Funções Computáveis e Máquinas de Turing

Computabilidade e Complexidade

Márcio Nicolau

2025-08-25

Table of contents

List of Figures

Introdução: O que significa "Computar"?

Nas aulas anteriores, fizemos uma descoberta chocante: existem infinitamente mais "problemas" (linguagens) do que "soluções" (programas). Isso implica que alguns problemas são fundamentalmente insolúveis. Mas essa conclusão nos deixa com uma pergunta central: o que, exatamente, significa que um problema *é* solucionável? O que é um "algoritmo"?

Nossa noção intuitiva de um algoritmo vem da nossa experiência com programação. Um algoritmo é uma sequência de passos bem definidos que resolve uma tarefa.

```python
def is_prime(n: int) -> bool:
    """Verifica se um número é primo."""
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

print(f"O número 29 é primo? {is_prime(29)}")
print(f"O número 30 é primo? {is_prime(30)}")
```

```
O número 29 é primo? True
O número 30 é primo? False
```

Este código Python é um algoritmo. Ele tem um conjunto finito de instruções, é preciso e sempre termina com uma resposta "sim" ou "não" (True/False). Mas para estudar os limites da computação de forma rigorosa, precisamos de um modelo matemático de um "algoritmo", que seja independente de qualquer linguagem de programação específica.

É aqui que entra Alan Turing. Em 1936, ele propôs um dispositivo teórico, a **Máquina de Turing**, para formalizar a ideia de um "procedimento mecânico". Este modelo simples, porém poderoso, tornou-se a base da teoria da computação.

O objetivo da aula de hoje é: > **Definir funções computáveis e introduzir o modelo de Máquina de Turing como a formalização de um algoritmo.**

O Modelo da Máquina de Turing (TM)

Imagine o processo de computação mais simples possível. Você tem uma longa fita de papel, um lápis com borracha e um conjunto de regras. A Máquina de Turing é a formalização dessa ideia.

Ela consiste em três partes principais:

1. **A Fita (Tape):** Uma fita infinitamente longa, dividida em células. Cada célula pode conter um único símbolo de um alfabeto. A fita armazena a entrada, os cálculos intermediários e a saída.
2. **O Cabeçote (Head):** Um dispositivo que pode ler o símbolo na célula atual, escrever um novo símbolo no lugar e se mover uma célula para a esquerda (L) ou para a direita (R).
3. **A Unidade de Controle (State Control):** Um "cérebro" com um número finito de estados internos. Em qualquer momento, a máquina está em um dos seus estados. O estado atual dita o que fazer a seguir, como uma memória de trabalho.

A "programação" de uma Máquina de Turing é sua **função de transição**, que é um conjunto de regras do tipo:

> "Se você está no **estado A** e o símbolo sob o cabeçote é **X**, então:

1. Escreva o símbolo **Y**.
2. Mova o cabeçote para a **Direita (R)**.
3. Mude para o **estado B**."

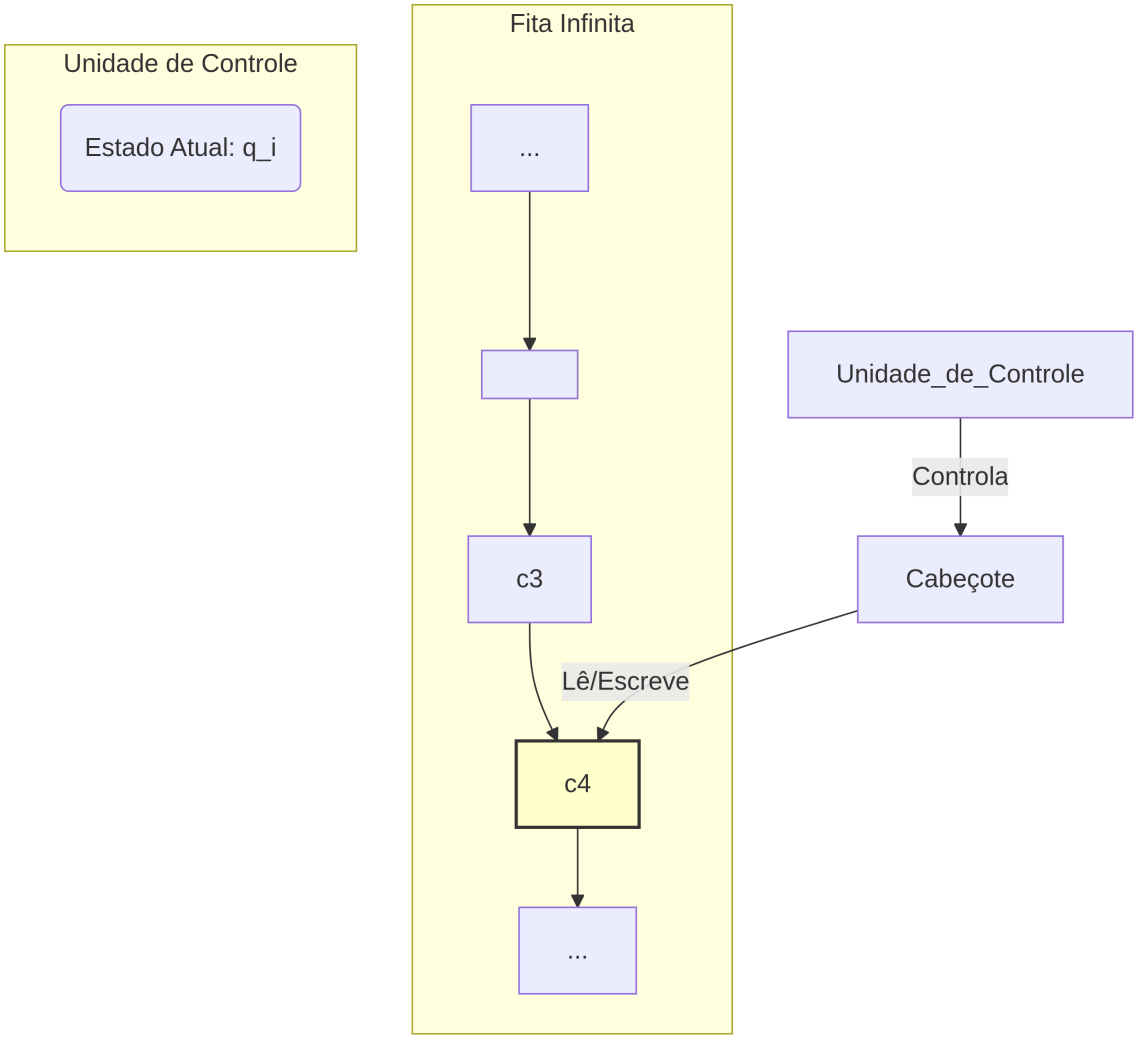

Figure 1: Componentes de uma Máquina de Turing.

Definição Formal

Formalmente, uma Máquina de Turing é uma 7-tupla $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$, onde:

- Q: Um conjunto finito de **estados**.

- Σ: O **alfabeto de entrada**. Os símbolos que podem estar na fita inicialmente. Não contém o símbolo "em branco".
- Γ: O **alfabeto da fita**. Inclui todos os símbolos de Σ mais o símbolo "em branco" ($\sqcup$).
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$: A **função de transição**. É o programa da máquina.
- $q_0 \in Q$: O **estado inicial**.
- $q_{accept} \in Q$: O **estado de aceitação**. Se a máquina entra neste estado, ela para e aceita a entrada.
- $q_{reject} \in Q$: O **estado de rejeição**. Se a máquina entra neste estado, ela para e rejeita a entrada.

(Definição baseada em (Sipser, 2012, p. 168)).

Como uma Máquina de Turing Computa

Quando uma Máquina de Turing (TM) recebe uma string de entrada `w`, ela a escreve no início da fita, posiciona o cabeçote no primeiro símbolo e inicia no estado q_0. A partir daí, a função de transição δ assume o controle. Existem três resultados possíveis para essa computação:

1. **Aceita:** A máquina eventualmente entra no estado q_{accept} e para. Dizemos que a TM **aceita** a string `w`.
2. **Rejeita:** A máquina eventualmente entra no estado q_{reject} e para. Dizemos que a TM **rejeita** a string `w`.
3. **Entra em Loop:** A máquina nunca para. Ela continua executando as transições indefinidamente.

Essa distinção nos leva a duas definições cruciais de "solucionabilidade":

Linguagem Turing-Reconhecível: Uma linguagem L é dita **reconhecível** se existe uma TM M tal que:

- Se $w \in L$, M aceita w.
- Se $w \notin L$, M ou rejeita w ou entra em loop.

Um reconhecedor nos dá uma resposta "sim" garantida para strings na linguagem, mas pode não nos dar uma resposta definitiva para strings fora dela.

Linguagem Turing-Decidível: Uma linguagem L é dita **decidível** se existe uma TM M que **para em todas as entradas**:

- Se $w \in L$, M aceita w.
- Se $w \notin L$, M rejeita w.

*Uma TM que decide uma linguagem é chamada de **decisor**. Ela sempre para e nos dá uma resposta clara de "sim" ou "não".*

Problemas **decidíveis** são o que consideramos "efetivamente solucionáveis" por um computador.

Funções Computáveis

Além de decidir linguagens (responder sim/não), as TMs podem calcular funções.

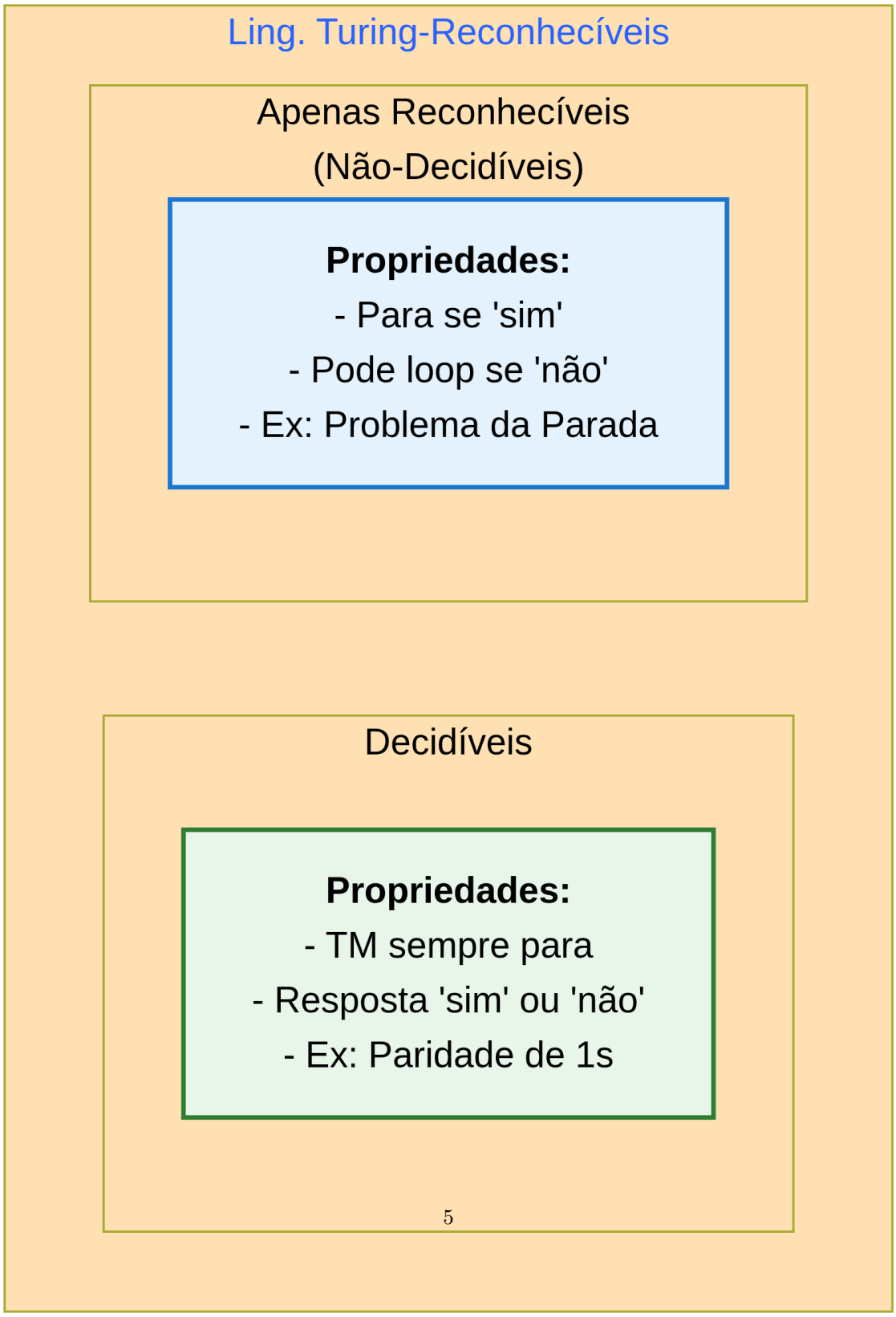

Figure 2: Partição das linguagens reconhecíveis em decidíveis e não-decidíveis.

> **!** Definição
>
> Uma função $f : \Sigma^* \to \Gamma^*$ é uma **função computável** (ou Turing-computável) se existe uma Máquina de Turing M que, para toda entrada $w \in \Sigma^*$, para com a string $f(w)$ na fita e nada mais.

Por exemplo, vamos considerar uma função simples: $f(w) = w$ concatenado com "01". Se a entrada for "110", a saída deve ser "11001".

Uma TM para computar essa função faria o seguinte:

1. Começando no estado q_0 no início da string de entrada.
2. Mova o cabeçote para a direita, passando por todos os símbolos da entrada, até encontrar o primeiro símbolo em branco ($\sqcup$).
3. Quando encontrar $\sqcup$, escreva '0'.
4. Mova para a direita.
5. Escreva '1'.
6. Pare. (Podemos definir um estado final q_{halt} para isso).

Isso demonstra que uma operação simples, que em Python seria `w + "01"`, corresponde a um procedimento mecânico que pode ser executado por uma Máquina de Turing.

A Tese de Church-Turing

Isso nos leva a uma das hipóteses mais importantes da ciência da computação. A **Tese de Church-Turing** não é um teorema que pode ser provado, mas uma afirmação que tem resistido ao teste do tempo.

> **Tese de Church-Turing:** A noção intuitiva de algoritmo é equivalente à noção de uma Máquina de Turing. ((Sipser, 2012, p. 182)).

Isso significa que qualquer coisa que possa ser "computada" por *qualquer* modelo de computação razoável (um programa Python, cálculo lambda, um computador quântico idealizado) pode também ser computada por uma Máquina de Turing.

> **i** Implicação prática
>
> Se quisermos provar que um problema é "insolúvel", basta provar que nenhuma Máquina de Turing pode decidí-lo. A Tese de Church-Turing nos dá a confiança de que nenhum avanço em linguagens de programação ou arquitetura de computadores tornará esse problema solúvel.

Verificação de Aprendizagem

1: Conceitual

Explique a diferença fundamental entre uma linguagem **Turing-reconhecível** e uma **Turing-decidível**. Por que, do ponto de vista prático de resolver um problema, a propriedade de ser "decidível" é muito mais forte e desejável?

2: Projeto de Alto Nível de uma TM

Descreva em alto nível (sem a tupla formal, apenas a estratégia e os passos) como uma Máquina de Turing poderia **decidir** a linguagem $L = \{w \mid w \text{ é uma string binária com um número par de 1s}\}$.

> Dica
>
> Pense em como você resolveria isso com "memória". Como os estados da TM podem servir como essa memória?

3: Conectando com Python (Tese de Church-Turing)

Considere a seguinte função em Python que decide a linguagem da Questão 2.

```python
def numero_par_de_uns(w: str) -> bool:
    """Decide se a string w tem um número par de 1s."""
    count = 0
    for char in w:
        if char == '1':
            count += 1
    return count % 2 == 0
```

Com base na Tese de Church-Turing, o que a existência deste programa Python nos permite concluir sobre a linguagem L? A linguagem L é decidível, reconhecível ou nenhuma das duas? Justifique sua resposta.

Referências Bibliográficas

SIPSER, Michael. **Introdução à Teoria da Computação**. 3. ed. São Paulo, Brasil: Cengage Learning, 2012.