Computabilidade

Teoria da Computação

INF05501

Solucionabilidade de Problemas

- **Objetivo** do estudo da solucionabilidade de problemas é investigar a **existência ou não de algoritmos que solucionem determinada classe de problemas**

- Isto é, quer-se determinar os limites da computabilidade

Solucionabilidade de Problemas (cont.)

- Sabendo-se que um problema não tem solução, **evita-se gasto de esforço e tempo em tentar solucioná-lo**

- Por exemplo, o décimo problema de Hilbert:

  "Existe ou não um algoritmo que determine se uma equação polinomial, com coeficientes inteiros, possui solução nos inteiros?"

- Em 1970, Matijasevic provou ser tal **problema sem solução**

Classes de Solucionabilidade de Problemas

- Problema Solucionável

  Um problema é dito Solucionável ou Totalmente Solucionável se existe um algoritmo (Máquina Universal) que solucione o problema tal que sempre pare para qualquer entrada, com uma resposta afirmativa (ACEITA) ou negativa (REJEITA)

Classes de Solucionabilidade de Problemas

- Problema Solucionável

  Um problema é dito Solucionável ou Totalmente Solucionável se existe um algoritmo (Máquina Universal) que solucione o problema tal que sempre pare para qualquer entrada, com uma resposta afirmativa (ACEITA) ou negativa (REJEITA)

- Logo, a Classe dos Problemas Solucionáveis é equivalente à Classe das Linguagens Recursivas

Classes de Solucionabilidade de Problemas (cont.)

- Problema Não-Solucionável

  Um problema é dito Não-Solucionável se **não** existe um algoritmo (Máquina Universal) que solucione o problema tal que sempre pare para qualquer entrada

Alguns Problemas Não-Solucionáveis

- Equivalência de Compiladores: Não existe **algoritmo genérico** que **sempre pare** capaz de comparar quaisquer dois compiladores de linguagens livres do contexto (reconhecidas pelo formalismo Autômato Não-Determinístico com Uma Pilha), tais como Pascal, verificando se são equivalentes (se reconhecem a mesma linguagem)

- Detector Universal de Loops: Dados um programa e uma entrada quaisquer, não existe **algoritmo genérico** capaz de verificar se o programa **vai parar ou não** para a entrada (Problema da Parada)

Não-Solucionabilidade

- Refere-se à **inexistência** de um método geral e efetivo para decidir se um **programa** para uma máquina universal **para para qualquer entrada**
- Portanto, **é possível existirem métodos específicos para programas particulares**
- Importância de problemas não-solucionáveis:
  - Permitem estabelecer, por si sós, **importantes resultados**
  - Demonstram as **limitações** da capacidade de expressarem-se **soluções através de programas**
  - Podem ser usados para **verificar que outros problemas também são não-solucionáveis**, usando o princípio da redução

Classes de Solucionabilidade de Problemas (cont.)

- Problema Parcialmente Solucionável

  Um problema é dito Parcialmente Solucionável ou Computável se existe um algoritmo (Máquina Universal) que solucione o problema tal que pare quando a resposta for afirmativa (ACEITA), mas, em caso de **resposta negativa, pode parar (REJEITA) ou processar indefinidamente (LOOP)**

Solucionabilidade Parcial

- Alguns problemas não-solucionáveis possuem **soluções parciais** (são computáveis)

- Isto é, existe um algoritmo **capaz de responder "sim"**, embora **possa ficar em loop infinito** para uma **resposta** que deveria ser **"não"**

Solucionabilidade Parcial

- Alguns problemas não-solucionáveis possuem **soluções parciais** (são computáveis)

- Isto é, existe um algoritmo **capaz de responder "sim"**, embora **possa ficar em loop infinito** para uma **resposta** que deveria ser **"não"**

- Logo, a Classe dos Problemas Parcialmente Solucionáveis é equivalente à Classe das Linguagens Enumeráveis Recursivamente

Classes de Solucionabilidade de Problemas (cont.)

- Problema Completamente Insolúvel

  Um problema é dito Completamente Insolúvel ou Não-Computável se **não** existe um algoritmo (Máquina Universal) que solucione o problema tal que pare quando a resposta for afirmativa (ACEITA)

- Problemas completamente insolúveis não possuem solução total nem parcial

Relacionamentos entre Classes

- Classe dos Problemas Parcialmente Solucionáveis contém propriamente a Classe dos Problemas Solucionáveis e parte da Classe dos Não-Solucionáveis

- Logo:
  - Todo **problema solucionável** é **parcialmente solucionável**
  - Existem **problemas não-solucionáveis** que possuem **solução parcial**

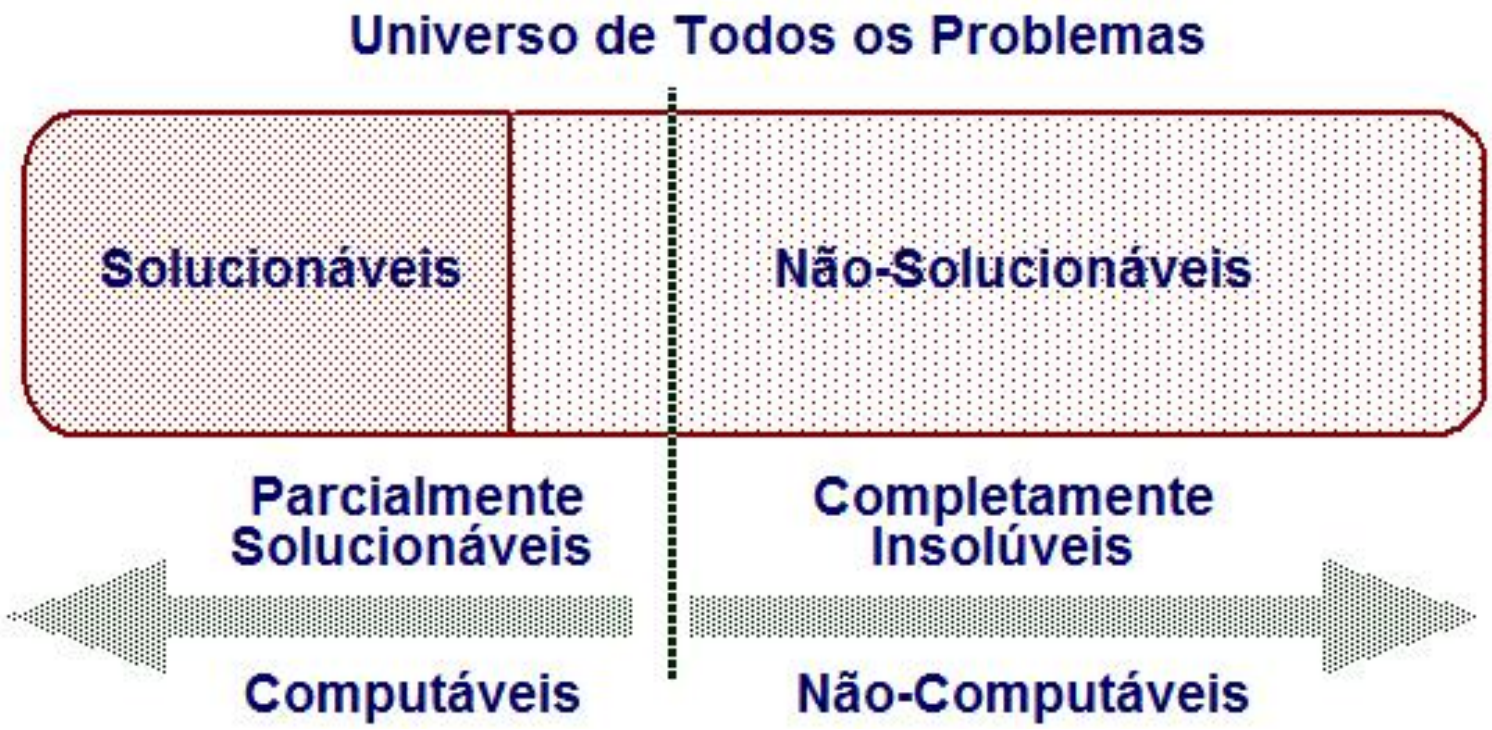
Relacionamentos entre Classes (cont.)

- Para qualquer algoritmo que solucione um **problema parcialmente solucionável** que é **não-solucionável**, sempre **existe pelo menos uma palavra de entrada que faz com que o algoritmo fique em loop**

- União da Classe dos Problemas Solucionáveis com a Classe dos Problemas Não-Solucionáveis é o Universo de Todos os Problemas

- União da Classe dos Problemas Parcialmente Solucionáveis com a Classe dos Problemas Completamente Insolúveis é o Universo de Todos os Problemas

Relacionamentos entre Classes (cont.)



Classes de Problemas

- Cardinal da Classe dos Problemas Computáveis é **contável**
- Cardinal da Classe dos Problemas Não-Computáveis é **não-contável**
- Assim, cardinal da Classe dos Problemas Não-Computáveis é "muito maior" que o da Classe dos Problemas Computáveis
- Logo, existem muito mais problemas não-computáveis do que computáveis

Princípio da Redução

- Para o estudo da solucionabilidade de um problema, pode-se partir de um **outro problema cuja classe de solucionabilidade seja conhecida**

- Isto é, dados dois problemas de decisão A e B pode ser possível **modificar ("reduzir") o problema** A de tal forma que ele **se porte como um caso do problema** B

Princípio da Redução

- Se A é não-solucionável (respectivamente, não-computável), então, como A é um caso de B, conclui-se que B também é não-solucionável (respectivamente, não-computável)

- Se B é solucionável (respectivamente, parcialmente solucionável), então, como A é um caso de B, conclui-se que A também é solucionável (respectivamente, parcialmente solucionável)

Problemas de Decisão

- Abordagem tradicional de análise de solucionabilidade concentra-se nos problemas de decisão (respostas “sim” ou “não”)
- Dado um **programa** P para uma **máquina universal** M, **decidir** se a função computada $\langle P, M \rangle$ é total (ou seja, se a **correspondente computação é finita**)
- São **mais fáceis de verificar**, visto que só há duas respostas possíveis
- Em geral, problemas de outros tipos podem ser (e são) **transformados em problemas de decisão**

Problemas de Decisão (cont.)

- Da mesma forma, a questão da solucionabilidade de problemas pode ser transformada em um problema de decisão

- Neste caso, investiga-se a **solucionabilidade de um problema através de verificação do problema de reconhecimento de linguagens** (o qual é um problema de decisão)

  – Problema é reescrito como um problema de decisão

  – **Argumentos** do problema são codificados como **palavras de um alfabeto**, gerando uma linguagem

Problema de Decisão (cont.)

- Assim, 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)

- Nesta situação
  - Problemas solucionáveis são ditos decidíveis
  - Problemas não-solucionáveis são ditos indecidíveis
  - Problemas parcialmente solucionáveis são ditos semidecidíveis

Codificação de Programas

- Veremos uma abordagem é específica para **programas monolíticos da Máquina Norma**
- Vimos a **função de codificação de programas como número naturais**, aqui denominada cod
- No entanto, ela **não é bijetora**, pois nem todo natural é codificação de algum programa

Codificação de Programas (cont.)

Seja $\mathbb{P}$ o **conjunto de todos os programas** do tipo que está sendo considerado (monolítico, iterativo ou recursivo). Considere a sequência de naturais $p_0,\ p_1,\ p_2,$ formada pelos **códigos dos programas de** $\mathbb{P}$ (ordenados pela relação "menor" sobre os naturais).

Então:

$cod\_bij = \lambda p.cod\_bij(p) : P \rightarrow \mathbb{N}$

$cod\_bij(p) = n$, para qualquer $p \in \mathbb{P}$, se o código de p é p_n

$cod\_bij^{-1} = \lambda n.cod\_bij^{-1}(n) : \mathbb{N} \rightarrow P$

$cod\_bij^{-1}(n) = p$, para qualquer $n \in \mathbb{N}$

Codificação de Programas (cont.)

- Algoritmo para determinar $cod\_bij(P)$
  1. Calcula $p = cod(P)$
  2. Verifica (usando $decod$) quantos números naturais menores do que p são codificações de programas
  3. Se forem $n - 1$, então $cod\_bij(P) = n$

- Algoritmo para determinar $cod\_bij^{-1}(n)$
  1. Verifica (usando $decod$) os n primeiros números naturais que denotam codificações de programas
  2. Se p é o n-ésimo natural, então o n-ésimo programa será a decodificação de p