Yicho version 0.1.0: User Manual

Tetsuo Yokoyama, Zhenjiang Hu, and Masato Takeichi

July 19, 2005

1 Introduction

Yicho is a monadic combinator library for supporting declarative specification of program transformation in Haskell. The main features are as follows:

- Expressive *higher-order patterns* and *higher-order matchig* can be used to specify abstract calculations.
- The patterns are *first-class* – they can be named, constructed by smaller ones, and passed as parameters. – for reusable calculation.
- The matching algorithm is *deterministic* and efficient.

For example, consider the *foldr promotion rule*, a well-known calculation rule [BdM96] to push a function f into a *foldr*.

$$\frac{\begin{array}{rcl} e' & = & f\ z \\ f\ (g\ x\ y) & = & g'\ x\ (f\ y) \end{array}}{f \circ \mathit{foldr}\ g\ z = \mathit{foldr}\ g'\ e'}$$

This can be straightforwardly expressed by the following Haskell code:

```
promotion :: RuleY -> RuleY
promotion laws exp = do
  [f,oplus,otimes,e,e'] <- pvars ["f","oplus","otimes","e","e'"]
  [| $f . foldr $oplus $e |] <=== laws [| $exp |]
  [| $e' |] <=== laws [| $f $e |]
  [| \x xs -> $otimes x ($f xs) |] <=== laws [| \x xs -> $f ($oplus x xs) |]
  ret [| foldr $otimes $e' |]
```

Function `promotion` takes calculation laws and an expression and returns an expression `[| foldr $otimes $e' |]`. In the third line, `f`,`oplus`,`otimes`,`e`, and `e'` are declared as variables. In the fourth line, `exp` is transformed by `laws` and matched with `[| $f . foldr $oplus $e |]`. The result match is kept track by `Y` monad, and it will instanciate the variables `f`, `oplus`, and `e` appearred in the following. It is worth noting that the fifth and sexth lines are almost same as the above calculation law.

The function `promotion` can only apply to the function composition where the second one is in the form of `foldr`. To promote an arbitrary function compositions, we may define

```
applypromotion :: RuleY -> RuleY
applypromotion laws exp = do
   [f,g,h] <- pvars ["f","g","h"]
   caseM [| $exp |] [
         [| $f . $g |] ===> do g1 <- applypromotion laws [| $g |]
                               promotion laws [| $f . $g1 |],
         [| $h |] ===> promotion laws [| $h . foldr (:) [] |]
                 ]
```

and we can describe

```
ex1 = applypromotion mylaws [| sum . map (^2) |]
```

This new definition can be used as

```
$(runY ex1) [1,2,3]
```

which is evaluated into `14`. It is worth noting that all the transformation is done at compile time and there is no overhead at run time.

2 How to Install and Uninstall Yicho

We tested on linux and GHC 6.4.1. We cannot install the Yicho system for Windows and GHC 6.4.1. To install the Yicho system, you need to type

```
$ runhaskell Setup.hs configure --ghc --prefix=$(HOME)
$ runhaskell Setup.hs build
$ runhaskell Setup.hs install --user
```

Here, you need to specify by `$(HOME)` where you are going to install Yicho. To uninstall Yicho, type

```
$ ghc-pkg unregister Yicho-$(VER) --user
```

Here, `$(VER)` should be replaced with the replaced with the corresponding version of Yicho (e.g. `0.1.0`).

3 How to Use Yicho

If you do not install the system, all you need to do is import `Yicho.hs` in your module. When compiling, you need to specify the option like

```
$ ghc -fglasgow-exts -package Yicho-*.*.*  *.hs
```

4 Documentations

For more information, please refer to

- The Yicho website
  http://www.ipl.t.u-tokyo.ac.jp/yicho
- Hierarchical Module Structure
  http://www.ipl.t.u-tokyo.ac.jp/yicho/doc/index.html

References

[BdM96] Richard Bird and Oege de Moor. *Algebra of Programming.* Prentice Hall, 1996.