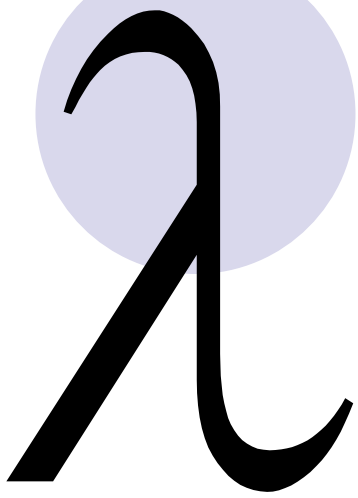
君ならどう書く - Haskell – 規定演技編



酒井 政裕

トピック

- パーサ
  - パーサ・コンビネータ
  - 漢数字のパーサ
  - 式のパーサ
- 式の評価
- まとめ

パーサ・コンビネータ

- パーサを定義し、また単純なパーサからより複雑なパーサを定義するための一連の関数。
- EDSL(Embeded Domain Specific Language)の典型的な例。
  - BNF風の表記がそのままプログラムに!?
- 実装は色々あるけど、今回は自前で実装。

パーサ・コンビネータ (cont'd)

式	言語	説明
return a	ε	常に成功しaを返すパーサ
p <|> q	p | q	pとqの選択
mzero	$\varnothing$	常に失敗するパーサ
char c	{c}	文字cを受理するパーサ
string s	{s}	文字列sを受理するパーサ
p >>= f	p f(x)	pとf(x)の連接 (xはpのパース結果)
p >> q	p q	pとqの連接
many p	p*	0回以上の繰り返し

モナド(Monad)

- compuatationを抽象化するインターフェース
  - 普通の計算だけでなく、色々な計算(「例外」「副作用」「非決定性」があったり無かったりとか)を統一して扱いたい。
  - 元々は圏論の概念だけど、それはどうでもいい。
- このパーサコンビネータもモナドになっている
  - returnと(>>=)が存在してある法則を満たす。
  - 既存のライブラリと糖衣構文(do記法)が使える。

漢数字のパーサ

- 最初に考えた文法
  - number ::= n1
  - n1 ::= n2 | n2? “無量大数” n2?
  - n2 ::= n3 | n3? “不可思議” n3?
  - …
  - nn ::= digit
- 効率が悪い

漢数字のパーサ (cont'd)

- 実装した文法
  - number ::= n1＼{ ε }
  - n1 ::= n2 (“無量大数” n2)?
  - n2 ::= n3 (“不可思議” n3)?
  - …
  - nn ::= digit | ε
- 特徴
  - 「一」は省略してもOK
  - 「一万億」なんてのも受け付けてしまう
    - これは後で気づいた。とりあえず放置。

漢数字のパーサ (cont'd)

```
number :: Parser Integer
number = do x <- number' units
                  case x of
                   Just y  -> return y
                   Nothing -> mzero

number' :: [(String, Integer)] -> Parser (Maybe Integer)
number' [] = liftM Just digit <|> return Nothing
number' ((s,u) : us) =
    do x <- number' us
       let p = do string s
                  y <- number' us
                  return (Just (fromMaybe 1 x * u + fromMaybe 0 y))
       p <|> return x
```

式のパーサ

- 左再帰を除去した形の文法でパース
  - expr　::= term (('＋'|'－‘) term)*
  - term　::= factor (('×'|'÷‘) factor)*
  - factor ::= ‘（’ expr ‘）’ | number

式のパーサ (cont'd)

```
-- genExpr(sub,op) ::= sub (op sub)*
genExpr :: Parser Expr -> Parser BinOp -> Parser Expr
genExpr sub op = do e  <- sub
                    xs <- many p
                    return (foldl (¥a (o,b) -> BinOpApp o a b) e xs)
   where p = do o <- op
                x <- sub
                return (o,x)

-- expr ::= term (('＋'|'－') term)*
expr :: Parser Expr
expr = genExpr term op
   where op =  (string "＋" >> return Add)
           <|> (string "－" >> return Sub)
```

式のパーサ (cont'd)

```
-- term ::= factor (('×'|'÷') factor)*
term :: Parser Expr
term = genExpr factor op
    where op =  (string "×" >> return Mul)
             <|> (string "÷" >> return Div)

-- factor ::= '(' expr ')' | number
factor :: Parser Expr
factor = parenthesize expr
       <|> do x <- number
              return (Number x)
```

パース結果のデータ構造

- data Expr
  = Number Integer
  | BinOpApp BinOp Expr Expr
  data BinOp = Add | Sub | Mul | Div

- 代数的データ型
  - タグ付のunionみたいなもの
  - パラメタを持たない場合はenumみたいなもの

式の評価

- このExprに表示的意味論を定義する
  - 意味領域

```
data Value = Maybe Rational
....
```

  - 意味関数

```
eval :: Expr -> Value
eval (Number x)  = fromIntegral x
eval (BinOpApp op a b) =
   evalBinOp op (eval a) (eval b)
....
```

実行霊もとい実行例

> 八÷(一一一÷五)

十

> 一×二×四×十六×二百五十六×六万五千五百三十六×四十二億九千四百九十六万七千二百九十六×千八百四十四京六千七百四十四兆七百三十七億九百五十五万千六百十六一一

百七十澗千四百十一溝八千三百四十六穣四百六十九禾予二千三百十七垓三千百六十八京七千三百三兆七千百五十八億八千四百十万五千七百二十七

まとめ

- Haskellの高い抽象化能力
- パーサ・コンビネータによるパーサの定義
- 代数的データ型による再帰的なデータ構造の表現