Introdução

Programação Funcional

Prof. Maycon Amaro

Objetivos da Disciplina

- Compreender características de linguagens funcionais modernas
- Noções básicas de sistemas de tipos, avaliação, estados e correção
- Habilidade para programar em uma linguagem funcional

Método de Aula

- Aulas teóricas expositivas: segundas e quartas, de 08:20 às 10:00, no Bloco de Salas.
- Listas de exercícios

Avaliações

Prova 1

- Aula de dúvidas: 19 de dezembro de 2022.
- Data da prova: **21 de dezembro** de 2022. Notas no próprio dia 21.

Provas 2 e 3

Em 2023.

Nota Final

Média aritmética das provas.

Contato, Atendimento e Bibliografia

Disponíveis no Moodle.
Verifiquem o e-mail institucional com frequência.

O que é Programação Funcional?

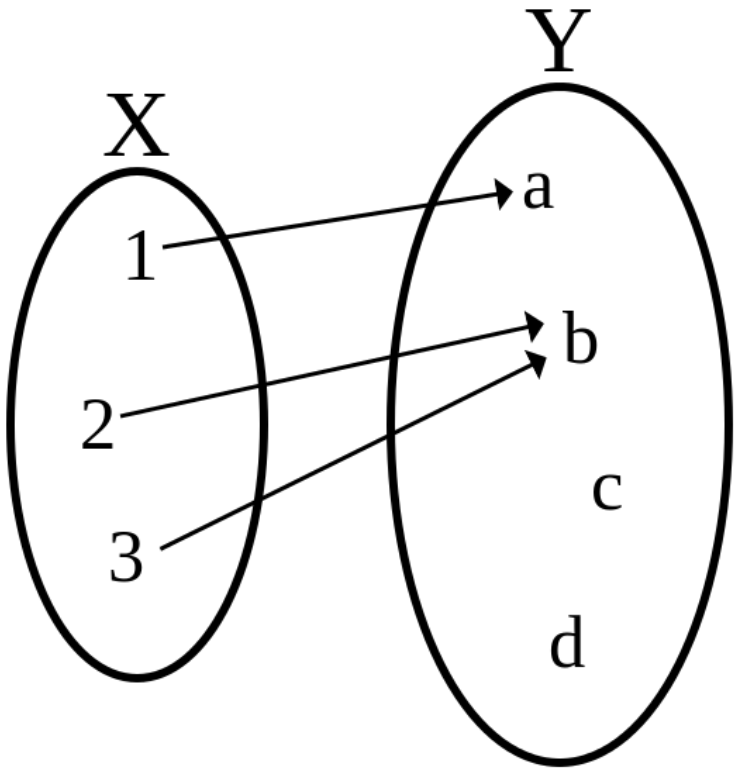
Programação Funcional

É um paradigma de programação que se baseia em funções modeladas por *funções matemáticas*. Programas são combinações de *expressões*.

Revisão: Função Matemática

Definição informal

Uma função é uma relação entre um conjunto de possíveis entradas e um conjunto de possíveis saídas. No exemplo abaixo, chamamos X de *domínio* e Y de *contra-domínio*.



Restrição

Para ser uma função, cada elemento do domínio só pode estar relacionado a no máximo um elemento do contra-domínio. O exemplo abaixo não é uma função válida.

$$f(1) = 2$$
$$f(2) = 7$$
$$f(1) = 3$$

Funções parciais e totais

- Uma função é *total* se está definida para todo elemento do domínio.
  - A função $f : \mathbb{Z} \to \mathbb{Z}$ tal que $f(x) = x + 1$ é total.
- Uma função é *parcial* se existe algum elemento do domínio para o qual ela não está definida.
  - A função $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ tal que $f(x, y) = x - y$ é parcial, pois só está definida para pares (x, y) em que $x \geq y$.

Funções em Ciência da Computação

Tipos como Conjuntos

- Todo tipo pode ser entendido como um conjunto, cujos elementos são os valores válidos desse tipo.
- Assim, uma função é um mapeamento entre elementos de um tipo A a um tipo B.

```
int f(int x) {
  return x + 1;
}
```

$\ldots, f(-1) = 0, f(0) = 1, f(1) = 2, \ldots.$

Comandos e Expressões

Em muitas linguagens, funções podem executar comandos além de responderem com uma expressão. Isso já é uma grande diferença entre funções matemáticas e computacionais.

```
int f(int x){
  printf("Olá, mundo!");
  return x + 1;
}
```

Veremos mais tarde que, no paradigma funcional, algumas restrições são impostas com relação à isso.

Respeito à restrição

Devido às regras de C, a função abaixo relaciona 1 somente com 2. Não viola a restrição.

```
int f(int x) {
  if (x == 1)
    return 2;

  if (x == 1)
    return 3;

  return x + 4;
}
```

Respeito à restrição

O elemento a qual 1 está relacionado agora depende da resposta do algoritmo de números aleatórios utilizado. Se soubermos o algoritmo e a semente, é possível determinar.

```
int f(int x) {
  if (x == 1) {
    int r = rand() * 100;
    if (r % 2 == 0)
        return 2;
    else
        return 3;
  }
  return x + 4;
}
```

Respeito à restrição: um exemplo em Python

A função abaixo mapeia 1 para a tupla $(2, 3)$. Na matemática, isso seria o correspondente à uma função $f : R \to R \times R$, ou ainda, $f : R \to R^2$. Usando tipos, podemos dizer que a função abaixo é `Double -> (Double, Double)`. Não é uma violação à definição matemática.

```
def f(x):
  if x == 1:
    return 2, 3
  else:
    return x, x + 1
```

Respeito à restrição

- É um pouco mais complicado garantir que a restrição de que cada elemento do domínio está mapeado a no máximo um elemento do contra-domínio é respeitada.
- Isso tem a ver com *efeitos colaterais*, que serão limitados pelo paradigma funcional.

Função parcial

A função em C abaixo não está definida para todos os valores do tipo `int`.

```
int f(int x) {
  if (x < 0) {
    return x * (-1);
  }
}
```

Função total

A função em C abaixo está definida para todos os valores do tipo `int`.

```
int f(int x) {
  return x / 2;
}
```

E agora?

Essa função é parcial ou total?

```
int f(int x) {
  if (x > 0)
    return x;
  if (x < 0)
    return x * (-1);
  if (x == 0)
    return f(0);
}
```

Terminação importa para a totalidade

A função entra em *loop* infinito se o argumento é 0. Ela não é total, 0 não está relacionado a um elemento de `int`.

```
int f(int x) {
  if (x > 0)
    return x;
  if (x < 0)
    return x * (-1);
  if (x == 0)
    return f(0);
}
```

E agora?

Essa função é total ou parcial?

```
int f(int x) {
  return x + 1;
}
```

Limitações práticas

- A função está *teoricamente* definida para todos os valores do tipo `int`, mas o tipo possui um limite superior em 2.147.483.647.
- Para esse valor, o resultado causa um *integer overflow*.
- Em muitas linguagens, como Python 3, esse limite já foi retirado.

```
Python 3.10.8 (main, Oct 12 2022, 00:00:00)
>>> 2147483647 * 2
4294967294
>>>
```

Transparência Referencial

Transparência Referencial

Observe este código. O que ele imprime?

```c
#include <stdio.h>

int z = 2;

int f(int x) {
  z++;
  return x + 1;
}

int main() {
  printf("%d\n", f(1));
  printf("%d\n", z);
}
```

Transparência Referencial

Se trocarmos a chamada da função pelo valor que ela retorna, o comportamento do programa se mantém igual?

```c
#include <stdio.h>

int z = 2;

int f(int x) {
  z++;
  return x + 1;
}

int main() {
  printf("%d\n", 2);
  printf("%d\n", z);
}
```

Transparência Referencial

- A resposta é não. A função `f` provoca um *efeito colateral* no programa antes de calcular e retornar uma expressão.
- Funções ou partes de programas que podem ter suas chamadas substituídas pelo resultado sem afetar o comportamento do programa possuem a propriedade da **transparência referencial**.

Requisitos para a Transparência Referencial

Para que uma função seja referencialmente transparente, ela:

- deve sempre retornar o mesmo resultado para uma mesma entrada
- deve não possuir efeitos colaterais

Funções referencialmente transparentes também são chamadas de **funções puras**.

Programação Funcional Pura

- Linguagens funcionais puras permitem apenas* a definição de funções puras. Um programa com uma função não pura será rejeitado.
- Ao decorrer da disciplina, trabalharemos principalmente com uma linguagem funcional pura: Haskell.

* Há algumas exceções.

Por quê estudar Programação Funcional?

1. Escalabilidade

- Projetos de software crescem, partes de código que afetam outras dão dor de cabeça e dificultam a manutenção.
- Programação funcional controla rigorosamente a mudança de estado, sendo naturalmente escalável.

2. Paralelismo

- Muitos problemas em programação paralela e distribuída se originam do fato de que o estado do programa é modificado durante a execução.
- Na programação funcional, o estado do programa está sob rígido controle, sendo naturalmente paralelizável.

3. Corretude

- Testar código é fundamental, mas às vezes precisamos de mais do que isso.
- A programação funcional é a mais simples de se provar propriedades sobre programas.

3. Já está na moda!

- Diversas linguagens de programação já estão incorporando conceitos de programação funcional. Entendê-los está cada vez mais se tornando essencial.
- Alguns exemplos: C++, Swift, Rust, Python 3, JavaScript, Java, Ruby.

Quem usa Programação Funcional?

Nubank

> *"Na época em que o Nubank foi fundado, Clojure pareceu a melhor opção para os problemas que precisávamos resolver. Hoje, com mais de 15 milhões de clientes, todas as áreas do Nubank usam Clojure e mais de 90% dos microsserviços são escritos nessa linguagem."*

— Bruno Rodrigues, *Tech Manager* do Nubank.

Leia mais em https://blog.nubank.com.br/o-que-e-clojure/.

Microsoft

F# é uma linguagem de programação funcional criada pela Microsoft e utilizada por ela e por várias organizações ao redor do mundo.

O site https://fsharp.org/testimonials/ apresenta dezenas de testemunhos.

Github

Github é a criadora da ferramenta Semantic, que faz parsing, análise e comparação de código dentre diversas linguagens. Semantic é escrita em Haskell e há até uma explicação oficial do porquê.

Leia mais em https: //github.com/github/semantic/blob/main/docs/why-haskell.md.

Meta

A empresa é uma das maiores patrocinadoras da *Fundação Haskell*. Diversos sistemas utilizados por ela foram escritos em Haskell. Exemplos: Sigma, Glean e Haxl.

No lançamento de Haxl, a Meta disse que:

> *Haskell é uma linguagem de programação pura que é muito apreciada pela comunidade de programação por seu sistema de tipos expressivo, rico ecossistema de bibliotecas, e implementações de alta qualidade. Essa combinação de propriedades nos permite o rápido desenvolvimento de software robusto com fortes garantias de correção e segurança.*

Leia mais em https://engineering.fb.com/2014/06/10/web/open-sourcing-haxl-a-library-for-haskell/.

Dúvidas?