Estruturas de Linguagem

Interpretação de Programas (com programação funcional)

Comandos

Francisco Sant'Anna

francisco@ime.uerj.br

http://github.com/fsantanna-uerj/EDL

Comandos (Statements)

- Unidade sintática que descreve uma ação em um programa imperativo
- Atribuição, Controle de Fluxo (sequência, condicional, repetição), Chamadas, etc
- Como representá-los em Haskell?
  - **atribuição**
  - sequência
  - condicional
  - repetição

```
data Cmd = Atr String Exp
         | Seq Cmd Cmd
         | Cnd Exp Cmd Cmd
         | Rep Exp Cmd
  deriving Show

c1 = Seq [ Atr “x” (Num 10),
           Atr “y” (Var “x”) ]

main = print c1
```

Comandos (Statements)

- Como avaliá-los em Haskell?
  - `avaliaCmd :: Cmd -> ???`
  - `avaliaCmd :: Tela -> Cmd -> Tela`
  - `avaliaCmd :: Teclado -> Tela -> Cmd -> Tela`
  - **`avaliaCmd :: Mem -> Cmd -> Mem`**
- 2 questões
  - o que um programa faz efetivamente?
  - como manter a memória (ambiente)?

```
c1 = Seq [ Atr "x" (Num 10),
           Atr "x" (Num 20),
           Atr "y" (Var "x") ]

main = avaliaCmd ??? c1 -> ???
```

```
c2 = Atr "x"
       (Add (Var "x") (Num 1)

main = avaliaCmd ??? c2 -> ???
```

Memória (Ambiente)

- `type Mem = [(String,Int)]`
  - Associa um identificador a um valor inteiro
  - O valor mais recente é adicionado no início

```
Cmd        Mem

           []

x = 10

           [(“x”,10)]

x = 20

           [(“x”,20),(“x”,10)]

y = x

           [(“y”,20),(“x”,20),(“x”,10)]
```

```
type Mem = [(String,Int)]

prog = Seq [ Atr “x” (Num 10),
             Atr “x” (Num 20),
             Atr “y” (Var “x”) ]

mem = avaliaCmd [] prog
```

Memória (Ambiente)

- Como manipular a memória?
  - **`consulta :: Mem -> String -> Int`**
  - **`escreve  :: Mem -> String -> Int -> Mem`**

```
type Mem = [(String,Int)]

consulta :: Mem -> String -> Int
consulta []           id = 0
consulta ((id',v'):l) id = if id == id' then
                             v'
                           else
                             consulta l id

escreve :: Mem -> String -> Int -> Mem
escreve mem id v = (id,v):mem
```

Comandos (Statements)

- Como manter a memória?
  - **avaliaCmd :: Mem -> Cmd -> Mem**

```
type Mem = [(String,Int)]
consulta :: Mem -> String -> Int
escreve  :: Mem -> String -> Int -> Mem

data Cmd = Atr String Exp
         | Seq Cmd Cmd

avaliaCmd :: Mem -> Cmd -> Mem
avaliaCmd mem (Atr id exp) =
                              escreve mem id v where
                                v = avaliaExp mem exp
avaliaCmd mem (Seq c1 c2)  =
                              avaliaCmd mem' c2 where
                                mem' = avaliaCmd mem c1

avaliaExp :: Mem -> Exp -> Int
...
avaliaExp mem (Var id) = consulta mem id
```

Comandos (Statements)

- Condicional

```
type Amb = String -> Int

data Cmd = Atr String Exp
         | Seq Cmd Cmd
         | Cnd Exp Cmd Cmd

avaliaExp :: Amb -> Exp -> Int
...

avaliaCmd :: Amb -> Cmd -> Amb
avaliaCmd amb (Atr id exp)     = ...
avaliaCmd amb (Seq c1 c2)      = avaliaCmd amb' c2
                                 where amb' = avaliaCmd amb c1
avaliaCmd amb (Cnd exp c1 c2) =
                                 if (avaliaExp amb exp) /= 0 then
                                   avaliaCmd amb c1
                                 else
                                   avaliaCmd amb c2
```