

Reforço: Máquinas de Turing e Linguagens

Computabilidade e Complexidade

Márcio Nicolau

2025-09-08

Table of contents

Objetivos da Aula	1
Conteúdo	1
A Máquina de Turing como um “Programa”	1
Guia Prático para Classificar Linguagens	2
Análise de Casos	4
Implicações em Código Python	4
Exercícios de Verificação	6
Referências Bibliográficas	6

List of Figures

1 Hierarquia de Classes de Linguagens por Decidibilidade	3
--	-------------------

Objetivos da Aula

- Praticar o projeto de alto nível de Máquinas de Turing para problemas específicos.
- Solidificar a habilidade de classificar linguagens na hierarquia de decidibilidade (R, RE, não-RE).
- Conectar o comportamento teórico das TMs com problemas práticos de programação.
- Revisar o Teorema Fundamental que relaciona as classes R, RE e co-RE.

Conteúdo

A Máquina de Turing como um “Programa”

É útil pensar em uma Máquina de Turing não como um dispositivo físico, mas como a especificação mais fundamental e explícita de um algoritmo. Tudo o que um programa em Python pode fazer, uma TM também pode (Tese de Church-Turing), embora de forma muito mais detalhada.

i Componentes Essenciais (Revisão)

- **Fita Infinita:** A memória do nosso “computador”.
- **Cabeçote:** O ponteiro que lê e escreve na memória, um símbolo por vez.
- **Estados Finitos:** A “memória de trabalho” do programa, que rastreia o estado atual do algoritmo (ex: “estou procurando um 0”, “encontrei um 1, agora voltando”, etc.).
- **Função de Transição:** O “código” do programa. Um conjunto de regras **if-then-else** que ditam o que fazer a seguir com base no estado atual e no símbolo lido.

Exemplo Prático: Projetando uma TM para $L = \{0^n 1^n \mid n \geq 0\}$

Vamos projetar, em alto nível, um **decisor** para esta linguagem clássica. A TM precisa verificar se há um número igual de 0s e 1s, e se todos os 0s vêm antes de todos os 1s. (Sipser, 2012)

Estratégia do Algoritmo (TM):

1. **Varredura Inicial:** Comece no início da fita. Se a primeira célula estiver em branco, aceite (caso da string vazia, $n = 0$). Se for um ‘1’, rejeite imediatamente.
2. **Marcar um ‘0’:** Se for um ‘0’, substitua-o por um símbolo de marcação, como ‘X’, e mude para um estado que significa “procurando um 1 correspondente”.
3. **Procurar por ‘1’:** Mova o cabeçote para a direita, ignorando outros ‘0’s e ‘Y’s (outra marcação), até encontrar um ‘1’.
4. **Marcar um ‘1’:** Se um ‘1’ for encontrado, substitua-o por ‘Y’ e mude para um estado que significa “voltando para o início”. Se um espaço em branco for encontrado antes de um ‘1’, rejeite (há mais 0s do que 1s).
5. **Voltar ao Início:** Mova o cabeçote para a esquerda até encontrar o ‘X’ mais à direita. Mova um passo para a direita para estar no início da próxima iteração.
6. **Repetir:** Volte ao passo 2.
7. **Verificação Final:** Se, no passo 2, em vez de um ‘0’, encontrarmos um ‘Y’, significa que todos os ‘0’s foram marcados. Agora, mude para um estado de verificação final. Mova para a direita. Se houver algum ‘1’ sobrando, rejeite (há mais 1s do que 0s). Se encontrarmos apenas ‘Y’s seguidos de um espaço em branco, **aceite**.

Este algoritmo **sempre para**, pois a cada iteração ele consome um ‘0’ e um ‘1’. Portanto, ele é um **decisor**, e a linguagem $L = \{0^n 1^n\}$ está na classe **R** (Decidível).

Guia Prático para Classificar Linguagens

A tarefa mais comum na teoria da computabilidade é, dada uma linguagem, classificá-la. A hierarquia é a chave.

Diagrama: Hierarquia de Classes de Linguagens (Revisão)

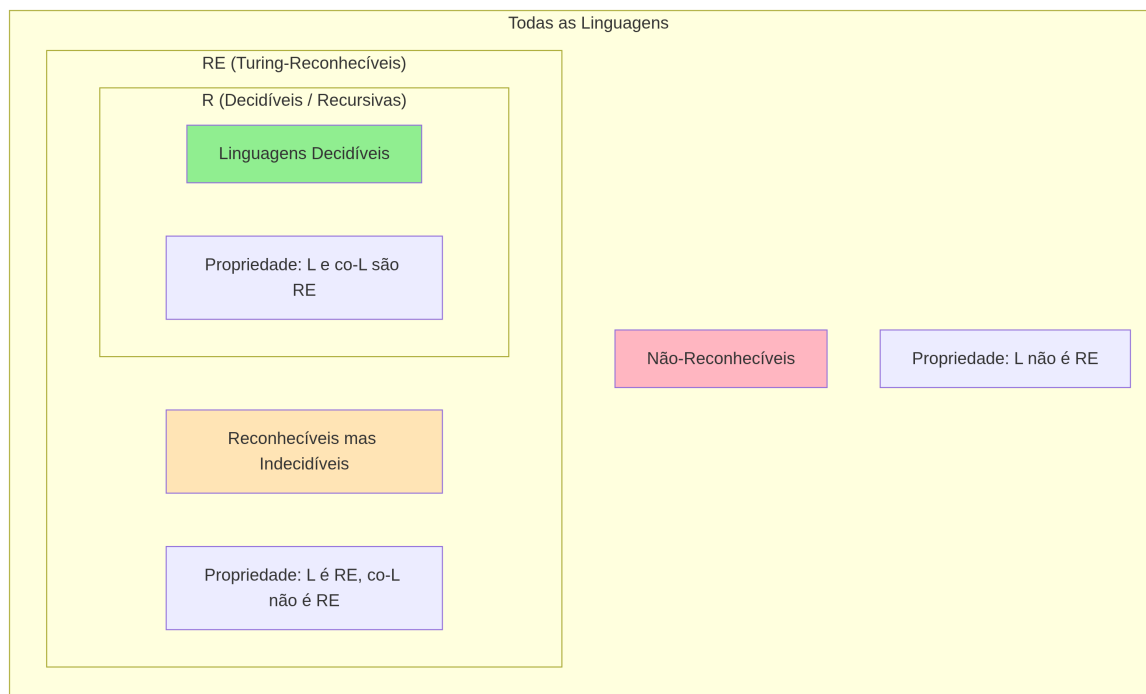


Figure 1: Hierarquia de Classes de Linguagens por Decidibilidade

! Como Classificar um Problema L

1. **Tente Construir um Reconhecedor:** É possível escrever um algoritmo que, para qualquer $w \in L$, para e diz “sim”? Pense em uma busca exaustiva. Se você pode simular ou gerar instâncias até encontrar uma que corresponda a w , então L é **RE**.
 - *Exemplo:* Para $A_{TM} = \{\langle M, w \rangle \mid M \text{ aceita } w\}$, o reconhecedor é óbvio: simule M em w .
2. **O Reconhecedor Sempre Para?:** Se o seu reconhecedor do passo 1 é garantido que para *mesmo para entradas que não estão em L* , então ele é um **decisor**, e L está em **R**.
3. **Se Não Parece Decidível, Considere o Complemento:** Pense na linguagem $\bar{L}$. Tente construir um reconhecedor para $\bar{L}$.
 - Se você conseguir construir um reconhecedor para L **E** para $\bar{L}$, então, pelo Teorema Fundamental, L é **decidível (R)**.
 - Se L é reconhecível mas você prova que $\bar{L}$ não é (muitas vezes por redução), então L está em **RE R**.
 - Se nem L nem $\bar{L}$ parecem reconhecíveis, L provavelmente não é RE nem co-RE.

Análise de Casos

Vamos aplicar nosso guia.

Caso 1: $E_{TM} = \{\langle M \rangle \mid L(M) = \emptyset\}$

- **É Reconhecível?** Não parece. Para provar que $L(M)$ é vazio, teríamos que simular M em *todas as infinitas strings possíveis* e garantir que nenhuma é aceita. Isso não pode ser feito em tempo finito.
- **E o Complemento, $\overline{E_{TM}} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$?** Este é **reconhecível**.
 - **Reconhecedor para $\overline{E_{TM}}$:** Dado $\langle M \rangle$, comece a enumerar todas as strings $w_1, w_2, w_3, \dots$ em ordem canônica. Simule M em cada uma delas em paralelo (usando uma TM multi-fitas ou técnica de “dovetailing”). Se *qualquer* uma dessas simulações aceitar, então sabemos que $L(M)$ não é vazio, e *nosso* reconhecedor para e aceita. Se $L(M)$ for de fato vazio, este processo nunca irá parar.
- **Conclusão:** $\overline{E_{TM}}$ é RE. Como já sabemos que E_{TM} é indecidível, ele não pode ser R. Portanto, E_{TM} é **co-RE mas não R**.

Implicações em Código Python

A diferença entre um decisor e um reconhecedor é análoga a funções que garantem terminar vs. funções que podem entrar em loop.

```
def decider_exemplo(lista: list, item: int) -> bool:
    """ Análogo a um DECISOR.
    Sempre termina e retorna uma resposta definitiva.
    Decide L = { (lista, item) | item está na lista }
    """
    print(f"Decidindo se {item} está em {lista}...")
    return item in lista

def reconhecer_exemplo(gerador_infinito, propriedade_func):
```

```

""" Análogo a um RECONHECEDOR.
Garante terminar se encontrar um item com a propriedade,
mas entrará em loop se nenhum item tiver a propriedade.
Reconhece L = { (gerador, prop) | existe um item no gerador com a prop }
"""

print(f"Reconhecendo se existe um número com a propriedade...")
for item in gerador_infinito:
    if propriedade_func(item):
        print(f"Encontrado: {item}!")
        return True # Termina e aceita
# Nunca chega aqui se não encontrar, entra em loop infinito

# --- Uso ---
print("--- Teste do Decisor ---")
print(f"Resultado: {decider_exemplo([2, 4, 6, 8, 10], 9)}\n")

def gerador_de_naturais():
    n = 0
    while True:
        yield n
        n += 1

print("--- Teste do Reconhecedor (caso de sucesso) ---")
# Reconhece se existe um número > 10 e divisível por 13
recognizer_exemplo(gerador_de_naturais(), lambda x: x > 10 and x % 13 == 0)

print("\n--- Teste do Reconhecedor (caso de loop) ---")
print("Tentando reconhecer se existe um número negativo...")
print("(Esta chamada nunca retornaria se executada)")
# recognizer_exemplo(gerador_de_naturais(), lambda x: x < 0)

```

```

--- Teste do Decisor ---
Decidindo se 9 está em [2, 4, 6, 8, 10]...
Resultado: False

```

```

--- Teste do Reconhecedor (caso de sucesso) ---
Reconhecendo se existe um número com a propriedade...
Encontrado: 13!

```

```

--- Teste do Reconhecedor (caso de loop) ---
Tentando reconhecer se existe um número negativo...
(Esta chamada nunca retornaria se executada)

```

Exercícios de Verificação

Atividade Prática

1. **Projeto de TM:** Descreva em alto nível uma Máquina de Turing que decide a linguagem $L = \{w \mid w \text{ é uma string binária com número ímpar de 1s}\}$. Esta TM precisa de uma “memória”? Como os estados podem ser usados para isso?
2. **Classificação:** Classifique a seguinte linguagem, justificando sua resposta: $L_{INFINITE} = \{\langle M \rangle \mid L(M) \text{ é uma linguagem infinita}\}$. $L_{INFINITE}$ está em R, RE R, co-RE R, ou nenhuma dessas?
3. **Conceitual:** Vimos que o conjunto de todas as linguagens é não-enumerável, mas o conjunto de todas as Máquinas de Turing é enumerável. Explique brevemente por que isso implica a existência de linguagens que não são sequer Turing-reconhecíveis.

Referências Bibliográficas

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