Teoria da Computação

A Tese de Church-Turing

Aula 10

Prof. Felipe A. Louza

UFU

Roteiro

Vamos ver algumas definições alternativas de máquinas de Turing, chamadas de **variantes de MTs**:

- MTs com mais de uma fita.
- MTs não-determinísticas.

Vamos ver que todas as variantes possuem o mesmo poder computacional:

- Decidem/Reconhecem o mesmo
- Chamamos esta invariância

Variantes de MTs podem ser úteis dependendo do problema estudado.

Introdução

Vamos ver algumas definições alternativas de máquinas de Turing, chamadas de **variantes de MTs**:

- MTs com mais de uma fita.
- MTs não-determinísticas.

Vamos ver que todas as variantes possuem o mesmo poder computacional:

- Decidem/Reconhecem a mesma **classe de linguagens**.
- Chamamos esta invariância de **robustez**.

Variantes de MTs podem ser úteis dependendo do problema estudado.

Introdução

Vamos ver algumas definições alternativas de máquinas de Turing, chamadas de **variantes de MTs**:

- MTs com mais de uma fita.
- MTs não-determinísticas.

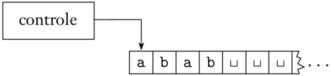
Vamos ver que todas as variantes possuem o mesmo poder computacional:

- Decidem/Reconhecem a mesma **classe de linguagens**.
- Chamamos esta invariância de **robustez**.

Variantes de MTs podem ser úteis dependendo do problema estudado.

Introdução

Por exemplo, no modelo de MT estudado (**MT padrão**), o cursor sempre vai ou para **E** ou para a **D**.



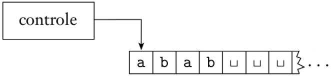
- Vamos propor uma variante que permite que o cursor *fique parado*.

$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{E, D, P\}$$

- Essa variante permite que as MTs reconheçam linguagens adicionais?
  - A resposta é **não**.

Introdução

Por exemplo, no modelo de MT estudado (**MT padrão**), o cursor sempre vai ou para **E** ou para a **D**.



- Vamos propor uma variante que permite que o cursor *fique parado*.

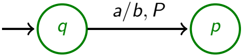
$$\delta : Q \times \Gamma \to Q \times \Gamma \times \{E, D, P\}$$

- Essa variante permite que as MTs reconheçam linguagens adicionais?
  - A resposta é **não**.

Introdução

Isso porque podemos *"converter"* qualquer MT com essa característica para uma **MT padrão**:

- Cada transição $\delta(q, a) = (p, b, P)$:

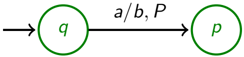


- É substituída por um par, com movimentos para a **D** e para **E**:

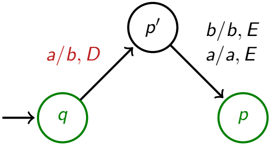
- Logo os dois modelos são **equivalentes**.
  - MT com movimento **P** $\rightarrow$ **MT padrão**; e
  - **MT padrão** $\rightarrow$ MT com movimento **P** (trivial).

Isso porque podemos *"converter"* qualquer MT com essa característica para uma **MT padrão**:

- Cada transição $\delta(q, a) = (p, b, P)$:



- É substituída por um par, com movimentos para a **D** e para **E**:

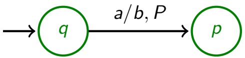


- Logo os dois modelos são **equivalentes**.
  - MT com movimento **P** → **MT padrão**; e
  - **MT padrão** → MT com movimento **P** (trivial).

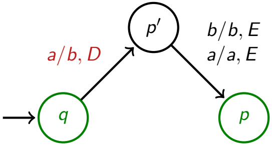
Introdução

Isso porque podemos *"converter"* qualquer MT com essa característica para uma **MT padrão**:

- Cada transição $\delta(q, a) = (p, b, P)$:



- É substituída por um par, com movimentos para a **D** e para **E**:



- Logo os dois modelos são **equivalentes**.
  - MT com movimento **P** $\rightarrow$ **MT padrão**; e
  - **MT padrão** $\rightarrow$ MT com movimento **P** (trivial).

Introdução

Esse pequeno exemplo mostra a chave para a equivalência entre as variantes de MTs:

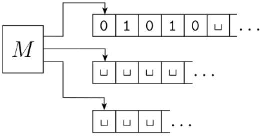
- Precisamos mostrar como simular um modelo pelo outro.

Roteiro

1. Variantes de Máquinas de Turing (MT)
   - MTs multifita
   - MTs não-determinísticas
   - MTs e outros modelos
2. A definição de Algoritmo
   - Os problemas de Hilbert
   - A tese de Church-Turing
   - Algoritmos e Máquinas de Turing
3. Referências

MTs multifita

Uma **MT multifita** é como uma **MT padrão** com k fitas.



- Cada fita tem o seu próprio cursor de leitura/escrita independente.
- Inicialmente, $w = w_1 w_2 \ldots w_n$ aparece na fita 1, e as outras fitas estão em branco.
- A função de transição é redefinida:

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{E, D, P\}^k$$

com isso:

$$\delta(q, a_1, a_2, \ldots, a_k) = (p, b_1, b_2, \ldots, b_k, \underbrace{E, D, \ldots, P}_{k})$$

MTs multifita

As MTs multifita aparentam ser mais poderosas do que as **MTs padrão**, mas na verdade não são.

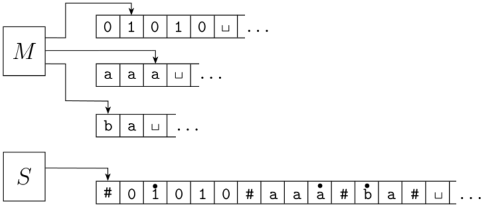
Teorema

Toda MT multifita tem uma máquina de Turing padrão que lhe é equivalente.

MTs multifita

Ideia da Prova:

- Mostramos como **converter** uma MT multifita em uma MT de fita única.
- Seja M uma MT multifita e S uma MT de fita única.

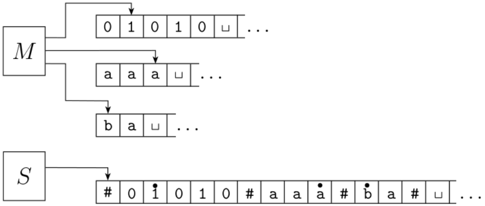


  - Vamos concatenar o conteúdo das k fitas na fita de S com o delimitador $\# \notin \Gamma$;
  - A posição de cada cursor é representada por um símbolo $\dot{w}$ (cursores virtuais).

MTs multifita

Ideia da Prova:

- Mostramos como **converter** uma MT multifita em uma MT de fita única.
- Seja M uma MT multifita e S uma MT de fita única.



  - Vamos concatenar o conteúdo das k fitas na fita de S com o delimitador $\# \notin \Gamma$:
  - A posição de cada cursor é representada por um símbolo $\dot{\mathbf{w}}$ (cursores virtuais).

MTs multifita

Ideia da Prova (continuação):

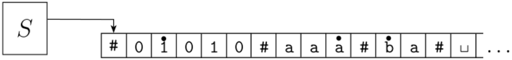
1. Primeiro colocamos w na fita de S na codificação proposta, isto é:

$$\#\dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$$

2. Para simular

$$\delta(q, a_1, a_2, \ldots, a_k) = (p, b_1, b_2, \ldots, b_k, \underbrace{E, D, \ldots, P}_{k})$$

   - A máquina S verifica cada cursor virtual para definir qual transição deve ser aplicada.



$$\delta(q, 1, a, b) = (p, 0, b, a, \underbrace{D, E, P}_{3})$$

MTs multifita

Ideia da Prova (continuação):

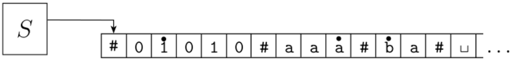
1. Primeiro colocamos w na fita de S na codificação proposta, isto é:

$$\#\dot{w}_1 w_2 \ldots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \ldots \#$$

2. Para simular

$$\delta(q, a_1, a_2, \ldots, a_k) = (p, b_1, b_2, \ldots, b_k, \underbrace{E, D, \ldots, P}_{k})$$

   – A máquina S verifica cada cursor virtual para definir qual transição deve ser aplicada.



$$\delta(q, 1, a, b) = (p, 0, b, a, \underbrace{D, E, P}_{3})$$

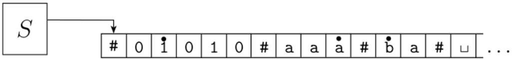
MTs multifita

Ideia da Prova (continuação):

3. Definida a transição

$$\delta(q, 1, a, b) = (p, 0, b, a, \underbrace{D, E, P}_{k})$$

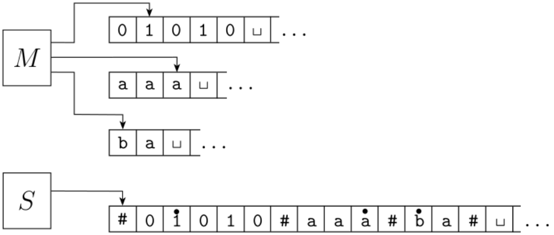
- A máquina S faz uma segunda leitura atualizando a fita.



MTs multifita

Ideia da Prova (continuação):

- Se em algum momento, algum cursor virtual vai parar sobre o i-ésimo $\#$:



- Na fita i de M ele está em uma posição $\sqcup$.
  - Então, na fita de S, deslocamos todas as células depois dessa posição um para a direita, e inserimos um $\dot{\sqcup}$.

MTs multifita

Prova:

- "Sobre a cadeia de entrada $w = w_1 w_2 \dots w_n$:

1. Primeiro S (**MT padrão**) põe sua fita no formato que representa todas as k fitas da MT M.

$$\#\dot{w}_1 w_2 \dots w_n \# \dot{\sqcup} \# \dot{\sqcup} \# \dots \#$$

2. Para simular um único movimento, S faz uma varredura na sua fita desde o primeiro $\#$, que marca a extremidade esquerda, até o $(k+1)$-ésimo $\#$, que marca a extremidade direita, de modo a determinar os símbolos sob os cursores virtuais. Então, S faz uma segunda passagem para atualizar as fitas conforme a função de transição de M estabelece.

3. Se em algum ponto S move um dos cursores virtuais sobre um $\#$, S desloca o conteúdo da fita, a partir dessa célula até o último $\#$, uma posição para a direita e escreve um símbolo em branco nessa célula da fita. Então ela continua a simulação tal qual anteriormente."

MTs multifita

Corolário

Uma linguagem é Turing-reconhecível se, e somente se, alguma máquina de Turing multifita a reconhece.

Roteiro

MTs não-determinísticas

Em uma MT não-determinística, uma computação pode proceder de várias maneiras.

- A função de transição tem a forma:

$$\delta : Q \times \Gamma \rightarrow 2^{(Q \times \Gamma \times \{L,R\})}$$

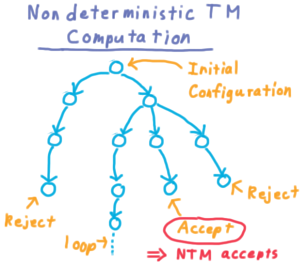
  ou seja,

$$\delta(q, a) = \{(p_i, b_i, D), (p_j, b_j, E), \ldots\}$$

- Dado um estado e um símbolo de lido:
  - Podemos ir para vários outros estados, escrever diferentes símbolos e mover para diferentes direções.

MTs não-determinísticas

A computação de uma MT não-determinística é uma árvore cujos ramos correspondem a diferentes computações possíveis.



- A MT não-determinística aceita uma cadeia ✓ se pelo menos um ramo leva ao estado de aceitação.

MTs não-determinísticas

As MTs não-determinísticas aparentam ser mais poderosas do que as **MTs padrão**, mas na verdade não são.

Teorema

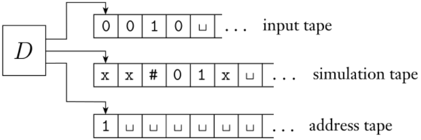
Toda MT não-determinística tem uma máquina de Turing padrão que lhe é equivalente.

O não-determinísmo não adiciona poder computacional às MTs.

MTs não-determinísticas

Ideia da Prova:

- Mostramos como **converter** uma MT não-determinística em uma MT multifita.
- Seja N uma MT não-determinística e D uma MT com 3-fitas.

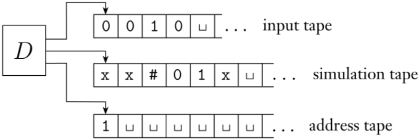


- A MT D simulará todas as possíveis computações não determinísticas.
  - Fita 1: contém a palavra w de entrada
  - Fita 2: é a fita de trabalho, mantém uma cópia da em uma computação de um ramo não-determinístico
  - Fita 3: guarda a posição de D na árvore de computação

MTs não-determinísticas

Ideia da Prova:

- Mostramos como **converter** uma MT não-determinística em uma MT multifita.
- Seja N uma MT não-determinística e D uma MT com 3-fitas.

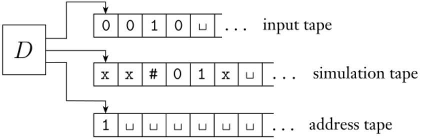


- A MT D simulará todas as possíveis computações não determinísticas.
  1. Fita 1: contém a palavra w de entrada $\leftarrow$ read-only.
  2. Fita 2: é a fita de trabalho, mantém uma cópia da fita de N em uma computação de um ramo não-determinístico.
  3. Fita 3: guarda a posição de D na árvore de computação.

MTs não-determinísticas

Ideia da Prova:

- Mostramos como **converter** uma MT não-determinística em uma MT multifita.
- Seja N uma MT não-determinística e D uma MT com 3-fitas.

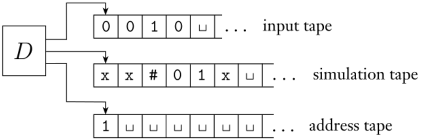


- A MT D simulará todas as possíveis computações não determinísticas.
  1. Fita 1: contém a palavra w de entrada $\leftarrow$ read-only.
  2. Fita 2: é a fita de trabalho, mantém uma cópia da fita de N em uma computação de um ramo não-determinístico.
  3. Fita 3: guarda a posição de D na árvore de computação.

MTs não-determinísticas

Ideia da Prova:

- Mostramos como **converter** uma MT não-determinística em uma MT multifita.
- Seja N uma MT não-determinística e D uma MT com 3-fitas.

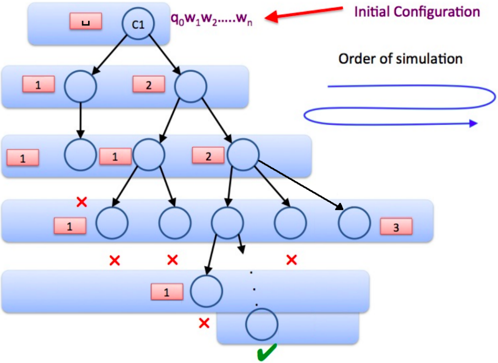


- A MT D simulará todas as possíveis computações não determinísticas.
  1. Fita 1: contém a palavra w de entrada $\leftarrow$ read-only.
  2. Fita 2: é a fita de trabalho, mantém uma cópia da fita de N em uma computação de um ramo não-determinístico.
  3. Fita 3: guarda a posição de D na árvore de computação.

MTs não-determinísticas

Como percorrer a árvore de computações?

- Primeiro, vamos enumerar cada nó da árvore: $\leftarrow$ nó 222?



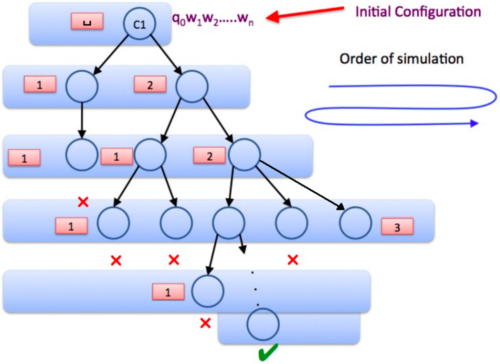
- A MT D simulará cada possível configuração da árvore com uma busca em largura $\leftarrow$ nó 223?

Porque busca em profundidade não funciona?

MTs não-determinísticas

Como percorrer a árvore de computações?

- Primeiro, vamos enumerar cada nó da árvore: $\leftarrow$ nó 222?



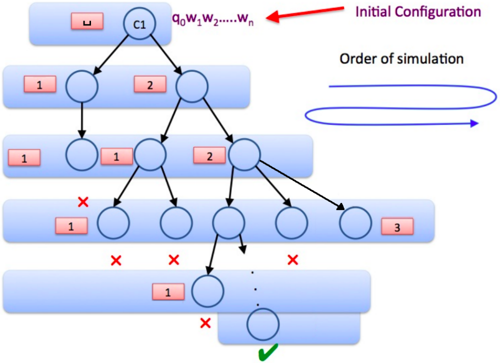
- A MT D simulará cada possível configuração da árvore com uma busca em largura $\leftarrow$ nó 223?

Porque busca em profundidade não funciona?

MTs não-determinísticas

Como percorrer a árvore de computações?

- Primeiro, vamos enumerar cada nó da árvore: $\leftarrow$ nó 222?



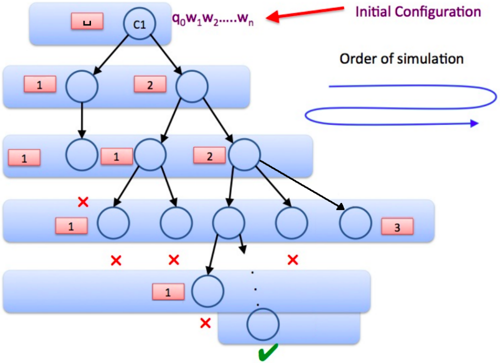
- A MT *D* simulará cada possível configuração da árvore com uma busca em largura $\leftarrow$ nó 223?

Porque busca em profundidade não funciona?

MTs não-determinísticas

Como percorrer a árvore de computações?

- Primeiro, vamos enumerar cada nó da árvore: ← nó 222?



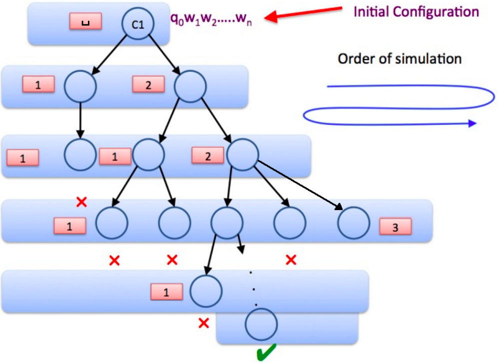
- A MT *D* simulará cada possível configuração da árvore com uma busca em largura ← nó 223?

Porque busca em profundidade não funciona?

MTs não-determinísticas

Como percorrer a árvore de computações?

- Primeiro, vamos enumerar cada nó da árvore: $\leftarrow$ nó 222?



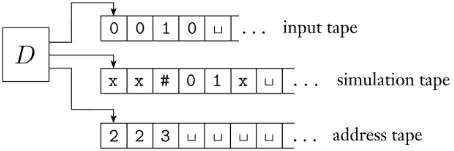
- A MT D simulará cada possível configuração da árvore com uma busca em largura $\leftarrow$ nó 223?

Porque busca em profundidade não funciona?

MTs não-determinísticas

Ideia da Prova (continuação):

- A 3ª fita indica qual o ramo de computação de N estamos simulando atualmente, e utiliza a 2ª fita (de trabalho).



- Para simular a busca em largura o endereço é incrementado:
  - Suponha que o número máximo de filhos na árvore é $b = 3$
  - A máquina considera as seguintes configurações:

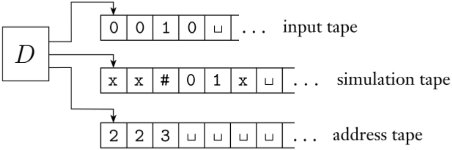
    $1, 2, 3, 11, 12, 13, 21, 22, 23, \ldots, 111, 112, 113, \ldots, 221, 222, 223, \ldots$

  (algumas podem ser inválidas)

MTs não-determinísticas

Ideia da Prova (continuação):

- A 3ª fita indica qual o ramo de computação de N estamos simulando atualmente, e utiliza a 2ª fita (de trabalho).



- Para simular a busca em largura o endereço é incrementado:
  1. Suponha que o número máximo de filhos na árvore é $b = 3$
  2. A máquina considera as seguintes configurações:

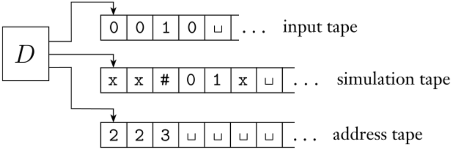
$$1, 2, 3, 11, 12, 13, 21, 22, 23, \ldots, 111, 112, 113, \ldots 221, 222, 223, 231 \ldots$$

(algumas podem ser inválidas)

MTs não-determinísticas

Ideia da Prova (continuação):

- A 3ª fita indica qual o ramo de computação de N estamos simulando atualmente, e utiliza a 2ª fita (de trabalho).



- Para simular a busca em largura o endereço é incrementado:
  1. Suponha que o número máximo de filhos na árvore é $b = 3$
  2. A máquina considera as seguintes configurações:

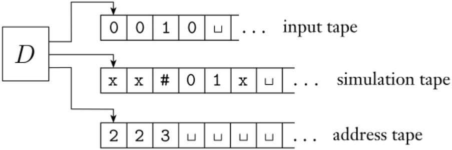
$$1, 2, 3, 11, 12, 13, 21, 22, 23, \ldots, 111, 112, 113, \ldots 221, 222, 223, 231 \ldots$$

(algumas podem ser inválidas)

MTs não-determinísticas

Ideia da Prova (continuação):

- A 3ª fita indica qual o ramo de computação de N estamos simulando atualmente, e utiliza a 2ª fita (de trabalho).



- Para simular a busca em largura o endereço é incrementado:
  1. Suponha que o número máximo de filhos na árvore é $b = 3$
  2. A máquina considera as seguintes configurações:

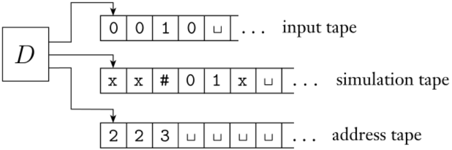
$$1, 2, 3, 11, 12, 13, 21, 22, 23, \ldots, 111, 112, 113, \ldots 221, 222, 223, 231 \ldots$$

(algumas podem ser inválidas)

MTs não-determinísticas

Ideia da Prova (continuação):

- A 3ª fita indica qual o ramo de computação de N estamos simulando atualmente, e utiliza a 2ª fita (de trabalho).



- Para simular a busca em largura o endereço é incrementado:
  1. Suponha que o número máximo de filhos na árvore é $b = 3$
  2. A máquina considera as seguintes configurações:

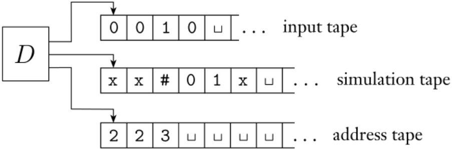
$$1, 2, 3, 11, 12, 13, 21, 22, 23, \ldots, 111, 112, 113, \ldots 221, 222, 223, 231 \ldots$$

(algumas podem ser inválidas)

MTs não-determinísticas

Ideia da Prova (continuação):

- A 3ª fita indica qual o ramo de computação de N estamos simulando atualmente, e utiliza a 2ª fita (de trabalho).



- Para simular a busca em largura o endereço é incrementado:
  1. Suponha que o número máximo de filhos na árvore é $b = 3$
  2. A máquina considera as seguintes configurações:

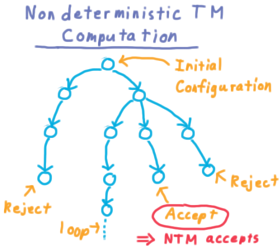
$$1, 2, 3, 11, 12, 13, 21, 22, 23, \ldots, 111, 112, 113, \ldots 221, 222, 223, 231 \ldots$$

(algumas podem ser inválidas)

MTs não-determinísticas

Ideia da Prova (continuação):

- Se em algum momento, um ramo alcança o estado q_{aceita}, então a MT D aceita w ✓.



- Caso contrário, D rejeita w e fica em *loop* (continua a simulação).

MTs não-determinísticas

Prova:

- "Sobre a cadeia de entrada $w = w_1 w_2 \dots w_n$:

1. Inicialmente, a fita 1 contém a entrada w e as fitas 2 e 3 estão vazias.
2. Copie a fita 1 para a fita 2.
3. Use a fita 2 para simular N com a entrada w sobre um ramo de sua computação não-determinística. Antes de cada passo de N, consulte o próximo símbolo na fita 3 para determinar qual escolha fazer entre aquelas permitidas pela função de transição de N. Se não restam mais símbolos na fita 3, ou se essa escolha não-determinística for inválida, aborte esse ramo indo para o passo (4). Também vá para o passo (4) se uma configuração de rejeição for encontrada. Se uma configuração de aceitação for encontrada, aceite a entrada ✓.
4. Substitua a cadeia na fita 3 pela próxima cadeia na ordem lexicográfica. Simule o próximo ramo da computação de N indo para o passo (2)."

MTs multifita

Corolário

Uma linguagem é Turing-reconhecível se, e somente se alguma máquina de Turing não-determinística a reconhece.

Roteiro

MTs e outros modelos

Vimos diferentes variantes das **MT-padrão**, todas equivalentes em poder.

- Muitos outros modelos para computação têm sido propostos, alguns similares a MTs, outros nem tanto.
- Todos compartilham a **característica essencial** das máquinas de Turing:

  acesso irrestrito a memória ilimitada

MTs e outros modelos

Todos os modelos com essas caracteríticas são equivalentes em poder[1].

- Similar ao que vemos entre linguagens de programação.
  - Todo programa que pode ser escrito em `Python`, pode ser escrito em `Java`, `C++`, ...

- Esta equivalência entre modelos de computação tem uma implicação profunda para o conceito de **"computação"**:

Muito embora possamos imaginar modelos de computação diferentes, a classe de algoritmos que eles descrevem permanece a mesma.

[1] Desde que eles não façam coisas absurdas, como executar uma sequência infinita de tarefas em um único passo.

MTs e outros modelos

Todos os modelos com essas caracteríticas são equivalentes em poder[1].

- Similar ao que vemos entre linguagens de programação.
  - Todo programa que pode ser escrito em `Python`, pode ser escrito em `Java`, `C++`, ...

- Esta equivalência entre modelos de computação tem uma implicação profunda para o conceito de **"computação"**:

Muito embora possamos imaginar modelos de computação diferentes, a classe de algoritmos que eles descrevem permanece a mesma.

[1] Desde que eles não façam coisas absurdas, como executar uma sequência infinita de tarefas em um único passo.

MTs e outros modelos

Todos os modelos com essas caracteríticas são equivalentes em poder[1].

- Similar ao que vemos entre linguagens de programação.
  - Todo programa que pode ser escrito em `Python`, pode ser escrito em `Java`, `C++`, `...`.

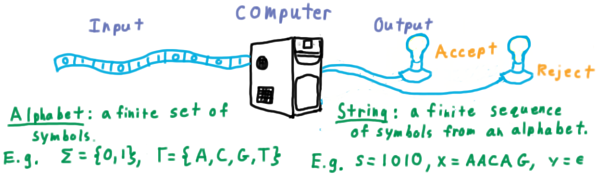
- Esta equivalência entre modelos de computação tem uma implicação profunda para o conceito de **"computação"**:

Muito embora possamos imaginar modelos de computação diferentes, a classe de algoritmos que eles descrevem permanece a mesma.

[1] Desde que eles não façam coisas absurdas, como executar uma sequência infinita de tarefas em um único passo.

MTs e computadores

Podemos comparar **MTs** com os computadores que usamos no dia a dia.

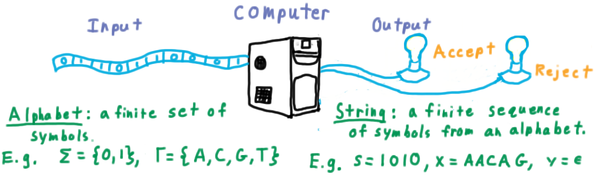


- Embora esses modelos pareçam bem diferentes, eles aceitam a mesma classe de linguagens.
  - Não é difícil ver que um computador pode simular uma **MT**[2].
  - Mais interessante é mostrar que uma **MT** consegue fazer tudo que um computador faz.

[2] Considerando que a memória do computador é suficientemente grande para todos os problemas (mas não infinita).

MTs e computadores

Podemos comparar **MTs** com os computadores que usamos no dia a dia.

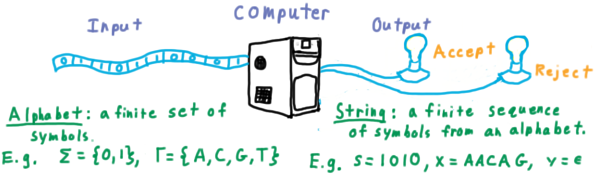


- Embora esses modelos pareçam bem diferentes, eles aceitam a mesma classe de linguagens.
  - Não é difícil ver que um computador pode simular uma **MT**[2].
  - Mais interessante é mostrar que uma **MT** consegue fazer tudo que um computador faz.

[2] Considerando que a memória do computador é suficientemente grande para todos os problemas (mas não infinita).

MTs e computadores

Podemos comparar **MTs** com os computadores que usamos no dia a dia.



- Embora esses modelos pareçam bem diferentes, eles aceitam a mesma classe de linguagens.
  - Não é difícil ver que um computador pode simular uma **MT**[2].
  - Mais interessante é mostrar que uma **MT** consegue fazer tudo que um computador faz.

[2] Considerando que a memória do computador é suficientemente grande para todos os problemas (mas não infinita).

MTs e computadores

Para isso, vamos definir um modelo de computador:

1. **Memória:**
   - Sequência indefinidamente longa de **palavras**, cada qual com um **endereço**.
2. **Programa:**
   - Está armazenado em algumas das palavras da memória;
   - Cada palavra representa uma instrução simples:
     - `read`, `write`, `load`, `store`, ...
   - Cada instrução envolve um número limitado (finito) de palavras
   - Cada instrução altera o valor de no máximo uma palavra
3. **Contador de instruções** (*program counter*):
   - Registrador especial que indica a próxima instrução a ser executada.

Assemelha-se ao paradigma básico de CPU/registrador/memória por trás dos computadores modernos.

MTs e computadores

Para isso, vamos definir um modelo de computador:

1. **Memória:**
   - Sequência indefinidamente longa de **palavras**, cada qual com um **endereço**.
2. **Programa:**
   - Está armazenado em algumas das palavras da memória;
   - Cada palavra representa uma instrução simples:
     - `read`, `write`, `load`, `store`, ...
   - Cada instrução envolve um número limitado (finito) de palavras
   - Cada instrução altera o valor de no máximo uma palavra
3. **Contador de instruções** (*program counter*):
   - Registrador especial que indica a próxima instrução a ser executada.

Assemelha-se ao paradigma básico de CPU/registrador/memória por trás dos computadores modernos.

MTs e computadores

Para isso, vamos definir um modelo de computador:

1. **Memória:**
   - Sequência indefinidamente longa de **palavras**, cada qual com um **endereço**.
2. **Programa:**
   - Está armazenado em algumas das palavras da memória;
   - Cada palavra representa uma instrução simples:
     - `read, write, load, store, ...`
   - Cada instrução envolve um número limitado (finito) de palavras
   - Cada instrução altera o valor de no máximo uma palavra
3. **Contador de instruções** (*program counter*):
   - Registrador especial que indica a próxima instrução a ser executada.

Assemelha-se ao paradigma básico de CPU/registrador/memória por trás dos computadores modernos.

MTs e computadores

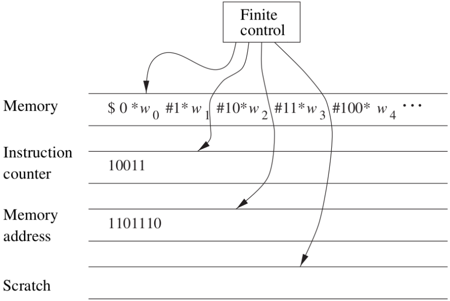
Para isso, vamos definir um modelo de computador:

1. **Memória:**
   - Sequência indefinidamente longa de **palavras**, cada qual com um **endereço**.
2. **Programa:**
   - Está armazenado em algumas das palavras da memória;
   - Cada palavra representa uma instrução simples:
     - `read`, `write`, `load`, `store`, ...
   - Cada instrução envolve um número limitado (finito) de palavras
   - Cada instrução altera o valor de no máximo uma palavra
3. **Contador de instruções** (*program counter*):
   - Registrador especial que indica a próxima instrução a ser executada.

Assemelha-se ao paradigma básico de CPU/registrador/memória por trás dos computadores modernos.

MTs e computadores

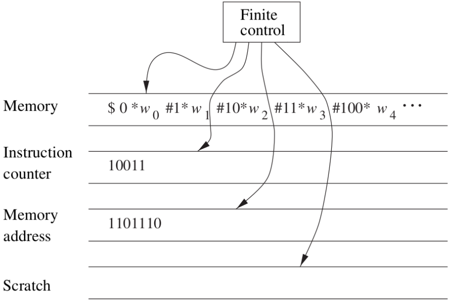
Utilizaremos uma **MT com 3-fitas** para simular o modelo proposto:



- Ciclo de instruções do computador:
  1. A MT procura na fita 1 o local apontado pelo conteúdo da fita 2.
  2. Interpreta o conteúdo da instrução.
  3. Se a instrução usa o valor de um endereço, copiamos para a fita 3.
  4. A MT então executa a instrução usando o rascunho.
  5. A MT incrementa o conteúdo da fita 2 e **recomeça o ciclo**.

MTs e computadores

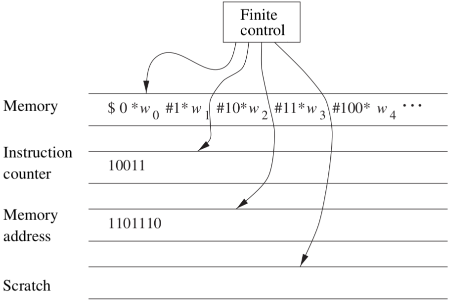
Utilizaremos uma **MT com 3-fitas** para simular o modelo proposto:



- Ciclo de instruções do computador:
  1. A MT procura na fita 1 o local apontado pelo conteúdo da fita 2.
  2. Interpreta o conteúdo da instrução.
  3. Se a instrução usa o valor de um endereço, copiamos para a fita 3.
  4. A MT então executa a instrução usando o rascunho.
  5. A MT incrementa o conteúdo da fita 2 e **recomeça o ciclo**.

MTs e computadores

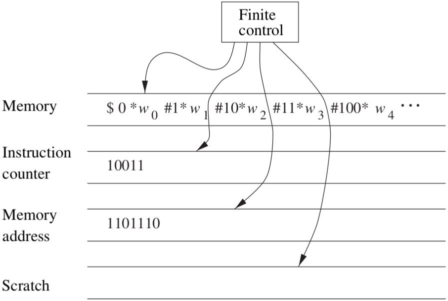
Utilizaremos uma **MT com 3-fitas** para simular o modelo proposto:



- Ciclo de instruções do computador:
  1. A MT procura na fita 1 o local apontado pelo conteúdo da fita 2.
  2. Interpreta o conteúdo da instrução.
  3. Se a instrução usa o valor de um endereço, copiamos para a fita 3.
  4. A MT então executa a instrução usando o rascunho.
  5. A MT incrementa o conteúdo da fita 2 e **recomeça o ciclo**.

MTs e computadores

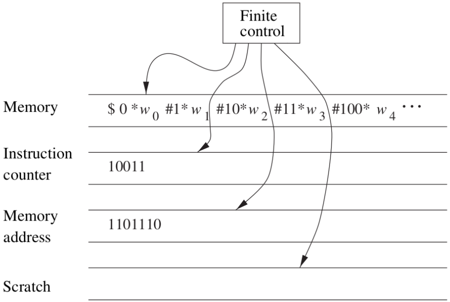
Utilizaremos uma **MT** com 3-fitas para simular o modelo proposto:



- Ciclo de instruções do computador:
  1. A MT procura na fita 1 o local apontado pelo conteúdo da fita 2.
  2. Interpreta o conteúdo da instrução.
  3. Se a instrução usa o valor de um endereço, copiamos para a fita 3.
  4. A MT então executa a instrução usando o rascunho.
  5. A MT incrementa o conteúdo da fita 2 e **recomeça o ciclo**.

MTs e computadores

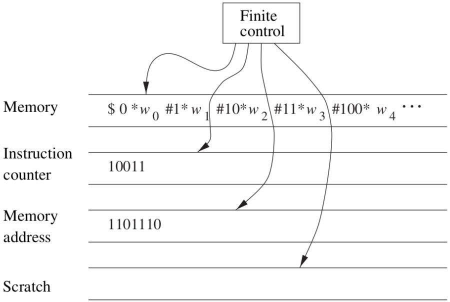
Utilizaremos uma **MT** com 3-fitas para simular o modelo proposto:



- Ciclo de instruções do computador:
  1. A MT procura na fita 1 o local apontado pelo conteúdo da fita 2.
  2. Interpreta o conteúdo da instrução.
  3. Se a instrução usa o valor de um endereço, copiamos para a fita 3.
  4. A MT então executa a instrução usando o rascunho.
  5. A MT incrementa o conteúdo da fita 2 e **recomeça o ciclo**.

MTs e computadores

Utilizaremos uma **MT** com 3-fitas para simular o modelo proposto:



- Ciclo de instruções do computador:
  1. A MT procura na fita 1 o local apontado pelo conteúdo da fita 2.
  2. Interpreta o conteúdo da instrução.
  3. Se a instrução usa o valor de um endereço, copiamos para a fita 3.
  4. A MT então executa a instrução usando o rascunho.
  5. A MT incrementa o conteúdo da fita 2 e **recomeça o ciclo**.

MTs e computadores

Embora a discussão anterior esteja longe de ser uma prova completa e formal, ela deve dar uma idéia de que podemos simular um computador típico em uma **MT**.

- Vamos ver melhor essa relação entre MTs e tudo o que pode ser computável . . . .

Roteiro

Computação efetiva

Ao longo do curso temos capturado o conceito de *"computação efetiva"*:

- Apresentamos modelos de computação cada vez mais poderosos: **AFs**, **APs**, **MTs**.
- Cada modelo é capaz de executar "algoritmos" mais gerais que o modelo anterior.

Mas, o que é um **algoritmo**?

- Informalmente: uma sequência finita de instruções simples para realizar uma tarefa
- Essa idéia de "procedimento" tem sido utilizada há muito tempo na Matemática[3]

[3] Ex.: Algoritmo de Euclides para encontrar o máximo divisor comum (300 a.C.).

Computação efetiva

Ao longo do curso temos capturado o conceito de *"computação efetiva"*:

- Apresentamos modelos de computação cada vez mais poderosos: **AFs**, **APs**, **MTs**.
- Cada modelo é capaz de executar "algoritmos" mais gerais que o modelo anterior.

Mas, o que é um **algoritmo**?

- Informalmente: uma sequência finita de instruções simples para realizar uma tarefa.
- Essa idéia de "procedimento" tem sido utilizada há muito tempo na Matemática[3]

[3] Ex.: Algoritmo de Euclides para encontrar o máximo divisor comum (300 a.C.).

Computação efetiva

Ao longo do curso temos capturado o conceito de *"computação efetiva"*:

- Apresentamos modelos de computação cada vez mais poderosos: **AFs**, **APs**, **MTs**.
- Cada modelo é capaz de executar "algoritmos" mais gerais que o modelo anterior.

Mas, o que é um **algoritmo**?

- Informalmente: uma sequência finita de instruções simples para realizar uma tarefa.
- Essa idéia de "procedimento" tem sido utilizada há muito tempo na Matemática[3]

[3] Ex.: Algoritmo de Euclides para encontrar o máximo divisor comum (300 a.C.).

A definição de algoritmo

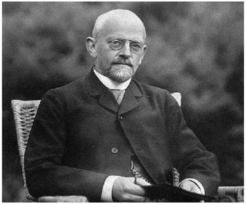
Mas qual a definição formal de algoritmo?

- Precisamos deste conceito para podermos saber os limites da computação.
- Essa definição surgiu apenas no **século XX**.

Roteiro

Um pouco de história

Em 1900, David Hilbert preferiu a sua agora-famosa palestra no *"Congresso Internacional de Matemáticos"* em Paris, com 23 desafios para o **século XX**.



- 10º problema:

  escrever "um processo pelo qual possa ser determinado, com um número finito de operações" se um polinômio tem, ou não, raízes inteiras.

Relembrando...

Um polinômio é uma soma de **termos**, em que cada termo é um produto de certas variáveis com uma constante, chamada de **coeficiente**.

- Exemplo de polinômio:

$$P = \underbrace{6x^3yz^2}_{termo} + 3xy^2 - x^3 - 10$$

- Uma raiz de P é uma valoração das variáveis que torne $P = 0$.
- Para o exemplo acima temos as raízes:

$$x = 5,\ y = 3 \text{ e } z = 0.$$

- Sabemos que alguns polinômios possuem raízes inteiras, outros não.

Vamos considerar apenas polinômios com coeficientes inteiros.

Relembrando...

Um polinômio é uma soma de **termos**, em que cada termo é um produto de certas variáveis com uma constante, chamada de **coeficiente**.

- Exemplo de polinômio:

$$P = \underbrace{6x^3yz^2}_{termo} + 3xy^2 - x^3 - 10$$

- Uma raiz de P é uma valoração das variáveis que torne $P = 0$.
- Para o exemplo acima temos as raízes:

$$x = 5,\ y = 3 \text{ e } z = 0.$$

- Sabemos que alguns polinômios possuem raízes inteiras, outros não.

Vamos considerar apenas polinômios com coeficientes inteiros.

Relembrando. . .

Um polinômio é uma soma de **termos**, em que cada termo é um produto de certas variáveis com uma constante, chamada de **coeficiente**.

- Exemplo de polinômio:

$$P = \underbrace{6x^3yz^2}_{termo} + 3xy^2 - x^3 - 10$$

- Uma raiz de P é uma valoração das variáveis que torne $P = 0$.
- Para o exemplo acima temos as raízes:

$$x = 5,\ y = 3 \text{ e } z = 0.$$

- Sabemos que alguns polinômios possuem raízes inteiras, outros não.

Vamos considerar apenas polinômios com coeficientes inteiros.

O 10º problema de Hilbert

Voltando ao 10º problema de Hilbert:

> *escrever "um processo pelo qual possa ser determinado, com um número finito de operações" se um polinômio tem, ou não, raízes inteiras.*
>
> $\longrightarrow$ Hilbert estava interessado em um **algoritmo**.

- Da maneira com que Hilbert definiu o problema, deu a entender que acreditava que tal algoritmo existia, alguém **só precisava** encontra-lo.

O 10º problema de Hilbert

Voltando ao 10º problema de Hilbert:

> *escrever "um processo pelo qual possa ser determinado, com um número finito de operações" se um polinômio tem, ou não, raízes inteiras.*
>
> $\longrightarrow$ Hilbert estava interessado em um **algoritmo**.

- Da maneira com que Hilbert definiu o problema, deu a entender que acreditava que tal algoritmo existia, alguém **só precisava** encontra-lo.

O 10º problema de Hilbert

Hoje sabemos que nenhum algoritmo existe para tal tarefa.

- Essa é uma tarefa computacionalmente **insolúvel**.

Mas para provar esse fato, era preciso uma definição formal do que era um **algoritmo**.

- O 10º problema de Hilbert teve que esperar um tempo...

O termo algoritmo teria sido cunhado por Ada Lovalace (1833) muito tempo antes em homenagem ao matemático Al-Khwarizmi (820 d.C.).

O 10º problema de Hilbert

Hoje sabemos que nenhum algoritmo existe para tal tarefa.

- Essa é uma tarefa computacionalmente **insolúvel**.

Mas para provar esse fato, era preciso uma definição formal do que era um **algoritmo**.

- O 10º problema de Hilbert teve que esperar um tempo...

O termo algoritmo teria sido cunhado por Ada Lovalace (1833) muito tempo antes em homenagem ao matemático Al-khwarizmi (820 d.C.).

O 10º problema de Hilbert

Hoje sabemos que nenhum algoritmo existe para tal tarefa.

- Essa é uma tarefa computacionalmente **insolúvel**.

Mas para provar esse fato, era preciso uma definição formal do que era um **algoritmo**.

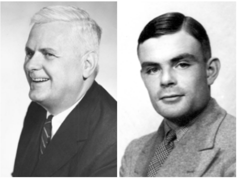
- O 10º problema de Hilbert teve que esperar um tempo...

O termo algoritmo teria sido cunhado por Ada Lovalace (1833) muito tempo antes em homenagem ao matemático Al-khwarizmi (820 d.C.).

Roteiro

A tese de Church-Turing

A definição formal de algoritmo veio apenas em 1936 com os trabalhos de **Alonzo Church** e **Alan Turing**.



- Church propôs um sistema notacional chamado de **cálculo-λ**.
- Turing, por sua vez, formalizou o conceito com **suas máquinas**.

MTs foram propostas com o nome de **a-machines**, *“automatic machines”*, Church quem as renomeou mais tarde.

A tese de Church-Turing

Estas duas definições foram demonstradas equivalentes.

- Tudo que um formalismo fazia, o outro também era capaz de fazer.

Essa conexão entre a noção informal e a definição precisa de algoritmo veio ser a chamada de **tese de Church-Turing**.

Noção intuitiva de algoritmos	é igual a	*algoritmos de máquina de Turing*

"Se uma função é efetivamente computável, então ela é computável por meio de uma ***máquina de Turing****."*

A tese de Church-Turing

A tese de Church-Turing provê a definição de algoritmo necessária para resolver o 10º problema de Hilbert.

- Vamos reformular o problema em **termos de linguagens**:

$$D = \{p \mid p \textit{ é um polinômio com raiz inteira}\}$$

- Queremos saber se D é **Turing-decidível**?

Ou seja, podemos escrever um **MT** que decide (sempre pára), e aceita $w \in D$ ✓ ou rejeita $w \notin D$ ✗.

A tese de Church-Turing

A tese de Church-Turing provê a definição de algoritmo necessária para resolver o 10^{o} problema de Hilbert.

- Vamos reformular o problema em **termos de linguagens**:

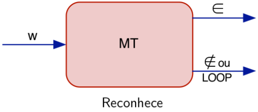
$$D = \{p \mid p \textit{ é um polinômio com raiz inteira}\}$$

- Queremos saber se D é **Turing-decidível**?

Ou seja, podemos escrever um **MT** que decide (sempre pára), e aceita $w \in D$ ✓ ou rejeita $w \notin D$ ✗.

A tese de Church-Turing

Primeiro, podemos mostrar que a linguagem D é **Turing-reconhecível**:



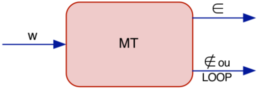
- Podemos construir uma **MT** M que reconhece D:
  - "Sobre a cadeia de entrada p:
    1. Calcule o valor de p com cada variável assumindo valores $0, -1, 1, -2, 2, -3, 3, \ldots$ Se em algum ponto o valor de $\mathbf{p} = \mathbf{0}$, aceite a cadeia ✓"
- A máquina não para quando $w \notin D \leftarrow$ Não decide

---

Exemplo, $p = 6x^3yz^2 + 3xy^2 - x^3 - 10 = 0$ quando $x = ?$, $y = ?$ e $z = ?$.

A tese de Church-Turing

Primeiro, podemos mostrar que a linguagem D é **Turing-reconhecível**:



Reconhece

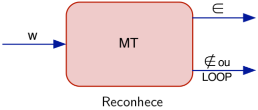
- Podemos construir uma **MT** M que reconhece D:
  - "Sobre a cadeia de entrada p:
    1. Calcule o valor de p com cada variável assumindo valores $0, -1, 1, -2, 2, -3, 3, \ldots$ Se em algum ponto o valor de $\mathbf{p} = \mathbf{0}$, aceite a cadeia ✓"
- A máquina não para quando $w \notin D \leftarrow$ Não decide

---

Exemplo, $p = 6x^3yz^2 + 3xy^2 - x^3 - 10 = 0$ quando $x = 5$, $y = 3$ e $z = 0$.

A tese de Church-Turing

Primeiro, podemos mostrar que a linguagem D é **Turing-reconhecível**:



- Podemos construir uma **MT** M que reconhece D:
  - "Sobre a cadeia de entrada p:
    1. Calcule o valor de p com cada variável assumindo valores $0, -1, 1, -2, 2, -3, 3, \ldots$ Se em algum ponto o valor de $\mathbf{p} = \mathbf{0}$, aceite a cadeia ✓"
- A máquina não para quando $w \notin D \leftarrow$ Não decide

Exemplo, $p = 6x^3yz^2 + 3xy^2 - x^3 - 10 = 0$ quando $x = 5$, $y = 3$ e $z = 0$.

Mas será que existe alguma outra **MT** M' que decide D?

- **Não**. Provado em 1970 por Matijasevič.

Portanto, D é **Turing-reconhecível** mas não é **Turing-decidível**.

- **Não** existe **um algoritmo** para o problema.

Para o caso em que p possuí apenas 1 variável é possível definir um "critério de parada", então a **MT** decide D'.

A tese de Church-Turing

Mas será que existe alguma outra **MT** M' que decide D?

- **Não**. Provado em 1970 por Matijasevič.

Portanto, D é **Turing-reconhecível** mas não é **Turing-decidível**.

- **Não** existe **um algoritmo** para o problema.

Para o caso em que p possuí apenas 1 variável é possível definir um "critério de parada", então a **MT** decide D'.

Mas será que existe alguma outra **MT** M' que decide D?

- **Não**. Provado em 1970 por Matijasevič.

Portanto, D é **Turing-reconhecível** mas não é **Turing-decidível**.

- **Não** existe **um algoritmo** para o problema.

Para o caso em que p possuí apenas 1 variável é possível definir um "critério de parada", então a **MT** decide D'.

A tese de Church-Turing

Mas será que existe alguma outra **MT** M' que decide D?

- **Não**. Provado em 1970 por Matijasevič.

Portanto, D é **Turing-reconhecível** mas não é **Turing-decidível**.

- **Não** existe **um algoritmo** para o problema.

Para o caso em que p possuí apenas 1 variável é possível definir um "critério de parada", então a **MT** decide D'.

A tese de Church-Turing

Tese de Church-Turing: define que tudo que é computável, deve ser computável por **Máquinas de Turing**.

- Não pode ser provada, ela é uma hipótese (ou definição), na verdade.
- No entanto, é universalmente aceita.
  - Nenhuma modificação já proposta para MTs aumenta seu poder.
  - Formalismos alternativos como λ-cálculo, sistemas de Post, funções μ-recursivas, e outros são todos **Turing-equivalentes**.

Note, entretanto, que para mostrar que a tese é falsa: basta achar uma máquina que compute uma função que uma **MT** não possa computar.

A tese de Church-Turing

Tese de Church-Turing: define que tudo que é computável, deve ser computável por **Máquinas de Turing**.

- Não pode ser provada, ela é uma hipótese (ou definição), na verdade.
- No entanto, é universalmente aceita.
  - Nenhuma modificação já proposta para MTs aumenta seu poder.
  - Formalismos alternativos como λ-cálculo, sistemas de Post, funções μ-recursivas, e outros são todos **Turing-equivalentes**.

Note, entretanto, que para mostrar que a tese é falsa: basta achar uma máquina que compute uma função que uma **MT** não possa computar.

A tese de Church-Turing

Tese de Church-Turing: define que tudo que é computável, deve ser computável por **Máquinas de Turing**.

- Não pode ser provada, ela é uma hipótese (ou definição), na verdade.
- No entanto, é universalmente aceita.
  - Nenhuma modificação já proposta para MTs aumenta seu poder.
  - Formalismos alternativos como λ-cálculo, sistemas de Post, funções μ-recursivas, e outros são todos **Turing-equivalentes**.

Note, entretanto, que para mostrar que a tese é falsa: basta achar uma máquina que compute uma função que uma **MT** não possa computar.

Roteiro

Algoritmos e Máquinas de Turing

Vamos mudar de foco no curso para algoritmos e problemas:

- Ao aceitar a tese de Church-Turing, assumimos que a definição de algoritmo é equivalente a algoritmos de **máquinas de Turing**.
- As linguagens representarão os problemas reais, e
- As **MTs** serão utilizadas como modelo de computação para executar algoritmos.

Fizemos isso com o 10º problema de Hilbert.

Algoritmos e Máquinas de Turing

Podemos descrever algoritmos para **MTs** em 3 níveis.

1. Descrição formal: dando os estados e as transições de uma MT.
2. Descrição da implementação: descrevemos como a máquina opera em termos de movimento de fita.
3. Descrição alto nível: pseudocódigo em linguagem natural para descrever o algoritmo, omitindo detalhes de implementação.
   - Em geral, utilizaremos esse nível de descrição.

Vimos exemplos em todos os níveis.

Algoritmos e Máquinas de Turing

Para apresentar algoritmos em alto nível para **MTs**, temos que a entrada deve ser sempre uma cadeia[4].

- Se desejamos fornecer um objeto O para uma **MT** que não é uma cadeia, primeiro é preciso codificá-lo em uma cadeia $\langle O \rangle$.
  - Exemplos: polinômios, grafos, gramáticas, autômatos.
  - Vamos assumir que as MTs sempre conseguem O em $\langle O \rangle$
- Se a entrada é um conjunto $O_1, O_2, \ldots, O_k$, codificamos todos em uma única cadeia $\langle O_1, O_2, \ldots, O_k \rangle$

[4] O mesmo acontece em um computador real (dados são armazenados como sequências binárias).

Algoritmos e Máquinas de Turing

Para apresentar algoritmos em alto nível para **MTs**, temos que a entrada deve ser sempre uma cadeia[4].

- Se desejamos fornecer um objeto O para uma **MT** que não é uma cadeia, primeiro é preciso codificá-lo em uma cadeia $\langle O \rangle$.
  - Exemplos: polinômios, grafos, gramáticas, autômatos, ...
  - Vamos assumir que as **MTs** sempre conseguem codificar O em $\langle O \rangle$.

- Se a entrada é um conjunto $O_1, O_2, \ldots, O_k$, codificamos todos em uma única cadeia $\langle O_1, O_2, \ldots, O_k \rangle$

[4] O mesmo acontece em um computador real (dados são armazenados como sequências binárias).

Algoritmos e Máquinas de Turing

Para apresentar algoritmos em alto nível para **MTs**, temos que a entrada deve ser sempre uma cadeia[4].

- Se desejamos fornecer um objeto O para uma **MT** que não é uma cadeia, primeiro é preciso codificá-lo em uma cadeia $\langle O \rangle$.
  - Exemplos: polinômios, grafos, gramáticas, autômatos, ...
  - Vamos assumir que as **MTs** sempre conseguem codificar O em $\langle O \rangle$.
- Se a entrada é um conjunto $O_1, O_2, \ldots, O_k$, codificamos todos em uma única cadeia $\langle O_1, O_2, \ldots, O_k \rangle$

[4] O mesmo acontece em um computador real (dados são armazenados como sequências binárias).

Algoritmos e Máquinas de Turing

Para apresentar algoritmos em alto nível para **MTs**, temos que a entrada deve ser sempre uma cadeia[4].

- Se desejamos fornecer um objeto O para uma **MT** que não é uma cadeia, primeiro é preciso codificá-lo em uma cadeia $\langle O \rangle$.
  - Exemplos: polinômios, grafos, gramáticas, autômatos, ...
  - Vamos assumir que as **MTs** sempre conseguem codificar O em $\langle O \rangle$.
- Se a entrada é um conjunto $O_1, O_2, \ldots, O_k$, codificamos todos em uma única cadeia $\langle O_1, O_2, \ldots, O_k \rangle$

[4] O mesmo acontece em um computador real (dados são armazenados como sequências binárias).

Algoritmos e Máquinas de Turing

Para apresentar algoritmos em alto nível para **MTs**, temos que a entrada deve ser sempre uma cadeia[4].

- Se desejamos fornecer um objeto O para uma **MT** que não é uma cadeia, primeiro é preciso codificá-lo em uma cadeia $\langle O \rangle$.
  - Exemplos: polinômios, grafos, gramáticas, autômatos, ...
  - Vamos assumir que as **MTs** sempre conseguem codificar O em $\langle O \rangle$.
- Se a entrada é um conjunto $O_1, O_2, \ldots, O_k$, codificamos todos em uma única cadeia $\langle O_1, O_2, \ldots, O_k \rangle$

[4] O mesmo acontece em um computador real (dados são armazenados como sequências binárias).

Exemplo

Seja A a linguagem de todas as *cadeias* representando grafos não-direcionados conexos.

$$A = \{\langle G \rangle \,|\, G \text{ é um grafo conexo } \}$$

- A linguagem A é **Turing-decidível**? Isto é, existe uma **MT** que decide A?
- Como seria a descrição de um alto nível dessa **MT**? Ou seja, qual seria o algoritmo?

Um grafo é conexo se todo vértice pode ser atingido a partir de qualquer outro vértice seguindo as arestas do grafo.

Exemplo

Seja A a linguagem de todas as *cadeias* representando grafos não-direcionados conexos.

$$A = \{\langle G \rangle \,|\, G \text{ é um grafo conexo } \}$$

- A linguagem A é **Turing-decidível**? Isto é, existe uma **MT** que decide A?
- Como seria a descrição de um alto nível dessa **MT**? Ou seja, qual seria o algoritmo?

Um grafo é conexo se todo vértice pode ser atingido a partir de qualquer outro vértice seguindo as arestas do grafo.

Exemplo

Seja A a linguagem de todas as *cadeias* representando grafos não-direcionados conexos.

$$A = \{\langle G \rangle \,|\, G \textit{ é um grafo conexo } \}$$

- A linguagem A é **Turing-decidível**? Isto é, existe uma **MT** que decide A?
- Como seria a descrição de um alto nível dessa **MT**? Ou seja, qual seria o algoritmo?

Um grafo é conexo se todo vértice pode ser atingido a partir de qualquer outro vértice seguindo as arestas do grafo.

Exemplo

Seja A a linguagem de todas as *cadeias* representando grafos não-direcionados conexos.

$$A = \{\langle G \rangle \,|\, G \text{ é um grafo conexo } \}$$

- A linguagem A é **Turing-decidível**? Isto é, existe uma **MT** que decide A?
- Como seria a descrição de um alto nível dessa **MT**? Ou seja, qual seria o algoritmo?

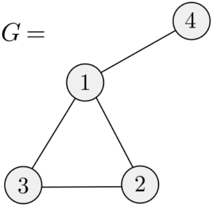
Um grafo é conexo se todo vértice pode ser atingido a partir de qualquer outro vértice seguindo as arestas do grafo.

Exemplo

Primeiro, como *codificar* G em $\langle G \rangle$?

- Um grafo é um par ordenado $G = (V, A)$, onde V é o conjunto de vértices e A o conjunto de arestas.
- Podemos codificar G como uma cadeia de símbolos:

$G =$



$\langle G \rangle =$

`(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`

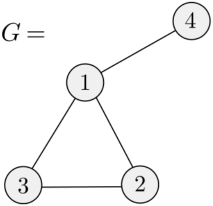
Nos próximos exemplos, assumimos que a **MT** codifica $O \to \langle O \rangle$.

Exemplo

Primeiro, como *codificar* G em $\langle G \rangle$?

- Um grafo é um par ordenado $G = (V, A)$, onde V é o conjunto de vértices e A o conjunto de arestas.
- Podemos codificar G como uma cadeia de símbolos:

$G =$



$\langle G \rangle =$

`(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`

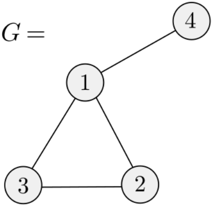
Nos próximos exemplos, assumimos que a **MT** codifica $O \to \langle O \rangle$.

Exemplo

A seguinte **MT** decide A:

- "Sobre a cadeia de entrada $\langle G \rangle$:
  1. Selecione o primeiro vértice de G e marque-o.
  2. Repita o passo seguinte até que nenhum vértice novo seja marcado.
  3. Para cada vértice em G, se ele estiver conectado a um vértice já marcado, marque-o.
  4. Verifique todos os vértice de G. Se eles estão todos marcados, aceite a cadeia ✓; senão, rejeite a cadeia ✗"

$G =$



$\langle G \rangle =$

`(1,2,3,4)((1,2),(2,3),(3,1),(1,4))`

Na próxima aula vamos falar de problemas não computáveis.

> *"Equivalente a tese de Church-Turing, se não há **MT** para resolver um problema, então o problema simplesmente não tem solução computacional!"*

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.
3. *"Linguagens formais e autômatos"* de Paulo F. B. Menezes, 2002.
4. Materiais adaptados dos slides do Prof. Evandro E. S. Ruiz, da USP.