

# プログラミング演習B ML編 第5回

2007/7/3 (通信コース)

2007/7/4 (情報コース)

住井

[http://www.kb.ecei.tohoku.ac.jp/  
~sumii/class/proenb2007/ml5/](http://www.kb.ecei.tohoku.ac.jp/~sumii/class/proenb2007/ml5/)

# 今日のポイント

1. 「組」とパターンマッチングの  
続き
2. 多相データ型
3. リストとリスト型

# レポートについて

課題の解答を

m1-enshu@kb.ecei.tohoku.ac.jp

にメールせよ。件名(Subject)は必ず

kadai5:A1TB2345:東北太郎

第何回の課題か(一桁の数字)

自分の学籍番号

自分の氏名

の形にすること(氏名以外半角)。

締め切りは一週間後の午前8時50分厳守。

質問は上述のアドレスにメールせよ。

- レポートの不正は試験の不正と同様に処置する。

# 前回の復習

- レコード：複数の値を組み合わせた値（ラベルで区別）

```
{ surname = "Sumii",  
  given = "Eijiro",  
  age = 20 }
```

- バリエーション：複数の値のどれか一つを表す値（コンストラクタで区別）

```
datatype int_or_error =  
  Int of int | Error
```

# 組 (pair, tuple)

- ラベルが番号であるような、特殊なレコード

```
- val t = ("Sumii", "Eijiro", 20) ;  
val t = ("Sumii", "Eijiro", 20) :  
  string * string * int  
- #1 t ;  
val it = "Sumii" : string  
- #2 t ;  
val it = "Eijiro" : string  
- #3 t ;  
val it = 20 : int
```

# 組の構文

組：(式<sub>1</sub>, 式<sub>2</sub>, 式<sub>3</sub>, ..., 式<sub>n</sub>)

- {1=式<sub>1</sub>, 2=式<sub>2</sub>, 3=式<sub>3</sub>, ..., n=式<sub>n</sub>}

というレコードと同じ

■ n = 0の空の組 () すなわち空のレコード {}  
のことを「ユニット」という

- 引数や返値が不要な関数において、  
ダミーの値としてよく用いられる

組の型：型<sub>1</sub> \* 型<sub>2</sub> \* 型<sub>3</sub> \* ... \* 型<sub>n</sub>

- {1:型<sub>1</sub>, 2:型<sub>2</sub>, 3:型<sub>3</sub>, ..., n:型<sub>n</sub>}

というレコード型と同じ

■ 空のレコード型 {} は unit 型と同じ

# 前回の復習2

## • バリエーションのパターンマッチング

```
- datatype itree =  
=   ILeaf of int  
= | INode of { left : itree,  
=           right : itree } ;  
datatype itree = ILeaf of int | INode of  
  {left:itree, right:itree}  
- fun isum t = case t of  
=   ILeaf i => i  
= | INode r => isum (#left r) +  
=           isum (#right r) ;  
val isum = fn : itree -> int
```

# バリエーション以外の パターンマッチング

## ● レコード

```
- fun isum t = case t of
=   ILeaf i => i
= | INode { left = l, right = r } =>
=   isum l + isum r ;
val isum = fn : itree -> int
```

## ● 組

```
- val t = ("Sumii", "Eijiro", 20) ;
val t = ("Sumii", "Eijiro", 20)
  : string * string * int
- case t of (s1, s2, _) => s2 ^ " " ^ s1 ;
val it = "Eijiro Sumii" : string
```



# バリエーション以外の パターンマッチング

## ● レコード

```
- fun isum t = case t of
=   ILeaf i => i
= | INode { left = l, right = r } =>
=   isum l + isum r ;
val isum = fn : itree -> int
```

## ● 組

```
- val t = ("Sumii", "Eijiro", 20) ;
val t = ("Sumii", "Eijiro", 20)
      : string * string * int
- case t of (s1, s2, _) => s2 ^ " " ^ s1 ;
val it = "Eijiro Sumii" : string
```

"Don't care"  
(どうでも良い)  
を表すパターン

# バリエーション以外の パターンマッチング (続き)

## ● 整数

```
- fun fib n = case n of
```

```
= 0 => 0
```

```
= | 1 => 1
```

```
= | n => fib (n - 1) + fib (n - 2) ;
```

```
val fib = fn : int -> int
```

上の二つに当てはまらないとき

◆ 参考：パターンマッチングは  
関数定義に直接記述することもできる

```
fun fib 0 = 0
```

```
  | fib 1 = 1
```

```
  | fib n = fib (n - 1) + fib (n - 2)
```

# 多相データ型

例題：次のデータ型を定義せよ。

1. 整数を葉とする木 `itree`
2. 文字列を葉とする木 `stree`
3. 浮動小数を葉とする木 `rtree`

# 解答例？

```
- datatype itree = ILeaf of int
=   | INode of itree * itree ;
datatype itree = ILeaf of int | INode of
  itree * itree
- datatype stree = SLeaf of string
=   | SNode of stree * stree ;
datatype stree = SLeaf of string | SNode of
  stree * stree
- datatype rtree = RLeaf of real
=   | RNode of rtree * rtree ;
datatype rtree = RLeaf of real | RNode of
  rtree * rtree
```

**同じことを何度も書いていて無駄！**

# 葉の型が多相的（型変数）である木のデータ型

```
- datatype 'a tree =  
= Leaf of 'a | Node of 'a tree * 'a tree ;  
datatype 'a tree = Leaf of 'a | Node of 'a  
tree * 'a tree  
- fun size t = (* 共通して使える関数の例 *)  
= case t of  
= Leaf _ => 1  
= | Node(l, r) => size l + size r ;  
val size = fn : 'a tree -> int  
- size (Node(Leaf 3, Leaf 5)) ;  
val it = 2 : int  
- size (Node(Leaf true, Leaf false)) ;  
val it = 2 : int
```

# 課題 5.1

1. 式 `Node(Leaf 3, Leaf 5)` と `Node(Leaf true, Leaf false)` の型は、それぞれ何になるか、確かめよ。
2. 型が `string tree` になるような式を三つ挙げよ。
  - `string tree` と `stree` は別の型なので気をつけよ
3. 式 `Node(Leaf 3, Leaf "abc")` の評価を試みて、結果を考察せよ。
4. `size` が 1, 3, 10 になるような木の例を、一つずつ作れ。それぞれ違う型にすること。

# 課題5.2

先の'a tree型の値を受け取って、それがLeafだったらtrueを、Nodeだったらfalseを返す関数is\_leafを定義せよ。

# 課題 5.3 (optional)

option型とは、  
datatype 'a option =  
SOME of 'a | NONE

と定義された多相データ型である。  
前回のデータ型 `int or error` の  
かわりに、option型を用いて、  
前回の `my_div` や `my_mod` に相当する  
関数を定義せよ。



# リストとリスト型

- ある一つの型の値を、順にいくつか並べたもの

[式<sub>1</sub>, 式<sub>2</sub>, 式<sub>3</sub>, ..., 式<sub>n</sub>]

- 多相テータ型の一種とみなせる

```
datatype 'a list =
```

```
  nil
```

空のリスト

```
  | :: of 'a * 'a list
```

先頭要素と、残りのリストをつなげたノード(**consセル**)

# 例

```
- [1, 2, 3] ;  
val it = [1,2,3] : int list  
- [true, false] ;  
val it = [true,false] : bool list  
- [] ;  
val it = [] : 'a list  
- nil ;  
val it = [] : 'a list  
- 1 :: [2, 3] ;  
val it = [1,2,3] : int list  
- true :: false :: [] ;  
val it = [true,false] : bool list
```

# 注意

- **::**は要素とリストを接続する

- `1 :: [2, 3] ;`

```
val it = [1,2,3] : int list
```

- `[1, 2] :: [3, 4] ;`

```
stdin:2.1-2.17 Error: operator and  
operand don't agree [literal]
```

...

- **リストとリストを連結するのは@**

- `[1, 2] @ [3, 4] ;`

```
val it = [1,2,3,4] : int list
```

# 課題5.4

1. 長さ（要素の数）が3であるような、型の異なるリストを4つ挙げよ。
2. `123` `:: x`と`"abc"` `:: x`のどちらも型エラーにならない、というリスト`x`は存在するか？

# リストのパターンマッチング

- リストも多相データ型的一种なので、パターンマッチングが使える

例：

```
fun length x =  
  case x of  
    nil => 0  
  | _ :: y => 1 + length y
```

# 課題 5.5

次の考え方に基づき、'a tree型の値  $t$  を深さ優先探索して 'a list型の値に変換する関数 `dfs` を定義せよ。

- $t$  が `Leaf x` の形だったら、 $x$  のみを要素とする、長さ1のリストを返す。
- $t$  が `Node (l, r)` の形だったら、左の木  $l$  の変換結果と、右の木  $r$  の変換結果を連結する。

# 課題 5.6 (optional)

1. 関数  $f$  とリスト  $[v_1, v_2, v_3, \dots, v_n]$  を受け取って、それぞれの要素に  $f$  を適用したリスト  $[f\ v_1, f\ v_2, f\ v_3, \dots, f\ v_n]$  を返す、という関数 `map` を書け。また、その型を考察せよ。
2. 先の関数 `length` と上の関数 `map` について、任意のリスト  $x$  と関数  $f$  に対し、`length x` と `length (map f x)` の返り値が (もしあれば) 等しいことを、 $x$  の長さに関する数学的帰納法で示せ。