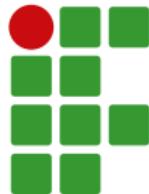


# Redutibilidade

Teoria da Computação – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes



**INSTITUTO  
FEDERAL**  
Brasília

---

Campus  
Taguatinga

# Sumário

Introdução

Turing-redutibilidade

Histórico

Redutibilidade por mapeamento

# Sumário

Introdução

Turing-redutibilidade

Histórico

Redutibilidade por mapeamento

# Introdução

- ▶ Anteriormente estabelecemos as Máquinas de Turing como modelo padrão de computação de propósito-geral.
- ▶ Apresentamos vários problemas que são decidíveis utilizando *MTs*.
- ▶ Mas também apresentamos problemas indecidíveis, como o problema  $A_{MT}$ , o problema da parada.
- ▶ Veremos agora como utilizar problemas que sabemos que são indecidíveis para mostrar que outros problemas também são.
- ▶ Utilizaremos o método de **redução** entre problemas.

# Reduções

- ▶ Uma redução é uma maneira de converter um problema em outro de forma que a solução para o segundo problema pode ser utilizada para resolver o primeiro problema.
- ▶ Podemos utilizar esse conceito informal para fazer uma analogia com o dia-a-dia.

# Reduções

## Exemplo

- ▶ Suponha que queiramos nos localizar em uma cidade nova.
- ▶ Localizar seria fácil se tivéssemos um mapa dela.
- ▶ Assim, podemos reduzir o problema de nos localizar na cidade ao problema de obter um mapa.
- ▶ Se resolvemos um segundo, podemos usar a solução para resolver o primeiro.

# Reduções

- ▶ As reduções sempre envolvem dois problemas, chamados  $A$  e  $B$ .
- ▶ Se  $A$  se reduz a  $B$ , podemos utilizar a solução de  $B$  para resolver  $A$ .
- ▶ No nosso exemplo:
  - ▶  $A$  = problema de se localizar na cidade.
  - ▶  $B$  = problema de encontrar um mapa.
- ▶ Note que a redutibilidade não diz nada sobre resolver  $A$  ou  $B$  independentemente, mas sim sobre resolver  $A$  na presença de uma solução de  $B$ .

# Reduções

## Exemplo

- ▶ O problema de viajar de Taguatinga para Paris se reduz ao problema de comprar uma passagem de avião entre as duas cidades.
- ▶ O problema de comprar a passagem se reduz ao problema de arrumar o dinheiro.
- ▶ O problema de arrumar o dinheiro se reduz ao problema de encontrar um emprego.
- ▶ Note que não especificamos como resolver cada problema, mas estamos falando que podemos resolver um problema dada a solução do outro.

# Reduções

- ▶ Reduções também ocorrem em problemas matemáticos.
- ▶ O problema de calcular a área de um retângulo se reduz ao problema de mensurar sua altura e largura.
- ▶ O problema de resolver um sistema linear de equações se reduz ao problema de inverter uma matriz.

# Reduções

- ▶ Reduções apresentam um importante papel em classificar os problemas em decidíveis ou indecidíveis.
- ▶ São importantíssimas em classificar problemas decidíveis em níveis de dificuldade, o que é de interesse para a área de Complexidade Computacional.

# Reduções

- ▶ Se um problema  $A$  se reduz a um problema  $B$ , o que podemos dizer sobre a **dificuldade** de  $B$  em relação a  $A$ ?

# Reduções

- ▶ Resolver  $A$  não pode ser mais difícil do que resolver  $B$ .
- ▶ A partir de uma solução de  $B$ , conseguimos resolver  $A$ .

# Reduções

- ▶ Se  $A$  se reduz a  $B$ , e  $A$  é indecidível, o que podemos dizer sobre  $B$ ?
- ▶ Se  $A$  se reduz a  $B$ , e  $B$  é decidível, o que podemos dizer sobre  $A$ ?

# Reduções

- ▶ Se  $A$  se reduz a  $B$ , e  $A$  é indecidível, o que podemos dizer sobre  $B$ ?
  - ▶  $B$  tem que ser indecidível.
- ▶ Se  $A$  se reduz a  $B$ , e  $B$  é decidível, o que podemos dizer sobre  $A$ ?

# Reduções

- ▶ Se  $A$  se reduz a  $B$ , e  $A$  é indecidível, o que podemos dizer sobre  $B$ ?
  - ▶  $B$  tem que ser indecidível.
- ▶ Se  $A$  se reduz a  $B$ , e  $B$  é decidível, o que podemos dizer sobre  $A$ ?
  - ▶  $A$  tem que ser decidível.

# Reduções

- ▶ Para provar que um problema é indecidível, basta mostrar que outro problema indecidível se reduz a ele.
- ▶ Nosso ponto de partida:  $A_{MT}$ .

# Sumário

Introdução

Turing-redutibilidade

Histórico

Redutibilidade por mapeamento

# Turing-redutibilidade

- ▶ Para podermos utilizar a técnica de redução entre problemas, precisamos formalizá-la.
- ▶ **Turing-redutibilidade.**

# Turing-redutibilidade

## Definição (Máquinas de Turing com oráculo)

*Um **oráculo** para uma linguagem  $B$  é um dispositivo externo que é capaz de dizer, para qualquer string  $w$ , se ela pertence ou não pertence a  $B$ .*

*Uma **Máquina de Turing com oráculo** tem a capacidade adicional de fazer consultas ao oráculo.*

# Turing Redutibilidade

- ▶ Ou seja, uma máquina de Turing com oráculo possui uma caixa preta que pode ser consultada para a Linguagem  $B$ .
- ▶ Esta caixa-preta diz se uma determinada palavra está ou não está em  $B$ , independentemente da indecidibilidade de  $B$ .

# Turing-redutibilidade

## Definição (Decidibilidade relativa)

*Uma linguagem  $B$  é decidível em relação a uma linguagem  $A$ , se existe uma Máquina de Turing com oráculo para  $A$  que é capaz de decidir  $B$ .*

# Turing-redutibilidade

## Definição (Turing-redutibilidade)

*Uma linguagem A é Turing-redutível a uma linguagem B, denotado por  $A \leq_T B$ , se A é decidível em relação a B.*

*Ou seja, se existe uma máquina com oráculo para B de modo que seja possível decidir A com ela, temos que  $A \leq_T B$ .*

# Turing-redutibilidade

- ▶ Com relação a estes conceitos, podemos concluir duas coisas.

# Turing-redutibilidade

## Teorema

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

# Turing-redutibilidade

Demonstração.

- ▶ Suponha que  $A \leq_T B$ , ou seja,  $A$  é decidível por através de uma máquina com oráculo para  $B$ .
- ▶ Se  $B$  é decidível então podemos trocar o oráculo de  $B$  por um procedimento que decida  $B$ , sem utilizar o oráculo para  $B$ .
- ▶ Assim, temos uma máquina de Turing que decide  $A$  ao eliminar este oráculo.



# Turing-redutibilidade

## Corolário

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

# Turing-redutibilidade

## Demonstração.

- ▶ Tome  $A \leq_T B$  e  $A$  indecidível e suponha  $B$  decidível.
- ▶ Se  $B$  é decidível, podemos trocar o oráculo que  $A$  utiliza para se reduzir a  $B$  pela própria máquina que decide  $B$ .
- ▶ Dessa forma, podemos decidir  $A$  através de  $B$  sem utilizar o oráculo.
- ▶ Impossível, pois  $A$  é indecidível.
- ▶  $\therefore B$  tem que ser indecidível.



# Turing-redutibilidade

- ▶ Agora que temos este conceito de Turing-redutibilidade, podemos utilizá-lo para mostrar que outros problemas são indecidíveis.
- ▶ Vamos utilizar da estrutura da prova do nosso último corolário e de  $A_{MT}$  para isto.

# Problemas indecidíveis

- ▶ Sabemos que  $A_{MT}$  é indecidível.
- ▶ Vamos considerar um problema parecido:  $HALT_{MT}$ .
- ▶  $HALT_{MT}$  concentra-se em determinar se uma Máquina de Turing para (aceitando ou rejeitando) ou não em uma determinada entrada:

$$HALT_{MT} = \{\langle M, w \rangle | M \text{ é uma MT e } M \text{ para sobre } w\}$$

# Problemas indecidíveis

- ▶ Provaremos que  $\text{HALT}_{MT}$  é indecidível ao mostrarmos uma redução de  $A_{MT}$  para  $\text{HALT}_{MT}$ .

# Problemas indecidíveis

## Teorema

$\text{HALT}_{\text{MT}}$  é *indecidível*.

# Problemas indecidíveis

## Ideia da prova

- ▶ A prova será por contradição.
- ▶ Assumiremos que  $\text{HALT}_{\text{MT}}$  é decidível e utilizaremos este fato para chegar a conclusão que  $A_{\text{MT}}$  é decidível, gerando um absurdo.

# Problemas indecidíveis

## Ideia da prova

- ▶ Suponha que temos uma MT  $R$  que decide  $\text{HALT}_{\text{MT}}$ .
- ▶ Podemos utilizar  $R$  para construir  $S$ , uma MT que decide  $A_{\text{MT}}$ .
- ▶ Uma abordagem inicial para construção de  $S$  é simular  $M$  sobre  $w$ .
- ▶ Se  $M$  aceita  $w$ ,  $S$  aceita.
- ▶ Se  $M$  rejeita  $w$  ou entra em loop,  $S$  deve dizer rejeita.
- ▶ O problema é: pela simulação não da conseguimos dizer se uma máquina está em loop.

# Problemas indecidíveis

## Ideia da prova

- ▶ Esta abordagem inicial não funciona.
- ▶ Vamos utilizar  $R$  ao nosso favor.
- ▶ Com  $R$ , testamos se  $M$  para sobre  $w$ :
  - ▶ Se  $R$  diz aceita, significa que  $M$  para sobre  $M$  (aceita ou rejeita). Fazemos a simulação de  $M$  sobre  $w$  e pegamos a resposta da simulação como a resposta de  $S$ .
  - ▶ Se não para, quer dizer que  $M$  entra em loop sobre  $w$ , e portanto  $M$  não aceita  $w$ . Logo,  $S$  deve dizer rejeita.

# Problemas indecidíveis

## Ideia da prova

- ▶ Assim, se a MT  $R$  existe, podemos decidir  $A_{\text{MT}}$  através de  $S$ .
- ▶ Mas sabemos que  $A_{\text{MT}}$  é indecidível.
- ▶ Contradição.
- ▶  $R$  não pode existir.
- ▶  $\therefore \text{HALT}_{\text{MT}}$  é indecidível.

# Problemas indecidíveis

- ▶ Agora podemos ir para a prova.

# Problemas indecidíveis

## Demonstração

---

### Algorithm 1: Construção da máquina $S$ que decide $A_{\text{MT}}$

---

**Input:**  $\langle M, w \rangle$ , uma descrição de  $M$  e uma entrada  $w$

**Output:** *aceita*, caso  $M$  aceita  $w$  e *rejeita* caso contrário .

1Rode  $R$  sobre a entrada  $\langle M, w \rangle$

2`if( R rejeita )`

3`return rejeita`

4`else`

5Simule  $M$  sobre  $w$  até  $M$  parar.

6`if( M aceita w )`

7`return aceita`

8`else`

9`return rejeita`

---

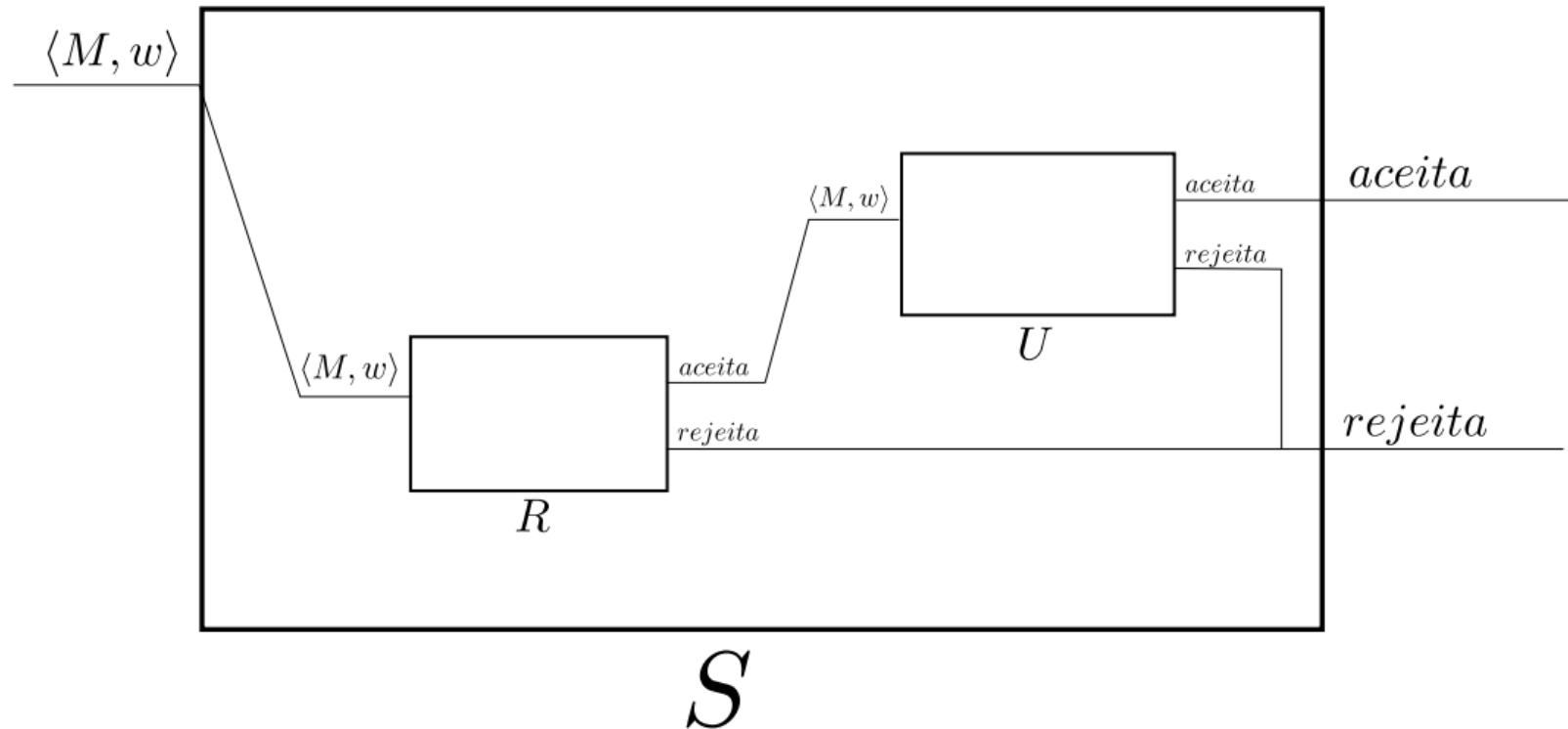
# Problemas indecidíveis

## Demonstração.

- ▶ Como  $R$  decide  $\text{HALT}_{\text{MT}}$ , mostramos como construir  $S$  que decide  $A_{\text{MT}}$ .
- ▶ Mas  $A_{\text{MT}}$  é indecidível.
- ▶  $R$  não pode existir.
- ▶  $\therefore \text{HALT}_{\text{MT}}$  é **indecidível**.



# Problemas indecidíveis



# Problemas indecidíveis

- ▶ A mesma prova pode ser feita utilizando a nossa definição de Turing-redutibilidade.
- ▶ Basta mostrar que, se tivermos uma máquina com oráculo  $R$  para  $\text{HALT}_{\text{MT}}$ , conseguimos decidir  $A_{\text{MT}}$ .
- ▶ Feito isso teremos que  $A_{\text{MT}} \leq_T \text{HALT}_{\text{MT}}$ , e como  $A_{\text{MT}}$  é indecidível, pelo teorema visto anteriormente,  $\text{HALT}_{\text{MT}}$  tem que ser indecidível também.

# Problemas indecidíveis

---

**Algorithm 2:** Mostrando que  $A_{\text{MT}} \leq_T \text{HALT}_{\text{MT}}$ 

**Input:**  $\langle M, w \rangle$

**Output:** aceita, caso  $M$  aceita  $w$  e rejeita caso contrário

1 Consulte o oráculo  $R$  para verificar se  $M$  para sobre  $w$

2 **if**( Se  $M$  para sobre  $w$  )

3 Simule  $M$  sobre  $w$

4 **if**(  $M$  aceita  $w$  )

5 **return** aceita

6 **else**

7 **return** rejeita

8 **else**

9 **return** rejeita

---

# Problemas indecidíveis

- ▶ Essas estratégias de prova são muito comuns para mostrar que certos problemas são indecidíveis.
- ▶ Com exceção do  $A_{\text{MT}}$ , que foi provado diretamente via método da diagonalização, podemos mostrar que outros problemas são indecidíveis via redução.
- ▶ Vamos mostrar que outro problema é indecidível utilizando a mesma estratégia.

# Problemas indecidíveis

- ▶ Tome o seguinte problema:

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

- ▶  $E_{MT}$  é a linguagem das máquinas que não aceitam nada.

# Problemas indecidíveis

## Teorema

$E_{MT}$  é *indecidível*.

# Problemas indecidíveis

## Ideia da prova

- ▶ Utilizaremos o mesmo *framework* da demonstração de indecidibilidade de  $\text{HALT}_{\text{MT}}$ .
- ▶ É uma prova por contradição.
- ▶ Vamos supor que  $E_{\text{MT}}$  é decidível e acabar concluindo que  $A_{\text{MT}}$  é decidível, o que é um absurdo.
- ▶ Dessa forma conclui-se que  $E_{\text{MT}}$  é indecidível.

# Problemas indecidíveis

## Ideia da prova

- ▶ Suponha a existência de uma máquina  $R$  que decide  $E_{\text{MT}}$ .
- ▶ Vamos utilizar  $R$  para construir uma máquina  $S$  que decide  $A_{\text{MT}}$ .
- ▶ Como construir  $S$ ?

# Problemas indecidíveis

## Ideia da prova

- ▶ Uma primeira abordagem para construção de  $S$  é rodar  $R$  sobre a entrada  $\langle M \rangle$  e verificar se  $R$  a aceita.
- ▶ Se  $R$  aceita  $\langle M \rangle$ , sabemos que  $L(M) = \emptyset$ , e portanto,  $S$  deve dizer rejeita.
- ▶ Se  $R$  rejeita  $\langle M \rangle$ , então  $L(M) \neq \emptyset$ , mas não sabemos se uma string  $w$  em particular é aceita por  $M$ , que é o que  $A_{MT}$  justamente quer.
- ▶ Precisamos de outra abordagem, esta não funciona.

# Problemas indecidíveis

## Ideia da prova

- ▶ Em vez de rodar  $R$  sobre  $\langle M \rangle$  rodaremos  $R$  sobre uma versão modificada de  $\langle M \rangle$ .
- ▶ Modificamos  $\langle M \rangle$  de modo que  $M$  rejeite todas as strings diferentes de  $w$ .
- ▶ Se a string for  $w$ , a versão modificada funciona como  $M$ .
- ▶ Você consegue construir tal máquina?

# Problemas indecidíveis

## Ideia da prova

- ▶ A ideia é fazer com que  $w$  faça parte da descrição da máquina modificada.
- ▶ Assim, para qualquer entrada  $x$ , basta compará-la com a palavra  $w$  embutida na descrição.
- ▶ Se as palavras não batem, a máquina modificada rejeita.
- ▶ Se as palavras batem, então  $x = w$  e a máquina modificada funciona como a máquina original.
- ▶ Só precisamos inserir alguns estados a mais na máquina original para efetuar esta comparação.

# Problemas indecidíveis

## Ideia da prova

- ▶ A versão modificada de  $\langle M \rangle$  rejeita todas as strings diferentes de  $w$ .
- ▶ Se  $R$  rejeita sobre a versão modificada de  $\langle M \rangle$ , sabemos que  $S$  deve aceitar a entrada  $\langle M, w \rangle$ .
- ▶ Se  $R$  aceita sobre a versão modificada de  $\langle M \rangle$ , sabemos que  $M$  não aceita  $w$ , e portanto,  $S$  deve rejeitar sobre a entrada  $\langle M, w \rangle$ .

# Problemas indecidíveis

- ▶ Agora podemos partir para a demonstração.

# Problemas indecidíveis

Demonstração.

- ▶ Chame a versão modificada de  $\langle M \rangle$  de  $M_1$ .

---

**Algorithm 3:** Máquina  $M_1$

---

**Input:**  $x \in \Sigma^*$

**Output:** aceita se  $x = w$  e rejeita caso contrário

1**if**(  $x \neq w$  )

2**return** rejeita

3**else**

4**Roda**  $M$  sobre  $w$  e aceita, se  $M$  aceita  $w$ .

---

# Problemas indecidíveis

Demonstração.

- Dado  $M_1$  agora podemos construir  $S$ .

---

### Algorithm 4: Construção de $S$

---

**Input:**  $\langle M, w \rangle$

**Output:** aceita se  $M$  aceita  $w$  e rejeita, caso contrário.

1 Utilize  $\langle M, w \rangle$  para construir  $M_1$

2 Rode  $R$  sobre  $\langle M_1 \rangle$

3 if(  $R$  aceita )

4 return rejeita

5 else

6 return aceita

---

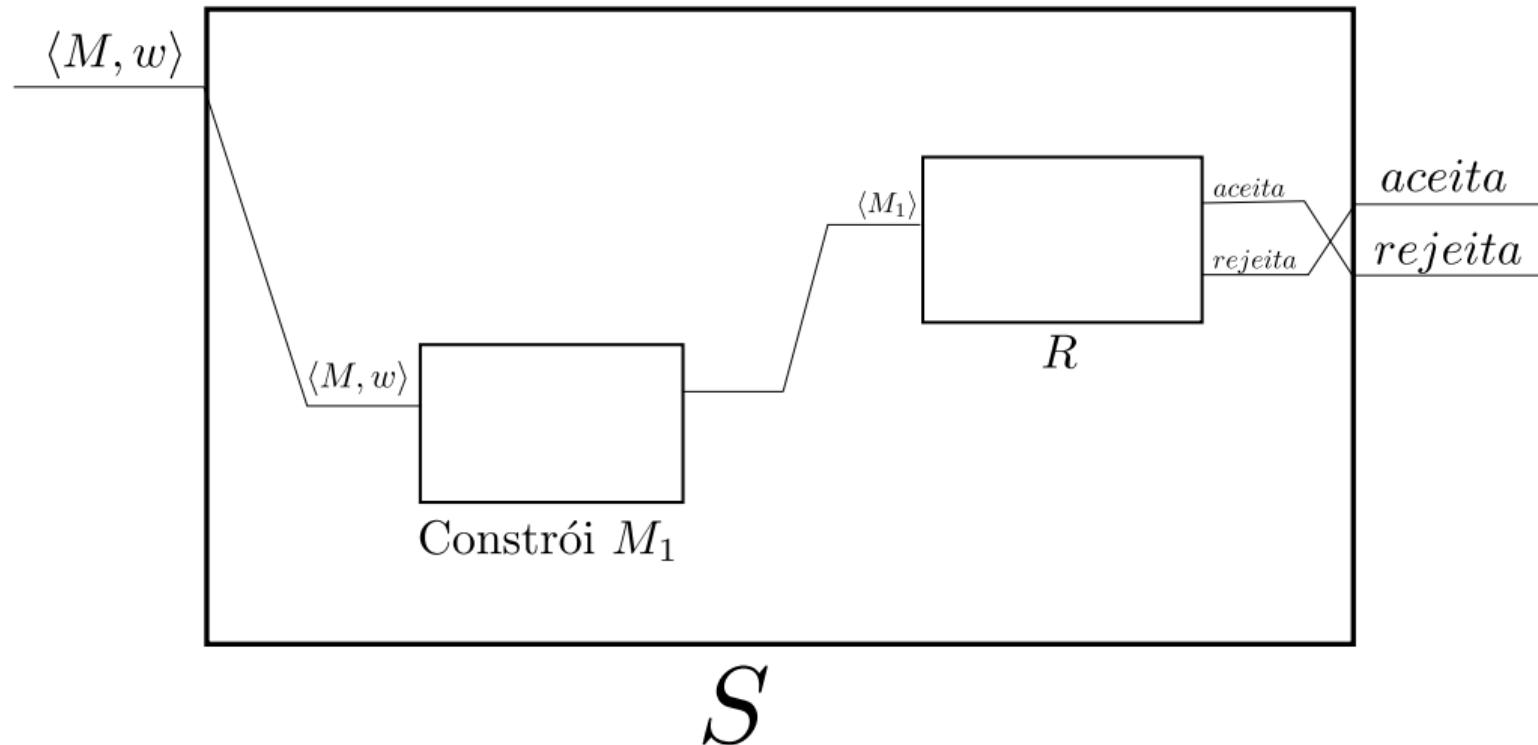
# Problemas indecidíveis

## Demonstração.

- ▶ Dado que  $R$  decide  $E_{MT}$ , mostramos que  $S$  decide  $A_{MT}$ .
- ▶ Mas  $A_{MT}$  é indecidível.
- ▶ A máquina  $R$  não pode existir.
- ▶ ∴  $E_{MT}$  é indecidível.



# Problemas indecidíveis



# Problemas indecidíveis

- ▶ Outra forma, é utilizar a mesma estratégia da redução de  $A_{\text{MT}}$  para  $\text{HALT}_{\text{MT}}$ .
- ▶ Assumimos que existe uma máquina com oráculo para  $E_{\text{MT}}$  e decidimos  $A_{\text{MT}}$ .
- ▶ Isso prova que  $A_{\text{MT}} \leq_T E_{\text{MT}}$  e portanto que  $E_{\text{MT}}$  é indecidível.

# Problemas indecidíveis

---

**Algorithm 5:** Mostrando que  $A_{\text{MT}} \leq_T E_{\text{MT}}$ 

**Input:**  $\langle M, w \rangle$  e  $R$ , uma máquina com oráculo para  $E_{\text{MT}}$

**Output:** *aceita*, caso  $M$  aceita  $w$  e *rejeita* caso contrário

- 1 Crie a máquina  $M_1$  como descrito na prova anterior
  - 2 Consulte o oráculo  $R$  para verificar se  $M_1$  não aceita nada
  - 3 **if** ( Se  $M_1$  não aceita nada )  
4 **return** *rejeita*
  - 5 **else**  
6 **return** *aceita*
-

# Problemas indecidíveis

- ▶ Até agora utilizamos a redução a partir de  $A_{MT}$  para mostrar que outro problema é indecidível.
- ▶ Às vezes, é mais fácil provar um teorema sobre indecidibilidade se partirmos de outro problema.
- ▶ Mostraremos agora que um problema é indecidível ao reduzirmos  $E_{MT}$  para ele.

# Problemas indecidíveis

- ▶ Tome o problema  $\text{EQ}_{\text{MT}}$ :

$$\text{EQ}_{\text{MT}} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ e } M_2 \text{ são MTs e } L(M_1) = L(M_2)\}$$

- ▶ Ou seja, queremos saber se dois programas possuem o mesmo comportamento.

# Problemas indecidíveis

## Teorema

$\text{EQ}_{\text{MT}}$  é *indecidível*

# Problemas indecidíveis

## Ideia da prova

- ▶ A prova é por contradição.
- ▶ Vamos mostrar que, se  $\text{EQ}_{\text{MT}}$  é decidível, conseguimos decidir  $E_{\text{MT}}$ .
- ▶ O que é um absurdo, pois sabemos que  $E_{\text{MT}}$  é indecidível.
- ▶ Assim concluímos que  $\text{EQ}_{\text{MT}}$  é indecidível.

# Problemas indecidíveis

## Ideia da prova

- ▶  $E_{MT}$  é o problema de determinar se  $L(M)$  é vazia para alguma máquina  $M$ .
- ▶  $EQ_{MT}$  é o problema de determinar se  $L(M_1) = L(M_2)$  para máquinas  $M_1$  e  $M_2$ .
- ▶ Assim, se  $L(M_1) = \emptyset$ , e a resposta de  $EQ_{MT}$  sobre a entrada  $\langle M_1, M_2 \rangle$  é **aceita**, conseguimos concluir que  $L(M_2) = \emptyset$ .
- ▶ O  $E_{MT}$  é um caso especial de  $EQ_{MT}$ .

# Problemas indecidíveis

## Demonstração.

- ▶ Tome como  $R$  a máquina que decide  $\text{EQ}_{\text{MT}}$ .
- ▶ Podemos construir  $S$  a partir de  $R$  da seguinte maneira.

# Problemas indecidíveis

Demonstração.

---

**Algorithm 6:** Construção de  $S$  que decide  $E_{\text{MT}}$ .

---

**Input:**  $\langle M \rangle$

**Output:** aceita, se  $L(M) = \emptyset$  e rejeita caso contrário.

1 Rode  $R$  sobre a entrada  $\langle M, M_1 \rangle$ , onde  $M_1$  é uma MT que rejeita qualquer entrada

2 **if**(  $R$  aceita )

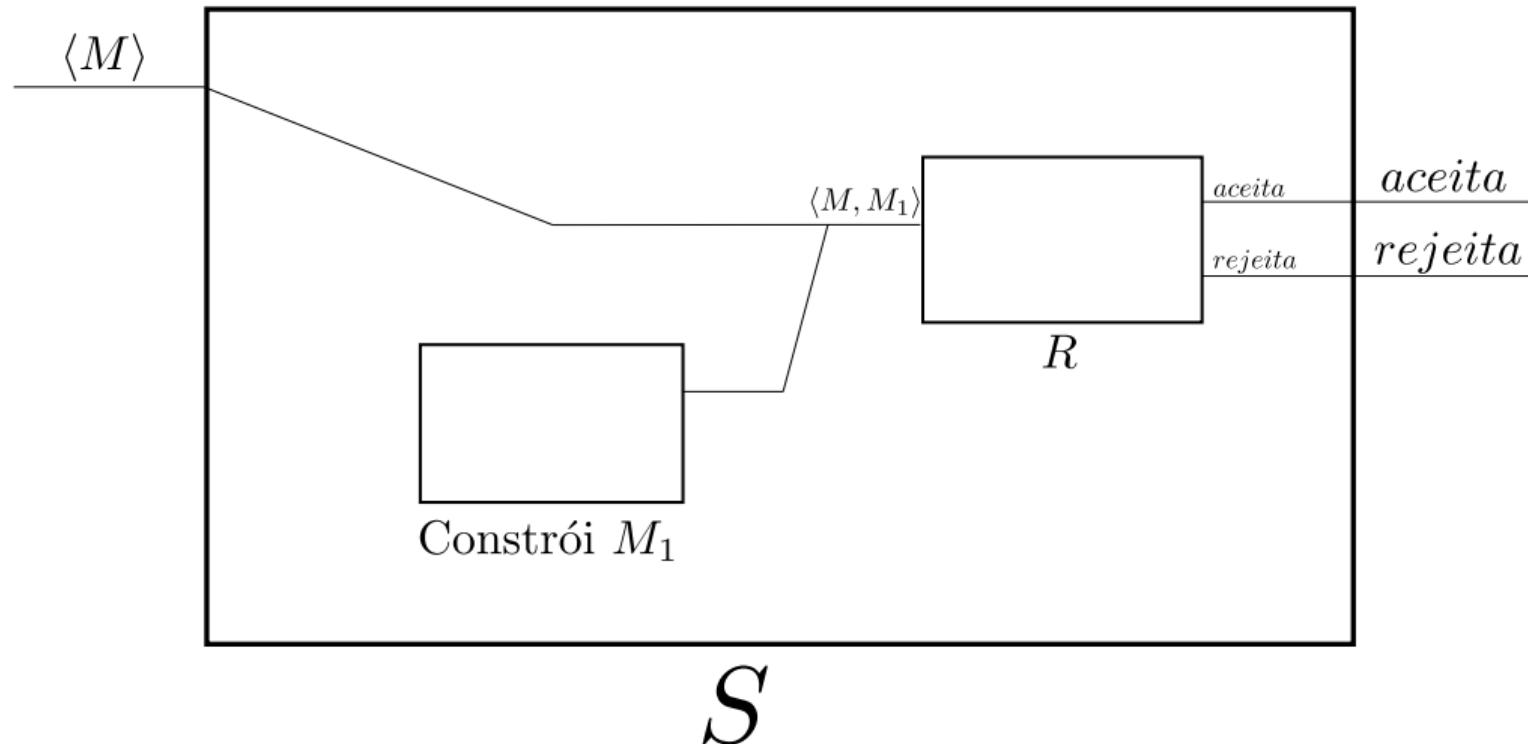
3 **return** aceita

4 **else**

5 **return** rejeita

---

# Problemas indecidíveis



# Sumário

Introdução

Turing-redutibilidade

Histórico

Redutibilidade por mapeamento

# Reduções via histórico de computação

- ▶ O método de histórico de computação é importante para mostrar que  $A_{MT}$  é redutível a outras linguagens.
- ▶ Este método é frequentemente utilizado quando o problema a ser provado indecidível envolve a busca da existência de algo.
- ▶ Por exemplo, este método pode ser utilizado para provar a indecidibilidade do décimo problema de Hilbert: buscar a existência de raízes inteiras de um polinômio.

# Reduções via histórico de computação

- ▶ O histórico de computação de uma máquina sobre uma entrada é simplesmente a sequência de configurações que a máquina tem ao processar uma entrada.
- ▶ Imagine que a cada função de transição, uma foto seja tirada da situação da máquina.
- ▶ É um registro completo da computação da máquina.

# Reduções via histórico de computação

- ▶ Históricos de computação são sequências **finitas**.
- ▶ Se uma máquina  $M$  não para sobre uma entrada  $w$ , dizemos que o histórico de computação não existe para  $M$  sobre  $w$ .
- ▶ Máquinas determinísticas possuem no máximo um histórico de computação para uma determinada entrada.
- ▶ Máquinas não-determinísticas podem possuir múltiplos históricos para uma mesma entrada.

# Reduções via histórico de computação

- ▶ Vamos definir a noção de histórico mais precisamente.

# Reduções via histórico de computação

## Definição (Histórico de computação)

- ▶ Seja  $M$  uma MT e  $w$  uma entrada.
- ▶ Um histórico de computação de aceitação de  $M$  em  $w$  é uma sequência de configurações  $C_1, C_2, \dots, C_l$ , onde  $C_1$  é a configuração inicial de  $M$  em  $w$  e  $C_l$  é uma configuração de aceitação de  $M$ .
- ▶ Cada  $C_i$  é derivado de  $C_{i-1}$  respeitando as regras de  $M$ .
- ▶ Similarmente, um histórico de computação de rejeição de  $M$  em  $w$  tem como  $C_l$  uma configuração de rejeição.

# Reduções via histórico de computação

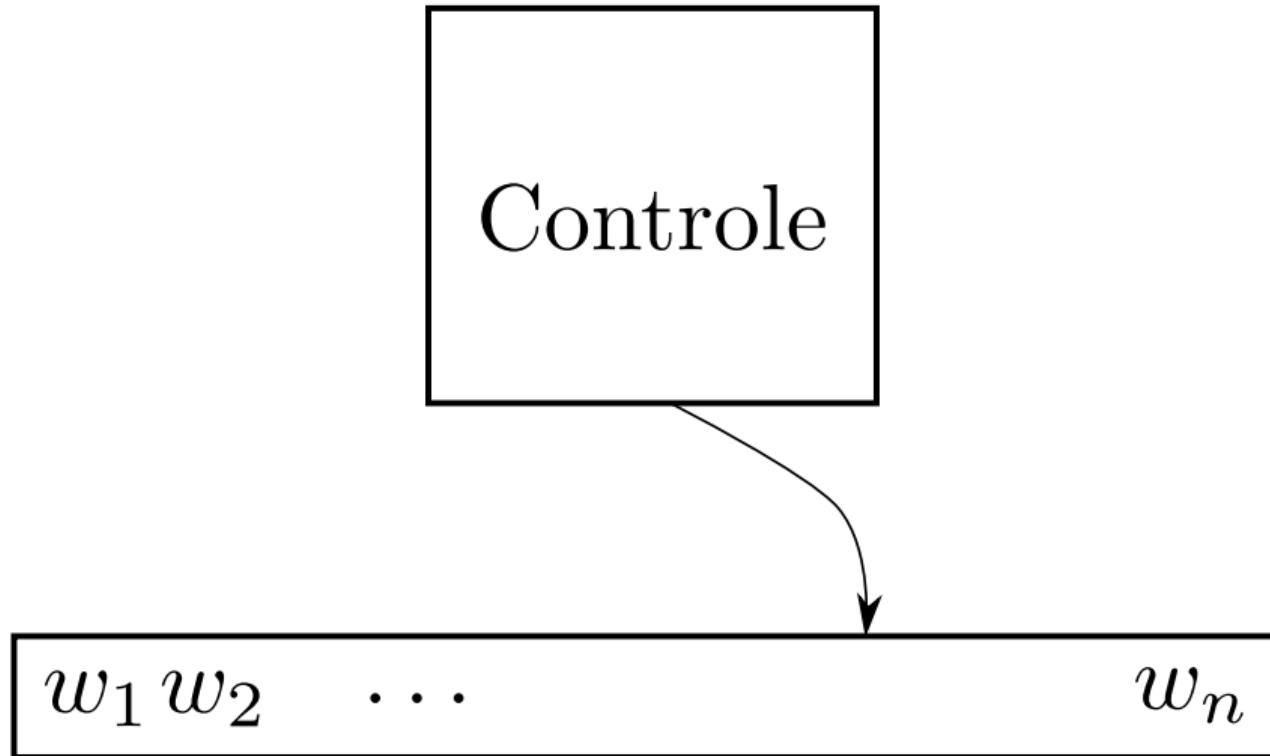
- ▶ Agora que sabemos o que é um histórico de computação podemos atacar problemas e mostrá-los indecidíveis.
- ▶ Vamos mostrar que um problema relacionado a um **autômato linearmente limitado** é indecidível via este método.

# Reduções via histórico de computação

## Definição (Autômato linearmente limitado)

- ▶ Um autômato linearmente limitado (*LBA* — *linear bounded automata*) é um tipo restrito de *MT*.
- ▶ A cabeça de leitura/escrita não pode se mover além do espaço delimitado pela entrada.
- ▶ Se a máquina tenta se mover para além de qualquer uma das extremidades da entrada, a cabeça continua na mesma posição.

# Autômato linearmente limitado



# Autômato linearmente limitado

- ▶ Apesar de ter memória limitada, um autômato linearmente limitado é bem poderoso.
- ▶ Vamos mostrar um primeiro resultado sobre ele.
- ▶ Tome a linguagem:

$$\text{ALBA} = \{\langle M, w \rangle \mid M \text{ é um LBA que aceita } w\}$$

# Autômato linearmente limitado

## Teorema

$A_{LBA}$  é *decidível*.

# Autômato linearmente limitado

- ▶ Para mostrar que  $A_{LBA}$  é decidível, vamos precisar provar um lema antes.

# Autômato linearmente limitado

## Lema

*Seja  $M$  um LBA com  $q$  estados e  $g$  símbolos em  $\Gamma$ , o alfabeto da fita. Existem no máximo  $qng^n$  configurações distintas de  $M$  para uma fita de comprimento  $n$ .*

# Autômato linearmente limitado

## Demonstração

- ▶ Lembre-se de que uma configuração é como se fosse uma foto do estado da máquina.
- ▶ Ela consiste de:
  - ▶ Estado.
  - ▶ Posição da cabeça.
  - ▶ Conteúdo da fita.

# Autômato linearmente limitado

## Demonstração

- ▶  $M$  tem  $q$  estados.
- ▶ O comprimento da fita é  $n$ , então a cabeça só pode ocupar  $n$  posições distintas.
- ▶ O conteúdo da fita possui  $g^n$  possibilidades.

# Autômato linearmente limitado

## Demonstração.

- ▶ O número total de combinações equivale ao produto das três quantidades.
- ▶  $qng^n$ .



# Autômato linearmente limitado

- ▶ Vamos provar agora o teorema sobre a decidibilidade de  $A_{LBA}$ .

# Autômato linearmente limitado

## Ideia da prova

- ▶ Para decidir de um LBA  $M$  aceita  $w$ , simulamos  $M$  sobre  $w$ .
- ▶ Durante a simulação, se  $M$  para e aceita/rejeita, aceitamos ou rejeitamos de acordo.
- ▶ O problema está quando  $M$  entra em loop sobre  $w$ .
- ▶ Temos que conseguir detectar loops, uma coisa que é impossível em MTs.

# Autômato linearmente limitado

## Ideia da prova

- ▶ No entanto, em LBAs, conseguimos detectar um loop.
- ▶ Se  $M$  repete alguma configuração, podemos concluir que ela repetirá essa configuração após algumas outras configurações.
- ▶ Pelo nosso lema, temos um número **finito** de configurações.
- ▶ Se após  $qng^n$  configurações a máquina não tiver parado, ela tem que estar em loop.

# Autômato linearmente limitado

Demonstração.

---

## Algorithm 7: Algoritmo que decide $A_{LBA}$

---

**Input:**  $\langle M, w \rangle$  tal que  $M$  é um LBA.

**Output:** aceita se  $M$  aceita  $w$ .

```
1Simule  $M$  sobre  $w$  por  $qng^n$  passos ou até que ela pare  
2if(  $M$  parou )  
3if(  $M$  aceitou  $w$  )  
4return aceita  
5else  
6return rejeita  
7else  
8return rejeita
```

---



# Autômato linearmente limitado

- ▶ Este teorema mostra uma diferença fundamental entre os LBAs e as MTs.
- ▶ Enquanto  $A_{LBA}$  é decidível,  $A_{MT}$  não é.
- ▶ Modelos diferentes com resultados diferentes no que tange o problema da aceitação.

# Autômato linearmente limitado

- ▶ Agora que ganhamos uma intuição sobre os LBAs, estamos prontos para mostrar resultados de indecidibilidade sobre eles.
- ▶ Tome a linguagem:

$$E_{LBA} = \{\langle M \rangle \mid M \text{ é um LBA tal que } L(M) = \emptyset\}$$

- ▶ Ou seja, queremos saber se a linguagem reconhecida por um LBA é vazia.
- ▶ Este problema é indecidível.

# Autômato linearmente limitado

## Teorema

$E_{LBA}$  é *indecidível*.

# Autômato linearmente limitado

## Ideia da prova

- ▶ Esta prova é obtida através da redução de  $A_{MT}$ .
- ▶ Se supomos que  $E_{LBA}$  é decidível através de um algoritmo  $R$ , como podemos mostrar que  $A_{MT}$  é decidível e gerar uma contradição?
- ▶ Dado uma entrada  $\langle M, w \rangle$  a ideia é construir um LBA  $B$  que é aceito por  $R$ , se e somente se,  $M$  não aceita  $w$ .
- ▶ Se  $R$  rejeita  $\langle B \rangle$ , então  $L(B) \neq \emptyset$ , e concluímos que  $A_{MT}$  diz aceita.
- ▶ Se  $R$  aceita  $\langle B \rangle$ , então  $L(B) = \emptyset$ , e concluímos que  $A_{MT}$  diz rejeita.

# Autômato linearmente limitado

## Ideia da prova

- ▶ Como construir  $B$  de  $M$  e  $w$ ?
- ▶ Vamos construir  $B$  para aceitar uma entrada  $x$  se  $x$  é um histórico de computação de aceitação para  $M$  sobre  $w$ .
- ▶ Lembre-se que um histórico de computação de aceitação é uma sequência de configurações  $C_1, C_2, \dots, C_l$  que  $M$  passa e aceita  $w$ .

# Autômato linearmente limitado

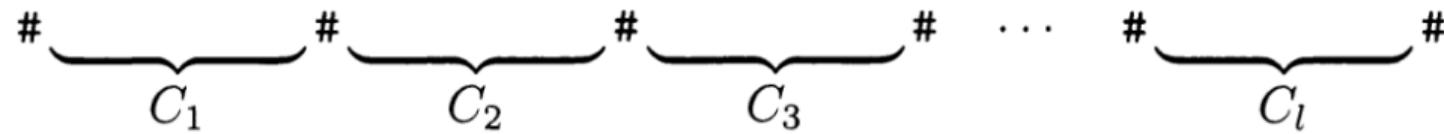


Figura: Uma possível entrada para  $B$

# Autômato linearmente limitado

## Ideia da prova

- ▶  $B$  funciona da seguinte maneira. Quando recebe  $x$ ,  $B$  aceita  $x$  se  $x$  é um histórico de computação de aceitação de  $M$  sobre  $w$ .
- ▶  $B$  então quebra cada configuração através de delimitadores  $\#$  e determina se as configurações satisfazem as condições de um histórico de computação de aceitação:
  - ▶  $C_1$  é a configuração inicial de  $M$  sobre  $w$ .
  - ▶ Cada  $C_{i+1}$  pode ser derivado de  $C_i$ .
  - ▶  $C_l$  é uma configuração de aceitação de  $M$ .

# Autômato linearmente limitado

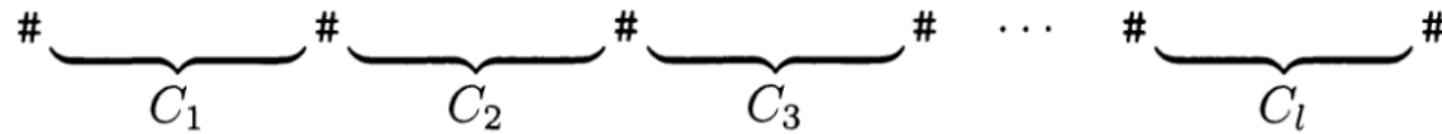


Figura: Uma possível entrada para  $B$

# Autômato linearmente limitado

## Ideia da prova

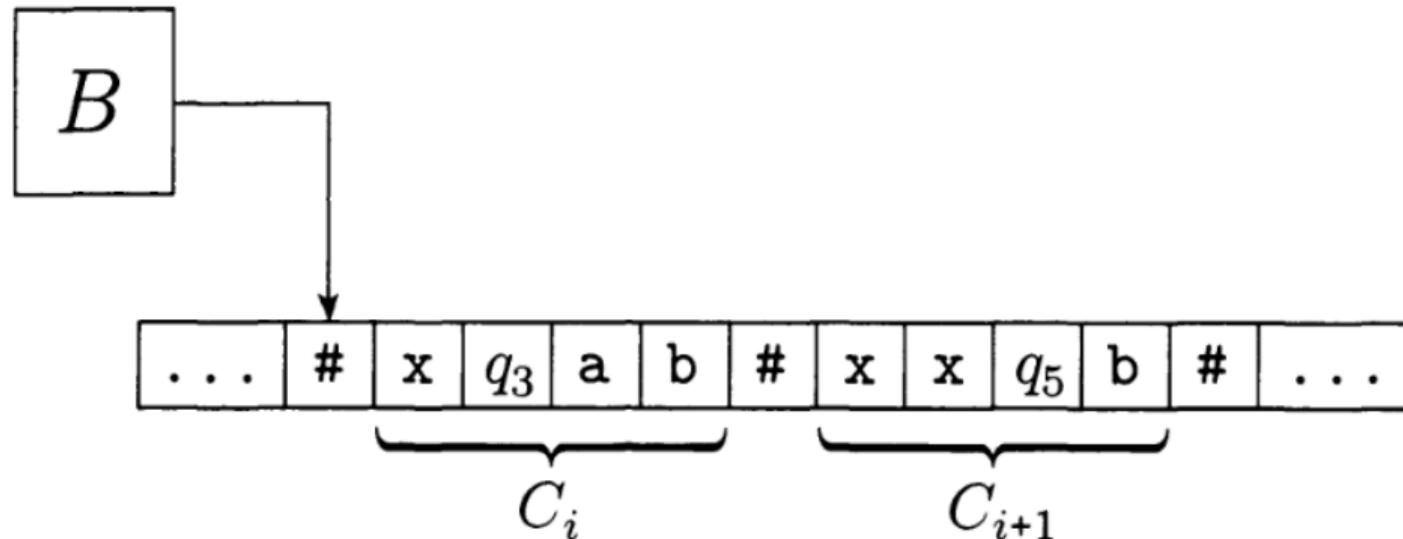
- ▶ A configuração inicial de  $C_1$  para  $M$  sobre  $w$  é a palavra  $q_0w_1w_2 \dots w_n$ , onde  $q_0$  é o estado inicial de  $M$  sobre  $w$ .
- ▶ Para verificar se  $C_l$  é uma configuração de aceitação, basta varrer e verificar se ela contém o estado  $q_{aceita}$ .
- ▶ A segunda condição é a mais difícil. Para cada par adjacente de configurações,  $B$  tem que checar se  $C_{i+1}$  pode decorrer de  $C_i$ .
- ▶ Isso envolve verificar se  $C_i$  e  $C_{i+1}$  são idênticas exceto pela célula apontada sobre a cabeça e adjacente a cabeça de  $C_i$ .
- ▶ Estas células tem que obrigatoriamente ser atualizadas de acordo com  $\delta$  de  $M$ .

# Autômato linearmente limitado

## Ideia da prova

- ▶  $B$  verifica isso ao zigzaguear sobre  $C_i$  e  $C_{i+1}$ .
- ▶ De modo a marcar as posições das fitas,  $B$  utiliza símbolos com um ponto em cima.
- ▶ Finalmente, se as três condições são aceitas,  $B$  aceita a sua entrada.

# Autômato linearmente limitado



# Autômato linearmente limitado

## Ideia da prova

- ▶ O propósito da criação de  $B$  não é rodar sobre uma entrada.
- ▶ É simplesmente servir de entrada para  $R$ .
- ▶ Uma vez que  $R$  retorna uma resposta, é possível dar uma resposta para  $A_{MT}$ .

# Autômato linearmente limitado

## Demonstração

---

### Algorithm 8: Demonstração de indecidibilidade de $E_{LBA}$

---

**Input:**  $\langle M, w \rangle$ , tal que  $M$  é uma MT e  $w$  é uma entrada

**Output:** aceita, se e somente se  $M$  aceita  $w$

1 Construa o LBA  $B$  de  $M$  e  $w$  de acordo com a ideia da prova

2 Rode  $R$  sobre  $\langle B \rangle$

3 if(  $R$  rejeita )

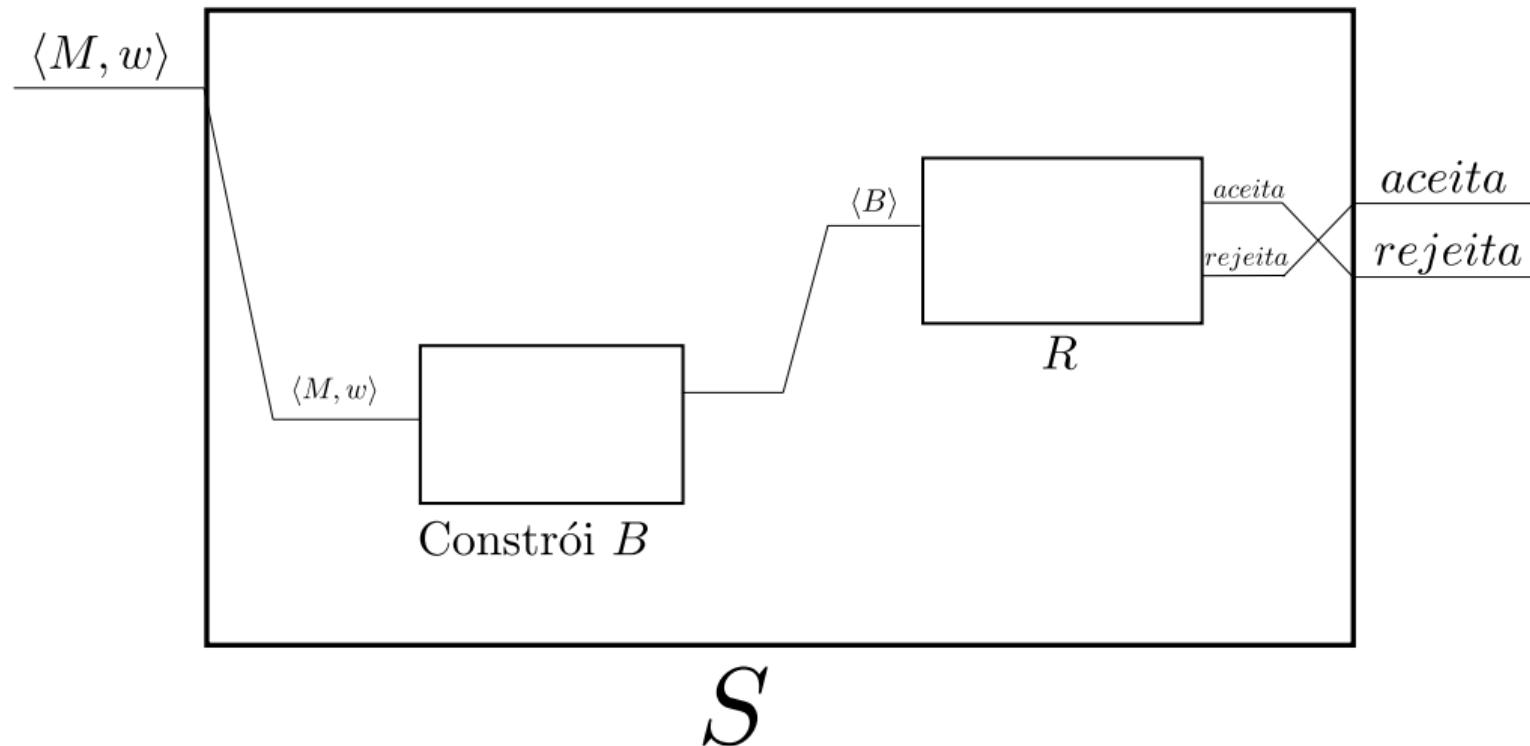
4 return aceita

5 else

6 return rejeita

---

# Autômato linearmente limitado



# Sumário

Introdução

Turing-redutibilidade

Histórico

Redutibilidade por mapeamento

# Redutibilidade por mapeamento

- ▶ Mostramos como utilizar reduções para provar que vários problemas são indecidíveis.
- ▶ Até o momento, utilizamos um tipo de redução chamada de Turing-redução.
- ▶ Mostraremos agora uma noção mais forte de redução, denominada *redução por mapeamento*.

# Redutibilidade por mapeamento

- ▶ A noção de reduzir um problema a outro pode ser definido formalmente de diversas formas.
- ▶ Estudaremos a **redutibilidade por mapeamento**.
- ▶ A grosso modo, ao utilizar a redutibilidade por mapeamento para reduzir um problema  $A$  para um problema  $B$  significa que existe uma **função computável** que converte instâncias do problema  $A$  em instâncias do problema  $B$ .

# Redutibilidade por mapeamento

- ▶ Se existe esta função computável, chamada de **redução**, podemos resolver  $A$  a partir da solução de  $B$ .
- ▶ Se sabemos resolver  $B$ , basta aplicar a redução nas instâncias de  $A$  para transformá-las em instâncias para  $B$  e resolver  $B$ , e logo,  $A$ .

# Sumário

Redutibilidade por mapeamento

Funções computáveis

Definição formal de redutibilidade por mapeamento

# Funções computáveis

- ▶ Uma máquina de Turing computa uma função  $f$  ao começar com  $x$  na fita e ao parar, deixa na fita  $f(x)$ .

# Funções computáveis

## Definição (Funções computáveis)

*Uma função  $f : \Sigma^* \rightarrow \Sigma^*$  é dita uma **função computável** se existe uma Máquina de Turing  $M$  que, para qualquer  $w \in \Sigma^*$ ,  $M$  para e deixa  $f(w)$  na fita.*

# Funções computáveis

## Exemplo

- ▶ Todas as operações aritméticas em inteiros são funções computáveis.
- ▶ Podemos construir uma máquina que recebe como entrada  $\langle m, n \rangle$  e retorna  $m + n$ .

# Funções computáveis

## Exemplo

- ▶ Funções computáveis também podem ser transformações em descrições de máquinas.
- ▶ Uma função computável  $f$  pode, por exemplo, receber uma entrada  $w \in \Sigma^*$  e retornar a descrição de uma MT  $\langle M' \rangle$  se  $w = \langle M \rangle$  é uma codificação de uma MT.
- ▶ A MT  $M'$  é uma máquina que reconhece a mesma linguagem de  $M$ , mas nunca tenta mover a cabeça de leitura/escrita à esquerda da posição inicial.
- ▶ A função  $f$  consegue isso ao adicionar vários estados na descrição de  $M$  e deve retornar  $\epsilon$ , caso  $w$  não seja uma codificação de MT válida.

# Sumário

Redutibilidade por mapeamento

Funções computáveis

Definição formal de redutibilidade por mapeamento

# Redutibilidade por mapeamento

- ▶ Agora que temos a noção formal de funções computáveis, podemos definir o que é de fato uma redutibilidade por mapeamento.

# Redutibilidade por mapeamento

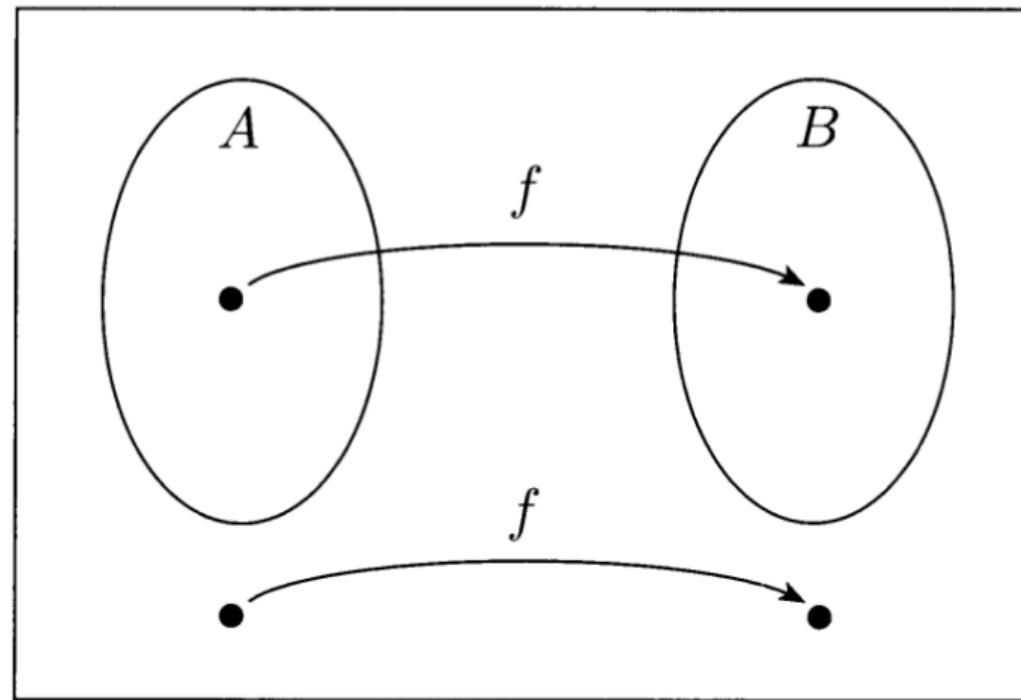
## Definição (Redutibilidade por mapeamento)

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

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

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

# Redutibilidade por mapeamento



# Redutibilidade por mapeamento

- ▶ Uma redução via mapeamento de  $A$  para  $B$  permite uma maneira de converter perguntas sobre a pertinência de uma entrada em  $A$  para uma pertinência de uma entrada em  $B$ .
- ▶ Para responder se  $w \in A$ , utilizamos  $f$  para mapear  $w$  em  $f(w)$  e testar se  $f(w) \in B$ .
- ▶ Em caso afirmativo, temos que  $w \in A$ .
- ▶ Em caso negativo, temos que  $w \notin A$ .
- ▶ Quem garante essa propriedade?

# Redutibilidade por mapeamento

- ▶ Uma redução via mapeamento de  $A$  para  $B$  permite uma maneira de converter perguntas sobre a pertinência de uma entrada em  $A$  para uma pertinência de uma entrada em  $B$ .
- ▶ Para responder se  $w \in A$ , utilizamos  $f$  para mapear  $w$  em  $f(w)$  e testar se  $f(w) \in B$ .
- ▶ Em caso afirmativo, temos que  $w \in A$ .
- ▶ Em caso negativo, temos que  $w \notin A$ .
- ▶ Quem garante essa propriedade?
- ▶ A função de redução.

# Redutibilidade por mapeamento

- ▶ Se um problema é redutível por mapeamento a um segundo problema, cuja solução já é conhecida, podemos obter uma solução para o primeiro problema.
- ▶ Capturamos esta noção com o seguinte teorema.

# Redutibilidade por mapeamento

## Teorema

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

# Redutibilidade por mapeamento

## Demonstração

- ▶ Seja  $M$  uma MT que decide  $B$  e  $f$  uma função de redução de  $A$  para  $B$ . Podemos descrever uma MT  $N$  que decide  $A$  da seguinte maneira:

# Redutibilidade por mapeamento

---

## Algorithm 9: Construção de $N$

---

**Input:**  $w$

**Output:** aceita, se  $w \in A$  e rejeita caso contrário

```
1Compute  $f(w)$ 
2Rode  $M$  sobre  $f(w)$ 
3if(  $f(w) \in B$  )
4return aceita
5else
6return rejeita
```

---

# Redutibilidade por mapeamento

## Demonstração.

- ▶ Claramente, se  $w \in A$ , então  $f(w) \in B$ , visto que  $f$  é uma redução de  $A$  para  $B$ .
- ▶ Assim,  $M$  aceita  $f(w)$  sempre que  $w \in A$  e  $M$  rejeita  $f(w)$  sempre que  $w \notin A$ .
- ▶  $N$  funciona como o esperado.



# Redutibilidade por mapeamento

- ▶ A partir do resultado anterior, podemos chegar no seguinte corolário.

# Redutibilidade por mapeamento

## Corolário

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

# Redutibilidade por mapeamento

- ▶ Vamos agora revisitar as nossas provas anteriores que utilizaram o conceito informal de redução e mostrar como o nosso conceito formal de redução se aplica.

# Redutibilidade por mapeamento

## Exemplo

- ▶ Mostramos anteriormente que  $\text{HALT}_{\text{MT}}$  é indecidível via uma redução de  $A_{\text{MT}}$ .
- ▶ Essa redução mostrou como uma MT que decide  $\text{HALT}_{\text{MT}}$  pode ser utilizada para decidir  $A_{\text{MT}}$ , gerando a esperada contradição.
- ▶ Vamos demonstrar uma redução via mapeamento de  $A_{\text{MT}}$  para  $\text{HALT}_{\text{MT}}$ .

# Redutibilidade por mapeamento

- ▶ Precisamos apresentar uma função computável  $f$  que recebe como entrada  $\langle M, w \rangle$  e dá como saída  $\langle M', w' \rangle$ , tal que:

$$\langle M, w \rangle \in A_{\text{MT}} \text{ se, e somente se, } \langle M', w' \rangle \in \text{HALT}_{\text{MT}}$$

- ▶ Utilizaremos uma máquina  $F$  para computar tal função.

# Redutibilidade por mapeamento

---

**Algorithm 10:** Computando a função  $f$  através de uma máquina  $F$ 

---

**Input:**  $\langle M, w \rangle$

**Output:**  $\langle M', w \rangle$

1 Construa  $M'$  tal que, sobre uma entrada  $x$ ,  $M'$  roda  $x$  sobre  $M$  e: se  $M$  aceita  $x$ ,  $M'$  aceita. Se  $M$  rejeita  $x$ ,  $M'$  entra em loop. Deixe  $\langle M', w \rangle$  na fita

---

# Redutibilidade por mapeamento

## Exemplo

- ▶ A redução via mapeamento de  $E_{MT}$  para  $EQ_{MT}$  baseia-se no teorema provado sobre a indecidibilidade de  $EQ_{MT}$  visto anteriormente.
- ▶ Neste caso, a redução  $f$  está mapeando a entrada  $\langle M \rangle$  em  $\langle M, M_1 \rangle$ , em que  $M_1$  é uma MT que rejeita tudo.

# Redutibilidade por mapeamento

## Exemplo

- ▶ Na demonstração da indecidibilidade de  $E_{MT}$  nós utilizamos uma redução a partir de  $A_{MT}$ .
- ▶ Vamos verificar como podemos converter esta redução em uma redução via mapeamento.
- ▶ Da redução original, podemos construir uma função  $f$  que recebe como entrada  $\langle M, w \rangle$  e produz  $\langle M_1 \rangle$ , sendo  $M_1$  uma máquina que aceita apenas  $w$  se e somente se  $M$  aceita  $w$ .

# Redutibilidade por mapeamento

- ▶  $M_1$  é a MT descrita na demonstração.
- ▶ Mas  $M$  aceita  $w$  se, e somente se,  $L(M_1)$  não é vazia, então  $f$  é um redução via mapeamento de  $A_{MT}$  para  $\overline{E_{MT}}$ .
- ▶ Mas ainda preservarmos o resultado de que  $E_{MT}$  é indecidível, uma vez que decidibilidade não é afetado por complementação.
- ▶ Mas não é uma redução via mapeamento de  $A_{MT}$  para  $E_{MT}$ .

# Redutibilidade por mapeamento

- ▶ A sensibilidade da redutibilidade via mapeamento sobre a complementação é importante para provar a não-reconhecibilidade de certas linguagens.
- ▶ Também podemos utilizar a redução via mapeamento para mostrar que problemas sequer são Turing-reconhecíveis.

# Redutibilidade por mapeamento

## Teorema

*Se  $A \leq_m B$  e  $B$  é Turing-reconhecível,  $A$  também é.*

# Redutibilidade por mapeamento

- ▶ A prova deste teorema é muito parecida com a demonstração anterior de que:  
*Se  $A \leq_m B$  e  $B$  é Turing-decidível, então  $A$  também é.*
- ▶ Exercício.

# Redutibilidade por mapeamento

## Corolário

*Se  $A \leq_m B$  e  $A$  não é Turing-reconhecível,  $B$  também não é.*

# Redutibilidade por mapeamento

- ▶ Uma aplicação típica deste corolário é tomar  $A$  como  $\overline{A_{\text{MT}}}$ .
- ▶ Sabemos que  $\overline{A_{\text{MT}}}$  não é Turing-reconhecível.
- ▶ Pela definição de redução via mapeamento, temos que se  $A \leq_m B$  então  $\bar{A} \leq_m \bar{B}$ .
- ▶ Para provar então que  $B$  não é Turing-reconhecível, basta mostrar que  $A_{\text{MT}} \leq_m \bar{B}$ .

# Redutibilidade por mapeamento

- ▶ É possível também utilizar reduções via mapeamento para demonstrar que linguagens não são Turing-reconhecíveis e nem co-Turing-reconhecíveis.

# Redutibilidade por mapeamento

## Teorema

$\text{EQ}_{\text{MT}}$  não é Turing-reconhecível e nem co-Turing-reconhecível.

# Redutibilidade por mapeamento

## Demonstração

- ▶ Primeiramente vamos mostrar que  $\text{EQ}_{\text{MT}}$  não é Turing-reconhecível.
- ▶ Basta mostrar que  $A_{\text{MT}}$  é redutível a  $\overline{\text{EQ}_{\text{MT}}}$ .

# Redutibilidade por mapeamento

---

**Algorithm 11:** Construindo a função de redução  $A_{MT}$  para  $\overline{EQ_{MT}}$ 

**Input:**  $\langle M, w \rangle$

**Output:**  $\langle M_1, M_2 \rangle$

- 1 Construa duas máquinas  $M_1$  e  $M_2$  da seguinte maneira:
    - 2  $M_1$  sobre qualquer entrada diz rejeita
    - 3  $M_2$  sobre qualquer entrada roda  $M$  sobre  $w$  e se  $M$  aceita,  $M_2$  aceita
  - 4 Deixe  $\langle M_1, M_2 \rangle$  na fita.
-

# Redutibilidade por mapeamento

## Demonstração

- ▶  $M_1$  não aceita nada.
- ▶ Se  $M$  aceita  $w$ ,  $M_2$  aceita qualquer coisa, e portanto as duas máquinas não são equivalentes.
- ▶ Se  $M$  não aceita  $w$ ,  $M_2$  não aceita nada, e portanto são equivalentes.
- ▶ Assim  $f$  é uma redução de  $A_{\text{MT}}$  para  $\overline{\text{EQ}_{\text{MT}}}$ .
- ▶ Portanto  $\text{EQ}_{\text{MT}}$  não é Turing-reconhecível.

# Redutibilidade por mapeamento

## Demonstração

- ▶ Vamos mostrar agora que  $\overline{EQ_{MT}}$  não é Turing-reconhecível, isto é, que  $EQ_{MT}$  não é co-Turing-reconhecível.
- ▶ Precisamos fornecer uma redução de  $A_{MT}$  para  $EQ_{MT}$ .

# Redutibilidade por mapeamento

---

**Algorithm 12:** Construindo a função de redução  $A_{MT}$  para  $EQ_{MT}$ 

**Input:**  $\langle M, w \rangle$

**Output:**  $\langle M_1, M_2 \rangle$

- 1 Construa duas máquinas  $M_1$  e  $M_2$  da seguinte maneira:
    - 2  $M_1$  sobre qualquer entrada diz aceita
    - 3  $M_2$  sobre qualquer entrada roda  $M$  sobre  $w$  e se  $M$  aceita,  $M_2$  aceita
    - 4 Deixe  $\langle M_1, M_2 \rangle$  na fita.
-

# Redutibilidade por mapeamento

## Demonstração

- ▶ É a mesma construção da função anterior, mas agora  $M_1$  aceita tudo em vez de rejeitar.
- ▶ Assim,  $M$  aceita  $w$ , se e somente se,  $M_1$  e  $M_2$  são equivalentes.
- ▶ Logo  $\text{AMT} \leq_m \text{EQ}_{\text{MT}}$  e portanto,  $\text{EQ}_{\text{MT}}$  não é co-Turing-reconhecível.

# Redutibilidade por mapeamento

## Demonstração.

- ▶ Como  $\text{EQ}_{\text{MT}}$  não é Turing-reconhecível e nem co-Turing-reconhecível, finalizamos a prova.

