Teoria da Computação

Decidibilidade e Redutibilidade

Aula 11

Prof. Felipe A. Louza

UFU

Roteiro

1. Decidibilidade
   - O método da diagonalização de Cantor
   - Linguagens e Máquinas de Turing
   - O problema da aceitação para MTs
2. Redutibilidade
   - O problema da parada
   - Redutibilidade por mapeamento
   - O problema da vacuidade para MTs
3. Referências

Introdução

Na aula anterior, vimos a máquina de Turing (MT) como um modelo de computador de propósito geral e apresentamos a **tese de Church-Turing**:

> *"Um problema tem solução algorítmica se, e somente se, ele tem solução em uma máquina de Turing"*

Agora, vamos investigar os limites da computação:

- Vamos ver problemas que não têm solução computacional.
- Esses problemas são chamados de **indecidíveis**.

Introdução

Em boa parte do curso (graduação ou pós), estudamos problemas que podem ser resolvidos por um computador.

- Pode ser uma surpresa saber que existem problemas sem solução computacional.

Por que estudar problemas sem solução algorítmica?

1. Saber que um problema não tem solução computacional é útil para que saibamos que é preciso modifica-lo/simplificá-lo.
2. Além disso, entender os **limites da computação** pode nos dar uma nova perspectiva[1].

[1]Como qualquer ferramenta, os computadores possuem capacidades e limitações que devem ser conhecidas.

Introdução

Em boa parte do curso (graduação ou pós), estudamos problemas que podem ser resolvidos por um computador.

- Pode ser uma surpresa saber que existem problemas sem solução computacional.

Por que estudar problemas sem solução algorítmica?

1. Saber que um problema não tem solução computacional é útil para que saibamos que é preciso modifica-lo/simplificá-lo.
2. Além disso, entender os **limites da computação** pode nos dar uma nova perspectiva[1].

[1] Como qualquer ferramenta, os computadores possuem capacidades e limitações que devem ser conhecidas.

Introdução

Em boa parte do curso (graduação ou pós), estudamos problemas que podem ser resolvidos por um computador.

- Pode ser uma surpresa saber que existem problemas sem solução computacional.

Por que estudar problemas sem solução algorítmica?

1. Saber que um problema não tem solução computacional é útil para que saibamos que é preciso modifica-lo/simplificá-lo.
2. Além disso, entender os **limites da computação** pode nos dar uma nova perspectiva[1].

[1] Como qualquer ferramenta, os computadores possuem capacidades e limitações que devem ser conhecidas.

Introdução

Vamos ver problemas que não são **Turing-decidíveis**[2].

Antes, vamos provar que *existem* tais problemas insolúveis.

Vamos ver que existem mais problemas (linguagens) do que algoritmos (MTs).

"O **conjunto de todas as linguagens** *é maior do que o* **conjunto de todas as MTs**.*"*

[2]Equivalente a dizer que o problema tem solução computacional.

Introdução

Vamos ver problemas que não são **Turing-decidíveis**[2].

Antes, vamos provar que *existem* tais problemas insolúveis.

- Vamos ver que existem mais problemas (linguagens) do que algoritmos (MTs).

  "O **conjunto de todas as linguagens** *é maior do que o* **conjunto de todas as MTs***."*

[2]Equivalente a dizer que o problema tem solução computacional.

Roteiro

1. Decidibilidade
   - O método da diagonalização de Cantor
   - Linguagens e Máquinas de Turing
   - O problema da aceitação para MTs
2. Redutibilidade
   - O problema da parada
   - Redutibilidade por mapeamento
   - O problema da vacuidade para MTs
3. Referências

O método da diagonalização

Como podemos comparar dois conjuntos A e B?

- Para conjuntos finitos é fácil! Simplesmente contamos os seus elementos.
- Como comparar conjuntos infinitos?

Em 1873, George Cantor propôs uma solução elegante para esse problema.

O método da diagonalização

Como podemos comparar dois conjuntos A e B?

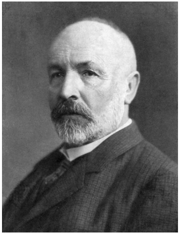
- Para conjuntos finitos é fácil! Simplesmente contamos os seus elementos.
- Como comparar conjuntos infinitos?

Em 1873, George Cantor propôs uma solução elegante para esse problema.

Como podemos comparar dois conjuntos A e B?

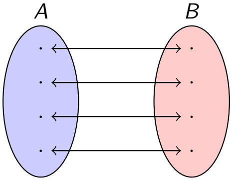
- Para conjuntos finitos é fácil! Simplesmente contamos os seus elementos.
- Como comparar conjuntos infinitos?

Em 1873, George Cantor propôs uma solução elegante para esse problema.



O método da diagonalização

Cantor observou que dois conjuntos finitos A e B possuem o mesmo tamanho, se existe um **emparelhamento**[3] de elementos de A para B.

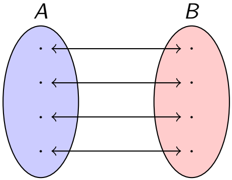


- Esse método compara os tamanhos de A e B sem recorrer a contagem.
- Essa idéia pode ser estendida para conjuntos infinitos.

[3] Correspondência um-para-um, ou função bijetora.

O método da diagonalização

Cantor observou que dois conjuntos finitos A e B possuem o mesmo tamanho, se existe um **emparelhamento**[3] de elementos de A para B.

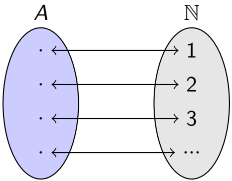


- Esse método compara os tamanhos de A e B sem recorrer a contagem.
- Essa idéia pode ser estendida para conjuntos infinitos.

[3] Correspondência um-para-um, ou função bijetora.

O método da diagonalização

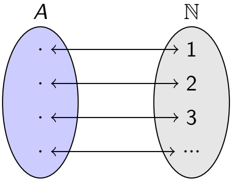
Um conjunto infinito A é **enumerável** (ou **contável**) se podemos construir uma **correspondência** de A com os números naturais $\mathbb{N}$.



- Caso contrário, A é **não-enumerável** (ou **incontável**).
- Dois conjuntos infinitos A e B têm o mesmo tamanho (ou cardinalidade) se ambos são **enumeráveis**.

O método da diagonalização

Um conjunto infinito A é **enumerável** (ou **contável**) se podemos construir uma **correspondência** de A com os números naturais $\mathbb{N}$.



- Caso contrário, A é **não-enumerável** (ou **incontável**).
- Dois conjuntos infinitos A e B têm o mesmo tamanho (ou cardinalidade) se ambos são **enumeráveis**.

Exemplo 1

Considere o conjunto dos números naturais pares

$$A = \{2, 4, 6, \dots\}$$

- Usando a definição de Cantor podemos ver que A e $\mathbb{N}$ possuem o mesmo tamanho[4].
- Para isso, mostramos a **correspondência** $f : \mathbb{N} \to A$, tal que $f(n) = 2n$:

n	$f(n)$
1	2
2	4
3	6
$\vdots$	$\vdots$

- A é **enumerável**, logo tem a mesma cardinalidade de $\mathbb{N}$.

[4] Intuitivamente parece que A tem a metade de elementos de $\mathbb{N}$.

Exemplo 1

Considere o conjunto dos números naturais pares

$$A = \{2, 4, 6, \dots\}$$

- Usando a definição de Cantor podemos ver que A e $\mathbb{N}$ possuem o mesmo tamanho[4].
- Para isso, mostramos a **correspondência** $f : \mathbb{N} \to A$, tal que $f(n) = 2n$:

n	$f(n)$
1	2
2	4
3	6
$\vdots$	$\vdots$

- A é **enumerável**, logo tem a mesma cardinalidade de $\mathbb{N}$.

[4] Intuitivamente parece que A tem a metade de elementos de $\mathbb{N}$.

Exemplo 2

Considere o conjunto dos números racionais positivos

$$Q = \{\frac{m}{n} \mid m, n \in \mathbb{N}\}$$

- Como mostrar que Q é **enumerável**?
- Podemos criar uma matriz infinita com $M_{ij} = i/j$ e "*enumerar*[5]" Q:

[5] O 1º elemento emparelhamos com $1 \in \mathbb{N}$, o 2º com o $2 \in \mathbb{N}$, e assim por diante.

Exemplo 2

Considere o conjunto dos números racionais positivos

$$Q = \{\frac{m}{n} \mid m, n \in \mathbb{N}\}$$

- Como mostrar que Q é **enumerável**?
- Podemos criar uma matriz infinita com $M_{ij} = i/j$ e "*enumerar*[5]" Q:

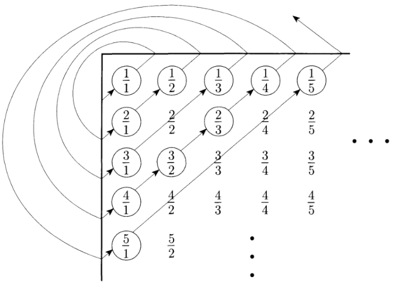
[5] O 1º elemento emparelhamos com $1 \in \mathbb{N}$, o 2º com o $2 \in \mathbb{N}$, e assim por diante.

Exemplo 2

Considere o conjunto dos números racionais positivos

$$Q = \{\frac{m}{n} \mid m, n \in \mathbb{N}\}$$

- Como mostrar que Q é **enumerável**?
- Podemos criar uma matriz infinita com $M_{ij} = i/j$ e "*enumerar*[5]" Q:



[5] O 1º elemento emparelhamos com $1 \in \mathbb{N}$, o 2º com o $2 \in \mathbb{N}$, e assim por diante.

O método da diagonalização

Então, para mostrar que um conjunto infinito é **enumerável** basta exibir uma uma **correspondência** com $\mathbb{N}$.

- Essa correspondência pode ser surpreendente como com os racionais positivos.
- Entretanto, para alguns conjuntos infinitos, **nenhuma** correspondência com $\mathbb{N}$ é possível.
  - Esses são os conjuntos **não-enumeráveis** (ou **incontáveis**).

Exemplo 3

Considere o conjunto dos números reais $\mathbb{R}$.

- Um número real é aquele que tem uma representação decimal.
- Os números $\pi = 3,1415926\ldots$ e $\sqrt{2}$ são exemplos de números reais.
- Cantor provou que $\mathbb{R}$ é **incontável**.
  - Ou seja, é impossível enumerar (contar) os números reais.

Exemplo 3

Considere o conjunto dos números reais $\mathbb{R}$.

- Um número real é aquele que tem uma representação decimal.
- Os números $\pi = 3,1415926\ldots$ e $\sqrt{2}$ são exemplos de números reais.
- Cantor provou que $\mathbb{R}$ é **incontável**.
  - Ou seja, é impossível enumerar (contar) os números reais.

Exemplo 3

A prova é por contradição:

- Suponha que existe uma **correspondência** $f : \mathbb{N} \to \mathbb{R}$.
- f pode ser qualquer uma, por exemplo a seguinte enumeração:

n	$f(n)$
1	3.14159...
2	5.555...
3	0.12345...
4	0.50000...
$\vdots$	$\vdots$

- Vamos *"construir"* um número $x \in \mathbb{R}$ em $[0, 1)$ e mostrar que ele não pode estar na enumeração.
  - Vamos tomar o i-ésimo dígito de $x \neq$ i-ésimo dígito em $f(i)$.
  - Observe que $x \neq f(1), f(2), f(3), \ldots$ $\leftarrow$ difere de $f(n)$ no n-ésimo dígito.

Exemplo 3

A prova é por contradição:

- Suponha que existe uma **correspondência** $f : \mathbb{N} \to \mathbb{R}$.
- f pode ser qualquer uma, por exemplo a seguinte enumeração:

n	$f(n)$
1	3.14159...
2	5.555...
3	0.12345...
4	0.50000...
$\vdots$	$\vdots$

- Vamos *"construir"* um número $x \in \mathbb{R}$ em $[0, 1)$ e mostrar que ele não pode estar na enumeração.
  - Vamos tomar o i-ésimo dígito de $x \neq$ i-ésimo dígito em $f(i)$.
  - Observe que $x \neq f(1), f(2), f(3), \ldots$ $\leftarrow$ difere de $f(n)$ no n-ésimo dígito.

Exemplo 3

A prova é por contradição:

- Suponha que existe uma **correspondência** $f : \mathbb{N} \to \mathbb{R}$.
- f pode ser qualquer uma, por exemplo a seguinte enumeração:

n	$f(n)$
1	$3.\underline{\mathbf{1}}4159\ldots$
2	$5.5\underline{\mathbf{5}}5\ldots$
3	$0.12\underline{\mathbf{3}}45\ldots$
4	$0.500\underline{\mathbf{0}}0\ldots$
$\vdots$	$\vdots$
x	$\mathbf{0.4641}\ldots$

- Vamos *"construir"* um número $x \in \mathbb{R}$ em $[0, 1)$ e mostrar que ele não pode estar na enumeração.
  - Vamos tomar o i-ésimo dígito de $x \neq$ i-ésimo dígito em $f(i)$.
  - Observe que $x \neq f(1), f(2), f(3), \ldots$ $\leftarrow$ difere de $f(n)$ no n-ésimo dígito.

Exemplo 3

A prova é por contradição:

- Suponha que existe uma **correspondência** $f : \mathbb{N} \to \mathbb{R}$.
- f pode ser qualquer uma, por exemplo a seguinte enumeração:

n	$f(n)$
1	$3.\underline{\mathbf{1}}4159\ldots$
2	$5.5\underline{\mathbf{5}}5\ldots$
3	$0.12\underline{\mathbf{3}}45\ldots$
4	$0.500\underline{\mathbf{0}}0\ldots$
$\vdots$	$\vdots$
x	$\mathbf{0.4641}\ldots$

- Vamos *"construir"* um número $x \in \mathbb{R}$ em $[0, 1)$ e mostrar que ele não pode estar na enumeração.
  - Vamos tomar o i-ésimo dígito de $x \neq$ i-ésimo dígito em $f(i)$.
  - Observe que $x \neq f(1), f(2), f(3), \ldots$ $\leftarrow$ difere de $f(n)$ no n-ésimo dígito.

Exemplo 3

Continuação:

- Então, existe um $x \in \mathbb{R}$ que **não vai** estar "emparelhado" com nenhum número natural.
- Portanto, a **correspondência** $f : \mathbb{N} \to \mathbb{R}$ não existe, e $\mathbb{R}$ é **incontável**

n	$f(n)$
1	3.**1**4159...
2	5.5**5**5...
3	0.12**3**45...
4	0.500**0**0...
$\vdots$	$\vdots$
x	**0.4641** ...

- Essa estratégia é conhecida como Método da Diagonalização.
- Vamos ver aplicações importantes na Teoria da Computação.

Exemplo 3

Continuação:

- Então, existe um $x \in \mathbb{R}$ que **não vai** estar "emparelhado" com nenhum número natural.
- Portanto, a **correspondência** $f : \mathbb{N} \to \mathbb{R}$ não existe, e $\mathbb{R}$ é **incontável**

n	$f(n)$
1	$3.\underline{\mathbf{1}}4159\ldots$
2	$5.5\underline{\mathbf{5}}5\ldots$
3	$0.12\underline{\mathbf{3}}45\ldots$
4	$0.500\underline{\mathbf{0}}0\ldots$
$\vdots$	$\vdots$
x	$\mathbf{0.4641}\ldots$

  - Essa estratégia é conhecida como Método da Diagonalização.
  - Vamos ver aplicações importantes na Teoria da Computação.

Exemplo 3

Continuação:

- Então, existe um $x \in \mathbb{R}$ que **não vai** estar "emparelhado" com nenhum número natural.
- Portanto, a **correspondência** $f : \mathbb{N} \to \mathbb{R}$ não existe, e $\mathbb{R}$ é **incontável**

n	$f(n)$
1	$3.\underline{\mathbf{1}}4159\ldots$
2	$5.5\underline{\mathbf{5}}5\ldots$
3	$0.12\underline{\mathbf{3}}45\ldots$
4	$0.500\underline{\mathbf{0}}0\ldots$
$\vdots$	$\vdots$
x	**0.4641** ...

  - Essa estratégia é conhecida como Método da Diagonalização.
  - Vamos ver aplicações importantes na Teoria da Computação.

Roteiro

1. Decidibilidade
   - O método da diagonalização de Cantor
   - Linguagens e Máquinas de Turing
   - O problema da aceitação para MTs
2. Redutibilidade
   - O problema da parada
   - Redutibilidade por mapeamento
   - O problema da vacuidade para MTs
3. Referências

Linguagens e Máquinas de Turing

Em particular, vamos mostrar que:

1. O conjunto de todas as **MTs** é contável.
2. O conjunto de todas as **linguagens** é incontável.

Concluiremos que há (muito) mais linguagens (problemas) do que **MTs** (algoritmos).

O conjunto de todas as MTs (algoritmos)

Para mostrar que o conjunto de todas as **MTs** é **contável**, primeiro observamos que Σ^* é contável, para qualquer Σ:

- Podemos **enumerar** as cadeias de Σ^* em ordem crescente de tamanho e ordem lexicográfica[6].
- Exemplo para $\Sigma = \{0, 1\}$:

n	0	1	2	3	4	5	6	7	8	9	...
$f(n)$	$\mathcal{E}$	0	1	00	01	10	11	001	010	011	...

Vimos que uma **MT** M pode ser codificada em uma cadeia $\langle M \rangle \in \Sigma^*$.

- Se considerarmos apenas codificações válidas de **MTs**, podemos **enumerar** todas as **MTs**.
- Logo o conjunto de todas as **MTs** é **contável**.

[6] Temos uma quantidade finita de cadeias de tamanho 1, 2, ...

O conjunto de todas as MTs (algoritmos)

Para mostrar que o conjunto de todas as **MTs** é **contável**, primeiro observamos que Σ^* é contável, para qualquer Σ:

- Podemos **enumerar** as cadeias de Σ^* em ordem crescente de tamanho e ordem lexicográfica[6].
- Exemplo para $\Sigma = \{0, 1\}$:

n	0	1	2	3	4	5	6	7	8	9	...
$f(n)$	$\mathcal{E}$	0	1	00	01	10	11	001	010	011	...

Vimos que uma **MT** M pode ser codificada em uma cadeia $\langle M \rangle \in \Sigma^*$.

- Se considerarmos apenas codificações válidas de **MTs**, podemos **enumerar** todas as **MTs**.
- Logo o conjunto de todas as **MTs** é **contável**.

[6]Temos uma quantidade finita de cadeias de tamanho $1, 2, \ldots$

O conjunto de todas as MTs (algoritmos)

Para mostrar que o conjunto de todas as **MTs** é **contável**, primeiro observamos que Σ^* é contável, para qualquer Σ:

- Podemos **enumerar** as cadeias de Σ^* em ordem crescente de tamanho e ordem lexicográfica[6].
- Exemplo para $\Sigma = \{0, 1\}$:

n	0	1	2	3	4	5	6	7	8	9	...
$f(n)$	$\mathcal{E}$	0	1	00	01	10	11	001	010	011	...

Vimos que uma **MT** M pode ser codificada em uma cadeia $\langle M \rangle \in \Sigma^*$.

- Se considerarmos apenas codificações válidas de **MTs**, podemos **enumerar** todas as **MTs**.
- Logo o conjunto de todas as **MTs** é **contável**.

[6] Temos uma quantidade finita de cadeias de tamanho $1, 2, \ldots$

O conjunto de todas as linguagens (problemas)

Para mostrar que o conjunto de todas as **linguagens** $\mathbb{L}$ é **incontável**, vamos usar o método da Diagonalização de Cantor:

- Por contradição, assuma que $\mathbb{L}$ é contável, então existe uma enumeração $L_1, L_2, L_3, \ldots$.
- Vamos definir uma tabela:
  - Cada célula $(i, j) = 1$ indica que $w_j \in L_i$.
  - Ex: $L_1 = \{w_1, w_4, \ldots\}$

	w_1	w_2	w_3	w_4	$\ldots$
L_1	1	0	0	1	$\vdots$
L_2	0	0	1	0	$\vdots$
L_3	1	0	0	1	$\vdots$
L_4	1	1	0	1	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Existe alguma linguagem $\hat{L} \in \mathbb{L}$ que não está nessa tabela?

Por hipótese a lista estaria completa.

O conjunto de todas as linguagens (problemas)

Para mostrar que o conjunto de todas as **linguagens** $\mathbb{L}$ é **incontável**, vamos usar o método da Diagonalização de Cantor:

- Por contradição, assuma que $\mathbb{L}$ é contável, então existe uma enumeração $L_1, L_2, L_3, \ldots$.
- Vamos definir uma tabela:
  - Cada célula $(i, j) = 1$ indica que $w_j \in L_i$.
  - Ex: $L_1 = \{w_1, w_4, \ldots\}$
- Existe alguma linguagem $\hat{L} \in \mathbb{L}$ que não esta nessa tabela?

	w_1	w_2	w_3	w_4	$\ldots$
L_1	1	0	0	1	$\vdots$
L_2	0	0	1	0	$\vdots$
L_3	1	0	0	1	$\vdots$
L_4	1	1	0	1	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Por hipótese a lista estaria completa.

O conjunto de todas as linguagens (problemas)

Para mostrar que o conjunto de todas as **linguagens** $\mathbb{L}$ é **incontável**, vamos usar o método da Diagonalização de Cantor:

- Por contradição, assuma que $\mathbb{L}$ é contável, então existe uma enumeração $L_1, L_2, L_3, \ldots$.

- Vamos definir uma tabela:
  - Cada célula $(i, j) = 1$ indica que $w_j \in L_i$.
  - Ex: $L_1 = \{w_1, w_4, \ldots\}$

	w_1	w_2	w_3	w_4	$\ldots$
L_1	1	0	0	1	$\vdots$
L_2	0	0	1	0	$\vdots$
L_3	1	0	0	1	$\vdots$
L_4	1	1	0	1	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

- Existe alguma linguagem $\hat{L} \in \mathbb{L}$ que não esta nessa tabela?

Por hipótese a lista estaria completa.

O conjunto de todas as linguagens (problemas)

Continuação:

- Vamos *"construir"* uma linguagem $L_d \in \mathbb{L}$ com as cadeias correspondentes ao **complemento** da diagonal:

	w_1	w_2	w_3	w_4	$\dots$
L_1	**1**	0	0	1	$\vdots$
L_2	0	**0**	1	0	$\vdots$
L_3	1	0	**0**	1	$\vdots$
L_4	1	1	0	**1**	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
L_d	**0**	**1**	**1**	**0**	$\vdots$

Observe que L_d não pode estar na lista, pois ela sempre difere de outra linguagem na tabela (em pelo menos 1 coluna).

Logo, a lista de linguagens não está completa $\implies \mathbb{L}$ não é contável.

O conjunto de todas as linguagens (problemas)

Continuação:

- Vamos *"construir"* uma linguagem $L_d \in \mathbb{L}$ com as cadeias correspondentes ao **complemento** da diagonal:

	w_1	w_2	w_3	w_4	$\dots$
L_1	**1**	0	0	1	$\vdots$
L_2	0	**0**	1	0	$\vdots$
L_3	1	0	**0**	1	$\vdots$
L_4	1	1	0	**1**	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
L_d	**0**	**1**	**1**	**0**	$\vdots$

  - Observe que L_d não pode estar na lista, pois ela sempre difere de outra linguagem na tabela (em pelo menos 1 coluna).
  - Logo, a lista de linguagens não está completa $\leftarrow \mathbb{L}$ não é **contável**.

O conjunto de todas as linguagens (problemas)

Continuação:

- Vamos *"construir"* uma linguagem $L_d \in \mathbb{L}$ com as cadeias correspondentes ao **complemento** da diagonal:

	w_1	w_2	w_3	w_4	$\ldots$
L_1	**1**	0	0	1	$\vdots$
L_2	0	**0**	1	0	$\vdots$
L_3	1	0	**0**	1	$\vdots$
L_4	1	1	0	**1**	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
L_d	**0**	**1**	**1**	**0**	$\vdots$

  - Observe que L_d não pode estar na lista, pois ela sempre difere de outra linguagem na tabela (em pelo menos 1 coluna).
  - Logo, a lista de linguagens não está completa $\leftarrow \mathbb{L}$ não é **contável**.

Portanto:

1. O conjunto de todas as **MTs** é contável.
2. O conjunto de todas as **linguagens** é incontável.

Como cada **MT** reconhece uma única linguagem:

- Logo, existem linguagens (problemas) que não podem ser decididas por nenhuma **MT** (algoritmo)!

Finalmente, vamos ver um problema que não tem solução computacional[7].

[7] Não existe **MT** que decide a linguagem.

Portanto:

1. O conjunto de todas as **MTs** é contável.
2. O conjunto de todas as **linguagens** é incontável.

Como cada **MT** reconhece uma única linguagem:

- Logo, existem linguagens (problemas) que não podem ser decididas por nenhuma **MT** (algoritmo)!

Finalmente, vamos ver um problema que não tem solução computacional[7].

[7] Não existe **MT** que decide a linguagem.

Roteiro

1. Decidibilidade
   - O método da diagonalização de Cantor
   - Linguagens e Máquinas de Turing
   - O problema da aceitação para MTs
2. Redutibilidade
   - O problema da parada
   - Redutibilidade por mapeamento
   - O problema da vacuidade para MTs
3. Referências

O problema da aceitação para MTs

O problema da aceitação para **MTs** consiste em decidir se uma **MT** M aceita uma da cadeia w.

- Vamos reescrever esse problema em termos de linguagens:

$$A_{MT} = \{\langle M, w \rangle \mid M \text{ é uma MT que aceita a cadeia } w\}$$

Algumas considerações:

1. Testar se M aceita w é o mesmo que verificar se $\langle M, w \rangle \in A_{MT}$.
2. Mostra que A_{MT} **não é decidível** é o mesmo que mostrar que **não existe** algoritmo para resolver o problema.

O problema da aceitação para MTs

O problema da aceitação para **MTs** consiste em decidir se uma **MT** M aceita uma da cadeia w.

- Vamos reescrever esse problema em termos de linguagens:

$$A_{MT} = \{\langle M, w\rangle \mid M \text{ é uma MT que aceita a cadeia } w\}$$

Algumas considerações:

1. Testar se M aceita w é o mesmo que verificar se $\langle M, w\rangle \in A_{MT}$.
2. Mostra que A_{MT} **não é** decidível é o mesmo que mostrar que **não existe** algoritmo para resolver o problema.

O problema da aceitação para MTs

O problema da aceitação para **MTs** consiste em decidir se uma **MT** M aceita uma da cadeia w.

- Vamos reescrever esse problema em termos de linguagens:

$$A_{MT} = \{\langle M, w\rangle \mid M \text{ é uma MT que aceita a cadeia } w\}$$

Algumas considerações:

1. Testar se M aceita w é o mesmo que verificar se $\langle M, w\rangle \in A_{MT}$.
2. Mostra que A_{MT} **não é** decidível é o mesmo que mostrar que **não existe** algoritmo para resolver o problema.

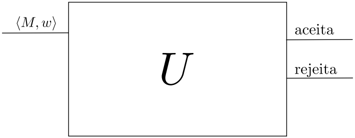
O problema da aceitação para MTs

Primeiro, vamos mostrar que A_{MT} é **Turing-reconhecível**.

- Para isso, apresentamos uma **MT** U que reconhece A_{MT}:

  $U =$ "Sobre a cadeia de entrada $\langle M, w\rangle$:

  1. Simule M sobre a cadeia w.
  2. Se M em algum momento M entra em q_{aceita}, aceite $\langle M, w\rangle$ ✓; se em algum momento M entra em $q_{rejeita}$, rejeite $\langle M, w\rangle$ ✗"



- Quando M aceita w, U aceita $\langle M, w\rangle$: U reconhece A_{MT}.
- Quando M entra em loop, U também entra: U não decide A_{MT}.

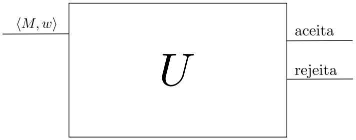
O problema da aceitação para MTs

Primeiro, vamos mostrar que A_{MT} é **Turing-reconhecível**.

- Para isso, apresentamos uma **MT** U que reconhece A_{MT}:

  $U =$ "Sobre a cadeia de entrada $\langle M, w \rangle$:

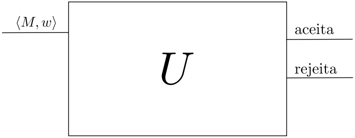
  1. Simule M sobre a cadeia w.
  2. Se M em algum momento M entra em q_{aceita}, aceite $\langle M, w \rangle$ ✓; se em algum momento M entra em $q_{rejeita}$, rejeite $\langle M, w \rangle$ ✗"



- Quando M aceita w, U aceita $\langle M, w \rangle$: U reconhece A_{MT}.
- Quando M entra em loop, U também entra: U não decide A_{MT}.

A Máquina de Turing Universal

A máquina U é interessante por si própria, ela também é conhecida como *máquina de Turing universal* (MTU):

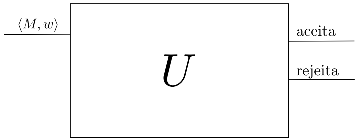


- A MTU pode *simular* o comportamento de qualquer outra **MT**.
- A descrição da **MT** M dada **como entrada** para U funciona como um programa (ou *software*), o que permite que U tenha comportamentos diferentes.
- A MTU é a base de vários resultados da Teoria da Computação.

Apresentada por A. Turing, a MTU teve um papel importante no estímulo ao desenvolvimento de computadores com o conceito de **programa armazenado**.

A Máquina de Turing Universal

A máquina U é interessante por si própria, ela também é conhecida como *máquina de Turing universal* (MTU):



- A MTU pode *simular* o comportamento de qualquer outra **MT**.
- A descrição da **MT** M dada **como entrada** para U funciona como um programa (ou *software*), o que permite que U tenha comportamentos diferentes.
- A MTU é a base de vários resultados da Teoria da Computação.

Apresentada por A. Turing, a MTU teve um papel importante no estímulo ao desenvolvimento de computadores com o conceito de **programa armazenado**.

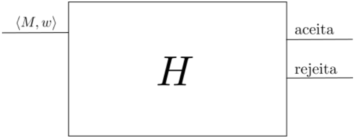
O problema da aceitação para MTs

Voltando para o problema A_{MT} (provar que não é **Turing-decidível**).

- Prova por contradição
- Vamos assumir que existe uma máquina H que **decide** A_{MT}

  $H =$ "Sobre a cadeia de entrada $\langle M, w \rangle$:

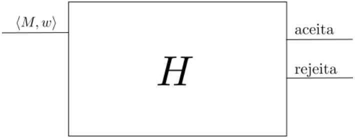
  1. Simule M sobre a cadeia w.
  2. Se M em algum momento M entra em q_{aceita}, aceite $\langle M, w \rangle$ ✓; e caso M falhe em aceitar w, rejeite $\langle M, w \rangle$ ✗"



O problema da aceitação para MTs

- Vamos reescrever o funcionamento dessa máquina hipotética H que **decide** A_{MT}:

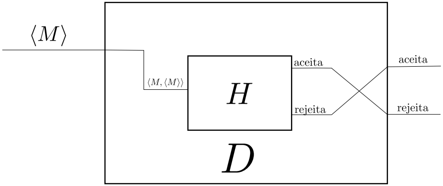
$$H(\langle M, w\rangle) = \begin{cases} \text{aceita}, & \text{se } M \text{ \underline{aceita} } w \\ \text{rejeita}, & \text{se } M \text{ \underline{não aceita} } w \end{cases}$$



Relembrando, $A_{MT} = \{\langle M, w\rangle \mid M \text{ é uma MT que aceita a cadeia } w\}$.

O problema da aceitação para MTs

- Se o **decisor** *H* existe, podemos constuir uma outra **MT** *D* que recebe $\langle M \rangle$ **como entrada** e utiliza *H* como subrotina.



1. A **MT** *D* chama *H* para determinar o que *M* faz quando recebe **como entrada** a sua própria descrição $\langle M \rangle$.
2. Em seguida, *D* **inverte** o resultado de *H*.

Não se atrapalhe com o fato da MT rodar sobre a sua própria descrição. Como um programa que roda com ele mesmo como entrada (ex. um compilador para a linguagem *C* pode ser escrito em *C*).

O problema da aceitação para MTs

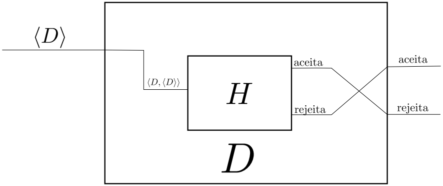
- Note que D terá o seguinte funcionamento ao receber **como entrada** uma **MT** M:

$$D(\langle M\rangle) = \begin{cases} \text{aceita}, & \text{se } M \ \underline{\text{não aceita}}\ \langle M\rangle \\ \text{rejeita}, & \text{se } M \ \underline{\text{aceita}}\ \langle M\rangle \end{cases}$$

O problema da aceitação para MTs

- Agora, veja o que acontece quando a D tem **como entrada** a sua própria descrição:

$$D(\langle D\rangle) = \begin{cases} \text{aceita}, & \text{se } D \text{ não aceita } \langle D\rangle \\ \text{rejeita}, & \text{se } D \text{ aceita } \langle D\rangle \end{cases}$$



- O que é um absurdo.

O problema da aceitação para MTs

Chegamos em um absurdo.

D *aceita a cadeia* $\langle D \rangle$, *se somente se,* D *rejeita a cadeia* $\langle D \rangle$

Logo, a **hipótese era falsa**, e não pode existir uma **MT** H que **decide** a linguagem A_{MT}.

- Portanto,

$$A_{MT} = \{\langle M, w \rangle \mid M \text{ é uma MT que aceita a cadeia } w\}$$

não é Turing-decidível.

O que equivale a dizer que esse problema **não tem solução** computacional (apesar de ser **Turing-reconhecível**).

O problema da aceitação para MTs

Chegamos em um absurdo.

$$\underline{D \text{ aceita a cadeia } \langle D \rangle}, \textit{ se somente se, } \underline{D \text{ rejeita a cadeia } \langle D \rangle}$$

Logo, a **hipótese era falsa**, e não pode existir uma **MT** H que **decide** a linguagem A_{MT}.

- Portanto,

$$A_{MT} = \{\langle M, w \rangle \mid M \text{ é uma MT que aceita a cadeia } w\}$$

  não é Turing-decidível.

O que equivale a dizer que esse problema **não tem solução** computacional (apesar de ser **Turing-reconhecível**).

Um problema Turing-irreconhecível

Para encerrar, vamos ver um problema que não é nem **Turing-reconhecível**.

- Considere o problema de decidir se uma **MT M** não aceita uma cadeia w.

$$\overline{A_{MT}} = \{\langle M, w\rangle \mid M \text{ é uma MT que } \textbf{não aceita} \text{ a cadeia } w\}$$

(complemento de A_{MT}).

- Em outras palavras, uma cadeia $\langle M, w\rangle \in \overline{A_{MT}}$ quando:
  1. A **MT** M pára e rejeita w; ou
  2. A **MT** M entra em loop.

Um problema Turing-irreconhecível

Para encerrar, vamos ver um problema que não é nem **Turing-reconhecível**.

- Considere o problema de decidir se uma **MT M** não aceita uma cadeia w.

$$\overline{A_{MT}} = \{\langle M, w\rangle \mid M \text{ é uma MT que } \textbf{não aceita} \text{ a cadeia } w\}$$

(complemento de A_{MT}).

- Em outras palavras, uma cadeia $\langle M, w\rangle \in \overline{A_{MT}}$ quando:
  1. A **MT** M pára e rejeita w; ou
  2. A **MT** M entra em loop.

Um problema Turing-irreconhecível

- Vamos provar que

$$\overline{A_{MT}} = \{\langle M, w \rangle \mid M \text{ é uma MT que } \textbf{não aceita} \text{ a cadeia } w\}$$

não é Turing-reconhecível.

- Prova (simples):
  - Sabemos que A_{MT} é mas não decidível
  - Se $\overline{A_{MT}}$ fosse , então verificar se

$$\langle M, w \rangle \in \overline{A_{MT}} \text{ seria o mesmo que } \langle M, w \rangle \notin A_{MT}$$

(o que é impossível)
  - Logo, $\overline{A_{MT}}$ não pode ser Turing-reconhecível.

Um problema Turing-irreconhecível

- Vamos provar que

$$\overline{A_{MT}} = \{\langle M, w\rangle \mid M \text{ é uma MT que } \textbf{não aceita} \text{ a cadeia } w\}$$

não é Turing-reconhecível.

- Prova (simples):

1. Sabemos que A_{MT} é **Turing-reconhecível** mas **não decidível**.
2. Se $\overline{A_{MT}}$ fosse **reconhecível**, então verificar se

$$\langle M, w\rangle \in \overline{A_{MT}} \text{ seria o mesmo que } \langle M, w\rangle \notin A_{MT}$$

(o que é impossível)

3. Logo, $\overline{A_{MT}}$ **não pode ser Turing-reconhecível**.

Um problema Turing-irreconhecível

- Vamos provar que

$$\overline{A_{MT}} = \{\langle M, w\rangle \mid M \text{ é uma MT que } \textbf{não aceita} \text{ a cadeia } w\}$$

não é Turing-reconhecível.

- Prova (simples):

  1. Sabemos que A_{MT} é **Turing-reconhecível** mas **não decidível**.
  2. Se $\overline{A_{MT}}$ fosse **reconhecível**, então verificar se

  $$\langle M, w\rangle \in \overline{A_{MT}} \text{ seria o mesmo que } \langle M, w\rangle \notin A_{MT}$$

  (o que é impossível)
  3. Logo, $\overline{A_{MT}}$ **não pode ser Turing-reconhecível**.

Um problema Turing-irreconhecível

- Vamos provar que

$$\overline{A_{MT}} = \{\langle M, w\rangle \mid M \text{ é uma MT que } \textbf{não aceita} \text{ a cadeia } w\}$$

não é Turing-reconhecível.

- Prova (simples):

1. Sabemos que A_{MT} é **Turing-reconhecível** mas **não decidível**.
2. Se $\overline{A_{MT}}$ fosse **reconhecível**, então verificar se

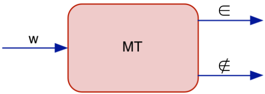
$$\langle M, w\rangle \in \overline{A_{MT}} \text{ seria o mesmo que } \langle M, w\rangle \notin A_{MT}$$

(o que é impossível)

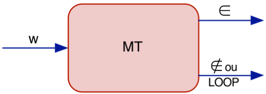
3. Logo, $\overline{A_{MT}}$ **não pode ser Turing-reconhecível**.

Problemas insolúveis

Turing-indecidíveis	**Turing-irreconhecíveis**
$A_{MT}, \overline{A_{MT}}$	$\overline{A_{MT}}$



Decide



Reconhece

Problemas insolúveis

Ao mostrar linguagens **Turing-indecidíveis/irreconhecíveis**, mostramos que existem problemas *sem solução computacional* (uma vez que aceitamos a **Tese de Church-Turing**).

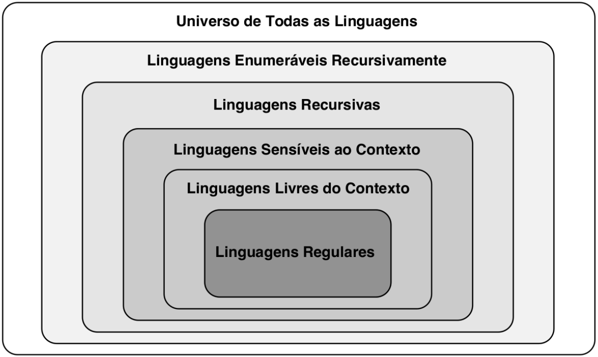


Figura: Hierarquia de Chomsky.

Roteiro

1. Decidibilidade
   - O método da diagonalização de Cantor
   - Linguagens e Máquinas de Turing
   - O problema da aceitação para MTs
2. Redutibilidade
   - O problema da parada
   - Redutibilidade por mapeamento
   - O problema da vacuidade para MTs
3. Referências

Redutibilidade

Vamos ver agora como provar que outros problemas são **indecidíveis** utilizando o conceito de *redutibilidade*:

- Uma **redução** é uma maneira de *converter* um problema A em um outro problema B.

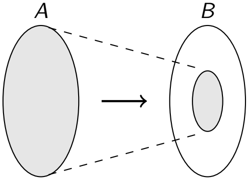
- Uma solução para B pode ser utilizada para resolver A.

Nem sempre é conveniente mostrar que um problema **é indecidível de forma direta**.

Redutibilidade

Vamos ver agora como provar que outros problemas são **indecidíveis** utilizando o conceito de *redutibilidade*:

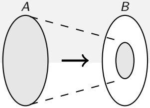
- Uma **redução** é uma maneira de *converter* um problema A em um outro problema B.



- Uma solução para B pode ser utilizada para resolver A.

Nem sempre é conveniente mostrar que um problema é **indecidível de forma direta**.

Redutibilidade



Se **um problema** *A* *é redutível* a **um problema** *B*, o que podemos dizer sobre a **dificuldade** de *A* em relação *B*?

- Resolver *A* não pode ser mais difícil do que resolver *B*.

Além disso:

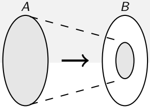
1. Se *B* é **decidível**, então *A* é **decidível**.
   Isso porque a solução de B pode ser usada para solucionar A.

2. Se *A* é **indecidível**, então *B* é **indecidível**.
   Isso porque se houvesse solução para B, haveria também solução para A, o que seria uma ***contradição****.*

Assim, para mostrar que **um problema** *P* é **indecidível**, basta reduzir um problema **indecidível** *Q* a ele.

Esse conceito é útil para classificar problemas em níveis de dificuldade.

Redutibilidade



Se **um problema** *A* *é redutível* a **um problema** *B*, o que podemos dizer sobre a **dificuldade** de *A* em relação *B*?

- Resolver *A* não pode ser mais difícil do que resolver *B*.

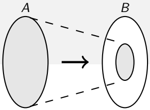
Além disso:

1. Se *B* é **decidível**, então *A* é **decidível**.
   Isso porque a solução de B pode ser usada para solucionar A.
2. Se *A* é **indecidível**, então *B* é **indecidível**.
   Isso porque se houvesse solução para B, haveria também solução para A, o que seria uma ***contradição****.*

Assim, para mostrar que **um problema** *P* é **indecidível**, basta reduzir um problema **indecidível** *Q* a ele.

Esse conceito é útil para classificar problemas em níveis de dificuldade.

Redutibilidade



Se **um problema** A *é redutível* a **um problema** B, o que podemos dizer sobre a **dificuldade** de A em relação B?

- Resolver A não pode ser mais difícil do que resolver B.

Além disso:

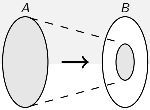
1. Se B é **decidível**, então A é **decidível**.
   Isso porque a solução de B pode ser usada para solucionar A.

2. Se A é **indecidível**, então B é **indecidível**.
   Isso porque se houvesse solução para B, haveria também solução para A, o que seria uma ***contradição****.*

Assim, para mostrar que **um problema** P é **indecidível**, basta reduzir um problema **indecidível** Q a ele.

Esse conceito é útil para classificar problemas em níveis de dificuldade.

Redutibilidade



Se **um problema** *A* *é redutível* a **um problema** *B*, o que podemos dizer sobre a **dificuldade** de *A* em relação *B*?

- Resolver *A* não pode ser mais difícil do que resolver *B*.

Além disso:

1. Se *B* é **decidível**, então *A* é **decidível**.

   Isso porque a solução de B pode ser usada para solucionar A.

2. Se *A* é **indecidível**, então *B* é **indecidível**.

   Isso porque se houvesse solução para B, haveria também solução para A, o que seria uma ***contradição****.*

Assim, para mostrar que **um problema** *P* é **indecidível**, basta reduzir um problema **indecidível** *Q* a ele.

Esse conceito é útil para classificar problemas em níveis de dificuldade.

Roteiro

O problema da parada

O problema da parada consiste em decidir se uma **MT** M pára (aceitando **ou** rejeitando) sobre uma da cadeia w.

$$PARA_{MT} = \{\langle M, w\rangle \mid M \text{ é uma MT e } M \text{ pára sobre a cadeia } w\}$$

- Para mostrar que $PARA_{MT}$ é **indecidível** faremos uma redução de $A_{MT} \to PARA_{MT}$.

Se houvesse um decisor para $PARA_{MT}$, esse poderia decidir A_{MT} o que seria um absurdo!

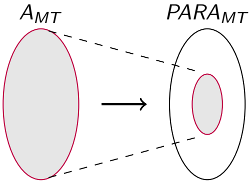
Sabemos que o problema da aceitação A_{MT} é **indecidível**.

O problema da parada

O problema da parada consiste em decidir se uma **MT** M pára (aceitando **ou** rejeitando) sobre uma da cadeia w.

$$PARA_{MT} = \{\langle M, w\rangle \mid M \text{ é uma MT e } M \text{ pára sobre a cadeia } w\}$$

- Para mostrar que $PARA_{MT}$ é **indecidível** faremos uma redução de $A_{MT} \to PARA_{MT}$.



Se houvesse um decisor para $PARA_{MT}$, esse poderia decidir A_{MT} o que seria um absurdo!

Sabemos que o problema da aceitação A_{MT} é **indecidível**.

O problema da parada

Prova por contradição:

- Vamos assumir que existe uma **MT** R que **decide** $PARA_{MT}$.
- Então, podemos construir a **MT** S que utiliza R:

$S =$ "Sobre a cadeia de entrada $\langle M, w \rangle$:

1. Rode **MT** R sobre a entrada $\langle M, w \rangle$ $\leftarrow$ decide se M pára.
2. Se R *rejeita*, rejeite $\langle M, w \rangle$ ✗ $\leftarrow$ M não aceita w e $\langle M, w \rangle \notin A_{MT}$
3. Se R *aceita*, simule M sobre w até que ela pare. $\leftarrow$ M pára.
4. Se M *aceita*, aceite $\langle M, w \rangle$ ✓, senão rejeite $\langle M, w \rangle$ ✗"

Note que S é capaz de decidir A_{MT} (não entra em loop):

- O que é um absurdo, pois sabemos que A_{MT} é **indecidível**.

O problema da parada

Chegamos em um absurdo.

Logo, a **hipótese era falsa**, e não pode existir uma **MT** R que **decide** a linguagem $PARA_{MT}$.

- Portanto,

  $$PARA_{MT} = \{\langle M, w\rangle \mid M \text{ é uma MT e } M \text{ pára sobre a cadeia } w\}$$

  não é Turing-decidível.

O que equivale a dizer que esse problema não tem solução computacional.

Problemas insolúveis

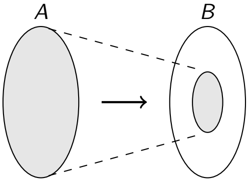
Turing-indecidíveis	Turing-irreconhecíveis
$A_{MT}, \overline{A_{MT}}$	$\overline{A_{MT}}$
$PARA_{MT}$	

Roteiro

Redutibilidade por mapeamento

Agora, vamos formalizar o conceito de *redutibilidade*:

- Estudaremos a **redutibilidade por mapeamento**.



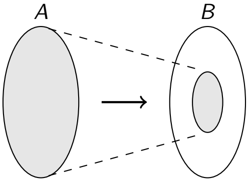
- Dizer que podemos **reduzir** um problema A a um problema B significa que existe uma *"função computável"* que converte instâncias de A em instâncias de B.

Existem outros modos de definir formalmente redutibilidade.

Redutibilidade por mapeamento

Agora, vamos formalizar o conceito de *redutibilidade*:

- Estudaremos a **redutibilidade por mapeamento**.



- Dizer que podemos **reduzir** um problema A a um problema B significa que existe uma *"função computável"* que converte instâncias de A em instâncias de B.

Existem outros modos de definir formalmente redutibilidade.

Funções computáveis

Definição

Uma função $f : \Sigma^* \to \Sigma^*$ é chamada de **função computável** se existe uma **MT** M que, para qualquer $w \in \Sigma^*$, M pára e deixa $f(w)$ na fita.

Exemplo:

- Todas as operações aritméticas sobre inteiros $(+, -, /, *)$ são funções computáveis.

  Podemos construir uma MT que recebe como entrada $\langle m, n \rangle$ e retorna $\langle m + n \rangle$.

Ou seja, f é **computável** se for possível escrever uma **MT** para ela.

Redutibilidade por mapeamento

Definição

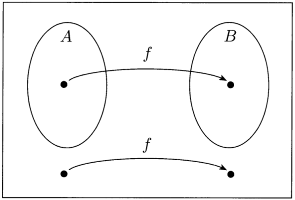
Uma linguagem A é **redutível por mapeamento** a uma linguagem B, denotado por $A \leq_m B$ se existe uma função computável $f : \Sigma^* \rightarrow \Sigma^*$, tal que, $\forall w \in \Sigma^*$

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

A função f é denominada de **redução** de A para B.

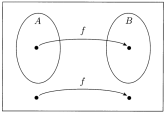
- Podemos *converter* questões do tipo

  "$w \in A$" em "$f(w) \in B$"



f não precisa ser bijetora.

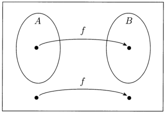
Redutibilidade por mapeamento



Se linguagem A é **redutível** a B, $A \leq_m B$, então:

1. Informalmente:
   - Existe um algoritmo para converter instâncias de um problema A em instâncias de um problema B.
2. Formalmente:
   - Existe uma **MT** que toma uma instância de A gravada em sua fita e pára com uma instância de B em sua fita.

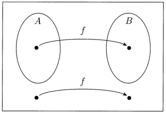
Redutibilidade por mapeamento



Se linguagem A é **redutível** a B, $A \leq_m B$, então:

1. Informalmente:
   – Existe um algoritmo para converter instâncias de um problema A em instâncias de um problema B.
2. Formalmente:
   – Existe uma **MT** que toma uma instância de A gravada em sua fita e pára com uma instância de B em sua fita.

Redutibilidade por mapeamento



Teorema

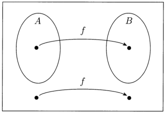
Se $A \leq_m B$, e B é **decidível**, então A é **decidível**.

Para usar esse teorema (mostrar que A é decidível), o nosso trabalho é:

1. Escolher um problema B **decidível**; e
2. Mostrar a **redução** f

Prova: exercício Lista 11.

Redutibilidade por mapeamento



Corolário

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

Essa é a nossa principal ferramenta para demonstrar **indecidibilidade** de problemas:

- Passo a passo:
  1. Assuma por contradição que B seja **decidível** pela **MT** R
  2. Escolha um problema A conhecido **indecidível**.
  3. Moste que $A \leq_m B$: construa a **MT** que computa a redução.
  4. Então $R(f(w))$ deve decidir $A \leftarrow$ Contradição!

Nosso trabalho: (2) e (3).

Roteiro

O problema da vacuidade para MTs

O problema da vacuidade para **MTs** consiste em decidir se a linguagem reconhecida por uma **MT** M é vazia.

$$V_{MT} = \{\langle M \rangle \mid M \text{ é uma MT e } L(M) = \varnothing\}$$

- Vamos mostrar uma **redução por mapeamento** $A_{MT} \leq_m V_{MT}$.

$$\text{“}\langle M, w \rangle \in A_{MT}\text{”} \Leftrightarrow \text{“}f(\langle M, w \rangle) \in V_{MT}\text{”}$$

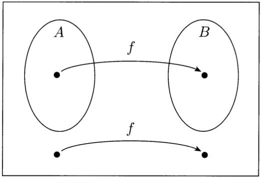
Sabemos que o problema da aceitação A_{MT} é indecidível, então V_{MT} é indecidível.

O problema da vacuidade para MTs

O problema da vacuidade para **MTs** consiste em decidir se a linguagem reconhecida por uma **MT** M é vazia.

$$V_{MT} = \{\langle M \rangle \mid M \text{ é uma MT e } L(M) = \varnothing\}$$

- Vamos mostrar uma **redução por mapeamento** $A_{MT} \leq_m V_{MT}$.



$$\text{“}\langle M, w \rangle \in A_{MT}\text{”} \Leftrightarrow \text{“}f(\langle M, w \rangle) \in V_{MT}\text{”}$$

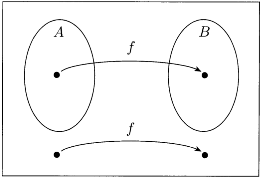
Sabemos que o problema da aceitação A_{MT} é indecidível, então V_{MT} é indecidível.

O problema da vacuidade para MTs

O problema da vacuidade para **MTs** consiste em decidir se a linguagem reconhecida por uma **MT** M é vazia.

$$V_{MT} = \{\langle M \rangle \mid M \text{ é uma MT e } L(M) = \varnothing\}$$

- Vamos mostrar uma **redução por mapeamento** $A_{MT} \leq_m V_{MT}$.



$$\text{“}\langle M, w \rangle \in A_{MT}\text{”} \Leftrightarrow \text{“}f(\langle M, w \rangle) \in V_{MT}\text{”}$$

Sabemos que o problema da aceitação A_{MT} é **indecidível**, então V_{MT} é **indecidível**.

O problema da vacuidade para MTs

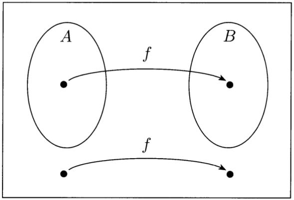
Ideia da prova:

- Precisamos projetar uma **MT** F que computa $f(\langle M, w \rangle) = \langle M' \rangle$

$$\langle M, w \rangle \in A_{MT} \quad \underline{\text{se e somente se}} \quad \langle M' \rangle \in V_{MT}$$

- A **MT** M' (resultante do mapeamento) pode ser calculada da seguinte forma:

$$L(M') = \begin{cases} \emptyset, & \text{se } M \text{ \underline{aceita} } w \\ \{w\}, & \text{se } M \text{ \underline{não aceita} } w \end{cases}$$



O problema da vacuidade para MTs

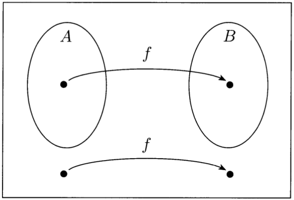
Ideia da prova:

- Precisamos projetar uma **MT** F que computa $f(\langle M, w\rangle) = \langle M'\rangle$

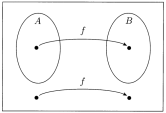
$$\langle M, w\rangle \in A_{MT} \quad \underline{\text{se e somente se}} \quad \langle M'\rangle \in V_{MT}$$

- A **MT** M' (resultante do mapeamento) pode ser calculada da seguinte forma:

$$L(M') = \begin{cases} \emptyset, & \text{se } M \text{ \underline{aceita} } w \\ \{w\}, & \text{se } M \text{ \underline{não aceita} } w \end{cases}$$



O problema da vacuidade para MTs



Prova:

- A seguinte **MT** F computa a redução f:

  $F =$ "Sobre a cadeia de entrada $\langle M, w \rangle$:

  1. Construa a seguinte **MT** M'

     $M' =$ "Sobre a cadeia de entrada x:

     1. Se $x \neq w$, **aceite** ✓
     2. Se $x = w$, rode M sobre w e **rejeite** ✗ se M aceitar w."

  2. Dê **como saída** $\langle M' \rangle$

Com isso, vamos supor que V_{MT} é decidível, então existe uma **MT** R para V_{MT}, nesse caso a **MT**:

$$S = R(F(\langle M, w \rangle)) \text{ é capaz de decidir } A_{MT}$$

- O que é um absurdo, pois sabemos que A_{MT} é **indecidível**.

Chegamos em um absurdo.

Logo, a **hipótese era falsa**, e não pode existir uma **MT** R que **decide** a linguagem V_{MT}.

- Portanto,

$$V_{MT} = \{\langle M \rangle \mid M \text{ é uma MT e } L(M) = \varnothing\}$$

não é Turing-decidível.

O que equivale a dizer que esse problema não tem solução computacional.

Problemas insolúveis

Turing-indecidíveis	**Turing-irreconhecíveis**
$A_{MT}, \overline{A_{MT}}$	$\overline{A_{MT}}$
$PARA_{MT}$	
V_{MT}	

Fim

Dúvidas?

Roteiro

Referências

Referências:

1. *"Introdução à Teoria da Computação"* de M. Sipser, 2007.
2. *"Introdução à Teoria de Autômatos, Linguagens e Computação"* de J. E. Hopcroft, R. Motwani, e J. D. Ullman, 2003.