Implementação de Verificação de Tipos em Haskell

Hélio Santana da Silva Júnior - 140142959
Jônatas Ribeiro Senna Pires - 140090983

6 de julho de 2017

1 Introdução

Esse documento apresenta uma implementação, em *literate Haskell*, do mecanismo de verificação de tipos de uma linguagem de programação funcional minimalista. Esta versão implementa a verificação de tipos para todos os elementos sintaticos, como aplicação de função, expressões lambda, let, if e expressões binarias.

2 Visão geral da linguagem

A linguagem LFCF suporta tanto expressões identificadas (LET) quanto identificadores e funções de alta ordem (com o mecanismo de expressoes lambda). O foco é na verificação de tipos, então não estão implementadas funções voltadas para a avaliação das expressões.

3 Definição da Árvore Sintática Abstrata

A implementação consiste na definição de um módulo Haskell mais alguns tipos auxiliares, como `Id` (um tipo sinônimo para uma `string`) e `Gamma`, que corresponde a um mapeamento de identificadores em tipos.

module *LFCFDTypes* **where**

```
type Id = String
type Gamma = [(Id, Tipo)]
```

Os tipos válidos são definidos com o tipo algébrico `Tipo`, que pode ser um tipo inteiro, um tipo booleano e um tipo função. O tipo função deve expressar tanto o tipo do argumento quanto o tipo do retorno. As expressões, conforme mencionado anteriormente, envolvem tanto valores inteiros quanto booleanos, bem como expressões binárias (soma, subtração, etc.), expressões `let`, `lambda`, aplicação de funções e `if-then-else`

```
data Tipo = TInt | TBool | TFuncao Tipo Tipo
deriving (Show, Eq)
data Expressao = ValorI Int
    | ValorB Bool
    | Soma Expressao Expressao
    | Subtracao Expressao Expressao
    | Multiplicacao Expressao Expressao
    | Divisao Expressao Expressao
    | Let Id Expressao Expressao
    | Ref Id
    | Lambda (Id, Tipo) Tipo Expressao
    | Aplicacao Expressao Expressao
    | If Expressao Expressao Expressao
deriving (Show, Eq)
```

A função que realiza a verificação de tipos recebe uma expressão, um ambiente `Gamma` e possivelmente retorna um tipo válido (por isso o retorno `Maybe Tipo`). Caso algum erro ocorra no sistema de tipos, essa função deve retornar `Nothing`. Isso permite o uso de uma notação baseada em monadas.

```
verificarTipos :: Expressao → Gamma → Maybe Tipo
```

Para alguns casos, a verificação de tipos é bem trivial, particularmente a verificação de tipos de expressões envolvendo valores inteiros, valores booleanos e expressões `lambda`

```
verificarTipos (ValorI n) _ = return TInt
verificarTipos (ValorB b) _ = return TBool
```

$$verificarTipos\ (Lambda\ (v, t1)\ t2\ exp)\ g = return\ (TFuncao\ t1\ t2)$$

Para outros casos, a verificação de tipos requer um certo grau de indução (seguindo as regras de derivação vistas em sala de aula). Para a soma, temos a seguinte regra de derivação:

$$\frac{\Gamma \vdash lhs : TInt \qquad \Gamma \vdash rhs : TInt}{\Gamma \vdash soma(lhs, rhs) : TInt}$$

que pode ser traduzida para Haskell como:

```
verificarTipos (Soma l r) gamma =
  verificarTipos l gamma >>= λlt →
  verificarTipos r gamma >>= λrt →
  if lt ≡ TInt ∧ rt ≡ TInt then return TInt else Nothing
```

Pode-se notar que a verificação do tipo das outras expressões binarias, como multiplicação, divisão e subtração segue a mesma regra de derivação. Apenas no caso da divisão que a implementação tem uma leve mudança, pois caso o denominador seja zero, a função deve retornar o tipo Nothing, pois divisões por zero são tratados como erro.

```
verificarTipos (Subtracao l r) gamma =
  verificarTipos l gamma >>= λlt →
  verificarTipos r gamma >>= λrt →
  if lt ≡ TInt ∧ rt ≡ TInt then return TInt else Nothing
verificarTipos (Multiplicacao l r) gamma =
  verificarTipos l gamma >>= λlt →
  verificarTipos r gamma >>= λrt →
  if lt ≡ TInt ∧ rt ≡ TInt then return TInt else Nothing
verificarTipos (Divisao l r) gamma =
  if r ≡ (ValorI 0)
    then Nothing
    else
      verificarTipos l gamma >>= λlt →
      verificarTipos r gamma >>= λrt →
      if lt ≡ TInt ∧ rt ≡ TInt then return TInt else Nothing
```

Similarmente, a verificação de expressões do tipo `let` requer um grau de indução. Supondo uma expressão `let v = e in c`, primeiro verificamos o

tipo da expressão nomeda (`e`) é bem tipada com tipo `t`, adicionamos uma associação `(v, t)` no ambiente `Gamma` original e computamos o tipo de `c` no novo ambiente. Em termos de regras de derivação, terámos:

$$\frac{\Gamma \vdash e : \tau_1 \qquad (x, \tau_1)\Gamma \vdash c : \tau_2}{\Gamma \vdash let(v, e, c) : \tau_2}$$

Em Haskell:

```
verificarTipos (Let v e c) gamma =
  verificarTipos e gamma >>= λt →
  verificarTipos c ((v, t) : gamma)
```

Em relaçao a referencias de variaveis, a implementação é relativamente simples. Deve-se primeiro pesquisar se a variavel está declarada no ambiente, caso dela não esteja, é retornado um erro de variavel não encontrada, caso a variavel esta declarada no ambiente é retornado o tipo associado a essa variavel.

```
verificarTipos (Ref var) gamma = pesquisar var gamma

pesquisar :: Id → Gamma → Maybe Tipo
pesquisar v [ ] = error "Variavel nao declarada."
pesquisar v ((i, e) : xs)
 | v ≡ i = Just e   -- return e
 | otherwise = pesquisar v xs
```

Na verificação dos tipos de expressões if a primeira coisa que deve ser analisada é a condição. Para que a expressão seja valida a condição deve ser do tipo booleano. Em seguida, o tipo da expressão deve ser ser determinada a partir dos tipos das clausulas then e else. Caso os tipos sejam iguais, este será o tipo da expressão como um todo. Caso eles sejam diferentes, a função deve retornar um erro de tipo. A regra para definição do tipo segue:

$$\frac{\Gamma \vdash condicao : \texttt{boolean} \qquad \Gamma \vdash then : \tau \qquad \Gamma \vdash else : \tau}{\Gamma \vdash \{\texttt{if}\ condicao\ then\ else\} : \tau}$$

Em Haskell:

```
verificarTipos (If c t e) gamma =
  verificarTipos c gamma >>= λlc →
```

```
if lc ≡ TBool
  then verificarTipos t gamma >>= λlt →
    verificarTipos e gamma >>= λre →
    if lt ≡ re then return lt else Nothing
  else Nothing
```

Por fim, temos a verfificação da expressão aplicação de função. Esta verificação possui alguns passos, primeiramente deve-se analisar se a definição da função é um expressão lambda, onde toda expressão lambda possui o tipo TFuncao tId tExp. Em seguida deve-se verificar o tipo do argumento da aplicação de função. Por ultimo é feito uma comparação entre o tipo do argumento da função com o tipo tExp. Caso esses tipos forem iguais, o tipo da aplicação será o tipo do argumento. Caso sejam diferentes, a função retornará o tipo Nothing.
A prova deste caso é definida assim:

$$\frac{\Gamma \vdash definicao : (\texttt{TFuncao}\ \tau_1\ \tau_2) \qquad \Gamma \vdash argumento : \tau_1}{\Gamma \vdash \{\texttt{Aplicacao}\ definicao\ argumento\} : \tau_2}$$

Em haskell:

```
verificarTipos (Aplicacao n a) gamma =
  verificarTipos n gamma >>= λt →
  case t of
    (TFuncao t1 t2) → verificarTipos a gamma >>= λarg →
      if arg ≡ t2
        then return arg
        else Nothing
    otherwise → error ("Aplicacao de funcao nao anonima")
```

4 Conclusão

O presente trabalho mostrou a implementação de um verificador simples e tipos em haskell, para uma linguagem que dá suporte para expressões lambda, let, if, aplicações de função e expressões binarias.