

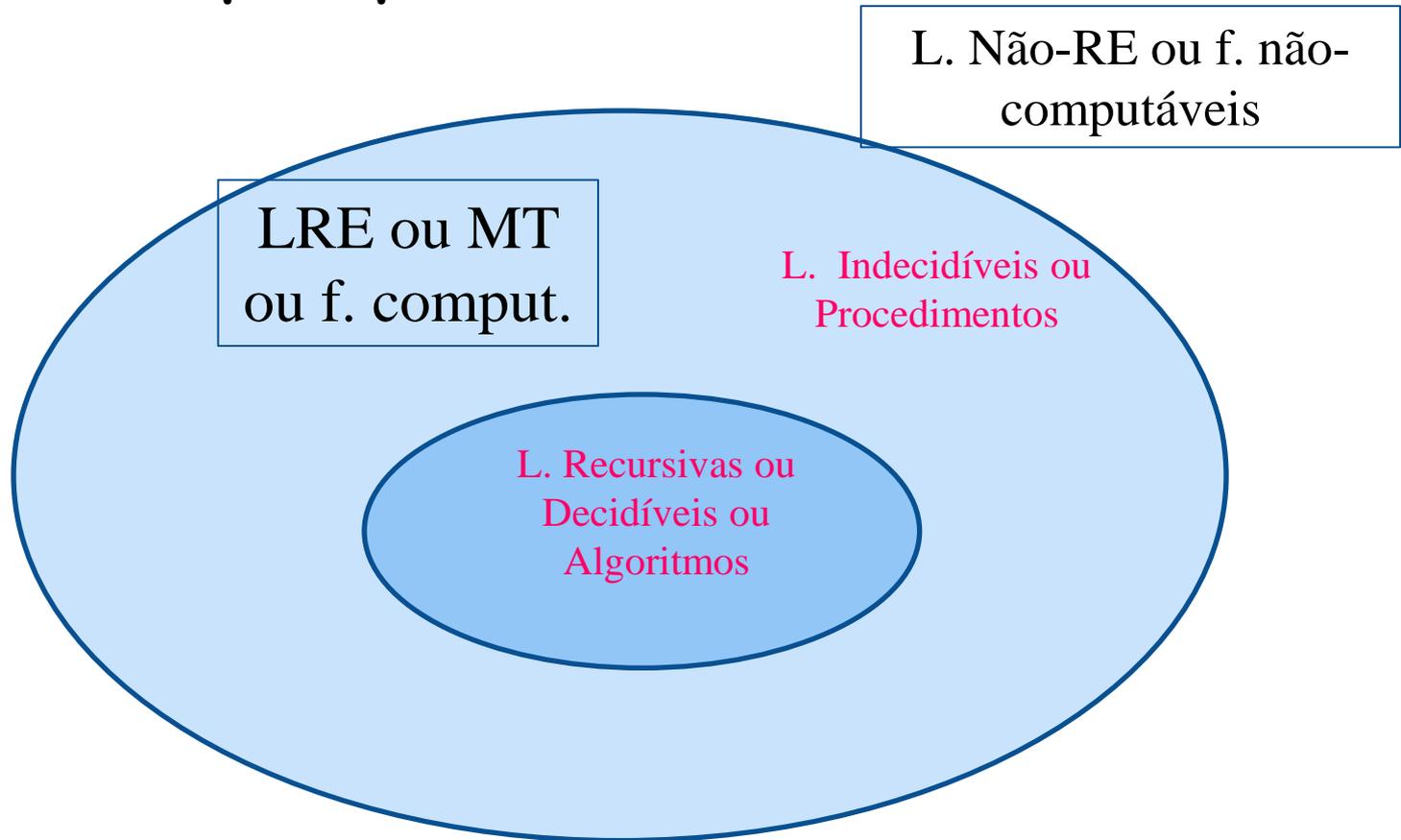
# Decidibilidade

## 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 os 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 e somente se  $D$  não termina com entrada  $x$ .

Logo,  $HALT$  não pode existir!

# Propriedades Indecidíveis

- Logo, as propriedades de procedimentos:

- É algoritmo ou não (Problema 1) e

- 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,  $\neg P$ , também é indecidível.

Se queremos verificar  $\neg P$  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:

*"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 existe 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 semi-decidível, pode-se dizer apenas se um procedimento tem 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  $p_1$ : o procedimento que resulta *true*, se  $P$  é decidível;

e  $p_2$ : o procedimento que resulta *true*, se  $\neg P$  é decidível.

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  $p_1$  retornar *true*  $\rightarrow$   $p$  retorna *true*;

Se  $p_2$  retornar *true*  $\rightarrow$   $p$  retorna *false*.

Portanto,  $p$  sempre termina e decide  $P$ .

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

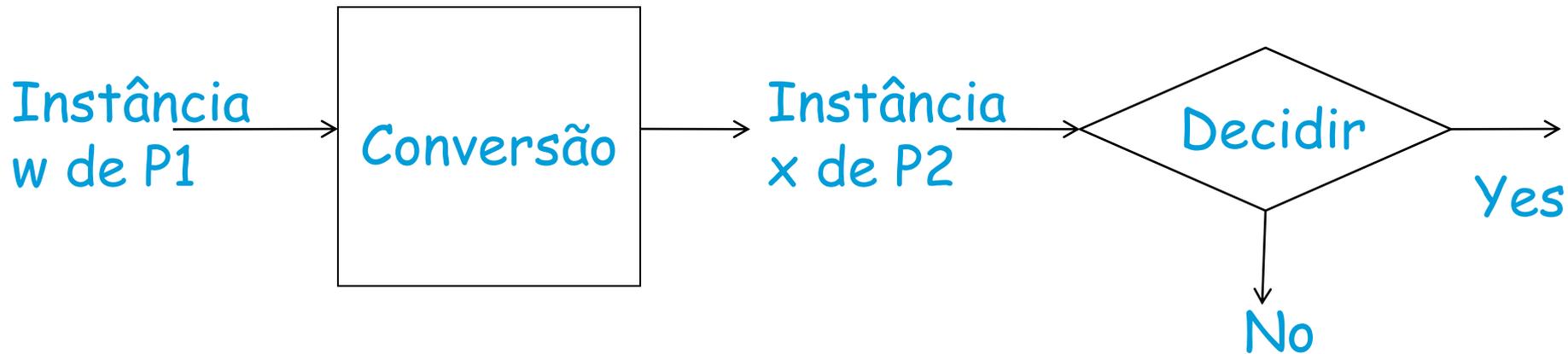
- Sabemos que a propriedade  $P$ , "*procedimento  $p$  termina para entrada  $x$* ", é semi-decidível. Se  $\neg P$ , "*procedimento  $p$  não termina para entrada  $x$* ", for semi-decidível, então, pelo Resultado anterior, teríamos que a propriedade  $P$ , "*procedimento  $p$  termina para entrada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

# Outros problemas 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  $Eq$  pára com quaisquer dados de entrada, e  $Eq(P, Q) = T$  se os procedimentos  $P$  e  $Q$  calculam a mesma função e  $Eq(P, Q) = F$  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.



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. Teste se  $x$  está em P2 e dê a mesma resposta sobre  $w$  e P1.

Se  $w$  está em P1, então  $x$  está em P2, e assim esse algoritmo imprime **Yes**. Se  $w$  não está em P1, então  $x$  não está em P2, e o algoritmo imprime **No**. Ou seja, ele decide P2 e P1. 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.

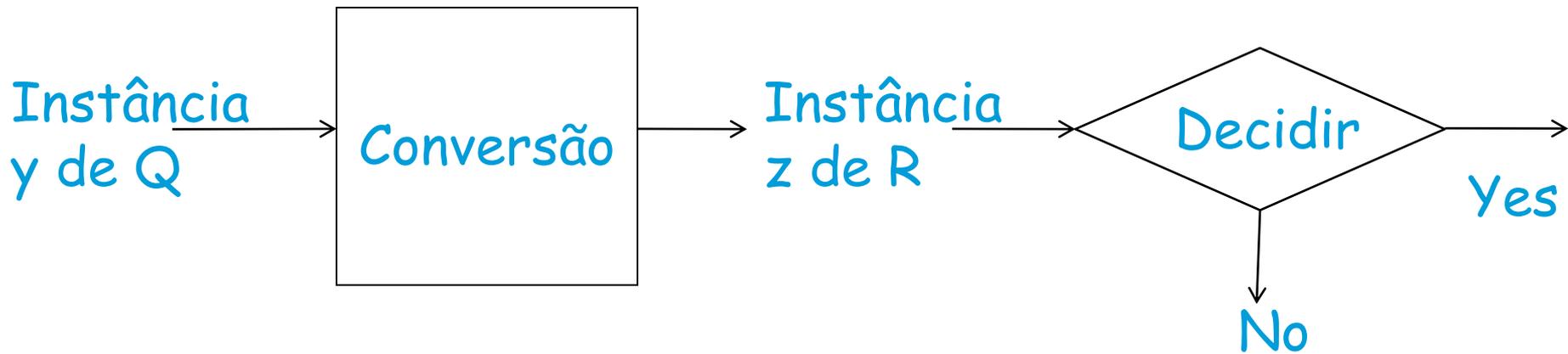
# Exemplo

- Vamos mostrar que a pergunta "o programa  $Q$ , dada a entrada  $y$ , chamará a função  $foo$ ?" é indecidível.
- Se  $Q$  não tem uma função  $foo$ , a resposta é direta.
- Se  $Q$  tem a função, ele pode ou não chamá-la para a entrada  $y$ .
- Seja  $P1$  o problema da parada (indecidível), e  $P2$  o problema de chamar  $foo$ .
- Supomos que existe um programa que decide  $P2$ .

# Exemplo

- Nosso trabalho é projetar um algoritmo que reduza  $P1$  a  $P2$ , ou seja, que converta o problema da parada ao problema de chamar foo.
- Ou seja, dado o programa  $Q$  e sua entrada  $y$  ( $P1$ ), devemos construir um programa  $R$  e uma entrada  $z$  ( $P2$ ) tais que  **$R$ , com entrada  $z$ , chame foo se e somente se  $Q$  com a entrada  $y$  parar:**

1. Se  $Q$  tem uma função chamada `foo`, renomeie essa função e todas as chamadas a ela. É claro que o novo programa  $Q1$  faz exatamente o que  $Q$  faz.
2. Adicione a  $Q1$  uma função `foo`. Essa função não faz nada e não é chamada. O programa resultante é  $Q2$ .
3. Modifique  $Q2$  de tal forma que quando for terminar (antes de parar), chame a função `foo` adicionada em (2). O programa resultante é  $R$  e a entrada  $z$  é igual a  $y$ .



- Agora assumamos que  $Q$  para com entrada  $y$ . Então  $R$  chamará  $foo$ . Porém, se  $Q$  com entrada  $y$  não parar,  $R$  nunca chamará  $foo$ . Se pudermos descobrir se  $R$  com a entrada  $z$  chama  $foo$ , então também saberemos se  $Q$  com a entrada  $y$  ( $y=z$ ) para.
- Mas sabemos que  $P1$  é indecidível, portanto  $R$  não pode existir e  $P2$  é indecidível.

# 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.<sub>23</sub>