Sergio Polimante Souto

Principais Conceitos de Programação Funcional

Os paradigmas de programação mais comum são imperativos, procedurais e orientados a objetos [1]. Apesar de muito eficientes e poderosos, escrever um algoritmo capaz de paralelizar o processamento, ou seja, dividir o código em diferentes linhas de processamento, pode ser muito difícil utilizando esses paradigmas. Esses paradigmas não são eficientes em garantir estado de variáveis e funções pois estes podem se alterar a qualquer instante. Dessa maneira, surge a necessidade de um paradigma de programação que facilite a implementação de códigos para serem processados paralelamente. Para isso foi utilizado o paradigma declarativo funcional [2]. Haskell é atualmente a linguagem de programação que possui propriedades que melhor atende às demandas de um paradigma funcional que facilite implementação de processamento paralelo [3]. Os parágrafos abaixo descreveram um pouco sobre cada uma dessas propriedades. [6]

Funções Puras: Funções puras são funções que seu valor de saída depende única e exclusivamente de seus valores de entrada. Um exemplo de função pura é a **math.sin(x).** A função sempre retornará o mesmo valor para um dado valor **x.** Quando uma função realiza outras funções além de calcular o valor de sua saída, ela é uma função impura. Uma função pura pode sempre ter sua chamada substituída pelo resultado desta chamada. Por exemplo: a função **math.sin(pi)** sempre retornará **zero**, portanto, ela pode ser substituída pelo valor **zero.** [4]

Funções de Primeira-Classe: Funções de primeira-classe são funções que podem ser tratadas como qualquer outra variável pela linguagem de programação. Por exemplo, funções de primeira-classe podem ser passadas como argumento para outras funções, retornadas como argumentos por funções, e podem ser atribuídas a uma variável. [5]

Variáveis são imutáveis: Variáveis não podem ter seus valores alterados em tempo de execução. Uma vez que possuem um valor atribuído a elas, esse valor será inalterado. Novas variáveis podem ser criadas, uma vez criadas e atribuídas de um valor, são inalteradas. [6]

Transparência Referencial: Quando uma função sempre retorna um mesmo valor, de maneira que em linha de código pode ser substituída pelo valor que retorna sem causar nenhum problema, então essa função possui Transparência Referencial. [6]

Baseada em Cálculo Lambda: Cálculo lambda é uma representação matemática de funções. Qualquer representação matemática pode ser expresso em Função Lambida e também qualquer função representada por uma Máquina de Turing pode ser expressa em cálculo lambda e vice versa. Em linguagens de programação como Assembly (e consequentemente C), o cálculo é feito similarmente a uma máquina de Turing, com uma instrução e um estado produz-se outro estado. Já no paradigma de programação funcional, não é assim que operações funcionam, esse não é o paradigma da linguagem. A maneira de processar operações na função lambda é similar ao fundamento do paradigma funcional de programação. Ou seja, saber cálculo lambda é fundamental para programar no paradigma funcional. [7]

Referências:

[1] **Which are the most used programming paradigms? Which are good coding examples?.** Disponível em: <https://www.quora.com/Which-are-the-most-used-programming-paradigms-Which-are-good-coding-examples

[2] **Programming paradigma.** Disponível em: <https://en.wikipedia.org/wiki/Programming_paradigm>

[3] **Concurrency (computer science)**. Disponível em: <https://en.wikipedia.org/wiki/Concurrency_(computer_science)>

[4] **Functional Programming: Pure Functions.** Disponível em: <https://www.sitepoint.com/functional-programming-pure-functions/>

[5] **First-class Function.** Disponível em: <https://developer.mozilla.org/en-US/docs/Glossary/First-class_Function>

[6] **Core Functional Programming Concepts.** Disponível em: <https://thesocietea.org/2016/12/core-functional-programming-concepts/>

[7] **Lambda Calculus.** Disponível em: <https://brilliant.org/wiki/lambda-calculus/>