

Introdução à Teoria da Computação

Computabilidade

Professor Murilo V. G. da Silva

Departamento de Informática
Universidade Federal do Paraná

2025 / 2

Detalhes de “baixo nível” em MTs

- Algoritmos podem tomar como entrada números, grafos, árvores, etc
MTs apenas tomam strings binárias como entrada!
- Precisamos codificar a entrada como strings binárias

Exemplo: Seja $G = (V, E)$, onde $V = \{v_1, v_2, v_3\}$ e $E = \{v_1 v_2, v_1 v_3\}$ um grafo.

Como a matriz de adjacência deste grafo é $\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$, então podemos usar a seguinte string para representar o grafo: 011100100.

esquema de codificação utilizado aqui: concatenação das linhas da matriz

Notação: Dado um objeto matemático S ,

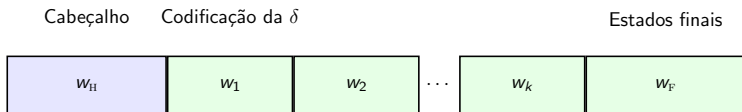
a notação $\lfloor S \rfloor$ se refere a string que representa a codificação de S em binário.

- No exemplo acima, $\lfloor G \rfloor = 011100100$.

Ponto chave: Máquinas de Turing também podem ser representadas em binário

Detalhes de “baixo nível” em MTs

- **Exercício:** Defina esquema de codificação para Máquinas de Turing:



(ver detalhes no livro)

- **Exercício:** Mostre qual é a string $\sqcup M_{01} \sqcup$ (M_{01} é o exemplo da primeira aula sobre MTs)

Mais detalhes de “baixo nível” para Máquinas de Turing:

Notação: Se uma MT toma múltiplas strings de entrada $(x_1, x_2, x_3, \dots, x_n)$, usaremos tanto a notação $M(x_1, x_2, \dots, x_n)$ quanto a notação $M(x_1x_2x_3\dots x_n)$.

Funções booleanas computáveis

Execução de um *Algoritmo* pode ser vista como uma **computação de uma função booleana**

Seja $f : \{0, 1\}^* \rightarrow \{0, 1\}$ uma função booleana.

Se existe um algoritmo M_f tal que:

- Se $f(x) = 1$, então M_f aceita x ;
- Se $f(x) = 0$, então M_f rejeita x

Neste caso dizemos que f é uma **função booleana computável**.

Funções booleanas computáveis

Agora considere a seguinte notação:

Notação: Suponha que uma string binária x é fornecida como entrada para uma MT M ,

Neste caso escreveremos:

- $M(x) = 1$ quando M aceita x e para.
- $M(x) = 0$ quando M rejeita x e para.
- $M(x) = \text{LOOP}$ quando M não para com a entrada x .
- $M(x) = \text{PARA}$ quando M para com a entrada x .

Observações elementares:

- (i) $M(x) = \text{PARA}$, então $M(x) = 0$ ou $M(x) = 1$.
- (ii) Se M é um algoritmo, então $\forall x \in \Sigma^*$, temos que $M(x) = \text{PARA}$.
- (iii) Se f é uma função booleana computável, então, por definição, $\exists M$ tal que $\forall x \in \Sigma^*$ temos $M(x) = f(x)$.

Funções computáveis

Notação: Dada uma MT $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ e $x \in \Sigma^*$. Escrevemos $M(x) = y$

- no caso em que $(\epsilon, q_0, x) \vdash_M^* (y', q_P, y'')$ tal que $y = y'y''$ e M para quando atinge a configuração (y', q_P, y'') .

(ver no livro: ambiguidade da notação)

- Exemplo: máquina M_{COPY} que duplica string: $M_{\text{COPY}}(x) = xx$ (exercício do livro)
- Exemplo: máquina M_{SOMA} : $M_{\text{SOMA}}(1010, 0010) = 1100$ (exercício do livro)
- Exemplo: Dados $a, b \in \mathbb{Z}$, máquina tal que $M(\ulcorner a \urcorner, \ulcorner b \urcorner) = \ulcorner a + b \urcorner$

Função computável: Uma função $f : \Sigma^* \rightarrow \Sigma^*$ para a qual existe uma MT M tal que $\forall x \in \Sigma^*, M(x) = f(x)$, é denominada uma *função computável*.

Note que uma *função booleana computável* é um caso particular de uma *função computável*.

O problema da parada

Def. O **problema da parada** é definido pela linguagem

$$L_H = \{ _L M _J x : \text{tal que } M \text{ é uma MT, } x \in \Sigma^* \text{ e } M(x) = \text{PARA} \}.$$

O que significaria resolver o problema da parada? Mostrar uma MT M_H que decide L_H .

Ou seja, uma MT M_H que tome $(_L M _J x)$ como entrada e faça:

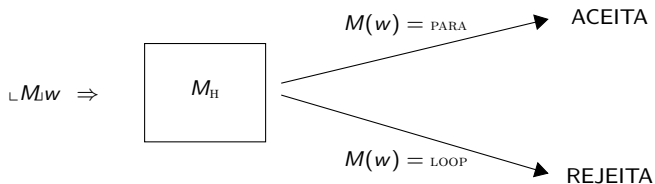
- Se $M(x) = \text{PARA}$, então $M_H(_L M _J x) = 1$.
- Se $M(x) = \text{LOOP}$, então $M_H(_L M _J x) = 0$.

O problema da parada

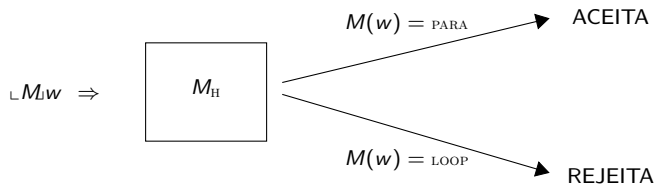
Teorema da Parada: Não existe algoritmo que decide a linguagem L_H .

Suponha que exista uma Máquina de Turing M_H que decida L_H .

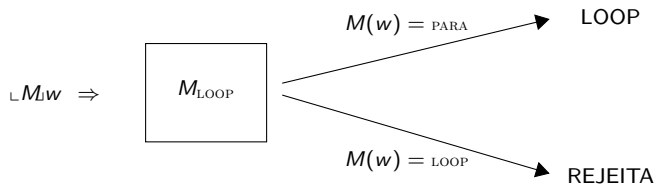
Portanto M_H se comporta assim:



O problema da parada



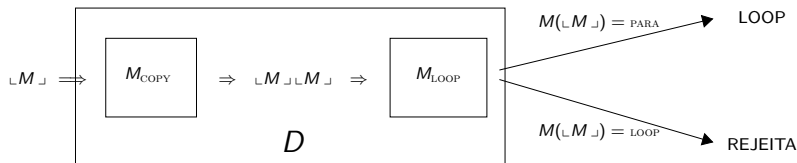
Se M_H existe, então M_{LOOP} (abaixo) deve existir:



O problema da parada

Agora vamos construir uma MT D da seguinte forma

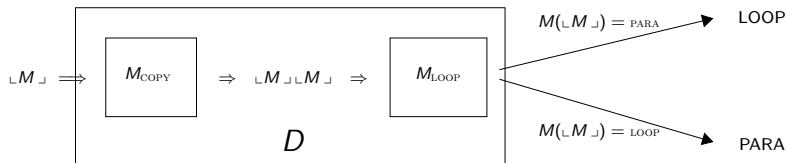
- D é a combinação de duas MTs distintas:
 - uma máquina M_{COPY} , que duplica a string de entrada: $M_{\text{COPY}}(w) = ww$
 - a máquina M_{LOOP} do slide anterior



O problema da parada

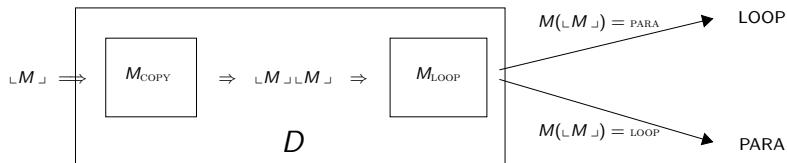
Agora vamos construir uma MT D da seguinte forma

- D é a combinação de duas MTs distintas:
 - uma máquina M_{COPY} , que duplica a string de entrada: $M_{\text{COPY}}(w) = ww$
 - a máquina M_{LOOP} do slide anterior



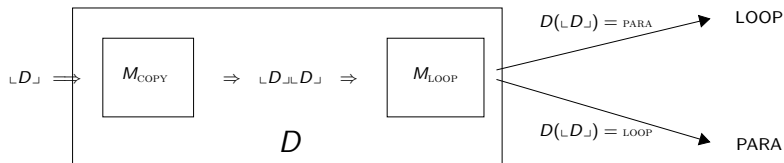
O comportamento de D com a entrada $\sqcup M \sqcup$ é o seguinte: $D(\sqcup M \sqcup) = \text{LOOP} \Leftrightarrow M(\sqcup M \sqcup) = \text{PARA}$.

O problema da parada



Lembrando: $D(\ulcorner M \urcorner) = \text{LOOP} \Leftrightarrow M(\ulcorner M \urcorner) = \text{PARA}$.

O que acontece quando fornecemos $\ulcorner D \urcorner$ como entrada para a máquina D ?



Ou seja, $D(\ulcorner D \urcorner) = \text{PARA} \Leftrightarrow D(\ulcorner D \urcorner) = \text{LOOP}$. Contradição, logo M_H não existe.

A Máquina de Turing Universal

Considere uma MT $\mathcal{U}$ que toma como entrada: uma máquina M e uma string x

obs: a rigor, a entrada de $\mathcal{U}$ é $(\perp M \perp, x)$

- $\mathcal{U}$ deve simular o comportamento de $M(x)$
($\mathcal{U}$ deve “executar” o programa M com a entrada x)
 - Ou seja, o resultado de $\mathcal{U}(\perp M \perp, x)$ deve ser o mesmo da computação de $M(x)$ (incluindo o caso em que $M(x) = \text{LOOP}$, quando $\mathcal{U}$ deve ficar em “loop infinito”).
 - Também deve valer no caso M computar funções não booleanas
Se $M(x) = y$, então $\mathcal{U}(\perp M \perp, x) = y$
- A primeira pergunta que devemos fazer: a máquina $\mathcal{U}$ existe?

Teorema: Existe uma MT $\mathcal{U}$ tal que $\forall$ MT M e $\forall x \in \Sigma^*$, temos $\mathcal{U}(\perp M \perp, x) = M(x)$.

A Máquina de Turing Universal

Teorema: Existe uma MT $\mathcal{U}$ tal que $\forall$ MT M e $\forall x \in \Sigma^*$, temos $\mathcal{U}(\lfloor M \rfloor, x) = M(x)$.

Idéia da Prova: A 7-tupla $\mathcal{U}$ é razoavelmente grande e complicada, mas ideia geral nem tanto.

Esboço de uma MT $\mathcal{U}_3$ com 3 fitas para a tarefa desejada:

(Pela equivalência entre uma e k fitas, $\mathcal{U}_3$ existe $\Rightarrow \mathcal{U}$ existe)

- Pré-processamento: $\lfloor M \rfloor$ deve ficar na primeira fita de $\mathcal{U}_3$ e x na segunda.
 - **Fita 1:** de $\mathcal{U}_3$: mantemos intacta a descrição do **programa M** que será executado
 - **Fita 2** de $\mathcal{U}_3$: exato **conteúdo da fita de M** a cada passo
 - **Fita 3:** de $\mathcal{U}_3$: **estado de M** a cada passo (um número em binário)
- A cada “ciclo” de $\mathcal{U}_3$ (isso leva um certo número de transições)
 $\mathcal{U}_3$ simula um passo de M e **atualiza a fita 2** para estar como a **fita (única) de M** .
(inclusive, o cabeçote da fita 2, fica posicionado na célula correta)
 - Caso M atingir estado final: $\mathcal{U}_3$ vai para estado final
(algum cuidado deve ser tomado para que $\mathcal{U}$ pare com a string correta na fita)
 - Caso M não tenha uma transição definida em $q \in Q \setminus F$: $\mathcal{U}_3$ irá para um estado não final sem nenhuma transição definida
 - Caso M fique em loop: $\mathcal{U}_3$ vai simplesmente continuar simulando M indefinidamente também

O problema da parada mais uma vez

Teorema: L_H é recursivamente enumerável.

Prova: A partir da MT universal $\mathcal{U}$, podemos construir uma MT que aceite L_H
(exercício do livro)

Corolário: $\mathcal{R} \subsetneq \mathcal{RE}$.

Prova: Consequência do teorema acima e de $\mathcal{R} \subseteq \mathcal{RE}$ (exercício do livro).

Teorema: Existe L tal que $L \notin \mathcal{RE}$.

Prova: O complemento de L_H não está em $\mathcal{RE}$. (exercício do livro)

Máquinas de Turing ou Pseudocódigo?

Considere o problema de testar por um número primo

- Com algum trabalho podemos projetar uma MT que faz isso
- Mas em geral **NÃO** vamos fazer isso.

Vamos apresentar um pseudocódigo, como abaixo:

Primo: (N)

```
1: if  $N = 1$  then  
2:   Retorna Falso  
3: for  $i = 2; i \leq \sqrt{N}; i++$  do  
4:   if  $N \bmod i = 0$  then  
5:     Retorna Falso  
6: Retorna Verdadeiro
```

Mais adiante vamos provar que MTs e programas escritos em *assembly* são equivalentes

- Programas escritos em C, C++, Python, etc são equivalentes a programas em assembly (isso vale para qualquer linguagem)
- Programas em pseudocódigo também (desde que o pseudocódigo seja escrito de maneira rigorosa)

Quando usar MTs e quando usar pseudocódigo? (ver discussão no livro)

MTs não determinísticas

Máquina de Turing não determinística (MTN): uma 7-tupla $N = (Q, \Sigma, \Gamma, (\delta_0, \delta_1), q_0, B, F)$.

- $Q, \Sigma, \Gamma, q_0, B, F$ iguais à definição de MTs;
- Um par (δ_0, δ_1) de **funções de transição**,
(sendo cada uma delas idêntica às funções de MT determinísticas)

Funcionamento da máquina: A cada passo, a máquina advinha qual função usar

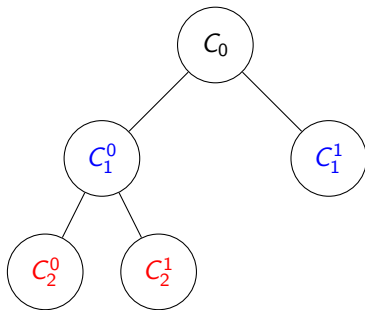
- Existem outras definições para MTNs,
e.g., uma única função $\delta(p, X)$ que devolve um conjunto de triplas

$$\{(q_1, Y_1, D_1), (q_2, Y_2, D_2), \dots, (q_k, Y_k, D_k)\}$$

Uma **configuração** de uma MTN: uma tripla (α, q, β) (análogo às MTs)

- Passos computacionais definidos de maneira análoga também
- Porém, existe uma **Árvore de computações possíveis**:
 $\Rightarrow$ árvore de configurações

Árvore de computações possíveis



Neste exemplo:

- Configuração inicial da máquina: C_0
- $\exists C_1^0$ e C_1^1 tal que as duas possibilidades são válidas: $C_0 \vdash_N C_1^0$ e $C_0 \vdash_N C_1^1$.
- $\exists C_2^0$ e C_2^1 tal que $C_1^0 \vdash_N C_2^0$ e $C_1^0 \vdash_N C_2^1$.
- Configurações em que a máquina para: C_2^0 , C_2^1 e C_1^1
(não necessariamente finais)

Computação não determinística

Def.: Linguagem de uma MTN N , denotado $L(N)$ (ver livro)

Teorema: Se L é aceita por uma MTN N , então $\forall x \in L$, a árvore de computações possíveis de N com x tem pelo menos um ramo finito.

Prova: Se $x \in L(N)$, existe configuração final C_F tal que $(\epsilon, q_0, x) \vdash_N^* C_F$.

Teorema: Se N é uma MTN, então existe uma MT M tal que $L(N) = L(M)$.

Prova: Busca em largura na árvore de computações possíveis (detalhes no livro)

Teorema: Se M é uma MT, então existe uma MTN N tal que $L(M) = L(N)$.

Prova: Seja $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$

- Faça $N = (Q, \Sigma, \Gamma, (\delta, \delta), q_0, B, F)$