Programação Funcional Haskell

Marcos Fagundes Caetano

Bacharel em Ciências da Computação - terceira fase, 2001
Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC), Brasil, 88040-050
Fone (0xx4) 333-9999, Fax (0xx48) 333-9999
caetano@inf.ufsc.br

Resumo

Este artigo apresenta uma explanação sobre a linguagem funcional *Haskell,* abordando suas principais características, para que usuários da linguagem *LISP* possam realizar uma analogia entre ambas. Iniciamos com uma abordagem sobre o ambiente disponibilizado ao usuário, passando pelos tipos de dados implementados, expressões, definição de função, e por fim, um exemplo de implementação em *Haskell*.

Palavras-chave: *Haskell*, *LISP*, linguagem funcional.

Abstract

This article presents an explanation about *Haskell* functional language, approaching its main characteristics, so that users of *LISP* language may make an analogy between them. We start with an approach about the environment which is available to the user, going through the kinds of implemented data, expressions, function definition, and, finally, an example of *Haskell* implementation.

Key-words: *Haskell, LISP*, functional language.

Introdução

A linguagem *Haskell* [1,2,3] é uma linguagem funcional com tipos de dados bem definidos [1], ou seja, os modelos para as estruturas de informação são baseados na Teoria dos Conjuntos, e as operações são descritas através de funções entre conjuntos.

Uma particularidade da linguagem *Haskell*, é a presença de um compilador e um interpretador (*Hugs*[4]). Ambos utilizam-se de um editor externo, sendo que o primeiro gera um executável para a plataforma utilizada, e o segundo realiza o trabalho de interpretação do código inserido.

No decorrer do artigo serão abordadas as características do interpretador, tais como: tipos de dados, expressões, definição de funções e implementação.

Características do Interpretador

É de responsabilidade do interpretador a avaliação das expressões informadas pelo usuário, como: expressões devem ser declaradas em um linha;

Tipos de Dados

Uma forte característica da linguagem *Haskell* é sua "tipagem" (*strongly typed*), isto é, é sempre possível determinar o tipo de uma determinada entidade. Em contra-ponto podemos fazer uma analogia com liguagens sem tipos, ou melhor, linguagens onde todas entidades partilham de um mesmo tipo, como é o caso de *LISP*.

Podemos citar os seguintes tipos base de *Haskell*:

- tipo unitário: é uma implementação do conjunto um, sendo assim, todos os demais tipos derivam desse.
- tipo lógico (bool): valores lógicos, podendo assumir como valores: t*rue*, *false*, *otherwise* (= *true*). Podem ser submetidos aos seguintes operadores: && (conjunção); || (disjunção); not (negação).

- tipo caracter (char): valores *Unicode* do nosso alfabeto, podendo assumir como valores símbolos da tabela *Unicode* (16bits).
- tipo numérico: valores numéricos, inteiros (*int*, *integer*), e reais (*float*, *double*).
- listas: conjunto das sequências finitas, eventualmente vazias, de elementos de um dado tipo. Dentre todos operadores podemos citar: *last* – retorna último elemento de uma lista; *init* – retorna o primeiro elemento de uma lista; ++ - realiza a concatenação entre duas listas.
- sequência de caracteres (*Strings*): podem ser considerados como uma lista de caracteres.

Expressões

Utilizamos o comando *:type* para setar o interpretador a mostrar informações sobre o tipo das expressões manipuladas no decorrer da seção.

A avaliação de uma expressão por parte do interpretador é realizada normalmente (esquerda para direita). Podendo o mesmo ser setado para que informe dados estatísticos a respeito das alterações realizadas (*:set +s*).

Definição de Funções

A definição de funções deve ser realizada em um novo arquivo, o qual deverá possuir a extensão *.hs* através do comando *:load <file>* as funções declaradas são disponibilizadas.

A definição de uma função é realizada através da escrita de uma equação, a qual irá determinar o comportamento da mesma. Deve-se obedecer a seguinte sintaxe: <nome_da_funcao> <argumento> = <expressão>

A linguagem *Haskell* não suporta mais de um argumento. Esse problema pode ser resolvido agrupando seus argumentos em um *n-duplo*:

- <funcao> :: (Int, Int) -> Int
  <funcao> <arg1, arg2> = <command>

Definição de Condicionais

Utilizamos de condicionais para determinar qual o fluxo de execução atende melhor nossas necessidades. Deve-se levar em consideração a sintaxe:

- *if <conditional> then <command1> else <command2>*
- | *<conditional1> = <command1>*
  | *<conditional2> = <command2>*

Definição de Recursividade

Para definição de uma função recursiva podemos utilizar como base o exemplo:

- <function> :: type -> type
  <function> <arg>
  | <condition1> = 0
  | <condition2> = <arg> * <function(arg-1)>

Exemplo

Um exemplo clássico para demostrar a sintaxe da linguagem *Haskell* é a implementação das funções *fatorial* e *Fibonnacci*

- função fatorial:

```
fact1 :: Int -> Int
fact1 n
| n < 0 = error "A função fact não está
                 definida para n < 0"
| n == 0 = 1
| n > o = fact1( n-1 ) * n
```

- função *Fibonnacci:*

```
fib :: Int -> Int
fib n
| n == 0 = 1
| n == 1 = 1
| otherwise = fib(n-1) + fib(n-2
```

Conclusão

Com esse artigo podemos conhecer um pouco a linguagem funcional *Halkker* – suas principais caracteristicas – sendo a mesma possuidora de poderosas ferramentas as quais não foram abordadas neste. Existe uma grande variedade de linguagens funcionais, sendo a *Halkker* , em particular, mais uma a implementar o paradigma de Programação Funcional.

Referências

[1] Richard Bird. *Introduction to Functional Programming Using Haskell*. Prentice Hall, 1998.

[2] Kevin Hammond, John Peterson, Lennart Augustsson, Joseph Fasel, Andrew Gordon, Simon Peyton-Jones, and Alastair Reid. *Standard Libraries for the Haskell Programming Language*, version 1.4 editon, April 6 1997.

[3] Paul Hudak. *The Haskell School of Expression.*Cambridge University Press, 2000.
[4] Mark Jones and John Peterson. *Hugs 1.4*, April 1997.