Problemas Algoritmicos e Indecidibilidade

O que pode ser computado?

Chegamos a um importante ponto do curso. Vamos agora estudar uma das questões mais fundamentais em Ciência da Computação:

Qual seria o limite do poder computacional de uma máquina?

Fato importante: É possível dar exemplos precisos de problemas que estão além da capacidade de computação.

Problemas de Decisão

Um ***problema de decisão*** é uma questão com um conjunto finito de parâmetros e que, para valores específicos desses parâmetros, tem resposta sim ou não.

EX: - A soma de 3 e 5 é igual a 8?

- A soma de x e y é igual a z?

O primeiro problema acima não tem parâmetros (e, portanto, tem um única ***instância***)

O segundo problema tem 3 parâmetros e tem infinitas ***instâncias*** , cada uma correspondente a uma tripla de valores para *x, y* e *z.*

- Um PD pode ser visto como um conjunto de questões, uma para cada possível combinação de valores dos parâmetros do problema.
- Cada uma dessas questões tem resposta sim ou não, e é chamada uma instância do problema.

Problemas de Decisão

Do ponto de vista da Ciência da Computação, qualquer problema cujo conjunto de instâncias é finito não tem muito interesse, pois pode sempre ser resolvido por meio de um algoritmo que simplesmente obtém a resposta para cada instância do problema diretamente de uma tabela que armazena a solução para cada uma das instâncias.

Problemas de Decisão como Linguagens

Para estudar o que significa um problema de decisão ser solúvel computacionalmente, precisamos de um método para padronizar tais problemas e torná-los fáceis de serem entendidos por um computador. A solução é representar cada instância como um string:

Ex: Uma instância para o problema “z é a soma de x e y”, seria o string “2+2=4?”

Q: Podemos agora definir teoricamente o que significa um problema ser solúvel computacionalmente. Como você definiria isso?

Problemas de Decisão como Linguagens

A: Um problema de decisão **P** é solúvel se existe um programa de computador, cujas possíveis entradas são instâncias desse problema (codificadas como strings), e que sempre pára, produzindo a resposta correta para cada instância. Além disso, a codificação das instâncias deve ser, ela própria, computável.

Em teoria de linguagens: Seja Σ um alfabeto no qual podem ser codificadas as instâncias de **P**. Seja *E* o conjunto das instâncias (codificadas) e *L* o conjunto de instâncias cuja resposta é SIM. Então **P** é dito ***decidível*** (ou solúvel computacionalmente) se *L* é decidível.

Problemas de Decisão
Problemas Básicos

Antes de atacar alguns problemas de decisão fundamentais em Computação, vamos considerar se existem algoritmos para…

- …determinar se um dado string pertence a uma dada linguagem, para linguagens de uma certa classe **C,** como linguagens aceitas por DFA's, NFA's, PDA's etc. ***Problema*** $A_{\mathbf{C}}$
- …determinar se uma dada linguagem, de uma certa classe **C**, é vazia. ***Problema*** $E_{\mathbf{C}}$
- …determinar se duas dadas linguagens, de uma certa classe **C,** são iguais. ***Problema*** $EQ_{\mathbf{C}}$

Cada um desses problemas é, por sua vez, codificado como uma linguagem.

Problemas de Decisão
Exemplo de Codificação

Vejamos como A_{DFA} pode ser codificado.

A_{DFA} é o problema de decidir

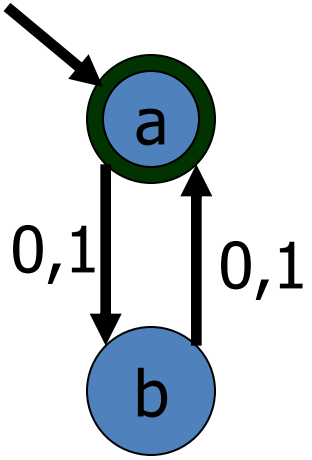
Dado: Um DFA *M* e um string *x*.

Decidir: *M* aceita *x* ?

A linguagem *L* de interesse é a linguagem que consiste de codificações de pares (*M, x*) tais que *M* aceita *x*. A codificação é denotada como "<*M, x* >". Vejamos a seguir como essa codificação pode ser feita.

Problemas de Decisão
Exemplo de Codificação

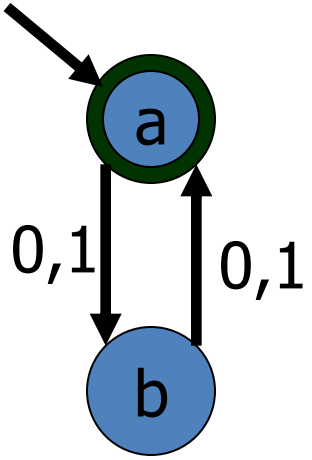
Considere o DFA:



Q: Como ele pode ser representado como um string?

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:

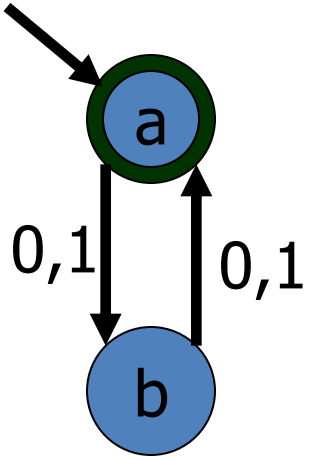


<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



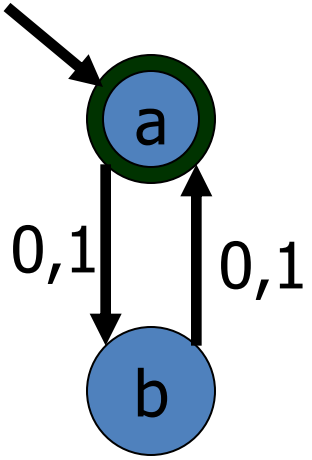
<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

Q

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



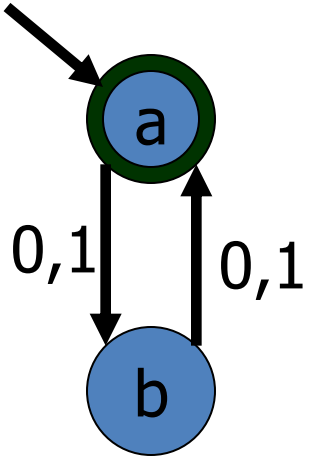
$<M>$ =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

Σ

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



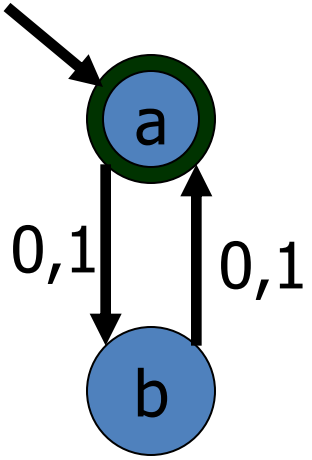
<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

δ

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



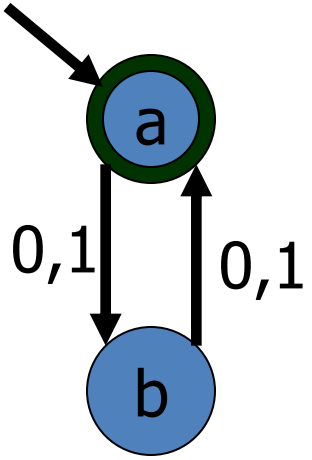
<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

q_0

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



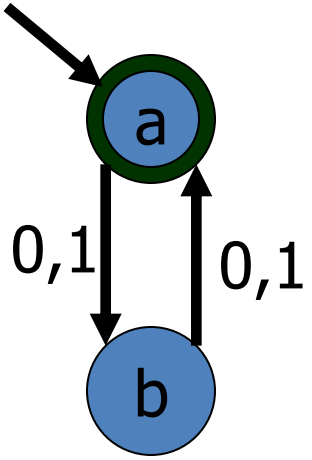
<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

F

Problemas de Decisão
Exemplo de Codificação

R: Esse DFA seria representado como a 5-tupla:



<*M* > =

({a,b},{0,1},{(a,0,b),(a,1,b),(b,0,a),(b,1,a)},a,{a})

A codificação <*M,w* > seria obtida adicionando "," e o string de entrada *w*.

Problemas de Decisão
O que de fato fazemos!

Agora que mostramos a idéia de como é possível a definição de problemas de decisão como linguagens (codificando instâncias do problema como strings), vamos, na prática, ignorar essa questão da codificação, lembrando-nos apenas que ela deve poder ser feita computacioalmente.

Mais simplesmente, vamos trabalhar diretamente sobre a estrutura de dados mais natural para a representação do problema (no caso de FA's, um grafo).

A_{DFA}

Q: Como você resolveria A_{DFA} ?

A_{DFA}

R: Simplesmente siga o caminho rotulado pelo string de entrada, a partir do estado inicial. Aceite se ele termina em um estado em *F*. Mais precisamente:

DFAaccept(DFA *M,* String $x_1\, x_2\, x_3 \ldots\, x_n$)

State $q = q_0$ //q_0 definido pela 5-tuple *M*

for(i = 1 to n)

$q = \delta(q,x_i)$ // δ definida pela 5-tuple *M*

if($q \in F$) // *F* definido pela 5-tuple *M*

return ACEPT

else

return REJECT

A_{NFA}

Q: E o problema A_{NFA}?

A_{NFA}

A: Apenas converta o NFA para um DFA e use a solução dada para o problema A_{DFA}.

NFAaccept(NFA *N,* String $x_1 x_2 x_3 \ldots x_n$)

DFA *M* = convertaDeterminista(*N*)

return DFAaccept(*M*, $x_1 x_2 x_3 \ldots x_n$)

A_{NFA}

NFAaccept2(NFA *N,* String $x_1\, x_2\, x_3 \ldots\, x_n$)

StateSet $S = \{q_0\}$ // S como um bit string

for(i = 1 to n)

$S = \delta(S, x_i)$ // é NFA, δ retorna um conj.

if(S e F não são disjuntos)

return ACCEPT

else

return REJECT

Linguagens Não Decidíveis

Existem questões de natureza computacional que não podem ser resolvidas por meio de um algoritmo? Uma possível abordagem para responder a essa pergunta é codificar problemas como linguagens e perguntar:

- Existem linguagens não decidíveis?
- Existem linguagens que não são nem mesmo reconhecíveis?
- Podemos construir um exemplo específico de linguagem não decidível?
- Existem linguagens não decidíveis de importância prática?

Linguagens Não Decidíveis
Prova de Existência

Vamos começar com uma prova indireta de que existem linguagens não decidíveis. De fato, vamos provar que existem linguagens não ***reconhecíveis***, e nem ***co-reconhecíveis***:

Lembre-se que uma linguagem *L* sobre um alfabeto Σ é reconhecível se existe uma TM que aceita todo string $s \in L$ (podendo possivelmente entrar em loop se $s \notin L$).

DEF: Uma linguagem *L* sobre o alfabeto Σ é ***coreconhecível*** se Σ^*-*L* é reconhecível.

Linguagens Não Decidíveis
Prova de Existência

A idéia é simples: Vamos mostrar que existem mais linguagens do que TM's (ou programas).

Note que essa prova não é construtiva: mostra que deve existir uma linguagem não reconhecível mas não mostra qual ela seria.

Toda TM pode ser codificada como um string binário. Consequentemente, a cardinality[1] do conjunto de TM's não é maior que a de {0,1}*.

THM: O conjunto de strings {0,1}* é ***enumerável***.

Conjunto Enumerável

Relembre a noção de conj. ***enumerável:***

- Um conjunto *S* é ***contável*** se existe uma função injetora $f : S \rightarrow \mathbf{N}$ de *S* para o conjunto dos números naturais. Um conjunto infinito é ***enumerável*** se existe uma bijeção $\mathbf{N} \rightarrow S$. (i.e., S é dito enumerável se é infinito e contável).

Enumerabilidade de {0,1}*

Prova. Intuitivamente podemos obter uma função injetora f : **N** → {0,1}* listando os strings em ordem lexicográfica:

ε 0 1 00 01 10 11 000 001 010 011 100…

1 2 3 4 5 6 7 8 9 10 11 12…

Enumerabilidade de {Linguagens-TM}

Consequentemente, como toda TM é descrita por um bitstring, apenas podem existir tantas TM's quanto bitstrings.

Não Enumerabilidade de $P(\Sigma^*)$

Veremos entretanto, a seguir, que o conjunto potência $P(\{0,1\}^*)$ – o conjunto de todas as linguagens binárias! – não é contável. Como corolário, o número de linguagens binárias é maior do que o de linguagens reconhecíveis ou co-reconhecíveis. Portanto:

THM: Existe uma linguagem binária que não é nem reconhecível nem coreconhecível.

Vamos então provar que $P(\{0,1\}^*)$ não é contável:

Não Enumerabilidade de $P(\Sigma^*)$

Note que qq subconjunto $T \in P(\Sigma^*)$ pode ser visto como uma função $f: \Sigma^* \rightarrow \{0,1\}$: $f(x) = 1$ se $s \in T$; $f(x) = 0$ se $s \notin T$.

EX: Considere a função $f : \Sigma^* \rightarrow \{0,1\}$ defined by the following table:

x	ε	0	1	00	01	10	11	000	001	010	011	100	101	111	0000	...
$f(x)$	1	1	1	1	0	0	1	1	0	1	0	0	1	1	1	...

Q: Qual é a linguagem representada por f ?

Não Enumerabilidade de $P(\Sigma^*)$

x	ε	0	1	00	01	10	11	000	001	010	011	100	101	111	0000	...
$f(x)$	1	1	1	1	0	0	1	1	0	1	0	0	1	1	1	...

R: f representa **pal**.

Suponha que $P(\Sigma^*)$ fosse enumerável. Então seria possível obter uma bijeção de **N** em $P(\Sigma^*)$ e portanto listar todas as linguagens binárias em uma sequência

$$L_1, L_2, L_3, L_4, L_5, L_6, L_7, \ldots$$

que supostamente contém *toda* linguagem de bitstrings. Essa lista determina uma tabela cujas linhas descrevem a função característica associada a cada linguagem:

Não Enumerabilidade de $P(\Sigma^*)$

← strings in {0,1}* →

	ε	0	1	00	01	10	11	…
L_1	1	1	1	1	0	0	1	
L_2	0	0	0	0	0	0	0	
L_3	1	0	0	0	0	0	0	
L_4	1	1	0	1	0	0	0	
L_5	1	1	1	1	1	1	1	
L_6	0	0	1	0	0	0	1	
L_7	1	0	0	1	1	1	1	
…								

Não Enumerabilidade de $P(\Sigma^*)$ Diagonal Diabólica de Cantor

Usando o argumento de *diagonalização* de Cantor podemos criar L_{evil} – uma linguagem que *não* está na lista. I.e., usamos a tabela para criar uma linha que se poderá provar que não está na tabela, provando que a tabela não contém todas as linguagens binárias – o que contradiz a a hipótese de que $P(\Sigma^*)$ é enumerável.

Não Enumerabilidade de $P(\Sigma^*)$ Diagonal Diabólica de Cantor

L_{evil} é criada do seguinte modo: A coluna j de L_{evil} é o oposto da coluna j da linha j *de* L_j. Em outras palavras, L_{evil} é a ***anti-diagonal*** da tabela. Isso garante que L_{evil} difere de todas as linguagens listadas na tabela em pelo menos um string: Se L_i contém o j^{esimo} string, então L_{evil} não o contém, e se L_i não contém o j^{esimo} string, então L_{evil} o contém.

$r_{evil} = \{$

Vejamos como isso é feito:

Diagonalização de Cantor

← strings in {0,1}* →

	ε	0	1	00	01	10	11	…
L_1								
L_2								
L_3								
L_4								
L_5								
L_6								
L_7								
…								
L_{evil}								

Diagonalização de Cantor

← strings in {0,1}* →

	ε	0	1	00	01	10	11	…
L_1	1	1	1	1	0	0	1	
L_2								
L_3								
L_4								
L_5								
L_6								
L_7								
…								
L_{evil}	0							

Diagonalização de Cantor

← strings in {0,1}* →

	ε	0	1	00	01	10	11	…
L_1	1	1	1	1	0	0	1	
L_2	0	0	0	0	0	0	0	
L_3								
L_4								
L_5								
L_6								
L_7								
…								
L_{evil}	0	1						

Diagonalização de Cantor

← strings in {0,1}* →

	ε	0	1	00	01	10	11	…
L_1	1	1	1	1	0	0	1	
L_2	0	0	0	0	0	0	0	
L_3	1	0	0	0	0	0	0	
L_4								
L_5								
L_6								
L_7								
…								
L_{evil}	0	1	1					

Diagonalização de Cantor

← strings in {0,1}* →

	ε	0	1	00	01	10	11	…
L_1	1	1	1	1	0	0	1	
L_2	0	0	0	0	0	0	0	
L_3	1	0	0	0	0	0	0	
L_4	1	1	0	1	0	0	0	
L_5								
L_6								
L_7								
…								
L_{evil}	0	1	1	0				

Diagonalização de Cantor

← strings in {0,1}* →

	ε	0	1	00	01	10	11	…
L_1	1	1	1	1	0	0	1	
L_2	0	0	0	0	0	0	0	
L_3	1	0	0	0	0	0	0	
L_4	1	1	0	1	0	0	0	
L_5	1	1	1	1	1	1	1	
L_6								
L_7								
…								
L_{evil}	0	1	1	0	0			

Diagonalização de Cantor

← strings in {0,1}* →

	ε	0	1	00	01	10	11	…
L_1	1	1	1	1	0	0	1	
L_2	0	0	0	0	0	0	0	
L_3	1	0	0	0	0	0	0	
L_4	1	1	0	1	0	0	0	
L_5	1	1	1	1	1	1	1	
L_6	0	0	1	0	0	0	1	
L_7								
…								
L_{evil}	0	1	1	0	0	1		

Diagonalização de Cantor

← strings in {0,1}* →

	ε	0	1	00	01	10	11	…
L_1	1	1	1	1	0	0	1	
L_2	0	0	0	0	0	0	0	
L_3	1	0	0	0	0	0	0	
L_4	1	1	0	1	0	0	0	
L_5	1	1	1	1	1	1	1	
L_6	0	0	1	0	0	0	1	
L_7	1	0	0	1	1	1	1	
…								
L_{evil}	0	1	1	0	0	1	0	

Uma linguagem não decidível

Agora sabemos que existe uma linguagem não reconhecível. De fato, a *maioria* das linguagens são não reconhecíveis, já que a cardinalidade de $P(\Sigma^*)$ é um infinito maior do que o da cardinalidade de Σ^*! Entretanto, a prova existencial não é construtiva. Gostaríamos de encontrar um exemplo de tal linguagem.

THM: A_{TM} é reconhecível, mas não decidível

A_{TM}: Reconhecível mas Não Decidível

A_{TM} pode ser reconhecida do seguinte modo:

"Sobre a entrada <*M*,*x*>:

1. Simule a execução da TM *M* sobre *x*
2. Se *M* aceita, então ACCEPT"

Esse algoritmo nos diz quando strings são aceitos, mas se o string é rejeitado, pode não haver uma saída, já que *M* pode, nesse caso, entrar em um loop infinito e, portanto, o algoritmo também entraria em loop ao simular *M*.

A_{TM}: Reconhecível mas Não Decidível

A_{TM} não é decível:

Suponha por contradição, que A_{TM} seja decidível. Então existe alguma TM, digamos *D*, que decide A_{TM}. Considere a codificação implícita $M \rightarrow \langle M \rangle$. Vamos usar *D* para construir uma TM diabólica *E* (usando diagonalização de Cantor), de modo a mostrar que A_{TM} não pode ser decidível.

A TM diabólica

E = “Sobre a entrada *w*

1. Se *w* não codifica uma TM, REJECT.
2. Senão, seja *M* a TM codificada por *w* Simule a execução da máquina *D* sobre a entrada <*M*,*w*>.
3. Se *D* aceita, REJECT
4. Se *D* rejeita, ACCEPT”

Como *D* é um decisor, *E* também o é.

A TM diabólica

Permanece a seguinte incômoda questão:

E aceita ou rejeita o próprio código <*E*>?

CASO 1) Aceita. Então *D* rejeita a entrada < *E*, <*E*> >. Portanto, pelo passo 4 do pseudocodigo de *E,* temos que *E* não aceita <*E*>. Isso contradiz a hipótese de aceitação!

CASE 2) Rejeita. Então *D* aceita a entrada < *E*, <*E*> >. Portanto, pelo passo 3 do pseudocodigo de *E*, temos que *E* aceita <*E*>. Isso contradiz a hipótese de rejeição!

Como nenhum caso pode ocorrer, conclui-se que *D* não pode existir! □

Consequências em Java

A não decidibilidade de A_{TM} tem consequências importantes:

THM (assumindo a tese de Church-Turing):
Não existe um algoritmo capaz de decidir se um programa Java arbitrário pára. I.e. a linguagem

$Halt_{Java}$ = { programas Java que
eventualmente páram }

não é decidível.

Não decidibilidade de $\text{Halt}_{\text{Java}}$

Porque? Suponha que $\text{Halt}_{\text{Java}}$ fosse decidível. Note que existem simuladores de TM's escritos em Java. Tal simulador **`TMSimulate.java`** tem como entrada dois argumentos – **`arg1,arg2`** – sendo **`arg1`** a codificação de uma TM e **`arg2`** um string sobre o qual a TM será executada. O simulador **`TMSimulate.java`** contém um método:

```
public static boolean run(String arg1, String arg2)
```

que retorna **`true`** sse *M* accepts **`arg2`**.

Não decidibilidade de $\text{Halt}_{\text{Java}}$

A contradição: O seguinte programa então resolveria A_{TM}, contradizendo o fato de que esse problema não é decidível (assumindo que TM's podem, por sua vez, simular – o que segue da tese de Church-Turing, mas pode também ser demonstrado diretamente).

Não decidibilidade de $\text{Halt}_{\text{Java}}$

```
public class D{
  public static void main(String[] args){
     if(TMSimulate.run(args[0],args[1]))
          System.exit(0);
     else
          while(true);
  }
}
```

Portanto, se existisse um algoritmo para decidir a parada de programas Java, ele poderia decidir A_{TM}, verificando quando o programa acima pára!

Não decidibilidade de E_{TM} e EQ_{TM}

Lembre-se que E_{TM} é o problema definido como:

"Dada uma TM M, $L(M) = \varnothing$?"

E EQ_{TM} é o problema de igualdade:

"Dadas TM's M e M', $L(M) = L(M')$?"

Note que EQ_{TM} é pelo menos tão difícil quanto E_{TM}, pois E_{TM} é um caso especial de EQ_{TM}.

Q: Porque?

Não decidibilidade de E_{TM} e EQ_{TM}

R: Basta que a segunda TM *M'* seja uma TM que rejeita qualquer entrada!

Portanto, para provar que EQ_{TM} não é decidível basta mostrar que E_{TM} não é decidível:

THM: E_{TM} não é decidível.

Prova. Suponha que E_{TM} seja decidível. Seja *F* um decisor para E_{TM}. Vamos usar *F* para resolver A_{TM}, aplicando *F* a uma classe de TM's cujo propósito é codificar instâncias do problema de aceitação como instâncias do problema de linguagem vazia.

Não decidibilidade de E_{TM} e EQ_{TM}

Dada uma TM *M* e um string *w* construímos a TM $K_{M,w}$ cuja linguagem é não vazia sse *w* é aceito por *M*. A descrição de $K_{M,w}$ é a seguinte:

"Sobre a entrada *x*

1. Apague *x* e substitua por *w*.
2. Simule a execução de *M* sobre a entrada *w*."

Então, se *M* aceita *w,* $K_{M,w}$ aceita *qualquer x* ! Por outro lado, se *M* não aceita *w,* então $K_{M,w}$ rejeita todo *x* ou entra em loop infinito para todo *x*. Resumindo:

$$L(K_{M,w}) = \begin{cases} \Sigma^*, & \text{if } \ w \in L(M) \\ \{\}, & \text{if } \ w \notin L(M) \end{cases}$$

Não decidibilidade de E_{TM} e EQ_{TM}

Se E_{TM} fosse decidível poderíamos decidir quando $L(K_{M,w})$ é vazia e, portanto, decidir quando M aceita w , já que:

$$L(K_{M,w}) = \begin{cases} \Sigma^*, \text{ if } \ w \in L(M) \\ \{\}, \text{ if } \ w \notin L(M) \end{cases}$$

Isso completa a prova de que E_{TM} e, por consequência, EQ_{TM} não são decidíveis □

A idéia é um caso particular de uma técnica geral para mostrar que um novo problema não é decidível: usar *redução* de um problema não decidível p/ esse problema.

Mais sobre E_{TM}

R: E_{TM} é co-reconhecível, mas não reconhecível: Podemos dizer quando a linguagem de uma TM *não é* vazia, simulando a TM para todas as entradas e então produzindo a saída NÃO-VAZIA quando a TM aceita algum string. E_{TM} não é reconhecível, porque, caso o fosse, isso significaria que E_{TM} seria decidível, o que já provamos que não é verdade.