Teoria da Computação

Computabilidade

Cristiano Lehrer, M.Sc.

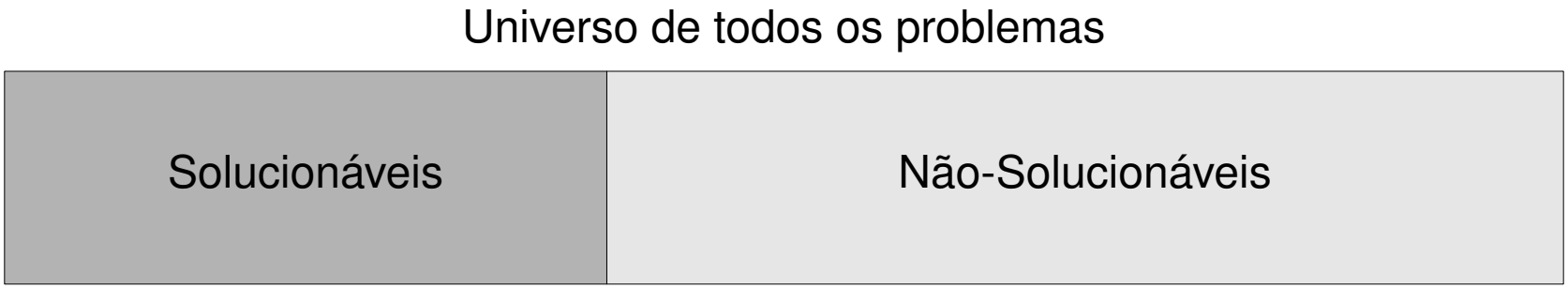
ybadoo

Introdução

- O objetivo do estudo da solucionabilidade de problemas é investigar a existência ou não de algoritmos que solucionem determinada classe de problemas.
- Ou seja, investigar os limites da computabilidade e, consequentemente, os limites do que pode efetivamente ser implementado em um computador.
- Em particular, o estudo da solucionabilidade objetiva evitar a pesquisa de soluções inexistentes.

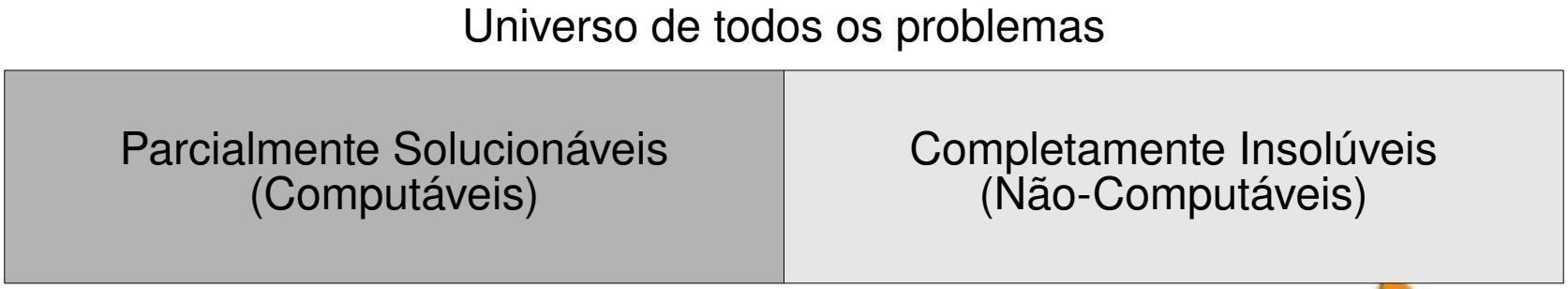
Classes de Solucionabilidade de Problemas (1/2)

- Problema Solucionável
  - Um problema é dito Solucionável ou Totalmente Solucionável se existe um algoritmo (Máquina Universal) que solucione o problema tal que sempre termine para qualquer entrada, com uma resposta afirmativa (aceita) ou negativa (rejeita).
- Problema Não-solucionável
  - Um problema é dito Não-Solucionável se não existe um algoritmo (Máquina Universal) que solucione o problema tal que sempre termine para qualquer entrada.

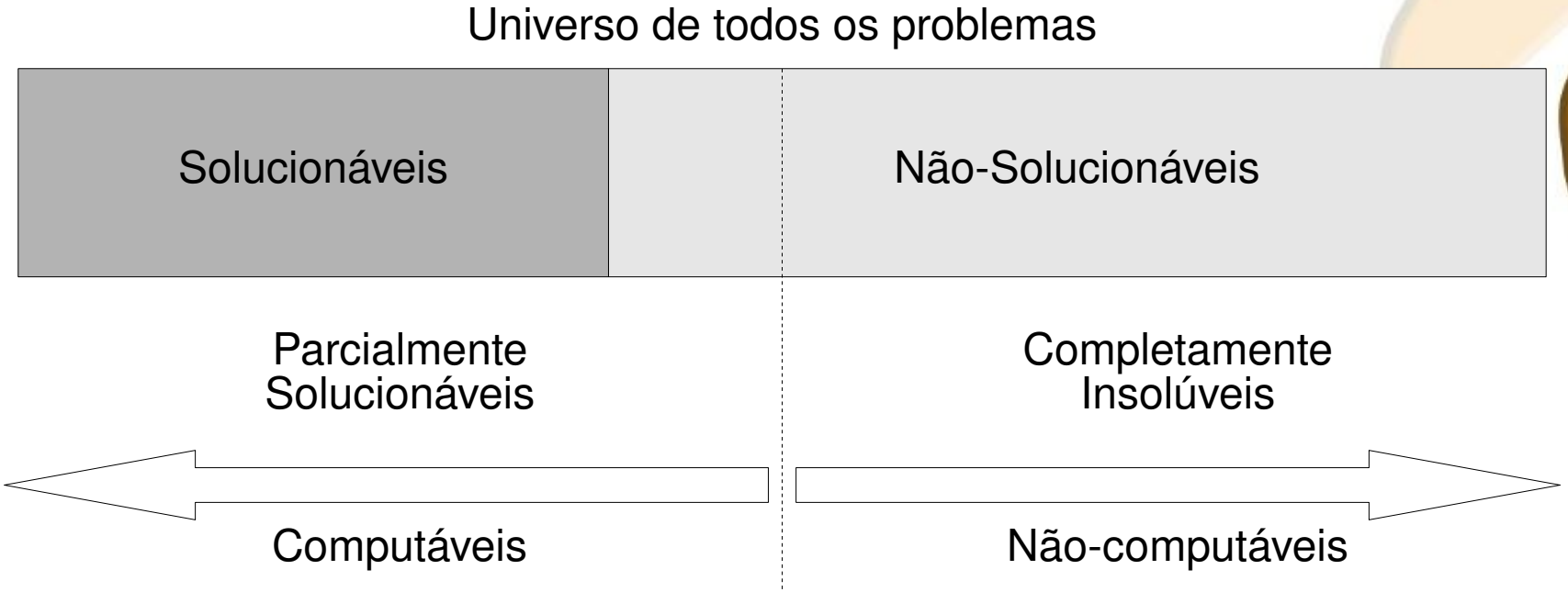


Classes de Solucionabilidade de Problemas (2/2)

- Problema Parcialmente Solucionável ou Computável
  - Um problema é dito Parcialmente Solucionável ou Computável se existe um algoritmo (Máquina Universal) que solucione o problema tal que termine quando a resposta é afirmativa (aceita).
  - Entretanto, quando a resposta esperada for negativa, o algoritmo pode terminar (rejeita) ou permanecer processando indefinidamente (*loop*).
- Problema Completamente Insolúvel ou Não-Computável
  - Um problema é dito Completamente Insolúvel ou Não-Computável se não existe um algoritmo (Máquina Universal) que solucione o problema tal que termine quando a resposta é afirmativa (aceita).



Conclusão (1/3)



Conclusão (2/3)

- A união das Classes dos Problemas Solucionáveis e dos Não-Solucionáveis é o Universo de Todos os Problemas.
- A união das Classes dos Problemas Parcialmente Solucionáveis e dos Completamente Insolúveis é o Universo de Todos os Problemas.
- A Classe dos Problemas Parcialmente Solucionáveis contém 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 solução parcial.
- Os problemas completamente insolúveis não possuem solução total nem parcial.

Conclusão (3/3)

- 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*.
- É importante observar que alguns problemas não-solucionáveis são parcialmente solucionáveis.

P versus NP (1/5)

- P = NP ou P ≠ NP ?!?
  - P → *polynomial time*
  - NP → *nondeterministic polynomial time*
  - O problema P versus NP é um dos Problemas do Prêmio Millenium (*Millenium Prize Problems*) proposto pelo Instituto Clay de Matemática (*Clay Mathematics Institute*).
  - Existe um prêmio de um milhão de dólares para quem resolver!
- A comunidade científica considera P ≠ NP

P versus NP (2/5)

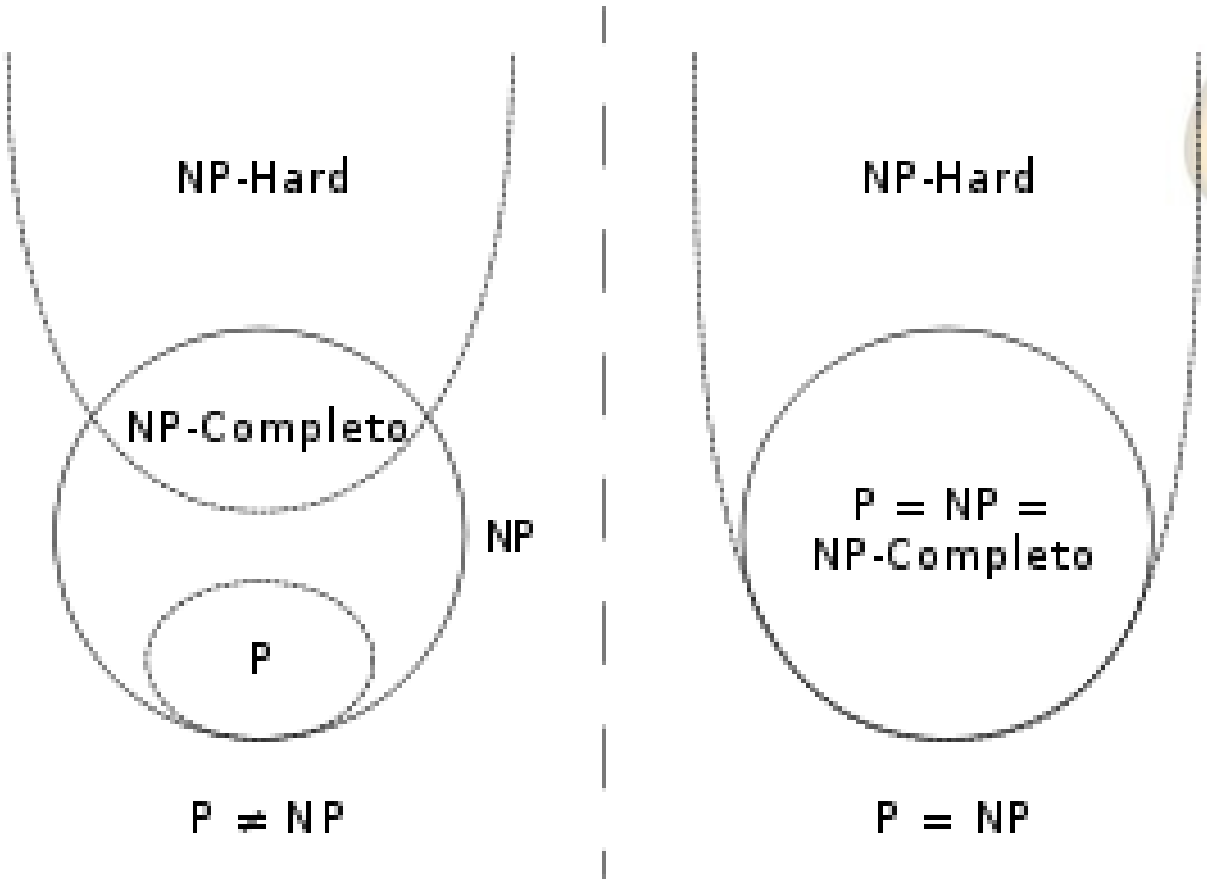
- Classe NP
  - A classe NP é a classe de **TODOS** os problemas de busca cuja solução pode ser encontrada e verificada em tempo polinomial por um algoritmo não determinístico.
    - Um algoritmo não determinístico é um tipo especial de algoritmo que adivinha corretamente em todos os passos.
- Classe P
  - A classe P é a classe de **TODOS** os problemas de busca que são resolvidos em tempo polinomial.
- Obs.: A definição original de NP não foi para problemas de busca, mas para problemas de decisão, que são perguntas algorítmicas que podem ser respondidas por SIM ou NÃO.

P versus NP (3/5)

- Classe NP-Completo
  - Um problema de busca é NP-Completo se todos os problemas de busca se reduzem a ele, ou seja, ele deve ser capaz de resolver todos os problemas de busca do mundo!
- Uma redução de um problema de busca **A** para um problema de busca **B** são dois algoritmos de tempo polinomial **f** e **h** que:
  - **f** transforma qualquer instância **x** de **A** em uma instância **f(x)** de **B**;
  - **h** transforma qualquer solução **S** de **f(x)** de volta em solução **h(S)** de **x**;
  - Se **f(x)** não tem solução então **x** também não tem.

ybadoo

P versus NP (4/5)



P versus NP (5/5)

- NP-Hard ou NP-Difícil ou NP-Complexo:
  - É uma classe de problemas que são, informalmente, pelo menos tão difíceis quanto os problemas mais difíceis em NP.
  - Um problema é NP-Hard se e somente se existe um problema NP-Completo *L* que é Turing-redutível em tempo polinomial para *H*.
  - Em outras palavras, *L* pode ser resolvido em tempo polinomial por uma Máquina de Turing não determinística com um oráculo para *H*.
  - Exemplo de problema NP-Hard e também NP-Completo:
    - Problema do Caixeiro Viajante.
  - Exemplo de problema NP-Hard que não é NP-Completo:
    - Problema da Parada.

Problema da Parada (1/2)

- Um dos mais importantes problemas não-solucionáveis conhecido.
- Definição formal:
  - Dada uma Máquina Universal *M* qualquer e uma palavra *w* qualquer sobre o alfabeto de entrada, existe um algoritmo que verifica se *M* finaliza, aceitando ou rejeitando, ao processar a entrada *w*?
- É um problema de decisão, do tipo sim/não.
- Alan Turing provou, em 1936, que um algoritmo genérico para resolver o problema da parada para todos os pares programa/entrada possíveis não pode existir.
- O problema da parada é indecidível.

Problema da Parada (2/2)

- Exemplos do Problema da Parada, cuja insolubilidade já foi provada, que podemos encontrar atualmente:
  - Garantir que um antivírus jamais executa código malicioso.
  - Garantir que um determinado compilador produza o código mais rápido possível.
  - Garantir que dois *parsers* diferentes aceitem exatamente a mesma linguagem?

Hipótese de Church (1/2)

- Turing propôs um modelo abstrato de computação, conhecido como Máquina de Turing, com o objetivo de explorar os limites da capacidade de expressar soluções de problemas.
- Trata-se, portanto, de uma proposta de definição formal da noção intuitiva de algoritmo.
- Diversos outros trabalhos, como Máquina de Post, bem como trabalhos mais recentes como Máquina de Norma e Autômato com Pilhas, resultaram em conceitos equivalentes ao de Turing.
- O fato de todos esses trabalhos independentes gerarem o mesmo resultado em termos de capacidade de expressar computabilidade é um forte reforço no que é conhecido como Hipótese de Church ou Hipótese de Turing-Church.

Hipótese de Church (2/2)

"A capacidade de computação representada pela Máquina de Turing é o limite máximo que pode ser atingido por qualquer dispositivo de computação".

- Em outras palavras, a Hipótese de Church afirma que qualquer outra forma de expressar algoritmos terá, no máximo, a mesma capacidade computacional da Máquina de Turing.
- Como a noção intuitiva de algoritmo ou função computável é intuitiva, a Hipótese de Church não é demonstrável.

Resumo dos Principais Conceitos (1/5)

- O principal conceito estudado é o de computabilidade, o qual é construído, usando noções de programas, máquinas e computações.
- São três conceitos distintos, mas diretamente relacionados, pois um programa para uma máquina pode induzir uma computação.
- Se ela for finita, então se define ainda a função computada por esse programa nessa máquina:
  - Ela descreve o que o programa faz.
- A distinção entre programa e máquina é importante na Ciência da Computação, uma vez que o programa (ou algoritmo) independe da máquina e possui uma complexidade estrutural e computacional (quantidade de trabalho necessário para resolver o problema).

Resumo dos Principais Conceitos (2/5)

- A partir do conceito de função computada, podem-se fazer comparações entre programas e entre máquinas e definir-se a equivalência dos mesmos.
- Se dois programas, em uma máquina, possuem a mesma função computada, ou seja, computam a mesma função, então eles são equivalentes.
- Se esses programas são equivalentes em qualquer máquina, então eles são equivalentes fortemente.
- Utiliza-se o conceito de equivalência de programas para eliminar instruções desnecessárias e para otimizar o programa.
- Dessa forma, pode-se dizer que um programa otimizado representa uma classe de programas que são equivalentes.

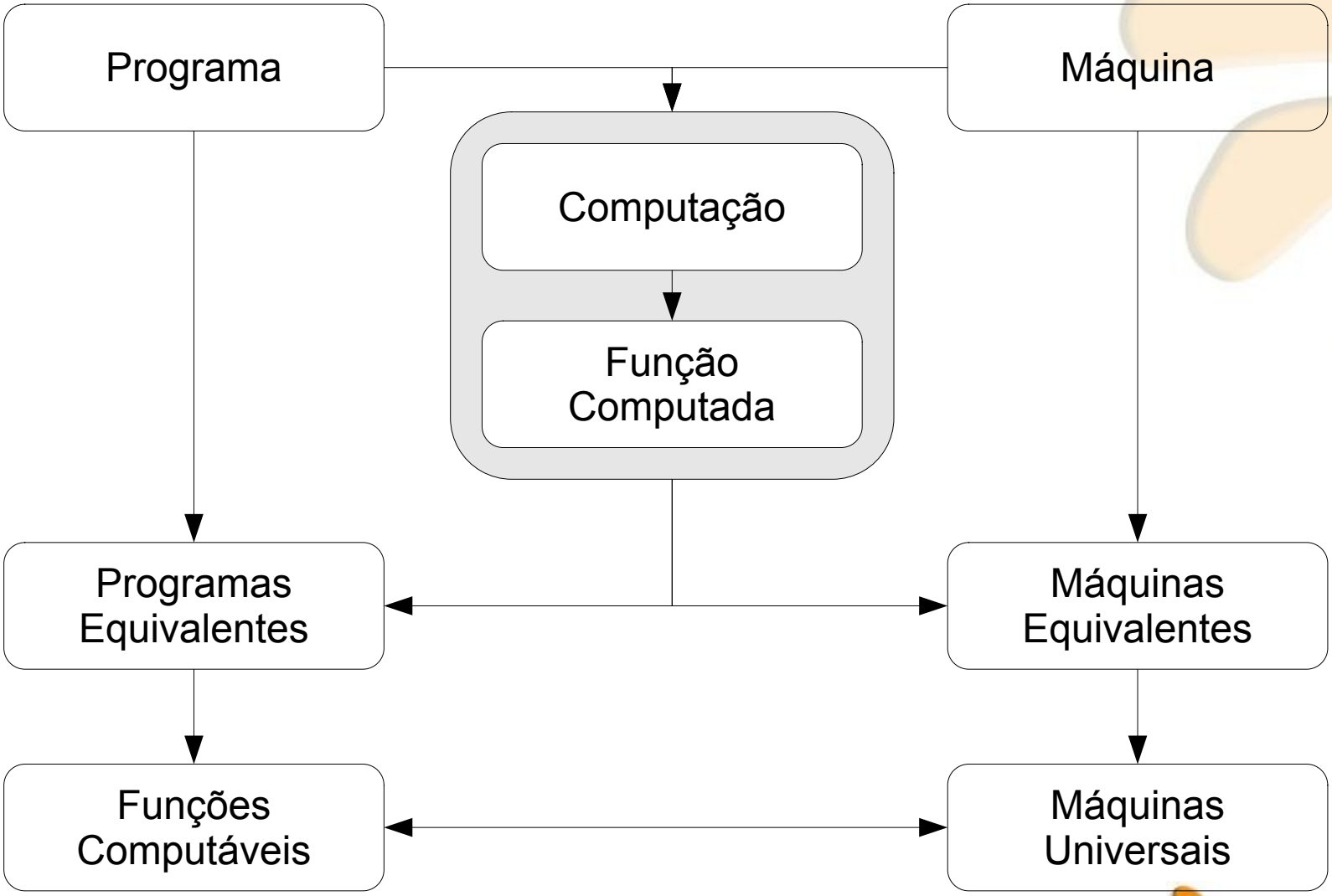
Resumo dos Principais Conceitos (3/5)

- A comparação entre máquinas é feita em função da noção de simulação, que, por sua vez, utiliza-se do conceito de equivalência de programas:
  - Se uma máquina simula outra, é porque, para qualquer programa da outra máquina, pode-se encontrar um programa desta que faça a mesma coisa.
- Se duas máquinas simulam-se mutuamente, é porque elas são equivalentes.
- Nesse caso, ambas têm o mesmo poder computacional.
- Foi observado que nem sempre o fato de acrescentar instruções ou recursos a uma máquina aumenta o poder computacional da mesma.
- Muitas vezes, ocorre que essa nova instrução ou esse novo recurso podem ser simulados pela máquina anterior, mantendo o seu poder computacional.

Resumo dos Principais Conceitos (4/5)

- Neste ponto, pode-se pensar em suas situações limites:
  - Qual a máquina mais poderosa?
  - Qual o conjunto ou a classe de funções computáveis?
- As respostas dessas perguntas são intuitivas, mas nem sempre são fáceis de serem verificadas.
- Diz-se que uma máquina é universal se toda função computável puder ser executada nela.
- E, segundo a Hipótese de Church, diz-se que uma função computável é aquela que pode ser processada numa Máquina de Turing ou equivalente.
- Assim, não se conseguem demostrar tais afirmações, devido ao fato de a noção de algoritmo ser algo intuitivo.
- Entretanto, diversas evidências fortalecem a Hipótese de Church.

Resumo dos Principais Conceitos (5/5)



ybadoo