Tangible Value

in Haskell
Conal Elliott

- http://www.youtube.com/watch?v=faJ8N0giqzw
- http://conal.net/papers/Eros/
- http://journal.conal.net/#
  [[separating IO from logic -- example]]
- http://conal-elliott.blogspot.com/search/label/TV
Eros
ICFP 2007, Freiburg
Conal Elliott
Tangible  Functional
Programming

Added TV of type ([Char],[Char]) -> [Char]
applications:  
1. user-friendly  
2. usable  
3. concrete  
4. visual  

libraries:  
1. programmer-friendly  
2. composable  
3. abstract  
4. syntactic
UNIX philosophy

• Write programs that do one thing and do it well
• Write programs to work together
• Write programs to handle text streams, because that is a universal interface

Doug McIlroy
godfat ~/p/l/l/l/l/proc> ls | sort | cat -n
1 bind.rb
2 chain.rb
3 compose.rb
4 curry.rb
TV
程式 原始
module Grading where

import Data.List (sort)
import Data.Map (Map,empty,keys,insertWith,findWithDefault)
import Text.Printf

import Interface.TV
import Interface.TV.OFun() -- work around GHC bug.  ticket #1145
grades = do
src <- readFile "tasks"
let pairs = map (split.words) (lines src)
grades = foldr insert empty pairs
mapM_ (draw grades) (sort (keys grades))
where
insert (s, g) = insertWith (++) s [g]
split [name,mark] = (name, read mark)
draw g s = printf "%s\t%s\tAverage: %f\n" s (show marks) avg
where
marks = findWithDefault (error "No such student") s g
avg = sum marks / fromIntegral (length marks) :: Double
gradingStr src = concatMap (draw grades) (sort (keys grades))
    where
    pairs = map (split.words) (lines src)
    grades = foldr insert empty pairs

    insert (s, g) = insertWith (++) s [g]
    split [name,mark] = (name, read mark)
    draw g s = printf "\%s\t\%s\tAverage: \%f\n" s (show marks) avg
        where
        marks = findWithDefault (error "No such student") s g
        avg = sum marks / fromIntegral (length marks) :: Double
type GradingStr = String -> String
gradingStr :: GradingStr

grades_2 = readFile "tasks" >>= return . gradingStr >>= putStrLn
in TV
type GradingStr = String -> String
gradingStr :: GradingStr

grades_2 = readFile "tasks" >>= return . gradingStr >>= putStrLn

gradingStrOut = oLambda (fileIn "tasks") stringOut
gradingStrT :: TV KIO GradingStr
gradingStrT = tv gradingStrOut gradingStr

grades_3 = runTV gradingStrT
視覺化
融合
first :: (a -> a') -> ((a, b) -> (a',b ))
second :: (b -> b') -> ((a, b) -> (a ,b'))
result :: (b -> b') -> ((a->b) -> (a->b'))

first f = \ (a, b) -> (f a,  b)
second g = \ (a, b) -> (  a, g b)
result g = \ f -> g . f
sf :: (b -> b') -> (a, (b , c))
    -> (a, (b', c))

sf = second.first

frsrf :: (c -> c') -> (a -> (f, b -> (c , g)), e)
    -> (a -> (f, b -> (c', g)), e)

frsrf = first.result.second.result.first

funFirst ::
    (d -> (c -> a)) -> ((d, b) -> (c -> (a, b)))
MV
分離
type TV a = (Out a, a)

type Out a = ...
put :: Put a -> Out a
opair :: Out a -> Out b -> Out (a, b)
olambda :: In a -> Out b -> Out (a->b)

type In a = ...
get :: Get a -> In a
ipair :: In a -> In b -> In (a,b)
Eros

- TypeCompose
- DeepArrow
- DataDriven
- Phooey
- TV
- GuiTV
- wxHaskell
- wxWidgets
To explore

- Tangible polymorphism?
- Direct structural tweaks
- Symmetric In/Out (ilambda)
- “GUIs are types” as GUI design guide
- TVs as composable MVC