Lambda-termide redutseerimine

- Lambda-termid

```
type Var  = String
data Term = Var Var
          | App Term Term
          | Lam Var  Term
```

- Vabade muutujate leidmine

```
freeVars :: Term -> [Var]
freeVars (Var x)     = [x]
freeVars (App e1 e2) = freeVars e1 `union` freeVars e2
freeVars (Lam x e)   = delete x (freeVars e)
```

Lambda-termide redutseerimine

- Olekuteisendusmonaad

```
newtype S s a = S (s -> (a,s))

instance Monad (S s) where
       return x = S (\s -> (x,s))
       (S f) >>= k = S (\s -> case f s of
                                (x,s') -> case k x of
                                            S g -> g s')
getS :: S s s
getS  = S (\s -> (s,s))

setS :: s -> S s ()
setS x = S (\s -> ((),x))

runS :: S s a -> s -> (a,s)
runS (S f) s = f s
```

Lambda-termide redutseerimine

- Uute muutujate genereerimine

```
newVar :: S Int Var
newVar = do i <- getS
            setS (i+1)
            return ("x" ++ show i)
```

Lambda-termide redutseerimine

- Substitutsioon

```
subst :: Term -> (Var,Term) -> S Int Term
subst t (x,e) = subs t
  where fvs = freeVars e
        subs (Var y) | x == y    = return e
                     | otherwise = return (Var y)
        subs (App e1 e2) = do e1' <- subs e1
                              e2' <- subs e2
                              return (App e1' e2')
        subs (Lam y e1) | x == y         = return (Lam y e1)
                        | notElem y fvs = do e1' <- subs e1
                                             return (Lam y e1')
                        | otherwise = do z    <- newVar
                                         e1'  <- subst e1 (y, Var z)
                                         e1'' <- subs e1'
                                         return (Lam z e1'')
```

Lambda-termide redutseerimine

- Ühesammuline reduktsioon (aplikatiivjärjekorras)

```
reduA :: Term -> S Int (Maybe Term)
reduA (Var x) = return Nothing
reduA (Lam x e) = do me' <- reduA e
                     case me' of
                       Just e' -> return (Just (Lam x e'))
                       Nothing -> return Nothing
```

Lambda-termide redutseerimine

- Ühesammuline reduktsioon (aplikatiivjärjekorras)

```
reduA (App e1 e2)
  = do me1 <- reduA e1
       case me1 of
         Just e1' -> return (Just (App e1' e2))
         Nothing  ->
             do me2 <- reduA e2
                case me2 of
                  Just e2' -> return (Just (App e1 e2'))
                  Nothing  -> case e1 of
                                Lam x e0 -> do e <- subst e0 (x,e2)
                                               return (Just e)
                                _        -> return Nothing
```

Lambda-termide redutseerimine

- Ühesammuline reduktsioon (normaaljärjekorras)

```
reduN :: Term -> S Int (Maybe Term)
reduN (Var x) = return Nothing
reduN (Lam x e) = do me' <- reduN e
                     return (fmap (\e' -> Lam x e') me')
reduN (Lam x e1 `App` e2) = do e <- subst e1 (x,e2)
                               return (Just e)
reduN (App e1 e2)
  = do me1 <- reduN e1
       case me1 of
         Just e1' -> return (Just (App e1' e2))
         Nothing  -> do me2 <- reduN e2
                        return (fmap (\e2' -> App e1 e2') me2)
```

Lambda-termide redutseerimine

- Reduktsioonijada genereerimine

```
iterateSM :: (a -> S Int (Maybe a)) -> a -> S Int [a]
iterateSM f x = do y <- f x
                   case y of
                     Just y' -> do ys <- iterateSM f y'
                                   return (x:ys)
                     Nothing -> return [x]

reduceA :: Term -> S Int [Term]
reduceA = iterateSM reduA

reduceN :: Term -> S Int [Term]
reduceN = iterateSM reduN
```

Lambda-termide redutseerimine

- Parametriseeritud olekuteisendusmonaad

```
newtype S m s a = S (s -> m (a,s))

instance Monad m => Monad (S m s) where
      return x    = S (\s -> return (x,s))
      (S f) >>= k = S (\s -> do (x,s') <- f s
                                case k x of
                                     S g -> g s')
getS :: Monad m => S m s s
getS  = S (\s -> return (s,s))

setS :: Monad m => s -> S m s ()
setS x = S (\s -> return ((),x))

runS :: Monad m => S m s a -> s -> m (a,s)
runS (S f) s = f s
```

Lambda-termide redutseerimine

- Parametriseeritud olekuteisendusmonaad

```
instance MonadPlus m => MonadPlus (S m s) where
      mzero                 = S (\s -> mzero)
      (S f) `mplus` (S g) = S (\s -> f s `mplus` g s)
```

- Uute muutujate genereerimine, substitutsioon

```
type StM a = S Maybe Int a

newVar :: StM Var
newVar = ...

subst :: Term -> (Var,Term) -> StM Term
subst t (x,e) = ...
```

Lambda-termide redutseerimine

- Ühesammuline reduktsioon (aplikatiivjärjekorras)

```
reduA :: Term -> StM Term
reduA (Var x)   = mzero
reduA (Lam x e) = reduA e >>= \e' -> return (Lam x e')
reduA (App e1 e2)
      = (reduA e1 >>= \e1' -> return (App e1' e2)) `mplus`
        (reduA e2 >>= \e2' -> return (App e1 e2')) `mplus`
        (case e1 of
           Lam x e0 -> subst e0 (x,e2)
           _        -> mzero)
```

Lambda-termide redutseerimine

- Ühesammuline reduktsioon (normaaljärjekorras)

```
reduN :: Term -> StM Term
reduN (Var x)   = mzero
reduN (Lam x e) = reduN e >>= \e' -> return (Lam x e')
reduN (Lam x e1 `App` e2) = subst e1 (x,e2)
reduN (App e1 e2)
      = (reduN e1 >>= \e1' -> return (App e1' e2)) `mplus`
        (reduN e2 >>= \e2' -> return (App e1 e2'))
```

Lambda-termide redutseerimine

- Reduktsioonijada genereerimine

```
iterateStM :: (a -> StM a) -> a -> StM [a]
iterateStM f x = (do ys <- f x >>= \ y -> iterateStM f y
                     return (x : ys)) `mplus`
                 return [x]

reduceA :: Term -> StM [Term]
reduceA = iterateStM reduA

reduceN :: Term -> StM [Term]
reduceN = iterateStM reduN
```