Decidibilidade

Hipótese de Church-Turing

Qualquer modelo de computador possível de ser construído fisicamente pode ser simulado por uma máquina de Turing.

- Pode ser de silício, DNA, neurônio, ou qualquer outra tecnologia

- Como Máquinas de Turing podem simular nossos computadores atuais e como nossos computadores atuais podem executar nossos pseudo-códigos, podemos pensar que ao descrever um pseudo-código estamos descrevendo uma máquina de Turing

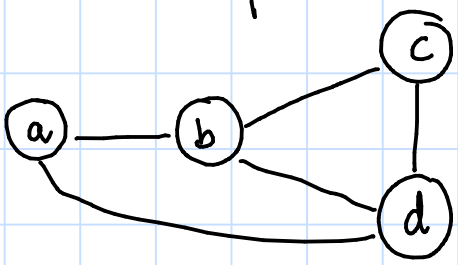
  ↓

  a máquina de Turing que simula a execução do computador executando o nosso programa.

- Chamaremos essa descrição de uma máquina de Turing de descrição de alto nível.

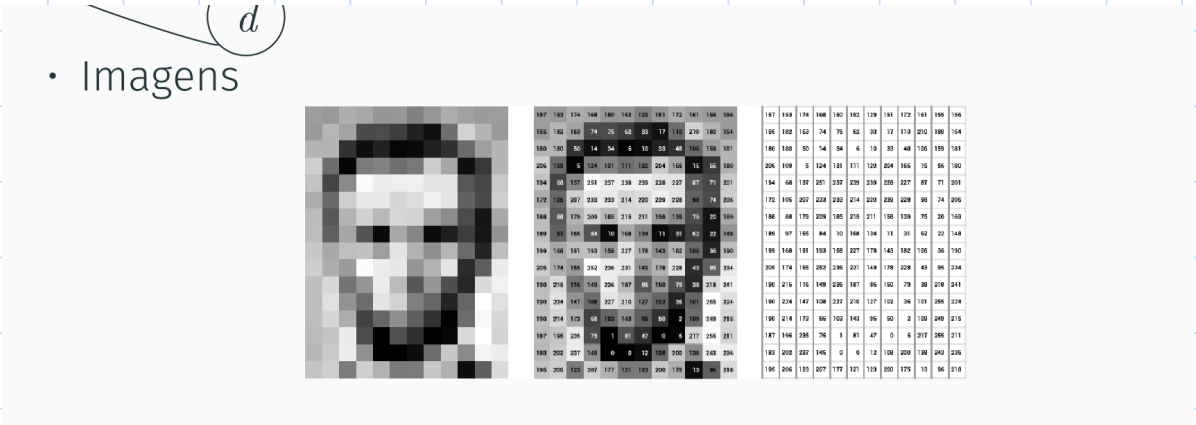
Já vimos que tudo é ~~número~~ cadeia:

- número: $5 \rightarrow$ "5" ou "00101"
- Polinômios: $4x^2 + 3y - 5 \rightarrow$ "4 x ^ 2 + 3y - 5"
- Grafos:



$\Rightarrow$ "({a, b, c, d}, {(a, b), (b, c), (c, d), (b, d), (a, d)})"

- imagens:



Definição

Se O é um objeto, denotemos por $\langle O \rangle$ a sua codificação em uma cadeia.

Definição

Se $O_1, O_2, \ldots, O_k$ é uma coleção de objetos, denotemos por $\langle O_1, O_2, \ldots, O_k \rangle$ a sua codificação em uma única cadeia.

Convenção: Descrição p/ máquina de Turing de alto nível

- Vamos descrever máquinas de Turing em blocos indentados e entre " "
- Se w for a entrada de uma MT, então w é uma cadeia.
- Se a entrada para uma MT for um objeto codificado <O>, então assumimos que a MT faz, antes de tudo, uma verificação implícita se a entrada é um objeto codificado corretamente. Caso não seja, a entrada é rejeitada.

Exemplo

* Um grafo é conexo se existe um caminho entre todo par de vertices
* Seja A a linguagem que contém todas as cadeias que representam grafos conexos.

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

M = " Sobre a entrada <G>, a codificação de um grafo G:

1. Selecione o primeiro nó de G e marque-o
2. Repita o próximo estágio até que nenhum novo nó seja marcado
3. Para cada nó em G, marque-o se ele estiver ligado a um nó marcado
4. Percorra a lista de nós de G para determinar se estão todos marcados. Se estiverem aceite; caso contrário, rejeite. "

Problema de aceitação para AFD

Problema: Dado um AFD M e uma cadeia w, M aceita w?
Existe um algoritmo que resolve esse problema?

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

A_{FD} é decidível?

↳ Resolver um problema é igual a decidir uma linguagem

Teorema A_{AFD} é decidível.

Demonstração

M = "Sobre a entrada $\langle B, w\rangle$, onde B é um AFD e w uma cadeia:

1. Simule a execução de B sobre w.
2. Se a simulação para em um estado final, aceite; caso contrário, rejeite

"

■

↳ Conseguimos escrever tal programa?
R: Sim

⇒ Então a MT é capaz de fazer isso pois ela consegue simular um computador.

Algoritmo

* entrada: $\langle B, w \rangle$
* Saída:
  * Sim se a simulação do AFD B com a cadeia de entrada w, para em um estado final.
  * Não caso contrário.

Algoritmo → programa → código de máquina → simulado por MT

$$A_{AFN} = \{\langle B, w \rangle \mid B \text{ é um AFN que aceita a cadeia } w\}$$

Teorema A_{AFN} é decidível.

Demonstração 1

N = "Sobre a entrada $\langle B, w \rangle$, onde B é um AFN e w uma cadeia:

1. Simule a execução de B sobre w.
2. Se a simulação para em um estado final, aceite; caso contrário, rejeite."

■

Demonstração 2

N = "Sobre a entrada $\langle B, w \rangle$, onde B é um AFN e w uma cadeia:

1. Construa um AFD C equivalente ao B.
2. Execute M sobre a entrada $\langle C, w \rangle$.
3. Se M aceita, então aceite; caso contrário rejeite."

■

$$A_{REGEX} = \{\langle R, w\rangle \mid R \text{ é uma regex que descreve } w\}$$

Teorema A_{regex} é decidível

Demonstração

P = "Sobre a entrada $\langle R, w\rangle$, onde R é uma expressão regular e w uma cadeia, faça:

1. Construa um AFN A que é equivalente à R.
2. Execute N sobre $\langle A, w\rangle$
3. Se N aceitar, aceite; caso contrário, rejeite."

□

Observação:

para propósitos de decidibilidade, AFD, AFN e regex são equivalentes pois a MT é capaz de converter uma representeção em outra.

Problema de testar se AFD reconhece alguma cadeia

Problema: Dado um AFD M, M reconhece alguma cadeia?
Existe um algoritmo que resolve esse problema?

$$V_{AFD} = \{\langle A\rangle \mid A \text{ é um AFD e } L(A) = \emptyset\}$$

V_{AFD} é decidível?

teorema V_{AFD} é decidível.

Demonstração

T = « Sobre a entrada $\langle A \rangle$, onde A é um AFD, faça:

1. Marque o estado inicial
2. Repita até que nenhum novo estado seja marcado
3. Marque um estado não marcado que possui uma transição vindo de um estado marcado
4. Se nenhum estado final estiver marcado, aceite; Caso contrário, rejeite

» ■

Problema:

Dado dois AFD's A e B, eles reconhecem a mesma linguagem?

Existe um algoritmo que resolve esse problema?

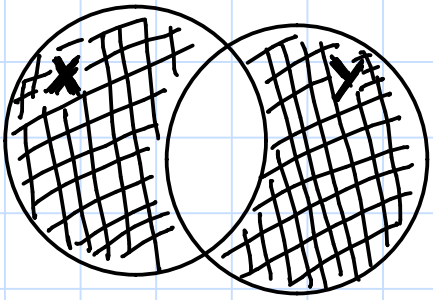
$$I_{AFD} = \{ \langle A, B \rangle \mid A \text{ e } B \text{ são AFD's e } L(A) = L(B) \}$$

I_{AFD} é decidível?

Teorema. I_{AFD} é decidível

A diferença simétrica entre dois conjuntos X e Y, denotada por $X \Delta Y$, é definida como

$$X \Delta Y = (X \cap \overline{Y}) \cup (\overline{X} \cap Y)$$



$$X = Y \iff X \Delta Y = \emptyset$$

Linguagens regulares são fechadas sob as operações de complemento, união, intersecção

Se A e B são AFD's, sabemos construir AFD's que reconhecem:

- $\overline{L(A)}$
- $L(A) \cup L(B)$
- $L(A) \cap L(B)$

$$L(A) \Delta L(B) = \left(L(A) \cap \overline{L(B)}\right) \cup \left(\overline{L(A)} \cap L(B)\right)$$

Teorema. I_{DFA} é decidível

Demonstração

F = "Sob a entrada $\langle A, B \rangle$, onde A e B são AFD's:

1. Construa um AFD C que reconhece $L(A) \Delta L(B)$
2. Execute a MT T sobre $\langle C \rangle$
3. Se T aceita, então aceite; caso contrário rejeite."

■

Problema de aceitação para uma Gramática

Problema: Dada uma gramática G e uma cadeia w, G gera w?
Existe um algoritmo que resolve esse problema?

problema fundamental em compiladores!

$$A_G = \{\langle G, w \rangle \mid G \text{ é uma gramática e } w \text{ uma cadeia}\}$$

A_G é decidível?

$S \to AB$

$A \to 0A \mid \varepsilon$

$B \to 1B2 \mid \varepsilon$

$S \Rightarrow AB \Rightarrow 0AB \Rightarrow 00AB \Rightarrow 0000AB \Rightarrow \ldots$

$B \Rightarrow 1B2 \Rightarrow 01B22 \Rightarrow 012 \Rightarrow \ldots$

$B \Rightarrow \varepsilon$

- se $w \in L \Rightarrow$ o algoritmo irá encontrar a derivação de w
- se $w \notin L \Rightarrow$ o algoritmo executará para sempre

esse algoritmo é um reconhecedor, mas não é decisor

Fórma normal de Chomsky (FNC)

* Formato das Produções
  - $A \to BC$
  - $A \to a$
* Símbolo inicial não aparece no lado direito.
* Se S é o símbolo inicial da gramática, permitimos $S \to \varepsilon$

Teorema 2.9 Toda linguagem livre de contexto pode ser gerada por uma gramática na fórma normal de Chomsky ■

$S \to AB$

$A \to AA \mid a$

$B \to BB \mid b$

4 Passos ↓

$S \Rightarrow AB \Rightarrow AAB \Rightarrow AAAB \Rightarrow AAABB$

$\Rightarrow aAABB \Rightarrow aaABB \Rightarrow aaaBB \Rightarrow aaabB$

$\Rightarrow aaabb$

Se a gramática está na FNC e w é uma palavra gerada pela gramática, então a derivação de w tem exatamente $2|w|-1$ derivações imediatas.

$|w| = 5$

$4 + 5$

Teorema A_G é decidível

Demonstração

S = " Sobre a entrada $\langle G, w \rangle$, onde G é uma gramática e w uma cadeia:

1. Construa uma gramática G' na forma normal de Chomsky equivalente à G.
2. Liste todas as derivações com $2|w|-1$ derivações imediatas; exceto se $w = \varepsilon$, onde listamos todas com uma derivação imediate.
3. Se alguma derivação gera w, aceite; caso contrário, rejeite. "

■

Problema de testar se a Gramática Gera alguma cadeia

Problema: Dada uma gramática G, G gera alguma cadeia?
Existe um algoritmo que resolve esse problema?

$$V_G = \{ \langle G \rangle : G \text{ é uma gramática e } L(G) = \emptyset \}$$

V_G é decidível?

Qual variável é capaz de Gerar sentenças?

$S \to ABCD$
$A \to BCA$
$A \to xyy$
$B \to CA$
$B \to AB$
$B \to BBBw$
$C \to CB$
$C \to ww$

$L(G) = \emptyset$

teorema V_G é decidível.

Demonstração

R = "Sobre a cadeia $\langle G \rangle$, onde G é uma gramática

1. Marque todos os terminais
2. Repita até que nenhuma nova variável seja marcada:
3. Marque toda variável A que é cabeça de uma produção $A \to U_1 U_2 U_3 \cdots U_k$, onde todo símbolo U_i está marcado
4. Se a variável inicial $\bar{n}$ está marcada, aceite; caso contrário, rejeite."

■

Problema: Dada duas gramáticas A e B, eles são equivalentes?
Existe um algoritmo que resolve esse problema?

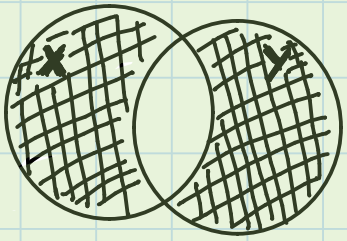
$$I_G = \{ \langle A, B \rangle : A \text{ e } B \text{ são gramáticas e } L(A) = L(B) \}$$

I_G é decidível?

Teorema. I_{APD} é decidível

A diferença simétrica entre dois conjuntos X e Y, denotada por $X \Delta Y$, é definida como

$$X \Delta Y = (X \cap \overline{Y}) \cup (\overline{X} \cap Y)$$



$$X = Y \iff X \Delta Y = \emptyset$$

NÃO é possível criar um algoritmo que recebe duas gramáticas e determina se elas são equivalentes.

Alg $X(G_1, G_2)$
Lo sim $L(G_1) = L(G_2)$.
não $L(G_1) \neq L(G_2)$.

→ Vamos provar isso qndo aprendermos reduções!

Teorema Toda linguagem livre de contexto é decidível.

Demonstração

* Seja L uma linguagem livre de Contexto
* Seja G uma gramática que gera L
* A MT M_L, descrita a seguir, decide L

M_L = "Sobre a entrada w, faça:

1. Execute a MT S sobre a entrada $\langle G, w \rangle$
2. Se S aceita, então aceite; caso contrário, rejeite."

O Problema da Parada

Introdução

* Vamos provar que existem problemas para os quais não existem algoritmos.

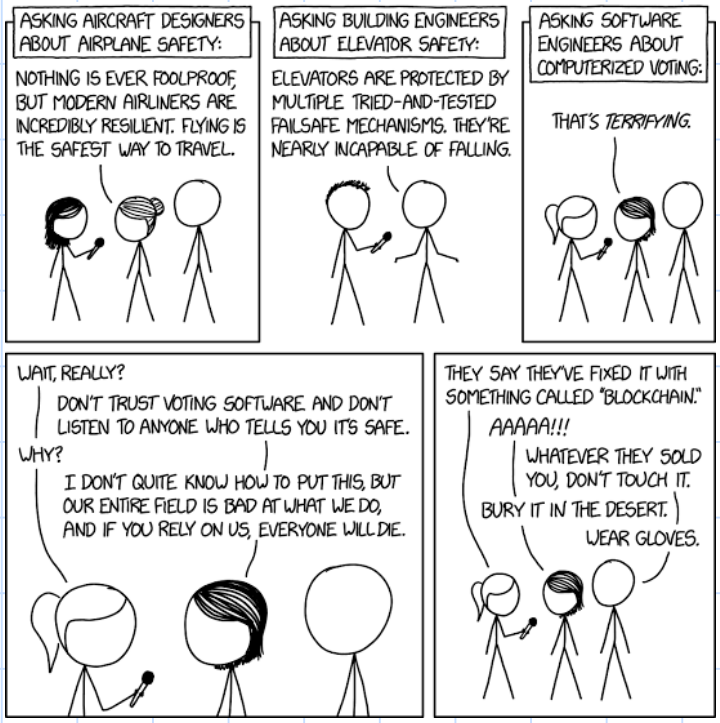
"ahh... professor... esses são aqueles problemas teóricos que ninguém quer realmente resolver!"

↳ Será?!

Problema

* Entrada: um programa (fonte) e uma especificação formal do que o programa deve fazer.
* Saída: sim, se o programa faz o que foi especificado; não, caso contrário.

(Esse problema é Indecidível)



Problema de aceitação para MT

Problema: Dado uma MT M e uma cadeia w, M aceita w?
Existe um algoritmo que resolve esse problema?

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

A_{MT} é decidível?

Teorema A_{MT} é indecidível. ← Já vamos provar isso!

* A_{AFD} e A_G eram decidíveis

Teorema A_{MT} é Turing-reconhecível

Demonstração

U = " Sobre a entrada $\langle M, w \rangle$, onde M é uma MT, faça:

1. Simule M sobre a entrada w;
2. Se M aceita, então aceite; caso contrário, rejeite.

"

■

* Note que U para se e somente se M para!

O Paradoxo do Barbeiro

Em uma cidade onde todos os homens tiram a barba, existe um barbeiro.

O barbeiro faz a barba de todos os homens que não fazem a propria barba e apenas destes.

Quem faz a barba do barbeiro?

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

Teorema A_{MT} é indecidível.

Demonstração

* Suponha, para uma contradição, que A_{MT} é decidível.
* Suponha que H é uma MT que decide A_{MT}
* Sobre uma entrada $\langle M, w\rangle$:

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

* Seja D a MT definida por

$D =$ " Sobre a entrada $\langle M\rangle$, onde M é uma MT

1. Execute H sobre $\langle M, \langle M\rangle\rangle$
2. Se H aceita, então rejeite; Se H rejeita, então aceite. "

> * Um compilador em C escrito em C compilando o próprio código.
> * Um formatador de código (tabs, espaçamento) formatando o proprio código.

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

* Qual a saída de $D(\langle D\rangle)$?

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

■

* Polinômios de uma variável: $4x^3 - 2x^2 + x + 7$

$$2D_1 = \{\langle p \rangle : p \text{ é um polinômio sobre } x \text{ com raiz inteira}\}$$

M_1 = " entrada $\langle p \rangle$, codificação de polinômio p

1. Avalie p sucessivamente com x sendo $0, 1, -1, 2, -2, 3, -3, \ldots$. Se em algum ponto der 0, aceite a cadeia. "

reconhecedor pr a linguagem $2D_1$

Implementação de M_1

```python
def encontraRaizInteira(polinomio: str) -> bool:
    # <encontraRaizInteira, 6x^3yz^2 + 3xy^2 - x^3 - 10> ∈ A_MT

    variaveis = extraiVariaveis(polinomio)
    for valores in GeradorAtribuicoes(variaveis):
        if evaluate(polinomio, valores) == 0:
            return True

def verificaPara(funcao: str, entrada: str) -> bool: # MT H
```

aceita

```python
def funcaoDoMal(funcao: str) -> bool:   # MT D

    if verificaPara(funcao, funcao):
        return False
    else:
        return True
```

```python
verificaPara_str = """
def verificaPara(funcao: str, entrada: str) -> bool:
```

```
"""

funcaoDoMal_str = """
def funcaoDoMal(funcao: str) -> bool:

    if verificaPara(funcao, funcao):
        return False
    else:
        return True
"""
```

```python
funcaoDoMal(funcaoDoMal_str)
```

$d_2 \times 256^1 + d_1 \times 256^0$

$356 = 3 \cdot 10^2 + 5 \cdot 10^1 + 6 \times 10^0$

$0101 =$

UI $\underline{\text{char}}$ = int pequeño

↳ [-255, 255]

$2^8 = 256$

(signed char = [-128, 127]

unsigned char [0, 256]

356

Entrada: 6 | 7 8 → base 256

7 2

256 1

	2	166

0
g
A
B
C

256
256
512

678
-512
166