Insper

Megadados

Aula 22 – Programação funcional

2021 – Engenharia

Fábio Ayres <fabioja@insper.edu.br>

Engenharia de Computação

“Fazer computação acontecer" é uma atividade complicada.

- Desde os elementos fundamentais de hardware digital,
- passando por sistemas operacionais e redes,
- até produtos finais *"user-facing"*.

Engenharia de Computação

Compare Engenharia de Computação com Engenharia Civil:

	Civil	Computação
Duração do projeto	Definida	Alguns projetos são bem definidos, outros não tem prazo final (e.g. serviços)
Natureza dos requerimentos	Fixos, e geralmente repetidos entre projetos	As vezes são bem definidos, muitas vezes o cliente e a equipe estão descobrindo os detalhes dos requerimentos iterativamente
Fase de design	Longa, todos os requerimentos devem ser analisados antes da implementação (construção) Designs comuns	Muito longa: escrever software É design! Designs únicos, com alguns padrões que emergem e desaparecem
Fase de implementação	Longa: construção física One-time-only	Curta: compilação e deploy. Contínua. Replicável.

Domando a complexidade

- *A matéria-prima da engenharia de computação é complexidade*
- Todas as ferramentas da engenharia de computação existem para domar a complexidade
- A estratégia mais poderosa para domar a complexidade é a construção de abstrações

Paradigmas de programação

Abstrações úteis para conceber, projetar e implementar sistemas computacionais

- Programação procedural
- Programação orientada a objetos
- Programação orientada a eventos
- Programação funcional
- Programação reativa
- Etc...

Interlúdio: o que é computação?

Na década de 1930 vários matemáticos buscaram o significado de "computável", em resposta a um desafio lançado por David Hilbert em 1928:

o Entscheidungsproblem

(ou problema da decisão)

Existe algum algoritmo que consiga sempre provar (verdadeira ou falsa) uma afirmação lógica (de primeira ordem) a partir dos axiomas e das regras da lógica?

Entscheidungsproblem

A resposta é não.

Em 1936, Alonzo Church e Alan Turing chegaram a essa resposta por meios diferentes, de modo independente.

Para conseguir responder o problema, Church e Turing tiveram que definir matematicamente a computação.

O que é computação?

- Alonzo Church
  - Cálculo lambda

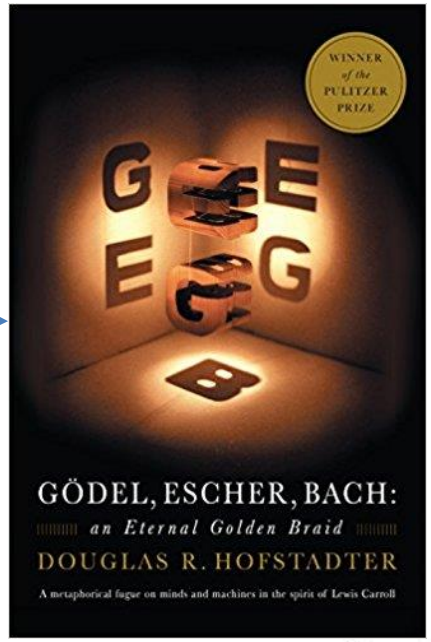
- Alan Turing
  - Máquinas de Turing

Interlúdio do interlúdio: Kurt Gödel

Publicou em 1931 "On Formally Undecidable Propositions of Principia Mathematica and Related Systems"

- *Qualquer sistema axiomático computável capaz de descrever a aritmética contém proposições verdadeiras e não demonstráveis.*
- Os resultados de Church e Turing foram baseados nos resultados de Gödel

Leitura super interessante!



Insper

Finalmente, o que é computação?

Em termos simples: uma função definida nos números naturais é computável se e somente se:

- [Alan Turing]: pode ser computada por uma máquina de Turing
- [Alonzo Church]: pode ser computada pelo cálculo lambda

Esta é a tese de Church-Turing

Cálculo Lambda

- Representa lógica, números, etc, como funções
- Estuda mais como combinar funções do que como aplicá-las a dados concretos

Cálculo Lambda

Exemplo de lambda:

$\lambda x.x$ – Função identidade

$(\lambda x.x + 1)\ 10$ – Função "soma 1", aplicada ao valor 10, resultando no valor 11

$(\lambda x.\lambda y.x + y)$ – Função que recebe um valor x_0 e retorna uma função $(\lambda y.x_0 + y)$

$(\lambda x.\lambda y.x + y)\ 10$ – Aplicação do resultado anterior para $x_0 = 10$, resultando na função $(\lambda y.10 + y)$

Cálculo Lambda

Exemplo de lambda:

$$(\lambda x.\lambda y.x + y)\ 10\ 20$$
$$\rightarrow (\lambda y.10 + y)\ 20$$
$$\rightarrow 10 + 20$$
$$\rightarrow 30$$

Cálculo Lambda

Considere os seguintes termos lambda:

$$TRUE \equiv \lambda x.\lambda y.x$$
$$FALSE \equiv \lambda x.\lambda y.y$$

Agora considere este termo lambda:

$$IFTHENELSE \equiv \lambda v.\lambda p.\lambda q.v\ p\ q$$

Será que funciona? O professor demonstrará na lousa porque Powerpoint e matemática ninguém merece.

Outros exemplos

https://en.wikipedia.org/wiki/Lambda_calculus

AND := λp.λq.p q p
OR := λp.λq.p p q
NOT := λp.p FALSE TRUE

Atividade: prove que

- NOT TRUE = FALSE
- AND TRUE FALSE = FALSE
- OR TRUE FALSE = TRUE

Outros exemplos

https://en.wikipedia.org/wiki/Lambda_calculus

0 := λf.λx.x
1 := λf.λx.f x
2 := λf.λx.f (f x)
3 := λf.λx.f (f (f x))

PLUS := λm.λn.λf.λx.m f (n f x)

Atividade: prove que PLUS 1 2 = 3

Listas e loops

Pares em cálculo lambda:

$$\begin{aligned}\text{pair} &\equiv \lambda x.\, \lambda y.\, \lambda z.\, z\ x\ y \\ \text{first} &\equiv \lambda p.\, p\ (\lambda x.\, \lambda y.\, x) \\ \text{second} &\equiv \lambda p.\, p\ (\lambda x.\, \lambda y.\, y)\end{aligned}$$

Sequencias: listas ligadas construídas a partir de pares

Loops: RECURSÃO

https://en.wikipedia.org/wiki/Church_encoding

Ok, e daí? Lições do cálculo lambda

Essencial ao cálculo lambda são os seguintes conceitos:

- Funções puras (*pure functions*)
  - Não existe *estado* sendo modificado e armazenado, apenas valores sendo passados entre funções

- Funções como objetos de primeira classe (*functions as first class objects*)
  - Assim como passamos valores, podemos passar funções

Ok, e daí? Lições do cálculo lambda

- Funções de ordem superior (*higher-order functions*)
    - Funções podem receber funções e retornar funções
- Aplicação parcial de argumentos
- Loops representados como recursões
- Ausência de variáveis e atribuições – ausência de estado

Programação funcional

O cálculo lambda NÃO é uma boa alternativa para implementação direta como linguagem de programação, mas a adoção de alguns princípios pode ser vantajosa.

- Robustez
- Claridade
- Mais preocupada com “o que calcular” do que “como calcular”

A programação funcional é inspirada em:

- Cálculo lambda, e
- Teoria de categorias

Programação funcional

Linguagens de programação funcionais

- Haskell, Lisp, etc

Linguagens exclusivamente imperativas, que não tem mecanismos nativos para programação funcional:

- C, Java (antes de Java 8), etc

Linguagens mistas, com mecanismos nativos para ajudar na programação funcional:

- Python, Scala, etc

Em todas estas linguagens, contudo, é possível adotar um estilo funcional de programação!

E agora, o ponto mais importante desta aula

Programação funcional e big data

- Programação funcional descreve o que queremos calcular **sem especificar como iterar sobre os dados**
  - Facilmente paralelizável

- Programação funcional usa **funções puras**
  - Robustez: se um bloco de cálculo falha (a máquina cai), podemos reiniciar o cálculo daquele bloco apenas, sem problemas

- **Ausência de estado global**
  - Facilita uso de memória distribuída

Programação funcional e big data

Estas são características ideais para computação distribuída de grandes massas de dados!

Os principais frameworks de computação big data são inspirados em programação funcional:

- Hadoop MapReduce
- Spark

Programação funcional em Python

- Funções como objetos de primeira classe
- Lambda
- Closures e aplicação parcial de argumentos
- List comprehensions
- Algumas *higher-order functions*:
    - Map
    - Filter
    - Reduce

Insper

www.insper.edu.br