Linguagens de programação funcional

Introdução

O paradigma de programação funcional, baseado em funções matemáticas, é a base de projeto dos estilos de linguagem não imperativos mais importantes. Esse estilo de programação é suportado por linguagens de programação funcional.
O desenvolvimento de linguagens de programação funcional tipadas, principalmente ML, Haskell, OCaml e F#, levou a uma expansão significativa das áreas da computação nas quais agora são usadas linguagens funcionais. À medida que essas linguagens são atualizadas, seu uso prático aumenta. Atualmente, são utilizadas em áreas como processamento de bancos de dados, modelagem financeira, análise estatística e bioinformática.

Funções matemáticas

Uma função matemática é um mapeamento de membros de um conjunto, chamado de conjunto domínio, para outro, chamado de conjunto imagem. As funções são geralmente aplicadas a um elemento específico do conjunto domínio, fornecido como parâmetro para a função.

Funções simples

Definições de funções são geralmente escritas como um nome de função, seguido de uma lista de parâmetros entre parênteses, seguidos pela expressão de mapeamento. Por exemplo,

$$cube(x) \equiv x * x * x$$, onde x é um número real.

Aplicações de funções são especificadas por um par que contém o nome da função com um elemento particular do conjunto domínio.
O elemento da imagem é obtido ao avaliarmos a expressão de mapeamento da função com o elemento do domínio substituído para as ocorrências do parâmetro.
Cada ocorrência de um parâmetro é vinculada a um valor do conjunto domínio e é uma constante durante a avaliação. Por exemplo, considere a seguinte avaliação de cube(x):

$$cube\,(2) = 2 * 2 * 2 = 8$$

Formas funcionais

Uma forma funcional, é aquela que recebe uma ou mais funções como parâmetros, ou que leva a uma função como resultado, ou ambos. Um tipo de forma funcional é a composição funcional, com dois parâmetros funcionais e leva a uma função cujo valor é o primeiro

parâmetro de função real aplicado ao resultado do segundo, escrita como uma expressão, usando ° como um operador, como em:

$$h \equiv f \circ g$$

se:

$$f(x) \equiv x + 2$$
$$g(x) \equiv 3 * x$$

então h é definida como:

$$h(x) \equiv f(g(x)) \text{ ou } h(x) = (3 * x) + 2$$

Existem outras formas funcionais, mas esses exemplos ilustram as características básicas de todas elas.

Fundamentos das linguagens de programação funcional

O objetivo do projeto de uma linguagem de programação funcional é imitar as funções matemáticas ao máximo possível.
Uma linguagem funcional fornece um conjunto de funções primitivas, um conjunto de formas funcionais para construir funções complexas a partir dessas funções primitivas, uma operação de aplicação de função e alguma estrutura ou estruturas para representar dados. Essas estruturas são usadas para representar os parâmetros e os valores computados pelas funções. Se uma linguagem funcional é bem projetada, ela requer apenas um número relativamente pequeno de funções primitivas.

A primeira linguagem de programação funcional: Lisp

Desenvolvida por John McCarthy no MIT, em 1959, Lisp é a linguagem mais antiga e mais utilizada (ou uma de suas descendentes).
Lisp foi a primeira linguagem funcional, mas, apesar de ter evoluído continuamente por meio século, não representa os mais recentes conceitos de projeto para linguagens funcionais. Além disso, com exceção da primeira versão, todos os dialetos de Lisp incluem recursos imperativos, como variáveis no estilo imperativo, sentenças de atribuição e iteração. (Variáveis no estilo imperativo são usadas para nomear células de memória, cujos valores podem ser modificados muitas vezes durante a execução de um programa.) Apesar disso e de suas formas um pouco não convencionais, as descendentes de Lisp original representam bem os conceitos fundamentais da programação funcional.

Haskell

Haskell (Thompson, 1999), é uma linguagem de programação puramente funcional baseada em um estilo de programação em que se enfatiza mais o que deve ser feito em detrimento de como deve ser feito. É uma linguagem que possui foco no alcance de soluções

para problemas matemáticos, clareza, e de fácil manutenção nos códigos, e possui uma variedade de aplicações e apesar de simples é muito poderosa.

F#

F# é uma linguagem de programação funcional .NET cujo núcleo é baseado em OCaml, que é uma descendente de ML e Haskell. Embora seja fundamentalmente uma linguagem funcional, ela contém recursos imperativos e suporta programação orientada a objetos. Uma das características mais importantes de F# é o fato de ter um IDE completo, uma ampla biblioteca de utilitários que suportam programação imperativa, orientada a objetos e funcional, além de interoperabilidade com uma coleção de linguagens não funcionais (todas as linguagens .NET).

Uma comparação entre linguagens funcionais e imperativas

Linguagens funcionais podem ter uma estrutura sintática muito simples. A estrutura de lista de Lisp, usada tanto para código quanto para dados, ilustra claramente isso. A sintaxe das linguagens imperativas é muito mais complexa. Isso dificulta sua aprendizagem e uso.

A semântica das linguagens funcionais também é mais simples que a das imperativas
.
Algumas pessoas que trabalham com programação funcional afirmam que seu uso resulta em um aumento de uma ordem de magnitude na produtividade, pois os programas funcionais têm apenas 10% do tamanho de seus correspondentes imperativos.

linguagens funcionais têm uma possível vantagem na legibilidade. Em muitos programas imperativos, os detalhes de lidar com variáveis obscurecem a lógica do programa. Considere uma função que computa a soma dos cubos dos primeiros n positivos inteiros. Em C, essa função provavelmente seria similar a:

```
int sum_cubes(int n){
        int sum = 0;
        for(int index = 1; index <= n; index++)
                sum += index * index * index;
        return sum;
}
```

Em Haskell, a função poderia ser:

```
sumCubes n = sum (map (^3) [1..n])
```