

# Aspectos Formais da Computação

Prof. Sergio D Zorzo

Departamento de Computação – UFSCar

1º semestre / 2017

Aula 16

# Indecidibilidade

# Indecidibilidade

- É o estudo sobre o que os computadores podem e o que não podem fazer
- Trata-se de uma limitação conceitual
- Existem problemas que não são “computáveis”
  - Ou: existem problemas para os quais não existe um algoritmo
  - Ou: existem linguagens para as quais não existem decisores (Máquinas de Turing que sempre param)

# Indecidibilidade

- Para que estudar indecidibilidade?
  - 1. Se você se depara com um problema insolúvel, não há alternativa
    - Precisa ser simplificado ou alterado
  - 2. Ajuda a ganhar perspectiva sobre a computação
    - Sabendo o que é insolúvel, você conhece os limites do que pode e não pode fazer
    - Ajuda no projeto de soluções algorítmicas

# Um problema insolúvel

- Problema da Correspondência de Post (PCP)
- Uma instância do PCP é:
  - Duas listas de strings, com o mesmo tamanho  $k$ :
    - $A = w_1, w_2, \dots, w_k$
    - $B = x_1, x_2, \dots, x_k$
  - Para cada  $i$ , o par  $(w_i, x_i)$  é:
    - Um par correspondente
    - Uma correspondência
- Uma solução para essa instância do PCP é:
  - Uma sequência de um ou mais inteiros
    - $S = i_1, i_2, \dots, i_m$
  - Que quando interpretados como índices nas listas  $A$  e  $B$ 
    - a concatenação das strings apontadas por  $S$  em  $A$  é igual à concatenação das strings apontadas por  $S$  em  $B$

# PCP

- Exemplo:
  - Considere a instância do PCP à direita:

	Lista A	Lista B
i	wi	xi
1	1	111
2	10111	10
3	10	0

- Essa instância tem solução:
  - $S = (2, 1, 1, 3)$ 
    - $i_1 = 2, i_2 = 1, i_3 = 1, i_4 = 3$
  - Pois:
    - $w_2 w_1 w_1 w_3 = 10111 \ 1 \ 1 \ 10 = 101111110$
    - $x_2 x_1 x_1 x_3 = 10 \ 111 \ 111 \ 0 = 101111110$
- Outra solução:
  - $S = (2, 1, 1, 3, 2, 1, 1, 3)$

# PCP

- Outro exemplo:

- Considere a instância do

PCP à direita:

- Essa instância NÃO tem solução!

- É simples demonstrar:

- Uma solução  $S = (i_1, i_2, i_3, \dots)$ , certo?

- $i_1 = 2$  e  $i_1 = 3$  é impossível, portanto  $i_1 = 1$ !

- Então  $S = (1, i_2, i_3, \dots)$ , certo?

- Então

- $A = 10\dots$
    - $B = 101\dots$
    - Certo?

	Lista A	Lista B
i	$w_i$	$x_i$
1	10	101
2	011	11
3	101	011

# PCP

- Continuando, e  $i_2$ , o  
que poderia ser?

- $i_2 = 1$  e  $i_2 = 2$  é impossível
  - Portanto  $i_2 = 3$ !

- Então temos:

- $S = (1, 3, i_3, \dots)$
- $A = 10101\dots$
- $B = 101011\dots$

- Nesse ponto,  $i_3 = 1$  e  $i_3 = 2$  é impossível

- Portanto  $i_3 = 3$
- $S = (1, 3, 3, \dots)$
- $A = 10101101\dots$
- $B = 101011011\dots$

- Da mesma forma,  $i_4=3, i_5=3, i_6=3$ , etc...

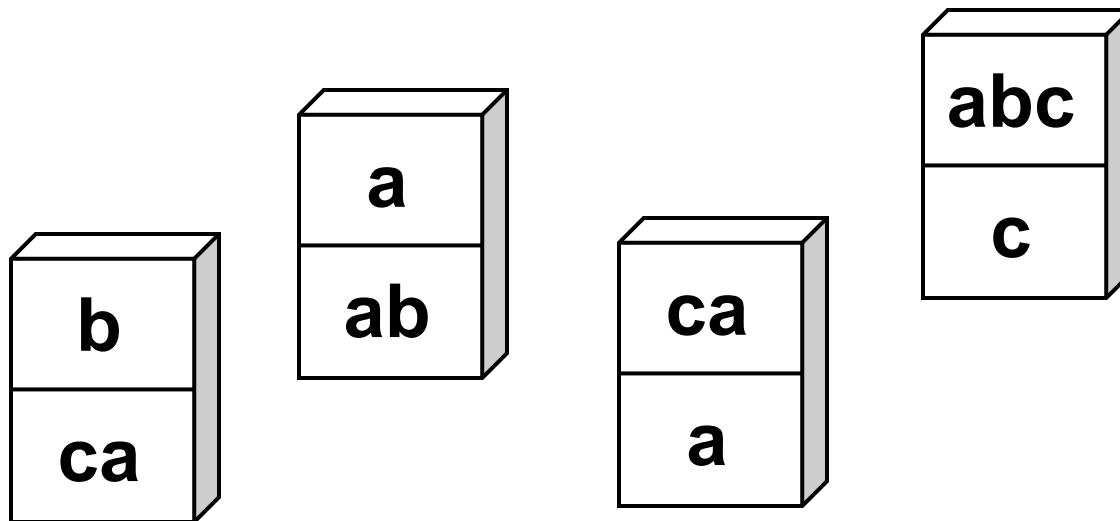
- Nunca vai parar! Ou seja, nunca haverá uma correspondência!

	Lista A	Lista B
i	$w_i$	$x_i$
1	10	101
2	011	11
3	101	011



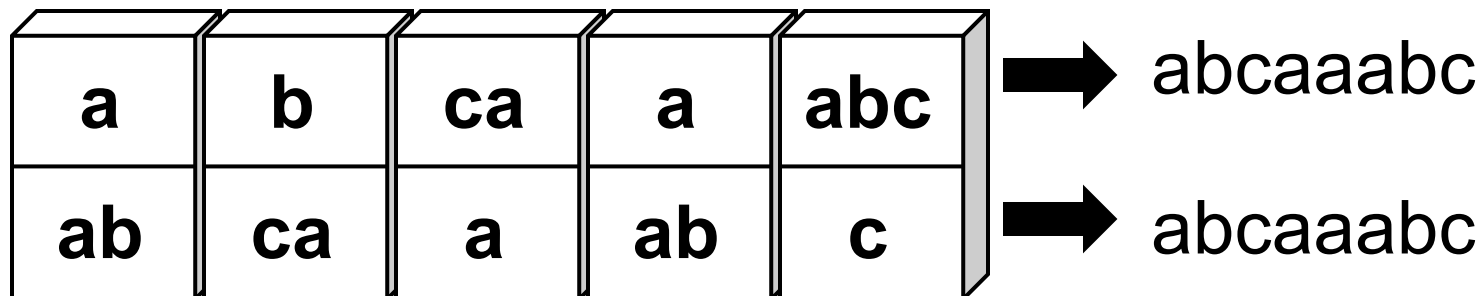
# PCP

- Outra forma de visualizar o PCP é imaginando um conjunto de peças de dominó, com strings de letras ao invés de números:



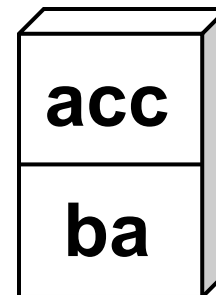
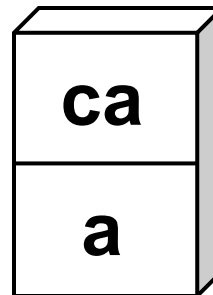
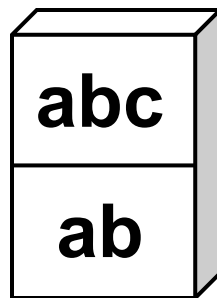
# PCP

- O objetivo é fazer uma lista de peças
  - Sem girá-las
  - Com repetições permitidas
  - Não precisa usar todas
- De forma que, lendo-se a linha de cima, tem-se a mesma string que lendo-se a linha de baixo



# PCP

- Para alguns conjuntos de peças, existe uma solução (exemplo anterior)
- Para outros (veja abaixo), não existe



# PCP

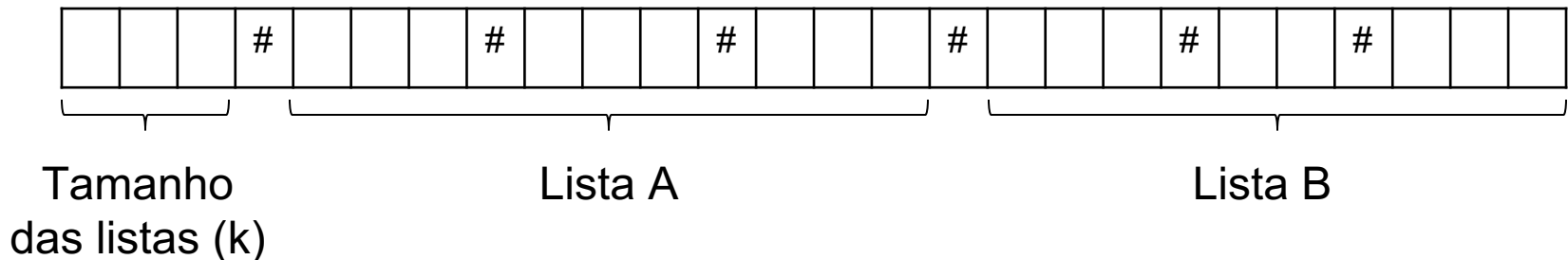
- O problema da correspondência de Post é:

***Dada uma instância do PCP, diga se essa instância tem uma solução***

- Esse problema é insolúvel
  - Não existe um algoritmo que consiga resolvê-lo

# Outra maneira de vermos o PCP

- Suponha uma versão binária do PCP (como a que vimos anteriormente)
  - Ou seja, as strings somente possuem 0s e 1s
- Uma linguagem que descreve instâncias do PCP poderia ser:
  - Linguagem LPCP é uma linguagem sobre  $\Sigma = \{0,1,\#\}$
  - Onde as cadeias representam instâncias do PCP, no seguinte formato:



- E as cadeias representam instâncias do PCP que possuem solução

# PCP como uma linguagem

- Exemplos

Cadeia c1

1	1	#	1	#	1	0	1	1	1	#	1	0	#	1	1	1	#	1	0	#	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

	Lista A	Lista B
i	wi	xi
1	1	111
2	10111	10
3	10	0

	Lista A	Lista B
i	wi	xi
1	10	101
2	011	11
3	101	011

Cadeia c2

1	1	#	1	0	#	0	1	1	#	1	0	1	#	1	0	1	#	1	1	#	0	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

# PCP como uma linguagem

- Nos exemplos anteriores
  - $c_1$  pertence à linguagem
  - $c_2$  não pertence à linguagem
  - nenhuma cadeia que não está no formato correto pertence à linguagem
- Dessa forma, o PCP pode ser visto como um problema de pertinência em uma linguagem
  - Que é a nossa definição de problema!
- Ou seja:
  - Dada uma cadeia  $c$ , determinar se ela pertence ou não à linguagem LPCP

# Outra forma de vermos a insolubilidade do PCP

- Veremos que não existe um algoritmo para o PCP (por enquanto, acredite que não existe)
- Na terminologia formal, isto significa que:
  - Não existe um decisor para a linguagem LPCP  
ou
  - É impossível projetar uma Máquina de Turing que sempre para (aceitando ou rejeitando) para a linguagem LPCP  
ou
  - É impossível construir um programa em C, Java, C#, Pascal ou qualquer outra linguagem de programação, que resolve este problema!



# Indecidibilidade

- Veremos:
  - Como provar que o PCP é insolúvel
    - Usaremos – obviamente – Máquinas de Turing e conceitos de linguagem para isso
    - Mas poderíamos fazer o mesmo com a noção de algoritmos!
- Veremos que existem duas formas de indecidibilidade
  - Existem problemas para os quais é impossível projetar uma MT
    - Linguagens não-recursivamente enumeráveis
  - Existem problemas para os quais é possível projetar uma MT, mas ela pode entrar em loop
    - Ou seja, não é um decisor
    - Linguagens não-recursivas
- Na prática, dá no mesmo, pois em ambos os casos não existe um algoritmo




# Uma linguagem que não é RE

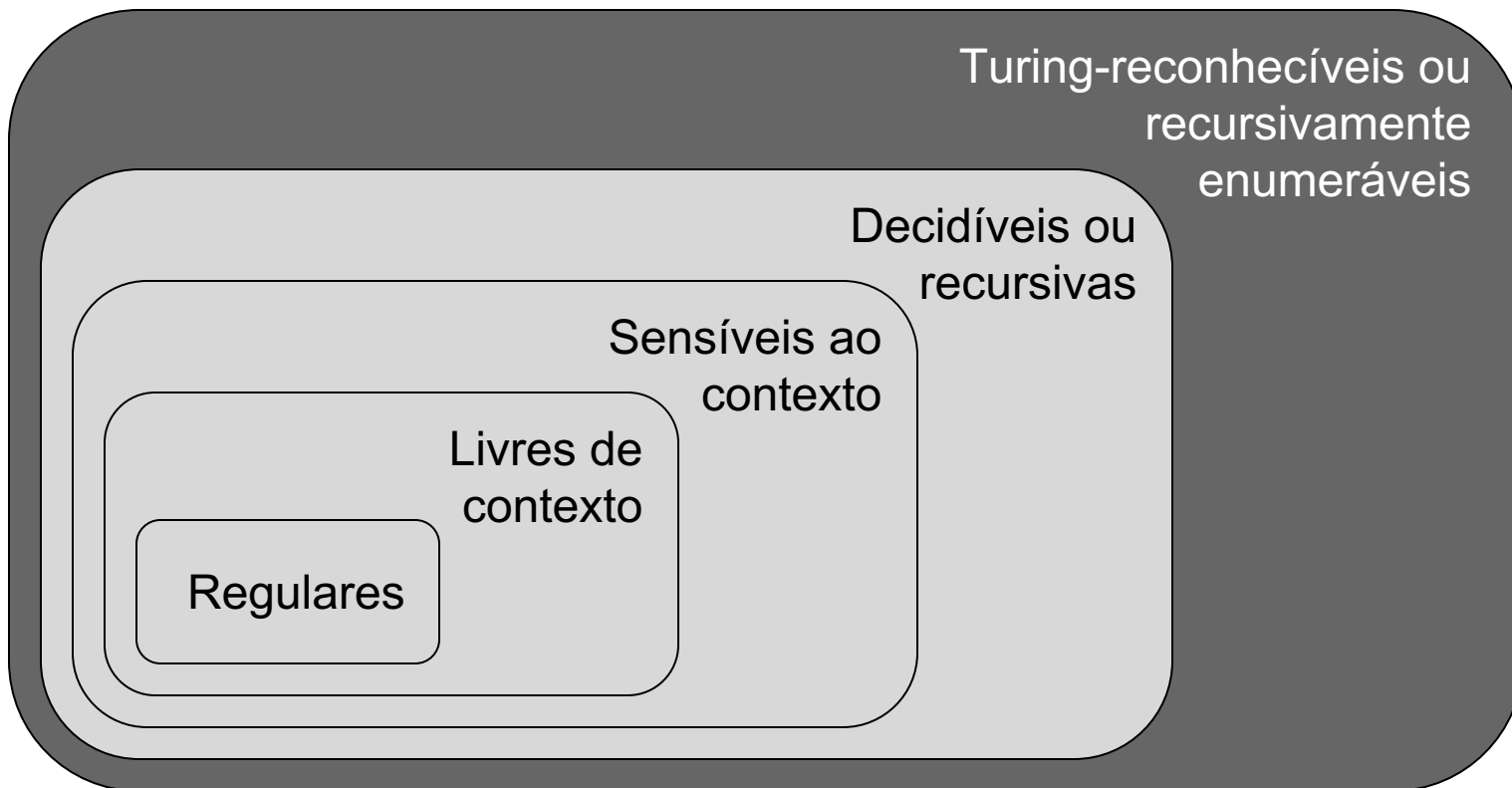
RE = Recursivamente Enumerável

# Uma linguagem que não é RE

Hierarquia	Gramáticas	Linguagens	Autômato mínimo
Tipo-0	Recursivamente Enumeráveis ou irrestritas	Recursivamente Enumeráveis	Máquinas de Turing
Tipo-1	Sensíveis ao contexto	Sensíveis ao contexto	MT com fita limitada
Tipo-2	Livres de contexto	Livres de contexto	Autômatos de pilha
Tipo-3	Regulares (Expressões regulares)	Regulares	Autômatos finitos

# Hierarquia de linguagens

-  Linguagens (problemas) decidíveis
-  Linguagens (problemas) indedidíveis      Não-recursivamente enumeráveis
-  Linguagens (problemas) recursivamente enumeráveis



# Hierarquia de linguagens



Existe uma MT que sempre para (decisor)

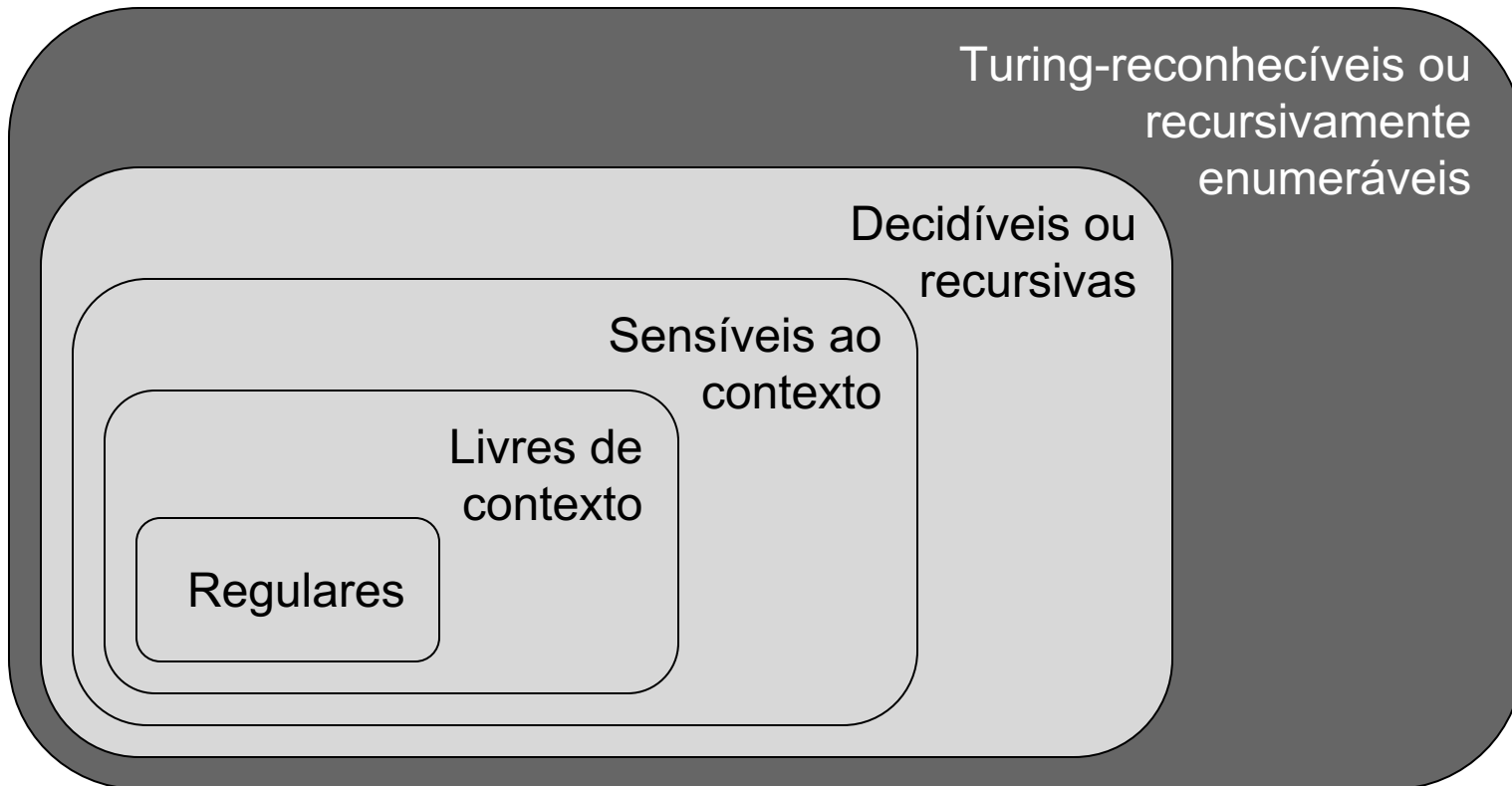


Não existe MT






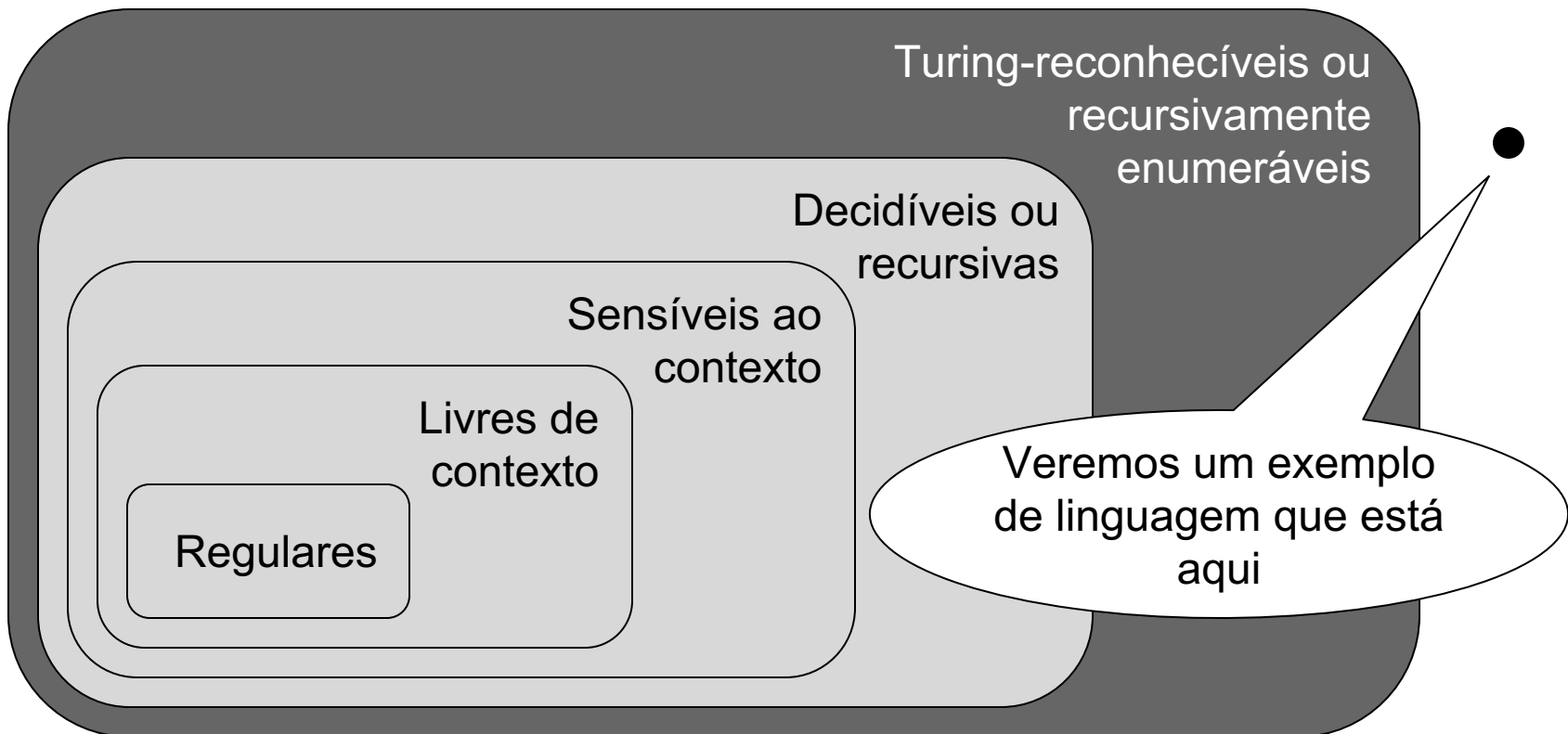
Existe uma MT, mas ela pode entrar em loop (reconhecedor)

Não-recursivamente enumeráveis



# Hierarquia de linguagens

-  Existe uma MT que sempre para (decisor)
  -  Não existe MT
  -  Existe uma MT, mas ela pode entrar em loop (reconhecedor)
- Não-recursivamente enumeráveis



# Mas antes, alguns conceitos

- Enumeração de strings binários
  - Veremos como codificar algumas coisas como strings binários
    - A exemplo do que fizemos na codificação das listas do PCP
  - Será útil atribuir inteiros a todos os strings binários possíveis
    - Cada string irá corresponder a um único inteiro
    - E cada inteiro irá corresponder a um único string

1	$\epsilon$
2	0
3	1
4	00
5	01
6	11
...	...
i	wi
...	...

# Mas antes, alguns conceitos

- Enumeração de strings binárias
  - Veremos como codificar algumas coisas como strings binários
    - A exemplo do que fizemos na codificação das listas do PCP
  - Será útil atribuir inteiros a todos os strings binários possíveis
    - Cada string irá corresponder a um único inteiro
    - E cada inteiro irá corresponder a um único string

segundo string

1	$\epsilon$
2	0
3	1
4	00
5	01
	11
...	...
i	$w_i$
...	...

quinto string

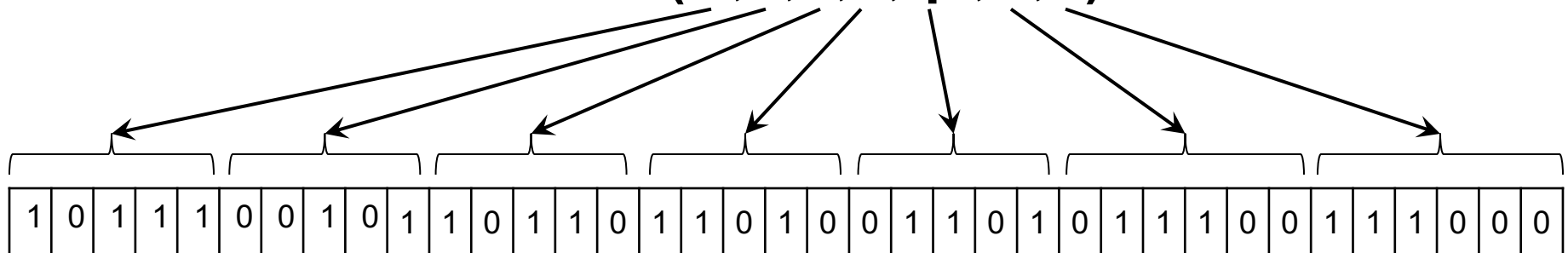
i-ésimo string



# Códigos para máquinas de Turing

- Vamos criar um código binário para máquinas de Turing
- Ou seja, representaremos uma máquina de Turing  $M$  em uma longa string de 0s e 1s

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



Atenção! Exemplo meramente ilustrativo. Essa não é uma codificação válida de uma MT!

# Códigos para máquinas de Turing

- Para que codificar uma MT em binário?
- Acabamos de enumerar todos os strings binários
- Poderemos, portanto, enumerar todas as MTs possíveis!
- Essa enumeração será útil na prova da indecidibilidade

Índice	MT	MT codificada em binário
1	M1	$\epsilon$
2	M2	0
3	M3	1
4	M4	00
...	...	...
293924	M293924	1011010100010011010101 00100101010010011101...
293925	M293925	1011010100010011010101 00100101010010011110...
...	...	...
i	Mi	wi
...	...	...

# Códigos para máquinas de Turing

- Para que codificar uma MT em binário?
- Acabamos de enumerar todos os strings binários
- Poderemos, portanto, enumerar todas as MTs possíveis!
- Essa enumeração será

Obviamente, as primeiras posições não são códigos “válidos” para MTs

Índice	MT	MT codificada em binário
1	M1	$\epsilon$
2	M2	0
3	M3	1
4	M4	00
...	...	...
293924	M293924	101101010001001101010100100101010010011101...
293925	M293925	101101010001001101010100100101010010011110...
...	...	...
i	Mi	wi
...	...	...

# Códigos para máquinas de Turing

- **$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$** 
  - Primeiro:  $\Sigma = \{0, 1\}$
  - Ou seja, assumimos que a entrada é codificada em binário
- Em seguida, precisamos codificar estados, símbolos de fita e sentidos E e D
  - $Q = \{q_1, q_2, \dots, q_r\}$ 
    - chamaremos os estados de  $q_1, q_2, \dots, q_r$ , para algum  $r$
    - $q_1$  será o estado inicial ( $q_0$ )
    - $q_2$  será o único estado de aceitação ( $F$ ) (é sempre possível converter uma MT com mais de um estado de aceitação para uma que tenha apenas um estado de aceitação)
    - Usaremos um código simples para representar cada  $q_i$
    - Ex:  $q_1 = 0$ ,  $q_2 = 00$ ,  $q_3 = 000$ ,  $q_4 = 0000$ , ...

# Códigos para máquinas de Turing

- **$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$** 
  - $\Gamma = \{X_1, X_2, \dots, X_s\}$ 
    - chamaremos os símbolos de fita de  $X_1, X_2, \dots, X_s$ , para algum  $s$
    - $X_1$  será o símbolo 0
    - $X_2$  será o símbolo 1
    - $X_3$  será B, o branco
    - Outros símbolos podem ser atribuídos aos inteiros restantes (4, 5, 6, ...)
    - Usaremos o código anterior para representar cada  $X_i$
    - Ex:  $X_1 = 0$ ,  $X_2 = 00$ ,  $X_3 = 000$ ,  $X_4 = 0000$ , ...

# Códigos para máquinas de Turing

- **$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$** 
  - Sentido E e D:
    - D1 = esquerda
    - D2 = direita
    - Usaremos o código anterior para representar cada Di
    - Ex: D1 = 0, D2 = 00

# Códigos para máquinas de Turing

- **$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$** 
  - $\delta$ :
    - Uma regra de transição no formato:
    - $\delta(q_i, X_j) = (q_k, X_l, D_m)$
    - É codificada como:
    - $0^i 10^j 10^k 10^l 10^m$
    - Nesse código,  $i, j, k, l$  e  $m$  são no mínimo 1, ou seja, não existirá nenhuma ocorrência de dois ou mais 1s consecutivos dentro de uma única transição

# Códigos para máquinas de Turing

- **$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$** 
  - M:
  - O código para a máquina M será:
    - $C_1 \ 11 \ C_2 \ 11 \ \dots \ C_{n-1} \ 11 \ C_n$
    - Onde cada um dos C's é o código para uma transição de M
- Obs: nenhum código válido para uma MT possui três 1s em sequência
  - Isso será útil depois



# Códigos para máquinas de Turing

- Ex:  $M = (\{q_1, q_2, q_3\}, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\})$
- onde  $\delta$  é definido pelas regras:
  - $\delta(q_1, 1) = (q_3, 0, D)$
  - $\delta(q_3, 0) = (q_1, 1, D)$
  - $\delta(q_3, 1) = (q_2, 0, D)$
  - $\delta(q_3, B) = (q_3, 1, E)$
- Os códigos para as regras são:
  - 0 1 00 1 000 1 0 1 00
  - 000 1 0 1 0 1 00 1 00
  - 000 1 00 1 00 1 0 1 00
  - 000 1 000 1 000 1 00 1 0
- O código para M é
  - 0100100010100 11 0001010100100 11  
00010010010100 11 0001000100010010

# Códigos para máquinas de Turing

- Existem muitos códigos “inválidos”
  - 1, 0, 11, 111111111111, etc
- Nesses casos, assumiremos que a MT possui apenas um estado e nenhuma transição
  - Ou seja, a MT para imediatamente sobre qualquer entrada, sem aceitar
  - Ou seja, a linguagem dessas máquinas é vazia ( $\emptyset$ )

Índice	MT	MT codificada em binário	Linguagem
1	M1	$\epsilon$	$\emptyset$
2	M2	0	$\emptyset$
3	M3	1	$\emptyset$
4	M4	00	$\emptyset$
...	...	...	...
293924	M293924	010010001010011 00010101001...	$L_{293924} = \{11, 101, \dots\}$
293925	M293925	10110110010010 1010010011110.. .	$\emptyset$
...	...	...	...
i	Mi	wi	Li
...	...	...	...

# Códigos para máquinas de Turing

- Existem muitos códigos “inválidos”
  - 1, 0, 11, 111111111111, etc
- Nesses casos, assumiremos que a MT possui apenas um estado e nenhuma transição
  - Ou seja, a MT para imediatamente sobre qualquer entrada, sem aceitar
  - Ou seja, a linguagem dessas máquinas é vazia ( $\emptyset$ )

Índice	MT	MT codificada em binário	Linguagem
1	M1	$\epsilon$	$\emptyset$
Código “inválido”		0	$\emptyset$
3	M3	1	$\emptyset$
Código “inválido”		00	$\emptyset$
...	...	...	...
Código “válido”		010010001010011 00010101001...	$L_{293924} = \{11, 101, \dots\}$
293925	M <sub>293925</sub>	10110110010010 1010010011110..	$\emptyset$
Código “inválido”		.	.
		...	...
I	MI	wi	Li
...	...	...	...

# Uma definição vital

# Uma definição vital

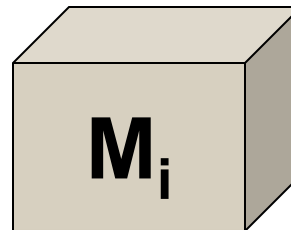
- A linguagem  $L_d$ , a *linguagem da diagonalização*, é o conjunto de strings  $w_i$ , tais que  $w_i$  não está em  $L(M_i)$
- Vamos detalhar essa definição:
  - $w_i$  é uma string:

0	1	0	0	1	1	0	0	1	0	1	1	0	1	1	0	0	0	1	1	0	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$w_i$

# Uma definição vital

- A linguagem  $L_d$ , a *linguagem da diagonalização*, é o conjunto de strings  $w_i$ , tais que  $w_i$  não está em  $L(M_i)$
- Vamos detalhar essa definição:
  - $w_i$  é uma string:
    - $w_i$  codifica uma MT (“válida” ou “inválida”) chamada  $M_i$

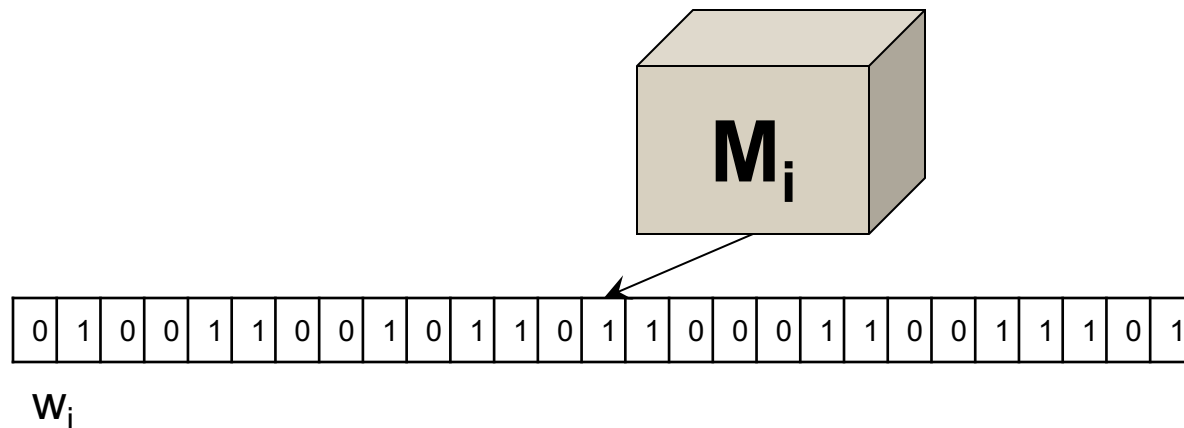


0	1	0	0	1	1	0	0	1	0	1	1	0	1	1	0	0	0	1	1	0	0	1	1	1	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$w_i$

# Uma definição vital

- A linguagem  $L_d$ , a *linguagem da diagonalização*, é o conjunto de strings  $w_i$ , tais que  $w_i$  não está em  $L(M_i)$
- Vamos detalhar essa definição:
  - $w_i$  é uma string:
    - $w_i$  codifica uma MT (“válida” ou “inválida”) chamada  $M_i$
    - Usaremos  $w_i$  como entrada para  $M_i$



# Uma definição vital

- Se  $M_i$  aceitar  $w_i$ ,  $w_i$  não faz parte de  $L_d$
- Se  $M_i$  não aceitar  $w_i$ ,  $w_i$  faz parte de  $L_d$
- Ou seja,  $L_d$  consiste de todos os strings que codificam máquinas de Turing que não aceitam quando recebem a si mesmas como entrada



# Diagonalização

- Considere a tabela à direita
  - Cada célula diz se uma Máquina de Turing  $M_i$  aceita o string de entrada  $w_j$ 
    - 1 significa “sim,  $M_i$  aceita  $w_j$ ”
    - 0 significa “não,  $M_i$  não aceita  $w_j$ ”
- Cada linha  $i$  é chamada de vetor característico para a linguagem  $L(M_i)$ 
  - Cada 1 nessa linha indica uma string que faz parte dessa linguagem
- Exs:
  - $M_4$  aceita  $w_2$  e  $w_4$
  - $M_3$  não aceita  $w_1$  nem  $w_2$
  - $M_2$  aceita  $w_1$  e  $w_2$

		$j$				
		1	2	3	4	...
$i$	1	0	1	1	0	...
	2	1	1	0	0	...
	3	0	0	1	1	...
	4	0	1	0	1	...
	...	...	...	...	...	...

Exemplo ilustrativo,  
pois as primeiras  
posições não são  
códigos “válidos”  
para MTs

# Diagonalização

- Nessa tabela, a diagonal é interessante
  - Informam as MTs que aceitam a si próprias como entrada
  - Ou seja, o complemento de  $L_d$
- Para construir  $L_d$ , basta complementar a diagonal
  - Nesse exemplo: 1000 ...
  - Ou seja:
    - $w_1$  pertence a  $L_d$
    - $w_2$  não pertence a  $L_d$
    - $w_3$  não pertence a  $L_d$
    - $w_4$  não pertence a  $L_d$
    - Etc...

		<i>j</i>				
		1	2	3	4	...
<i>i</i>	1	0	1	1	0	...
	2	1	1	0	0	...
	3	0	0	1	1	...
	4	0	1	0	1	...
		...	...	...	...	...

# Diagonalização

- Suponha que  $L_d$  fosse  $L(M)$  para alguma MT  $M$ 
  - Então existiria uma MT cujo vetor característico é o complemento da diagonal da tabela à direita
  - Ou seja, em alguma linha, por exemplo  $k$ , o complemento da diagonal (1000...) iria aparecer

		<i>j</i>						
		1	2	3	4	...	k	...
<i>i</i>	1	0	1	1	0	...		...
	2	1	1	0	0	...		...
	3	0	0	1	1	...		...
	4	0	1	0	1	...		...
	...	...	...	...	...	...		...
	k	1	0	0	0	...	?	...
	...	...	...	...	...	...		...

# Diagonalização

- Observe a célula marcada com “?”
  - Ela fica no encontro entre a diagonal e a linha hipotética  $k$
  - Que valor deveria ser colocado nessa célula?
- Precisamos olhar sob o ponto de vista da linha  $k$  e da diagonal
- Devemos analisar o processo de “transposição” da diagonal para a linha  $k$

		$j$						
		1	2	3	4	...	$k$	...
$i$	1	0	1	1	0	...		...
	2	1	1	0	0	...		...
	3	0	0	1	1	...		...
	4	0	1	0	1	...		...
	...	...	...	...	...	...		...
	$k$	1	0	0	0	...	?	...
	...	...	...	...	...	...		...

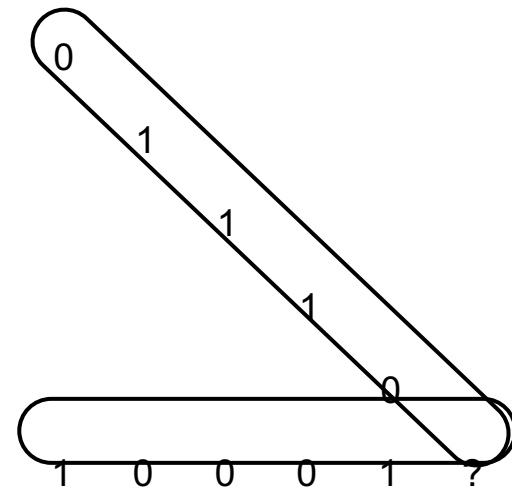
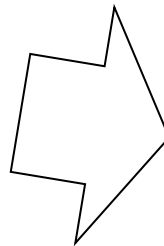
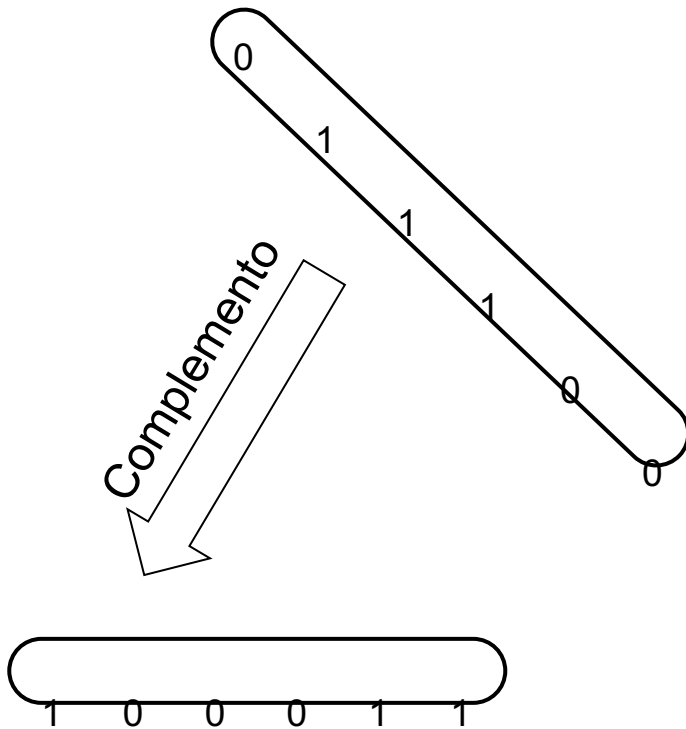
# Diagonalização

		<i>j</i>						
		1	2	3	4	5	6	...
<i>i</i>	1	0	1	1	0	1	1	...
	2	1	1	0	0	1	0	...
	3	0	0	1	1	1	0	...
	4	0	1	0	1	1	0	...
	5	1	1	0	1	0	1	...
	6	1	0	0	0	1	X	...
	...	...	...	...	...	...	...	...

- Acompanhe no exemplo
- Suponha que  $k = 6$
- Suponha que  $X = 0$ 
  - Diagonal = 011100
  - Complemento = 100011

# Diagonalização

- Acompanhe no exemplo
- Suponha que  $k = 6$
- Suponha que  $X = 0$ 
  - Diagonal = 011100
  - Complemento = 100011



# Diagonalização

- O mesmo aconteceria para qualquer  $k$ , e para qualquer  $X$
- Ou seja, há um paradoxo, uma contradição
- Nossa suposição de que  $M_k$  existe deve ser falsa
- Portanto, não existe uma máquina de Turing  $M_k$
- Ou seja, a linguagem  $L_d$  não é reconhecível por uma máquina de Turing

**Ou seja,  $L_d$  não é uma linguagem recursivamente enumerável**

# Uma linguagem não-RE

- O que significa isso?
- O “problema”  $L_d$  é indecidível
- Ou seja, dada uma máquina de Turing, é impossível decidir (determinar/resolver) se ela aceita a si mesma como entrada
- Ou seja, não existe um algoritmo que faça isso, para qualquer máquina de Turing



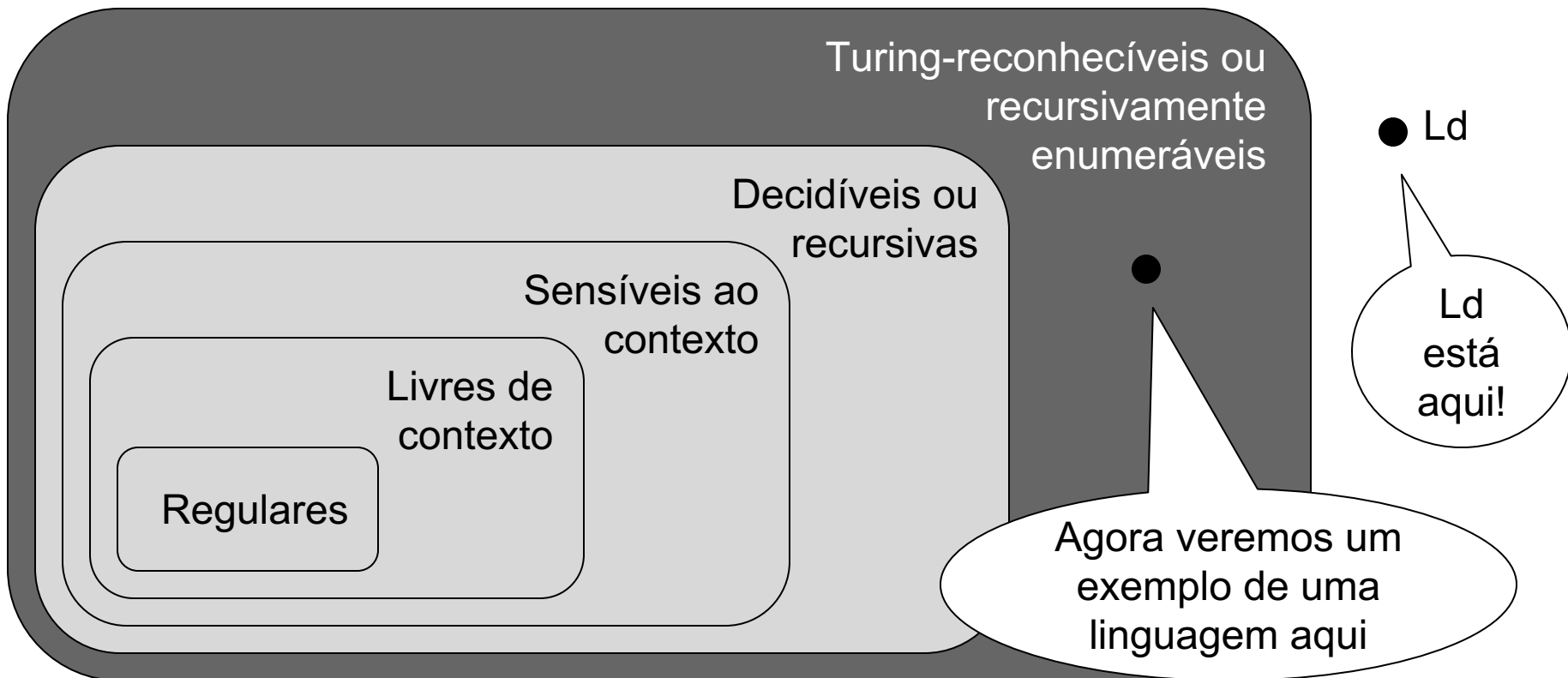
Um problema indecidível que é RE

# Um problema indecidível que é RE

- Agora iremos refinar a estrutura das linguagens RE (ou Turing-reconhecíveis)
- Dividiremos em duas classes:
  - Algoritmos: problemas para os quais existe uma MT que reconhece a linguagem, mas também reconhece (decide) as cadeias que não pertencem à linguagem.
    - São as MTs que sempre param, independente do fato de alcançar ou não um estado de aceitação
  - Linguagens RE que não são aceitas por nenhuma máquina de Turing com garantia de parada
    - Inconvenientes: se a entrada estiver na linguagem, saberemos disso mas, se a entrada não estiver na linguagem, a MT poderá continuar funcionando para sempre, e nunca teremos certeza de que a entrada não será aceita mais tarde

# Hierarquia de linguagens

- Existe uma MT que sempre para (decisor)
  - Não existe MT
  - Existe uma MT, mas ela pode entrar em loop (reconhecedor)
- Não-recursivamente enumeráveis



# Linguagens recursivas

- Uma linguagem  $L$  é recursiva se  $L = L(M)$  para alguma máquina de Turing  $M$  tal que:
  - Se  $w$  está em  $L$ , então  $M$  aceita (e portanto para)
  - Se  $w$  não está em  $L$ , então  $M$  para eventualmente, embora nunca entre em um estado de aceitação
- Uma MT desse tipo corresponde à nossa noção informal de um “algoritmo”
  - Nesse caso,  $L$  é um “problema” decidível, ou seja, existe um algoritmo que o resolve
- Na prática, a divisão entre recursiva/não-recursiva é mais importante do que a divisão RE/não-RE

# Complementos de linguagens recursivas e RE

- Existem alguns teoremas bastante simples de se provar, úteis nas demonstrações a seguir:
- Teorema: Se  $L$  é uma linguagem recursiva, o complemento de  $L$  ( $\sim L$ ) também o é.
- Prova:
  - Se  $L$  é recursiva, existe uma MT que sempre para, aceitando ou não a entrada
  - Basta modificar MT, de forma a inverter aceitação/não-aceitação, e iremos obter uma MT' que também sempre para (a modificação não altera esse fato)
  - MT' irá aceitar quando MT rejeita, e rejeitar quando MT aceita, reconhecendo exatamente o complemento de  $L$
  - Ou seja,  $\sim L$  é recursiva, pois existe um decisor

# Complementos de linguagens recursivas e RE

- Teorema: Se  $L$  e seu complemento são ambas RE, então  $L$  é recursiva (assim como seu complemento, pelo teorema anterior)
- Prova:
  - Se  $L$  é RE, existe uma MT1 que sempre para aceitando quando a entrada é um  $w$  que pertence a  $L$  (embora possa não parar nunca caso não pertença)
  - Se  $\sim L$  é RE, existe uma MT2 que sempre para aceitando quando a entrada é um  $w$  que não pertence a  $L$  (embora possa não parar nunca caso pertença)
  - Basta construir uma MT' que simula MT1 e MT2, aceitando quando MT1 aceitar (e parar), e rejeite quando MT2 aceitar (e parar)
  - Dessa forma, MT' sempre para, aceitando ou rejeitando, portanto  $L$  é recursiva.

# Complementos de linguagens recursivas e RE

- Ou seja, existem apenas quatro possibilidades:
  - **$L$  e  $\sim L$  são ambas recursivas**
  - **Nem  $L$  nem  $\sim L$  é RE**
  - **$L$  é RE mas não-recursiva, e  $\sim L$  não é RE**
  - **$\sim L$  é RE mas não recursiva, e  $L$  não é RE**
- Todas as outras possibilidades são excluídas pelos teoremas anteriores
- Exemplo:  $L_d$  não é RE, e portanto  $\sim L_d$  não pode ser recursiva
  - Ou seja, pode até existir uma MT para  $\sim L_d$ , mas não há garantia de que ela vá parar sempre

# Complementos de linguagens recursivas e RE

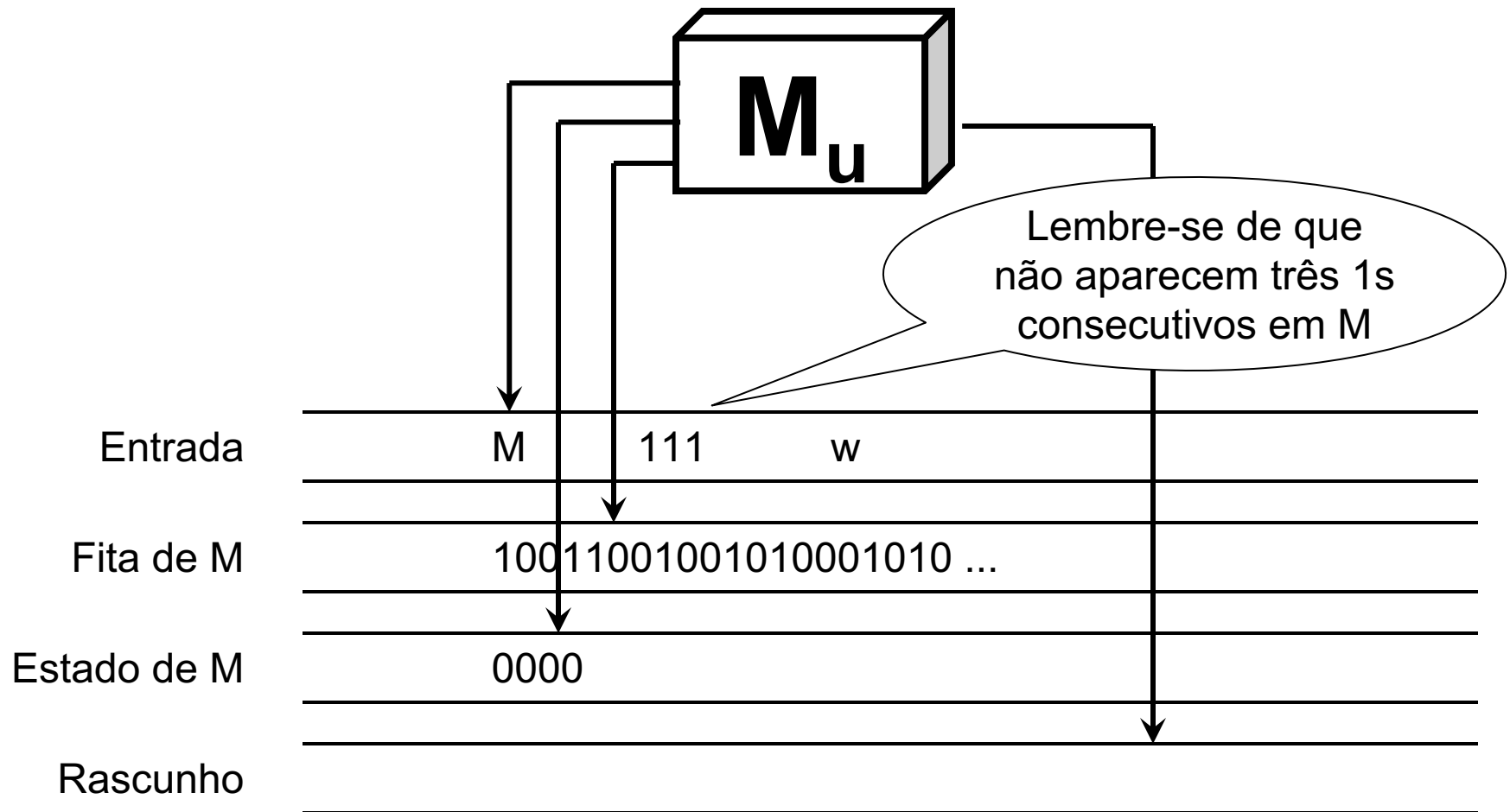
- Esses resultados são intuitivos
  - Suponha que exista um problema indecidível (como o PCP)
  - Suponha que eu conseguisse resolver a versão “negada” do PCP
    - Ou seja, encontrar as instâncias do PCP para as quais não existe solução
    - Bastaria inverter a resposta, e pronto, resolvi um problema insolúvel!
    - Mas isso é impossível, já que o PCP é indecidível!



# A linguagem universal

- Vimos anteriormente que uma MT pode simular um computador, executando um programa armazenado
  - Utilizando várias fitas, que armazenam o programa, os dados, rascunho, etc...
- E se esse programa armazenado for uma máquina de Turing?
  - Mais especificamente, a codificação binária que vimos no início dessa aula?
  - É possível construir uma máquina de Turing que executa máquinas de Turing codificadas?

# Máquina de Turing universal



# Máquina de Turing universal

- $M_u$  opera da seguinte forma:
  1. Examina a entrada para ter certeza que é um código válido
  2. Inicializa a segunda fita para conter a entrada  $w$
  3. Insere 0 (estado inicial de  $M$ ) na terceira fita e move a cabeça da segunda fita de  $M_u$  para a primeira célula simulada
  4.  $M_u$  procura na primeira fita uma transição  $0^i10^j10^k10^l10^m$ , olhando:
    1. Na fita 3 em busca de  $i$
    2. Na fita 2 em busca de  $j$

# Máquina de Turing universal

- $M_u$  opera da seguinte forma:
  4. Encontrando a transição  $0^i 1 0^j 1 0^k 1 0^l 1 0^m$  :
    - a. Muda o conteúdo da fita 3 para  $0^k$
    - b. Substitui  $0^j$  na fita 2 por  $0^l$  (usando o rascunho para fazer o deslocamento e administrar o espaço)
    - c. Move a cabeça na fita 2 para a posição do próximo 1 à esquerda ou direita, dependendo de  $m$  ( $m=1 \rightarrow$  esquerda,  $m=2 \rightarrow$  direita)
  5. Se  $M$  não tem nenhuma transição,  $M_u$  para
  6. Se  $M$  entrar em estado de aceitação,  $M_u$  aceita

# A linguagem universal

- A entrada da MT universal é um par  $(M, w)$ , onde  $M$  é uma MT codificada em binário e  $w$  é uma string binária
- Em alguns casos,  $M$  aceita  $w$ , em outros,  $M$  não aceita  $w$
- O conjunto de todos os pares  $(M, w)$ , tal que  $M$  aceita  $w$  é conhecido como linguagem universal, ou  $L_u$

# A linguagem universal

- Qual é o “problema” descrito pela linguagem universal?
  - Dada uma máquina de Turing  $M$  e uma cadeia qualquer  $w$ , determinar se  $M$  aceita  $w$
- Pensando em termos mais práticos:
  - Dado um programa de computador, e as possíveis entradas e saídas, esse “problema” consiste em verificar, automaticamente (algoritmicamente), se o programa funciona conforme o esperado
  - Se for possível resolver esse problema, eliminaríamos a necessidade de testes de software!!!
    - Bastaria construir um algoritmo (ou MT, ou programa) que fosse um verificador automático

# A linguagem universal

- Se a linguagem universal fosse decidível:
  - O Windows não teria bugs
  - Nossa vida de programadores seria muito mais fácil
- Mas..... (como você já deve ter adivinhado)
  - $L_u$  é indecidível!
  - É RE! Mas não recursiva!!
  - Ou seja ... Não existe algoritmo
  - Precisamos testar nossos programas
  - Teremos muitos e muitos bugs pela frente

# O problema da parada

- Semelhante à  $L_u$ , mas mais genérico
  - Seja  $H(M)$  o conjunto de entradas  $w$  tais que uma MT  $M$  para em  $w$  (aceitando ou não)
  - O problema da parada da MT é:
    - Dado um par  $(M, w)$ , decidir se  $M$  para em  $w$  ou não
- É o mesmo problema
  - Também é RE, mas não recursivo
  - Ou seja, indecidível



# Indecidibilidade da linguagem universal

- Teorema:  $L_u$  é RE mas não é recursiva
- A prova de que  $L_u$  é RE já foi feita
  - $M_u$  existe (mostramos anteriormente)
  - Portanto  $L_u$  é Turing-reconhecível
  - Portanto  $L_u$  é RE
- Continuando
  - Vamos supor que  $L_u$  fosse recursiva (buscaremos uma contradição)
    - Então,  $\sim L_u$  (o complemento de  $L_u$ ) também deve ser recursiva
    - Ou seja, existe uma MT  $M_{nu}$  que aceita  $\sim L_u$

# Indecidibilidade da linguagem universal

- Vamos supor que exista  $M_{nu}$ , tal que  $L(M_{nu}) = \sim L_u$ 
  - Ou seja, dada uma entrada  $(M, w)$ ,  $M_{nu}$  aceita se  $M$  rejeita  $w$ , e  $M_{nu}$  rejeita se  $M$  aceita  $w$
- E se eu aplicar, como entrada para  $M_{nu}$ , a entrada  $w11w$ ? Ou melhor, um par  $(w, w)$ ?
  - Ou melhor: um par  $(M, M)$
  - Ou seja,  $w$  é um determinado  $w_i$ , da nossa enumeração de  $M_i$ 's anterior
    - Ou seja,  $w$  codifica uma MT qualquer

# Indecidibilidade da linguagem universal

- Vemos então que  $M_{nu}$  é uma máquina bastante poderosa!!!
  - O que significa  $M_{nu}$  aceitar  $w111w$ ?
    - Significa que a máquina  $M$  rejeita a si mesma como entrada
  - O que significa  $M_{nu}$  rejeitar  $w111w$ ?
    - Significa que a máquina  $M$  aceita a si mesma como entrada
- Peraí: essa é a linguagem  $L_d$ !!
  - Quer dizer que  $M_{nu}$  decide a linguagem  $L_d$ ?
  - Mas  $L_d$  é indecidível!!
  - Exatamente: aí está a contradição!!
  - $M_{nu}$  não pode existir!
  - Ou seja,  $L_u$  é não-RE!
  - Ou seja,  $L_u$  não é recursiva!!

# Hierarquia de linguagens



Existe uma MT que sempre para (decisor)

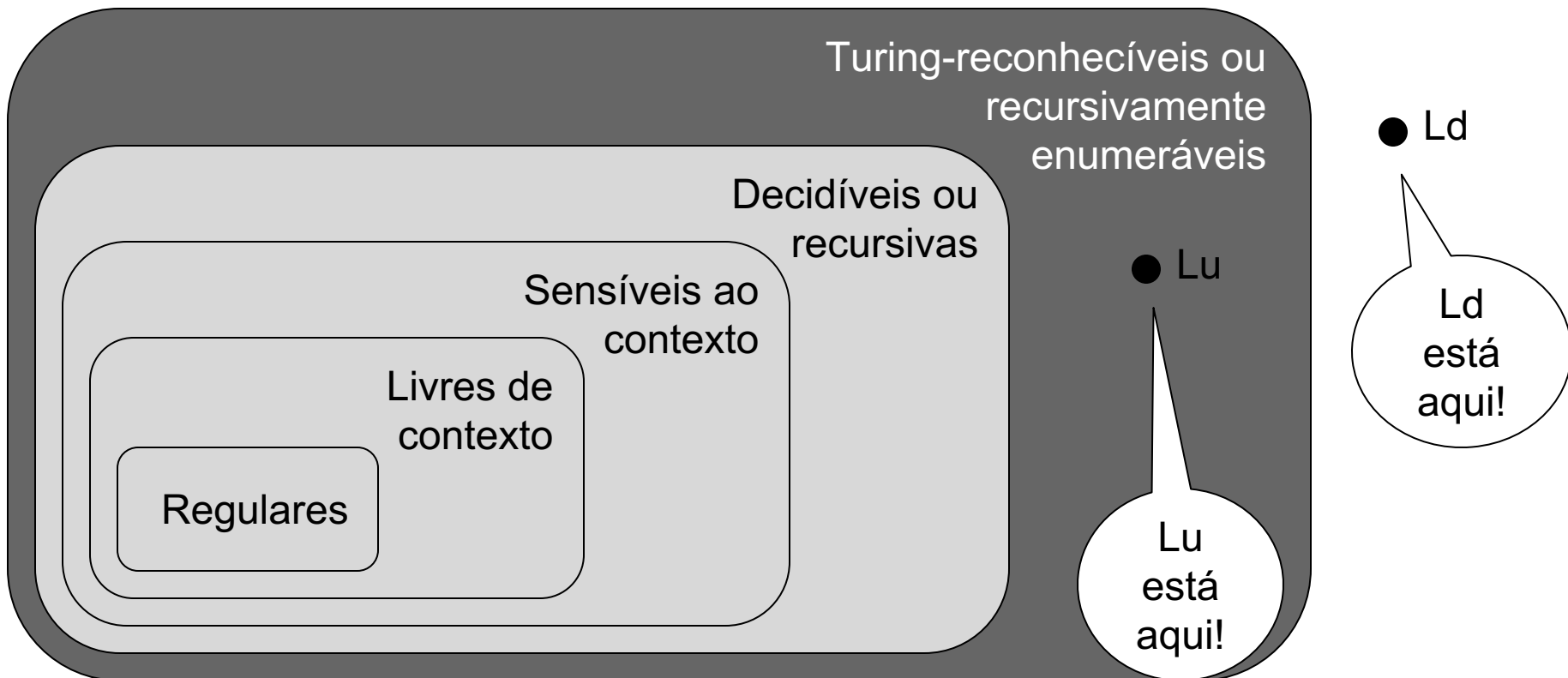


Não existe MT



Existe uma MT, mas ela pode entrar em loop (reconhecedor)

Não-recursivamente  
enumeráveis



Fim