

Problemas Decidíveis e Indecidíveis

Márcio Nicolau

2025-09-22

Table of contents

Objetivos da Aula	1
Conteúdo	2
1. Classificação de Problemas Computacionais	2
2. O Teorema Fundamental da Decidibilidade	3
Exemplos Clássicos de Problemas Indecidíveis	3
Técnicas de Demonstração de Indecidibilidade	4
Aplicação Prática: Verificação de Software	5
Implicações Práticas da Indecidibilidade	6
Exercícios de Verificação	7
Diagrama: Mapa de Reduções entre Problemas Clássicos	7
Referências Bibliográficas	7

List of Figures

1	Hierarquia de Classes de Linguagens por Decidibilidade	3
2	Mapa de Reduções entre Problemas Indecidíveis Clássicos	8

Objetivos da Aula

- Classificar problemas computacionais como decidíveis ou indecidíveis
- Compreender a relação entre o Problema da Parada e a indecidibilidade
- Aplicar técnicas de redução para demonstrar indecidibilidade
- Distinguir entre problemas decidíveis, reconhecíveis e não-reconhecíveis

Conteúdo

Definições essenciais

- **Problema Decidível:** existe uma Máquina de Turing que sempre para e decide corretamente se a entrada pertence ou não à linguagem.
- **Problema Indecidível:** não existe tal Máquina de Turing. A linguagem pode ser reconhecível (RE) ou não-reconhecível.
- **Reconhecível (RE):** existe MT que aceita todas as entradas da linguagem; pode não parar para entradas fora da linguagem.
- **Co-reconhecível (co-RE):** o complemento da linguagem é reconhecível.
- **Teorema fundamental:** Uma linguagem é decidível se e somente se é reconhecível E co-reconhecível.

1. Classificação de Problemas Computacionais

De acordo com Sipser (2012), podemos classificar os problemas computacionais em uma hierarquia baseada em sua tratabilidade computacional:

1.1. Problemas Decidíveis (Classe R)

Problemas para os quais existe um algoritmo (Máquina de Turing) que sempre para e fornece a resposta correta.

Exemplos de problemas decidíveis:

- Teste de primalidade de números
- Verificação se uma gramática livre de contexto gera uma string específica
- Equivalência de autômatos finitos determinísticos
- Conectividade em grafos finitos

1.2. Problemas Reconhecíveis mas Indecidíveis (Classe RE $\setminus$ R)

Problemas para os quais existe um algoritmo que aceita todas as instâncias positivas, mas pode não parar para instâncias negativas.

Exemplo principal: - $A_{TM} = \{\langle M, w \rangle \mid M \text{ aceita } w\}$ (Problema da Parada - versão aceitação)

1.3. Problemas Não-Reconhecíveis (Fora de RE)

Problemas para os quais não existe sequer um reconhecedor.

Exemplo principal:

- $\overline{A_{TM}} = \{\langle M, w \rangle \mid M \text{ não aceita } w\}$ (Complemento do Problema da Parada)

2. O Teorema Fundamental da Decidibilidade

! Teorema Central

Teorema: Uma linguagem L é decidível se e somente se L é reconhecível e $\bar{L}$ (complemento de L) é reconhecível.

Prova (esboço):

- $(\Rightarrow)$: Se L é decidível, então L e $\bar{L}$ são claramente reconhecíveis.
- $(\Leftarrow)$: Se L é reconhecível por M_1 e $\bar{L}$ é reconhecível por M_2 , construa um decisor que executa M_1 e M_2 em paralelo. Uma das duas sempre aceita, determinando se $w \in L$ ou $w \in \bar{L}$.

Diagrama: Hierarquia de Classes de Linguagens

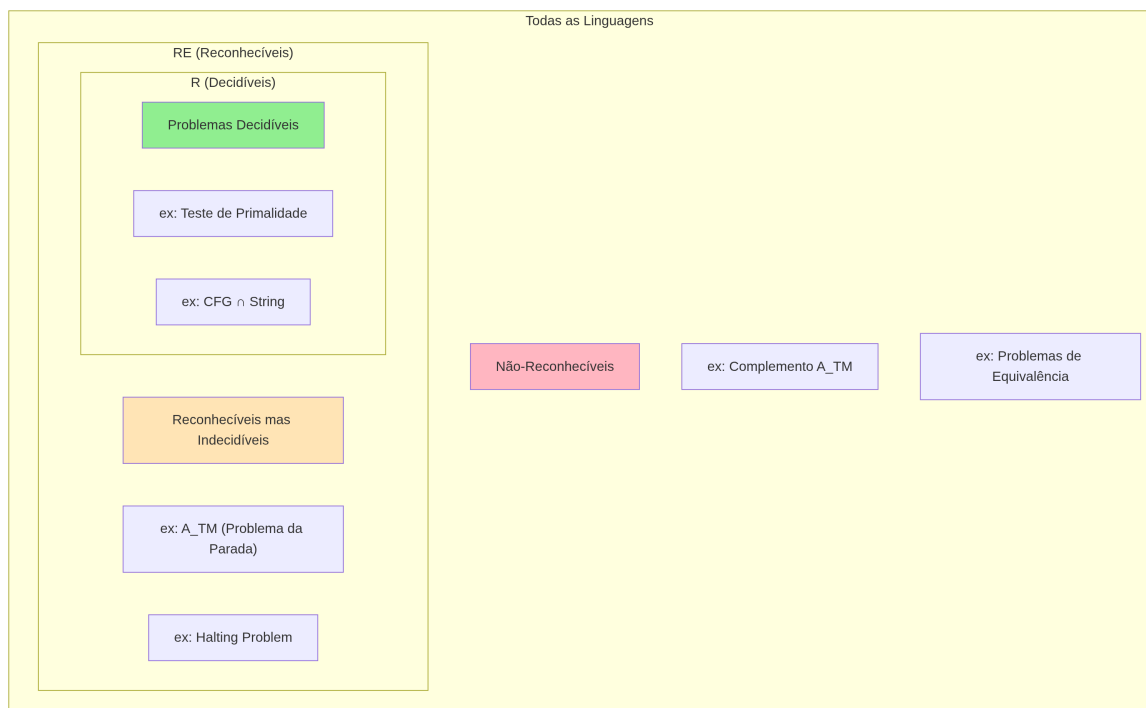


Figure 1: Hierarquia de Classes de Linguagens por Decidibilidade

Exemplos Clássicos de Problemas Indecidíveis

O Problema da Parada (Halting Problem)

Como visto na aula anterior, o problema fundamental:

$$HALT_{TM} = \{\langle M, w \rangle \mid M \text{ para na entrada } w\}$$

Por que é indecidível? A prova por diagonalização de Turing mostra que assumir a existência de um decisor leva a uma contradição lógica.

Problema da Aceitação (A_{TM})

$$A_{TM} = \{\langle M, w \rangle \mid M \text{ aceita } w\}$$

- **Status:** Reconhecível mas indecidível
- **Reconhecedor:** Simule M em w ; aceite se M aceita
- **Por que indecidível?** Redução do Problema da Parada

Problema do Estado Vazio

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

Decide se uma Máquina de Turing aceita alguma string.

Prova de indecidibilidade por redução:

- Assuma que E_{TM} é decidível
- Construa um decisor para A_{TM} usando o decisor de E_{TM}
- Contradição!

Problema da Equivalência

$$EQ_{TM} = \{\langle M_1, M_2 \rangle \mid L(M_1) = L(M_2)\}$$

Status: Nem reconhecível nem co-reconhecível

Técnicas de Demonstração de Indecidibilidade

Diagonalização

Técnica direta usada para provar que A_{TM} é indecidível. Constrói-se uma máquina “antagonista” que contradiz qualquer suposto decisor.

Redução (Redutibilidade)

Definição: Uma linguagem A é **redutível** a uma linguagem B (escrito $A \leq_m B$) se existe uma função computável f tal que:

$$w \in A \Leftrightarrow f(w) \in B$$

Teorema da Redução: Se $A \leq_m B$ e B é decidível, então A é decidível.

Contraposição: Se $A \leq_m B$ e A é indecidível, então B é indecidível.

Exemplo de Redução: $A_{TM} \leq_m HALT_{TM}$

Vamos mostrar que o Problema da Parada é indecidível usando redução.

Construção da redução:

Dado $\langle M, w \rangle$, construímos $\langle M', w \rangle$ onde M' é uma modificação de M :

$M'(w)$:

1. Simule M em w
2. Se M aceita, aceite
3. Se M rejeita, entre em loop infinito

Análise:

- Se M aceita w : M' para (aceitando) $\langle M', w \rangle \in HALT_{TM}$
- Se M não aceita w : M' não para $\langle M', w \rangle \notin HALT_{TM}$

Logo, $\langle M, w \rangle \in A_{TM} \Leftrightarrow \langle M', w \rangle \in HALT_{TM}$

Aplicação Prática: Verificação de Software

```
def analisador_estatico_limitado(codigo, max_loops=1000):
    """
    Simulação de um analisador estático que tenta detectar loops infinitos.
    Limitação: só pode detectar loops 'óbvios' ou usar timeout.
    """
    # Contadores para detectar padrões suspeitos
    loop_counter = 0

    # Simula análise do código (muito simplificada)
    linhas = codigo.split('\n')

    for linha in linhas:
        if 'while True:' in linha and 'break' not in codigo:
            return "LOOP INFINITO DETECTADO (óbvio)"
        if 'for' in linha or 'while' in linha:
            loop_counter += 1

    if loop_counter > 3:
        return "POSSÍVEL COMPLEXIDADE ALTA (heurística)"

    return "ANÁLISE INCONCLUSIVA - não pode garantir terminação"

# Exemplo de uso
codigo_suspeito = """
def funcao_problematizada(n):
    while n > 0:
        n = n + 1 # Oops! Loop infinito

```

```

        return n
    """

codigo_complexo = """
def funcao_complexa(n):
    while condition_based_on_input(n):
        n = transform(n)
        if complex_condition(n):
            break
    return n
"""

print("Análise do código suspeito:")
print(analisador_estatico_limitado(codigo_suspeito))
print("\nAnálise do código complexo:")
print(analisador_estatico_limitado(codigo_complexo))
print("\nConclusão: Análise completa é impossível devido à indecidibilidade!")

```

Análise do código suspeito:
 ANÁLISE INCONCLUSIVA - não pode garantir terminação

Análise do código complexo:
 ANÁLISE INCONCLUSIVA - não pode garantir terminação

Conclusão: Análise completa é impossível devido à indecidibilidade!

Implicações Práticas da Indecidibilidade

Limitações dos Compiladores

- Impossível detectar todo código morto
- Impossível otimizar perfeitamente todos os programas
- Análise estática tem limites fundamentais

Verificação Formal

- Verificação automática completa é impossível
- Necessidade de especificações e invariantes
- Ferramentas semi-automáticas são o máximo possível

Segurança de Software

- Impossível verificar automaticamente se um programa é “seguro”
- Análise de malware tem limitações teóricas
- Necessidade de abordagens heurísticas

Exercícios de Verificação

Atividade Prática

1. **Classificação:** Determine se os seguintes problemas são decidíveis, reconhecíveis mas indecidíveis, ou não-reconhecíveis:
 - $L_1 = \{\langle M \rangle \mid M \text{ aceita pelo menos 100 strings}\}$
 - $L_2 = \{\langle M \rangle \mid M \text{ aceita exatamente 42 strings}\}$
 - $L_3 = \{\langle M_1, M_2 \rangle \mid L(M_1) \cap L(M_2) = \emptyset\}$
2. **Redução:** Mostre que $E_{TM} \leq_m \overline{A_{TM}}$ (o complemento do problema de aceitação).
3. **Construção:** Dado que A_{TM} é indecidível, prove que $\overline{A_{TM}}$ não é reconhecível.

Diagrama: Mapa de Reduções entre Problemas Clássicos

Referências Bibliográficas

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

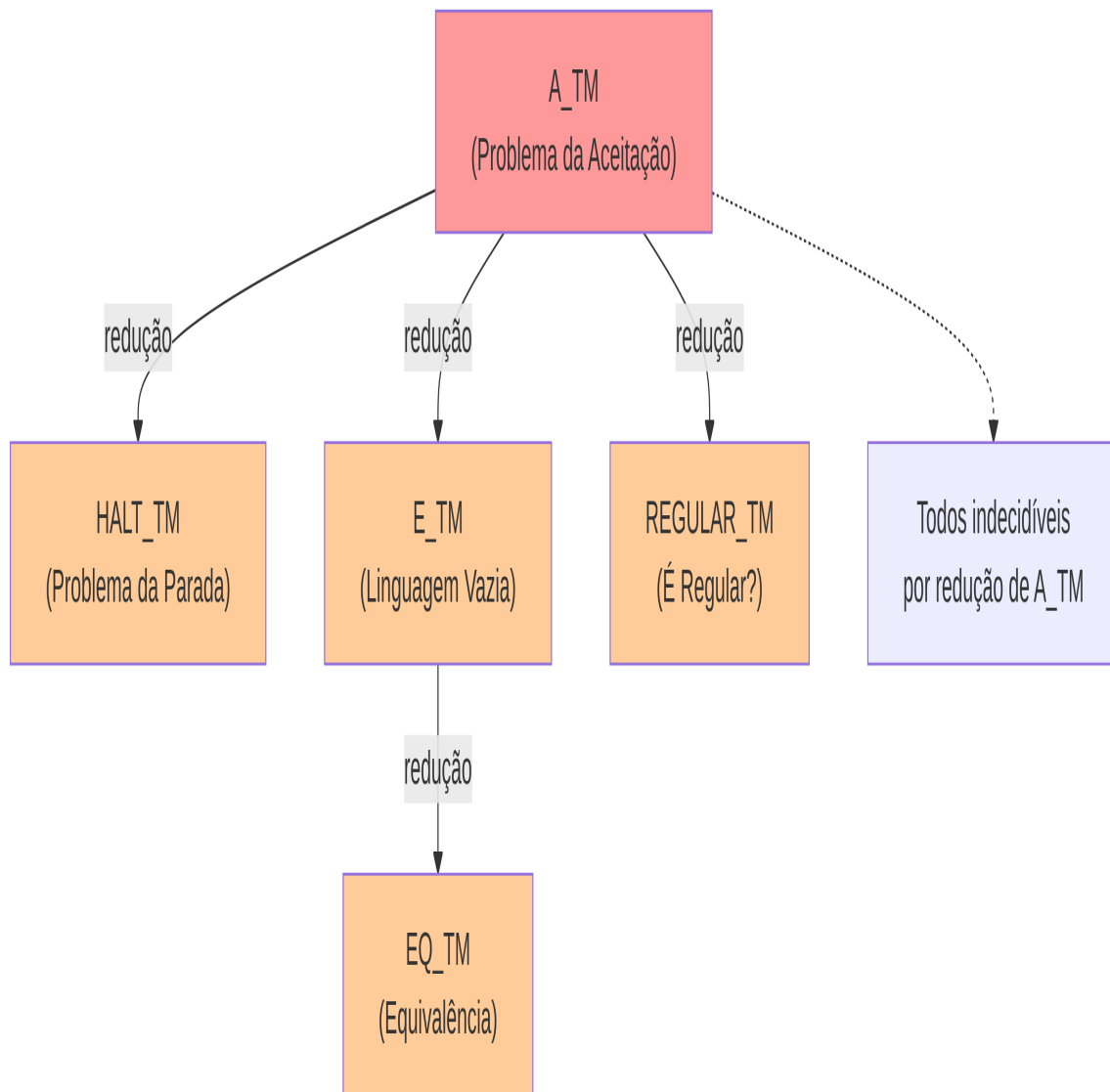


Figure 2: Mapa de Reduções entre Problemas Indecidíveis Clássicos