Decidibilidade

Preâmbulo

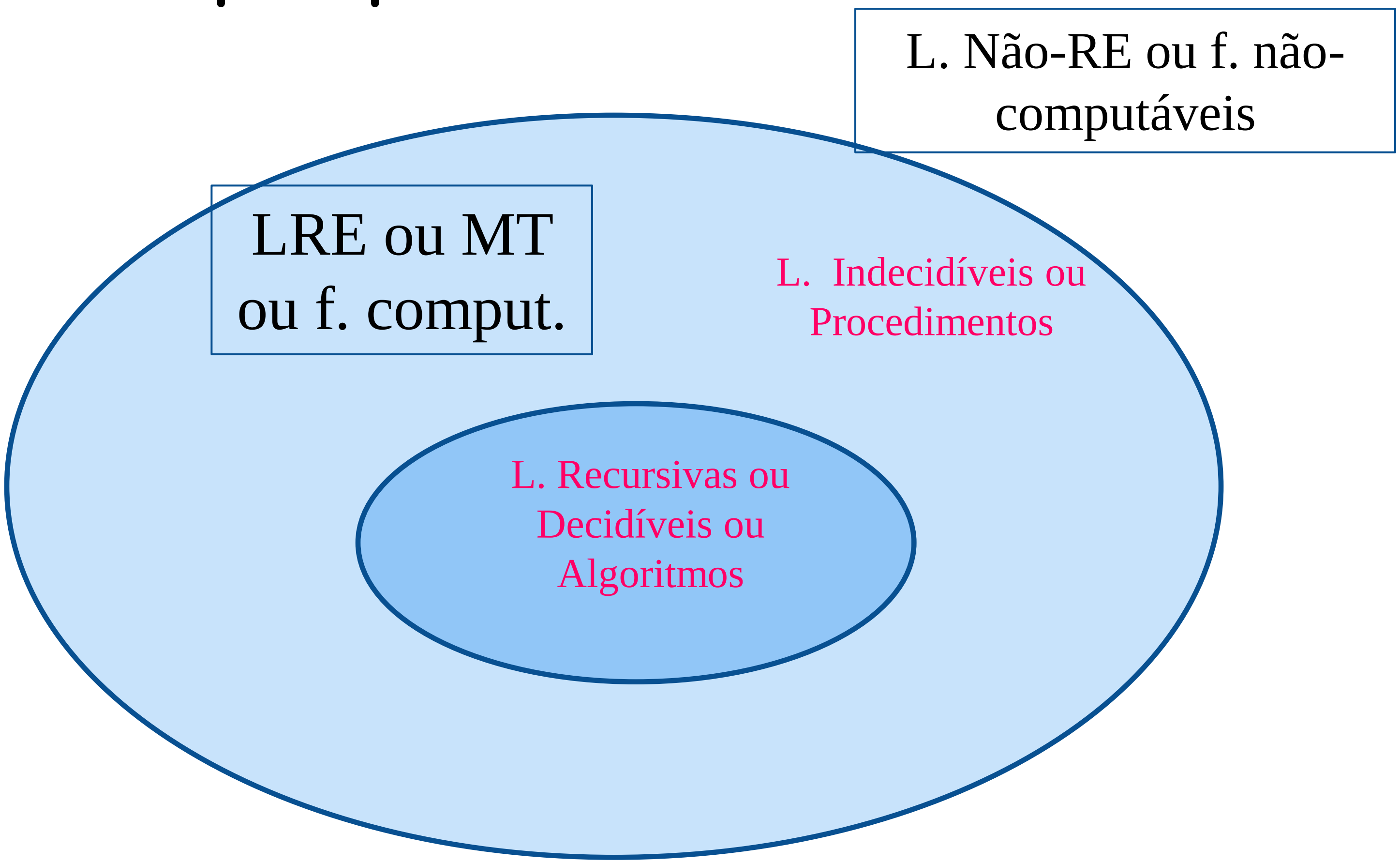
Problemas Computáveis

- Máquinas de Turing ou Funções Computáveis ou Linguagens Recursivamente Enumeráveis ***LRE*** podem ser divididas em 2 classes:

  (1) as MT que, para qualquer cadeia de entrada, sempre terminam, ou seja, sempre respondem se a cadeia faz parte ou não da linguagem. Em outras palavras, **decidem** a linguagem. Essas linguagens são chamadas Linguagens *Recursivas ou Decidíveis*, e essas MT correspondem aos **Algoritmos.**

  (2) as MT que, para qualquer cadeia de entrada, terminam aceitando a cadeia, se ela fizer parte da linguagem, ou podem funcionar indefinidamente sobre entradas que elas não aceitam. Em outras palavras, **aceitam** a linguagem. Tais linguagens são chamadas *Linguagens Indecidíveis,* e essas MT correspondem aos **Procedimentos.**

- Problemas ou Linguagens *Indecidíveis* são aqueles para as quais não existe nenhum *algoritmo*, ou seja, uma MT que sempre para.



Pergunta

- O que caracteriza as funções indecidíveis (para as quais há procedimento, mas não algoritmo)?

Ou

- Que tipo de propriedade (característica da linguagem) pode ser decidida ou não?

Exemplos clássicos de funções indecidíveis

Problema 1.: Existe um procedimento – na verdade, um algoritmo - (p.ex., em Pascal) que toma como entrada um outro procedimento qualquer, *p,* e retorna *true* se *p* é um algoritmo, ou *false*, caso contrário?

Resposta: Não!

Prova: Por contradição

Suponha que tal procedimento exista. Vamos chamá-lo de ALG. Então a declaração de ALG é da forma:

function ALG (procedure p) : boolean;

<corpo da função>

Podemos, então, usar a função ALG para definir novos procedimentos:

```
procedure Problema (x: integer);
begin
      while ALG(Problema) do nil
end;
```

Pergunta: o procedimento "***Problema*** " **é algoritmo?**

→ Suponha que **sim**. Então *ALG(Problema)* é *true* e o comando *while* nunca termina, e portanto, *Problema* nunca termina, e **não é algoritmo**. Contradição!

→ Suponha que **não**. Então *ALG(Problema)* é *false* e o comando *while* termina, e portanto, *Problema* termina, e **é um algoritmo**. Contradição!

→Portanto, *Problema termina se Problema não termina*

Logo, *ALG* não pode existir.

Problema 2. (da Parada): Existe um procedimento – na verdade, um algoritmo - HALT, que toma como entrada um procedimento *p* e um inteiro x, e
retorna *true* se *p* termina com entrada x e *false*, se *p* não termina com entrada x?

Resposta: Não!

Prova: Por contradição.

Suponha que HALT exista. Então podemos escrever um procedimento Pascal D:

```
procedure D (x: integer);
begin
        while HALT(D, x) do nil
end;
```

Pergunta: D termina com entrada x?

→ Suponha que **sim**. Então *HALT(D, x)* é *true* e D **não termina com entrada x**. Contradição!

→ Suponha que **não**. Então *HALT(D, x)* é *false* e D **termina com entrada x**. Contradição!

Portanto, *D termina com entrada x se D não termina com entrada x.*

Logo, HALT não pode existir!

Propriedades Indecidíveis

- Logo, as propriedades de procedimentos:
  - P1: É algoritmo ou não (Problema 1) e
  - P2: Termina para uma entrada x (Problema 2)

são indecidíveis – ou seja, não há algoritmos que as decidam.

Propriedades Indecidíveis

Corolário: Se uma propriedade **P é indecidível**, então a negação desta propriedade, **¬P, também é indecidível.**

Se queremos verificar ¬P num procedimento *A*, temos que decidir P executando *A*, e quando *A* retorna *true*, a saída é *false*, e quando *A* retorna *false*, a saída é *true.*

Daí, as propriedades ¬ P1 e ¬ P2:
"não termina para alguma entrada" e "não termina para entrada x" são ambas **indecidíveis**.

Propriedades Semi-Decidíveis

Um atributo menos rigoroso de propriedades de procedimentos é introduzido por:

DEF.: Uma propriedade de procedimento P é dita **semi-decidível** se existir um procedimento que, quando dado um procedimento *p*, resulta *true*, se *p* tem a propriedade P. (nada se espera se ele não tiver a propriedade P)

Propriedades Semi-Decidíveis

Obs.1:

- A noção de semi-decidibilidade é mais fraca que a de decidibilidade. Se uma propriedade P é **decidível**, então sempre se pode dizer se um procedimento **tem ou não tem** a propriedade P. Já se P é <u>semi-decidível</u>, pode-se dizer apenas se um procedimento <u>tem</u> a propriedade P.

- **Corolário:** Se P é decidível, certamente ela é semi-decidível.

Propriedades Semi-Decidíveis

Teorema: A propriedade de procedimento "*termina para entrada x*" é semi-decidível.

Prova: O procedimento pode ser expresso em Pascal como:

```
function TERM (procedure f): boolean;
begin
    f(x);
    TERM:= true
end;
```

Repare que, se f(x) terminar, TERM também termina; se f(x) não terminar, TERM não termina.

Entretanto, existem muitas propriedades que não são sequer semi-decidíveis.

Resultado: Se P é semi-decidível e $\neg$ P é semi-decidível, então P é decidível.

Prova: Assuma que ambos P e $\neg$ P são semi-decidíveis.

Sejam $\mathbf{p_1}$: o procedimento que resulta *true*, se seu argumento tem a propriedade P;

e $\mathbf{p_2}$: o procedimento que resulta *true*, se seu argumento tem a propriedade $\neg$ P.

Podemos, então, construir um procedimento (algoritmo) *p* que executa ou simula p_1 e p_2 em paralelo e espera que um dos 2 retorne *true.* Desde que P é ou *true* ou *false*, exatamente um dos 2 procedimentos **deve** retornar com o valor *true*.

Se p1 retornar *true* → *p* retorna *true;*

Se p2 retornar *true* → *p* retorna *false*.

<u>Portanto</u>, ***p* sempre termina e decide P**.

Este resultado é útil quando queremos mostrar que uma propriedade não é semi-decidível. Por exemplo:

- Sabemos que a propriedade P, *"procedimento p termina para entrada x"*, é semi-decidível. Se $\neg$P, *"procedimento p não termina para entrada x"*, for semi-decidível, então, pelo Resultado anterior, teríamos que a propriedade P, "*procedimento p termina para entrada x*", é decidível – o que sabemos ser falso.
- Concluímos, então, que $\neg$P, *"procedimento p não termina para entrada x"*, não é semi-decidível.

Outras propriedades indecidíveis

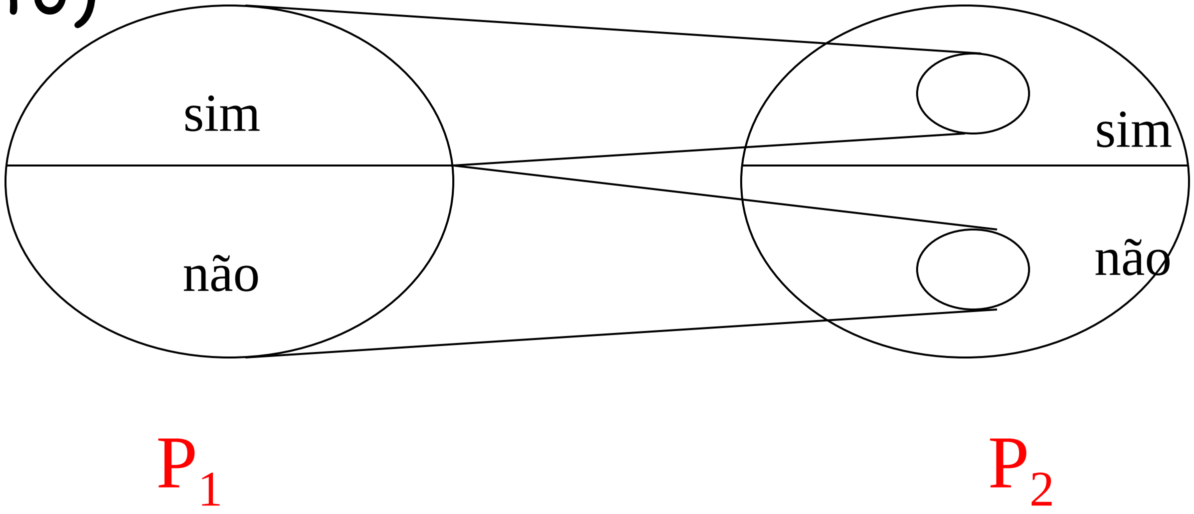
- **Problema da equivalência de programas:** Não existe um algoritmo que decide se dois procedimentos dados P e Q são equivalentes; mais precisamente, não existe um programa Eq(P, Q) tal que Eq para com quaisquer dados de entrada, e Eq(P, Q) = True se os procedimentos P e Q calculam a mesma função e Eq(P, Q) = False em caso contrário. Note que P e Q calculam a mesma função se para qualquer entrada ou ambos não param, ou ambos param com a mesma resposta.
- **Problema da Satisfatibilidade:** É indecidível se uma expressão lógica, formada com os conectivos e quantificadores lógicos $\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\forall$, $\exists$, é satisfatível, ou seja, tem valor lógico verdadeiro para quaisquer valores de seus símbolos.
- É indecidível se uma expressão formada com os símbolos 0, 1, +, *, =, conectivos lógicos $\neg$, $\wedge$, $\vee$, $\Rightarrow$, variáveis e quantificadores lógicos $\forall$ e $\exists$, é um Teorema da Aritmética.

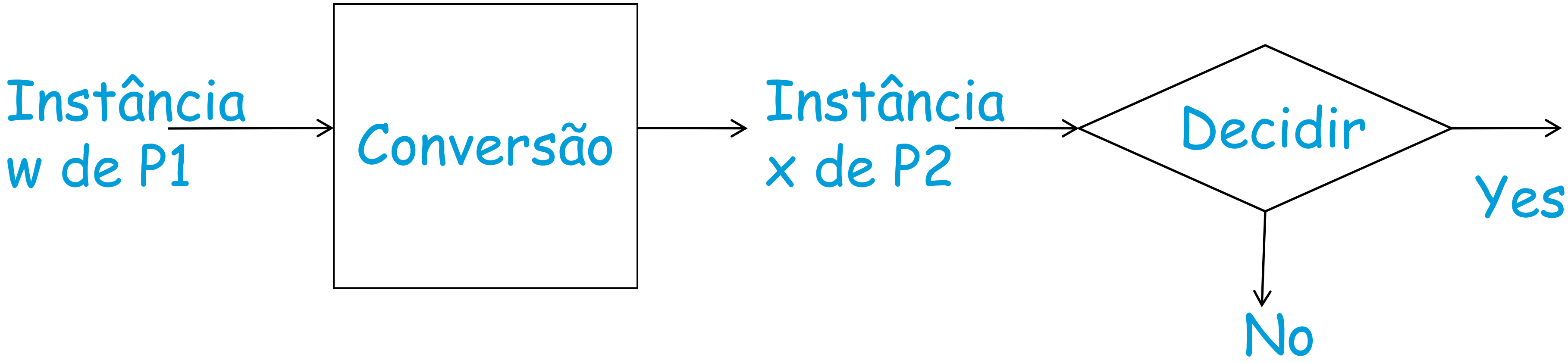
Redução de um problema a outro para mostrar indecidibilidade

- Se sabemos que P1 é indecidível, e queremos mostrar que P2 é indecidível, podemos tentar:
- reduzir P1 a P2 e,
- se pudéssemos resolver P2 (ou seja, se P2 fosse decidível), então poderíamos usar essa solução para resolver P1.
- Mas como P1 é indecidível, então P2 não pode ser decidível.

Redução de Problemas

- *P_1 se reduz a P_2* quando existe um algoritmo que converte instâncias de P_1 em instâncias de P_2 que têm a mesma resposta.(P_1 é o que se conhece; P_2 é a incógnita – nunca o oposto)





O bloco "Conversão" deve converter instâncias de P1 em instâncias de P2 que têm a mesma resposta. E:

1. Dada uma instância de P1, ou seja, uma dada cadeia w que pode ou não estar na linguagem P1, aplique o algoritmo de conversão para produzir uma cadeia x.

2. A resposta sobre w e P1 (se w pertence a P1) será a mesma de x e P2.

Assim, se fosse possível decidir P2, então seria possível decidir P1 também. Mas, como P1 é sabidamente indecidível, então temos uma prova por contradição de que o algoritmo de decisão para P2 não pode existir; isto é, P2 é indecidível.

Teorema: **Se existe uma redução de P_1 a P_2, então:**

1. Se P_1 é indecidível, então P_2 também o é.

2. Se P_1 é não-RE, então P_2 também o é.

(se houvesse uma MT para reconhecer P_2, ela tb seria capaz de reconhecer P_1, já que a redução implica em respostas iguais para candidatos “reduzidos”. Assim, P_1 seria RE, contrariando a hipótese.)

Propriedades indecidíveis sobre MT

- Dada uma MT, ela aceita a linguagem vazia?

Sejam:

$L_e = \{M | L(M) = \emptyset\}$ – *conj. das MT cuja linguagem é vazia*

Refrazeando a pergunta: Dada uma MT, ela pertence a L_e?

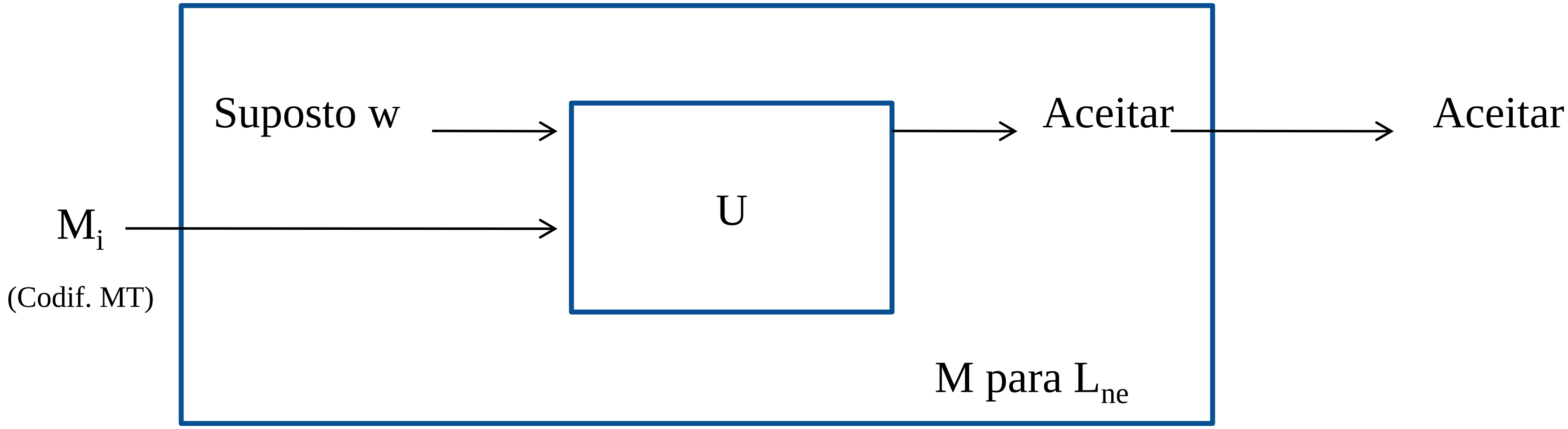
Seja $L_{ne} = \{M | L(M) \neq \emptyset\}$ – *conj. das MT cuja linguagem contém ao menos uma cadeia*

Logo, L_e e L_{ne} são **complementos** uma da outra.

L_{ne} é a "mais fácil" das duas e é RE, mas não recursiva, ou seja, é indecidível. Por outro lado, L_e é não-RE, ou seja, não computável. Por que?

Teorema: L_{ne} é recursivamente enumerável.

Prova: Temos que exibir uma MT para ela.



A operação de M segue:

-M toma como entrada o código de uma MT M_i

-Usando sua capacidade não-determinística, M supõe uma entrada w que M_i poderia aceitar.

-M testa se M_i aceita w. Para essa parte, M pode simular a MT universal U.

-Se M_i aceita w, então M aceita sua própria entrada, M_i.

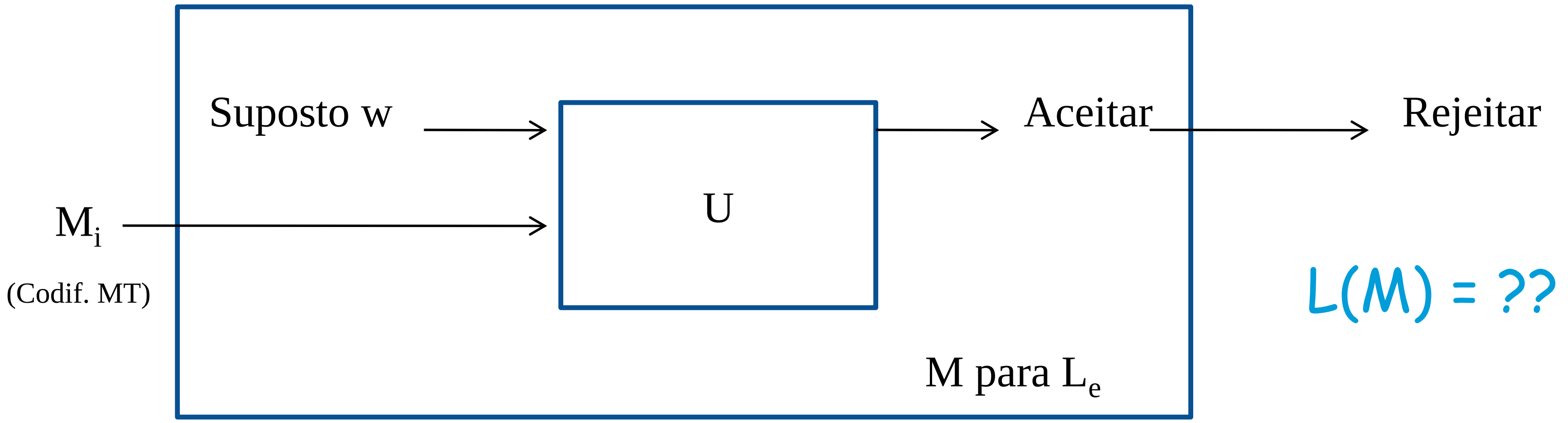
Dessa maneira, se M_i aceita até mesmo uma única cadeia, M irá supor essa cadeia (entre todas as outras) e aceitará M_i. Porém, se $L(M_i) = \varnothing$, então nenhuma suposição de w levará à aceitação por M_i e assim M não aceitará M_i. Desse modo, $L(M) = L_{ne}$.

Teorema: **L_{ne} é não-recursiva (indecidível).**

(para MT cuja linguagem é vazia, M não para)

Teorema: **L_e não é RE (é não computável).**

(não existe uma MT que reconhece L_e)



Teorema de Rice: Todas as propriedades não-triviais das linguagens RE são indecidíveis.

- Se a linguagem aceita por uma MT é finita
- Se a linguagem aceita por uma MT é uma LR
- Se a linguagem aceita por um MT é uma LLC
- Se a linguagem aceita por uma MT é não vazia

- Atenção: características sobre MT, e não sobre as linguagens aceitas, podem ser decidíveis. Ex. é decidível se uma MT tem cinco estados. Basta examinar o código da MT e contar o número de estados que aparecem em qualquer de suas transições.

Há uma analogia do Teorema de Rice para programas: qualquer propriedade não-trivial que envolva aquilo que o programa faz é indecidível:

- Se termina para uma entrada x;
- Se termina para todas as entradas;
- Se é equivalente a outro programa;
- etc.

Decidibilidade e Intratabilidade

- Distinguir problemas indecidíveis é importante também para orientar programadores sobre o que podem fazer via programação.
- No entanto, alguns problemas, embora decidíveis, exigem tempo demais para sua resolução. São chamados "intratáveis", e mais do que os indecidíveis, são enfrentados diariamente e apresentam muitos desafios.
- Precisamos, assim, de ferramentas que nos ajudem a decidir se um problema é indecidível ou intratável e o que fazer nesse último caso.