Teoria da Computação **FCT-UNL, 2º semestre 2023-2024**

Notas 10: Máquinas de Turing e a tese de Church-Turing

Autor: João Ribeiro

Introdução

Nestas notas introduzimos *máquinas de Turing*, o modelo de computação mais importante na informática e a base da nossa noção rigorosa de "algoritmo".

10.1 Um pouco de história

O interesse em estudar a existência de algoritmos para resolver grandes classes de problemas apareceu muito antes das primeiras tentativas de definir formalmente a noção de "algoritmo". Por exemplo, já no século XVII Leibniz tentou construir uma máquina que conseguisse decidir a veracidade de qualquer afirmação matemática. Mais tarde, em 1900, Hilbert publicou os seus 23 problemas de matemática em aberto mais importantes para o século XX. Entre estes, o 10º problema de Hilbert pedia o desenho de um algoritmo que, dada uma qualquer equação diofantina, tal como

$$x^2 + 2xy + y^2 = 0,$$

decide correctamente se esta tem soluções inteiras (isto é, soluções com $x, y \in Z$). Em 1928, Ackermann e Hilbert renovaram o desafio de desenhar um algoritmo que, dada qualquer afirmação matemática, consiga decidir a sua veracidade (a partir de um conjunto de axiomas). Este é o famoso *Entscheidungsproblem*.

O estudo da *existência* de algoritmos para os problemas acima referidos necessita, primeiro, de uma definição formal de algoritmo que acreditemos "capturar" (ou "simular") todos os computadores/algoritmos que possamos construir no mundo real. Nesta cadeira estudámos já alguns modelos de computação, tais como AFDs e os autómatos de pilha. Apesar de serem conceitos fundamentais na informática, é fácil convencermo-nos de que estes modelos capturam apenas computadores/algoritmos muito simples. Por exemplo, a linguagem $\{a^n b^n \mid n \in \mathbb{N}\}$ não é regular e a linguagem $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ não é livre de contexto, mas certamente conseguem escrever programas simples em Java que reconhecem estas linguagens.

Precisamos, portanto, de considerar modelos de computação muito mais poderosos, que capturem, em particular, tudo o que conseguimos programar em linguagens como Java, C, C++, Python, etc. O Entscheidungsproblem foi resolvido na negativa em simultâneo por Church [Chu36], para o seu modelo de cálculo-λ, e por Turing [Tur37], para o seu modelo de Máquinas de Turing, no período 1935–1937. Ou seja, Church e Turing mostraram que qualquer algoritmo que possa ser simulado por cálculo-λ ou por uma Máquina de Turing não consegue resolver o *Entscheidungsproblem*. Pouco

depois, Church, Kleene, Rosser, e Turing mostraram que estes dois modelos de computação são equivalentes.

Tese de Church-Turing. O modelo de Máquinas de Turing (MTs), que estudaremos no resto desta cadeira, rapidamente ganhou lugar como a "definição correcta da computação". A *tese de Church-Turing* captura exactamente isto – uma função é calculada por algum processo "fisicamente realizável" no mundo real se e só se pode ser computada por uma MT.[1] Ao longo dos últimos quase 90 anos, o modelo de MTs tem-se revelado extremamente robusto. Por exemplo, qualquer programa escrito em qualquer linguagem de programação existente pode ser computado por uma MT. Muita da teoria da computação moderna, incluindo grande parte da teoria da complexidade e da teoria da criptografia, é desenvolvida com base no modelo das MTs.

Se acreditarmos na tese de Church-Turing, então a inexistência de uma MT para resolver um dado problema mostra que este problema está fora do alcance de *qualquer* processo computacional que consigamos conjurar! O Entscheidungsproblem é um exemplo de tal problema. O 10º problema de Hilbert só foi mostrado ser indecidível por MTs já em 1970 através de trabalho brilhante de Davis, Matiyasevich, Putnam, e Robinson ao longo de mais de 20 anos. Nas próximas aulas aprofundaremos o estudo da teoria da computabilidade com MTs, e veremos mais exemplos de problemas que estão fora do alcance de qualquer computador.

10.2 Máquinas de Turing

A base de uma MT é intuitiva, e captura a computação humana de forma natural. Uma MT consiste numa fita que se estende infinitamente para direita e numa cabeça que se move ao longo da fita. Esta cabeça pode ler, apagar, e escrever símbolos na fita, que inicialmente contém apenas o input e também funciona como memória ilimitada. Antes do início da computação, o input finito encontra-se nos quadrados mais à esquerda da fita, os restantes quadrados à direita do input estão em branco, e a cabeça da MT aponta para o primeiro quadrado da fita (o quadrado mais à esquerda). Usaremos o símbolo $\sqcup$ para denotar um quadrado em branco.

A MT tem também uma *central processing unit* (CPU) que pode estar num de um número finito de estados possíveis. A MT actualiza o seu estado baseada no símbolo que a cabeça lê, e, simultaneamente, dá-lhe ordens como "apaga o símbolo nesse quadrado da fita e move-te para a direita", ou "escreve o símbolo a no quadrado e move-te para a esquerda". Consideramos que a fita não se estende para a esquerda do input. Se a cabeça está a apontar para o primeiro quadrado da fita e a MT ordena que esta se mova para a esquerda, a cabeça simplesmente não se move.

Existem dois estados especiais – um de aceitação e um de rejeição. A MT pára quando entra num destes estados, e aceita ou rejeita o input baseada no estado em que termina a computação. Apesar de MTs poderem funcionar como autênticos computadores, computando funções mais complexas, vamos focar-nos somente na resolução de problemas de decisão.

[1] Existem versões mais ambiciosas desta tese que têm em conta a eficiência da computação, tal como a "Extended Church-Turing thesis", que afirma que qualquer modelo de computação pode ser "eficientemente simulado" por MTs. Esta versão entra em possível conflicto com o modelo de computação quântica, tendo levado à consideração da "Quantum Extended Church-Turing thesis". Para exemplos de discussões interessantes sobre estas teses, leiam 1 e 2.

Tanto MTs como AFDs consistem num CPU que vai actualizando o seu estado (de entre um conjunto finito de possibilidades) à medida que a computação progride. No entanto, existem diferenças muito importantes:

- A cabeça de uma MT pode mover-se para a esquerda ou para a direita. Uma MT não tem de ler o input de forma sequencial.
- Uma MT pode apagar, guardar, e consultar informação na fita (infinita).
- Sempre que uma MT entra no estado de aceitação ou no estado de rejeição, não é possível sair destes estados.
- Uma MT pode parar e aceitar o input, parar e rejeitar o input, *ou então entrar num loop e não parar*.

Antes de definirmos MTs formalmente, discutimos alguns exemplos. Seja $L \subseteq \{a, b\}^*$ a linguagem das sequências que contêm aa como substring. Gostaríamos de desenhar uma MT M_1 que reconhece esta linguagem. Intuitivamente, a MT deve ler o input da esquerda para a direita, e, quando vir dois a's consecutivos, deve passar para o estado de aceitação e parar. No entanto, se entretanto chegar a um quadrado em branco sem encontrar dois a's consecutivos, deve passar para o estado de rejeição e parar. Tal como para autómatos, podemos descrever a MT pretendida através de um diagrama de estados, ilustrado na Figura 10.1.

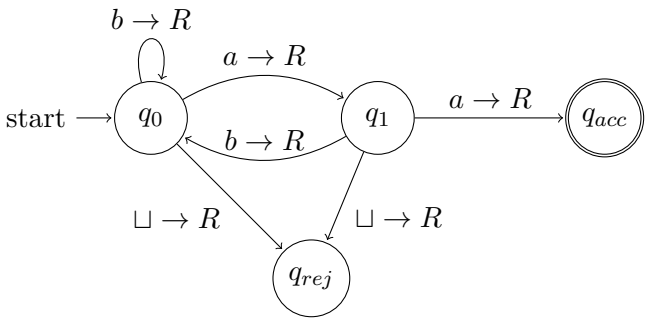

Figure 10.1: Diagrama de estados da MT M_1.

Passamos a explicar como interpretar o diagrama da Figura 10.1. Relembramos que antes do início da computação o input w está escritos nos primeiros quadrados da fita, e a cabeça aponta para o primeiro símbolo de w. Segundo o diagrama, o estado inicial da nossa MT é q_0. Concretamente, seja $w = ababaab$. Podemos representar a *configuração* inicial da nossa computação como

$$q_0 ababaab \sqcup \sqcup \ldots$$

o que quer dizer que a MT está no estado q_0 e com a cabeça a apontar para o símbolo imediatamente à direita de q_0. Por norma, vamos omitir os espaços em branco à direita da fita, e escrever

$$q_0 ababaab$$

para denotar a configuração inicial. Como a cabeça lê um a no estado q_0, seguimos o diagrama e aplicamos a regra $a \to R$, que significa "ao ler a, não escrever nada e mover a cabeça um passo para a direita". Actualizamos também o estado da MT para q_1. A configuração passa a ser

$$aq_1babaab.$$

Continuando a seguir o diagrama, obtemos as seguintes configurações

$$\begin{aligned} &abq_0abaab \\ &abaq_1baab \\ &ababq_0aab \\ &ababaq_1ab \\ &ababaaq_{acc}b. \end{aligned}$$

Concluímos que a MT passa para o estado de aceitação e pára imediatamente. Portanto, a MT aceita o input $w = ababaab$.

Consideramos agora um exemplo mais complicado. Queremos descrever uma MT que reconheça a linguagem $L = \{a^n \# b^n \mid n \in \mathbb{N}\}$, que não é regular. Intuitivamente, a MT não consegue contar números arbitrariamente grandes (ou seja, não consegue lembrar-se de quantos a's viu na primeira metade), pois o número de estados possíveis é limitado. No entanto, pode ir "marcando" os a's que já viu, e, por cada a que marca, mover-se para a metade dos b's para marcar um b. Se todos os b's já estiverem marcados, isto quer dizer que existem mais a's dos que b's, e a MT rejeita. Caso contrário, volta à metade dos a's e repete o processo até não existir mais nenhum a por marcar. Se nesta altura ainda existir algum b por marcar, então a MT rejeita, pois significa que existem mais b's do que a's. Um diagrama de estados (algo complexo!) que descreve esta MT M_2 está ilustrado na Figura 10.2.

Existem algumas diferenças no diagrama de estados da Figura 10.2. Primeiro, removemos o estado de rejeição e todas as transições para este estado. Como para autómatos, assumimos que se a cabeça ler um símbolo para qual não existe transição definida no estado actual da MT, então esta rejeita o input e pára imediatamente. Segundo, também temos transições da forma "$a \to \mathtt{x}, R$", que devem ser interpretadas como "se leres um a, substitui por um `x` e move-te um quadrado para a direita". Terceiro, também temos transições da forma $a \to L$, que significam "se leres um a, não escrevas nada e move-te um quadrado para a esquerda". A interpretação das transições do tipo $\alpha \to \beta, L$ é análogo ao caso $\alpha \to \beta, R$, só que a cabeça move-se para a esquerda.

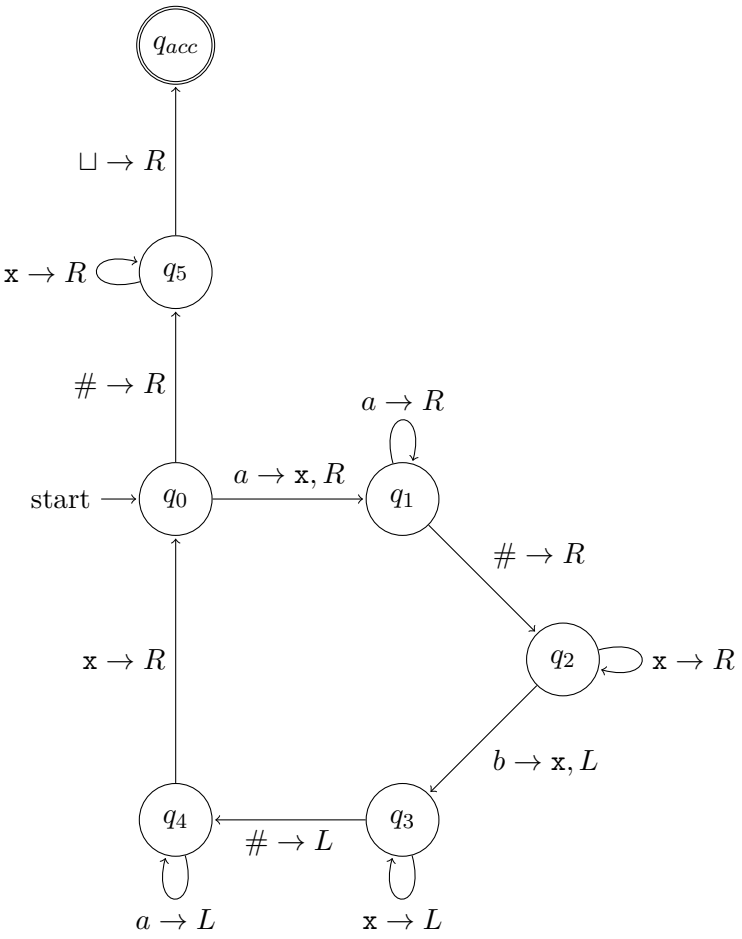

Figure 10.2: Diagrama de estados da MT M_2.

Apresentamos um snapshot parcial das configurações da MT M_2 no input $w = aa\#bb$:

$$
\begin{array}{l}
q_0aa\#bb \\
\mathtt{x}q_1a\#bb \\
\cdots \\
\mathtt{x}a\#q_2bb \\
\mathtt{x}aq_3\#\mathtt{x}b \\
\cdots \\
q_4\mathtt{x}a\#\mathtt{x}b \\
\mathtt{x}q_0a\#\mathtt{x}b \\
\cdots \\
\mathtt{xx}\#\mathtt{x}q_2b \\
\mathtt{xx}\#q_3\mathtt{xx} \\
\cdots \\
\mathtt{xx}q_0\#\mathtt{xx} \\
\cdots \\
\mathtt{xx}\#\mathtt{xx} \sqcup q_{acc}
\end{array}
$$

10.2.1 Definição formal

Apresentamos abaixo a definição formal de uma MT, bem como outras definições úteis.

Definição 10.1 (Máquina de Turing) *Uma* máquina de Turing M *é dada por um tuplo* $M = (S, \Sigma, \Gamma, \delta, s, q_{acc}, q_{rej})$*, onde:*

- S *é o conjunto de estados;*
- Σ *é o alfabeto de input, tal que* $\sqcup \notin \Sigma$*;*
- Γ *é o alfabeto da fita, tal que* $\sqcup \in \Gamma$ *e* $\Sigma \subseteq \Gamma$*;*
- $\delta : S \times \Gamma \to S \times \Gamma \times \{L, R\}$ *é a função de transição;*
- s *é o estado inicial;*
- q_{acc} *e* q_{rej} *são os estados de aceitação e rejeição, respectivamente.*

Já introduzimos acima o conceito de *configurações*, que servem para representar a fita, a posição da cabeça, e o estado da MT num dado ponto da computação. Discutimos este conceito mais formalmente agora.

Definição 10.2 *Seja* $M = (S, \Sigma, \Gamma, \delta, s, q_{acc}, q_{rej})$ *uma MT qualquer. Dizemos que a configuração* $uaqbv$ *com* $u, v \in \Gamma^*$*,* $a, b \in \Gamma$*, e* $q \in S$ *resulta na configuração* $uacq'v$ *se* $\delta(q, b) = (q', c, R)$*. Análogamente, dizemos que a configuração* $uaqbv$ *com* $u, v \in \Gamma^*$*,* $a, b \in \Gamma$*, e* $q \in S$ *resulta na configuração* $uq'acv$ *se* $\delta(q, b) = (q', c, L)$*.*

Definição 10.3 (Aceitação e rejeição) *Seja* $M = (S, \Sigma, \Gamma, \delta, s, q_{acc}, q_{rej})$ *uma MT qualquer. Dizemos que* M aceita *o input* $w \in \Sigma^*$ *se existe uma sequência de configurações* $C_0, C_1, \ldots, C_m$ *tal que:*

- $C_0 = sw$ *é a configuração inicial de* M*;*
- C_i *resulta em* C_{i+1} *para todo o* $i \in \{0, 1, \ldots, m-1\}$*;*
- *O estado de* M *na configuração final* C_m *é* q_{acc}*.*

Análogamente, dizemos que M rejeita w *se o estado de* M *na configuração* C_m *é* q_{rej}*.*

10.3 Nas próximas notas/aulas

Definir MTs formalmente não é a componente principal das coisas realmente importantes que queremos discutir. No entanto, é importante treinar a definição formal de algumas MTs para

ficarmos confortáveis com a discussão destas máquinas a um nível mais alto. Nas próximas aulas não nos vamos focar em descrições formais de MTs. Vamos apenas descrevê-las a alto nível (tal como escrever pseudocódigo de um algoritmo para ser implementado em Java), estando confiantes de que, se quiséssemos, poderíamos implementar o algoritmo em Java/C/C++/Python/etc, e por isso também numa MT.

10.4 Para explorar

Aconselhamos a leitura de [Sip13, Chapter 3].

Se quiserem uma visão mais aprofundada da contribuição original de Alan Turing, recomendamos o livro de Petzold [Pet08]. As página da Wikipedia sobre Turing-completeness e a tese de Church-Turing são interessantes.

References

[Chu36] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, 1936.

[Pet08] Charles Petzold. *The Annotated Turing: A guided tour through Alan Turing's historic paper on computability and the Turing machine.* Wiley Publishing, 2008.

[Sip13] Michael Sipser. *Introduction to the Theory of Computation.* CEngage Learning, 3rd edition, 2013.

[Tur37] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.