第5章　Haskellの Sexy Type

ここでは Haskell の型システムの “sexy な”（非標準の）拡張の一部を紹介する。

5.1　Multiparameter Type Classes

型クラスが複数個のパラメータを取ることを許す。

```
class Iso a b where
  iso :: a -> b
  osi :: b -> a

instance Iso a a where iso = id

class Sub a b where
  sub :: a -> b
  sup :: b -> Maybe a
```

5.1.1　Functional Dependencies

型パラメータの間の “依存性” を明示できる。

```
class Collects e ce | ce -> e where
  empty  :: ce
  insert :: e -> ce -> ce
  member :: e -> ce -> Bool
```

5.1.2　Associated Types

型パラメータによって異なる実装を持つ型を表現する。

```
-- 要するに、こういうことがしたい。
--   data Array e
--   data Array Int = IntArray UIntArray
--   data Array Bool = BoolArray BitVector
--   data Array (a,b) = PairArray (Array a) (Array b)
```

```
class ArrayElem e where
  data Array e
  index :: Array e -> Int -> e

instance ArrayElem Int where
  data Array Int = IntArray UIntArr
  index (IntArray ar) i = indexUIntArr ar i

instance (ArrayElem a, ArrayElem b) => ArrayElem (a,b) where
  data Array (a,b) = PairArray (Array a) (Array b)
  index (PairArray ar br) i = (index ar i, index br i)
```

5.2 Lazy Functional State Threads (runST)

“状態” をプログラマが直接操作できないようにして、効率的な更新 (in-place update) ができるようにし、さらに “参照型” を効率的に実装できるようにする。

```
newVar :: a -> ST s (MutVar s a)
readVar :: MutVar s a -> ST s a
writeVar :: MutVar s a -> a -> ST s ()

runST :: ∀a. (∀s. ST s a) ->  a

-- 以下のプログラムは型エラーになる
-- let v = runST (newVar True)
-- in  runST (readVar v)
```

5.2.1 Uniqueness Type *(Concurrent Clean)*

型に “uniqueness” という属性を付加することで、データの single-threadedness を保証し、in-place update が可能であることを保証する。

（以下の例は Haskell 風の書き方に書き直している。）

```
fwritec:: Char -> *File -> *File

WriteABC:: *File -> *File
WriteABC file = fwritec 'c' (fwritec 'b' (fwritec 'a' file))

-- 以下のようなコードは型エラーになる。
-- (file, fwritec 'a' file)
```

5.3 Implicit Parameters

型クラスの仕組みを利用して、動的スコープを持つ変数を実現する。

```
sortBy :: (a -> a -> Bool) -> [a] -> [a]

sort   :: (?cmp :: a -> a -> Bool) => [a] -> [a]
sort    = sortBy ?cmp

least   :: (?cmp :: a -> a -> Bool) => [a] -> a
least xs = head (sort xs)

min :: [a] -> a
min  = let ?cmp = (<=) in least
```

5.4 Generalized Algebraic Data Types (GADTs)

構成子の型を明示的に与えることができる。

```
data Term :: * -> *  where
   Lit  :: Int -> Term Int
   Inc  :: Term Int -> Term Int
   IsZ  :: Term Int -> Term Bool
   If   :: Term Bool -> Term a -> Term a -> Term a
   Pair :: Term a -> Term b -> Term (a,b)
   Fst  :: Term (a,b) -> Term a
   Snd  :: Term (a,b) -> Term b
```

戻り値の型が常に `Term a` というわけではない。

```
eval :: Term a -> a
eval (Lit i) = i
eval (Inc t) = eval t + 1
eval (IsZ t) = eval t == 0
eval (If b t e) = if eval b then eval t else eval e
eval (Pair a b) = (eval a, eval b)
eval (Fst t) = fst (eval t)
eval (Snd t) = snd (eval t)
```

```
data Monad a where
   Unit :: a -> Monad a
   Bind :: Monad a -> (a -> Monad b) -> Monad b
```

5.5 Lexically Scoped Type Variables

明示的な型 (type annotation) を、プログラムのさまざまな部分式に対して指定することができる。

```
f :: forall a. [a] -> [a]
f xs = ys ++ ys
     where
       ys :: [a]
       ys = reverse xs
```

5.5.1 Arbitrary-rank Explicit Universal Quantification

明示的な型を適宜与えることによって、関数に higher-rank 型を与えることができる。

```
foo :: ([Bool], [Char])
foo = let
        f :: (forall a. [a] -> [a]) -> ([Bool], [Char])
        f x = (x [True, False], x [ 'a ', 'b '])
      in f reverse
```

この章の参考文献

[1] Simon Peyton Jones, Mark P. Jones and Erik Meijer 「Type classes: an exploration of the design space」 Haskell Workshop, 1997 年 6 月

[2] Mark P. Jones 「Type Classes with Functional Dependencies」 Proceedings of the 9th European Symposium on Programming (LNCS 1782), 2000 年 3 月

[3] Manuel M. T. Chakravarty, Gabriele Keller, Simon Peyton Jones, and Simon Marlow 「Associated types with class」 32nd ACM symposium on Principles of Programming Languages (POPL'05) New York, 2005 年 1 月

[4] John Launchbury and Simon Peyton Jones 「Lazy Functional State Threads」 SIGPLAN Conference on Programming Language Design and Implementation (PLDI '94), 1994 年

[5] Erik Barendsen and Sjaak Smetsers 「Uniqueness Type Inference」 Proceedings of the 7th International Symposium on Programming Languages: Implementations, Logics and Programs (PLILPS '95), 1995 年

[6] Jeffrey R. Lewis, Mark B. Shields, Erik Meijer and John Launchbury 「Implicit parameters: dynamic scoping with static types」 27th ACM Symposium on Principles of Programming Languages (POPL'00), Boston, 2000 年 1 月

[7] Simon Peyton Jones, Dimitrios Vytiniotis, Stephanie Weirich and Geoffrey Washburn 「Simple unification-based type inference for GADTs」 Proceedings of the Eleventh ACM SIGPLAN International Conference on Functional Programming, Portland, Oregon, 2006 年 9 月

[8] Dimitrios Vytiniotis, Stephanie Weirich, and Simon Peyton Jones 「Boxy types: Inference for higher-rank types and impredicativity」 Proceedings of the Eleventh ACM SIGPLAN International Conference on Functional Programming, Portland, Oregon, 2006 年 9 月