



UNIVERSIDADE DE SÃO PAULO  
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO  
Departamento de Ciências de Computação

# SCC-505 - Capítulo 4

## Máquinas de Turing e a Teoria da Computabilidade

João Luís Garcia Rosa<sup>1</sup>

<sup>1</sup>Departamento de Ciências de Computação  
Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo  
<http://www.icmc.usp.br/~joaoluis>

2012

# Sumário

- 1 A Máquina de Turing Universal
  - A Máquina de Turing e Funções Numéricas [4]
  - A Tese de Church-Turing
  - A Máquina Universal
- 2 Indecidibilidade
  - A Linguagem de Diagonalização
  - O Problema da Parada
  - Problemas Indecidíveis

# Sumário

- 1 A Máquina de Turing Universal
  - A Máquina de Turing e Funções Numéricas [4]
  - A Tese de Church-Turing
  - A Máquina Universal
- 2 Indecidibilidade
  - A Linguagem de Diagonalização
  - O Problema da Parada
  - Problemas Indecidíveis

# Algoritmo

- Um **algoritmo**<sup>1</sup> é um conjunto finito de instruções que, se seguido, realiza uma determinada tarefa. Deve satisfazer os seguintes critérios [3]:
  - 1 **Entrada:** Zero ou mais quantidades são supridas;
  - 2 **Saída:** No mínimo uma quantidade é produzida;
  - 3 **Clareza:** Cada instrução é clara e não ambígua;
  - 4 **Finitude:** Se o algoritmo for percorrido passo a passo (*trace*), então em todos os casos, o algoritmo termina depois de um número finito de passos;
  - 5 **Efetividade:** Toda instrução deve ser muito básica, de tal forma que possa ser realizada, em princípio, por uma pessoa usando apenas lápis e papel. Não é suficiente que cada operação seja definida como no critério 3; ela também tem de ser factível.

---

<sup>1</sup> A palavra "algoritmo" vem do nome de um matemático persa (825 d.C.), [Abu Ja'far Mohammed ibn Musa al Khowarizmi](#).

## Funções Numéricas: Notação Unária

- Sempre se pensou apenas nas máquinas de Turing como aceitadores de linguagem.
- É também importante usar estas máquinas como dispositivos que computam funções numéricas, isto é, que mapeiam  $\mathbb{N}^k \rightarrow \mathbb{N}$ .
- Pretende-se codificar o conjunto dos números naturais na notação unária.
- Então o código para 0 é 1, o código para 1 é 11, 2 é 111, 3 é 1111, etc.
- Escreve-se  $n^u$  para simbolizar o  $n$  codificado em unário.

# Funções Numéricas: Notação Unária

- **Definição:** Uma máquina de Turing  $M$  computa uma função  $\varphi_M^{(k)}$  de aridade  $k$  como se segue.
  - Na entrada  $(n_1, \dots, n_k)$ ,  $n_1, \dots, n_k$  são colocados na fita de  $M$  em unário, separados por brancos simples.
  - A cabeça de  $M$  é colocada sobre o 1 mais a esquerda de  $n_1^u$ , e o controle de estados finitos de  $M$  é colocado em  $q_0$ .
  - Em outras palavras,  $M$  tem DI inicial

$$q_0 n_1^u B n_2^u B \dots B n_k^u$$

- Se e quando  $M$  terminar o processamento, os 1's na fita são contados e seu total é o valor de  $\varphi_M^{(k)}(n_1, \dots, n_k)$ .
- Se  $M$  nunca para, diz-se que  $\varphi_M^{(k)}(n_1, \dots, n_k)$  é indefinido.
- Refere-se a  $\varphi_M^{(k)}$  como a ( $k$ -ésima) semântica de  $M$ .

# Funções Numéricas: Notação Unária

- **Exemplo:** A máquina de Turing sem nenhuma quintupla (isto é, ela para qualquer que seja a DI) computa a função sucessora  $\varphi^{(1)}(n) = s(n) = n + 1$ .
  - Entretanto, deve-se checar que, como uma calculadora para uma função de duas variáveis, ela computa a função  $\varphi^{(2)}(x, y) = x + y + 2$ .
- **Exemplo:** Considere a seguinte máquina de Turing.

$$(q_0 \ 1 \ q_1 \ 1 \ R)$$
$$(q_1 \ 1 \ q_0 \ 1 \ R)$$
$$(q_1 \ B \ q_1 \ B \ R)$$

Esta máquina de Turing computa a seguinte função de uma variável

$$\begin{aligned}\varphi_M^{(1)}(n) &= n + 1, \text{ se } n \text{ é ímpar;} \\ &= \perp, \text{ caso contrário}\end{aligned}$$

# Sumário

- 1 A Máquina de Turing Universal
  - A Máquina de Turing e Funções Numéricas [4]
  - A Tese de Church-Turing
  - A Máquina Universal
- 2 Indecidibilidade
  - A Linguagem de Diagonalização
  - O Problema da Parada
  - Problemas Indecidíveis



# A Tese de Church

- Depois de meio século de estudos, achou-se que as funções computáveis são invariantes ao longo de uma grande faixa de diferentes mecanismos de definição - cada sistema formal estudado foi mostrado computar ou todas as funções Turing-computáveis ou algum subconjunto delas.
- Isto levou o lógico matemático americano Alonzo Church a formular a **tese de Church**:
  - **todos os mecanismos de computação definem a mesma classe de funções computáveis.**
- Um conceito familiar em ciência da computação é que quando um computador,  $M$ , for suficientemente de “propósito geral”, um programa escrito em qualquer outra máquina pode ser recodificado para fornecer um programa para  $M$  que computará a mesma função.

# O Resultado de Turing

- Apresenta-se um resultado de 1936 devido ao matemático inglês A. M. Turing que antecipa o computador digital por quase uma década e ainda carrega a ideia inicial da sentença anterior: ou seja, que existe uma máquina de Turing  $U$  que é universal, no sentido de que o comportamento de qualquer outra máquina  $M$  pode ser codificado como uma cadeia  $e(M)$  tal que  $U$  processará qualquer cadeia da forma  $(e(M), w)$  da forma como  $w$  seria processado por  $M$ ; diagramaticamente, significa

$$\text{se } w \Rightarrow_M^* w' \text{ então } (e(M), w) \Rightarrow_U^* w'$$

# Sumário

- 1 A Máquina de Turing Universal
  - A Máquina de Turing e Funções Numéricas [4]
  - A Tese de Church-Turing
  - A Máquina Universal
- 2 Indecidibilidade
  - A Linguagem de Diagonalização
  - O Problema da Parada
  - Problemas Indecidíveis

# A Máquina Universal

- Suponha uma máquina com  $p$  cabeças percorrendo uma única fita na qual são impressos símbolos do alfabeto  $\Sigma'$ .
- A qualquer tempo a caixa de controle estará no estado  $q$  de  $Q$  e receberá como entrada os  $p$  símbolos percorridos pelas suas cabeças, isto é, um elemento de  $(\Sigma')^p$ .
- A saída da unidade de controle é um elemento de  $((\Sigma')^p \times M) \cup \{pare\}$ , onde  $M = I \mid I$  é uma instrução possível às cabeças para mover ao máximo a distância 1.
- Então, a máquina de Turing é especificada pelas quintuplas

$$q_i \ x_j \ q_l \ x_k \ I$$

# A Máquina Universal

- com a única diferença de que os  $x$ 's e os  $l$ 's são “vetores,” e que se deve empregar uma convenção para resolver conflitos se duas cabeças tentam imprimir símbolos diferentes num único “quadrado.”
- Uma computação de tal máquina começa com a atribuição de um estado à unidade de controle e a atribuição das posições iniciais para as cabeças.
- Como de costume, é assumido que a fita tem no máximo finitamente muitos quadrados não brancos.
- Então a computação prossegue normalmente, parando quando e apenas quando nenhuma quintupla começando com  $q_i x_j$  é aplicável.

# Máquinas de Turing Multicabeças

- Diz-se que uma cadeia  $w$  é aceita por uma máquina de Turing de 2 cabeças e 2 fitas se, quando  $w$  é escrita na fita 1 e a cabeça 1 percorre o símbolo mais a esquerda de  $w$  no estado  $q_0$ , e a fita 2 está inicialmente em branco,  $M$  finalmente para em seu estado de aceitação  $q_a$ .

# Máquinas de Turing Não Determinísticas

- **Teorema:** Seja  $L$  uma linguagem que possa ser reconhecida por uma máquina de Turing de  $k$  cabeças e  $k$  fitas. Então  $L = T(M)$  para alguma máquina de Turing de 1 cabeça e 1 fita.
- **Definição:** Uma máquina  $M$  é chamada de máquina de Turing **não determinística** se ela incluir quintuplas que especifiquem movimentos múltiplos para um dado par estado/símbolo, isto é, se ela é definida como na Definição de Máquina de Turing do capítulo 3, exceto que agora  $\delta$  mapeia  $Q \times \Sigma'$  a subconjuntos de  $Q \times \Sigma' \times \{L, S, R\}$ .

# Máquinas de Turing Não Determinísticas

- Diz-se que uma máquina de Turing não determinística  $M$  aceita uma cadeia  $w$  se existir alguma cadeia de transições da máquina na entrada  $w$  que alcança uma DI de parada que inclui o estado de aceitação  $q_a$ .
- A definição de aceitação está de acordo com a definição de aceitação não determinística dada para aceitadores de estados finitos não determinísticos.
- O próximo resultado mostra que, assim como com os AFNs, o não determinismo não adiciona potência computacional nova ao modelo determinístico original.
- (Isto não é verdadeiro para todas as máquinas que foram estudadas - autômatos de pilha não determinísticos (capítulo 2) são mais potentes que os autômatos de pilha determinísticos.)



# Máquinas de Turing Não Determinísticas

- **Teorema:** Seja  $M$  uma máquina de Turing não determinística que aceita  $L$ . Então existe uma máquina determinística  $M'$  que também aceita  $L$ .
  
- **Teorema:** Uma linguagem é do tipo 0 se e somente se ela for o conjunto de aceitação de alguma máquina de Turing.

# Sumário

- 1 A Máquina de Turing Universal
  - A Máquina de Turing e Funções Numéricas [4]
  - A Tese de Church-Turing
  - A Máquina Universal
- 2 Indecidibilidade
  - A Linguagem de Diagonalização
  - O Problema da Parada
  - Problemas Indecidíveis

# Uma Linguagem que não é RE

- É possível achar numa enumeração de máquinas de Turing a máquina  $M_i$ , a “ $i$ -ésima máquina de Turing”.
- Imagine que seu código binário seja  $w_i$ .
- Muitos inteiros não correspondem a nenhuma máquina de Turing.
- Se  $w_i$  não é um código de máquina de Turing válido,  $M_i$  será a máquina de Turing com um estado e nenhuma transição.
- Ou seja,  $M_i$  é uma máquina de Turing que para para qualquer entrada.
- Portanto,  $T(M_i)$  é  $\emptyset$  se  $w_i$  falha ao ser um código de uma máquina de Turing válida.

# Uma Linguagem que não é RE

- **Definição:** A linguagem  $L_d$ , a **linguagem de diagonalização**, é o conjunto de cadeias  $w_i$  tal que  $w_i$  não está em  $T(M_i)$ .
- Ou seja,  $L_d$  consiste de todas as cadeias  $w$  tal que a máquina de Turing  $M$  cujo código é  $w$  não aceita quando recebe  $w$  como entrada.
- A razão do nome linguagem de “diagonalização” pode ser entendida através da figura 3.
- A tabela informa, para todo  $i$  (linha) e  $j$  (coluna), se a máquina de Turing  $M_i$  aceita a cadeia de entrada  $w_j$ : 1 significa “sim” e 0 significa “não”.
- Pode-se pensar na  $i$ -ésima linha como o *vetor característico* para a linguagem  $T(M_i)$ ; ou seja, os 1's nesta linha indicam as cadeias que são elementos desta linguagem.

# Uma Linguagem que não é RE

		$w_i$						
		1	2	3	4	.	.	.
$M_i$	1	0	1	1	0	.	.	.
	2	1	1	0	0	.	.	.
	3	0	0	1	1	.	.	.
	4	0	1	0	1	.	.	.
	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.
	.	.	.	.	.	.	.	.

Figure : Tabela que representa aceitação de cadeias por máquinas de Turing.

# Uma Linguagem que não é RE

- Os valores na diagonal dizem se  $M_i$  aceita  $w_i$ .
- Para construir a  $L_d$ , complementa-se a diagonal.
- Por exemplo, se a figura 3 for a tabela correta, então a diagonal complementada seria 1, 0, 0, 0, ...
- Portanto,  $L_d$  contém  $w_1 = \lambda$ , não contém  $w_2$  a  $w_4$ , que são 0, 1 e 00, e assim por diante.
- O truque de complementar a diagonal para construir o vetor característico de uma linguagem que não pode ser a linguagem que aparece em nenhuma linha é chamado de **diagonalização**.

# Uma Linguagem que não é RE

- Isto funciona porque o complemento da diagonal é ele próprio um vetor característico que descreve a pertinência em alguma linguagem, a  $L_d$ .
- Este vetor característico discorda em alguma coluna com toda linha da tabela.
- Portanto, o complemento da diagonal não pode ser o vetor característico de nenhuma máquina de Turing.
- **Teorema:**  $L_d$  não é uma linguagem recursivamente enumerável (RE). Ou seja, não há nenhuma máquina de Turing que aceite  $L_d$ .

# Uma Linguagem que não é RE

- **PROVA:** Suponha que  $L_d$  seja  $T(M)$  para alguma máquina de Turing  $M$ . Como  $L_d$  é uma linguagem sobre o alfabeto  $\{0, 1\}$ ,  $M$  estaria na lista das máquinas de Turing, já que esta lista inclui todas as máquinas de Turing com alfabeto de entrada  $\{0, 1\}$ . Portanto, há pelo menos um código para  $M$ , por exemplo,  $M = M_i$ . Agora será que  $w_i$  está em  $L_d$ ?
  - Se  $w_i$  está em  $L_d$ , então  $M_i$  aceita  $w_i$ . Mas então, pela definição de  $L_d$ ,  $w_i$  não está em  $L_d$ , porque  $L_d$  contém apenas aqueles  $w_j$  tal que  $M_j$  **não** aceita  $w_j$ .
  - Similarmente, se  $w_i$  não está em  $L_d$ , então  $M_i$  não aceita  $w_i$ . Portanto, por definição de  $L_d$ ,  $w_i$  **está** em  $L_d$ .

Como  $w_i$  não pode estar e não estar em  $L_d$  ao mesmo tempo, há uma contradição na suposição de que  $M$  existe. Ou seja,  $L_d$  não é uma linguagem recursivamente enumerável. ♦



# Sumário

- 1 A Máquina de Turing Universal
  - A Máquina de Turing e Funções Numéricas [4]
  - A Tese de Church-Turing
  - A Máquina Universal
- 2 Indecidibilidade
  - A Linguagem de Diagonalização
  - O Problema da Parada
  - Problemas Indecidíveis

# O Problema da Parada

- Tendo demonstrado a potência e flexibilidade de computação efetiva, como capturadas nas máquinas de Turing, mostrar-se-á agora que mesmo esta potência e flexibilidade têm limites.
- Uma questão muito prática para o cientista de computação é: este programa processará dados da forma pretendida?
- Mesmo mais fundamental, talvez, do que determinar se ou não um programa está correto é dizer se ou não ele terminará, fornecendo a resposta correta.
- Reescrevendo a última questão em termos das máquinas de Turing, então, tem-se o seguinte:

# O Problema da Parada

## Definição

**O Problema da Parada:** Dada uma máquina de Turing  $M_n$  com semântica  $\varphi_n$  e uma cadeia de dados  $w$ ,  $\varphi_n(w)$  é definida? Isto é,  $M_n$  sempre para se iniciada no estado  $q_0$  percorrendo o quadrado mais a esquerda de  $w$ ?

# O Problema da Parada

- Seria muito bom se fosse possível achar algum testador de parada universal que, quando dado um número  $n$  e uma cadeia  $w$ , poderia efetivamente dizer se ou não a máquina  $M_n$  sempre parará se sua entrada de dados for  $w$ .
- Entretanto, o teorema 32 dirá que tal máquina não existe.
- Antes de mostrar que nenhum procedimento efetivo pode resolver o problema da parada, veja por que o procedimento óbvio falha.
- Vai-se pegar a máquina universal  $U$  e “rodá-la” na fita  $(e(M_n), w)$ .
- Quando ela para, o controle é transferido para uma sub-rotina que imprime um 1 para significar que a computação de  $M_n$  em  $w$  parou realmente.

# O Problema da Parada

- Mas quando o controle pode ser transferido para uma sub-rotina que imprime um 0 para significar que  $M_n$  nunca para com os dados  $w$ ?
- Depois de muitas simulações por  $U$ , é possível concluir que  $M_n$  **nunca** parará em  $w$ , ou que, depois de muito tempo, as computações pararão?
- Esta abordagem claramente falha.
- Mas para provar que todas as abordagens que tentam decidir a terminação algorítmicamente necessariamente falharão é tarefa bem menos óbvia, e para isso deve-se usar um argumento baseado no argumento diagonal usado por Cantor para mostrar que nenhuma enumeração poderia incluir todos os números reais.

## Argumento diagonal de Cantor

$E_0 =$	m	m	m	m	m	m	m	m	m	m	m	m	...
$E_1 =$	w	w	w	w	w	w	w	w	w	w	w	w	...
$E_2 =$	m	w	m	w	m	w	m	w	m	w	m	w	...
$E_3 =$	w	m	w	m	w	m	w	m	w	m	w	...	
$E_4 =$	w	m	m	w	w	m	w	m	w	m	w	...	
$E_5 =$	m	w	m	w	w	m	w	m	w	m	w	...	
$E_6 =$	m	w	m	w	w	m	w	w	m	w	m	...	
$E_7 =$	w	m	m	w	m	w	m	w	m	w	m	...	
$E_8 =$	m	m	w	m	w	m	w	m	w	m	w	...	
$E_9 =$	w	m	w	m	m	w	w	m	w	w	m	...	
$E_{10} =$	w	w	m	w	m	w	m	w	m	m	w	...	
$E_{11} =$	m	w	m	w	w	m	w	m	m	w	m	...	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	
$E_u \neq$	w	m	w	w	m	w	m	m	m	m	w	...	

Figure : Uma ilustração do argumento diagonal de Cantor para a existência de conjuntos incontáveis. A sequência final ( $E_u$ ) não pode ocorrer em nenhum outro lugar na listagem acima [7].

# O Problema da Parada

## Cantor

O matemático russo Georg Ferdinand Ludwig Philipp Cantor provou que os números reais não são contáveis em 1874. Ele produziu seu famoso “argumento diagonal” em 1890, que deu uma segunda prova, mais enfática e interessante, de que os números reais não são contáveis.

- Para antever o estudo do problema da parada, primeiro será fornecido um resultado que relaciona a enumeração das máquinas de Turing com a discussão de funções numéricas.
- Lembre-se que uma função Turing-computável  $\psi$  é total se ela retorna uma saída para toda entrada.

# O Problema da Parada

- **Teorema:** Não existe nenhuma função Turing-computável total  $g$  que enumera as funções Turing-computáveis totais no seguinte sentido: a função Turing-computável  $\psi$  é total se e somente se  $\psi$  for igual à função  $\varphi_{g(n)}$  computada por  $M_{g(n)}$  para algum  $n$ .
- **Teorema: (A Insolubilidade do Problema da Parada).** Considere a função  $pare : \mathbb{N} \rightarrow \mathbb{N}$  definida como:

$$pare(x) = \begin{cases} 1 & \text{se } M_x \text{ para na entrada } x \\ 0 & \text{se } M_x \text{ nunca para em } x \end{cases}$$

Então  $pare$  não é Turing-computável.



# O Problema da Parada

- Antes de provar este resultado, far-se-á algumas observações.
  - Primeiro, note que *pare* é uma função total com certeza.
  - Ela tem um valor definido para toda entrada.
  - O que se está tentando mostrar é que *pare* não é computável: não existe nenhum método (formalmente expresso como uma máquina de Turing) para calcular os valores de *pare*.
  - Note também que o Teorema 32 é muito conservador.
  - Apenas considera-se a não computabilidade de Turing.
  - De fato, se a tese de Church estiver subscrita, *pare* não é computável por nenhum algoritmo especificado em qualquer linguagem de programação.

# O Problema da Parada

- **PROVA DO TEOREMA:** Suponha que se construa uma matriz  $\mathbb{N} \times \mathbb{N}$ , estabelecendo a entrada  $(i, j)$  para  $\downarrow$  se a computação de  $\varphi_i(j)$  termina, enquanto estabelece-se a  $\uparrow$  caso contrário. Para construir uma função  $\varphi$  não no conjunto de  $\varphi$ 's adota-se o argumento diagonal de Cantor: movendo para baixo na diagonal, “inverte-se as setas,” dando a  $\varphi(x)$  o comportamento da parada oposto a  $\varphi_x(x)$ :

$$\varphi(x) = \begin{cases} 1 & \text{se } \varphi_x(x) \text{ não é definido} \\ \perp & \text{se } \varphi_x(x) \text{ é definido} \end{cases}$$

Mas isto apenas diz que

$$\varphi(x) = \begin{cases} 1 & \text{se } \textit{pare}(x) = 0 \\ \perp & \text{se } \textit{pare}(x) = 1 \end{cases}$$

# O Problema da Parada

- Agora se *pare* for computável, existe certamente uma máquina de Turing que computa  $\varphi$ .
- Entretanto, o argumento diagonal garante que  $\varphi$  **não é computável**, se  $\varphi$  for  $\varphi_j$  para algum  $j$  ter-se-ia  $\varphi_j(j) = 1$  se  $\varphi_j(j) = \perp$  enquanto  $\varphi_j(j) = \perp$  se  $\varphi_j(j) = 1$  - uma contradição.
- Assim nenhum método de cálculo para *pare* pode existir. ♦
- Portanto, exibiu-se um problema geral em ciência da computação, natural, interessante e valioso, que não tem solução algorítmica. (Este resultado é extremamente fundamental e está fortemente relacionado com o resultado famoso do lógico austríaco Kurt Gödel da incompletude das teorias formais da aritmética.)

# Teorema da Incompletude de Gödel

## Teorema da Incompletude de Gödel

O matemático alemão David Hilbert estabeleceu em 1900 que os matemáticos deveriam buscar expressar a matemática na forma de um sistema formal, consistente, completo e decidível. Em 1931, Gödel provou que o ideal de Hilbert era impossível de satisfazer, mesmo no caso da aritmética simples. Este resultado é conhecido como primeiro teorema da incompletude de Gödel. Mais tarde, em 1939, Turing e Church mostraram independentemente que nenhum sistema formal consistente da aritmética é decidível, nem mesmo a lógica de predicados de primeira ordem, considerado mais fraco, é decidível [1].

# Sumário

- 1 A Máquina de Turing Universal
  - A Máquina de Turing e Funções Numéricas [4]
  - A Tese de Church-Turing
  - A Máquina Universal
- 2 Indecidibilidade
  - A Linguagem de Diagonalização
  - O Problema da Parada
  - Problemas Indecidíveis

# Problemas Indecidíveis

- **Definição:** Um conjunto  $S \subset \mathbb{N}$  é **decidível** se a pertinência em  $S$  pode ser determinada algoritmicamente. Isto é,  $S$  é decidível (ou recursivo ou solúvel) se a função característica de  $S$ ,

$$\chi_S(x) = \begin{cases} 1 & \text{se } x \in S \\ 0 & \text{se } x \notin S \end{cases}$$

é Turing-computável.

- Se  $S$  não é decidível, diz-se que  $S$  é **indecidível**, ou insolúvel ou não recursivo ou, em casos que requer ênfase considerável, recursivamente insolúvel.

# Problemas Indecidíveis

## Algumas linguagens não são decidíveis

Lembre-se da enumeração feita com as máquinas de Turing. Como existem incontáveis linguagens e somente um número contável de máquinas de Turing (enumeração), conclui-se que algumas linguagens não são decidíveis por máquinas de Turing, nem mesmo reconhecidas por máquinas de Turing.

# Problemas Indecidíveis

- Tem-se um exemplo de um conjunto indecidível: o conjunto

$$K = \{ n \mid \varphi_n(n) \text{ retorna um valor} \}$$

é recursivamente insolúvel. O próximo resultado mostra como mapear a indecidibilidade de  $K$  na teoria da linguagem.

- **Definição:** Seja  $G$  uma gramática do tipo 0 sobre algum alfabeto  $V \cup \Sigma$ . O **problema de dedução** para  $G$  é o problema de determinar, dadas as cadeias arbitrárias  $x, y$  em  $(V \cup \Sigma)^*$ , se  $x \Rightarrow^* y$  em  $G$ .
- **Teorema:** O problema de dedução para gramáticas do tipo 0 é indecidível.



# Problemas Indecidíveis

- Contrastando com o resultado anterior, tem-se o seguinte:
- **Teorema:** O problema de dedução para gramáticas sensíveis ao contexto é decidível.
- **Corolário:** O problema de dedução para gramáticas livres de contexto é decidível.
- **Definição:** Dada uma classe de gramáticas  $C$ , o **problema de esvaziamento** para  $C$  é o problema de determinar para  $G$  arbitrária em  $C$ , se a linguagem  $L(G)$  é vazia.
- **Teorema:** O problema de esvaziamento para gramáticas livres de contexto é decidível.

# Linguagens Recursivas

- Linguagens recursivamente enumeráveis (RE) são aceitas (reconhecidas) por máquinas de Turing.
- Linguagens RE podem ser agrupadas em duas classes:
  - 1 Classe 1 (**linguagens recursivas**): cada linguagem  $L$  nesta classe tem uma máquina de Turing (pensada como um algoritmo) que não apenas aceita cadeias de  $L$ , como também indica quais cadeias não estão em  $L$  através da **parada**.
  - 2 Classe 2 (**RE mas não recursivas**): cada linguagem  $L$  nesta classe tem uma máquina de Turing (não pensada como um algoritmo) que aceita cadeias de  $L$ , mas pode **não parar** quando uma cadeia de entrada não está em  $L$ .

# Linguagens Recursivas

- **Definição:** Formalmente, uma linguagem  $L$  é **recursiva** se  $L = T(M)$  para alguma máquina de Turing  $M$  tal que:
  - 1 Se  $w \in L$ , então  $M$  aceita (e portanto para no estado de aceitação).
  - 2 Se  $w \notin L$ , então  $M$  rejeita (para num estado de não aceitação).
- Uma máquina de Turing deste tipo corresponde à noção formal de algoritmo.
- Uma dada linguagem  $L$ , pensada como um problema, é chamada de **decidível** se  $L$  é uma linguagem recursiva; e **indecidível** caso contrário.
- A existência ou não existência de um algoritmo para resolver o problema (isto é, o problema é decidível ou indecidível) é mais importante do que a existência de uma máquina de Turing para resolver o problema.

# Linguagens Recursivas

## Problemas decidíveis

A classe de **problemas decidíveis** é equivalente à classe das linguagens recursivas.

## Problemas parcialmente decidíveis

A classe dos **problemas parcialmente decidíveis** é equivalente à classe das linguagens recursivamente enumeráveis.

# Decidibilidade e Aceitabilidade

## Decidibilidade e Aceitabilidade

A noção de decidibilidade é mais restrita que a de aceitabilidade (ser reconhecível), uma vez que neste último caso, é permitido que a máquina de Turing nunca pare.

Decidibilidade = Algoritmo.

Aceitabilidade = Procedimento.

Reconhecedores são mais poderosos que decididores.

# A Linguagem Universal

- **Definição:** A **linguagem universal**  $L_U$  é o conjunto de cadeias binárias, que codificam um par  $(M, w)$ , onde  $M$  é uma máquina de Turing com o alfabeto de entrada binário e  $w$  é uma cadeia em  $(0 + 1)^*$ , tal que  $w$  está em  $T(M)$ . Ou seja,  $L_U$  é o conjunto de cadeias que representam uma máquina de Turing e uma entrada aceita por esta máquina de Turing. A **máquina de Turing universal**  $U$  processa  $L_U$ , ou seja,  $L_U = T(U)$ .
- $L_U$  não é recursiva, apesar de ser uma linguagem RE.

# Linguagens Recursivas, RE e não-RE

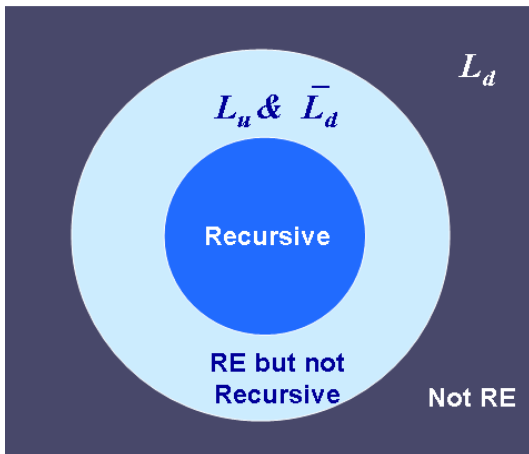


Figure : Relacionamento entre linguagens recursivas, RE e não-RE.

# Bibliografia I



[1] Copeland, J.

*Biography of Turing*, 2000.

[http://www.alanturing.net/turing\\_archive/pages/Reference%20Articles/Bio%20of%20Alan%20Turing.html](http://www.alanturing.net/turing_archive/pages/Reference%20Articles/Bio%20of%20Alan%20Turing.html).



[2] Hopcroft, J. E., Ullman, J. D.

*Formal Languages and Their Relation to Automata*.  
Addison-Wesley Publishing Company, 1969.






[3] Horowitz, E., Sahni, S., and Rajasekaran, S.

*Computer Algorithms*.

Computer Science Press, 1998.



# Bibliografia II

-  [4] Moll, R. N., Arbib, M. A., and Kfoury, A. J.  
*An Introduction to Formal Language Theory.*  
Springer-Verlag, 1988.
-  [5] Rosa, J. L. G.  
*Linguagens Formais e Autômatos.*  
Editora LTC. Rio de Janeiro, 2010.
-  [6] Turing, A.M.  
On Computable Numbers, with an Application to the  
Entscheidungsproblem.  
*Proceedings of the London Mathematical Society*, 2 42:  
230-65, 1937.

# Bibliografia III



[7] Wikipedia.

[http://en.wikipedia.org/wiki/Cantor's\\_diagonal\\_argument](http://en.wikipedia.org/wiki/Cantor's_diagonal_argument)