Teoria da Computação

Máquinas de Turing e Computabilidade

Mirtha Lina Fernández Venero
mirtha.lina@ufabc.edu.br

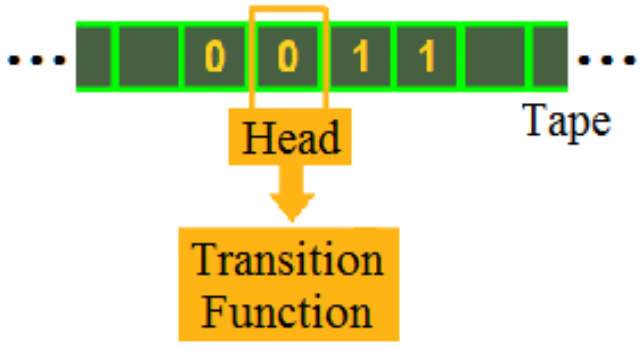
outubro 2017

Sumário

Universidade Federal do ABC

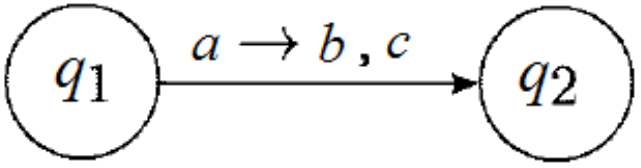
A Máquina de Turing

Proposta por Alan Turing em 1936



Definição: $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$

- Q conjunto *finito* de estados
- Σ alfabeto de entrada
- Γ alfabeto da fita tal que $\Sigma \subseteq \Gamma$ e $\sqcup \in \Gamma$ onde $\sqcup$ denota uma célula da fita em branco
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{E, D\}$ função de transição de estados



- F conjunto de estados finais

A Máquina de Turing - Características

- A fita é infinita com todas as células em branco exceto as da cadeia de entrada
- A cabeça da máquina inicialmente está posicionada no primeiro símbolo. Ela pode mover-se à direita ou à esquerda
- A máquina pode ler e também escrever sobre a fita
- **Configuração**: é representada como uqv onde q é o estado atual uv a cadeia na fita e a cabeça está posicionada sobre o primeiro símbolo de v
- **Movimento**: $xqay \vdash xbq'y$ se $\delta(q,a) = (q',b,D)$
  $xqay \vdash xq'by$ se $\delta(q,a) = (q',b,E)$
- Os estados de aceitação fazem efeito imediato

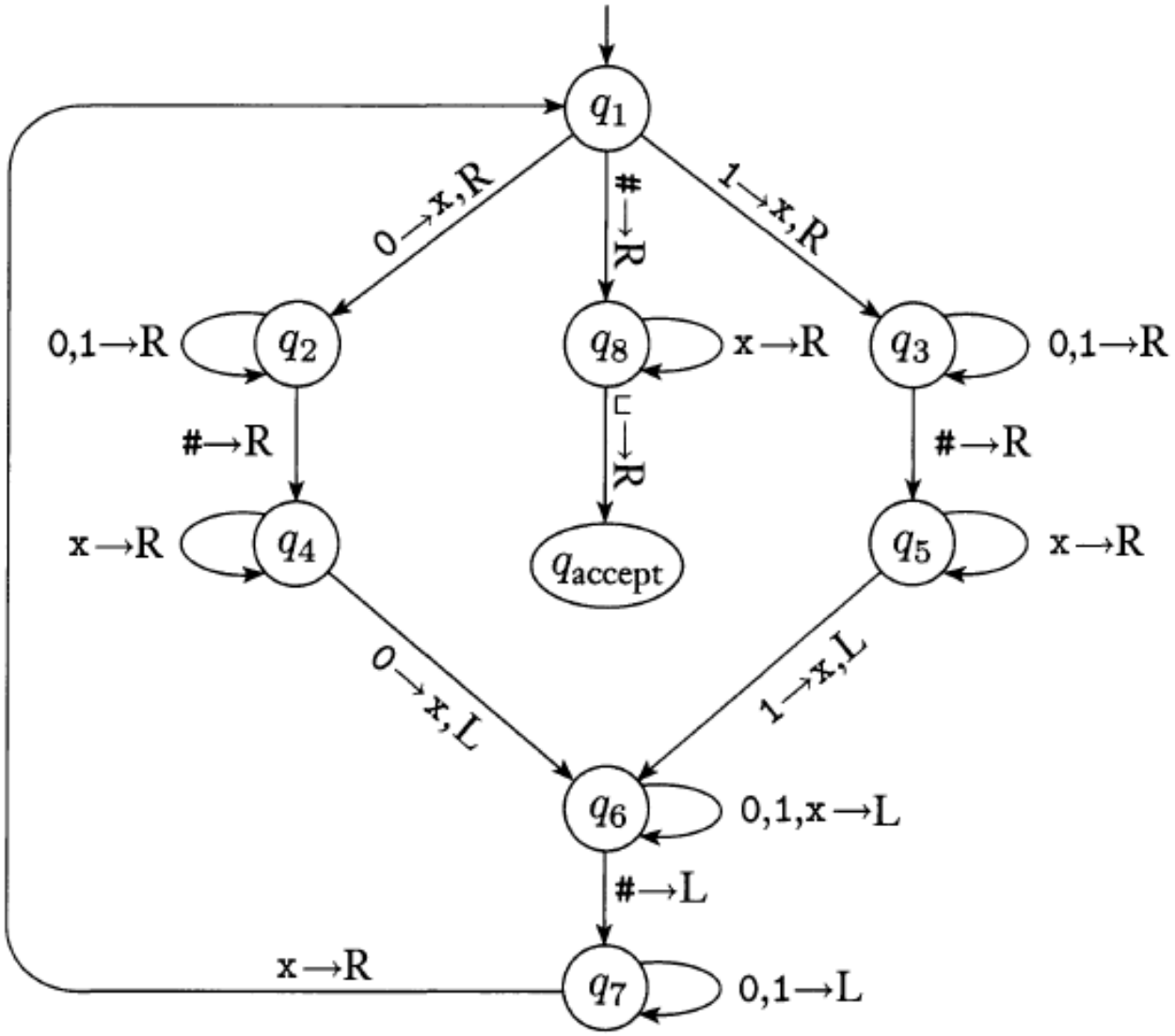
$$L(M) = \{\ w \in \Sigma^* \mid q_0 w \overset{*}{\vdash} xqy, q \in F\ \}$$

Exercícios: Escreva máquinas de Turing para

1. Incrementar um número binário
2. Aceitar a linguagem $\{ 0^n 1^n \mid n \geq 0 \}$
3. Aceitar a linguagem $\{ w \mid w \in (a+b)^*,\ w = \bar{w} \}$
4. Aceitar a linguagem $\{ w\#w \mid w \in (0+1)^* \}$

Universidade Federal do ABC

Exemplo de MT para a linguagem $\{ w\#w \mid w \in (0+1)^* \}$



Outras definições equivalentes

$$M = (Q, \Sigma, \Gamma, \delta, q_0)$$

- Não define estados finais e δ é uma função parcial
- Usa o reconhecimento por parada, i.e.

$$L(M) = \{\ w \in \Sigma^* \mid q_0 w \overset{*}{\vdash} xqay, a \in \Gamma, \nexists\ \delta(q, a) = (q', b, d)\ \}$$

Exercício: Escreva duas MTs para a linguagem uma com reconhecimento por estado final e outra com reconhecimento por parada para a seguinte linguagem $(a + b + c)^* - ab(a + b + c)^*$

Outras definições equivalentes

$M = (Q, \Sigma, \Gamma, \delta, q_0)$

- Não define estados finais e δ é uma função parcial
- Usa o reconhecimento por parada, i.e.

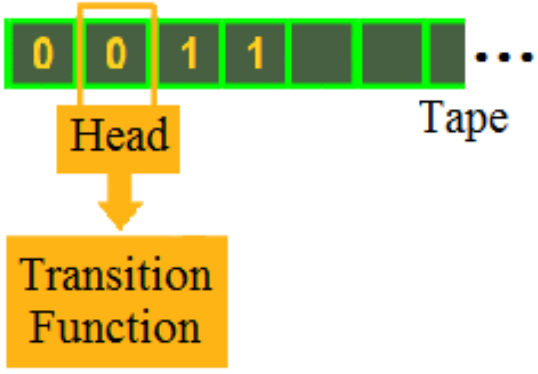
$$L(M) = \{\ w \in \Sigma^* \mid q_0 w \overset{*}{\vdash} xqay, a \in \Gamma, \nexists\ \delta(q, a) = (q', b, d)\ \}$$

Exercício: Escreva duas MTs para a linguagem uma com reconhecimento por estado final e outra com reconhecimento por parada para a seguinte linguagem $(a + b + c)^* - ab(a + b + c)^*$

Como converter uma MT por estado final em outra por parada? $M_p = (Q \cup \{l\}, \Sigma, \Gamma, \delta, q_0)$ onde o reconhecimento de toda cadeia que não pertence à linguagem entra no estado $l \in Q$ tal que $\forall a \in \Gamma, \delta(l, a) = (l, a, D)$

Outras definições equivalentes (Sipser)

$$M = (Q, \Sigma, \Gamma, \delta, q_0, q_{aceita}, q_{rejeita})$$

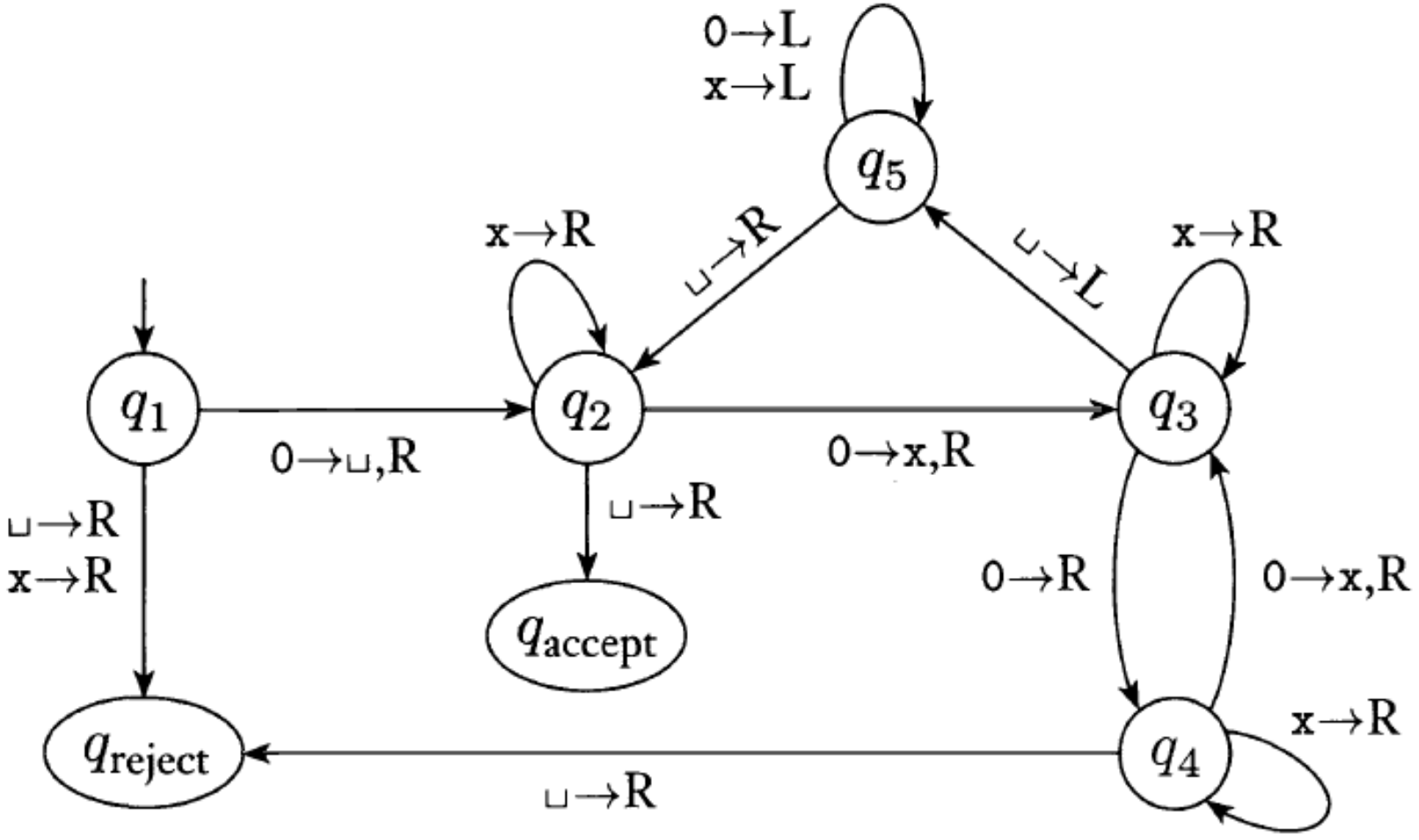


- A fita é infinita e a cadeia de entrada está escrita nas células mais à esquerda
- A cabeça da máquina inicialmente está posicionada no primeiro símbolo. Ela não pode-se movimentar à esquerda desse símbolo. Nesse caso fica na mesma posição
- Os estados de aceitação e rejeição ($q_{aceita} \neq q_{rejeita}$) fazem efeito imediato. Se não forem atingidos a computação continua para sempre

Universidade Federal do ABC

Exemplo de MT para a linguagem $\{0^{2^n} | n \geq 0\}$

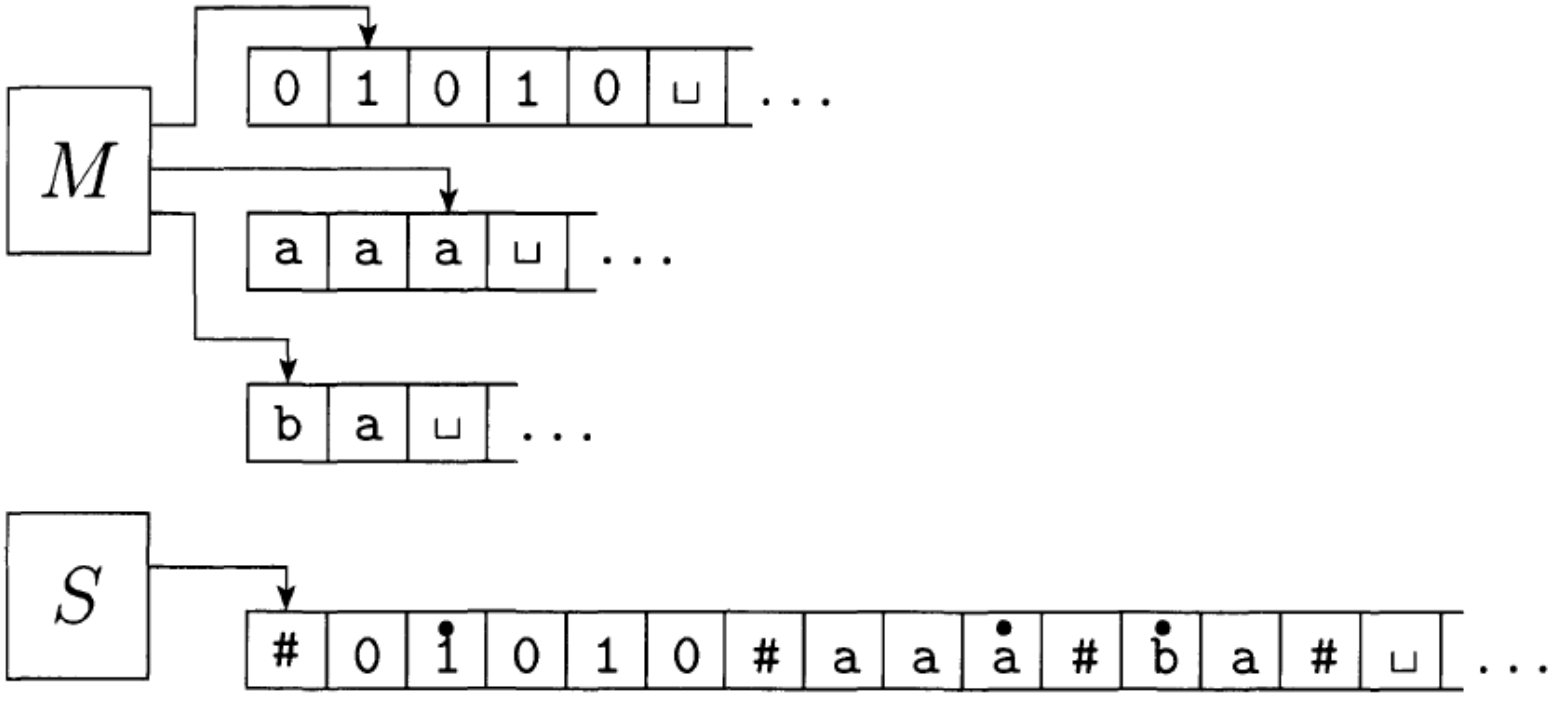
Ideia: Aceitar se há só um zero. Senão faça varreduras da cadeia apagando (com x) cada vez metade dos zeros até não ter nenhum.



Sumário

Máquina de Turing multifita

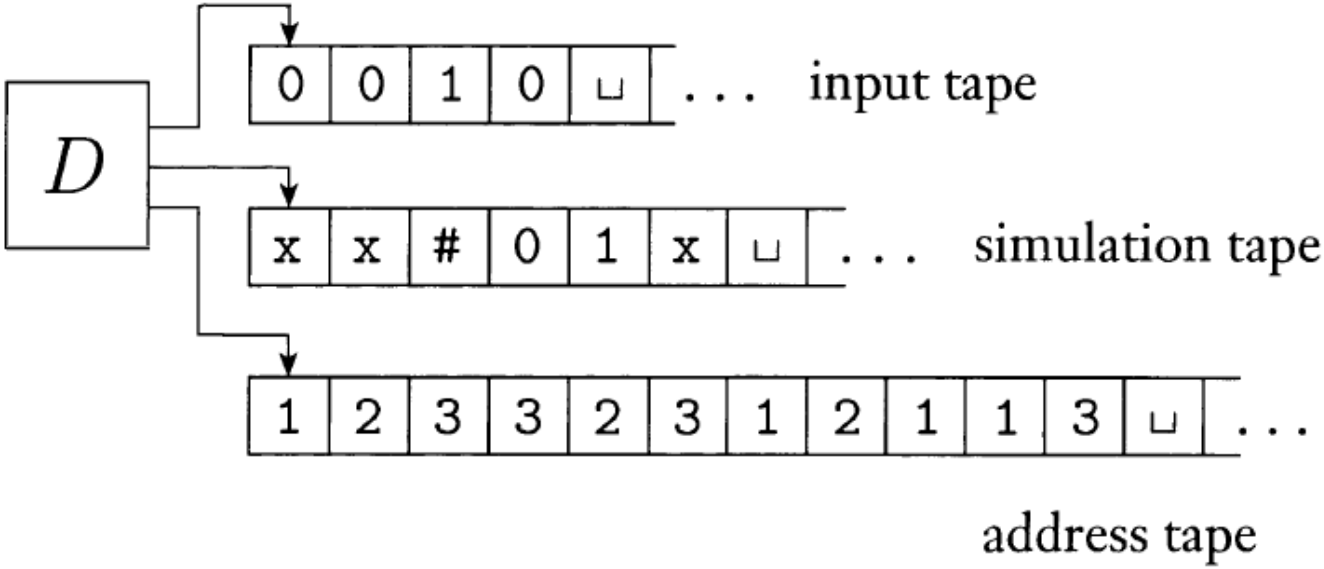
$$\delta_M : Q \times \Gamma^k \to Q \times \Gamma^k \times \{E, D, P\}^k$$



https://turingmachinesimulator.com/

Máquina de Turing não determinística

$$\delta_{ND} : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{E, D, P\})$$



Tese de Church – Turing : Tudo o que é intuitivamente computável também é computável usando uma Máquina de Turing*

- *"A man provided with paper, pencil, and rubber, and subject to strict discipline" (Turing 1948)*
- λ-cálculo tipado / funções recursivas (computáveis)
- gramáticas não restritivas / sistemas Post / sistemas L
- autômatos com duas pilhas / celulares
- programas asm / C / Java / JavaScript / PHP / LISP / Haskell / Python / Prolog / R / Ruby / MATLAB /...
- Parallel / Distribuited / Quantum / DNA computing

* Não pode ser provada; porém pode ser refutada.

Sumário

Decidibilidade

O reconhecimento duma cadeia por uma MT pode ter três resultados possíveis: **aceitação**, **rejeição**, **loop infinito**

- Uma linguagem L é **Turing-reconhecível** (ou **recursivamente enumerável**) se existe uma MT que aceita todas as cadeias de L (**procedimento**). Para as cadeias que não pertencem a L, M pode rejeitar ou cair num loop infinito
- Uma linguagem L é **Turing-decidível** (**recursiva** ou simplesmente **decidível**) se existe uma MT que aceita todas as cadeias de L e rejeita as que não pertencem a L (**algoritmo**)

Exemplo: As linguagens livres de contexto são decidíveis

Teorema 1: As linguagens decidíveis são fechadas respeito à $\cup, \cap, \ ^c, \cdot, \ ^*, \ ^+$

Decidibilidade

O reconhecimento duma cadeia por uma MT pode ter três resultados possíveis: **aceitação**, **rejeição**, **loop infinito**

- Uma linguagem L é **Turing-reconhecível** (ou **recursivamente enumerável**) se existe uma MT que aceita todas as cadeias de L (**procedimento**). Para as cadeias que não pertencem a L, M pode rejeitar ou cair num loop infinito
- Uma linguagem L é **Turing-decidível** (**recursiva** ou simplesmente **decidível**) se existe uma MT que aceita todas as cadeias de L e rejeita as que não pertencem a L (**algoritmo**)
- Uma linguagem L é **não-decidível** se não existe nenhuma MT que decide L

Decidibilidade

O reconhecimento duma cadeia por uma MT pode ter três resultados possíveis: **aceitação**, **rejeição**, **loop infinito**

- Uma linguagem L é **Turing-reconhecível** (ou **recursivamente enumerável**) se existe uma MT que aceita todas as cadeias de L (**procedimento**). Para as cadeias que não pertencem a L, M pode rejeitar ou cair num loop infinito
- Uma linguagem L é **Turing-decidível** (**recursiva** ou simplesmente **decidível**) se existe uma MT que aceita todas as cadeias de L e rejeita as que não pertencem a L (**algoritmo**)
- Uma linguagem L é **não-decidível** se não existe nenhuma MT que decide L

Se $\mathcal{D}ec$ é o conjunto das linguagens decidíveis, $\exists\ L \notin \mathcal{D}ec$?

Exemplo de linguagem não decidível

Décimo problema de Hilbert (1900): Descrever, em um número finito de operações (i.e. um algoritmo), se uma equação diofantina $p(x_1, x_2, \ldots, x_n) = 0$ com coeficientes inteiros tem uma raíz inteira

- p é um polinômio com coeficientes inteiros
- Em termos de linguagens formais o problema seria descrito como

$$L_D = \{\ p(x_1, x_2, \ldots, x_n) \mid p\ tem\ raiz\ inteira\ \} \in \mathcal{D}ec\ ?$$

- Hilbert acreditava que era possível provar **sim**. Provar **não** era difícil pois precisava-se ter uma definição formal de algoritmo (só fornecida por Church e Turing em 1936)

Exemplo de linguagem não decidível

Décimo problema de Hilbert (1900): Descrever, em um número finito de operações (i.e. um algoritmo), se uma equação diofantina $p(x_1, x_2, \ldots, x_n) = 0$ com coeficientes inteiros tem uma raíz inteira

$$L_D = \{\ p(x_1, x_2, \ldots, x_n) \mid p\ tem\ raiz\ inteira\ \} \in \mathcal{D}ec\ ?$$

$$L_{D1} = \{\ p(x) \mid p\ tem\ raiz\ inteira\ \} \in \mathcal{D}ec\ ?$$

- L_{D1} é uma linguagem Turing-reconhecível. O que deve fazer a MT? Avaliar p nos valores $0, 1, -1, 2, -2, \ldots$ e conferir se o resultado é 0. Se sim, aceita
- L_{D1} também é Turing-decidível pois é possível determinar o intervalo inteiro $[-c, c]$ onde as raízes do polinômio residem.
- Em 1970 foi provado que L_D é não decidível

Matiyasevich, Y. V. The Diophantineness of enumerable sets. (Russian) Dokl. Akad. Nauk SSSR, 191, 1970

Exemplo de linguagem não decidível - PCP

O problema da correspondência de Post (1946): Dadas duas listas $\alpha_1, \ldots, \alpha_n$ e $\beta_1, \ldots, \beta_n$ de símbolos sobre alfabeto Σ ($|\Sigma| > 1$), determinar se existe uma sequência de índices entre $1..N$ tais que $\alpha_{i_1}, \ldots, \alpha_{i_K} = \beta_{i_1}, \ldots, \beta_{i_k}$

Exemplo: 1 2 3 4

$$\left\{ \left[\frac{\mathtt{b}}{\mathtt{ca}}\right], \left[\frac{\mathtt{a}}{\mathtt{ab}}\right], \left[\frac{\mathtt{ca}}{\mathtt{a}}\right], \left[\frac{\mathtt{abc}}{\mathtt{c}}\right] \right\}$$

Solução: 2 1 3 2 4

$$\left[\frac{\mathtt{a}}{\mathtt{ab}}\right] \left[\frac{\mathtt{b}}{\mathtt{ca}}\right] \left[\frac{\mathtt{ca}}{\mathtt{a}}\right] \left[\frac{\mathtt{a}}{\mathtt{ab}}\right] \left[\frac{\mathtt{abc}}{\mathtt{c}}\right]$$

Post E. L. A variant of a recursively unsolvable problem, Bull. Amer. Math. Soc., 52, 1946

Sumário

Maquina de Turing Universal

Uma MT pode receber como entrada outra MT? Como codificar $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$?

- Usar notação unária para $Q = \{q_0, \ldots, q_m\}$, $\Gamma = \{a_1, \ldots, a_n\}$ e $\{E, D\}$

s	a_1	...	a_n	q_0	...	q_m	E	D
$c(s)$	1		1^n	1		1^{m+1}	1	11

- Usar notação binária para a função de transição de estados

$$c(\delta(q_i, a) = (q_j, b, d)) = 1^{i+1}\ 0\ c(a)\ 0\ 1^{j+1}\ 0\ c(b)\ 0\ c(d)$$

- Para codificar M: codificar os estados de F separados por 0; separar F dos elementos de δ por 00

Maquina de Turing Universal

Exemplo: $c(M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, \sqcup\}, \delta, q_0, \sqcup, \{q_2\}))$

$$\delta(q_0, a) = (q_1, a, D),\ \ \delta(q_0, \sqcup) = (q_2, \sqcup, D),\ \ \delta(q_1, b) = (q_0, b, D)$$

Resposta:

s	a	b	$\sqcup$	q_0	q_1	q_2	E	D
$c(s)$	1	11	111	1	11	111	1	11

$$c(\delta(q_0, a) = (q_1, a, D)) = 1\ 0\ 1\ 0\ 11\ 0\ 1\ 0\ 11$$

$$c(\delta(q_0, \sqcup) = (q_2, \sqcup, D)) = 1\ 0\ 111\ 0\ 111\ 0\ 111\ 0\ 11$$

$$c(\delta(q_1, b) = (q_0, b, D)) = 11\ 0\ 11\ 0\ 1\ 0\ 11\ 0\ 11$$

$$c(M) = 111\ 00\ 10101101011\ 00\ 1011101110111011\ 00\ 1101101011011$$

Maquina de Turing Universal

$$U = (Q_U, \{0,1\}, \{0,1,\sqcup,\ldots\}, \delta_U, q_0, q_{aceita}, q_{rejeita})$$

- Recebe como entrada uma MT $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ e uma cadeia $w \in \Sigma$ codificada como

$$c(< M, w >) = c(M)\ 000\ c(w)$$

- Aceita a linguagem

$$L(U) = \{\ c(< M, w >) \mid w \in L(M)\ \}$$

- Pode ser especificada usando 3 fitas: a primeira recebe a entrada, a segunda simula a fita de M e a terceira armazena o estado atual do reconhecimento de w usando M

Maquina de Turing Universal - Funcionamento

1. Analisar a cadeia de entrada. Rejeitar caso não seja da forma $c(M)\ 000\ c(w)$
2. Copiar $c(w)$ na segunda fita colocando a cabeça no inicio
3. Procurar $c(q)$ em F na primeira fita: se achar **aceitar** senão copiar $c(q_0)$ na terceira fita colocando a cabeça no inicio
4. Repetir: Seja $q \in Q$ e $a \in \Sigma$ os símbolos cujo códigos estão representados no início da terceira e segunda fita.
   - 4.1 Procure por uma transição $c(q)0c(a)0c(q')0c(b)0c(d)$ na primeira fita
   - 4.2 Se não achar, procurar $c(q)$ em F na primeira fita: se achar **aceitar** senão **rejeitar**
   - 4.3 substitua $c(q)$ por $c(q')$ na fita 3 colocando a cabeça no inicio
   - 4.4 substitua $c(a)$ por $c(b)$ na fita 2
   - 4.5 movimente a cabeça da fita 2 dependendo de d

Universidade Federal do ABC

Problema da Parada: Dada $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ e $w \in \Sigma$, determinar se M pára com w como entrada (**HP**)

Teorema: **HP** é recursivamente enumerável porém não é decidível

Prova: https://www.youtube.com/watch?v=92WHN-pAFCs

Por contradição, suponhamos existe uma MT H tal que

$$H(M, w) = \begin{cases} aceita & se\ w \in L(M) \\ rejeita & se\ w \notin L(M) \end{cases}$$

Construímos D que recebe como entrada uma MT M, executa $H(M, c(M))$ e da como saída a negação do resultado

$$D(M) = \begin{cases} rejeita & se\ H(M, c(M))\ aceita \\ aceita & se\ H(M, c(M))\ rejeita \end{cases}$$

Universidade Federal do ABC

Problema da Parada: Dada $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ e $w \in \Sigma$, determinar se M pára com w como entrada (**HP**)

Por contradição, suponhamos existe uma MT H tal que

$$H(M, w) = \begin{cases} aceita & se\ w \in L(M) \\ rejeita & se\ w \notin L(M) \end{cases}$$

Construímos D que recebe como entrada uma MT M, executa $H(M, c(M))$ e da como saída a negação do resultado

$$D(M) = \begin{cases} rejeita & se\ H(M, c(M))\ aceita \\ aceita & se\ H(M, c(M))\ rejeita \end{cases}$$

$$D(M) = \begin{cases} rejeita & se\ c(M) \in L(M)\ aceita \\ aceita & se\ c(M) \notin L(M)\ rejeita \end{cases}$$

Universidade Federal do ABC

Problema da Parada: Dada $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$ e $w \in \Sigma$, determinar se M pára com w como entrada (**HP**)

Construímos D que recebe como entrada uma MT M, executa $H(M, c(M))$ e da como saída a negação do resultado

$$D(M) = \begin{cases} rejeita & se\ H(M, c(M))\ aceita \\ aceita & se\ H(M, c(M))\ rejeita \end{cases}$$

$$D(M) = \begin{cases} rejeita & se\ c(M) \in L(M)\ (aceita) \\ aceita & se\ c(M) \notin L(M)\ (rejeita) \end{cases}$$

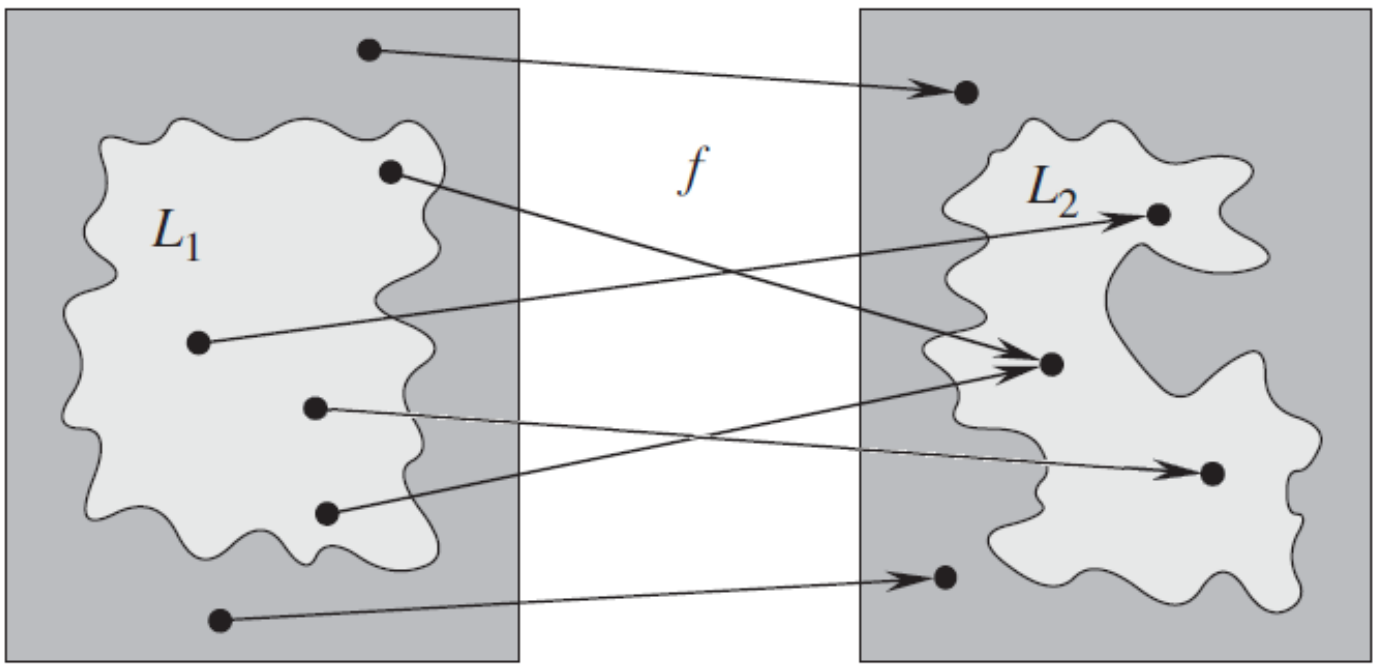
$$D(D) = \begin{cases} rejeita & se\ c(D) \in L(D)\ (aceita) \\ aceita & se\ c(D) \notin L(D)\ (rejeita) \end{cases}$$

Contradição: D rejeita $c(D)$ quando D aceita $c(D)$?! □

Sumário

Redutibilidade

Uma linguagem L_1 é redutível à uma linguagem L_2 ($L_1 \leq L_2$) se existe uma função computável $f : \Sigma^* \rightarrow \Sigma^*$ tal que
$\forall w \in \Sigma^*, w \in L_1 \Leftrightarrow f(w) \in L_2$



Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. Introduction to Algorithms

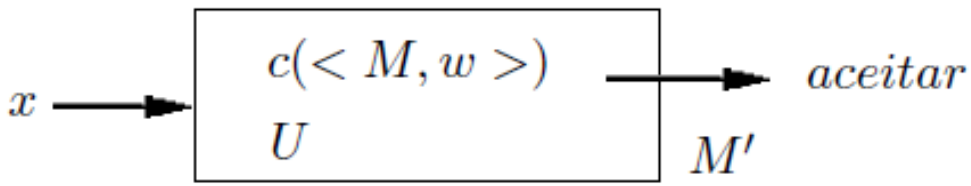
Universidade Federal do ABC

Redutibilidade

Uma linguagem L_1 é redutível à uma linguagem L_2 ($L_1 \leq L_2$) se existe uma função computável $f : \Sigma^* \rightarrow \Sigma^*$ tal que
$\forall w \in \Sigma^*, w \in L_1 \Leftrightarrow f(w) \in L_2$

Exemplo: Seja $L_U = \{ < M, w > \mid M \ para \ com \ w \}$ e $L_{ne} = \{ M \mid L(M) \neq \emptyset \}$. *Como provar $L_U \leq L_{ne}$?*

Precisamos transformar toda $< M, w >$ numa $< M' >$

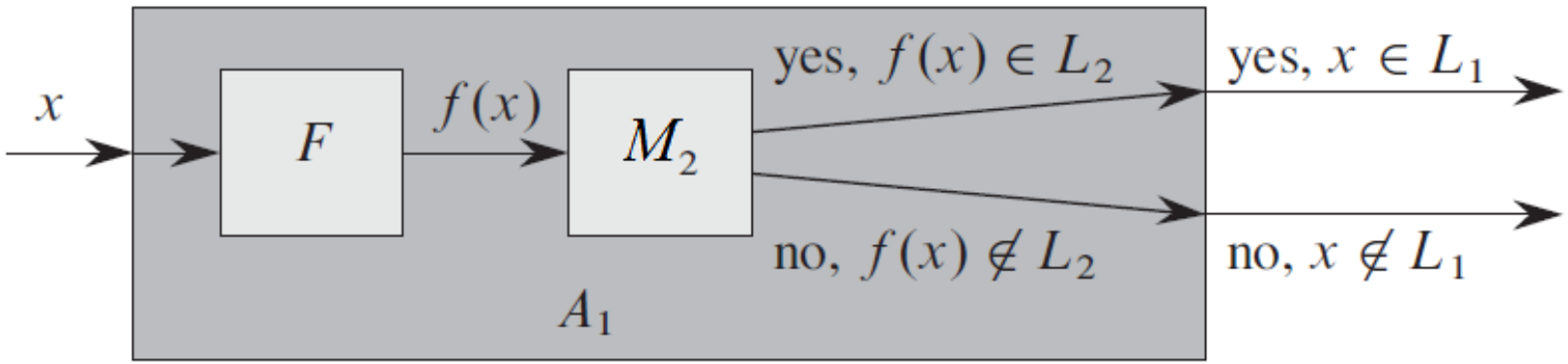


- M' substitui x por $c(< M, w >)$ e simula U. Se U aceita então M' aceita
- M pára com w sse $L(M') \neq \emptyset$

Redutibilidade

Uma linguagem L_1 é redutível à uma linguagem L_2 ($L_1 \leq L_2$) se existe uma função computável $f : \Sigma^* \to \Sigma^*$ tal que
$\forall w \in \Sigma^*, w \in L_1 \Leftrightarrow f(w) \in L_2$

Teorema 2: Se $L_1 \leq L_2$ e L_2 é decidível então L_1 é decidível



Redutibilidade

Uma linguagem L_1 é redutível à uma linguagem L_2 ($L_1 \leq L_2$) se existe uma função computável $f : \Sigma^* \rightarrow \Sigma^*$ tal que
$\forall w \in \Sigma^*, w \in L_1 \Leftrightarrow f(w) \in L_2$

Teorema 3: Se $L_1 \leq L_2$ e L_1 é não decidível então L_2 é não decidível. Se L_1 não é recursivamente enumerável então L_2 não é recursivamente enumerável

Exemplo: Seja $L_U = \{ < M, w > \mid M \ para \ com \ w \}$ e $L_{ne} = \{ < M > \mid L(M) \neq \emptyset \}$.

$L_U \leq L_{ne}$ *então* L_{ne} não é decidível

Redutibilidade

Uma linguagem L_1 é redutível à uma linguagem L_2 ($L_1 \leq L_2$) se existe uma função computável $f : \Sigma^* \rightarrow \Sigma^*$ tal que
$\forall w \in \Sigma^*, w \in L_1 \Leftrightarrow f(w) \in L_2$

Teorema 3: Se $L_1 \leq L_2$ e L_1 é não decidível então L_2 é não decidível. Se L_1 não é recursivamente enumerável então L_2 não é recursivamente enumerável

Outros exemplos de problemas não decidíveis:

- *Entscheidungsproblem*
- Uma MT reconhece uma linguagem regular/finita?
- Uma MT pára se a fita está em branco ($\epsilon \in L(M)$)?
- Uma MT reconhece uma palavra que começa por 0?

Sumário

Linguagens não recursivamente enumeráveis

Como sabemos que existem?

- Para todo Σ, o conjunto Σ^* é contável
- O conjunto de todas as MTs é contável pois cada MT tem uma codificação binária finita.
- Porém, de forma similar a $\mathbb{R}$, o conjunto $\mathcal{P}(\Sigma^*)$ é incontável

$$\begin{array}{rccccccccccc}
\Sigma^* = \{ & \varepsilon & , & 0 & , & 1 & , & 00 & , & 01 & , & 10 & , & 11 & , & 000 & , & 001 & , & \cdots & \} \\
A = \{ & & & 0 & , & & & 00 & , & 01 & , & & & & & 000 & , & 001 & , & \cdots & \} \\
\chi_A = & 0 & & 1 & & 0 & & 1 & & 1 & & 0 & & 0 & & 1 & & 1 & & \cdots &
\end{array}$$

Portanto, existem mais linguagens do que MTs

Linguagens não recursivamente enumeráveis - Exemplo

Tabela para representar a linguagem que aceita cada MT

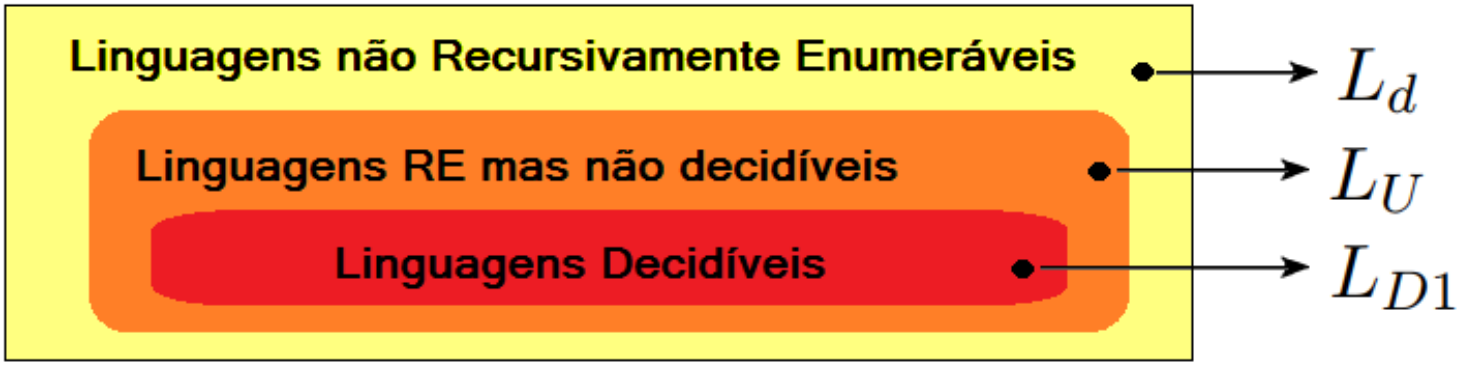
	1	2	3	4	...	Cadeias
1	0	1	1	0	...	
2	1	1	0	1	...	
3	1	1	1	1	...	
4	0	0	1	0	...	
...	.	.	.	.	...	
MT						

$$T[i,j] = \begin{cases} 1 & se\ w_j \in L(M_i)\ (aceita) \\ 0 & se\ w_j \notin L(M_i)\ (rejeita) \end{cases}$$

A linguagem da diagonalização L_d é o conjunto de todas as cadeias $w_i \notin L(M_i)$ (o complemento da diagonal da tabela).

Teorema 4: L_d é uma linguagem não recursivamente enumerável

Linguagens não recursivamente enumeráveis



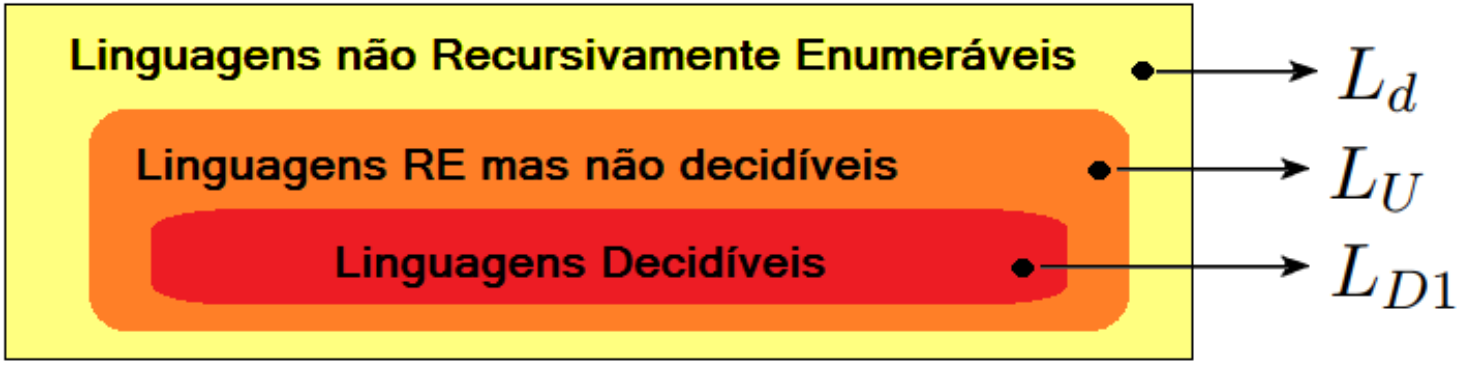
Teorema 5: Se L e L^c são RE então L é decidível. Portanto,

- ou L e L^c são decidíveis,
- ou L e L^c são não RE
- ou L é RE e L^c é não RE (ou vice-versa)

Exemplo: $L_{ne} = \{M \mid L(M) \neq \emptyset\}$ e $L_e = \{M \mid L(M) = \emptyset\}$

Já provamos $L_U \leq L_{ne}$; portanto L_{ne} não é decidível porém ela é RE. Como $L_e = L_{ne}^c$ e pelo Teorema 5 não é RE.

Linguagens não recursivamente enumeráveis



Teorema 5: Se L e L^c são RE então L é decidível. Portanto,

- ou L e L^c são decidíveis,
- ou L e L^c são não RE
- ou L é RE e L^c é não RE (ou vice-versa)

Exemplo: $L_{eq} = \{< M_1, M_2 > \ | L(M_1) = L(M_2)\}$. $L_e \leq L_{eq}$ transformando $< M >$ em $< M, M_\emptyset >$ onde $M_\emptyset$ é uma MT que rejeita todas as entradas. Portanto pelo Teorema 3 L_{eq} não é RE.

Sumário

Propriedades das LREs

Qualquer conjunto de linguagens define uma propriedade

- Existem duas propriedades **triviais**:
  1. $P_F = \emptyset$: o conjunto das propriedades sempre falsas, i.e. que nenhuma LRE satisfaz
  2. $P_T = \mathcal{P}(\Sigma^*)$: o conjunto das propriedades sempre verdadeiras, i.e. que toda LRE satisfaz

  Exemplo: A propriedade de uma linguagem RE não ser reconhecida por uma MT é uma propriedade trivial. **Qual?**

- Uma propriedade P é **monotônica** se para todas duas linguagens L_1 e L_2, se $L_1 \subseteq L_2$ e $L_1 \in P$ então $L_2 \in P$

  Exemplo: A propriedade de uma linguagem incluir a cadeia vazia é uma propriedade monotônica. **Por quê?**

Teorema de Rice I

As propriedades das LREs podem ser pensadas como problemas sobre MTs, i.e. o conjunto das linguagens que satisfazem uma propriedade P, L_P, pode ser descrito pelas MT M tais que $L(M) \in L_P$

Teorema de Rice I: Toda propriedade não trivial das (w.r.t.) LREs é não decidível

Ideia da Prova: Se $\emptyset \notin P$, usa-se uma redução de **HP**$\leq L_P$. Se $\emptyset \in P$, usa-se o Teorema 5

Exemplo: A propriedade de uma linguagem não ser vazia é uma propriedade não trivial; portanto não é decidível. Também não decidíveis são e.g. incluir a cadeia vazia e ser infinita.

Teorema de Rice II

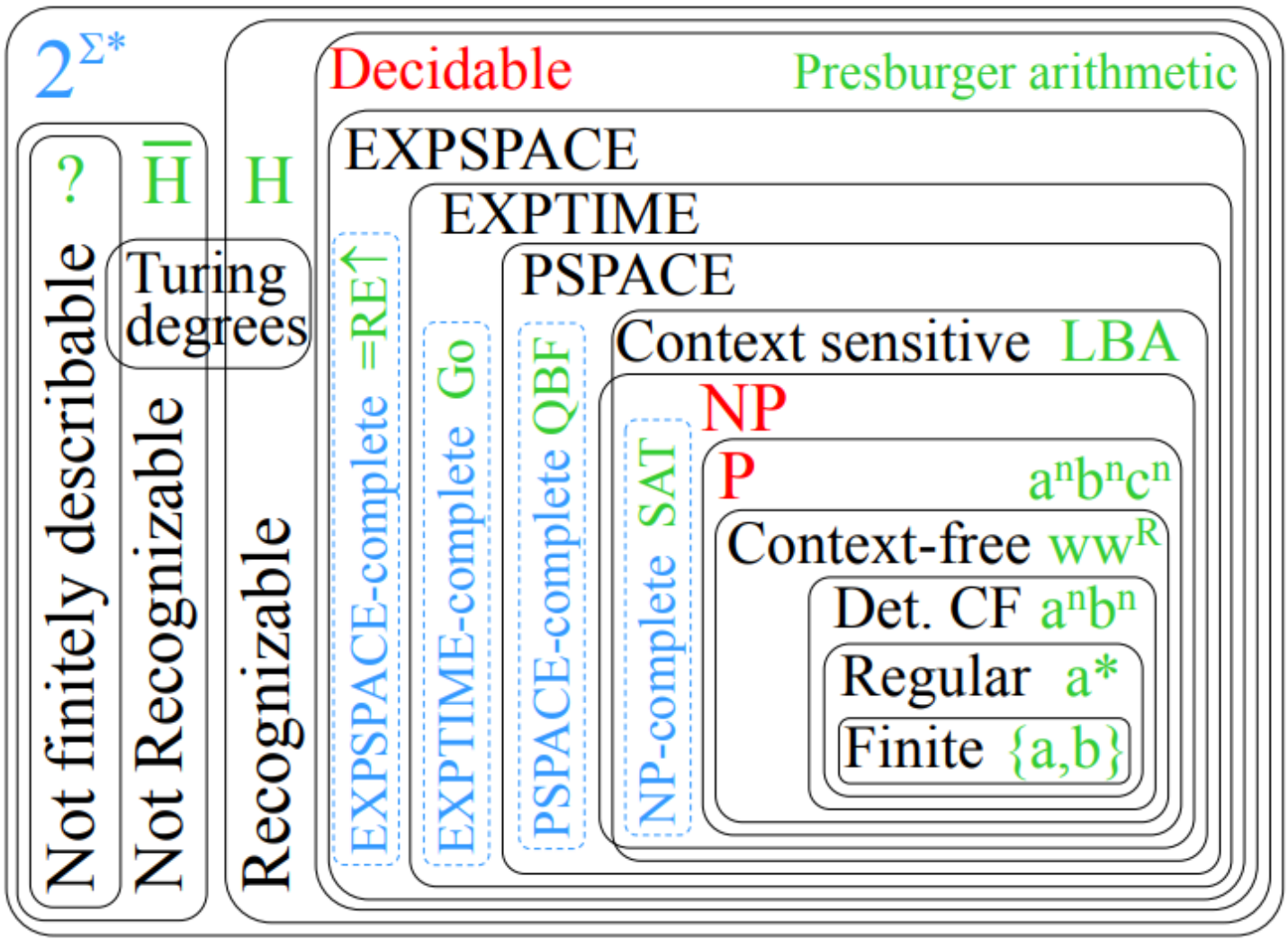
As propriedades das LREs podem ser pensadas como problemas sobre MTs, i.e. o conjunto das linguagens que satisfazem uma propriedade P, L_P, pode ser descrito pelas MT M tais que $L(M) \in L_P$

Teorema de Rice II: Toda propriedade não-monotônica das (w.r.t.) LREs é não recursivamente enumerável

Ideia da Prova: Usa-se uma redução de $\mathbf{HP}^c \leq L_P$.

Exemplo: A propriedade de uma linguagem ser finita (regular, livre de contexto) é uma propriedade não monótona. Portanto, não é RE.

Hierarquia de Chomsky estendida

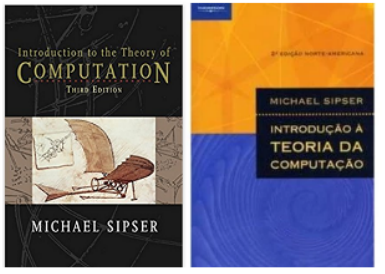


http://www.cs.virginia.edu/~robins/cs3102/slides/Theory_Slides_Computational_Universality.pdf

Universidade Federal do ABC

Sumário

Bibliografia Básica



1. **M. Sipser. Introduction to the Theory of Computation, Cengage Learning 3rd ed, 2012**
2. **J. E. Hopcroft, R. Motwani and J. D. Ullman. Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 2001**
3. H. R. Lewis, C. H. Papadimitriou. Elementos de Teoria da Computação. 2a edição, Bookman Companhia Ed., 2004.
4. P. Linz, An Introduction to Formal Languages and Automata, 6th edition, Jones & Bartlett Learning, 2016
5. A. Maheshwari, M.Smid. Introduction to Theory of Computation, 2014

http://cg.scs.carleton.ca/~michiel/TheoryOfComputation