CURSO: Ciência da Computação PERÍODO: 4o. DISCIPLINA: Técnicas Alternativas de Programação	CÓDIGO: CI062 PROFESSOR: Andrey AULA: 16

1 Apresentação

Na aula de hoje vamos apresentar e discutir os conceitos de Programação Funcionalista

2 Desenvolvimento

2.1 Linguagens de Programação Funcionalistas

As linguagens de programação que seguem o paradigma funcional se espelham no conceito de função matemática para compor seus programas. É um dos estilos de linguagem de programação não imperativa mais importantes. Dentre as principais linguagens de programação deste estilo temos o LISP, Scheme LISP, o ML e o Haskell. [Sebesta, 2009]

2.2 Função Matemática

Uma função matemática é uma correspondência entre membros do conjunto domínio com outro do conjunto imagem.

$Domínio -> correspondência -> Imagem$

A ordem de avaliação de suas expressões é controlada por recursão e por expressões condicionais e não pela sequência de iterações comuns em linguagens imperativas. As funções matemáticas definem valores, enquanto que uma função de linguagem de programação produz valores. Uma função matemática não tem efeitos colaterais, por isso define sempre o mesmo valor. Elas não tem variáveis no sentido das linguagens imperativas e por esse motivo não podem haver efeitos colaterais.

As funções matemáticas são normalmente definidas pelo nome da função, uma lista de parâmetros e pela expressão de correspondência entre os conjuntos domínio e imagem.

$cubo(x) \equiv x * x * x$, em que x é um número real

As Expressões Lambda, definidas por Church [Church, 1941] separam a tarefa de definir a função da tarefa de nomeá-las. Elas especificam os parâmetros da função e o seu mapeamento. Uma Expressão Lambda ou função sem nome pode ser:

$\lambda(x)x * x * x$

E sua avaliação passando-se um valor para x:

$(\lambda(x)x * x * x)(2) -> 8$

2.3 Funções de Ordem Superior

A composição de funções é uma forma funcional onde o valor de uma função é definido por dois parâmetros funcionais e leva a uma função cujo valor é o primeiro parâmetro de função real aplicado ao resultado do segundo. Por exemplo:

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

Outras Formas Funcionais: Construção
$g(x) \equiv x * x$
$h(x) \equiv 2 * x$
$i(x) \equiv x/2$
$[g, h, i](4) -> (16, 8, 2)$

Aplicar a todos α
$h(x) \equiv x * x$
$\alpha(h, (2, 3, 4)) -> (4, 916)$

2.4 Fundamentos de Linguagens Funcionais

Dentre as principais características das Linguagens de programação funcionais temos:

- Linguagens de Programação Funcionais não usam variáveis ou atribuições;
- As repetições são feitas por meio de recursões e não laços;
- Programas são definições de funções e da especificação da aplicação destas.

A execução de uma função com os mesmos parâmetros da sempre os mesmos resultados. A isto chamamos **Transparência Referencial**. A **recursão de cauda** é quando a chamada recursiva é a última expressão na função. Um compilador pode converter a recursão de cauda em um laço minimizando o custo das chamadas recursivas.

2.5 Principais Linguagens Funcionais

2.5.1 LISP

Lisp foi a primeira linguagem de programação funcionalista. Foi muito usada nas primeiras aplicações de IA dada a sua característica de facilitar a manipulação de listas. Em Lisp existem apenas dois tipos de objetos: átomos e listas.

Exemplo: Listas

```
(A B (C D) E)
```

A notação λ é usada para especificar funções. Definições de funções, aplicações e dados tem o mesmo formato. Por exemplo:

```
(A B C)
```

pode significar:

- Uma lista com A, B e C ou
- A aplicação da função A com parâmetros B e C

2.5.2 Scheme LISP

Tem como características:

- Escopo estático;
- Funções são entidades de primeira classe.

Exemplo:

```
(+ 5 2) produz 7
(LAMBDA (x) (SQRT(*X X X)))
((LAMBDA (x) (SQRT(*X X X))) 7)

(DEFINE (cube x) (*X X X))
(cube 4)
```

2.5.3 ML

Tem como características:

- Sintaxe semelhante ao PASCAL;
- Inferência de tipos;
- Fortemente tipada;
- Não puramente funcional.

Exemplo:

```
fun fac : (int -> int) 0 = 1
   | fac n = n * fac (n-1);

fun quadrado(x:real) = x*x;
fun reverso(L) =
         if L = nil then nil
         else reverso(tl(L)) @ [hd(L)];
```

2.5.4 Haskell

Tem como características:

- Puramente Funcional;
- Sem variáveis e atribuições;
- Fortemente tipada;
- Avaliação preguiçosa para melhorar desempenho.

Exemplo:

```
fatorial :: Integer -> Integer
fatorial 0 = 1
fatorial n | n > 0 = n * fatorial (n-1)
```

3 Atividades

4 Bibliografia

Referências

[Church, 1941] Church, A. (1941). *The calculi of lambda-conversion*. Princeton University Press.

[Sebesta, 2009] Sebesta, R. W. (2009). *Conceitos de linguagens de programação*. Bookman Editora.