# Linguagens Recursivas e Recursivamente Enumeráveis

Computabilidade e Complexidade

Márcio Nicolau

2025-09-08

## Table of contents



## List of Figures



## Introdução: Certeza vs. Esperança

Nas aulas anteriores, estabelecemos que uma Máquina de Turing (TM) pode ter três desfechos ao processar uma entrada: aceitar, rejeitar ou entrar em loop. Essa distinção não é apenas um detalhe técnico; ela está no cerne da diferença entre problemas que podemos resolver completamente e problemas para os quais só podemos ter uma "esperança" de solução.

Imagine que você tem um programa para verificar uma conjectura matemática.

- **Cenário 1 (Certeza):** O programa sempre para, respondendo "Verdadeira" ou "Falsa".

- **Cenário 2 (Esperança):** O programa busca por uma prova. Se encontrar, ele para e diz "Verdadeira". Se não encontrar, ele pode continuar buscando para sempre, sem nunca nos dar uma resposta definitiva de "Falsa".

Esses dois cenários correspondem a duas classes de linguagens fundamentais na teoria da computação, que usam os termos históricos **recursiva** e **recursivamente enumerável**.

O objetivo da aula de hoje é:

> **Distinguir linguagens computáveis (recursivas/decidíveis) e parcialmente computáveis (recursivamente enumeráveis/reconhecíveis) em Máquinas de Turing.**

## Linguagens Decidíveis (Recursivas)

Esta é a classe de linguagens que corresponde ao nosso ideal de "problema resolvido". Para esses problemas, temos um algoritmo que sempre nos dá uma resposta clara, "sim" ou "não", em um tempo finito (Sipser, 2012).

> ❗ Definição: Linguagem Decidível (ou Recursiva)
>
> Uma linguagem $L$ é **decidível** (ou **recursiva**) se existe uma Máquina de Turing $M$ que é um **decisor**. Um decisor é uma TM que **sempre para** em qualquer entrada.
> - Se $w \in L$, a TM $M$ aceita $w$.
> - Se $w \notin L$, a TM $M$ rejeita $w$.
>
> A TM $M$ nunca entra em loop. A classe de todas as linguagens decidíveis é denotada por **R**.

Em termos de programação, uma linguagem decidível é aquela para a qual podemos escrever uma função que sempre termina e retorna um booleano `True` ou `False`.

```python
def eh_palindromo(w: str) -> bool:
    """
    Esta função SEMPRE termina e retorna True ou False.
    Ela decide a linguagem dos palíndromos.
    """
    return w == w[::-1]

# Esta função é um algoritmo que decide um problema.
print(f"'arara' é palíndromo? {eh_palindromo('arara')}")
print(f"'python' é palíndromo? {eh_palindromo('python')}")
```

## Linguagens Reconhecíveis (Recursivamente Enumeráveis)

Esta é uma classe de linguagens mais ampla e um pouco mais sutil. Para esses problemas, temos um algoritmo que pode confirmar a resposta "sim", mas pode não ser capaz de confirmar a resposta "não".

**!** Definição: Linguagem Reconhecível (ou Recursivamente Enumerável)

Uma linguagem $L$ é **reconhecível** (ou **recursivamente enumerável**) se existe uma Máquina de Turing $M$ que é um **reconhecedor**.

- Se $w \in L$, a TM $M$ aceita $w$.
- Se $w \notin L$, a TM $M$ rejeita $w$ **OU entra em loop**.

A classe de todas as linguagens reconhecíveis é denotada por **RE**.

O termo "recursivamente enumerável" vem de uma definição alternativa equivalente: uma linguagem é RE se existe uma TM, chamada de **enumerador**, que imprime (enumera) todas as strings da linguagem, uma por uma (a ordem não importa e repetições são permitidas).

## A Relação Entre as Classes

Claramente, se uma linguagem é decidível, ela também é reconhecível. Um decisor é apenas um reconhecedor mais "bem-comportado" que nunca usa a opção de entrar em loop. Portanto, **R** é um subconjunto de **RE**.

Mas será que **R = RE**? Ou seja, será que todo problema para o qual podemos verificar uma resposta "sim" também é um problema para o qual podemos sempre obter uma resposta definitiva? A resposta é **não**.

O que separa essas duas classes é o conceito de **complemento**. O complemento de uma linguagem $L$, denotado `co-L`, é o conjunto de todas as strings que *não* estão em $L$.

**i** Teorema Fundamental

Uma linguagem $L$ é **decidível (R)** se, e somente se, tanto $L$ quanto seu complemento, `co-L`, são **reconhecíveis (RE)**. (Sipser, 2012, p. 222).

**Prova (intuição):**

- Se $L$ é decidível, seu complemento também é (basta trocar os estados de aceitação e rejeição do decisor). Como toda linguagem decidível é reconhecível, tanto $L$ quanto `co-L` são reconhecíveis.
- Agora, suponha que temos um reconhecedor $M_L$ para $L$ e um reconhecedor $M_{coL}$ para `co-L`. Para decidir se uma string $w$ está em $L$, podemos construir uma nova TM que simula $M_L$ e $M_{coL}$ em $w$ "em paralelo" (por exemplo, alternando um passo de cada em uma TM de 2 fitas). Para qualquer $w$, ela pertence a $L$ ou a `co-L`. Portanto, um dos dois reconhecedores **tem que parar e aceitar**. Se $M_L$ aceita, aceitamos $w$. Se $M_{coL}$ aceita, rejeitamos $w$. Como um deles garantidamente para, nossa nova TM é um decisor.

Isso nos leva a uma visão clara do universo das linguagens.

A consequência mais importante disso é: se encontrarmos uma linguagem $L$ que é reconhecível, mas cujo complemento `co-L` **não** é reconhecível, então podemos concluir que $L$ **não é decidível**.

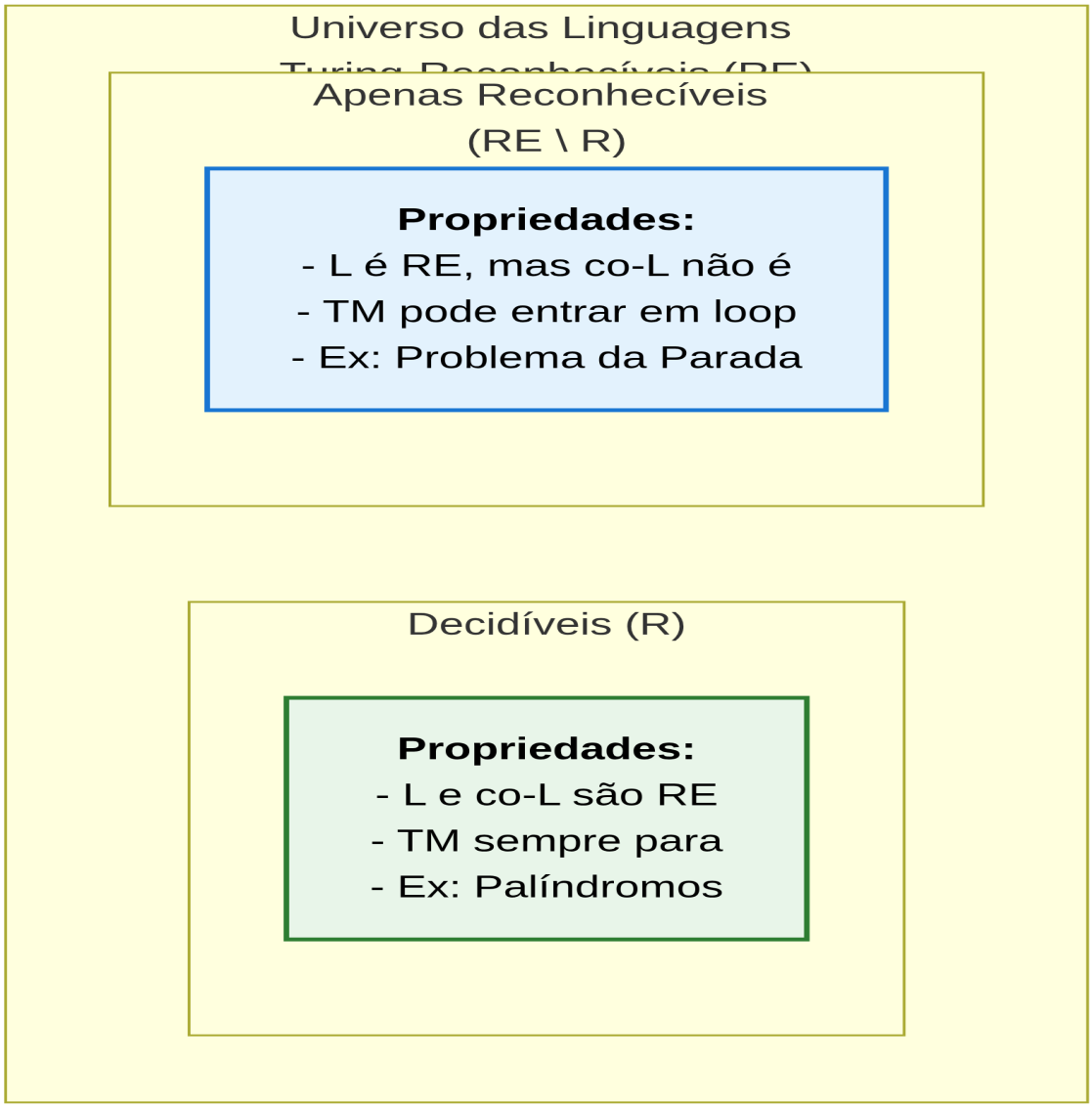


Figure 1: O universo das linguagens reconhecíveis, particionado por decidibilidade.

# O Problema da Parada: O Grande Exemplo

O exemplo canônico que vive na região "Apenas Reconhecível" é o famoso **Problema da Parada**.

Definimos a linguagem $A_{TM}$ como:

$A_{TM} = \{\langle M, w\rangle \mid M \text{ é uma TM e } M \text{ aceita a entrada } w\}$

- $A_{TM}$ **é Turing-Reconhecível (RE):** Podemos construir uma TM (chamada de Máquina de Turing Universal) que recebe $\langle M, w\rangle$ como entrada e simula a execução de $M$ em $w$. Se a simulação de $M$ para e aceita, nossa TM universal também para e aceita. Se $M$ rejeita ou entra em loop, nossa simulação também rejeita ou entra em loop. Isso se encaixa perfeitamente na definição de um reconhecedor.
- $A_{TM}$ **não é Decidível (R):** Provaremos isso formalmente na próxima aula usando diagonalização, mas o resultado é que não existe um algoritmo que possa, para *qualquer* programa e *qualquer* entrada, determinar se o programa irá parar e aceitar.

Como $A_{TM}$ é RE mas não é R, o teorema anterior nos diz que seu complemento, `co-`$A_{TM}$, **não pode ser Turing-Reconhecível**. Este é o nosso primeiro exemplo de um problema que não é nem mesmo parcialmente solucionável.

# Verificação de Aprendizagem

### 1: Conceitual

Se você entrega um programa a uma Máquina de Turing que reconhece a linguagem $A_{TM}$ e a máquina roda por um milhão de anos sem parar, o que você pode concluir sobre o seu programa? Por quê?

### 2: Relação entre Classes

Suponha que um colega afirme ter encontrado uma linguagem $L$ tal que nem $L$ nem seu complemento `co-L` são Turing-reconhecíveis. Isso é possível? Explique sua resposta com base no que vimos sobre as classes R e RE.

### 3: Conexão com Python

Considere a seguinte função Python, que tenta resolver o Problema de Collatz para um número `n`. A conjectura diz que este processo sempre chegará a 1.

```python
def collatz_chega_a_um(n: int) -> bool:
    """
    Verifica se a sequência de Collatz para n chega a 1.
    A conjectura é que isso é verdade para todo n > 0.
    """
    if n <= 0:
        return False

    seq = {n}
    while n != 1:
```

```
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1

        # Detecção de ciclo (caso a conjectura seja falsa)
        if n in seq:
            return False # Entrou em um loop que não inclui 1
        seq.add(n)

    return True # Chegou a 1
```

Até hoje, ninguém sabe se esta função sempre termina para todos os `n` (embora se acredite que sim). Se houvesse um `n` para o qual a função entrasse em loop infinito, como o comportamento dessa função se assemelharia ao de um **reconhecedor** de uma linguagem? Qual seria essa linguagem?

## Referências Bibliográficas

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