Teoria da Computação

Computabilidade e complexidade computacional

Computabilidade e Complexidade

- **Computabilidade**: verifica a existência de algoritmos que resolva uma classe de linguagens – trata a possibilidade da sua construção

- **Complexidade**: trata da eficiência da computação (dos algoritmos) em computadores existentes

  - Complexidade temporal: tempo de processamento exigido

  - Complexidade espacial: espaço de armazenamento exigido

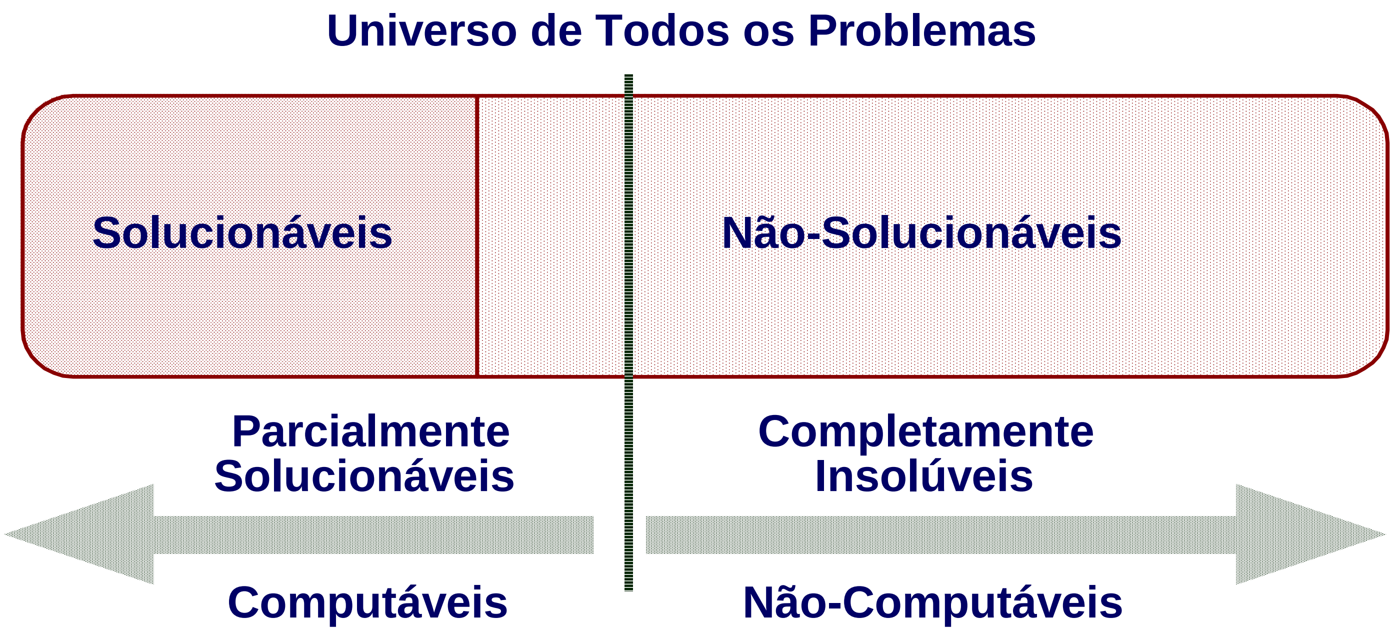
Custo computacional

Computabilidade computacional

Computabilidade - Visão geral

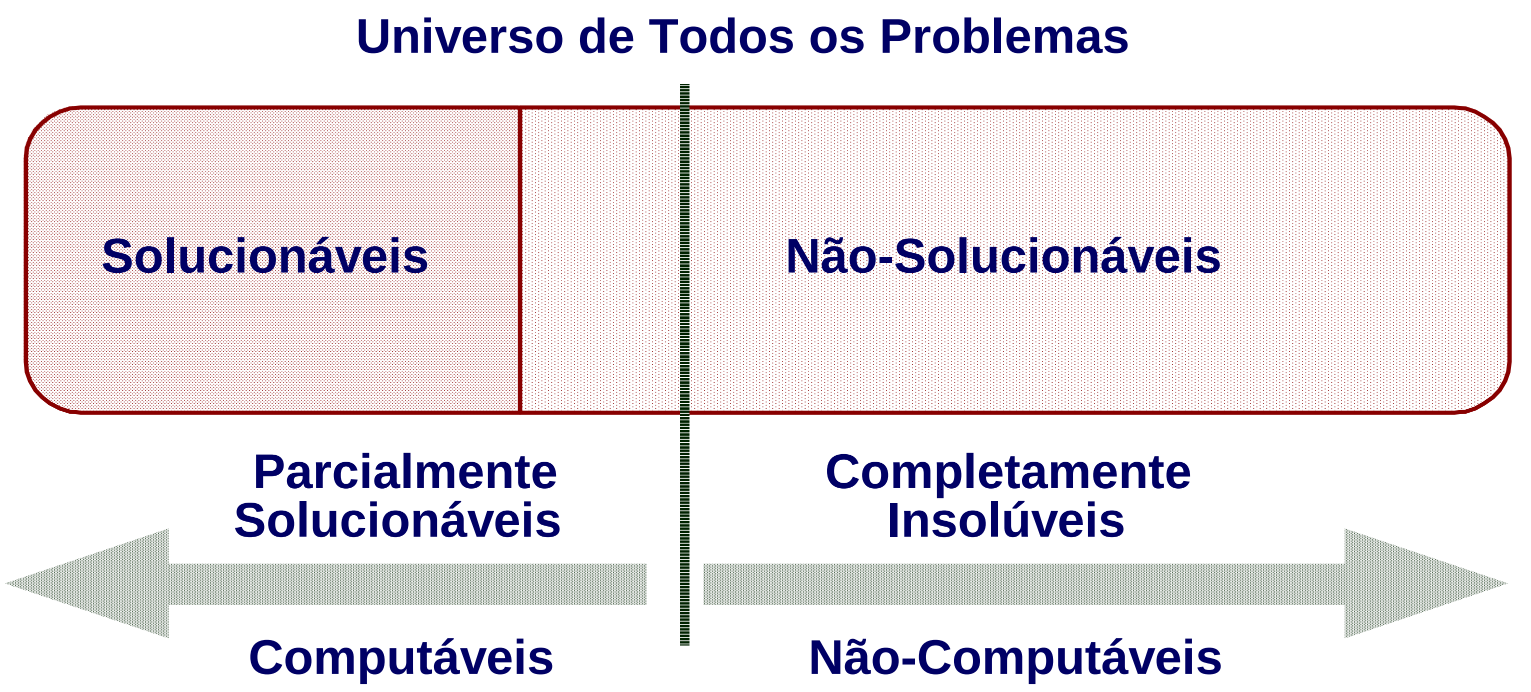
Quais problemas os computadores conseguem efetivamente resolver? Como isso pode ser verificado?

Concentra-se nos problemas com respostas binárias (problemas sim/não ou problemas de decisão).



Computabilidade – problemas computáveis

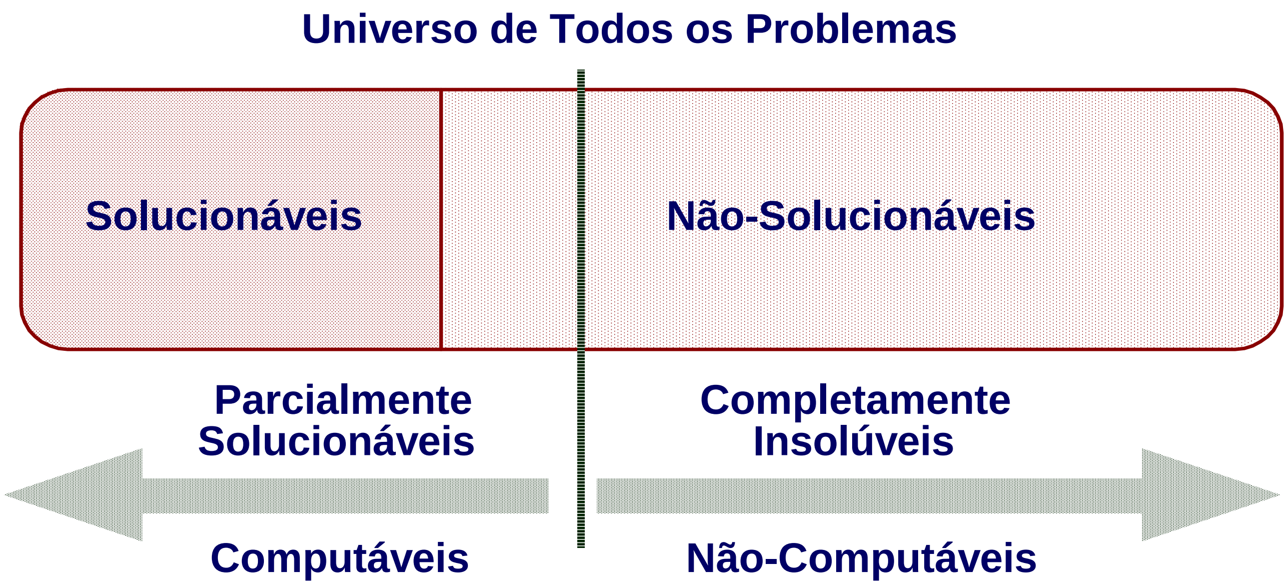
- A classe de problemas computáveis (solucionáveis) é equivalente à *Classe das Linguagens Recursivas*



Computabilidade – problemas não-computáveis

- **Exemplo de problemas não-computáveis na computação**

  - Determinar se dois programas são equivalentes.
  - Determinar se uma gramática livre de contexto é ambígua.
  - Determinar se duas gramáticas livres de contexto são equivalentes.



Computabilidade

Questões que **NÃO** devem ser consideradas na investigação da computabilidade:

- limitações reais de uma implementação, como pode por exemplo, tamanho de espaço de endereços, tamanho da memória principal, questões de tempo e outros.

Computabilidade

- Em estudos iniciados por David Hilbert (início do século XX) foram propostos formalismos para a verificação da computabilidade, sendo os mais importantes:
  - Máquina de Post
  - Máquina com Pilhas
  - Máquina de Turing
- **Estas máquinas** (podem simular computadores reais) não apresentam as limitações citadas anteriormente, logo podem ser **utilizadas** para **verificar** o que um dispositivo de computação pode calcular em um determinado tempo (execução em tempo finito) ➔ ***Decidibilidade***

Computabilidade – Tese de Churc-Turing

- **Tese de Church**: *"Qualquer computação que pode ser executada por meios mecânicos pode ser executada por uma Máquina de Turing".*

- Principais razões pelas quais a maioria dos investigadores demonstram que esta tese é verdadeira:
  - As MT's são tão gerais que podem simular qualquer computação;
  - Nunca ninguém descobriu um modelo algorítmico mais geral que as MT's.

Porque estudar problemas não-solucionáveis

- Principais motivos para o estudo dos problemas não-solucionáveis:

1. Verificar as capacidades e limitações dos computadores;
2. Evitar a pesquisa de soluções inexistentes – problemas que não podem ser resolvidos computacionalmente (Indecidíveis)
3. Para verificar que outros problemas também são insolúveis (princípio da redução).

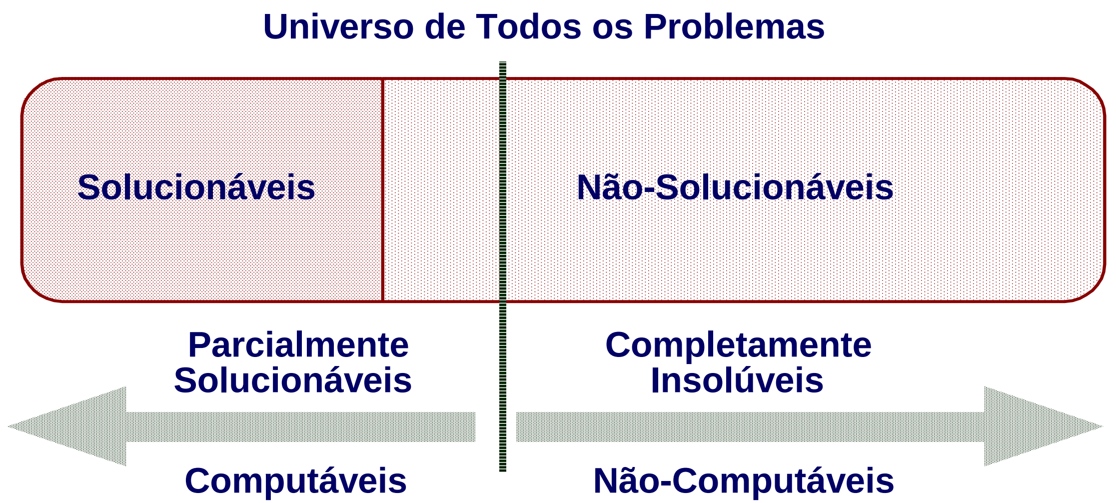
VERIFICAR O QUE É QUE UM COMPUTADOR PODE FAZER!

Computabilidade

- **MT** podem ser divididas em duas classes:

1) MT que, para qualquer cadeia de entrada, sempre terminam, ou seja, sempre respondem se a cadeia $\in$ ou $\notin$ a linguagem. Essas linguagens são chamadas *Linguagens Recursivas ou Decidíveis* ***(computáveis)***

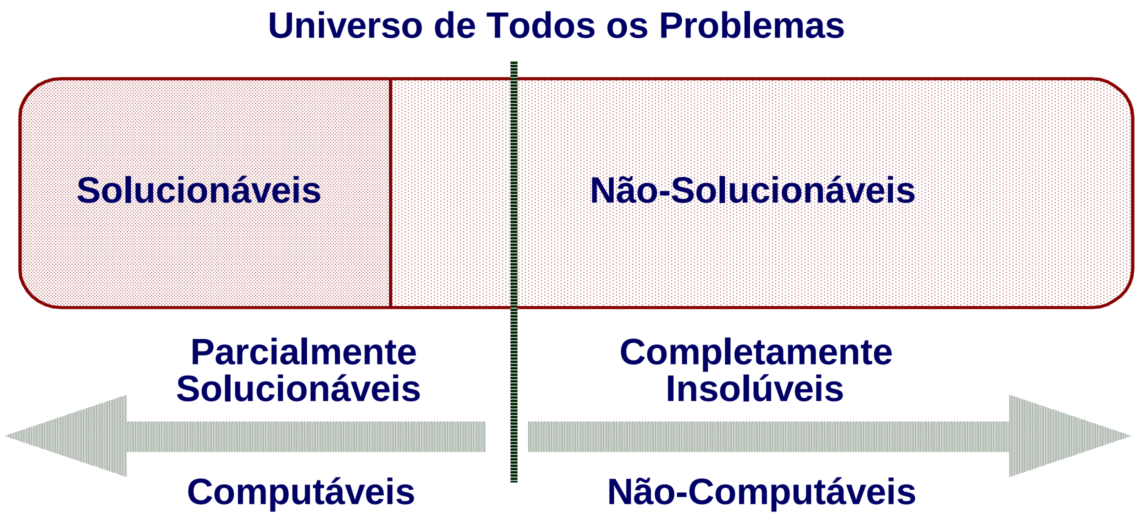
(M,w)= aceita w (alcança um estado final) ou rejeita w (para em um estado que não é final)



Computabilidade

- **MT** podem ser divididas em duas classes:

2) 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 *(loop)*. Em outras palavras, não é possível determinar se aceita ou não a entrada. Tais linguagens são chamadas Linguagens *Indecidíveis (não-computáveis)*



Problemas não-solucionáveis

- Outros exemplos de problemas (não-solucionáveis) clássicos da computação:

  - Detector universal de loops (problema da parada): Dado um programa e uma entrada qualquer, não existe um algoritmo genérico capaz de verificar se o programa vai parar ou não para a entrada.

  - Equivalência de compiladores: Não existe algoritmo genérico que sempre para e que seja capaz de comparar quaisquer dois compiladores de LLC, e verificar se são equivalentes.

Computabilidade – Exemplo

- Classifique o problema de acordo com a entrada (x)

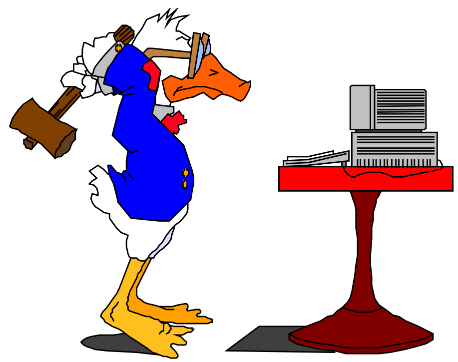
Programa constante

```
Read x;
While x ≠ 10 do
    x := x + 1;
Print x;
End;
```

****

Para x>10 o programa não para!

Vai ficar calculando a vida toda?
Não para?



Computabilidade – Exemplo

- Classifique o problema de acordo com a entrada (x)

Programa constante

Read x;

While x ≠ 10 do

    x := x + 1;

Print x;

End;

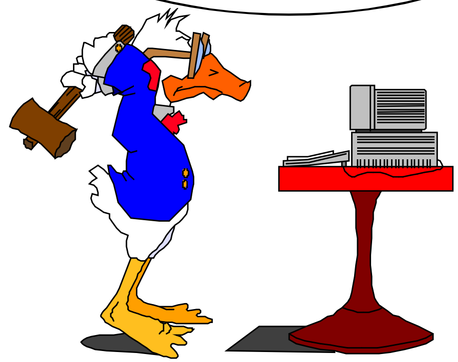
****

Para x>10 o programa não para!

Vai ficar calculando a vida toda?

Não para?

parcialmente solucionável



Decidibilidade - abordagem

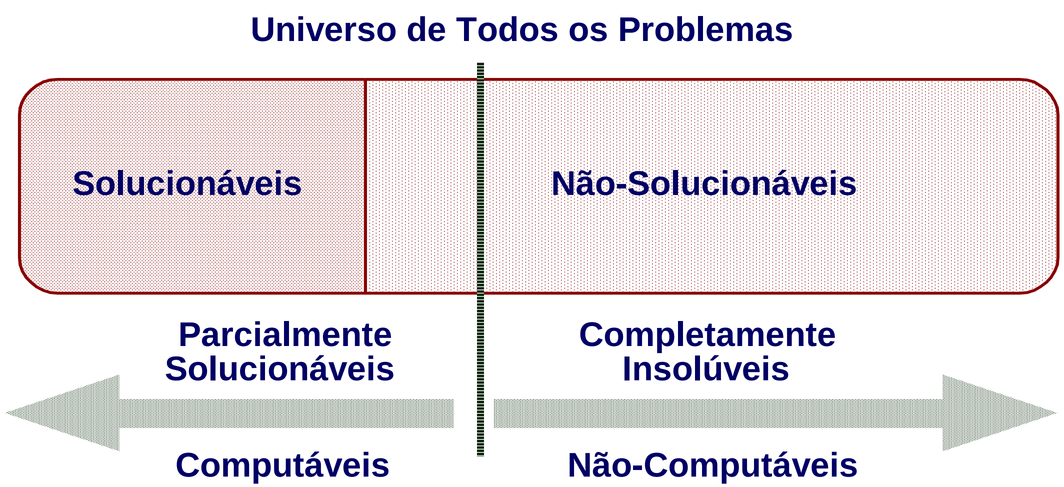
- **A classe das LR está ligada ao conceito de decidibilidade.**
  - Um problema de decisão PD (Sim/Não ou Aceita/Rejeita) P é decidível se, e somente se, certa linguagem associada a P for recursiva.
  - Logo, determinar se certo problema é decidível é basicamente estabelecer se determinada linguagem é recursiva (MT que a reconhece).
    - **Não considera a quantidade de tempo gasto e sim se ele é finito (o conceito que trabalha com a quantidade de tempo gasto é o da Complexidade)**

Problemas não-solucionáveis são parcialmente solucionáveis

- Alguns problemas não-solucionáveis são *parcialmente solucionáveis*, ou seja, existe um algoritmo capaz de responder SIM, embora, eventualmente, possa ficar em um loop infinito para uma resposta que deveria ser NÃO.

- *A Classe dos Problemas Parcialmente Solucionáveis é equivalente à Classe das Linguagens Recursivamente Enumeráveis* .

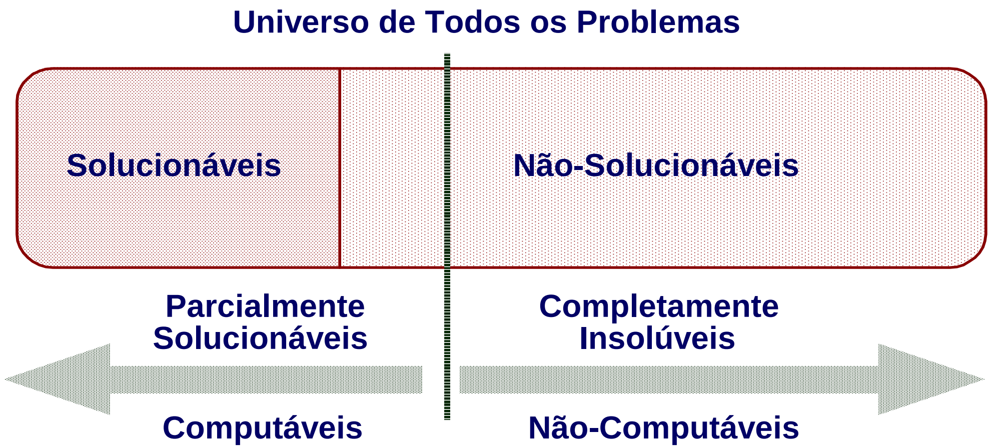
- Os Problemas *Parcialmente Solucionáveis* são Computáveis



Computabilidade

- Divisões das Classes de Problemas:
  - Problemas Computáveis (*Contáveis*).
  - Problemas Não-computáveis (*Não-contáveis*).

- A Classe dos Problemas Não-Computáveis é "muito maior" do que a Classe dos Problemas Computáveis.

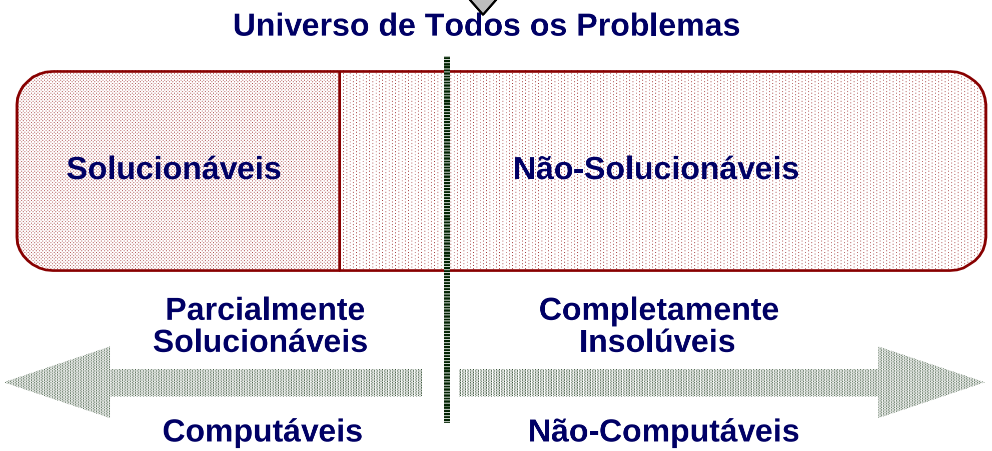


Computabilidade

- Divisões das Classes de Problemas:
  - Problemas C
  - Problemas Não *veis)*.

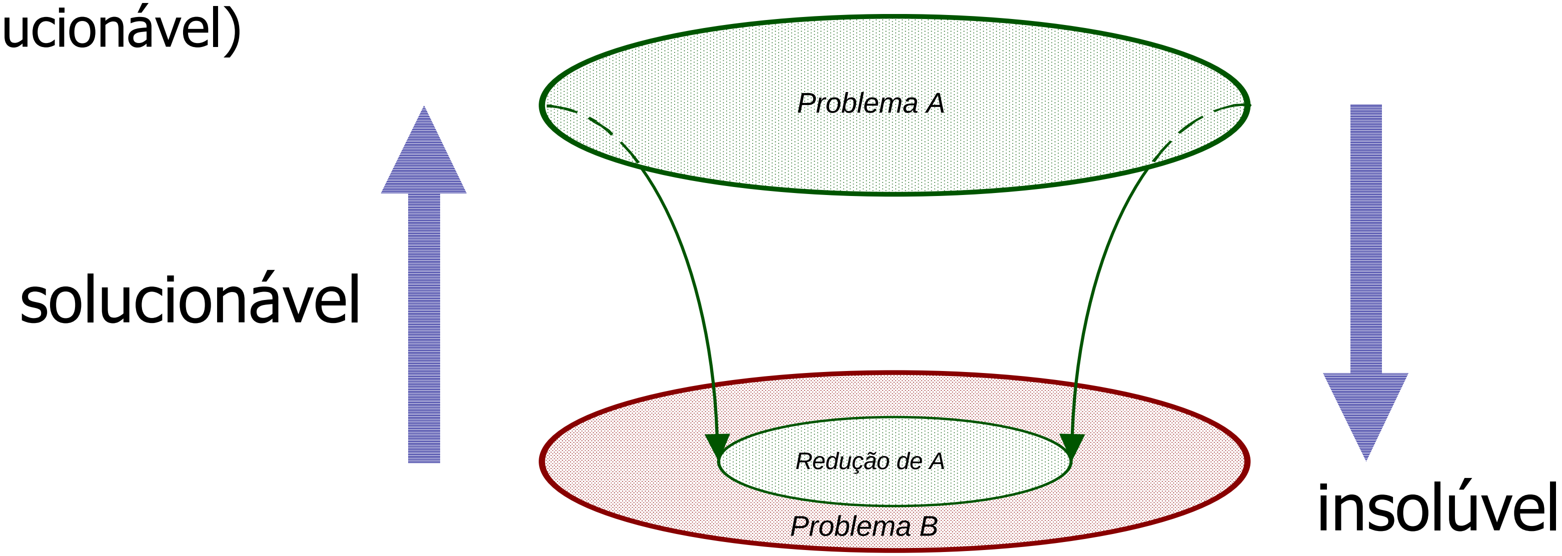
Como constatar esta divisão?

- A Classe do Computáveis é "muito maior" do que a Classe d Problemas Computáveis.



- *Princípio da Redução:* Investigar a solucionabilidade de um problema a partir de outro, cuja classe de solucionabilidade é conhecida.

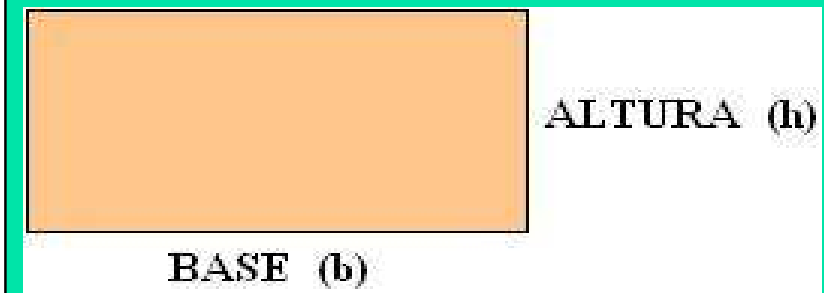
  - Sejam A e B dois problemas de decisão. Suponha que é possível modificar (reduzir) A de tal forma que ele se porte como um caso de B.
  - Se A é não-solucionável (não-computável), então, como A é um caso de B, conclui-se que B também é não-solucionável.
  - Se B é solucionável (parcialmente solucionável) então, como A é um caso de B, conclui-se que A também é solucionável (parcialmente-solucionável)



Princípio da Redução

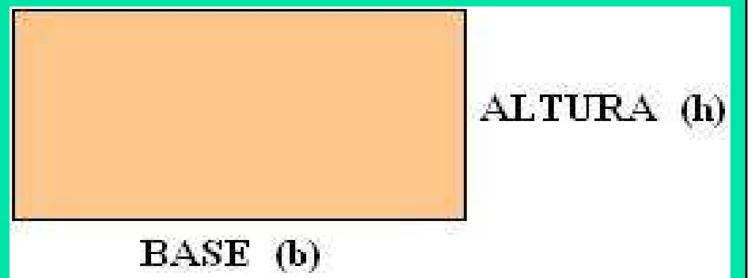
- Exemplo:

Problema 1: obter a área do retângulo



reduz

Problema 2: obter informações de ALTURA e BASE do retângulo



Classes de Solucionabilidade de Problemas

- **Problema Solucionável**: existe um algoritmo (Máquina Universal) que soluciona o problema, tal que sempre pare para qualquer entrada com uma resposta afirmativa (ACEITA) ou negativa (REJEITA).

- **Problema Não-Solucionável**: não existe um algoritmo (Máquina Universal) que solucione o problema tal que sempre pare para qualquer entrada.

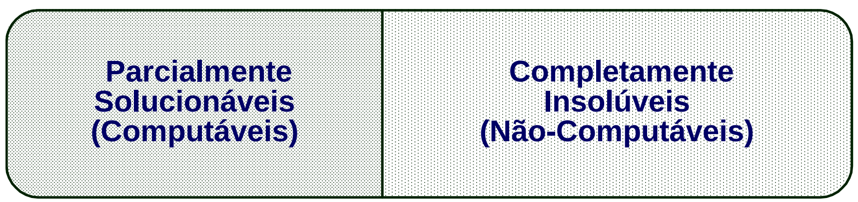
Universo de Todos os Problemas

Classes de Solucionabilidade de Problemas

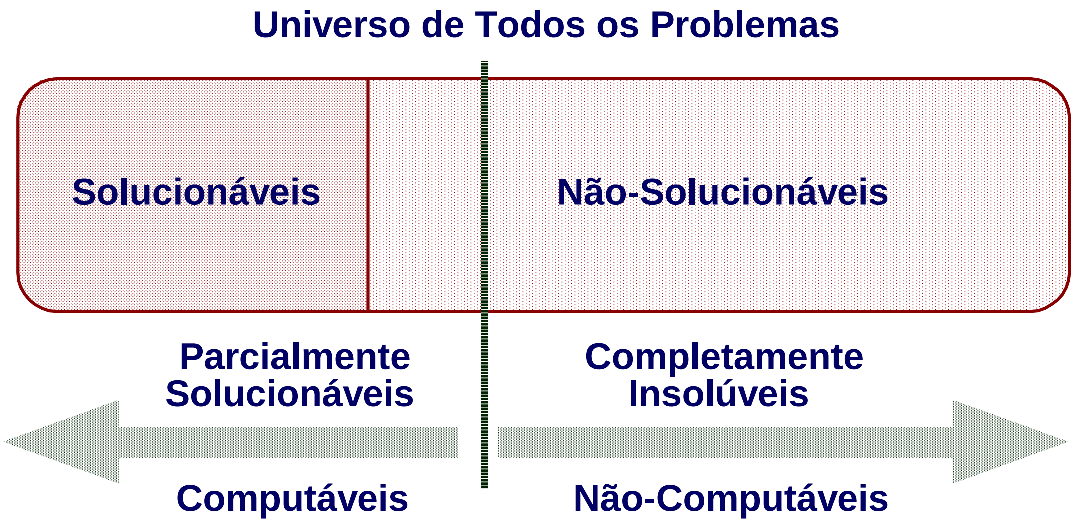
- **Problema Parcialmente Solucionável ou Computável:** existe um algoritmo (Máquina Universal) tal que sempre para quando a resposta é afirmativa (ACEITA). Entretanto, quando a resposta esperada for negativa, o programa pode parar (REJEITA) ou permanecer processando indefinidamente (LOOP).

- **Problema Completamente Insolúvel ou Não-Computável:** não existe um algoritmo (Máquina Universal) que solucione o problema tal que sempre para quando a resposta é afirmativa (ACEITA).

Universo de Todos os Problemas



Relação entre as Classes de Problemas



- A **união** das classes dos problemas **Solucionáveis** com a dos **não-solucionáveis** é o **universo de todos os problemas**.
- A união das classes dos problemas **Parcialmente Solucionáveis** com a dos **Completamente Insolúveis** é o **universo de todos os problemas**.
- A classe dos problemas **Parcialmente Solucionáveis** contem propriamente a classe dos problemas **Solucionáveis** e parte da classe dos problemas **Não-solucionáveis.**
- Todo problema **Solucionável** é **Parcialmente Solucionável**.
- Existem problemas **Não-solucionáveis** que possuem soluções parciais (ACEITA/REJEITA).
- Os problemas **Completamente Insolúveis** não possuem solução total nem parcial.

Problemas de Decisão

- São problemas do tipo SIM/NÃO com o objetivo de investigar a computabilidade.

- A ideia básica é verificar se a função associada a eles é computável em uma máquina universal.

Problemas de Decisão

- A propriedade de um problema de decisão é dada pela seguinte ideia:
  - Dado um programa P para uma máquina universal M, decidir se a função computada (P, M) é total, ou seja, se a correspondente computação é finita.
  - Sendo assim, a não-solucionabilidade refere-se à inexistência de um método geral para decidir se um programa para uma máquina universal pare para qualquer entrada.

Problemas de Decisão

- *Relembrando..:* A existência de programas não solucionáveis é importante por diversas razões, como por exemplo:

  - Alguns desses problemas não-solucionáveis permitem estabelecer importantes resultados para a Ciência da Computação (como a inexistência de um detector universal de loops).

  - Demonstrar limitações da capacidade de se expressar soluções através de programas.

Problemas de Decisão

- A investigação da solucionabilidade de programas em máquinas universais pode ser vista como um problema de reconhecimento de linguagens:

  - O problema é reescrito como um problema de decisão capaz de gerar respostas do tipo SIM/NÃO.

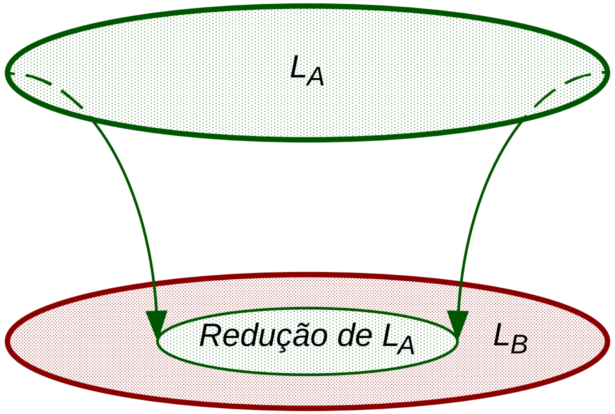
  - Os argumentos do problema SIM/NÃO são codificados como palavras de um alfabeto, gerando uma linguagem.

  - A questão da solucionabilidade de um problema pode ser traduzida como uma investigação se a linguagem gerada é recursiva (problema solucionável) ou enumerável recursivamente (problema parcialmente solucionável).

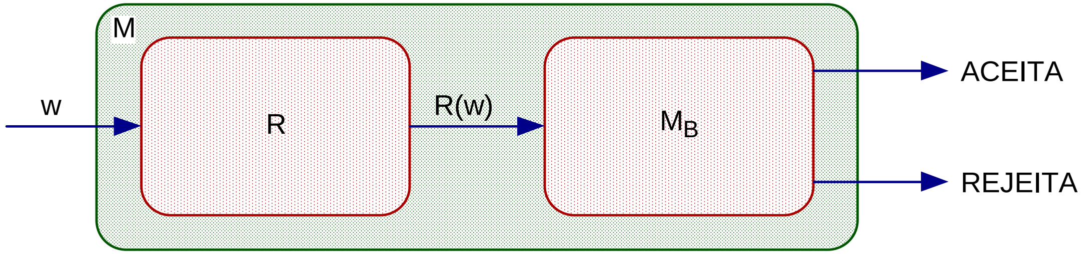
Princípio da Redução

- Consiste basicamente na construção de um algoritmo de mapeamento entre as linguagens que traduzem os problemas.
- **Máquina de Redução**: Suponha dois problemas, A e B e as correspondentes linguagens L_A e L_B. Uma Máquina de Redução R de L_A para L_B sobre um alfabeto Σ é tal que, para $w \in \Sigma$:
  - Se $w \in L_A$, então $R(w) \in L_B$.
  - Se $w \notin L_A$, então $R(w) \notin L_B$.
- Os seguintes resultados são válidos:
  - Se L_B é recursiva, então L_A é recursiva
  - Se L_B é recursivamente enumerável, então L_A é recursivamente enumerável
  - Se L_A não é recursiva, então L_B não é recursiva.
  - Se L_A não é enumerável recursivamente, então L_B não é enumerável recursivamente.

1. Exemplo: Seja R Máquina de Turing de Redução que sempre para e que reduz L_A a L_B.



- Suponha que L_B é uma linguagem recursiva. Então existe M_B, Máquina Universal, que aceita L_B e sempre pára para qualquer entrada.
  - Seja a Máquina Universal M definida



- As seguintes conclusões podem ser estabelecidas:
  - M sempre pára para qualquer entrada, pois R e M_B sempre param;
  - se $w \in L_A$, então M aceita w, pois $R(w) \in L_B$
  - se $w \notin L_A$, então M rejeita w, pois $R(w) \notin L_B$
  - **Portanto, M aceita L_A e sempre pára para qualquer entrada.**
- Logo, L_A é uma linguagem recursiva

Princípio da redução

- Exemplo:

- Para um aluno se formar em Ciência da Computação precisa ser aprovado em todas as disciplinas que compõem a grade curricular do curso, entre elas, teoria da computação.

Princípio da redução

- Exemplo:

- Para um aluno se formar em Ciência da Computação precisa ser aprovado em todas as disciplinas que compõem a grade curricular do curso, entre elas, teoria da computação.

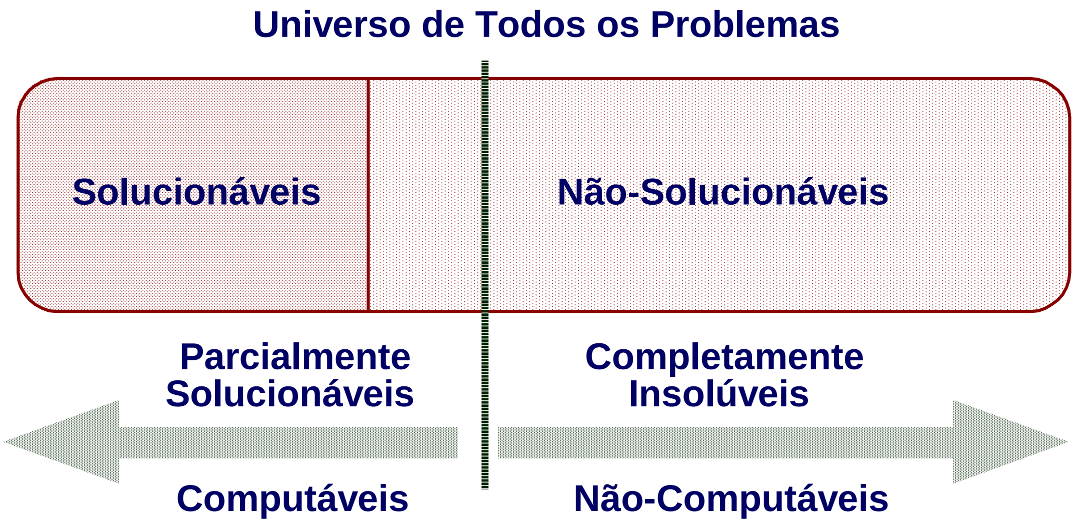
- **Problema A:** "O aluno foi aprovado em teoria da computação?"
- **Problema B:** "O aluno vai se formar?"

Princípio da redução

- Se um aluno *x* não é aprovado em teoria da computação, ou seja, a resposta de A é **não**, pode-se determinar a solução de B para o aluno *x*.

- Se um aluno *y* tem sua formatura confirmada, ou seja, a resposta de B é **sim**, logo, o aluno *y* também foi aprovado em teoria da computação (e nas demais disciplinas), e, portanto, a resposta de A também é **sim**.

Classes de Problemas



- Como classificar problemas considerando questões de tempo?

Complexidade Computacional
➔ Máquina de Turing

Exercícios

- Livro *Teoria da Computação: Máquinas Universais e Computabilidade* (Tiarajú) → Exercícios: 5.1, 5.4, 5.8 e 5.9.

- Dê três exemplos para cada um dos tipos de problemas considerando o universo total de problemas: solucionáveis, parcialmente solucionáveis e completamente insolúveis. Explique como que foi feita a classificaçãa em cada uma das categorias.