Implementação de uma Linguagem Funcional

Rodrigo Bonifácio

Maio de 2013

Esquecendo um pouco a manipulação de formas geométricas e imagens, vamos trabalhar agora com a implementação de um interpretador para uma linguagem funcional.

Versão zero da nossa linguagem

Suporte a valores do tipo inteiro e duas operações sobre valores inteiros $+, -$.

Representação da AST

```
module Language where

data Exp = IConst Int
         | Add Exp Exp
         | Sub Exp Exp
```

Interpretador

```
module Interpreter where

import Language

eval :: Exp → Int
eval (IConst v) = v
eval (Add lhs rhs) = eval lhs + eval rhs
eval (Sub lhs rhs) = eval lhs - eval rhs
```

. . . e alguns casos de testes

```
module Tests where

import Language
import Interpreter

import Test.HUnit

add00  = Add (IConst 0) (IConst 0)
add34  = Add (IConst 3) (IConst 4)
add3_4 = Add (IConst 3) (IConst (-4))
add4_4 = Add (IConst 4) (IConst (-4))

test0 = TestCase (assertEqual "Add 0 0" 0 (eval add00))
test1 = TestCase (assertEqual "Add 3 4" 7 (eval add34))
test2 = TestCase (assertEqual "Add 3 (-4)" (-1) (eval add3_4))
test3 = TestCase (assertEqual "Add 4 (-4)" (0) (eval add4_4))

allTests = TestList [TestLabel "test0" test0
                    ,TestLabel "test1" test1
                    ,TestLabel "test2" test2
                    ,TestLabel "test3" test3
                    ]
```

Versão um da nossa linguagem

Suporte a valores dos tipos inteiro e booleano, bem como operações lógicas ($\vee, \wedge, \neg$) e aritméticas ($+, -, \times, \div$).

- Noção de tipo torna-se necessária

- O que fazer quando tentamos somar um inteiro com um booleano?

Versão dois da nossa linguagem

Incluir o suporte a expressões do tipo `let`. Com isso, temos que trabalhar a noção de ambiente de execução, em que identificadores são associados a expressões e também podem ser avaliados. Com essa versão da linguagem, conceitos como escopo e ligação precisam ser discutidos.

- identificador não definido
- identificador com múltiplas definições
- estratégias de substituição

Versão três da nossa linguagem

Incluir o suporte a declaração e aplicação de funções. Uma aplicação de função é uma expressão que, quando avaliada, também retorna um valor.

- como representar declarações de funções?
- como checar o tipo de uma aplicação de função?
- como avaliar uma aplicação de função?

Declaração de Função / Expressão

```
type FormalArgs = [Id]

data FuncDecl = FuncDecl Id FormalArgs Exp

data Exp = IConst Int
         | BConst Bool
         | And Exp Exp
         | Or Exp Exp
         | Not Exp
         | Add Exp Exp
         | ◦ ..
         | Let Id Exp Exp
         | RefId Id
         | App Id Args
 deriving(Show)
```

Funções `baseType` e `eval`

```
baseType :: Exp → Env → [FuncDecl] → Type
...

eval :: Exp → Env → [FuncDecl] → Value
...
```