Problemas Indecidíveis & Parcialmente Decidíveis

Que tipo de problemas são insolúveis por computador? Todos eles são teóricos?? A resposta é não!

Por exemplo, se dermos como entrada para um processo de verificação da corretude de programas um programa e a especificação formal da tarefa para a qual foi projetado, não conseguimos automatizar este processo.

Isto quer dizer que o problema geral de verificação de software não é solúvel por computador! Veremos outros exemplos práticos.

Indecidível significa:

Para um dado problema não existe um algoritmo que:

1. SEMPRE dá a resposta correta e,
2. SEMPRE termina.

Se desistimos da propriedade 2, teremos, por exemplo, antivírus que falham algumas vezes, pois sabemos que não são perfeitos, mas são os melhores que precisamos

Introdução

- Objetivos:
- Mostrar vários tipos de problemas insolúveis por computador para compreendermos quem são eles e aprender técnicas para provar a insolubilidade.
- Conhecer a MTU e a importância que teve em estimular o desenvolvimento de computadores com programas armazenados.

- Tópicos:
- Classes de Computabilidade
- Máquina de Turing Universal (MTU)
- O Problema da Parada
- Prova da Indecidibilidade do Problema da Parada
- Teorema de Rice --- a indecidibilidade do Problema da Parada não é um fenômeno único!
- Outros problemas indecidíveis e parcialmente decidíveis
- Redução de Problemas
- 4 técnicas para provar indecidibilidade: contradição, diagonalização, Teorema de Rice, Princípio/Método da Redução

Objetivos do Estudo da Computabilidade

Ser ou não ser, parar ou não parar, eis as questões!

- Investigar a existência ou não de algoritmos que solucionem determinada classe de problemas.
- Investigar os limites do que pode ser implementado em um computador.
- Evitar a pesquisa soluções inexistentes.
  - O 10º problema da lista de Hilbert lançada em 1900, que consiste na existência ou não de um algoritmo que determine se uma equação polinomial qualquer, com coeficientes inteiros possui solução nos inteiros, só foi provado não ter solução em 1970!

10º problema de Hilbert

Existe um algoritmo que decide se uma equação polinomial com várias variáveis (p.e. $x^2y + 3yz - y^2 - 17 = 0$) tem uma solução inteira?

Exemplos de ED:
Teorema de Pitágoras:
$x^2 + y^2 = z^2$

First degree Diophantine Equations with two variables (**aX + bY = c**)

Nota: Hilbert´s 10th problem asked if a technique for solving a general Diophantine existed. A general method exists for the solution of first degree Diophantine equations. However, the impossibility of obtaining a general solution was proven by Yuri Matijasevic in 1970.
(http://mathworld.wolfram.com/DiophantineEquation.html)

Abordagem Utilizada

- Utilizar problemas de decisão.

- Verificar a computabilidade de um problema se resume em verificar se determinada linguagem é recursiva, associando as condições de aceita/rejeita de uma MTU às respostas Sim/Não do problema de decisão.

- Assim, na TC verificamos a decidibilidade de problemas.

A classe de problemas decidíveis

é equivalente à classe das linguagens recursivas

- Alguns problemas não decidíveis são parcialmente decidíveis, isto é, são reconhecidos/aceitos por MT

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

- A cardinalidade da classe dos problemas não-computáveis é muito maior do que a dos problemas computáveis.

Computáveis é contável

Não-computáveis é não-contável

Porque usamos MT para verificar questões de computabilidade?

- Ao usar MT que trabalham com uma fita infinita, nós podemos capturar melhor
  - o que um computador poderá fazer, se não hoje, então num futuro próximo quando a memória limitada não for um problema...isto é, quando estas se tornarem muito grandes.

Diagonalização

- A prova original da indecidibilidade do problema da parada usa a técnica da diagonalização, descoberta por Cantor em 1873.

- Cantor estava preocupado em medir o tamanho de conjuntos infinitos. Se nós temos dois conjuntos infinitos como podemos dizer se um é maior do que o outro ou se eles tem o mesmo tamanho?

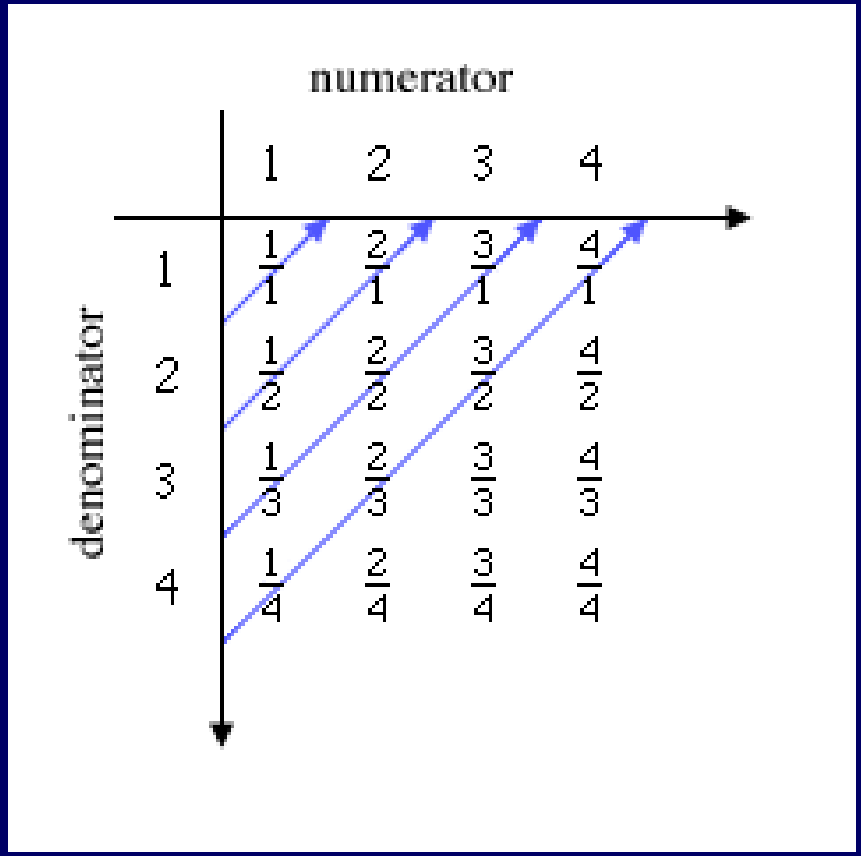
- Para conjuntos finitos nós contamos o número de elementos, mas se tentamos contar o número de um conjunto infinito....nós nunca terminaremos.

Proposta de Cantor

- Para conjuntos finitos nós pareamos os dois, assim comparamos os tamanhos sem ter que contar. Cantor estendeu esta idéia para conjuntos infinitos.
- Exemplo: números naturais e números pares.

Def. Um conjunto é contável se ele é finito ou tem o mesmo tamanho que N

- Q (conjuntos dos números racionais positivos) tem o mesmo tamanho dos N.
- A figura ao lado dá a correspondência de N para Q. Pulamos elementos repetidos e continuando assim obtemos a lista de todos os elementos de Q



1, 1/2, 2, 1/3, 3, 1/4, 2/3, 3/2, 4,...

Exemplos de conjuntos infinitos e contáveis

Podemos encontrar funções de correspondência entre $\mathbb{N}$ e $\mathbb{Z}$, $\mathbb{N}^2$, {a}*, {a,b}*:

0	1	2	3	4	5	6	…
0	$+1$	-1	$+2$	-2	$+3$	-3	…
(0,0)	(0,1)	(1,0)	(0,2)	(1,1)	(2,0)	(0,3)	…
ε	a	aa	aaa	aaaa	aaaaa	aaaaaa	…
ε	a	b	aa	ab	ba	bb	…

Teo (Cantor) R é incontável

- Provamos que não existe nenhuma função de correspondência entre R e N por contradição.
- Suponha que exista uma correspondência f, então temos que mostrar que f falha e não funciona como devia.
- Nós vamos construir um número x em R que não tem par com nenhum dos N que é uma contradição.

Suponha que exista uma f

n	f(n)
1	3.14159...
2	55.5555...
3	0.12345...
.	
.	
.	

x será um número entre 0 ..1 e nosso objetivo será mostrar que x <> f(n) para qualquer n

X = 0.4641...

Nós sabemos que x não é f(n) para qualquer n pois ele difere de f(n) no e-nésimo dígito da fração.

Para evitar problemas com 0.19999999... e 0.2000000 nós nunca selecionamos 0 e 9 na construção de x

Como temos x provamos que tal função não existe

Método da diagonalização

- Surgiu com a prova do teorema sobre o conjunto R ser incontável.

Importância dele para a TC

- Este teorema mostra que algumas linguagens não são decidíveis por MT nem mesmo reconhecidas por MT
  - pois existem incontáveis linguagens e somente um número contável de MT.

- Isto é, nós podemos ordenar as MT (ou programas) e para programas do mesmo tamanho ordená-los lexicograficamente, assim teremos o 1o programa, o 2o ...

- Existem infinitamente menos programas do que problemas e a única razão que não parece ser assim é porque nos preocupamos com problemas simples e bem estruturados e estes são geralmente decidíveis.

- Prova de que algumas linguagens não são reconhecíveis por MT em SIPSER, p. 164

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

Decidibilidade = Algoritmo

Aceitabilidade = Procedimento

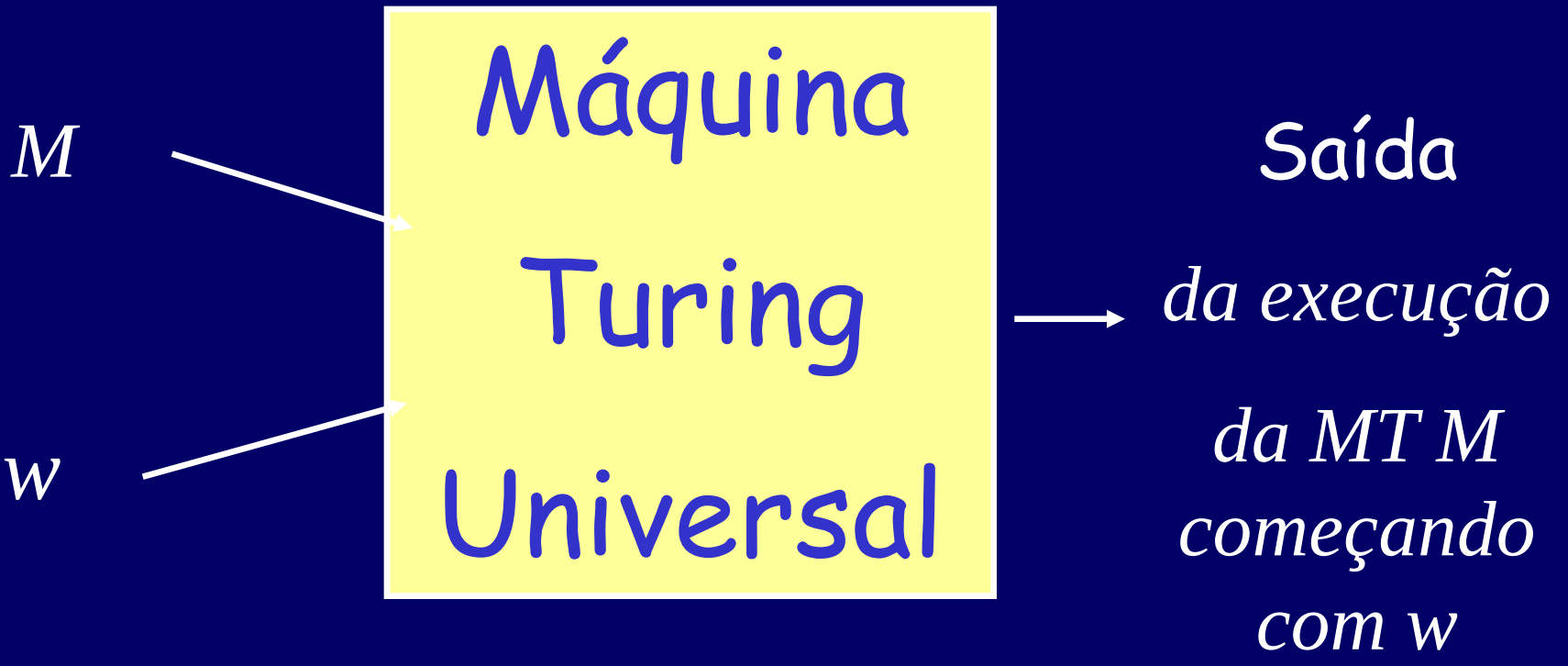
(reconhecedores são mais poderosos que decididores)

Antes de apresentar o primeiro problema indecidível...

- Vamos conhecer a MTU máquina de turing universal:
  - aquela que é capaz de simular qualquer outra máquina de turing a partir da descrição da máquina.

Entrada: < Descrição de alguma MT *M*, *w* >

Saída: resultado de rodar *M* com *w*



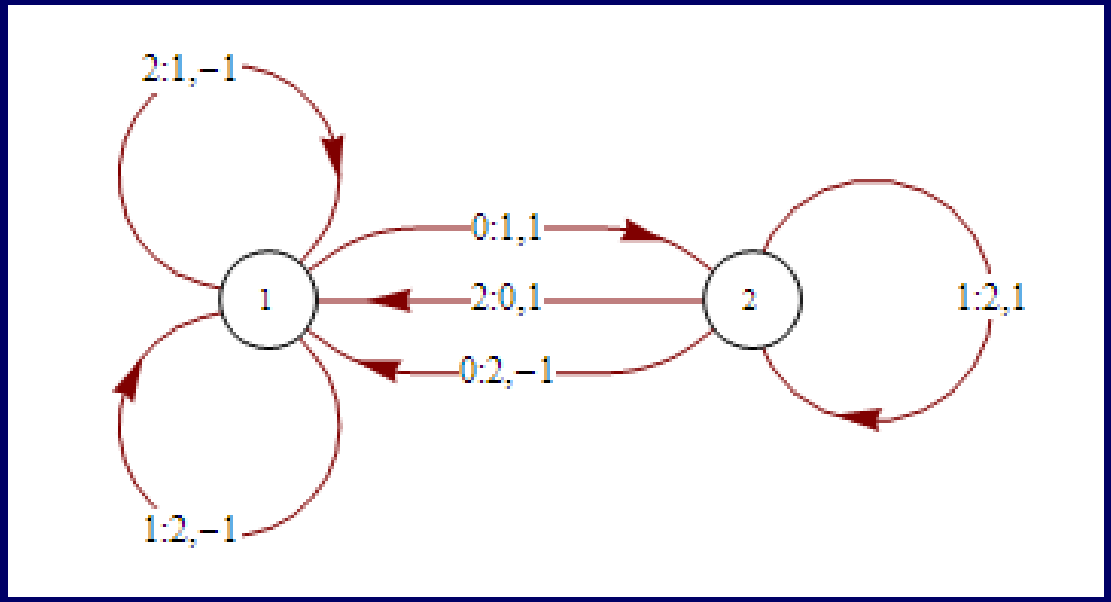
Curiosidades

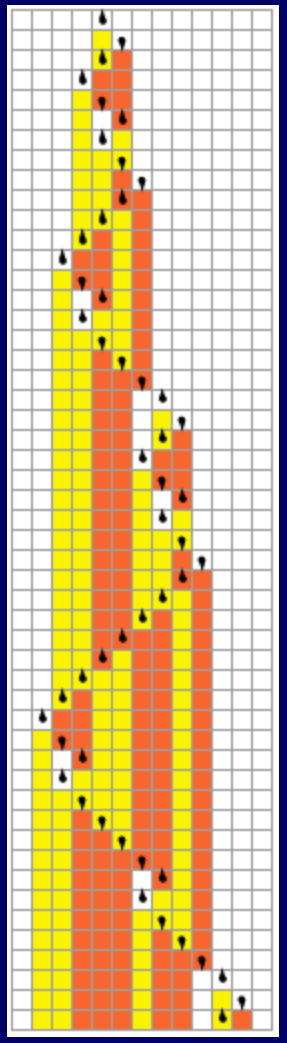
- Existem MTU com:
  - 4 símbolos, 7 estados (Marvin Minsky)
  - 4 símbolos, 5 estados
  - 2 símbolos, 22 estados
  - 18 símbolos, 2 estados
  - 2 estados, 5 símbolos (Stephen Wolfram)

- Novembro 2007: 2 estados, 3 símbolos

2-estados, 3-símbolos MTU

Seqüências de Configurações





SCIENTIFIC AMERICAN

News - October 25, 2007

***A New Kind of Science* Author Pays Brainy Undergrad $25,000 for Identifying Simplest Computer**
But will it jumpstart Stephen Wolfram's scientific revolution?

By JR Minkel

Five years ago, grown-up wunderkind Stephen Wolfram did his darnedest to alter the course of scientific history. The former particle physicist, who is by all accounts a g
of computers,
tome modest

And when tha

Alex Smith, University of Birmingham

Of course, simplicity is in the eye of the beholder. The 2,3 Turing machine described in the dense new 40-page proof "chews up a lot of tape" to perform even a simple job, Smith says. Programming it to calculate 2 + 2, he notes, would take up more memory than any known computer contains. And image processing? "It probably wouldn't finish before the end of the universe," he says.

MT Universal

- Uma MT Universal é uma MT que pode simular o comportamento de uma MT qualquer, mesmo da própria MT Universal.

Informalmente:

- Temos que utilizar um esquema de codificação para isto. Isto é, encontrar uma forma de descrever qualquer MT como uma seqüência linear de símbolos que sirva de entrada na fita da MTU.
  - Uma MT pode ser programada para decodificar a representação.
- A codificação pode ser feita de muitas formas, pois uma MT pode traduzir um esquema de codificação para outro.

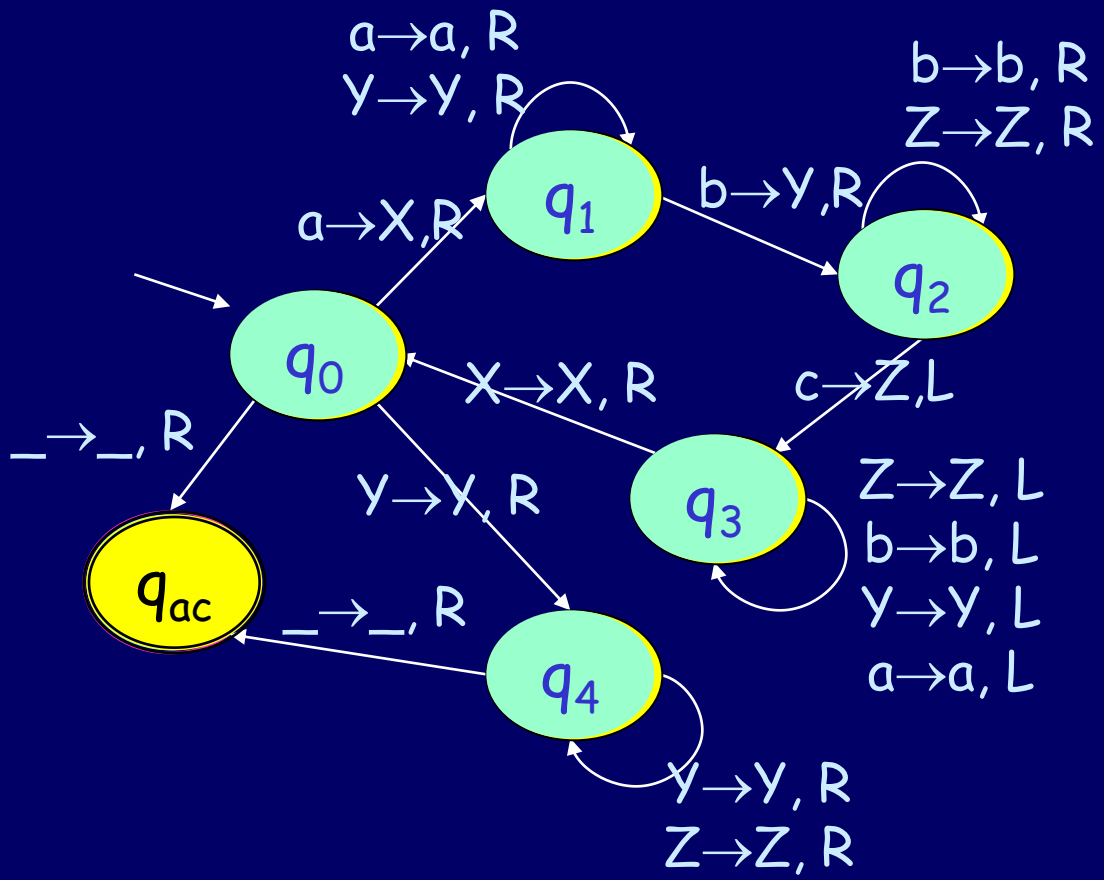
Exemplo de Codificação

- Temos que descrever o diagrama de transição (lista de transições) que pode, por convenção, ser precedido pelo estado inicial. Cada elemento da lista pode ser separado por *.

- Cada transição pode ser dada pelo:
  - estado fonte, estado destino, e rótulos da forma <a/b, L> ou <a/b, R>.

- A descrição da MT e a entrada pode ser separada por $, por exemplo.

q0**q0 q4 <Y/Y, R> * q4 q4 <Y/Y, R> * q4 q4 <Z/Z, R> * q4 qac <_/_,R> * q0 q1 <a/X, R> ...



Formalmente: Dada uma descrição <M,w> de uma MT M e uma entrada w, podemos simular M sobre w?

- Nós podemos fazer isto via uma MTU U (2-fitas) que contém M e w na primeira fita:
- Checar se M é uma MT
  Seja M = (Q,Σ,Γ,δ,q0,qaccept,qreject)
- Escrever a configuração inicial
  < q0w > na segunda fita.
- Repetir até que a configuração de parada seja alcançada
  - Troque a configuração da fita 2 pela próxima configuração de acordo com δ
- "Aceite" se qaccept é alcançado; "Rejeite" se qreject

Cada configuração mostra o símbolo da cadeia em foco à direita do estado e os símbolos já analisados à esquerda do estado: <11q01100>

MTU

- Vejam uma configuração e explicação detalhada de uma MTU em (H,M,U, 2001) p. 377 a 379.

O Problema da Parada

- A existência da MTU U mostra que a linguagem

$A_{TM} = \{\langle M,w\rangle \mid M \text{ é uma MT que aceita } w\}$

é reconhecível por MT.

Linguagem reconhecível por MT

Uma linguagem L é "Reconhecível por MT" se existe uma MT M tal que para todas as cadeias w:

- Se $w \in L$ M entra em q_{accept}
- Se $w \notin L$ M ou entra em q_{reject}

ou M nunca termina

Nós podemos também *decidir* A_{TM}?

- O problema acontece com os casos em que M não pára com a cadeia w. Por isto denominamos ele o problema da parada. .

Detector Universal de Loops

- Em geral nós não podemos decidir se a MT irá parar ou não com w, assim A_{TM} is indecidível.

2 formas de provar o Problema da Parada

- Contradição
- Método da Diagonalização

1) Provando a Indecidibilidade do Problema da Parada por Contradição

- Já que MTs são equivalentes a linguagens de programação vamos definir o Problema da Parada em termos de linguagens.

- O problema tem duas entradas: o texto de um programa R na linguagem L e uma entrada X para R.

- O problema pergunta se R pára para a entrada X, que denotamos por $R(X)\downarrow$.
  O caso de R não parar sobre X é denotado por $R(X)\uparrow$

O Problema da parada

- Queremos provar a seguinte afirmação:

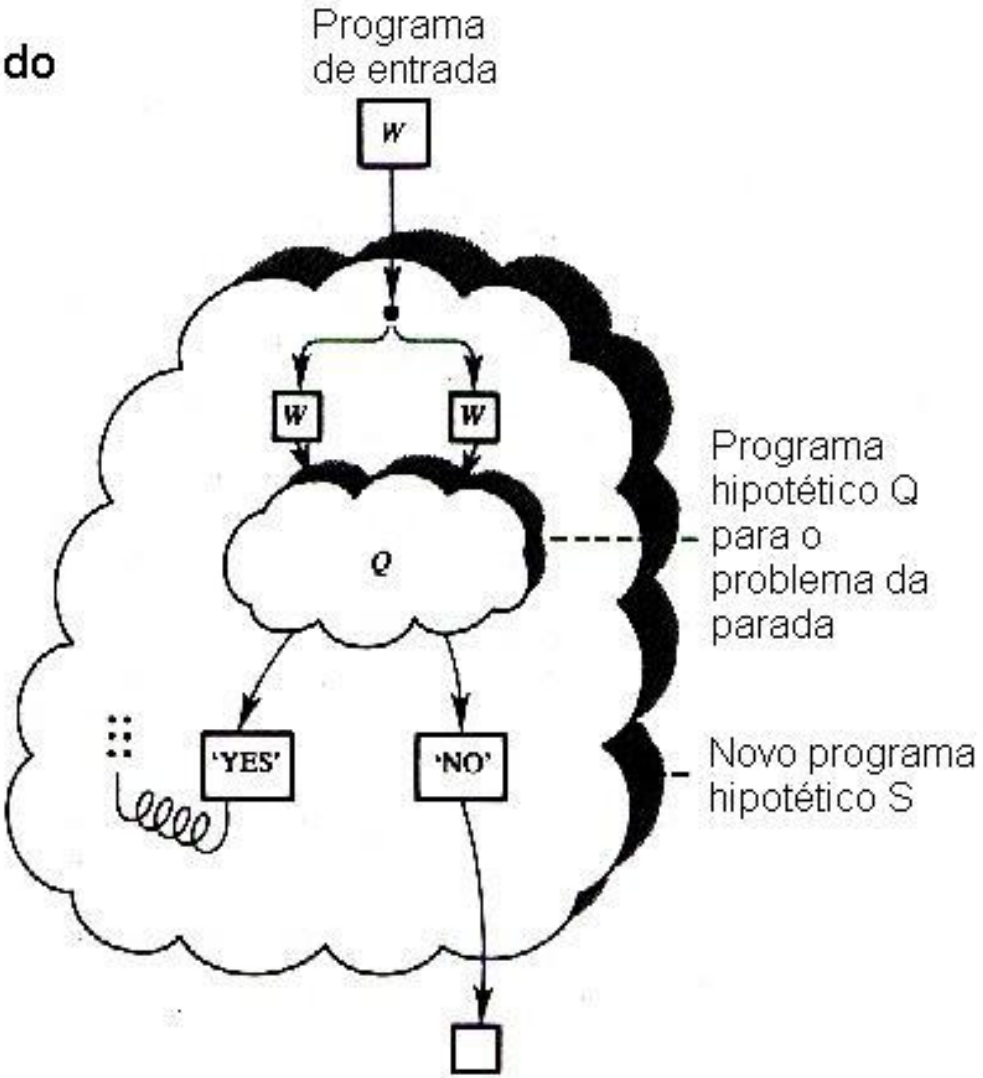
- Não existe nenhum programa em L que, aceitando qualquer par <R,X> consistindo de um texto de um programa R em L e uma cadeia de entrada X, termina após um tempo finito e responde "Sim" se R pára quando roda sobre a entrada X e "Não" se R não pára quando roda sobre a entrada X.

- Nós vamos provar a inexistência deste programa por contradição.

- Vamos assumir que tal programa existe, chamaremos ele de Q, e vamos derivar uma contradição desta afirmação.

- Já que Q existe podemos usar seu código para definir outros programas. Vamos fazer isto com o novo programa S.
  - S tem uma única entrada, um programa W escrito na linguagem L.
  - Após ler sua entrada, S faz uma cópia dela.
  - S então ativa Q com as duas cópias de W como entradas (como um compilador que compila a si próprio).
  - S espera Q terminar e, pela nossa hipótese Q deve terminar com "Sim" ou "Não".
  - Se Q diz "Sim" S entra em loop, se Q diz "Não" S pára.

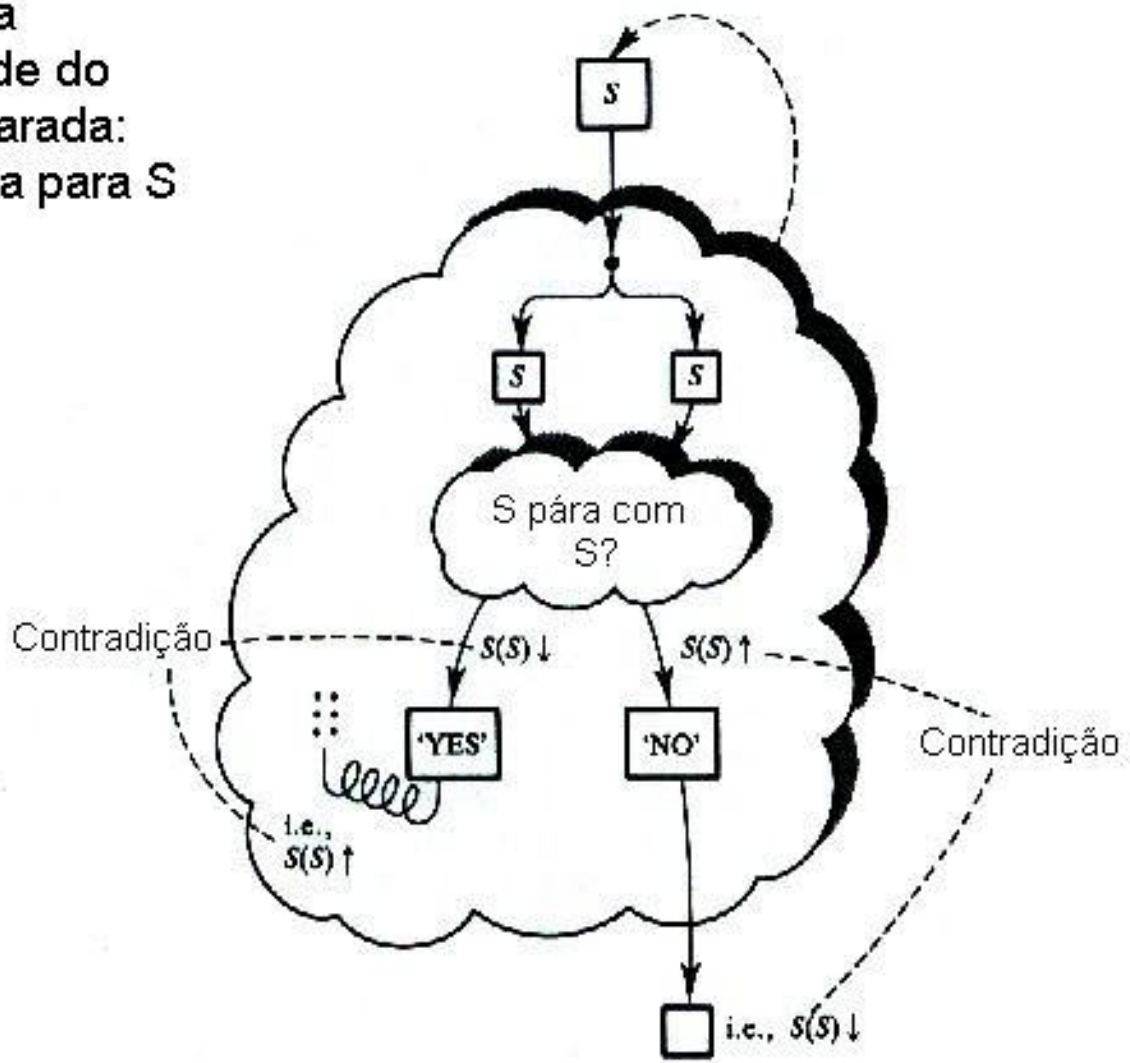
Provando a indecidibilidade do problema da parada: o programa S



- Vamos mostrar que existe algo errado com S.
- Lembre que para toda escolha de uma entrada legal W, S deve ou parar ou não.
- Vamos mostrar que existe um certo programa de entrada para o qual S não pode terminar, mas ele também não pode não terminar.
- Esta é uma contradição e usamos ela para provar que a nossa suposição da existência de Q é falsa, assim provando que o programa da parada é indecidível.

Quem é esta entrada para S????

Provando a indecidibilidade do problema da parada: S como entrada para S



- O programa S não pode terminar quando roda com ele mesmo e não pode não terminar.
- Algo está errado com S e, desde que, S foi construído de várias partes legais, a única parte problemática é o código do programa Q, cuja assumida existência nos levou a construir S.

A conclusão é que o programa Q que resolve o programa da parada não pode existir.

2) Provando a Indecidibilidade pelo Método da Diagonalização

- Embora a prova por contradição seja uma prova matemática rigorosa, muitas pessoas ficam desconfortáveis com ela.

- Porém, a natureza da referência própria subjacente à prova está incorporada em um método fundamental de prova usado por Cantor, um matemático do século 19.

- Esta técnica é chamada de diagonalização e será usada para provar a indecidibilidade do problema da parada.

- A prova pode ser visualizada imaginando uma tabela infinita (Tabela 1) com todos os programas na linguagem L (vertical) versus todas as entradas possíveis (horizontal). Para simplificar, vamos considerar as entradas com inteiros.

  - Na junção da i-ésima linha com a j-ésima coluna temos indicado se o i-ésimo programa pára com j ou não.
  - Assim, a i-ésima linha é a descrição completa (infinita) da informação sobre a parada do i-ésimo programa de L.

A prova da indecidibilidade vista como diagonalização

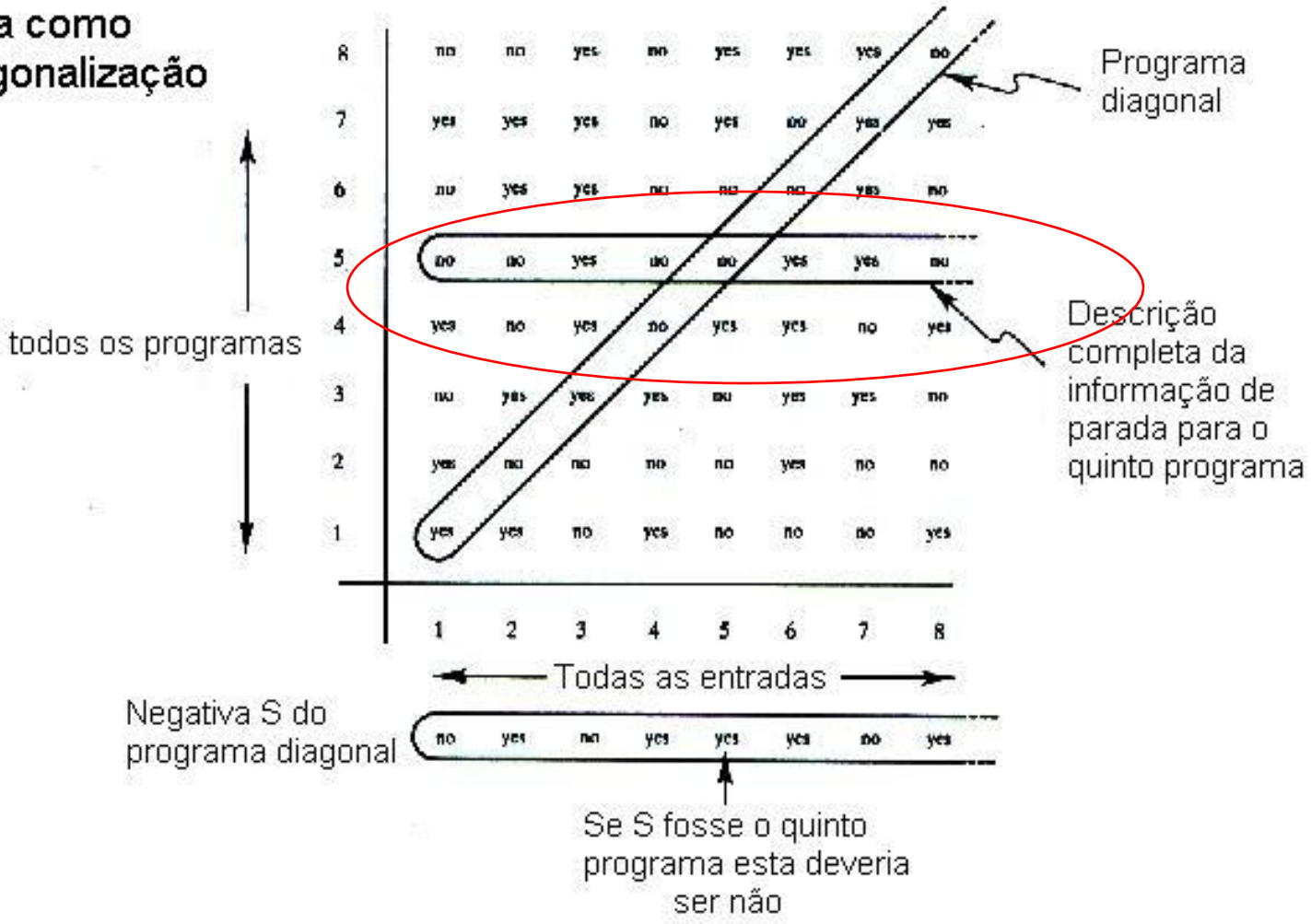


Tabela 1

- Vamos construir um programa imaginário similar ao programa S e também vamos chamá-lo de S.
  - O comportamento de parada de S é a negativa da linha diagonal da Tabela 1.

- Com este cenário é fácil provar que o problema da parada é indecidível.

- Assuma que nós podemos decidir com um programa Q se um dado programa em L pára sobre uma dada entrada.

- O programa S fica assim:
  - dado J como entrada, ele iria na lista de programas, encontraria o j-ésimo e submeteria o programa e a entrada J ao programa Q que implementa o problema da parada.

- S funciona como já o descrevemos e, assim, S se comporta como a negativa da diagonal da Tabela 1.

- Isto leva a uma contradição, pois desde que S é um programa em L ele deveria estar na lista vertical (Y), mas ele não pode.

- Por construção, se S é o quinto programa, por exemplo, ele não pode parar com a entrada 5 (existe um "no"), todavia S pára o que é uma contradição.

- É impossível S ser um programa de L pois ele se comporta diferente para uma entrada de cada programa em L.

A prova da indecidibilidade vista como diagonalização

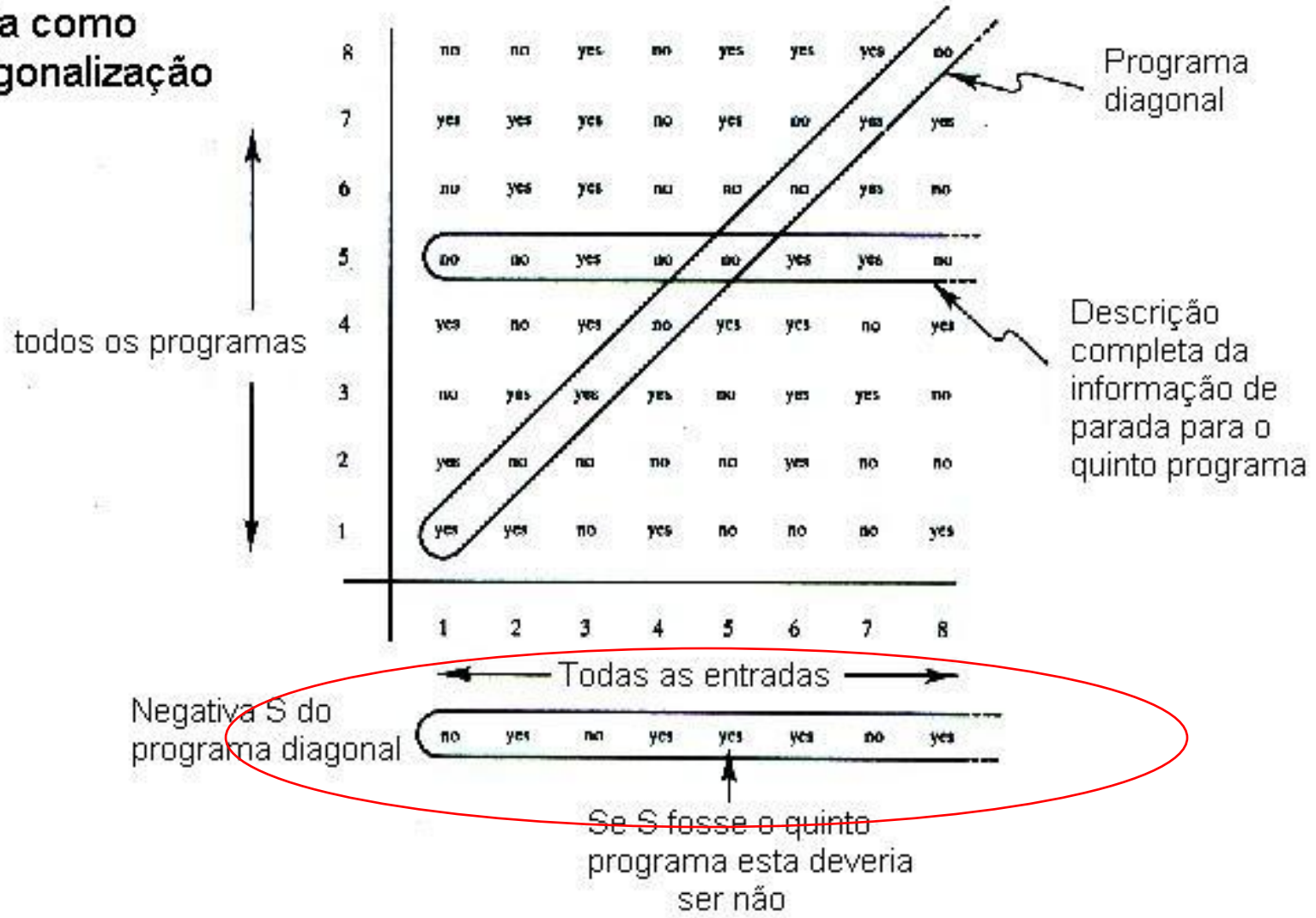
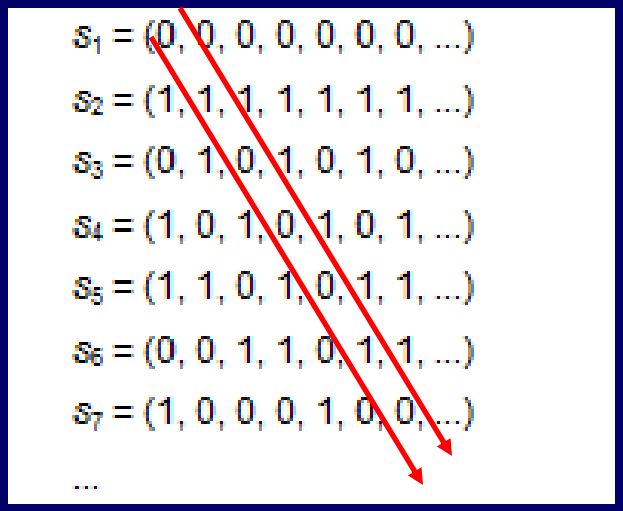


Tabela 1

Entendendo o argumento da diagonalização – conjuntos incontáveis

- Existem infinitos conjuntos que não são contáveis. E provamos isto pelo argumento da diagonalização. Conjunto dos Reais é um deles!

- Outro conjunto: Seja uma sequencia infinita S da forma (s_1, s_2, s_3, ...) onde cada elemento s_i é uma sequencia infinita de 1´s ou 0´s.
- Esta sequencia é contável, pois cada natural n é associado com um e somente um elemento da sequencia.
- Seja a sequencia:



$$s_0 = (\underline{1}, \underline{0}, \underline{1}, \underline{1}, \underline{1}, \underline{0}, \underline{1}, \ldots)$$

- Para cada m e n seja $s_{n,m}$ o $m^{ésimo}$ elemento da $n^{ésima}$ sequencia na lista.
- Por exemplo: $s_{2,1}$ é o primeiro elemento da segunda sequencia.
  - É possivel construir uma sequencia s_0 em que seu primeiro elemento é diferente do primeiro elemento da primeira sequencia e o segundo diferente do segundo da segunda sequencia e assim vai... :

$$s_{0,n} \neq s_{n,n}.$$

- Esta nova sequencia s_0 é diferente de toda sequencia na lista

- Em resumo: pela sua definição s_0 não está contida na sequencia contável S.

- Seja T um conjunto de todas as sequencias infinitas de 0s e 1s. Pela sua definição este conjunto deve conter s_0, que é apenas mais uma sequencia de 0s e 1s.
  - Entretanto, s_0 não aparece em S. E assim T não coincide com S.

- Este argumento se aplica a todo conjunto S de sequencias de 0s e 1s,
  - então concluimos que T não pode ser contável. *T* é incontável, pois não pode se colocar em correspondência com os naturais.

  - Diagonalização mostra que dada uma lista S, contável, de elementos, sempre existirá um elemento x de S que não ocorre em s1,s2,...

O Paradoxo de Russell

- Pergunta: um conjunto pode ser elemento dele próprio?

- O conjunto de todos os inteiros não é um inteiro.
- O conjunto de todos os gorilas não é um gorila.

- MAS: o conjunto de todas as idéias abstratas é uma idéia abstrata.

- Assim, se permitimos que um conjunto seja elemento dele próprio,
  - conseguimos criar vários paradoxos na teoria dos conjuntos, como mostra Russel.

Russell's Paradox: Let A be the collection of all sets that are not elements of themselves (like integers or gorillas). In formal set notation: $A = \{ S \mid S \notin S \}$. Now, we can ask, is A an element of this set (i.e., is $A \in A$?). The answer, tragically, is yes and no. If A is an element of this set, then by the definition of A, it is impossible for A to belong to A (since A cannot be an element of itself). On the other hand, if A is not an element of this set, then A must be an element of A (according to the definition of the set). Neither case can be true, which causes a contradiction.

http://www.stanford.edu/class/cs103/handouts/18%20Infinite%20Sets%20and%20Countability.pdf

Outra visão do problema

- O "**Paradoxo da Referência Própria**" ocorre quando nós forçamos uma MT S entrar em desacordo com ela mesma.

- Por um lado, S sabe o que acontecerá na entrada <S>, mas então decide fazer outra coisa.

- *" Você não pode ter certeza o que você fará no futuro, porque você pode decidir mudar suas ações e criar um paradoxo."*

Esqueçam a Máquina do Tempo!

Um problema não reconhecível por MT

- Vimos que A_{TM} não é decidível por MT, mas é reconhecível por MT.

$$A_{TM} = \{\langle M,w\rangle \mid M \text{ é uma MT que aceita } w\}$$

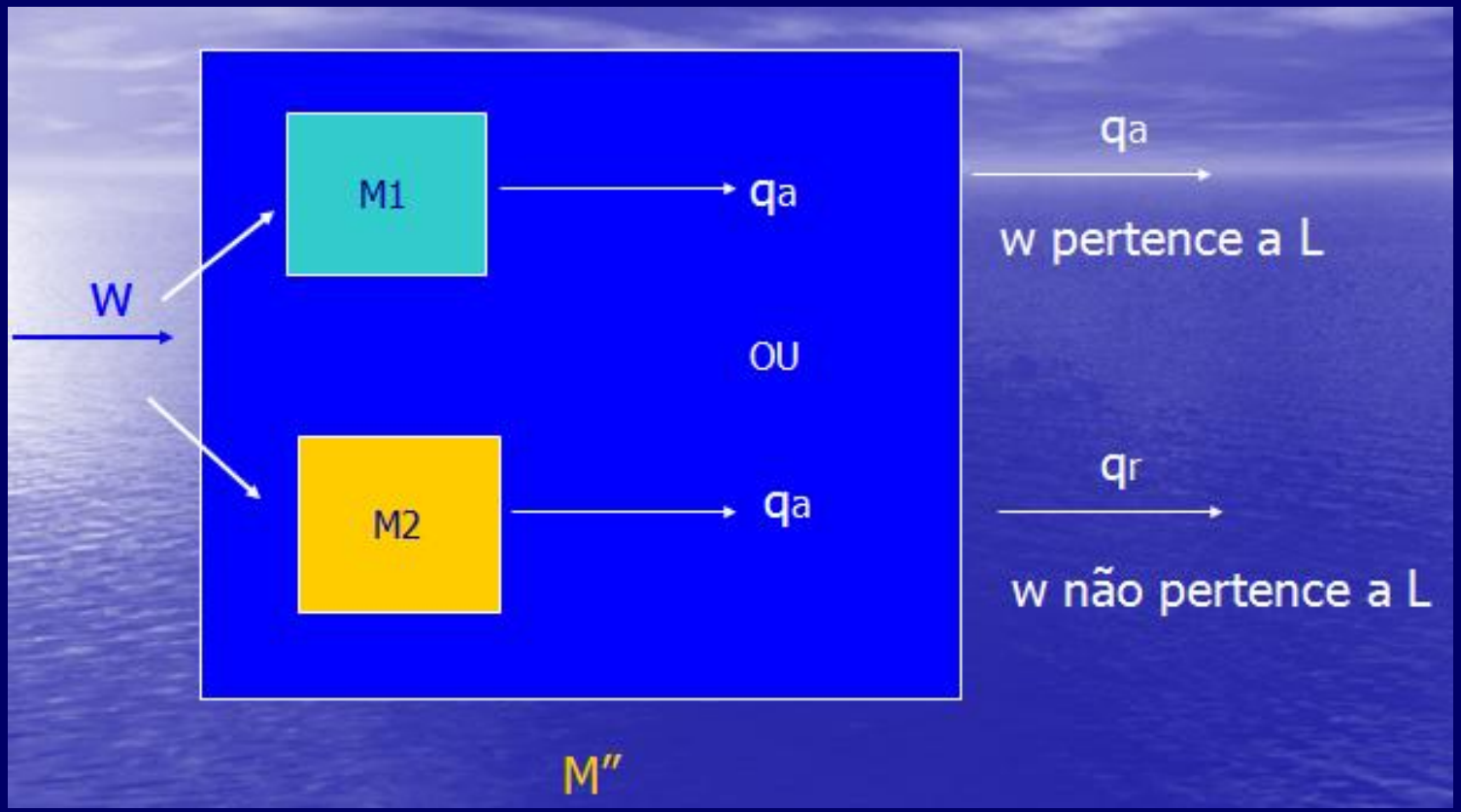
- Veremos uma linguagem que não é reconhecível por MT.

Teorema 4.16 (Sipser): Se uma linguagem A is reconhecível e seu complemento $\bar{A}$ é reconhecível, então A é decidível.

(complemento de L é a linguagem consistindo de todas as cadeias que não pertencem à linguagem)

Prova: Rode MTs for A e $\bar{A}$ em paralelo sobre a entrada x.
Espere por uma das MT aceitarem.
Se a MT para A aceitou: "aceite x";
Se a MT para $\bar{A}$ aceitou: "rejeite x".

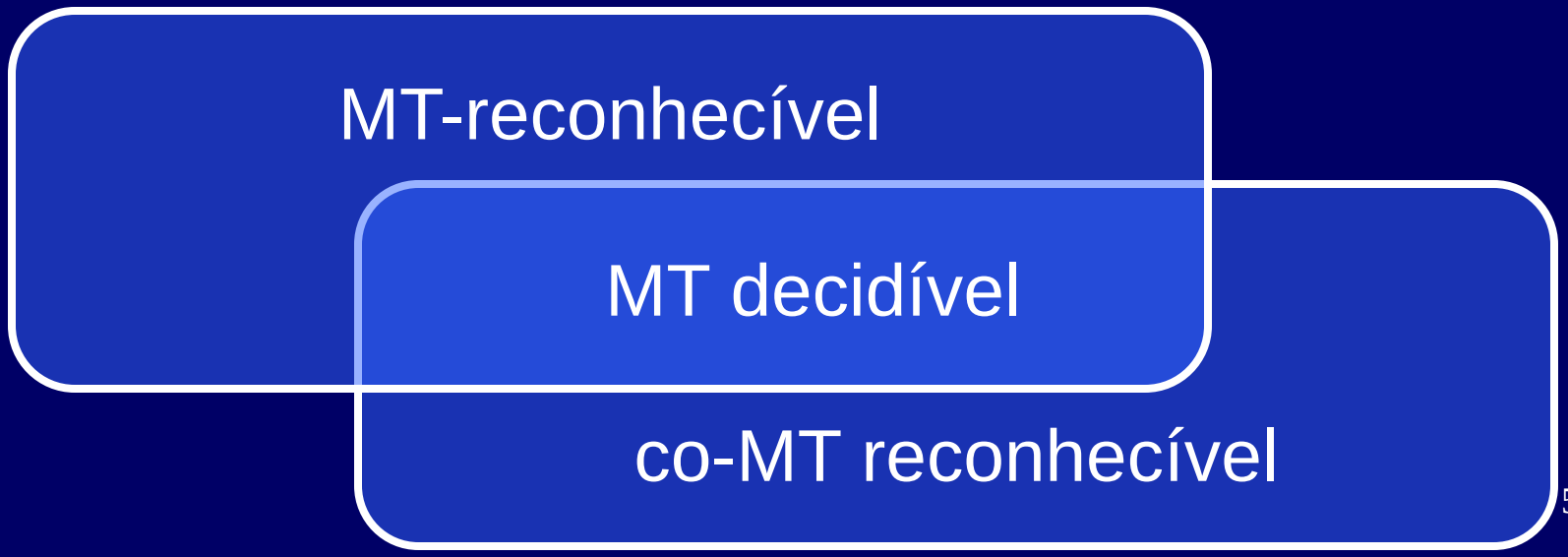
Prova: Rode MTs for A e $\bar{A}$ em paralelo sobre a entrada x.
Espere por uma das MT aceitarem.
Se a MT para A aceitou: "aceite x";
Se a MT para $\bar{A}$ aceitou: "rejeite x".



$\bar{A}_{TM}$ não é MT-Reconhecível

Pelo resultado anterior $\bar{A}_{TM}$ não pode ser MT-reconhecível, porque isto implicaria que A_{TM} seria decidível.

Chamamos linguagens como $\bar{A}_{TM}$ de co-MT reconhecível



Problemas Indecidíveis relacionados a MTUs

- Problema da Totalidade (variação do problema da parada para qualquer entrada):

  Dada uma máquina universal M qualquer (Turing, Norma, Post), existe um algoritmo que verifique se M pára, aceitando ou rejeitando, ao processar qualquer entrada? Verifica se M é um algoritmo!

- Problema da Equivalência:

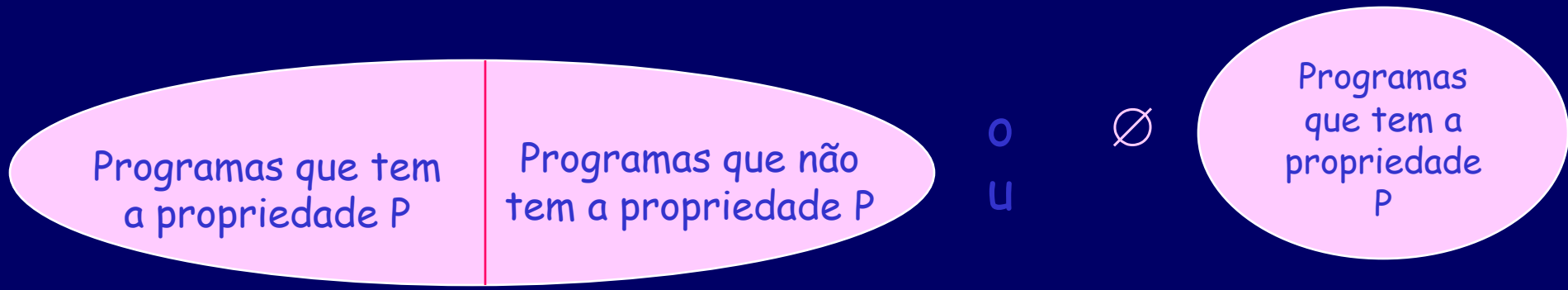
  Dadas duas MTU M1 e M2 L(M1) = L(M2)?

3) Usando o Teorema de Rice para Provar Indecidibilidade

- Este teorema mostra que a indecidibilidade do Problema da Parada não é um fenômeno isolado!

- Teo: Se A é uma propriedade extensiva não-trivial de programas, então A é indecidível.

- Uma propriedade de programas é especificada pela divisão do mundo dos programas em duas partes:



- Uma propriedade não-trivial é uma que é satisfeita por pelo menos um, mas não todos os programas.

- Uma propriedade extensiva (externa) depende exclusivamente do comportamento de entrada e saída do programa,
  - e assim é independente da aparência, tamanho e outras propriedades chamadas de intensivas (interna).

- Exemplo: autoria, local e data da publicação são critérios externos de um TEXTO, sendo que o assunto e o tipo de texto são critérios internos.

- Prop P1: O número de variáveis no programa P é maior que 100?
- Prop P2: Uma MT M tem mais que 7 estados?
  - Vejam que nesses 2 casos falo da máquina propriamente dita.

- Prop P3: Uma MT M roda em tempo polinomial?

(aqui falo do comportamento e é não-trivial: pois existem linguagens decidíveis por MT em tempo exponencial de vários graus e outras nem sabemos se estão em P)

Portanto, não existem Checadores de Complexidade!!!

Vejam que aqui falo de comportamento de um algoritmo.

Quase todas as propriedades da linguagem de MT são indecidíveis:

$Regular_{TM}$ = { <M> | L(M) é uma linguagem regular }

$Finita_{TM}$ = { <M> | L(M) é uma linguagem finita }

LC_{TM} = { <M> | L(M) é uma linguagem LC }

Indecidibilidade + Rice + Tese de Church-Turing

Linguagens e Problemas: qualquer problema pode ser colocado como um problema de reconhecimento de linguagem

Indecidibilidade: linguagens indecidíveis que não podem ser decididas por nenhuma MT

Teorema de RICE: toda propriedade não trivial sobre a linguagem da MT são indecidíveis

Tese de Church-Turing: qualquer computação mecânica pode ser feita por uma MT

Conclusão: qualquer propriedade não trivial sobre computações mecânicas não podem ser decididas !

Teorema de Rice nos diz que não podemos predizer os aspectos interessantes do comportamento de um algoritmo de forma algorítmica!

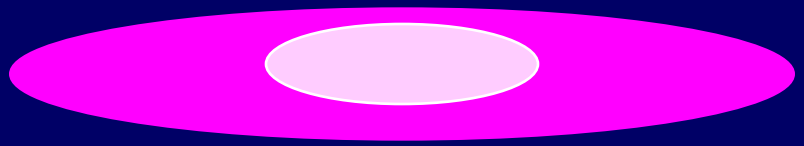
4) Provando a Indecidibilidade pelo Método da Redução

- Princípio da Redução: consiste em se verificar a computabilidade de um problema a partir de outro cuja de computabilidade já é conhecida.
- Redução é uma atividade do dia a dia:
- O problema de viajar de São Paulo para Paris se reduz

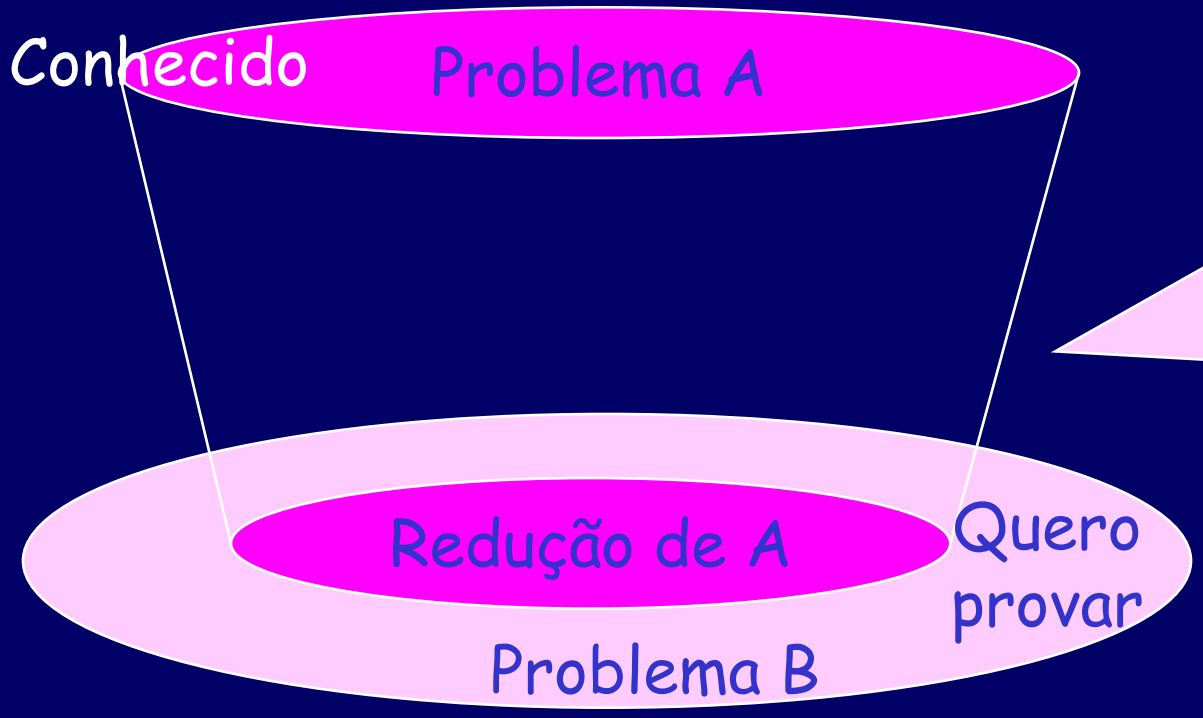
  ao problema de comprar uma passagem de avião entre as duas cidades que se reduz

  Ao problema de ganhar dinheiro para a passagem que se reduz

  Ao problema de encontrar um trabalho !



- Sejam A e B dois problemas de decisão. Suponha que é possível modificar (reduzir) o problema A de tal forma que ele se porte como um caso do problema B.
  - Se A é indecidível, então como A é um caso de B, conclui-se que B também é indecidível.
  - Se B é decidível então então como A é um caso de B, conclui-se que A é também decidível.



Para provar que P é indecidível: mostre que algum outro problema indecidível conhecido se reduz a ele.

Teo 5.10 Máquinas de Redução (Divério&Menezes, 99)

Suponha dois problemas A e B e suas linguagens LA (conhecido), LB. Uma máquina de redução R de LA para LB é tal que:

a) Se $w \in LA$ então $R(w) \in LB$
b) Se $w \notin LA$ então $R(w) \notin LB$

O mapeamento é uma função total

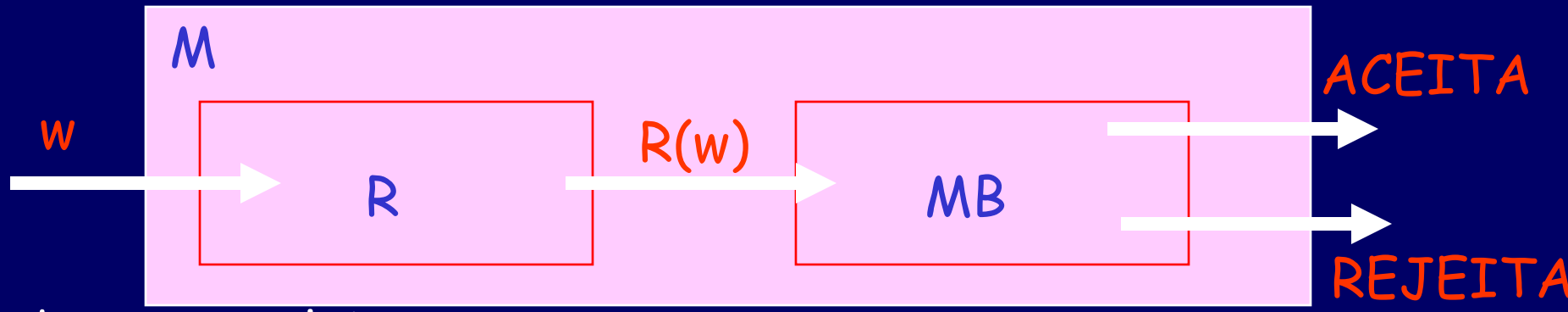
Resultados do Princípio da Redução

a) Se LB é recursiva então LA é recursiva
b) Se LB é RE então LA é RE

c) Se LA não é recursiva então LB não é recursiva
d) Se LA não é RE então LB não é RE

Provas

a) Suponha que LB é recursiva. Então existe uma MTU MB que aceita LB e sempre pára para qq entrada:



Podemos concluir:

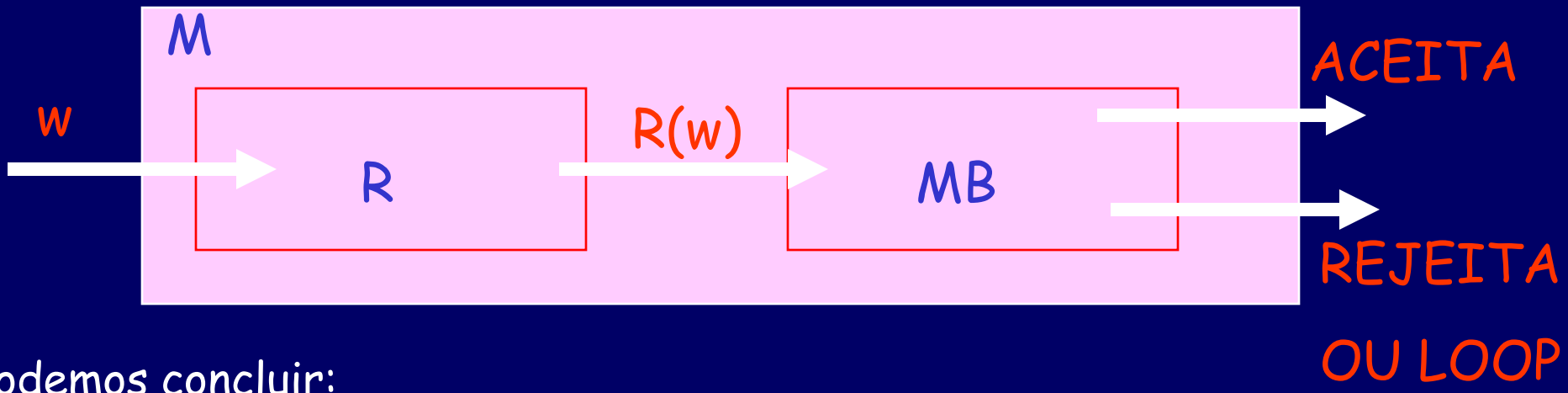
- M sempre pára para qq entrada, pois R e MB sempre param;
- Se w $\in$ LA então M aceita w, pois R(w) $\in$ LB
- Se w $\notin$ LA então M rejeita w pois R(w) $\notin$ LB

Portanto, M aceita LA e sempre pára para qq entrada. Logo LA é recursiva. Dizemos que M decide LA.

b) Suponha que LB é RE. Então existe uma MTU MB tal que:

ACEITA(MB) = LB

REJEITA(MB) $\cup$ LOOP(MB) = Σ^* - LB. Seja M:



Podemos concluir:

- Se w $\in$ LA então M aceita w, pois R(w) $\in$ LB
- Se w $\notin$ LA então M rejeita ou fica em loop para w pois MB rejeita ou fica em loop para R(w)

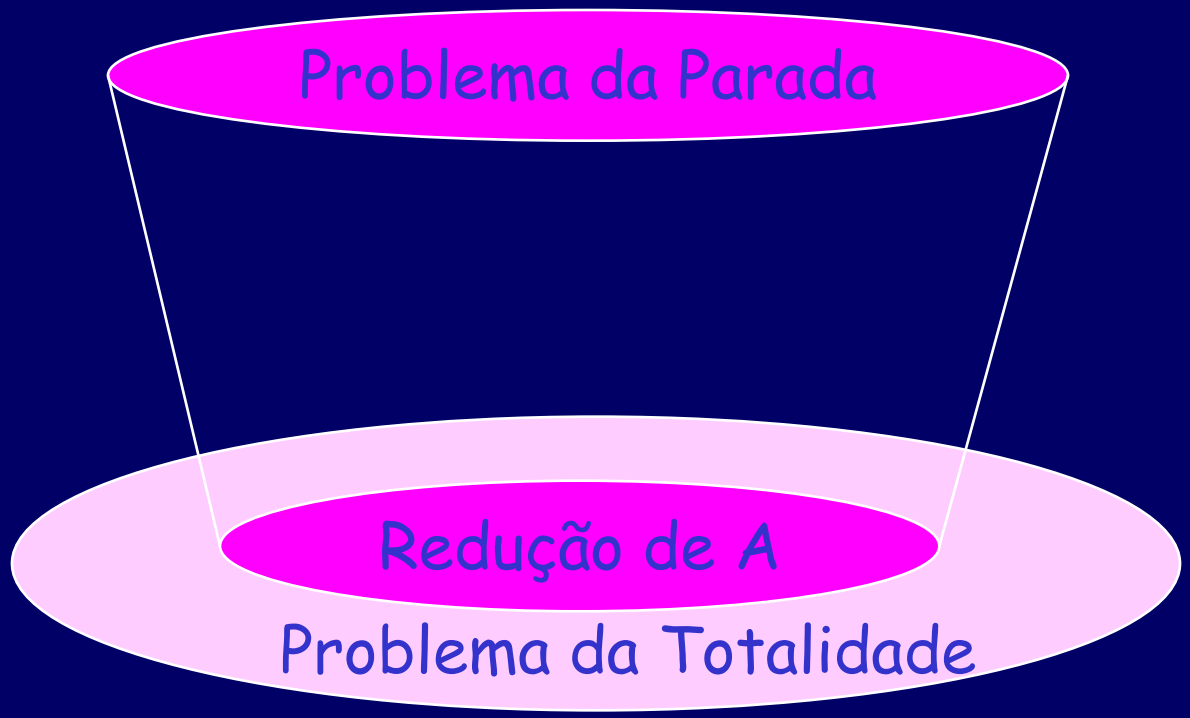
Portanto, M aceita LA mas pode ficar em loop para entradas não pertencentes a LA. Logo LA é RE. Dizemos que M reconhece LA.

c) e d) Por contraposição, as afirmações c) e d) são equivalentes às afirmações a) e b), respectivamente, pois

$$(p \rightarrow q) \Leftrightarrow (\neg q \rightarrow \neg p)$$

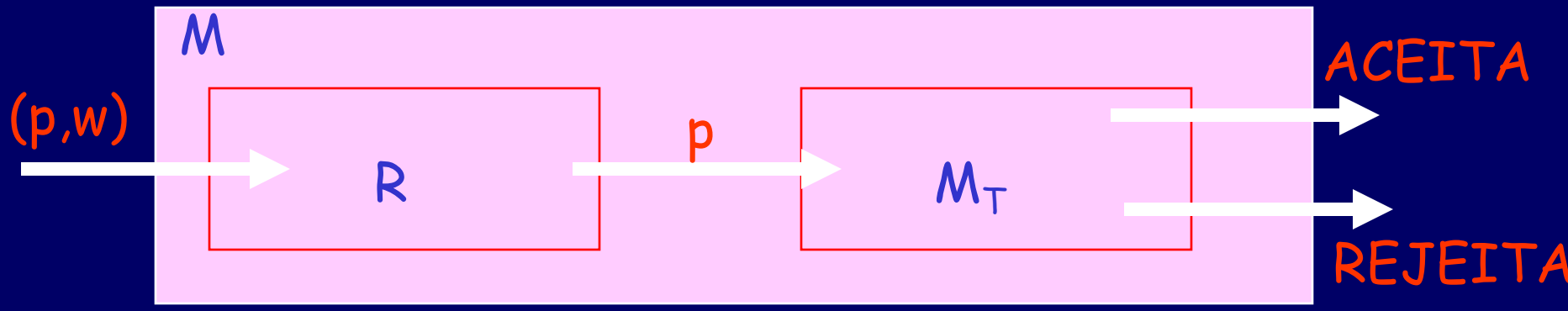
Teo: Problema da Totalidade é indecidível

L_T ={m | m = código(M) que é uma MT que sempre pára, aceitando ou rejeitando qq entrada}



Prova

- Suponha que L_T é recursiva. Então existe uma MTU M_T que sempre pára e Aceita(M_T) = L_T.
- Suponha uma MTU R que toma (p,w) e gera a projeção da primeira componente.
- Seja M a seguinte MTU Parada.
- Parada foi reduzido a Totalidade pois:
- Se $(p,w) \in L_P$ então $R((p,w)) = p \in L_T$
- Se $(p,w) \notin L_P$ então $R((p,w)) = p \notin L_T$



Como foi suposto que Totalidade é decidível então Parada é também uma linguagem recursiva, o que é absurdo!

Logo Totalidade não é decidível.

Problemas indecidíves para Ling rec enumeráveis L

- L é vazia?
- L é finita?
- L contém 2 cadeias diferentes de mesmo tamanho?
- Todos estes problemas são indecidíveis

L contém 2 cadeias diferentes de mesmo tamanho?

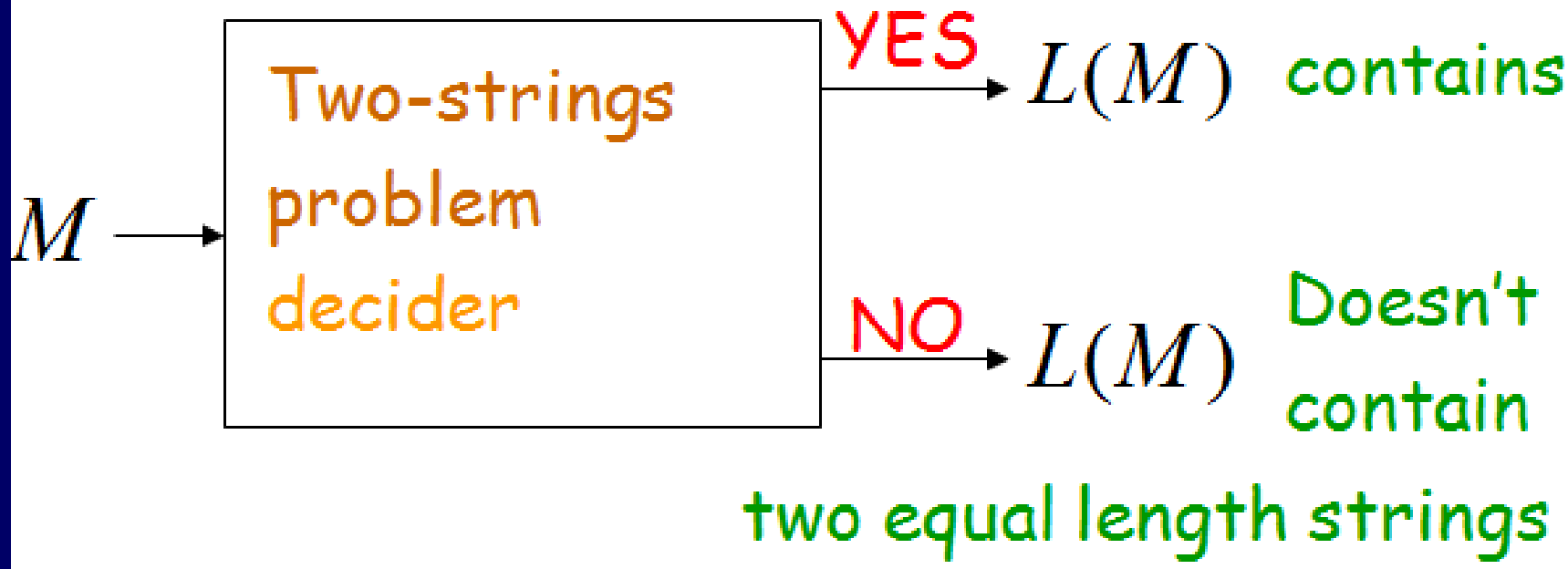
Theorem:

For a recursively enumerable language L it is undecidable to determine whether L contains two different strings of same length

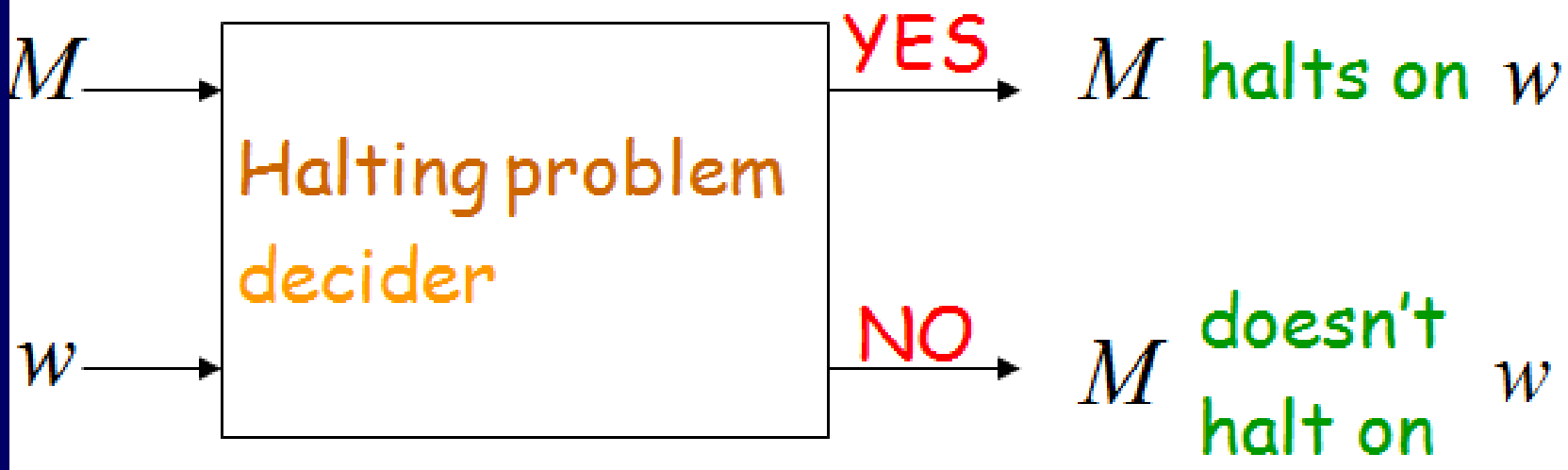
Proof: We will reduce the halting problem to this problem

Let M be the TM with $L(M) = L$

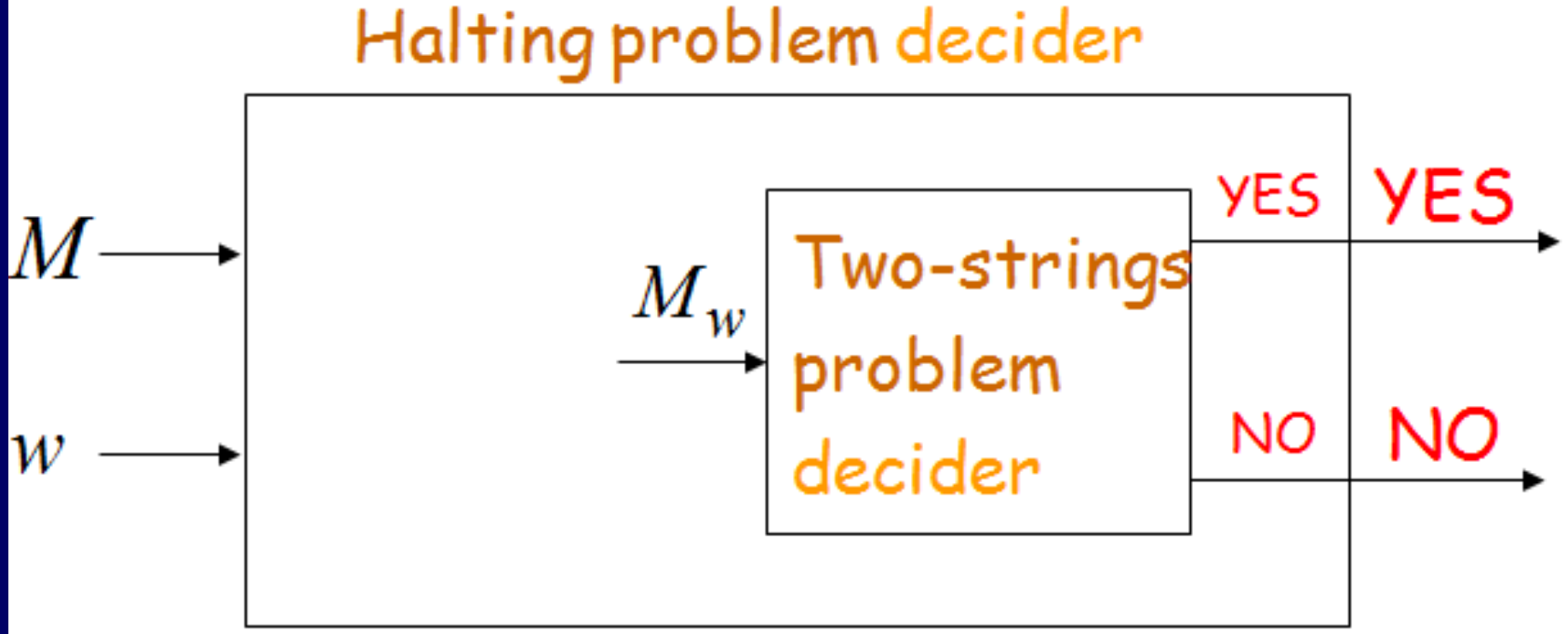
Suppose we have the decider
for the two-strings problem:



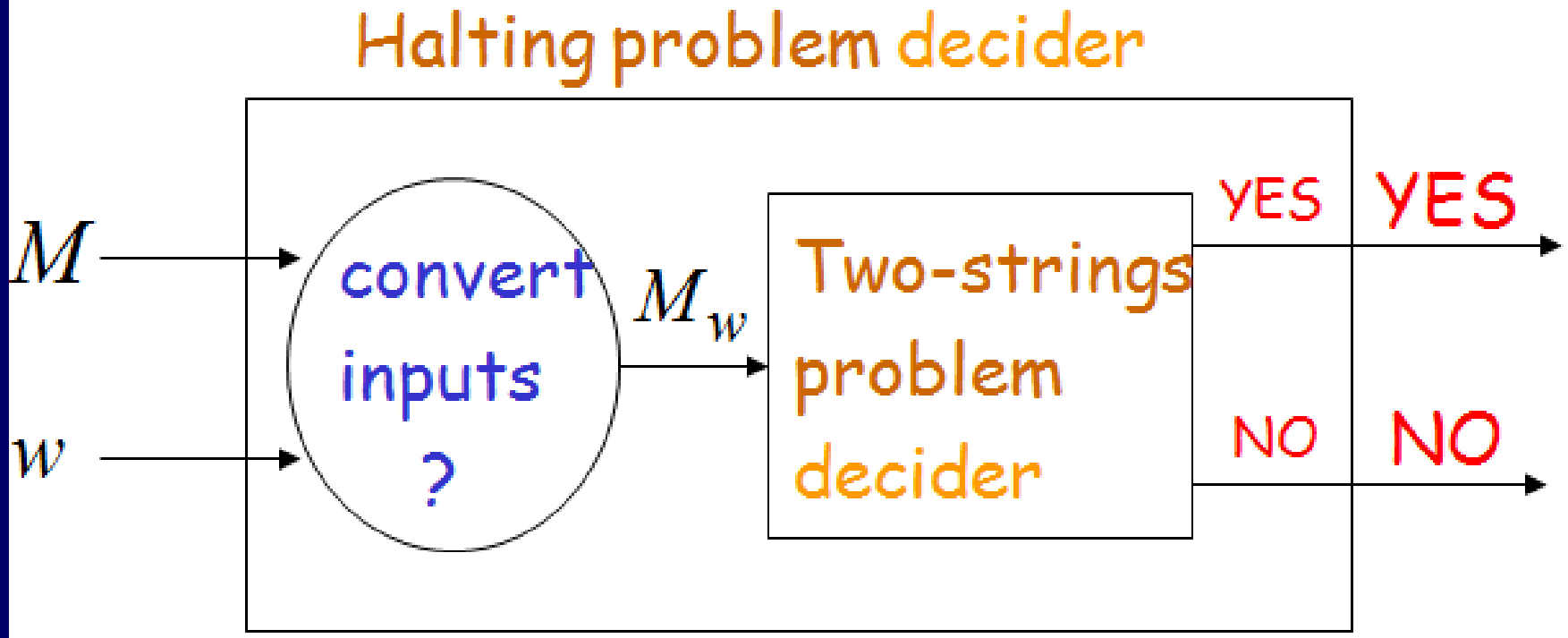
We will build a decider for the halting problem:



We want to reduce the halting problem to the two-strings problem



We need to convert one problem instance to the other problem instance



Construct machine M_w :

On arbitrary input string s

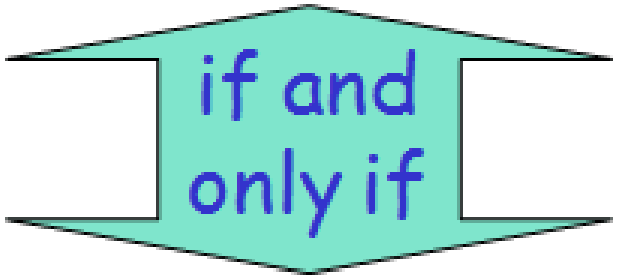
Initially, simulate M on input w

When M enters a halt state,
accept if $s = a$ or $s = b$

(two equal length strings $L(M_w) = \{a, b\}$)

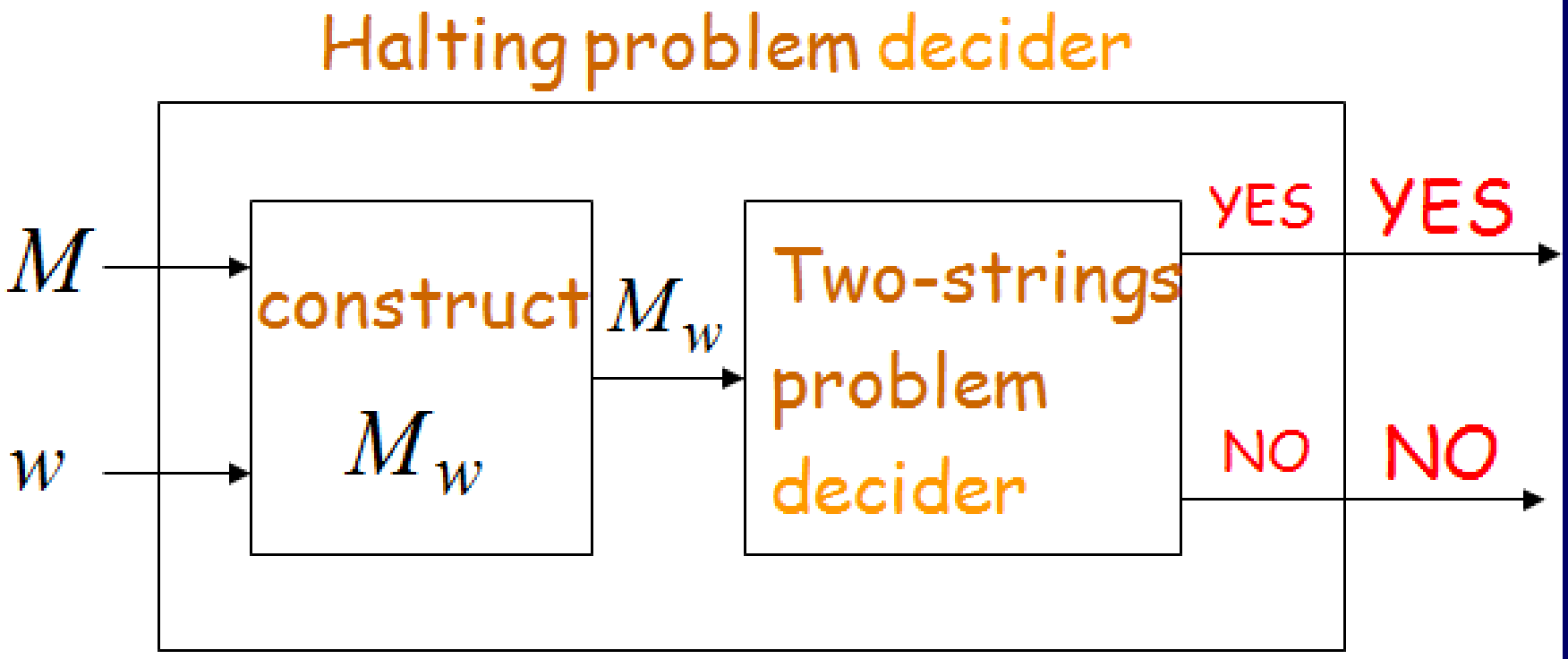
Otherwise, reject s ($L(M_w) = \varnothing$)

M halts on w



M_w accepts two equal length strings

M_w accepts a and b



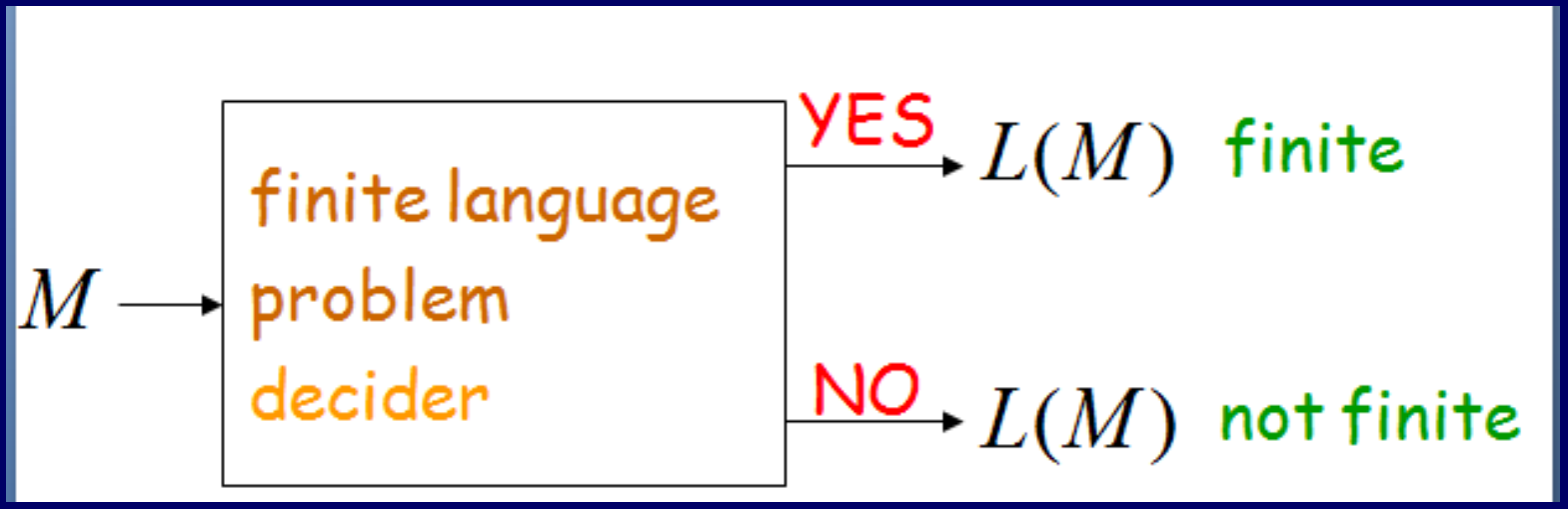
Como foi suposto que Linguagem contém 2 cadeias de mesmo tamanho é decidível então Parada é também uma linguagem recursiva, o que é absurdo!

Logo Linguagem contém 2 cadeias de mesmo tamanho não é decidível.

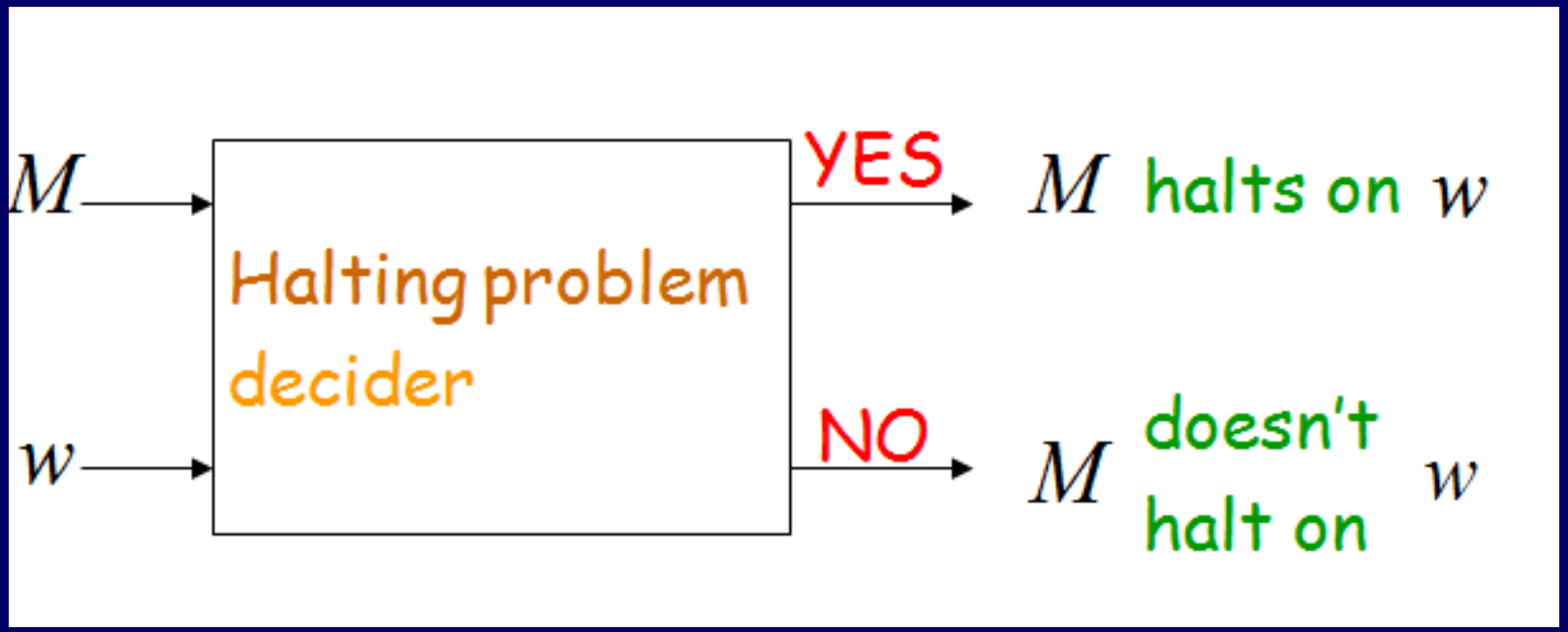
$Finita_{TM}$ = { <M> | L(M) é uma linguagem finita}

- Prova: reduzir o problema da parada para este problema

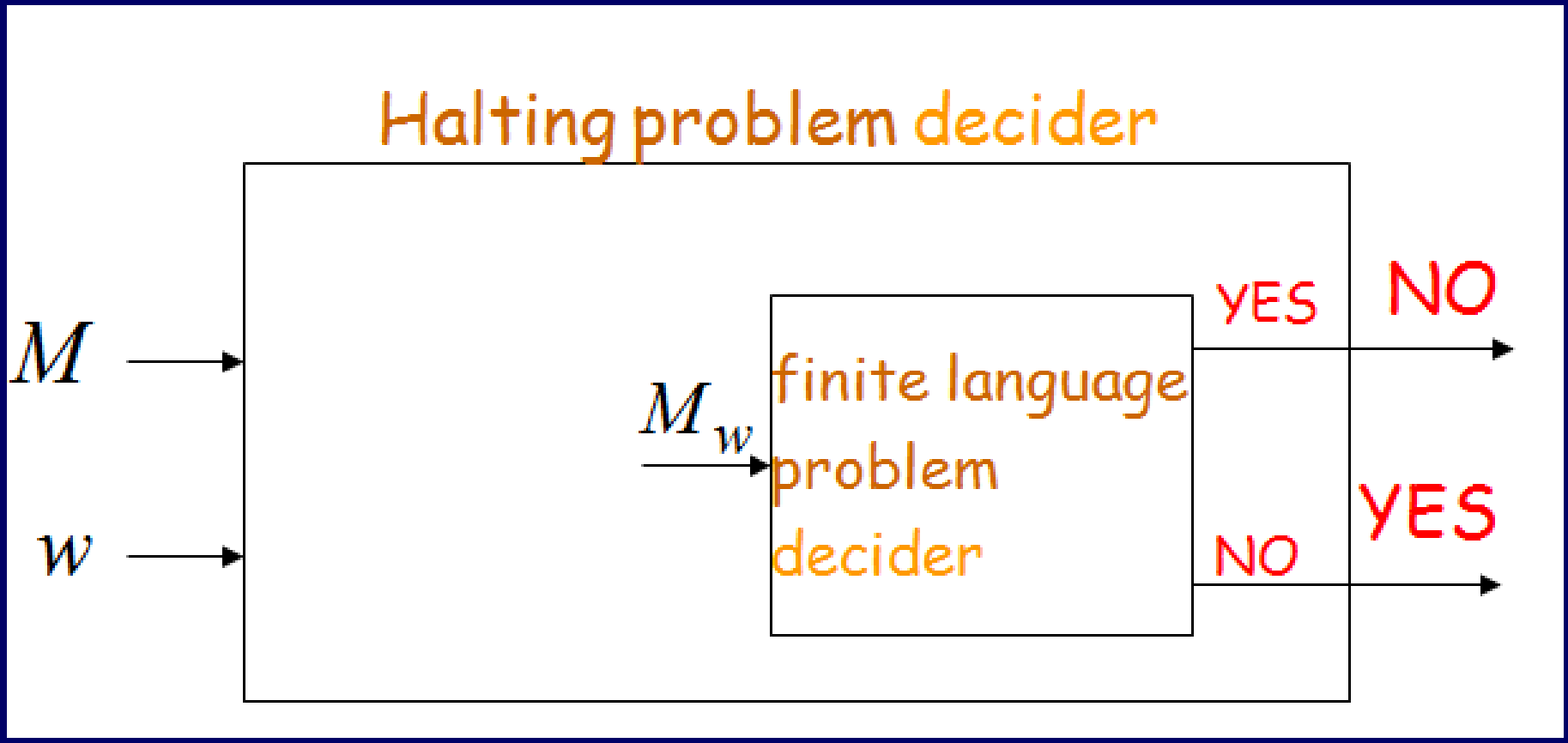
- Seja M a MT com L(M) = L
- Suponha que temos um algoritmo para o problema da linguagem finita



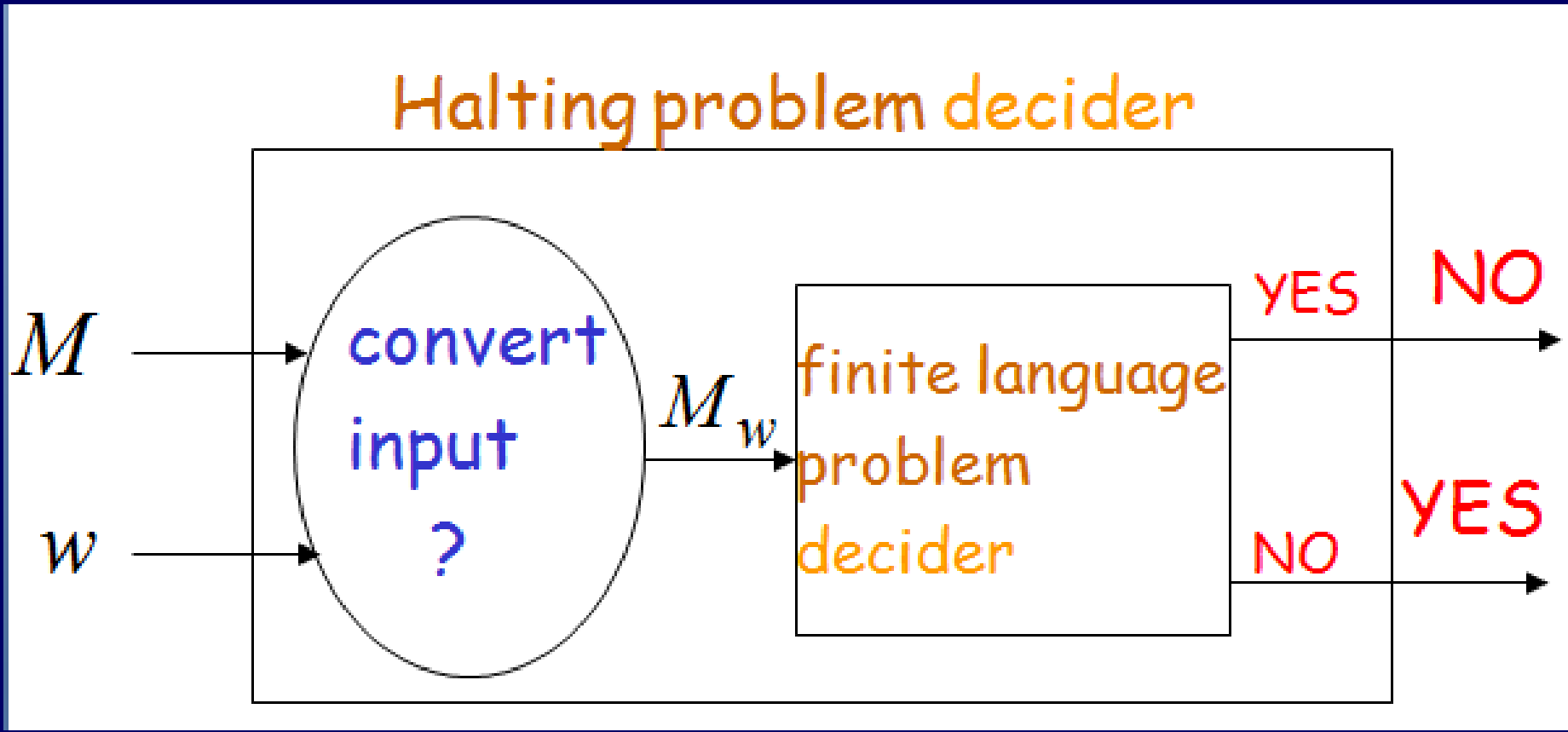
Vamos construir um algoritmo para o problema da parada



Desejamos reduzir Parada para o problema da Linguagem Finita



Precisamos converter uma instância para outra



Construct machine M_w:

On arbitrary input string s

Initially, simulates M on input w

If M enters a halt state,
accept s (Σ^* inifinite language)

ACEITA TUDO!

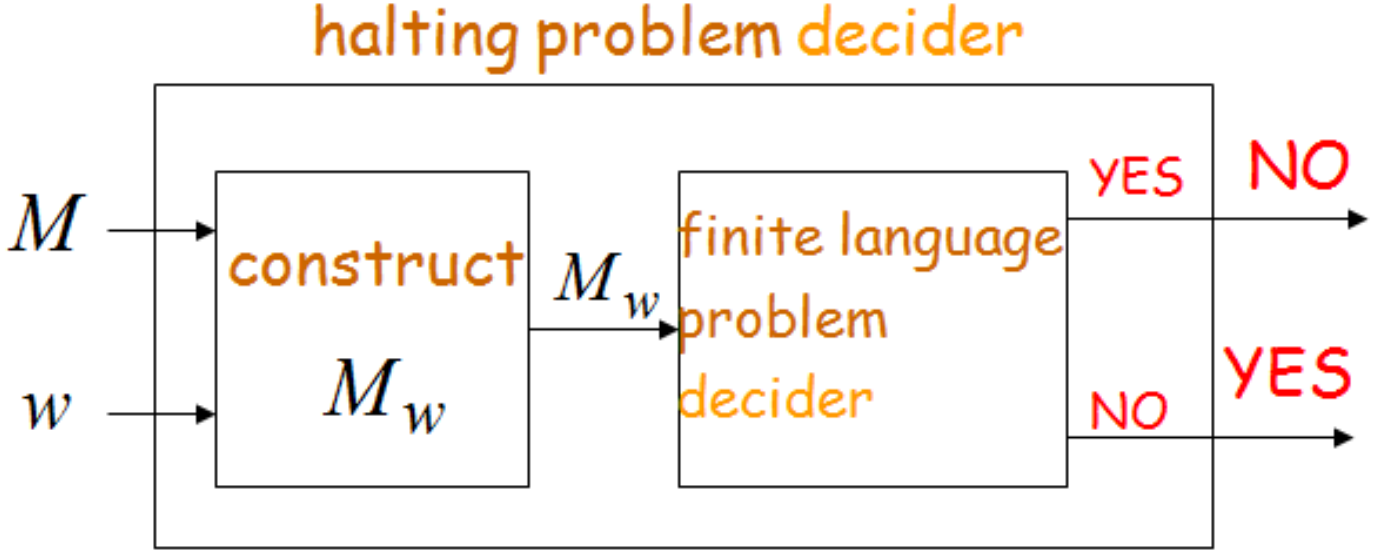
Otherwise, reject s ($\varnothing$ finite language)

M halts on w

if and only if

$L(M_w)$ is infinite

$L(M_w) = \Sigma^*$



Como foi suposto que Linguagem Finita é decidível então Parada é também uma linguagem recursiva, o que é absurdo!

Logo Linguagem Finita não é decidível.

Fontes para pesquisa

- Pesquisem em:
  - cap 4 e 5 do Sipser
  - cap 9 do (H,M,U, 2001)
  - cap 5 do (Divério e Menezes, 1999 – ou edições mais atuais deste)
  - Cap 8 e 9 do Harel, 1992 (segunda edição)

Exemplos de problemas indecidíveis

- PCP (Post Correspondence Problem)
- Azulejamento
- Equivalência de programas (ou de 2 MT)
- Busy Beaver
- Validade do Cálculo de Predicados
- Ambigüidade de GLC
- Se 2 GLC são equivalentes
- Propriedade da linguagem da MT ser: Regular, Finita, Livre de Contexto