モナドで作る
正規表現コンビネータライブラリ

原井 彰弘
（はらい あきひろ）

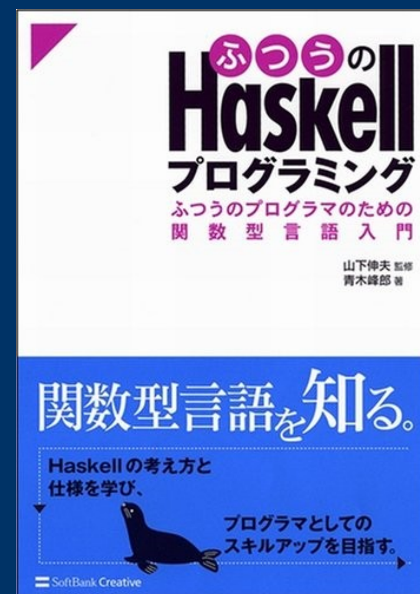
自己紹介

原井 彰弘（はらい あきひろ）

Twitter: @no_ga_itai（脳痛さん）

Haskell を知ったのは
「ふつける」が出る半年くらい前
2005 年末くらい？

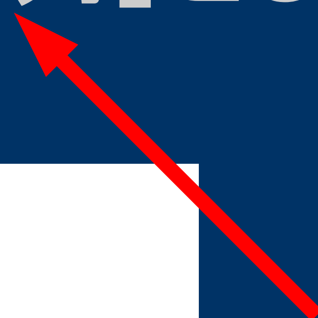
Java で必ず final を付ける
自分の性格によくマッチ



モナドを使った正規表現コンビネータライブラリ

きっかけ その１

GHC 付属の正規表現ライブラリが
正規表現を「文字列」として扱うことに不満



- Text.Read.Lex
- Regex
  - Text.Regex.Base
    - Text.Regex.Base.Context
    - Text.Regex.Base.Impl
    - Text.Regex.Base.RegexLike
  - Text.Regex.Posix
    - Text.Regex.Posix.ByteString
    - Text.Regex.Posix.String
    - Text.Regex.Posix.Wrap
- Text.Show

DSL では無理？

きっかけ その2

Yet Another Haskell Tutorial

Chapter 9 "Monads" 119 ページ

to use existing monads and (2) learning how to write new ones. If you want to use Haskell, you must learn to use existing monads. On the other hand, you will only need to learn to write your own monads if you want to become a "super Haskell guru." Still, if you can grasp writing your own monads, programming in Haskell will be much more pleasant.

「Haskell のグルになりたい者のみモナドを作れ」

「されど、モナドの作り方が分かれば Haskell ライフはより豊かになるだろう」

モナドを使って
Parsec っぽい
正規表現コンビネータを作れば
一挙両得？？

モナドの能力的に可能か？

バックトラックは？

後方参照は？

モナドの能力的に可能か？

- バックトラックは？
  - 従来型 NFA ならば
リストと遅延評価を使って自然に書ける
- 後方参照は？
  - 言語内 DSL のメリットを生かせば
自由自在

こんな感じに使います

わかりやすさ重視のため
冗長性には目をつぶってください・・・

/^(ab)*cd$/

```
testRegex :: Regex ()
testRegex = do
    rxCaret
    (matched, val) <- rxStar $ do
        rxOneChar 'a'
        rxOneChar 'b'
    rxOneChar 'c'
    rxOneChar 'd'
    rxDollar
    return ()
```

rxStar 以下で
マッチした文字列

rxStar から返される
任意の値

実演

メリットとデメリット

メリット

- 強い静的型付け
- 表現の再利用が容易
- 表現の動的な作成と変更が可能
- （文字列ではなく）任意のトークン列への適用が可能（未実装）

デメリット

- 記述が冗長
- 最適化はHaskellコンパイラ依存

実装(一部)

```
newtype Regex a =
  Rx { runRegex :: (RxTarget -> [(RxTarget, a)]) }

data RxTarget = RxTarget {
    rxTargetBefore :: String, --  逆順
    rxTargetMatched :: String, --  逆順
    rxTargetAfter :: String }

instance Monad Regex where
    m >>= next = Rx $ ¥target -> let
        thisCand = runRegex m target
        getNextCand (target', a) = runRegex (next a) target'
        in concatMap getNextCand thisCand
    return a = Rx $ ¥target -> [(target, a)]
```

実装ずみの機能

^　$　*　(　)　*?　.　|
[ ]

後方参照など

モナドで全部できる！

モナドってすごい！

Haskellってすごい！

ご清聴ありがとうございました

twitter: @no_ga_itai（脳痛）

よかったらフォローしてください