Projeto Haskell
Especificação da Primeira Parte

Rodrigo Bonifácio

Setembro de 2013

Evolução do Interpretador

A terceira versão da *mini linguagem funcional* implementada durante as aulas suporta operações sobre valores inteiros e booleanos, referências a expressões nomeadas, expressões `Let` e aplicação de funções. Com isso, a sintaxe abstrata da linguagem é definida como:

```
type Id = String

type Args = [Exp]

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

Como a linguagem suporta a aplicação de funções, precisamos (a) ter uma representação para declarar funções e (b) passar uma lista de declarações de funções para as funções que verificam os tipos (`baseType`) e avaliam (`eval`) expressões.

```
data Type = IntType
          | BooleanType
          | Undefined
 deriving (Show, Eq)
```

```
data Value = IntValue Int
           | BooleanValue Bool
   deriving(Show, Eq)

type FormalArgs = [Id]
type Binding = (Id, Exp)

type Env = [Binding]

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

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

Atividade: Implemente as seguintes modificações na linguagem

- Suporte a tipos nos argumentos formais. Atualmente, os argumentos formais são basicamente identificadores. Acrescente a noção de tipo aos argumentos formais, e evolua a checagem de tipos para verificar se algum erro de tipos ocorre na aplicação de funções, dada uma inconsistência de tipos entre os argumentos atuais e os argumentos formais. (25% da entrega).

- Considere a função definida, na nossa linguagem, como:

  ```
  f :: FuncDecl
  f  = FuncDecl "f" ["p"] (RefId "n")
  ```

  Note que a função `f` possui um identificador `n` livre. Considere a seguinte expressão:

  ```
  -- exp = let x = 5 in f 3
  exp :: Exp
  exp = Let "n" (IConst 5) (App "f" [(IConst 3)])
  ```

  A implementação atual utiliza a semântica de **escopo dinâmico**; com isso, a avaliação da expressão `exp` reduz para o valor `IntValue 5`. Na estratégia de **escopo estático**, o escopo de um identificador corresponde a uma região sintaticamente delimitada (e fora do contexto de execução); e um erro *idenitificador não declarado* deveria ser reportado na avaliação da expressão `exp`. Altere a implementação atual para suportar a semântica estática. (25% da entrega).

- Implemente uma expressão `If-Then-Else`, conforme discutido em sala, e verifique se a implementação atual suporta chamadas recursivas de funções (escreva casos de testes para isso). Implemente o suporte a funções recursivas, caso ainda não seja suportado pela implementação atual. (25% da entrega).

- Ler o Capítulo 7 do livro "Programming Languages: Application and Interpretation" (Shriram Krishnamurthi) e evolua a linguagem para suportar a noção de *funções como valor*. (25% da entrega).