Why Haskell matters

Emacs in Haskell

Jens Petersen

<petersen@redhat.com>

レッドハット株式会社

Linux Conference 2001

明治記念会館

２００１年９月

概要

Haskell

- **関数型言語**
- **静的型**
- **高階関数**
- **遅延評価**
- **インプリメンテーション**

Emacs

- Emacs **の問題点**
- **新** Emacs: Emacs in Haskell

関数型言語

関数型プログラミング

- 命令の実行よりも、式の評価を強調
- 関数を使って値を組み合わせ、式を構成

高階関数

- つまり関数は「first class」
- モジュール化が更に進むこと

動的型

- Lisp と Scheme

静的型

- 「strict」な ML や「lazy」な Haskell

Haskell

汎用の純粋関数型プログラミング言語

プログラミング言語設計の分野での最近の革新の多くが取入れ

高階関数

非正格の意味論

静的多相型付け

利用者定義の代数的データ型

パターン照合

リストの内包表記

モジュールシステム

遅延評価（「lazy」）

プリミティブデータ型

任意倍長整数

```
1234567890987654321 :: Integer
```

固定倍長整数

```
42 :: Int
```

Character

```
'a' :: Char
```

Boolean

```
True, False :: Bool
```

浮動小数点数

```
3.14159 :: Float
Double
```

リスト

```
1:2:3:[] :: [Int]
[1, 2, 3] :: [Integer]
['a', 'b', 'c'] :: [Char]
"Hello" :: String = [Char]
```

タプル

```
('b',4) :: (Char,Integer)
```

関数

```
inc :: Integer -> Integer
inc n = n+1
```

```
inc (inc 3) => 5
```

多相型 と 型変数

```
a,b,c...

length :: [a] -> Integer
length []     =  0
length (x:xs) =  1 + length xs

          length [1, 2, 3]  =>  3
          length [’a’, ’b’, ’c’] => 3
          length [[1], [2], [3]] => 3

infixr 5 ++
(++) :: [a] -> [a] -> [a]
[] ++ ys = ys
(x:xs) ++ ys == x:(xs ++ ys)

          [1, 2, 3] ++ [4, 5, 6, 7]
                  => [1, 2, 3, 4, 5, 6, 7]
```

利用者定義のデータ型

```
data Bool = False | True
```

```
data Color = Red | Yellow | Green | Blue
```

タプル

```
data Point a = Pt a a
```

```
Pt  2.0  3.0 :: Point Float
Pt  'a'  'b' :: Point Char
Pt True False :: Point Bool
```

Recursive Types

ツリー

```
data Tree a = Leaf a
            | Branch (Tree a) (Tree a)


Branch :: Tree a -> Tree a -> Tree a
Leaf :: a -> Tree a

fringe :: Tree a -> [a]
fringe (Leaf x) =  [x]
fringe (Branch left right) =
              fringe left ++ fringe right
```

Type Synonyms

```
type String = [Char]

type Person = (Name,Address)
type Name = String
data Address = None | Addr String

type AssocList a b = [(a,b)]
```

Arithmetic Sequences

```
[1..10] =>  [1,2,3,4,5,6,7,8,9,10]
[1,3..10] => [1,3,5,7,9]
[1,3..] => [1,3,5,7,9, ... (infinite sequence)
```

List Comprehensions

```
[ f x | x <- xs ]
[ (x,y) | x <- xs, y <- ys ]

quicksort :: (Ord a) => [a] -> [a]
quicksort  [] =  []
quicksort (x:xs) =
   quicksort [y | y <- xs, y<x ]
   ++ [x]
   ++ quicksort [y | y <- xs, y>=x]
```

高階関数

```
map :: (a->b) -> [a] -> [b]
map f [] =  []
map f (x:xs) =  f x : map f xs
```

```
map inc [1, 3, 5] => [2, 4, 6]
```

遅延評価

```
ones = 1 : ones

numsFrom n = n : numsFrom (n+1)

squares = map (^2) (numsfrom 0)

take :: Int -> [a] -> [a]
take 0 _ = []
take _ [] = []
take n (x:xs) | n > 0 = x : take (n-1) xs
take _ _ = error "take: negative argument"
```

```
take 3 ones => [1, 1, 1]
take 4 (numsFrom 6) => [6, 7, 8, 9]
take 5 squares => [0,1,4,9,16]
```

遅延評価 (2)

```
zip :: [a] -> [b] -> [(a,b)]
zip (x:xs) (y:ys)       = (x,y) : zip xs ys
zip  xs     ys          = []

fib :: [Integer]
fib = 1:1:[ a+b | (a,b) <- zip fib (tail fib)]
```

Type Classes

```
class Eq a where
  (==) :: a -> a -> Bool

          (==) :: (Eq a) => a -> a -> Bool
elem :: (Eq a) => a -> [a] -> Bool
x ‘elem‘ [] = False
x ‘elem‘ (y:ys) = x==y || (x ‘elem‘ ys)

instance Eq Integer where
  x == y =  x ‘integerEq‘ y
instance Eq Float where
  x == y =  x ‘floatEq‘ y

instance (Eq a) => Eq (Tree a) where
  Leaf a == Leaf b =  a == b
  (Branch l1 r1) == (Branch l2 r2) =
     (l1==l2) && (r1==r2)
  _ == _ =  False
```

Input/Output

```
getChar :: IO Char
putChar :: Char -> IO ()
```

```
main :: IO ()
main =  do c <- getChar
           putChar c
```

```
getLine :: IO String
getLine = do c <- getChar
             if c == '\n'
               then return ""
               else do l <- getLine
                       return (c:l)


todoList :: [IO ()]
todoList = [putChar 'a',
             do putChar 'b'
                putChar 'c',
             do c <- getChar
                putChar c]
sequence_ :: [IO ()] -> IO ()
```

コンパイラ

GHC 5.00.2 (Glasgow Haskell Compiler)

http://haskell.org/ghc/

- BSD **ライセンス**
- **インタラクティブモード** (ghci)
- **パッケージ**
- **スレッド**
- **プロフィール**
- happy (yacc for Haskell)
- hsc2hs

コンパイラ（続き）

NHC98 1.00

http://www.cs.york.ac.uk/fp/nhc98/

- Artistic **ライセンス**
- hmake
- hat

インタープリタ

Hugs98

http://www.haskell.org/hugs/

スレッド

Haskell 98 Library Report

Ratio
Complex
Numeric
Ix
Array
List
Maybe
Char
Monad
IO
Directory
System
Time
Locale
CPUTime
Random

Haskell Libraries (GHC and Hugs) (1)

concurrent
Concurrent

data
Edison, FiniteMap, Set

lang
Addr, Bits, ByteArray, CCall, CError, CForeign,
CTypes, CTypesISO, CString, DiffArray, Dynamic, Exception, Foreign, ForeignObj, ForeignPtr, GlaExts, IArray, Int, IOExts, LazyST, MArray, MarshalAlloc, MarshalArray, MarshalError, MarshalUtils, MutableArray, NumExts, PackedString, Ptr, ShowFunctions, ST, Stable, StableName, StablePtr, Storable, StorableArray, Weak, Word

Haskell Libraries (2)

net	**BSD, Socket, PrimSocket, URI**
posix	**PosixDB, PosixErr, PosixFiles, PosixIO, PosixProcEnv, PosixProcPrim, PosixTTY, PosixUtil**
text	**HaXmL, MatchPS, Parsec, Pretty, Regex, RegexString**
util	**GetOpt, Memo, QuickCheck, Readline, Select**

ライブラリ

gtk+hs

http://www.cse.unsw.edu.au/~chak/haskell/gtk/

HaXmL

http://www.cs.york.ac.uk/fp/HaXmL/

FFI

http://haskell.org/ghc/docs/latest/set/ffi.html

qForeign

http://qforeign.sourceforge.net/

アプリケーション

Haskell Web Server

http://www.haskell.org/~simonmar/bib.html

linkchk

http://www.01.246.ne.jp/~juhp/linkchk/

Emacs の問題点

- C のコードベースとデザインが古い
- 拡張言語である Emacs Lisp の動的型付け
- レキシカルスコープがない
- Emacs Lisp は遅い
- モジュールがない
- スレッドがない
- 他

Emacs in Haskell

新しいプロジェクト

Haskell **で書かれた 新** Emacs

Haskell **は拡張言語**

gtk

スレッド

モジュール

新しい多目的環境を提供

- dired
- **エディタ**
- **メール**
- **ブラウザ**
- Haskell **等の開発環境**
- **シェル**