Mudeli metaprogrammeerimine Haskellis

Ivo Seeba

Template haskell

- Haskell 98 laiend
- Allows you to do type-safe compile-time meta-programming
- Mudeleid on vaja:
  - Tingimuslik kompileerimine
  - Program reification
  - Algoritmiline programmi konstruktsioon
  - Abstraktsioonid, mis teeb abstraktsioonid keeles ligipääsetavateks
  - Optimeerimine

Teemadest

- Kvaaskodeeringu mehhanism monaadarvutusse tõlkimisega, on vastava teegi ülemises osas.
- Staatiline tüübikontrolli algoritm tüübikontrolli ja kompileerimisaja vahel.
- Valmisprogrammeeritud osade kasutamine.
- Juhtumi analüüs – esindatud koodiga tavaline algebraline andmetüüp võimaldab kasuada koodi mehhanisme.
- Monaadi teek varustatud metaprogrammeerimise osaga.
- Võimaldab programmeerijal leida erinevaid andmestruktuure.
- Metaprogrammeerimine annab hulga deklaratsioone.

Peamine idee

- Kuidas teha funktsiooni printf
  - printf "Error: %s on line %d." msg line
  - $(printf "Error: %s on line %d") msg line
    - (\ s0 -> \ n1 -> "Error: " ++ s0 ++ " on line " ++ show n1)
  - printf definitsioon
    - data Format = D | S | L String
    - parse :: String -> [Format]
    - printf :: String -> Expr
    - printf s = gen (parse s)
  - Näiteks:
    - parse "%d is %s" returns [D, L " is ", S]

Veel paindlikke konstruktsioone

- case x of (a,b,c) -> a
- $(sel 1 3) x
- sel :: Int -> Int -> Expr
- sel i n = [| \x -> case x of ... |]
- sel :: Int -> Int -> Expr
- sel i n = lam [pvar "x"] (caseE (var "x") [alt])
- where alt :: Match
- alt = simpleM pat rhs
- pat :: Patt
- pat = ptup (map pvar as)
- rhs :: Expr
- rhs = var (as !! (i-1)) -- !! is 0 based
- as :: [String]
- as = ["a"++show i | i <- [1..n]]

Veel paindlikke konstruktsioone

- Süntaksi konstruktsiooni funktsioonid
  - Mudelite süntaks:
    - pvar :: String -> Patt -- x
    - ptup :: [Patt] -> Patt -- (x,y,z)
    - pcon :: String -> [Patt] -> Patt -- (Fork x y)
    - pwild :: Patt -- _
  - Avaldiste süntaks:
    - var :: String -> Expr -- x
    - tup :: [Expr] -> Expr -- (x,3+y)
    - app :: Expr -> Expr -> Expr -- f x
    - lam :: [Patt] -> Expr -> Expr -- \ x y -> 5
    - caseE :: Expr -> [Match] -> Expr -- case x of ...
    - simpleM :: Patt -> Expr -> Match -- x:xs -> 2

Veel paindlikke konstruktsioone

- sel :: Int -> Int -> Expr
- sel i n = [| \ x -> $(caseE [| x |] [alt]) |]
- where
- alt = simpleM pat rhs
- pat = ptup (map pvar as)
- rhs = var (as !! (i-1))
- as = ["a"++show i | i <- [1..n]]
- genPE :: :: String -> Int -> ([Patt],[Expr])
- genPE s n = let ns = [s++(show i) | i <- [1..n]]
- in (map pvar ns, map var ns)

- apps :: [Expr] -> Expr
- apps [x] = x
- apps (x:y:zs) = apps ([| $x $y |] : zs)

Veel paindlikke konstruktsioone

```
- mkZip :: Int -> Expr -> Expr
- mkZip n name = lam pYs (caseE (tup eYs) [m1,m2])
-   where
-     (pXs, eXs) = genPE "x" n
-     (pYs, eYs) = genPE "y" n
-     (pXSs,eXSs) = genPE "xs" n
-     pcons x xs = [p| $x : $xs |]
-     b = [| $(tup eXs) : $(apps(name : eXSs)) |]
-     m1 = simpleM (ptup (zipWith pcons pXs pXSs)) b
-     m2 = simpleM (ptup (copies n pwild)) (con "[]")
```

Deklaratsioonid ja reifikatsioonid

- data T a = Tip a | Fork (T a) (T a) deriving(Eq)
- data T a = Tip a | Fork (T a) (T a)
- splice (genEq (reifyDecl T)

Kvaaskodeering, Skoopimine ja kodeerimise monaad

- Kvaaskodeerimine (näiteks x-i ja y-i ümbernimetamine)
  - cross2a :: Expr -> Expr -> Expr
  - cross2a f g = [| \ (x,y) -> ($f x, $g y) |]
  - prompt> cross2a (var "x") (var "y")
  - Displaying top-level term of type: Expr
  - \ (x0,y1) -> (x x0,y y1)
  - cross2b f g
  - = lam [ptup [pvar "x", pvar "y"]]
  - (tup [app f (var "x"),app g (var "y")])
  - prompt> cross2b (var "x") (var "y")
  - Displaying top-level term of type: Expr
  - \ (x,y) -> (x x,y y)

Kvaaskodeering, Skoopimine ja kodeerimise monaad

- Secrets Revealed

```
cross2c :: Expr -> Expr -> Expr
cross2c f g =
   do { x <- gensym "x"
      ; y <- gensym "y"
      ; ft <- f
      ; gt <- g
      ; return (Lam [Ptup [Pvar x,Pvar y]]
                    (Tup [App ft (Var x)
                          ,App gt (Var y)]))
}

type Expr = Q Exp
type Decl = Q Dec
gensym :: String -> Q String
```

Kvaaskodeering, Skoopimine ja kodeerimise monaad

- Funktsioonid süntaksi konstrueerimiseks

```
App :: Exp -> Exp -> Exp
app :: Expr -> Expr -> Expr
app x y = do { a <- x; b <- y; return (App a b)}
cross2d :: Expr -> Expr -> Expr
cross2d f g
   = do { x <- gensym "x"
        ; y <- gensym "y"
        ; lam [ptup [pvar x, pvar y]]
              (tup [app f (var x)
                   ,app g (var y)])
}
```

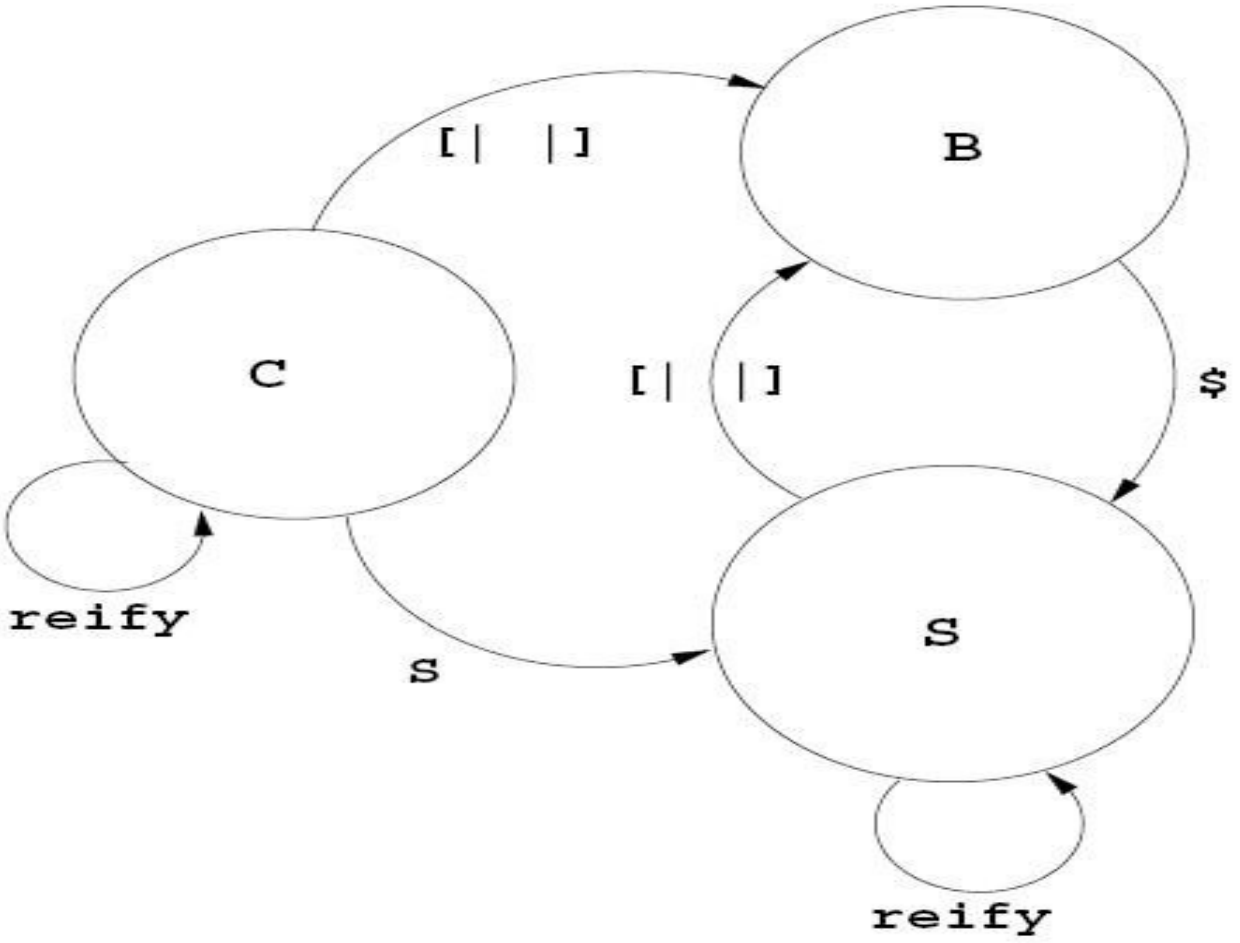
Kvaaskodeering, Skoopimine ja kodeerimise monaad

- Funktsioonid süntaksi konstrueerimiseks

```
cross2e f g =
   do { (vf,p) <- genpat (ptup [pvar "x",pvar "y"])
      ; lam [p] (tup[app f (vf "x"),app g (vf "y")])
      }
cross2e f g =
      do { (vf,p) <- genpat [p| (x,y) |]
         ; lam [p] [| ( $f $(vf "x"), $g $(vf "y") )  |]
         }
```

Typing Template Haskell

- $(printf "Error: %s on line %d") "urk" 341
- „Liimi keha“ tüüpkontroll (First type check the body of the splice; in this case it is printf „Error: %s on line %d“)::Expr), kompileerimine ja käivitamine ning tulemusprogrammi tüübikontroll.
- f x = $(zipN x)
- f :: Int -> Expr
- f x = [| foo $(zipN x) |]
- g x = $(h [| x*2 |])



Avaldised ja deklaratsioonid

- Avaldiste reeglid:
- Deklaratsioonid
  - Kas on õige programm
    - splice (genZips 20)
    - foo = zip3 "fee" "fie" "fum"
  - Grupeerimine, tavakohane sõltuvusanalüüs, tüübikontroll,
- Deklaratsiooni ühendamise (splicing) piirangud
    - f :: Int -> Expr
    - f x = [| let
    - splice (h x)
    - in (p,q)
    - |]
    - f :: Int -> Expr
    - f x = letE (h x) (tup [var "p", var "q"])

States: $s \subseteq C, B, S$

EXPRESSIONS: $\Gamma \vdash_s^n expr : \tau$

$$\frac{\Gamma \vdash_B^{n+1} e : \tau}{\Gamma \vdash_{C,S}^n [|e|] : \mathrm{Q\,Exp}}\ \textsc{Bracket}$$

$$\frac{\Gamma \vdash_S^{n-1} e : \mathrm{Q\,Exp}}{\Gamma \vdash_B^n \$e : \tau}\ \textsc{EscB} \qquad \frac{\Gamma \vdash_C^0 e' : \tau \quad \mathrm{runQ}\, e \mapsto e' \quad \Gamma \vdash_S^{-1} e : \mathrm{Q\,Exp}}{\Gamma \vdash_C^0 \$e : \tau}\ \textsc{EscC}$$

$$\frac{x \in \Gamma}{\Gamma \vdash_{C,S}^n \texttt{reifyDecl}\ x\ : \mathrm{Q\,Dec}}\ \textsc{ReifyDecl}$$

$$\frac{\Gamma;(x:(\tau_x,n)) \vdash_s^n e : \tau}{\Gamma \vdash_s^n \backslash x \texttt{->} e : \tau_x \to \tau}\ \textsc{Lam} \qquad \frac{\Gamma x = (\tau, m) \quad n \geq m}{\Gamma, \vdash_s^n x : \tau}\ \textsc{Var}$$

DECLARATIONS: $\Gamma \vdash_s^n decl : \Gamma' \qquad \Gamma \vdash_s^n [decl] : \Gamma'$

$$\frac{\Gamma;(x:(\tau_1,n));(f:(\tau_1 \to \tau_2,n)) \vdash_s^n e : \tau_2}{\Gamma \vdash_s^n f\, x = e : \{(f:\tau_1 \to \tau_2)_s\}}\ \textsc{Fun}$$

$$\frac{\Gamma \vdash_C^0 [d_1, \ldots, d_n] : \Gamma' \quad \mathrm{runQ}\, e \mapsto [d_1, \ldots, d_n] \quad \Gamma \vdash_C^{-1} e : \mathrm{Q\,[Dec]}}{\Gamma \vdash_C^0 \mathrm{splice}\, e : \Gamma'}\ \textsc{Splice}$$

Figure 2. Typing rules for Template Haskell

Uuesti kodeeringu monaadist

- Reifikatsioon
  - module M where
  - data T a = Tip a | Fork (T a) (T a)
  - repT :: Decl
  - repT = reifyDecl T
  - lengthType :: Type
  - lengthType = reifyType length
  - percentFixity :: Q Int
  - percentFixity = reifyFixity (%)
  - here :: Q String
  - here = reifyLocn
  - Data "M:T" ["a"]
  - [Constr "M:Tip" [Tvar "a"],

Uuesti kodeeringu monaadist

```
Constr "M:Fork"
        [Tapp (Tcon (Name "M:T")) (Tvar "a"),
        Tapp (Tcon (Name "M:T")) (Tvar "a")]]
[]

assert :: Expr -- Bool -> a -> a
assert = [| \ b r ->
             if b then r else
                    error ("Assert fail at "
                           ++ $reifyLocn) |]

find xs n = $assert (n<10) (xs !! n)
find xs n =
              (\ b r -> if b then r else
                        error ("Assert fail at " ++ "line 22 of Foo.hs"))
              (n < 10) (xs !! n)
cassert :: Expr -- Bool -> a -> a
cassert = do { mb <- reifyOpt "DEBUG"
                  ; if isNothing mb then  [| \b r -> r |]
                    else assert }
```

Uuesti kodeeringu monaadist

- Ebaõnnestumine (Failure)
  - zipN :: Int -> Expr
  - zipN n
  - | n <= 1 = fail "Arg to zipN must be >= 2"
  - | otherwise = ...as before...
  - fail :: String -> Q a
- Sisend ja väljund
  - splice (genXML "foo.xml")
  - qIO :: IO a -> Q a
- Koodi trükkimine
  - instance Show Exp
  - instance Show Dec
  - jne...
  - main = do { e <- runQ (sel 1 3) ; putStr (show e) }

Uuesti kodeeringu monaadist

- Q implementeerimine
  - newtype Q a = Q (Env -> IO a)
- Kvaasi koteerimine ja leksiline skoopimine
- Risti-staadiumis püsivus
  - module T(genSwap) where
  - swap (a,b) = (b,a)
  - genSwap x = [| swap x |]
  - module Foo where
  - import T(genSwap)
  - swap = True
  - foo = $(genSwap (4,5))
  - App (Var "T:swap") (Tup [Lit (Int 4), Lit (Int 5)])

Uuesti kodeeringu monaadist

- Risti-staadiumis püsivus
  - App (Var "T:swap") (Tup [Lit (Int 4), Lit (Int 5)])
  - genSwap :: (Int,Int) -> Expr
  - genSwap x = do { t <- lift x; return (App (Var "T:swap") t) }
  - class Lift t where
  - lift :: t -> Expr
  - instance Lift Int
  - lift n = lit (Int n)
  - instance (Lift a,Lift b) => Lift (a,b) where
  - lift(a,b) = tup [lift a, lift b]
  - genSwap x = swap x
- Dünaamiline skoopimine
  - genSwapDyn x = [| $(var "swap") x |]

Kvaasikoodi implementeerimine

- trE :: VEnv -> Exp -> Exp
- trE cl (App (Var "swap") (Var "x"))
- trE cl (App a b) = App (App (Var "app") (trans a)) (trans b)
- trE cl (Cond x y z) = App (App (App (Var "cond") (trans x)) (trans y)) (trans z)
- trE cl ... = ...
- trE cl (App a b) = rep "app" (trEs cl [a,b])
- trE cl (Cond x y z) = rep "cond" (trEs cl [x,y,z])
- trEs :: VEnv -> [Exp] -> [Exp]
- trEs cl es = map (trE cl) es
- rep :: String -> [Exp] -> Exp
- rep f xs = apps (Var f) xs
- where apps f [] = f
- apps f (x:xs) = apps (App f x) xs

Kvaasikoodi implementeerimine

- type VEnv = String -> VarClass
- data VarClass = Orig ModName | Lifted | Bound
- trP :: Pat -> ([Statement Pat Exp Dec],Pat)

Muud samalaadsed tööd

- C++ mudelid
- Skeemi makrod
- MetaML ja selle derivatsioonid
  - Metaml
  - MetaO’Caml
  - MacroML
  - Dynamic Typing

Kirjandus

- http://research.microsoft.com/en-us/um/people/simonpj/papers/meta-haskell/meta-haskell.pdf
- http://www.haskell.org/haskellwiki/Template_Haskell
- http://www.haskell.org/th/
- http://en.wikipedia.org/wiki/Template_Haskell

Tänan kuulamast, küsimused