Pequena Introdução a Haskell

Tiago Silva Pereira
tiagosp@inf.ufsc.br
Leonardo Maruyama de Carvalho
lmc@inf.ufsc.br
Ciência da Computação

February 11, 2003

Abstract

This is a very short introduction for Haskell programming. This document intend to give the readers a basic ideia on how to do a simple program using Haskell and how to recognize a program that was write using Haskell. But this is, as we said before, a very short introduction, but we will give some tips on how to find more info on this issue.

Key-words: Haskell, Functional, Lambda.

Abstract

Esta é uma breve introdução a linguagem de programação Haskell. O principal objetivo deste texto é dar ao leitor uma idéia básica de como fazer um simples programa utilizando Haskell e como reconhecer um programa escrito em Haskell. Mas como já foi dito anteriormente é apenas uma breve introdução mas iremos dar algumas dicas de como achar mais informações nesta área, para aqueles que se interecem.

Palavras-chave: Haskell, Funcional, Lambda.

Introdução

Haskell é mais uma das várias linguagens de programação funcional. E entre todas, Haskell, é a mais pura de todas,o que de certa forma acarreta muitos efeitos colaterais. Haskell é uma linguagem altamente tipada e a maioria dos operadores são infixos e não prefixados. Haskell tem sua base no Calculo-Lambda. O maior mantedor e atualizador desta linguagem é a universidade de Yale.

Definição inicial

Pela caracteírstica da linguagem a ser estudada(Haskell), fica claro que aqueles que detiverem um conhecimento prévio em Cálculo-Lambda terão maior facilidade no entendimento da linguagem, porém, por este ser apenas uma breve introdução ao Haskell aqueles que não detiverem o conhecimento prévio de Cálculo-Lambda poderá entender por completo esta introdução.

Uma grande característica do Haskell, assim como outras linguagens funcionais, é que uma vez a variável é criada não se

pode mais alterar o seu valor durante toda a execução do programa, diferente de outros tipos de linguagens tipo C. Em Haskell as variáveis são como variáveis matemáticas.

Em Haskell não existe nenhum tipo de construtores de loops tipo "for" e o "while". E esses tipos de estruturas não existem pois como já foi dito antes não se pode mudar os valores das variáveis e isso impossibilita estes tipos de estruturas. Para se fazer algo muito parecido com um loop se utiliza funções recursivas, por exemplo:

```
   fac1 :: Int --> Int
fac1 n = if n==1 then 1
else (n* fac1(n-1))
```

Não existe a necessidade em se haver qualquer tipo de ordem nas definições das funções de um programa em Haskell e o código que segue é executado normalmente:

```
   j = i + 1
i = 5
```

Sintaxe

Um programa em Haskell é composto por uma série de definições de funções. A essas funções são associados nomes, como se fossem definições de variáveis(em outras linguagens imperativas), e esses nomes são imutáveis dentro do escopo do programa, por exemplo:

```
myNum :: Int
myNum = 12 + 13
square :: Int --> Int
square n = n*n
double :: Int --> Int
double n = 2*n
douSqr :: Int --> Int
douSqr n = square (double n)
```

As definições dessas funções são divididas em duas partes. A primeira parte é a definição dos tipos utilizados na função. São definidos os tipos de entrada e saida como exemplo:

```
exemplo :: Int --> Int
```

Isso quer dizer que será passado um inteiro para a função que retornará um outro valor inteiro. Na segunda parte é definida a função propriamente dita, por exemplo:

```
exemplo n =n*n
```

Se não for declarado os tipos da função o compilador Haskell irá analisar a função em tempo de compilação e criará essa definição automaticamente.

Listas

Também é possível utilizar listas em Haskell, como por exemplo:

```
   prodLista [] = 0
prodLista [x] = x
prodLista [x:xs] =
x * prodLista xs
```

Uma grande vantagem de Haskell é a possibilidade em se trabalhar com listas verdadeiramente infinitas. Já que só será computado a parte da lista que realmente importa. Um bom exemplo disso é o algoritimo para o cálculo de números primos como exemplo:

```
   primes :: [Int]
primes = sieve [2 ...]
sieve (x:xs) = x :
sieve [y|y<-xs,(y rem x)/=0]
memberOrd :: Ord a => [a]
-> a -> Bool
memberOrd (x:xs) n
|x < n  =memberOrd xs n
|x == n =True
```

```
|otherwise =False
ePrimo n = memberOrd primes n
```

Guards

Uma outra estrutura utilizada em Haskell, que se parece muito com o if...then...else das linguagens imperativas, é o "guards", essa estrutura possui a seguinte sintaxe:

```
   prodLista2 lst
|length lst==0 =0
|length lst==1 =head lst
|otherwise     =head lst
* prodLista2 tail lst
```

Considerações Finais

Em Haskell um comentário de uma linha é sempre precedido por dois traços "–" e comentários de multiplas linhas utilizam "-.....-". Os arquivos fontes em Haskell possuem terminação .hs, quando utilizando Haskell comum. Porém quando utilizando Script Haskell .lhs e todas as linhas do código são comentários a não ser as linhas começadas com o sinal de maior " ", *porexemplo* :

```
Programa teste
    > teste :: Int --> Int
    > teste n = n/2
Isso retorna a metade do valor
passado
```

Uma importante caractéristica de Haskell e que causa bastente erros e confusões é que funções têm "vantagem" sobre outros operadores, ou seja as funções são avaliadas primeiramente, por exemplo:

```
   double n = n*2
res1 = double 5\^ 2
--res1 = 100 pois (5*2)\^ 2
res2 = double (5\^ 2)
--res2 = 50 pois (5\^ 2)*2
res3 = double double 5
--res3 = um erro
```

Conclusão

Haskell é uma linguagem bastante intuitiva e é a mais pura linguagem funcional. Existem vários compiladores de Haskell para os diversos Sistemas Operacionais e um dos principais interpretadores dessa linguagem é o Hugs, que pode ser encontrado no site Haskell.org. Espero ter sido claro nesta introdução e ter criado um interesse em todos por essa linguagem muito interassante.

Nota Bibliográfica

1. http://haskell.org
2. http://scgwiki.iam.unibe.ch:8080/SCG/uploads/330/haskellTutorial.pdf
3. http://lmf.di.uminho.pt/Temas/Haskell/haskell98-report/index.html