プログラミング演習B
ML編　第5回

2011/6/7（コミ）
2011/6/8（情報・知能）

住井

http://www.kb.ecei.tohoku.ac.jp/~sumii/class/proenb2011/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

電気・情報系内のマシンから

http://130.34.188.208/ （情報・知能）

http://130.34.188.209/ （コミ）

にアクセスし、画面にしたがって提出せよ。締め切りは一週間後厳守。

- 初回は画面にしたがい自分のアカウントを作成すること。
- 「プログラム」のテキストボックスがある課題では、プログラムとしてsmlに入力した文字列のみを過不足なく正確にコピー＆ペーストして提出せよ。（smlの出力は「プログラム」ではなく考察に含めて書くこと。）
- プログラムの課題でも必ず考察を書くこと。
- 提出したレポートやプログラムの実行結果は「提出状況」から確認できる。
  - 質問はml-enshu@kb.ecei.tohoku.ac.jp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 }
```

- バリアント：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：$(式_1,\ 式_2,\ 式_3,\ \ldots,\ 式_n)$
  - $\{1=式_1,\ 2=式_2,\ 3=式_3,\ \ldots,\ n=式_n\}$ 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：$型_1 * 型_2 * 型_3 * \ldots * 型_n$
  - $\{1:型_1,\ 2:型_2,\ 3:型_3,\ \ldots,\ n:型_n\}$ 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

- バリアント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
```

バリアント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
```

バリアント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

バリアント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
```

- 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. 浮動小数点数を葉とする木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(t1, t2) => size t1 + size t2 ;
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`streee`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

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。

・option型は最初から定義されているので再定義しないこと！

リストとリスト型

- 一つの型の値を、順にいくつか並べたもの

  $[式_1, 式_2, 式_3, \ldots, 式_n]$

- 多相デ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) *)
```

  - list型も最初から定義されているので再定義しないこと！

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の一種なので、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[$v_1, v_2, v_3, \ldots, v_n$]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, …, 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. 上の関数mapと第3回の関数composeについて、任意のリストxおよび関数fとgに対し、
   `map (compose f g) x`と
   `(map g (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。