) = 11111$.[11] (Se houver apenas uma máquina em discussão, então podemos omitir o subscrito e escrever apenas ϕ).



Essa definição tem dois pontos delicados, ambos necessários para que a associação entrada-saída seja bem definida. Um é que apenas especificar que a máquina começa com σ na fita não é suficiente, uma vez que a posição inicial da cabeça pode alterar a saída.[12] E, a definição omite os espaços em branco de σ e τ, uma vez que a máquina não seria capaz de distinguir espaços em branco no fim dessas sequências dos espaços em branco que fazem parte da fita sem limites.

A definição diz “Se $\mathcal{P}$ parar …” E se não o fizer?

Definição.

Se para uma máquina de Turing o valor de uma computação não for definido para alguma entrada $\sigma \in \Sigma_0$ então dizemos que a função calculada pela máquina **diverge**,

11Os matemáticos começaram o assunto estudando o cálculo eficaz das funções matemáticas, principalmente funções da teoria dos números. Eles trabalhavam frequentemente com o alfabeto mais simples, $\Sigma = \{\mathtt{B}, 1\}$. Esta abordagem revelou-se frutífera, portanto pesquisadores ainda discutem frequentemente o assunto nestes termos.

12 Alguns autores não exigem que o primeiro caractere da saída esteja sob a cabeça. Mas dessa maneira é mais elegante.

escrevemos $\boldsymbol{\phi(\sigma)}\boldsymbol{\uparrow}$ (ou $\phi(\sigma) = \perp$). Quando a máquina tem um valor de saída associado, dizemos que a sua função converge, escrito $\boldsymbol{\phi(\sigma)}\boldsymbol{\downarrow}$. Se ϕ for definida para cada entrada em Σ_0, então é uma **função total**. Se divergir para pelo menos um membro de Σ_0, então é uma **função parcial**.

Muito importante: note a diferença entre uma máquina $\mathcal{P}$ e a função computada por essa máquina $\phi_{\mathcal{P}}$. Por exemplo, a máquina $\mathcal{P}_{\text{pred}}$ é um conjunto de 4-uplas mas a função predecessora é um conjunto de pares entrada-saída, que podemos escrever como $x \to \text{pred}(x)$. Outro exemplo da diferença é que as máquinas param ou não param, enquanto as funções convergem ou divergem.

Essa definição parece permitir apenas funções com uma única entrada e saída; e as funções com múltiplas entradas ou saídas? Por exemplo, a função que aceita dois números naturais a e b e devolve a^b é intuitivamente computável mecanicamente mas não está obviamente contemplada.

O truque aqui é considerar a cadeia de entrada como uma codificação, uma representação, de múltiplas entradas. Desta forma, podemos obter uma função de duas ou mais entradas a partir de uma única cadeia de entrada.

OK, então, e que tal computar com não-números? Por exemplo, podemos querer encontrar o caminho mais curto entre dois nós de um grafo. Numa extensão do parágrafo anterior, para computar com um grafo, precisamos encontrar uma forma de representá-lo por uma cadeia de caracteres. Programas que funcionam com grafos primeiramente decodificam a cadeia de entrada, depois computam a resposta, e terminam por codificar a resposta como uma cadeia de saída.

Estas codificações podem parecer esquisitas, e são. (Claro, uma linguagem de programação faz conversões de decimal para binário e vice-versa que são de certa forma semelhantes). Mas uma máquina de Turing simplesmente manipula caracteres e nós externamente fornecemos uma interpretação.

Quando descrevemos a função computada por uma máquina, normalmente omitimos a parte sobre a interpretação das cadeias de caracteres. Poderíamos dizer, “isto mostra que $\phi(3) = 5$” em vez de, “isto mostra que ϕ recebendo uma cadeia que representa 3 para uma cadeia que representa 5”.

Observação. Os primeiros pesquisadores, trabalhando antes que os computadores estivessem amplamente disponíveis, precisavam de argumentos robustos de que existe um cálculo mecânico de, digamos, a função que recebe um número n e devolve o n-ésimo primo. Assim, eles trabalharam nos detalhes, demonstrando que as suas definições e argumentos eram concordantes com a sua intuição através da construção de um grande conjunto de evidências. Mas aqui tomamos um rumo diferente. A nossa experiência diária com as máquinas à nossa volta é que elas são capazes de usar o seu alfabeto, binário, para obter uma representação razoável de tudo o que a nossa intuição diz ser computável. Assim, o nosso desenvolvimento não consistirá em fixar uma codificação e trabalhar através dos detalhes de demonstrar que certas funções, que pensamos serem obviamente computáveis, são de fato computáveis pelas definições que demos. Omitimos isto simplesmente para chegarmos mais cedo ao material interessante. A seção seguinte dirá mais a este respeito.

Definição. Função computável

Uma **função computável**, ou **função recursiva**[13], é uma função total ou parcial que é computada por alguma máquina de Turing. Um **conjunto computável**, ou **conjunto recursivo**, é aquele cuja função característica[14] é computável. Uma máquina de Turing **decide** para um conjunto se computa a função característica desse conjunto. Uma relação é computável apenas se for computável como um conjunto.

Encerramos esta seção com um resumo. Fornecemos uma caracterização da computação mecânica. Vemo-la como um processo pelo qual um sistema físico progride através de uma sequência de passos discretos que são locais, o que significa que toda a ação tem de ocorrer dentro de uma célula da cabeça. Isto levou a uma definição precisa de quais as funções que são mecanicamente computáveis. Na próxima subsecção discutiremos esta caracterização, incluindo a evidência que leva à sua aceitação generalizada.

10.2. A Tese de Church

História. Os algoritmos sempre desempenharam um papel central na matemática. O exemplo mais simples é uma fórmula como a que dá a altura de uma bola que caiu da Torre Inclinada de Pisa, $h(t) = -4{,}9t^2 + 56$. Este é um tipo de programa: obtém-se a altura resultante ao elevar ao quadrado o tempo fornecido como entrada, multiplicando por −4,9, e, finalmente, adicionando 56.

Nos anos 1670's Gottfried Wilhelm von Leibniz, o co-criador do Cálculo, construiu a primeira máquina capaz de fazer adição, subtração, multiplicação, divisão, e também raízes quadradas. Isto levou-o a especular sobre a possibilidade de uma máquina que manipulasse não só números mas também símbolos e que pudesse assim determinar a verdade das afirmações científicas. Para resolver qualquer disputa, Leibnitz escreveu, os estudiosos poderiam simplesmente dizer, "*Calculemus*"![15] Esta é uma versão do Entscheidungsproblem.

O verdadeiro impulso para compreender a computação surgiu em 1927 a partir do teorema da Incompletude de Gödel. Este diz que para qualquer sistema de axioma (suficientemente poderoso) existem afirmações que, embora verdadeiras em qualquer modelo dos axiomas, não são demonstráveis a partir desses axiomas. Gödel forneceu um algoritmo no qual se introduz os axiomas e a afirmação é produzida. Isto tornou evidente a necessidade de definir o que é 'algorítmico' ou 'intuitivamente computável mecanicamente' ou 'eficaz'.

Uma série de matemáticos propôs formalizações. Um deles foi Alonzo Church,[16] que propôs o **λ-cálculo**. Church e os seus alunos utilizaram este sistema para derivar muitas funções que são intuitivamente calculáveis mecanicamente, incluindo as fun-

13 Este uso para o termo "função recursiva" está em desuso.

14 A função característica de um conjunto S, denotada por $\mathbb{1}_S$ é a função booleana para determinar a membresia: $\mathbb{1}_S(s) = 1$ se $s \in S$ e $\mathbb{1}_S(s) = 0$ se $s \notin S$.

15 "Calculemos!" em Latim.

16 Depois de produzir o seu modelo de máquina, Turing tornou-se aluno de doutorado de Church em Princeton.

ções polinomiais e funções da teoria dos números, tais como encontrar o resto da divisão. Eles não conseguiram encontrar nenhuma tal função que o λ-cálculo não conseguisse fazer. Church sugeriu a Gödel, o mais proeminente especialista na área, que o conjunto de funções eficazes, o conjunto de funções que são intuitivamente calculáveis mecanicamente, que não é precisamente determinado, é o mesmo que o conjunto de funções que são λ-computáveis, o que é verdade. Mas Gödel não estava convencido.

Isso mudou quando Gödel leu a análise magistral de Turing, esboçada na seção anterior. Ele posteriormente escreveu: “Que esta se trata realmente da definição correta de computabilidade mecânica foi estabelecida inquestionavelmente por Turing”.

A Tese de Church.

O conjunto de coisas que podem ser computadas por um mecanismo discreto e determinístico é o mesmo que o conjunto de coisas que podem ser computadas por uma máquina de Turing.[17]

A Tese de Church é central para a Teoria da Computação. Ela diz que os nossos resultados técnicos têm uma maior importância — eles descrevem os dispositivos que estão nas nossas mesas e nos nossos bolsos. Assim, nesta seção fazemos uma pausa para expandir alguns pontos, particularmente aqueles que a experiência tem mostrado que podem levar a mal-entendidos.

Evidência. Não podemos provar a Tese de Church. Ou seja, não podemos fornecer uma prova matemática. A definição de uma máquina de Turing, ou do lambda cálculo ou outros esquemas equivalentes, formaliza a noção de “eficaz” ou “intuitivamente computável mecanicamente”. Quando um pesquisador concorda que ela explica corretamente ‘computável num mecanismo discreto e determinístico’ e concorda em trabalhar dentro dessa formalização, fica então livre de proceder com raciocínio matemático sobre estes sistemas. Assim, de certa forma, a Tese de Church vem antes da matemática, ou em todo o caso situa-se fora do trabalho habitual de derivação e verificação da matemática. Turing escreveu: “Todos os argumentos que podem ser fornecidos apelam, fundamentalmente, à intuição e, por esta razão, são bastante insatisfatórios do ponto de vista matemático”.

Apesar de não ser a conclusão de um sistema dedutivo, a Tese de Church é muito amplamente aceita. Daremos quatro pontos a seu favor que persuadiram Gödel, Church, e outros contemporâneos, e que ainda hoje convencem os pesquisadores — **cobertura**, **convergência**, **consistência**, e **clareza**.

Primeiro, **cobertura**: tudo aquilo que as pessoas consideraram intuitivamente computável mostrou-se ser computável por uma máquina de Turing. Isto inclui não só as funções da teoria dos números investigadas pelos pesquisadores na década de 1930, mas também tudo o que já foi computado por cada programa escrito para cada computador existente, porque todos eles podem ser compilados para serem executados numa máquina de Turing.

17 Alguns autores chamam-na de Tese de Church-Turing. Aqui, nós optamos por chamá-la apenas de Tese de Church, já que Turing já tem a máquina, podemos dar a posse da tese para Church.

Apesar deste peso de evidência, o argumento por cobertura colapsaria se alguém exibisse sequer um contra-exemplo, uma operação que pode ser feita em tempo finito num dispositivo discreto e determinístico fisicamente realizável, mas que não pode ser feita numa máquina de Turing. Portanto, este argumento é forte mas, pelo menos, possivelmente não decisivo.

O segundo argumento é a **convergência**: para além de Turing e Church, muitos outros pesquisadores na época e desde então têm proposto outros modelos de computação. No entanto, apesar desta variação, a nossa experiência é que cada modelo produz o mesmo conjunto de funções computáveis. Por exemplo, Turing mostrou que o conjunto de funções computáveis com o seu modelo de máquina é igual ao conjunto de funções computáveis com o λ-cálculo de Church.

Agora, todos podem estar enganados. Poderia haver algum erro sistemático de pensamento em torno deste ponto. Durante séculos os geômetras pareceram incapazes de imaginar a possibilidade de o Postulado das Paralelas de Euclides não se manter e talvez uma cegueira cultural semelhante esteja a acontecer aqui. No entanto, se um certo número de pessoas muito inteligentes se afastam e trabalham independentemente sobre uma questão, e quando regressam, verifica-se que embora tenham adotado uma grande variedade de abordagens, todos obtiveram a mesma resposta, então é bem possível supor que é a resposta certa. No mínimo, a convergência diz que há algo de natural e convincente neste conjunto de funções.

Um argumento não disponível para Turing, Church, Gödel, e outros na década de 1930, uma vez que ele depende do trabalho realizado desde então, é a **consistência**: os detalhes da definição de uma máquina de Turing não são essenciais para o que pode ser computado. Por exemplo, podemos mostrar que uma máquina de uma só fita pode computar todas as funções que podem ser feitas por uma máquina com duas ou mais fitas. Assim, o fato de as máquinas da nossa definição terem apenas uma fita não é um ponto essencial.

Semelhantemente, as máquinas cuja fita não tem limites em apenas uma direção podem computar todas as funções computáveis com uma fita sem limites em ambas as direções. E as máquinas com mais de uma cabeça de leitura/escrita computam as mesmas funções que as máquinas com apenas uma. Quanto aos símbolos, podemos computar qualquer função intuitivamente calculável apenas utilizando um único símbolo para além do branco que cobre todas as células da fita inicial, ou seja, com $\Sigma = \{1, B\}$. Do mesmo modo, restringindo as máquinas para que só possam escrever uma única vez em cada posição, ou seja, que não possam mudar as marcas uma vez que estão na fita, também é suficiente para computar este conjunto de funções. Além disso, embora a restrição a máquinas com apenas um estado não seja suficiente, as máquinas de dois estados são tão poderosas quanto as máquinas sem limites de estados dadas na nossa definição.

Há mais uma condição que não altera o conjunto de funções calculáveis, o determinismo. Lembre-se de que a definição de máquina de Turing dada acima não permite, digamos, ambas as instruções $q_5 1 R q_6$ e $q_5 1 L q_4$ na mesma máquina, porque ambas começam com $q_5 1$. Se abandonarmos esta restrição, então a classe de máquinas que obtemos é chamada de **não determinística**. Teremos mais a dizer sobre isto posteriormente, mas a coleção de máquinas não determinísticas de Turing calcula o mesmo conjunto de funções que a coleção de máquinas determinísticas.

Assim, em qualquer ponto em que a definição da máquina de Turing parece fazer uma escolha arbitrária, fazer uma escolha diferente ainda produz o mesmo conjunto de funções computáveis. Isto é persuasivo na medida em que qualquer definição adequada do que é computável deveria possuir esta propriedade; por exemplo, se máquinas de duas fitas computassem mais funções do que máquinas de uma fita e máquinas de três fitas computassem mais do que estas, então identificar o conjunto de funções computáveis com aquelas computáveis por máquinas de uma fita seria uma tolice. Mas tal como no argumento anterior, embora isto signifique que a classe de funções computáveis por máquina de Turing é natural e abrangente, ainda deixa em aberto uma pequena fenda para uma possibilidade de a classe não esgotar a lista de funções que são mecanicamente computáveis.

O argumento mais persuasivo para a Tese de Church — o que levou Gödel a mudar de ideia e o que convence os estudiosos ainda hoje — é a **clareza**: A análise de Turing é convincente. Gödel notou isto na citação dada acima e Church sentiu da mesma forma, escrevendo que as máquinas de Turing têm, "a vantagem de tornar a identificação com eficácia… evidente imediatamente".

O que a Tese de Church não diz é que em todas as circunstâncias a melhor forma de compreender um cálculo discreto e determinístico é através do modelo da máquina de Turing. Por exemplo, um analista numérico que estude o desempenho na prática de um algoritmo de ponto flutuante[18] deve utilizar um modelo computacional que tenha registradores. A Tese de Church diz que o cálculo poderia, em princípio, ser feito por uma máquina de Turing, mas para esta utilização os registradores são mais apropriados.[19]

A Tese de Church também não diz que as máquinas de Turing são tudo o que existe para qualquer computação no sentido de que se, digamos, você estiver estudando um sistema de frenagem anti-bloqueio de automóvel (ABS), então o modelo da máquina de Turing é responsável pelos cálculos lógicos e aritméticos, mas não por todo o sistema, com entradas de sensores e saídas de atuadores. S Aaronson fez esta observação: "Suponha que eu… [argumentei] que… a tese [de Church] não capta toda a computação, porque as máquinas de Turing não podem assar pão…. Nunca ninguém afirmou que uma máquina de Turing poderia lidar com todas as interações possíveis com o mundo externo, sem primeiro ligá-la a periféricos adequados. Se você quiser uma máquina de Turing para torrar pão, é necessário ligá-la a uma torradeira; então a TM pode facilmente lidar com a lógica interna da torradeira".

Na mesma linha, podemos obter dispositivos físicos que fornecem um fluxo de bits aleatórios. Estes não são bits pseudo-aleatórios que são calculados por um método

18 *Ponto flutuante* é o nome da representação binária para números reais mais utilizada nos computadores. Ela guarda algumas diferenças em relação às representações para números inteiros que vimos anteriormente. Como o foco desse curso é no discreto, não veremos detalhes sobre essa representação.

19 Os neurologistas também consideram que as máquinas de Turing não são o modelo mais adequado. Note-se, no entanto, que dizer que um modelo de cérebro com interrupções é um modelo mais adequado não é o mesmo que dizer que as operações do cérebro não poderiam, em princípio, ser feitas utilizando uma máquina de Turing como substrato.

que é determinístico mas que passa nos testes estatísticos. Em vez disso, a física bem estabelecida diz-nos que estes bits são verdadeiramente aleatórios. A sua relevância aqui é que a Tese de Church apenas afirma que as máquinas de Turing modelam os cálculos discretos e determinísticos que podemos fazer depois de recebermos bits de entrada de um tal dispositivo.

Uma questão empírica? A Tese de Church postula que as máquinas de Turing podem fazer qualquer computação que seja discreta e determinística. Isso levanta uma grande questão: mesmo se aceitarmos a Tese de Church, será que podemos fazer mais indo além do discreto e determinístico? Por exemplo, os métodos analógicos — passar lasers através de um gás, digamos, ou algum tipo de magia subatômica — permitir-nos-iam computar coisas que nenhuma máquina de Turing pode computar? Ou serão estas a última palavra em máquinas fisicamente possíveis? Será que Turing, naquele dia, deitado na margem do rio, intuiu tudo o que as experiências com a realidade jamais encontrariam para o possível?

Como amostra grátis da conversa, é possível provar que existe um caso em que a equação da onda[20] tem condições iniciais que são computáveis (para os números reais iniciais x existe um programa que recebe como entrada $i \in \mathbb{N}$ e fornece a i-ésima casa decimal de x), mas a solução única não é computável. Então o tanque de ondas modelado por esta equação computa algo que as máquinas de Turing não conseguem?

Neste caso, podemos protestar dizendo que um aparato experimental está sujeito a problemas de ruído e medição, incluindo um número finito de casas decimais nos instrumentos, etc. Mas mesmo que uma análise cuidadosa da física de um tanque de ondas nos leve a desconsiderá-lo como computando de forma confiável uma função, ainda podemos questionar se existem outros aparatos que o fariam.

Esta grande questão permanece em aberto. Até agora, nenhuma análise de uma noção mais ampla de computação mecânica fisicamente possível no caso não discreto tem o apoio que a análise de Turing tem colhido no seu domínio mais restrito. Em particular, ninguém produziu ainda um exemplo amplamente aceito de um mecanismo não-discreto que computa uma função que nenhuma máquina de Turing computa.

Não prosseguiremos nesta via, limitando-nos a observar que a comunidade de pesquisadores pesou ao tomar a Tese de Church como base para o seu trabalho. Para nós, “computação” referir-se-á ao tipo de trabalho que Turing analisou. Isto porque queremos pensar em manipular símbolos, não em análises numéricas e não em torradas.

Usando a Tese de Church. A Tese de Church afirma que cada um dos modelos de computação — por exemplo, máquinas de Turing, e λ-cálculo — são maximamente capazes. Aqui enfatizamo-lo porque isso confere aos nossos resultados uma maior importância. Quando, por exemplo, descrevermos mais tarde uma função para a qual podemos provar que nenhuma máquina de Turing a pode computar, então, com a tese em mente, estenderemos essa declaração técnica para significar que esta função não pode ser computada por qualquer dispositivo discreto e determinístico.

Outro aspecto da tese de Church é que, como cada um dos modelos é maximamente capaz, estes, e outros equivalentes que não descreveremos, computam todos as mesmas coisas. Assim, podemos fixar um deles como a nossa formalização preferida e

20 Uma equação diferencial parcial que descreve a propagação das ondas.

prosseguir com a análise matemática. Podemos escolher, por exemplo, as máquinas de Turing.

Por fim, também vamos aproveitar a tese de Church para facilitar a nossa vida. Escrever algumas máquinas de Turing dá-nos uma visão mais profunda sobre a computabilidade mas, ao mesmo tempo, depois que escrevermos algumas máquinas de Turing, pode-se muito bem descobrir que fazer mais máquinas não proporciona mais esclarecimentos. Pior, concentrar-se demais nos detalhes das máquinas de Turing (ou nos detalhes de baixo nível de qualquer modelo computacional) pode obscurecer pontos maiores. Assim, se conseguirmos ser claros e rigorosos sem ter de lidar realmente com um grande volume de detalhes, então ficaremos satisfeitos.

A Tese de Church ajuda nisto. Muitas vezes, quando queremos mostrar que algo é computável por uma máquina de Turing, vamos primeiro argumentar que é intuitivamente computável e depois citar a Tese de Church para afirmar que é, portanto, computável por uma máquina de Turing. Com isso, o nosso argumento pode prosseguir: "Seja P essa máquina..." sem nunca termos exibido um conjunto de instruções de 4-uplas. Claro que há algum perigo de nos enganarmos com o "intuitivamente computável", mas todos nós temos muito mais experiência com isto do que as pessoas na década de 1930, de tal maneira que o perigo é mínimo. O lado positivo é que podemos fazer progressos rápidos neste material; podemos conseguir que as coisas sejam feitas.

Em muitos casos, para afirmar que algo é intuitivamente computável, vamos produzir um programa, ou esboçar um programa, fazendo essa coisa. Para isto gostamos de utilizar uma linguagem de programação moderna, como por exemplo Python.

10.3. Créditos

Todas as seções, foram adaptadas (traduzidas e modificadas) de [1], que está disponível sob a licença Creative Commons Attribution-ShareAlike 4.0 (CC BY-SA 4.0).

10.4. Referências

1. Hefferon, Jim. *Theory of Computation: Making Connections*. Disponível em http://hefferon.net/computation/

10.5. Licença