

Introdução à Teoria da Computação

A Tese de Church-Turing

Professor Murilo V. G. da Silva

Departamento de Informática
Universidade Federal do Paraná

2025 / 2

Perspectiva Histórica

Início do século XX: matemáticos notaram que a demonstração de teoremas se assemelha a um procedimento mecânico.

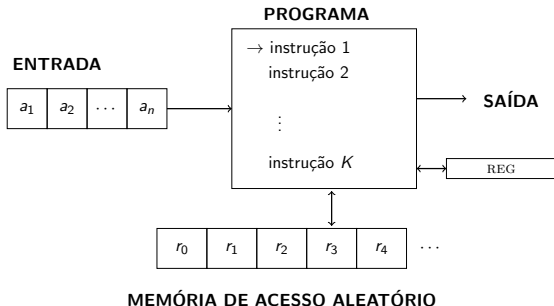
- **David Hilbert (1928)**: queria um “procedimento efetivo” que decida se uma afirmação matemática é verdadeira ou falsa. (queria um algoritmo para o *Entscheidungsproblem*)
- **Kurt Gödel (1931)**: provou que nem todo teorema verdadeiro pode ser provado dentro de sistemas consistentes e suficientemente expressivos.

Mas, pelo menos, existe um algoritmo capaz de decidir se há uma prova?

- **Alonzo Church e Alan Turing (1936)**: provaram que não existe tal algoritmo. Ambos formalizaram o conceito de algoritmo.
- Turing ainda introduziu a propriedade de *universalidade*: computadores de propósito geral podem executar algoritmos como entrada.
- Vamos discutir a tese de Máquinas de Turing (e modelos equivalentes), de fato, correspondem ao que queremos dizer por algoritmo.

⇒ Antes disso, vamos provar que MTs são capazes de expressar programas em assembly

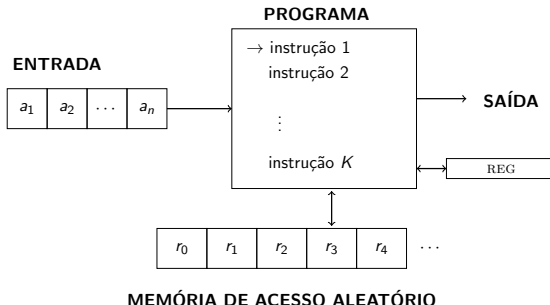
O Modelo RAM – Programas Assembly



Exemplo: Programa para somar dois números

1. READ
2. STORE ^1
3. READ
4. ADD ^1
5. STORE ^1
6. HALT

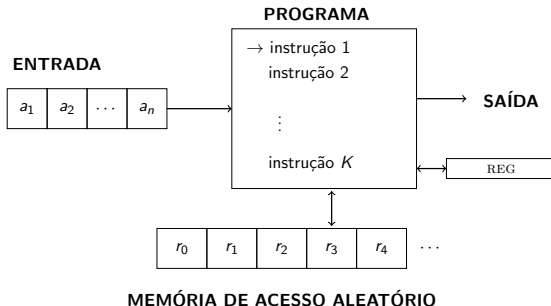
O Modelo RAM – Programas Assembly



Exemplo: Programa para somar dois números

- | | | | |
|----|-------|------------|---|
| 1. | READ | | Lê número inteiro a_1 da entrada e carrega em REG |
| 2. | STORE | $\wedge 1$ | |
| 3. | READ | | |
| 4. | ADD | $\wedge 1$ | |
| 5. | STORE | $\wedge 1$ | |
| 6. | HALT | | |

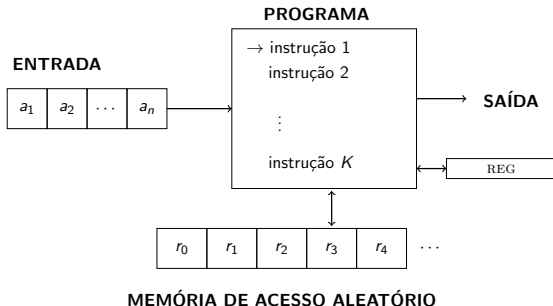
O Modelo RAM – Programas Assembly



Exemplo: Programa para somar dois números

- | | | | |
|----|-------|------------|--|
| 1. | READ | | Lê número inteiro a_1 da entrada e carrega em REG |
| 2. | STORE | $\wedge 1$ | Armazena o número de REG na posição de memória r_1 |
| 3. | READ | | |
| 4. | ADD | $\wedge 1$ | |
| 5. | STORE | $\wedge 1$ | |
| 6. | HALT | | |

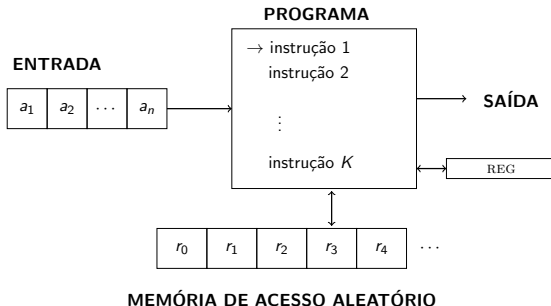
O Modelo RAM – Programas Assembly



Exemplo: Programa para somar dois números

- | | | | |
|----|-------|------------|--|
| 1. | READ | | Lê número inteiro a_1 da entrada e carrega em REG |
| 2. | STORE | $\wedge 1$ | Armazena o número de REG na posição de memória r_1 |
| 3. | READ | | Lê número inteiro a_2 da entrada e carrega em REG |
| 4. | ADD | $\wedge 1$ | |
| 5. | STORE | $\wedge 1$ | |
| 6. | HALT | | |

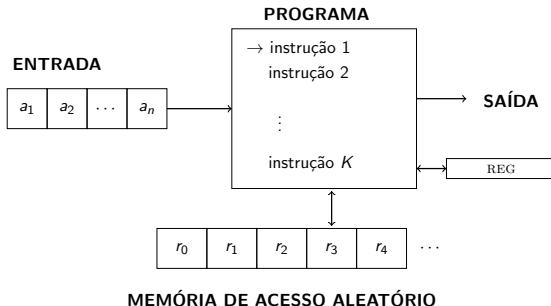
O Modelo RAM – Programas Assembly



Exemplo: Programa para somar dois números

- | | | | |
|----|-------|------------|--|
| 1. | READ | | Lê número inteiro a_1 da entrada e carrega em REG |
| 2. | STORE | $\wedge 1$ | Armazena o número de REG na posição de memória r_1 |
| 3. | READ | | Lê número inteiro a_2 da entrada e carrega em REG |
| 4. | ADD | $\wedge 1$ | Soma número em REG com número em r_1 ; |
| 5. | STORE | $\wedge 1$ | |
| 6. | HALT | | |

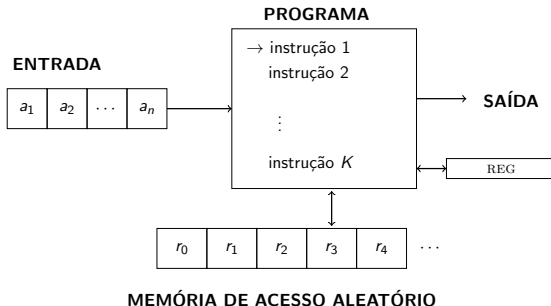
O Modelo RAM – Programas Assembly



Exemplo: Programa para somar dois números

- | | | | |
|----|-------|------------|--|
| 1. | READ | | Lê número inteiro a_1 da entrada e carrega em REG |
| 2. | STORE | $\wedge 1$ | Armazena o número de REG na posição de memória r_1 |
| 3. | READ | | Lê número inteiro a_2 da entrada e carrega em REG |
| 4. | ADD | $\wedge 1$ | Soma número em REG com número em r_1 ; Resultado fica em REG |
| 5. | STORE | $\wedge 1$ | |
| 6. | HALT | | |

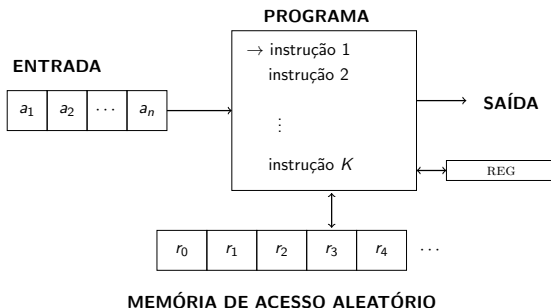
O Modelo RAM – Programas Assembly



Exemplo: Programa para somar dois números

- | | | | |
|----|-------|------------|--|
| 1. | READ | | Lê número inteiro a_1 da entrada e carrega em REG |
| 2. | STORE | $\wedge 1$ | Armazena o número de REG na posição de memória r_1 |
| 3. | READ | | Lê número inteiro a_2 da entrada e carrega em REG |
| 4. | ADD | $\wedge 1$ | Soma número em REG com número em r_1 ; Resultado fica em REG |
| 5. | STORE | $\wedge 1$ | Armazena valor de REG em r_1 |
| 6. | HALT | | |

O Modelo RAM – Programas Assembly



Exemplo: Programa para somar dois números

- | | | | |
|----|-------|------------|--|
| 1. | READ | | Lê número inteiro a_1 da entrada e carrega em REG |
| 2. | STORE | $\wedge 1$ | Armazena o número de REG na posição de memória r_1 |
| 3. | READ | | Lê número inteiro a_2 da entrada e carrega em REG |
| 4. | ADD | $\wedge 1$ | Soma número em REG com número em r_1 ; Resultado fica em REG |
| 5. | STORE | $\wedge 1$ | Armazena valor de REG em r_1 |
| 6. | HALT | | Para a execução |

Modelo RAM – Semântica completa

READ		A i -ésima chamada desta instrução carrega em REG o valor a_i da entrada.
WRITE		Escreve o conteúdo de REG na saída.
LOAD	i	Carrega o inteiro i em REG.
LOAD	$\wedge i$	Carrega o conteúdo de r_i em REG.
STORE	$\wedge i$	Armazena o conteúdo de REG no registrador r_i da memória.
ADD	i	Seja $v(\text{REG})$ o valor contido em REG. Depois da instrução, o registrador REG passa a conter o valor $v(\text{REG}) + i$.
ADD	$\wedge i$	Sejam $v(\text{REG})$ e $v(r_i)$ os valores em REG e r_i , respectivamente. Depois da instrução, o valor em REG passa a ser $v(\text{REG}) + v(r_i)$.
SUB	i	Seja $v(\text{REG})$ o valor contido em REG. Depois da instrução, o registrador REG passa a conter o valor $v(\text{REG}) - i$.
SUB	$\wedge i$	Sejam $v(\text{REG})$ e $v(r_i)$ os valores em REG e r_i , respectivamente. Depois da instrução, o valor em REG passa a ser $v(\text{REG}) - v(r_i)$.
MULT	i	Seja $v(\text{REG})$ o valor contido em REG. Depois da instrução, o registrador REG passa a conter o valor $v(\text{REG}) \times i$.
MULT	$\wedge i$	Sejam $v(\text{REG})$ e $v(r_i)$ os valores em REG e r_i , respectivamente. Depois da instrução, o valor em REG passa a ser $v(\text{REG}) \times v(r_i)$.
JUMP	i	Muda o valor do contador de programa para i .
JZERO	i	Muda o contador de programa para i , caso o conteúdo de REG seja zero.
JPOS	i	Muda o contador de programa para i , caso o conteúdo de REG seja positivo.
JNEG	i	Muda o contador de programa para i , caso o conteúdo de REG seja negativo.
HALT		Interrompe a execução do programa. (ver livro)

Modelo RAM

Exemplo: Programa que lê três números a, b, c e verifica se $a + b = c$.

- Se $a + b = c$, escreve 1 na saída; caso contrário, escreve 0.

```
1.  READ
2.  STORE  ~1
3.  READ
4.  ADD    ~1
5.  STORE  ~1
6.  READ
7.  SUB    ~1
8.  JZERO  11
9.  WRITE  0
10. HALT
11. WRITE  1
12. HALT
```

Simulando instruções usando MTs de 5 fitas

Vamos simular uma Máquina RAM com uma MT de 5 fitas.

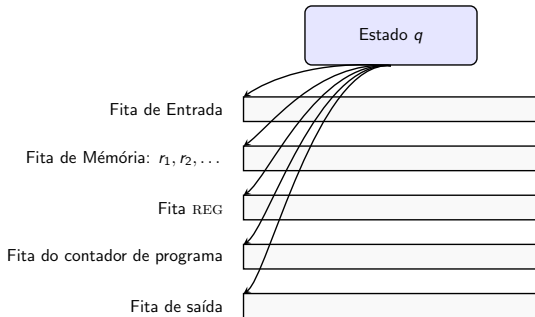
Fita 1: Entrada;

Fita 2: Memória (registradores $r_1, r_2, \dots$ em uso);

Fita 3: Registrador REG;

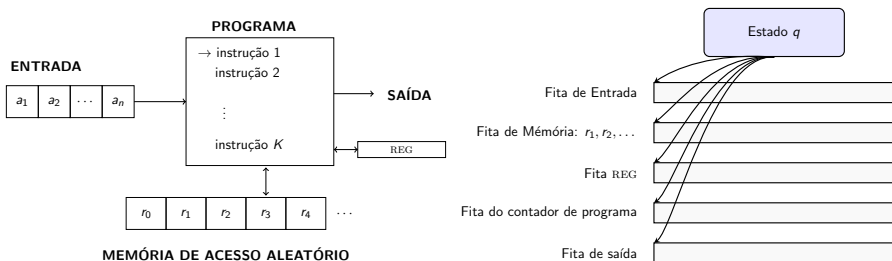
Fita 4: *Contador de programa*;

Fita 5: Saída;



- Exemplo: Digamos que na memória r_1 contenha 7 e r_3 contenha 4:
Neste caso a fita 2 contém $_1_7_3_4_$

Simulando instruções usando MTs de 5 fitas



Exercícios:

- (a) Simular a instrução $\text{ADD } r_3$: soma conteúdo da fita REG ao conteúdo de r_3 (memória), guarda o resultado na fita REG, e termina sem alteração nas demais fitas,

Demais exercícios: (simular instruções LOAD 4, READ, JUMP 10, etc)

Simulando Máquinas RAM com MTs

Teorema: Para todo Programa Assembly Π , existe uma MT M_Π tal que $L(M_\Pi) = L(\Pi)$.

Esboço da Prova: Vamos criar Máquinas de Turing distintas, cada uma delas com cinco fitas, que executam cada uma das instruções assembly. Mais precisamente,

- A partir de $\Pi = \pi_1, \pi_2, \dots, \pi_k$, primeiramente, especifique as MTs $M_1, M_2, \dots, M_k$; (cada MT pode ser construída como no exercício anterior)
- Partindo destas k máquinas, podemos construir a MT M_Π com cinco fitas que contém “subrotinas” $M_1, \dots, M_k$. A MT M_Π executa em ciclos:
 - A cada ciclo, M_Π olha na fita do contador de programa e decide qual sub-rotina M_i deve ser executada;
 - Após executar M_i , o contador é atualizado por M_Π da seguinte forma:
se a sub-rotina não for do tipo JUMP, atualiza o valor para $i + 1$;
caso contrário, a alteração já terá sido feita pela própria sub-rotina M_i .

Teorema: Para toda MT M , existe um Programa Assembly Π_M tal que $L(\Pi_M) = L(M)$.

Esboço da Prova: A partir de M , podemos especificar um programa que tem uma tabela para a função δ . A cada passo, aplica a função δ na entrada.

Equivalência entre modelos de computação

Teorema: Se um algoritmo pode ser escrito em um dos modelos de computação abaixo, então também pode ser escrito no outro modelo:

- (1) Máquinas de Turing com k fitas, para qualquer k ;
- (2) Algoritmos em pseudocódigo.

- Quais outros modelos são equivalentes a Máquinas de Turing? Muitos!
- Modelo muito importante: Modelo matemático para especificar algoritmos quânticos
⇒ modelo de circuitos quânticos

Equivalência entre modelos de computação

Teorema: Se um algoritmo pode ser escrito em um dos três modelos de computação abaixo, então também pode ser escrito nos outros dois modelos:

- (1) Máquinas de Turing com k fitas, para qualquer k ;
- (2) Algoritmos em pseudocódigo;
- (3) Circuitos Quânticos

No modelo (3), um programa é uma *matriz* U , a entrada é um *vetor* v

Execução do programa: Basta fazer multiplicação $U \cdot v$ para obter a saída

- **Richard Feynman (1982)**: De acordo com as leis da mecânica quântica, a evolução temporal de um sistema físico isolado (átomos, elétrons, fótons, etc) pode ser modelada, com precisão arbitrária, por uma abstração matemática chamada *circuito quântico*.
- **David Deutsch (1985)**: Qualquer sistema físico, e não apenas sistemas quânticos microscópicos, pode, em princípio, ser simulado por um computador quântico universal.

Em outras palavras, **circuitos quânticos, em certo sentido, modelam “qualquer objeto físico”**

Equivalência entre modelos de computação

O enunciado do Teorema da Equivalência poderia ser expandido para incluir muitos outros modelos de computação. Por exemplo:

- (1) Qualquer linguagem de programação que rode em computadores conhecidos;
- (2) Máquinas de Turing com fitas n -dimensionais, em vez de fitas unidimensionais;
- (3) Cálculo λ ;
- (4) Funções μ -recursivas;
- (5) O modelo *P-systems* para computação celular;
- (6) O modelo *SCRNs* para computação por reações químicas;
- (7) Modelo ATAM para computação com moléculas de DNA;
- (8) Modelo de Circuitos Quânticos e Máquinas de Turing Quânticas.

Equivalência entre modelos de computação

O enunciado do Teorema da Equivalência poderia ser expandido para incluir muitos outros modelos de computação. Por exemplo:

- (1) Qualquer linguagem de programação que rode em computadores conhecidos;
- (2) Máquinas de Turing com fitas n -dimensionais, em vez de fitas unidimensionais;
- (3) Cálculo λ ;
- (4) Funções μ -recursivas;
- (5) O modelo P -systems para computação celular;
- (6) O modelo $SCRNs$ para computação por reações químicas;
- (7) Modelo ATAM para computação com moléculas de DNA;
- (8) Modelo de Circuitos Quânticos e Máquinas de Turing Quânticas.

Acima: modelos de interesse matemáticos e modelos para sistemas físicos

O que afirma a Tese de Church-Turing

Tese de Church-Turing (TCT): Máquinas de Turing capturam o conceito de *algoritmo*.

- Além de **algoritmo**, outros termos associados a tese: *computação efetiva*, *procedimento efetivo*, *processo computacional*, *algoritmo executável*, etc
- Existem duas leituras principais do que a TCT quer dizer:
 - **Interpretação matemática:** a TCT é uma *definição formal* para algoritmo.
 - **Interpretação física:** a TCT é uma *afirmação sobre o mundo físico*, ou seja, sobre o que pode ser computado por sistemas fisicamente realizáveis.

Nesta segunda leitura, a TCT normalmente é enunciada de uma forma diferente:

“Se um problema computacional pode ser resolvido por algum dispositivo fisicamente realizável, então ele pode ser resolvido por uma Máquina de Turing”

Interpretação matemática da TCT

A analogia do círculo:

- Todos temos uma **ideia intuitiva** do que é um círculo.
- Para estudar o conceito rigorosamente, precisamos de uma **definição exata**.

A definição matemática padrão:

Para todo real $r > 0$ e todo $c = (c_1, c_2) \in \mathbb{R}^2$, **um círculo é o conjunto C de pontos $(x, y) \in \mathbb{R}^2$ a distância r do centro c** . Mais precisamente

$$C = \{(x, y) \in \mathbb{R}^2 : (x - c_1)^2 + (y - c_2)^2 = r^2\}.$$

- **Tese:** qualquer definição que expresse o queremos dizer por círculo é **equivalente a essa**.
Exemplo: Um círculo é um espaço homeomorfo ao conjunto $\{(x, y) \in \mathbb{R}^2 : x^2 + y^2 = 1\}$
- O mesmo raciocínio vale para a Tese de Church-Turing: Uma MT é a **definição formal** do conceito intuitivo de algoritmo.
- Portanto, nesta leitura da tese:
 - Se **existe** uma Máquina de Turing que sempre para para decidir um problema, então **por definição existe um algoritmo** para o problema.
 - Se **não existe** tal máquina, então **por definição não existe algoritmo**.

Interpretação matemática da TCT

- Outras definições surgiram, mas são equivalentes a Máquinas de Turing:
e.g., Cálculo λ , funções μ -recursivas, etc
- *Hipercomputação*: Área que estuda definições de modelos mais poderosos que MTs
 $\Rightarrow$ O problema é que estas definições sempre acabam tendo passos “mágicos”.

“Computabilidade mecanicamente definida foi estabelecida por Alan Turing sem dúvida alguma.”
— Kurt Gödel

A TCT como afirmação empiricamente verificável

TESE DE CHURCH-TURING: *Se um problema computacional pode ser resolvido por algum dispositivo fisicamente realizável, então ele pode ser resolvido por uma Máquina de Turing*

Vantagem desta leitura da tese: há um critério para a refutação, caso a tese se revele falsa:

- Assim, como em qualquer hipótese empírica, a Tese de Church-Turing deve ser rejeitada caso seja refutada experimentalmente (e.g., um dispositivo que resolva, consistentemente, instâncias do problema da parada). **Não existe nada nem próximo a isso.**

Razão importante sustentando a Tese:

- Circuitos quânticos são equivalentes a Máquinas de Turing
- Vários modelos de física, química, biologia molecular são equivalentes a MTs
- *Hipercomputação*: modelos que resolvem problemas não computáveis tipicamente são variações de computação analógica:
 - Armazenamento de dados com uma quantidade infinita de informação (e.g., o número π com todos os seus infinitos dígitos)
 - Recuperação destes dados sem erros.
 - Existem obstáculos para estes modelos impostos pela física teórica
Exemplo: *limitante de Bekenstein*, que impõe um limite à quantidade de informação que uma região finita do espaço pode conter.

Interpretando incorretamente a Tese de Church-Turing

Ideia presente no artigo de Alan Turing de 1950: *Cérebros humanos, sendo “um tipo de máquina”, podem ser simulados por Máquinas de Turing.*

- Note, termos contemporâneos, a segunda leitura da TCT implica essa ideia

Antes de qualquer coisa, importante observar fatos triviais que a TCT não afirma:

- Não afirma que sabemos como realizar tal simulação: ainda não compreendemos totalmente o funcionamento do cérebro. Basta supor que o cérebro respeite as leis da física.
- Não implica que usar Máquinas de Turing seja a melhor estratégia para a construção de sistemas de inteligência artificial ou que o cérebro tenha arquitetura semelhante à dos computadores atuais. (ponto completamente irrelevante)

Algo mais importante que a TCT não afirma

- A TCT não trata de conceitos como *mente* ou *consciência*.
- Esses temas pertencem à filosofia da mente, que inclui várias posições a respeito disso: dualismo, materialismo, funcionalismo, hilomorfismo, etc.