

Redutibilidade e Completude

Computabilidade e Complexidade

Márcio Nicolau

2025-10-20

Table of contents

Objetivos da Aula	1
Conteúdo	1
O Conceito de Redutibilidade	1
Usando Reduções para Provar Indecidibilidade	2
Problemas Completos: Os “Mais Difíceis”	4
Aplicação em Código: A Ideia da Redução	4
Exercícios de Verificação	7
Referências Bibliográficas	7

List of Figures

1	A função computável ‘f’ atua como uma ponte entre o problema A e o problema B.	3
2	A _{TM} é o “cume da montanha” da classe RE. Todos os outros problemas em RE podem ser “escalados” até ele através de uma redução.	5

Objetivos da Aula

- Definir formalmente o conceito de redutibilidade entre problemas.
- Aplicar a redutibilidade como a principal ferramenta para provar a indecidibilidade de novos problemas.
- Entender o que significa um problema ser “completo” para uma classe de linguagens.
- Identificar A_{TM} como o problema canônico RE-Completo.

Conteúdo

O Conceito de Redutibilidade

Nós provamos que o problema da aceitação, A_{TM} , é indecidível usando uma complexa prova por diagonalização. Agora, para cada novo problema que suspeitamos ser indecidível, teremos que criar uma nova prova por diagonalização do zero? Felizmente, não.

A **reducibilidade** é uma técnica que nos permite usar um problema que *já sabemos* ser difícil (como A_{TM}) para provar que um novo problema também é difícil.

i A Intuição da Redução

Reduzir um problema A a um problema B significa mostrar que, se tivéssemos uma “caixa mágica” que resolve B , poderíamos usá-la para resolver A . Isso implica que B é **pelo menos tão difícil quanto** A . Se A é indecidível, B não pode ser decidível.

Definição Formal (Reducibilidade por Mapeamento):

Uma linguagem A é **reducível por mapeamento** (ou many-one redutível) à linguagem B , denotado $A \leq_m B$, se existe uma função **computável** f que transforma instâncias de A em instâncias de B , tal que para toda string w :

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

A função f é chamada de **redução** de A para B . (Sipser, 2012, p. 235)

Diagrama: Como Funciona uma Redução

Usando Reduções para Provar Indecidibilidade

A principal aplicação da reducibilidade na teoria da computabilidade é o seguinte teorema:

! Teorema da Redução para Indecidibilidade

Se $A \leq_m B$ e a linguagem A é **indecidível**, então a linguagem B também deve ser **indecidível**.

Prova (por contradição):

Suponha que B fosse decidível por uma TM M_B . Poderíamos então construir um decisor para A da seguinte forma: 1. Dada uma entrada w para o problema A . 2. Use a função computável f para transformar w em $f(w)$. 3. Execute o decisor M_B na entrada $f(w)$. 4. Se M_B aceita, aceite w . Se M_B rejeita, rejeite w .

Este procedimento seria um decisor para A . Mas nós sabemos que A é indecidível. Chegamos a uma contradição. Portanto, nossa suposição de que B era decidível deve ser falsa.

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

Vamos provar que $HALT_{TM} = \{\langle M, w \rangle \mid M \text{ para em } w\}$ é indecidível, usando A_{TM} como nosso ponto de partida.

Construção da Redução (f):

A redução f recebe uma entrada $\langle M, w \rangle$ para o problema A_{TM} e deve produzir uma saída $\langle M', w' \rangle$ para o problema $HALT_{TM}$.

Definimos $f(\langle M, w \rangle) = \langle M', w \rangle$, onde M' é uma nova TM que se comporta da seguinte maneira:

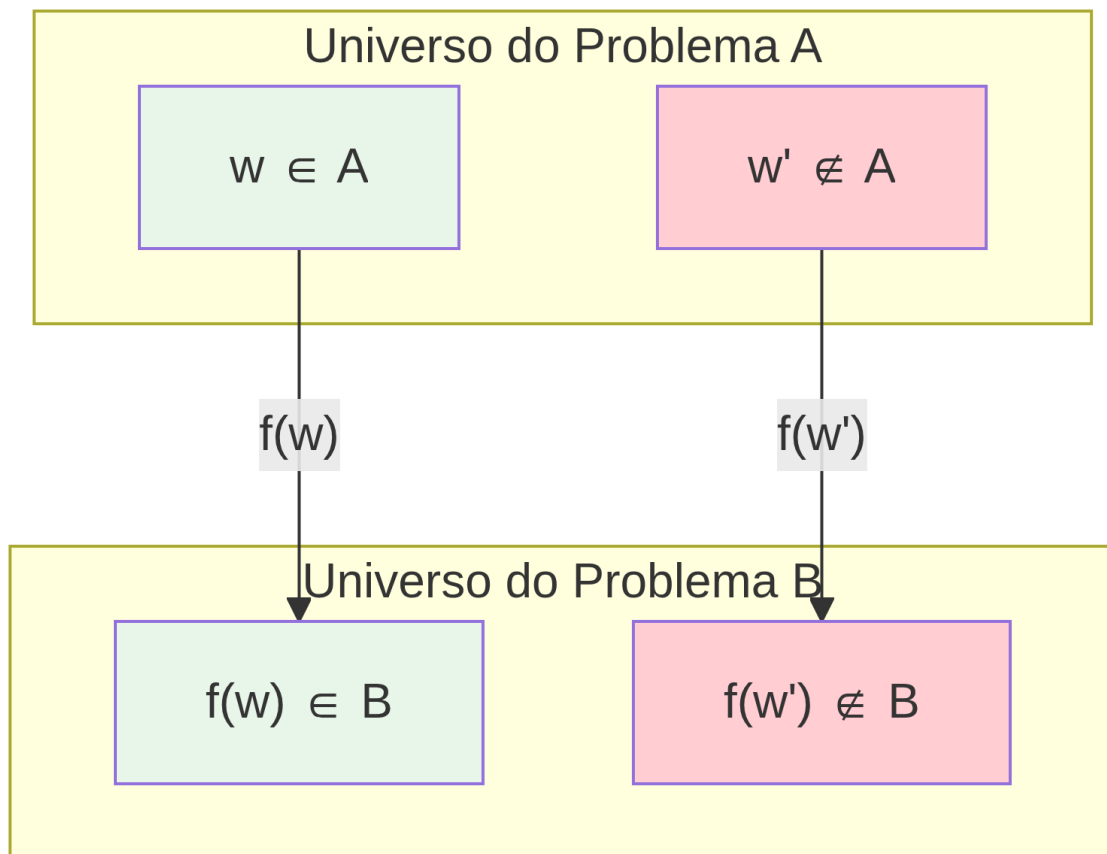


Figure 1: A função computável 'f' atua como uma ponte entre o problema A e o problema B.

$M'(x)$:

1. Execute M na entrada w (ignorando a entrada original x).
2. Se M aceitar w , então M' aceita x (e para).
3. Se M rejeitar w , então M' entra em um loop infinito.

Análise da Redução:

- Se $\langle M, w \rangle \in A_{TM}$ (ou seja, M aceita w), então a máquina M' irá parar na sua etapa 2 para qualquer entrada. Portanto, $\langle M', w \rangle \in HALT_{TM}$.
- Se $\langle M, w \rangle \notin A_{TM}$ (ou seja, M rejeita ou entra em loop em w), então a máquina M' ou entrará em loop na sua etapa 1 (se M entrar em loop) ou na sua etapa 3 (se M rejeitar). Em ambos os casos, M' não para. Portanto, $\langle M', w \rangle \notin HALT_{TM}$.

A equivalência $\langle M, w \rangle \in A_{TM} \iff \langle M', w \rangle \in HALT_{TM}$ é mantida. Como A_{TM} é indecidível, concluímos que $HALT_{TM}$ também é indecidível.

Problemas Completos: Os “Mais Difíceis”

Dentro de uma classe de complexidade (como RE), alguns problemas se destacam por serem os “mais difíceis” do grupo. Um problema é “o mais difícil” se qualquer outro problema daquela classe pode ser reduzido a ele.

Definição: Completude

Um problema B é **completo** para uma classe de linguagens $\mathcal{C}$ (ou $\mathcal{C}$ -completo) se ele satisfaz duas condições:

1. B está em $\mathcal{C}$ ($B \in \mathcal{C}$).
2. Todo problema A em $\mathcal{C}$ é redutível a B ($\forall A \in \mathcal{C}, A \leq_m B$).

A_{TM} é RE-Completo

O problema da aceitação, A_{TM} , é o exemplo canônico de um problema **RE-Completo**. (Sipser, 2012)

1. $A_{TM} \in \mathbf{RE}$: Nós já estabelecemos que A_{TM} é reconhecível por uma Máquina de Turing Universal que simula a máquina de entrada.
2. **Todo problema em RE é redutível a A_{TM}** : Seja L uma linguagem qualquer em RE. Por definição, existe uma TM, M_L , que a reconhece. A redução f de L para A_{TM} é trivial: $f(w) = \langle M_L, w \rangle$.
 - Se $w \in L$, então M_L aceita w , e, por definição, $f(w) = \langle M_L, w \rangle \in A_{TM}$.
 - Se $w \notin L$, então M_L não aceita w , e, por definição, $f(w) = \langle M_L, w \rangle \notin A_{TM}$. A redução funciona.

Diagrama: O Status de A_{TM} como RE-Completo

Aplicação em Código: A Ideia da Redução

A construção de uma redução é como escrever um “wrapper” ou uma “função de pré-processamento”. O código Python abaixo não implementa uma redução de TM (que seria muito complexo), mas ilustra a *lógica* de transformar a instância de um problema em outro.

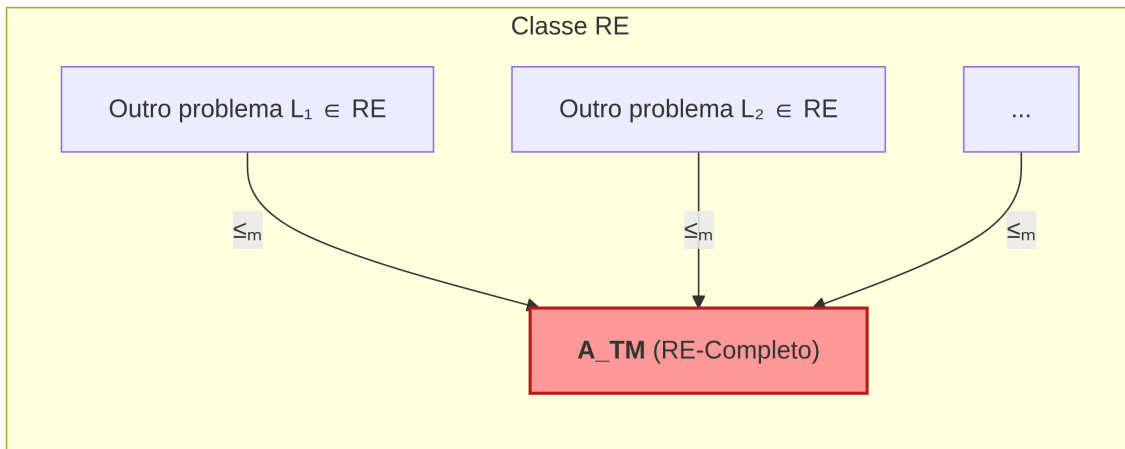


Figure 2: A_TM é o “cume da montanha” da classe RE. Todos os outros problemas em RE podem ser “escalados” até ele através de uma redução.

```
# Suponha que temos uma "caixa mágica" que resolve um problema B
def magic_solver_for_B(instancia_b):
    """Esta é a nossa hipotética caixa mágica (oráculo) para o problema B.
    Ex: B = Problema da Parada"""
    print(f" [Oráculo B] Recebeu a instância: {instancia_b}")
    # A lógica interna é desconhecida, mas sabemos que funciona.
    # Para este exemplo, vamos supor que ele resolve HALT_TM.
    # Instância é um tuple (maquina_str, entrada_str)
    maquina, entrada = instancia_b
    if "loop" in maquina:
        print(" [Oráculo B] Concluiu: NÃO PARA.")
        return False
    else:
        print(" [Oráculo B] Concluiu: PARA.")
        return True

# Queremos resolver o problema A, que sabemos ser difícil
# A = Problema da Aceitação (A_TM)

def reduction_A_to_B(instancia_a):
    """Esta é a nossa função de redução 'f'.
    Transforma uma instância de A_TM em uma de HALT_TM."""
    print(f"[Redução] Recebeu instância de A: {instancia_a}")
    maquina_M, entrada_w = instancia_a

    # Construímos a descrição da nova máquina M'
```



```

[Redução] Recebeu instância de A: ("def M(w): return 'rejeita'", 'input_string')
[Redução] Produziu instância de B: ('\n      def M_prime(x):\n          # Simula M em w\n[Oráculo B] Recebeu a instância: ('\n      def M_prime(x):\n          # Simula M em w\n[Oráculo B] Concluiu: NÃO PARA.
Resultado final para A: NÃO ACEITA

False

```

```

resultado_
resultado_

```

Exercícios de Verificação

Atividade Prática

1. **Conceitual:** Se você prova que $A \leq_m B$ e você sabe que a linguagem B é **decidível**, o que você pode concluir sobre a linguagem A ? Por quê?
2. **Redução:** Considere a linguagem $ALL_{TM} = \{\langle M \rangle \mid L(M) = \Sigma^*\}$, ou seja, o conjunto de TMs que aceitam *todas* as strings. Prove que ALL_{TM} é indecidível por meio de uma redução a partir de A_{TM} .
3. **Completeness:** O problema $HALT_{TM}$ também é RE-completo. Para provar isso, quais duas condições você precisaria verificar?

Referências Bibliográficas

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