Programação e matemática não são a mesma coisa

Aula para disciplina de Métodos Formais

Gabriela Moreira

Departamento de Ciência da Computação - DCC
Universidade do Estado de Santa Catarina - UDESC

28 de fevereiro de 2024

Conteúdo

Outline

Essa aula não é sobre

Essa aula não é sobre:

- “Precisa saber matemática pra programar?”
- “Quem tem base matemática programa melhor?”

. . . que muitas vezes são derivados de “precisa ter faculdade pra trabalhar com programação?”

Essa aula não é sobre

Essa aula não é sobre:

- "Precisa saber matemática pra programar?"
- "Quem tem base matemática programa melhor?"

... que muitas vezes são derivados de "precisa ter faculdade pra trabalhar com programação?"

Vocês estão cursando Ciência da Computação, então independente disso tudo, vão sim aprender uma base lógica/matemática.

Essa aula é sobre

Como matemática e programação se relacionam, e a importância de entender as diferenças.

Métodos Formais são sobre matemática ou sobre programação?

Essa aula é sobre

Como matemática e programação se relacionam, e a importância de entender as diferenças.

Métodos Formais são sobre matemática ou sobre programação?

Vamos trabalhar com ambos no mesmo ambiente, e o domínio sobre quando usar uma perspectiva ou outra é a principal habilidade para se escrever uma boa **especificação formal**.

No geral, nós, programadores, tendemos à **perspectiva da programação**, e precisamos nos esforçar para descrever algumas coisas na **perspectiva matemática**.

Para isso, primeiro precisamos entender as semelhanças e diferenças

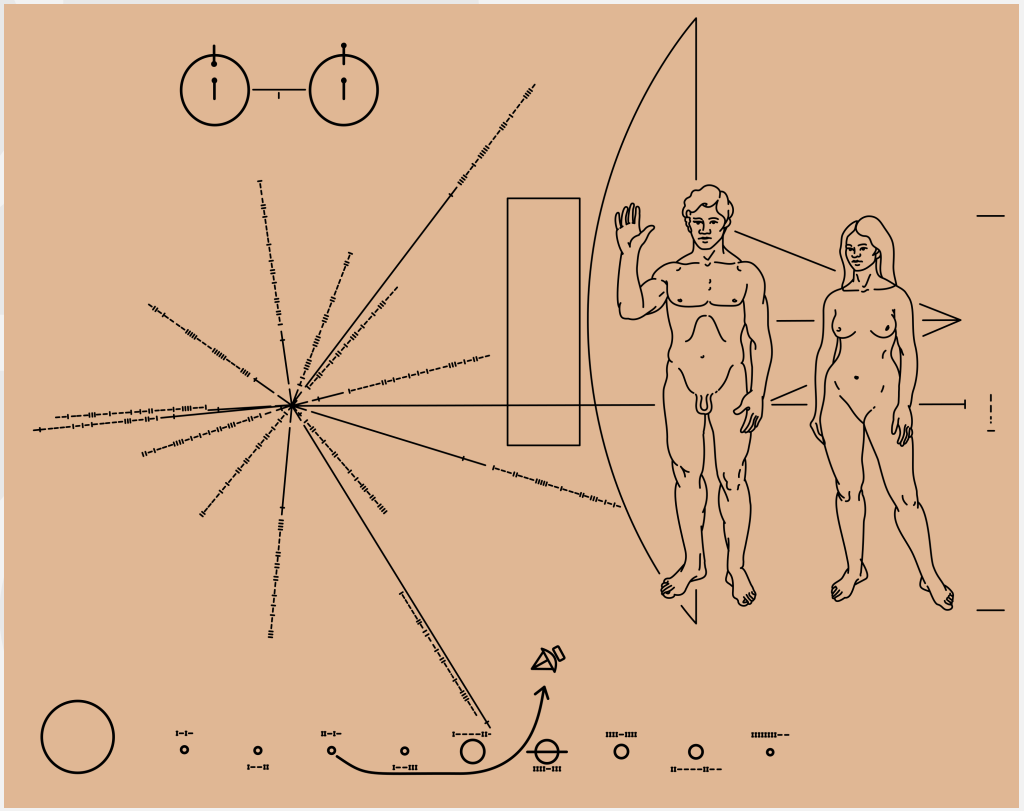
Outline

Correspondências interessantes - está tudo interligado!

- Em 1930, Church e Turing provam, de forma independente, que o problema Entscheidungsproblem não pode ser resolvido
  - Church criando o cálculo lambda
  - Turing criando as máquinas de Turing
  - São equivalentes!

Linguagem universal

- (WADLER, 2015) E se mandássemos cálculo lambda para os alienígenas? Seriam eles capazes de compreender?



Isomorfismo de Curry-Howard I

- Proposições como tipos
- Provas como programas
- Simplificação de provas como avaliação de programas

Lógica	Tipos
Falso	`Void`
Verdadeiro	`()`
$a \vee b$	`Either a b`
$a \wedge b$	`(a,b)`
$a \implies b$	`a -> b`

Isomorfismo de Curry-Howard II

Exemplo: implicação e abstração + aplicação

Lógica	Tipos
$\dfrac{}{\Gamma_1, \alpha, \Gamma_2 \vdash \alpha}\,\mathsf{Ax}$	$\dfrac{}{\Gamma_1, x : \alpha, \Gamma_2 \vdash x : \alpha}$
$\dfrac{\Gamma, \alpha \vdash \beta}{\Gamma \vdash \alpha \to \beta} \to I$	$\dfrac{\Gamma, x : \alpha \vdash t : \beta}{\Gamma \vdash \lambda x.t : \alpha \to \beta}$
$\dfrac{\Gamma \vdash \alpha \to \beta \qquad \Gamma \vdash \alpha}{\Gamma \vdash \beta} \to E$	$\dfrac{\Gamma \vdash t : \alpha \to \beta \qquad \Gamma \vdash u : \alpha}{\Gamma \vdash t\ u : \beta}$

Interpretação algébrica para tipos

- A teoria das categorias define um nível ainda mais alto de abstração para enxergar algumas coisas. Um dos exemplos mais simples de uma categoria é a categoria dos conjuntos (e das funções entre eles) (DE FRANÇA, 2019).
- As chamadas categorias cartesianas fechadas podem ser relacionadas a nossa álgebra de ensino médio
  - A categoria dos tipos é uma delas!

Lógica	Tipos	Álgebra
Falso	`Void`	0
Verdadeiro	`()`	1
$a \vee b$	`Either a b`	$a + b$
$a \wedge b$	`(a,b)`	$a * b$
$a \implies b$	`a -> b`	b^a

Exponenciação como tipos de funções I

Vamos escrever tipos função (`a -> b`) como operações de exponenciação da álgebra:

Exponenciação como tipos de funções I

Vamos escrever tipos função (`a -> b`) como operações de exponenciação da álgebra:

- $a^0 = 1$ tem assinatura `Void -> a`. Apenas uma função tem essa assinatura (em Haskell, `absurd`)

Exponenciação como tipos de funções I

Vamos escrever tipos função (`a -> b`) como operações de exponenciação da álgebra:

- $a^0 = 1$ tem assinatura `Void -> a`. Apenas uma função tem essa assinatura (em Haskell, `absurd`)
- $a^1 = a$ tem assinatura `() -> a`. O número de funções com esse tipo é o mesmo número de valores do tipo `a`.
  - Por exemplo, pra `a` sendo `bool`, temos `f x = false` e `f x = true`

Exponenciação como tipos de funções I

Vamos escrever tipos função (`a -> b`) como operações de exponenciação da álgebra:

- $a^0 = 1$ tem assinatura `Void -> a`. Apenas uma função tem essa assinatura (em Haskell, `absurd`)
- $a^1 = a$ tem assinatura `() -> a`. O número de funções com esse tipo é o mesmo número de valores do tipo `a`.
  - Por exemplo, pra `a` sendo `bool`, temos `f x = false` e `f x = true`
- $1^a = 1$ tem assinatura `a -> ()`. Apenas uma função tem essa assinatura (`f x = ()`)

Exponenciação como tipos de funções II

- a^{b+c} tem assinatura `Either b c -> a`
  - Para isso, temos que definir os casos `Left` com tipo `b -> a` e `Right` com tipo `c -> a`
  - Ou seja, $a^{b+c} = a^b * a^c$

Exponenciação como tipos de funções II

- a^{b+c} tem assinatura `Either b c -> a`
  - Para isso, temos que definir os casos `Left` com tipo `b -> a` e `Right` com tipo `c -> a`
  - Ou seja, $a^{b+c} = a^b * a^c$
- $(a^b)^c$ tem assinatura `c -> (b -> a)`
  - Lembrando de currying, sabemos que isso é equivalente a `(c,b) -> a`.
  - Ou seja, $(a^b)^c = a^{(b*c)}$

Exponenciação como tipos de funções II

- a^{b+c} tem assinatura `Either b c -> a`
  - Para isso, temos que definir os casos `Left` com tipo `b -> a` e `Right` com tipo `c -> a`
  - Ou seja, $a^{b+c} = a^b * a^c$
- $(a^b)^c$ tem assinatura `c -> (b -> a)`
  - Lembrando de currying, sabemos que isso é equivalente `a (c,b) -> a`.
  - Ou seja, $(a^b)^c = a^{(b*c)}$
- $(a * b)^c$ tem assinatura `c -> (a, b)`
  - Equivalente a um par de funções `c -> a` e `c -> b`
  - Ou seja, $(a * b)^c = a^c * b^c$

Funções parciais

Agora, um caso mais tangível para voltarmos um pouco para a nossa realidade.

- Na matemática, funções podem ser totais ou parciais
  - Para transformar funções parciais em totais, adicionamos o valor bottom ($\perp$) ao co-domínio e mapeamos todos os valores anteriormente indefinidos ao bottom.
- Na computação, funções parciais precisam retornar o tipo soma. Dependendo da linguagem, pode ser algo como:
  - `f(x: int): int | undefined`
  - `int -> Maybe int`

Outline

Erros vs indefinições

- Na matemática, algumas fórmulas são indefinidas.
  - Divisão não está definida para denominador 0
  - Exponenciação não está definida para 0^0
- Na programação, precisamos **definir** o que acontece nesses cenários
  - Normalmente, o que queremos é reportar algum tipo de erro
  - Programação envolve humanos. Humanos erram e precisam entender aonde erraram.
    - "Opa, você tentou dividir por 0 na linha X coluna Y" - pode salvar alguém de horas de debugging

Funções vs `Maps`

Funções matemáticas podem ser programadas através de funções ou `Maps` (KONNOV, 2024). Pense nos exemplos

1. Função de um número para seu dobro.
2. Função do nome da pessoa para sua idade.

Funções vs `Maps`

Funções matemáticas podem ser programadas através de funções ou `Maps` (KONNOV, 2024). Pense nos exemplos

1. Função de um número para seu dobro.
2. Função do nome da pessoa para sua idade.

Na programação, vamos considerar os fatores

- Uso de Memória
- Velocidade de resposta

Funções vs `Maps`

Funções matemáticas podem ser programadas através de funções ou `Maps` (KONNOV, 2024). Pense nos exemplos

1. Função de um número para seu dobro.
2. Função do nome da pessoa para sua idade.

Na programação, vamos considerar os fatores

- Uso de Memória
- Velocidade de resposta

Numa especificação formal, memória e velocidade não importam da mesma forma

Implementação vs definição

Imagine a seguinte definição:

- Dada uma função que ordena uma lista de inteiros

O que você pensou sobre essa função?

Implementação vs definição

Imagine a seguinte definição:

- Dada uma função que ordena uma lista de inteiros

O que você pensou sobre essa função?
Bem possível que pensou em um ou mais algoritmos de ordenação (i.e. bubble sort, selection sort, quick sort)

Implementação vs definição

Imagine a seguinte definição:

- Dada uma função que ordena uma lista de inteiros

O que você pensou sobre essa função?
Bem possível que pensou em um ou mais algoritmos de ordenação (i.e. bubble sort, selection sort, quick sort)

Na matemática, não importa **como** a ordenação é feita. A função em questão poderia ser descrita mais precisamente por:

- Seja $f : \overline{\mathbb{Z}} \to \overline{\mathbb{Z}}$ tal que $f(x)_i \leq f(x)_{i+1}$ para todo $i \in [0, |x| - 1)$

Implementação vs definição

Imagine a seguinte definição:

- Dada uma função que ordena uma lista de inteiros

O que você pensou sobre essa função?
Bem possível que pensou em um ou mais algoritmos de ordenação (i.e. bubble sort, selection sort, quick sort)

Na matemática, não importa **como** a ordenação é feita. A função em questão poderia ser descrita mais precisamente por:

- Seja $f : \overline{\mathbb{Z}} \to \overline{\mathbb{Z}}$ tal que $f(x)_i \leq f(x)_{i+1}$ para todo $i \in [0, |x| - 1)$

Numa especificação formal, se não há relevância no algoritmo de ordenação (contanto que ele, de fato, ordene), e podemos economizar recursos na verificação ao especificar somente a propriedade de ordenação.

Em resumo

- Matemática e programação estão muito interligados
- Contudo, há diferenças nos níveis de abstração entre o que costumamos descrever em definições matemáticas e em programas.
  - Em programas, nos importamos com memória e velocidade, o que normalmente não é representado na matemática.
  - Em programas, precisamos detalhar **como** cada função é implementada, enquanto na matemática podemos somente definir funções pelas suas propriedades.
    - Inclusive, precisamos detalhar o que acontece em casos indefinidos pela matemática, como divisão por 0.

Referências

DE FRANÇA, F. O. **Tipo função**. Disponível em: <https://haskell.pesquisa.ufabc.edu.br/teoria-das-categorias/09-tipofuncao/>.

KONNOV, I. **You should not care about memory in protocol specifications**. Disponível em: <https://konnov.github.io/protocols-made-fun/quint/2024/01/14/maps.html>.

WADLER, P. Propositions as types. **Commun. acm**, v. 58, n. 12, p. 75–84, Nov. 2015.

Programação e matemática não são a mesma coisa

Aula para disciplina de Métodos Formais

Gabriela Moreira

Departamento de Ciência da Computação - DCC
Universidade do Estado de Santa Catarina - UDESC

28 de fevereiro de 2024