ML 演習 第2回

おおいわ

June 6, 2000

今回の内容

- リスト
- 多相関数
- ユーザ定義データ型
  - レコード型
  - バリアント型
- 多相データ型と多相関数

リスト (1)

- list 型

```
# [true; false];;
- : bool list = [true; false]
# let l1 = [1; 2; 3; 4; 5];;
val l1 : int list = [1; 2; 3; 4; 5]
# 100 :: l1;;
- :  int list = [100; 1; 2; 3; 4; 5]
# l1 @ [6; 7];;
- : int list = [1; 2; 3; 4; 5; 6; 7]
```

リスト (2)

- リストの操作: パターンマッチを使う

```
# let rec sum l = match l with
          [] -> 0
        | hd :: tl -> hd + sum tl
val sum : int list -> int = <fun>
# sum [1; 2; 3; 4; 5];;
- : int = 15
```

多相関数 (1)

普通 α → α と読む

- 例1: 恒等関数

```
# let id x = x;;
id : 'a -> 'a = <fun>
# id 1;;
- : int = 1;;
# id [true; false];;
- : bool list = [true; false]
# id sum;;
- : int list -> int = <fun>
```

多相関数 (2)

- 例2: fst, snd

```
# let fst (x, _) = x;;
val fst : 'a * 'b -> 'a = <fun>
# let snd (_, y) = y;;
val snd : 'a * 'b -> 'b = <fun>
```

cf. _: 何とでもマッチする匿名パターン

多相関数 (3)

- 例3

```
# let rec mk_list n v =
    if n = 0 then []
    else v :: mk_list (n-1) v;;
val mk_list: int -> 'a -> 'a list
# mk_list 3 "1";;
- : string list = ["1"; "1"; "1"]
# mk_list 2 true;;
- : bool list = [true; true]
```

多相関数 (4)

- 例4: map

```
# let rec map f l = match l with
      [] -> []
    | hd :: tl = f hd :: map f tl;;
val map : ('a -> 'b) ->
            'a list -> 'b list = <fun>
# map fib [1; 2; 3; 4; 5];;
- : int list = [1; 1; 2; 3; 5]
```

多相関数 (5)

- 型の明示的な制限

```
# let f1 x = (x, x);;
val f1 : 'a -> 'a * 'a = <fun>
# let f2 (x:int) = (x, x);;
val f2 : int -> int * int = <fun>
# let f3 x = ((x, x):int * int);;
val f3 : int -> int * int = <fun>
# f3 "string";
This expression has type string ...
```

独自データ型の定義

- レコード (record)
  - 複数の値の組の型
- バリアント (variant)
  - 複数の値の種類のうち1つを値とする型

レコード型 (1)

- 例: 複素数の直交座標表示

```
# type complex =
            {re : float; im : float};;
type complex =
      { re : float; im : float; }
# let c1 = {re = 5.0; im = 3.0};;
val c1 : complex
       = {re=5.000000; im=3.000000}
# c1.re;;
- : float = 5.000000
```

レコード型 (2)

- パターンマッチング

```
# let add_comp
        {re=r1; im=i1} {re=r2; im=i2}
    = {re = r1+r2; im = i1+i2};;
val add_comp = complex -> complex
                       -> complex = <fun>
# add_comp c1 c1;;
- : complex
        = {re=10.000000; im=6.000000}
```

バリアント型

- 例1: 整数をノードにもつ木

```
# type itree = Leaf
      | Node of int * itree * itree;;
  type itree = Leaf
        | Node of int * itree * itree
# Leaf;;
- : itree = Leaf
# Node(5,Leaf,Leaf);;
- : itree = Node (5, Leaf, Leaf)
```

バリアント型 (2)

- 木のノードの値の合計を求める関数

```
# let rec sum_itree t = match t with
    Leaf -> 0  |  Node(a,t1,t2) ->
     a + sum_itree t1 + sum_itree t2;;
val sum_itree : itree -> int = <fun>
# sum_itree Leaf;;
- : int = 0
# sum_itree
  (Node(4,Node(5,Leaf,Leaf),Leaf));;
- : int = 9
```

多相データ型 (1)

- 例2: 要素をノードにもつ木

```
# type 'a tree = Leaf | Node of
                 'a * 'a tree * 'a tree;;
  type 'a tree = Leaf | Node of
                   'a * 'a tree * 'a tree
# Node(5,Leaf,Leaf);;
- : int tree = Node (5, Leaf, Leaf)
# Node("ocaml",Leaf,Leaf);;
- : string tree =
          Node ("ocaml", Leaf, Leaf)
```

多相データ型と多相関数

- tree の深さを計算する関数

```
# let rec depth t = match t with
    Leaf -> 0 | Node(_,t1,t2) ->
    let (d1,d2)=(depth t1,depth t2)
    in 1+(if d1>d2 then d1 else d2);;
val depth : 'a tree -> int = <fun>
# depth(Node(5,Node(4,Leaf,Leaf),Leaf));;
- : int = 2;
```

その他の構文 (1)

- 関数を作る構文 (1)
  - fun(, let) → 多引数関数と1つのパターン

```
# (fun x y -> x + y) 2 3;;
- : int = 5
# let f x y = x + y;;
val f : int -> int -> int = <fun>
# f 2 3;;
- : int = 5
```

その他の構文 (2)

- **関数を作る構文** (2)
  - function: 1**引数だが複数パターン**

```
# let null = function
            [] -> true | _ -> false;;
val null : 'a list -> bool = <fun>
# let rec map f = function
          [] -> []
        | hd::tl -> f hd :: map f tl;;
val map : ('a -> 'b) ->
            'a list -> 'b list = <fun>
```

その他の構文 (3)

- コメント
  - (* と *) の間

```
# 1 + (* this is comment *) 2;;
- : int 3
```

  - 入れ子にできる

```
# 1 + (* 2 + (* 3 + *) 4 + *) 5;;
- : int = 6
```

課題1

- 引数のリストを逆順にしたリストを返す関数 rev: α list → α list を定義せよ。

```
# rev [1; 2; 3; 4; 5];;
- : int list = [5; 4; 3; 2; 1]
```

  - ヒント1: 2引数の補助関数で結果の後ろの方から順番に生成。
  - ヒント2: どうしてもわからなければ @ を使ってでも…。

課題2

- 判定関数とリストを受け取り、元のリストの要素のうち条件を満たす要素だけからなるリストを生成する関数
  filter: (α → bool) → α list → α list
  を定義せよ。

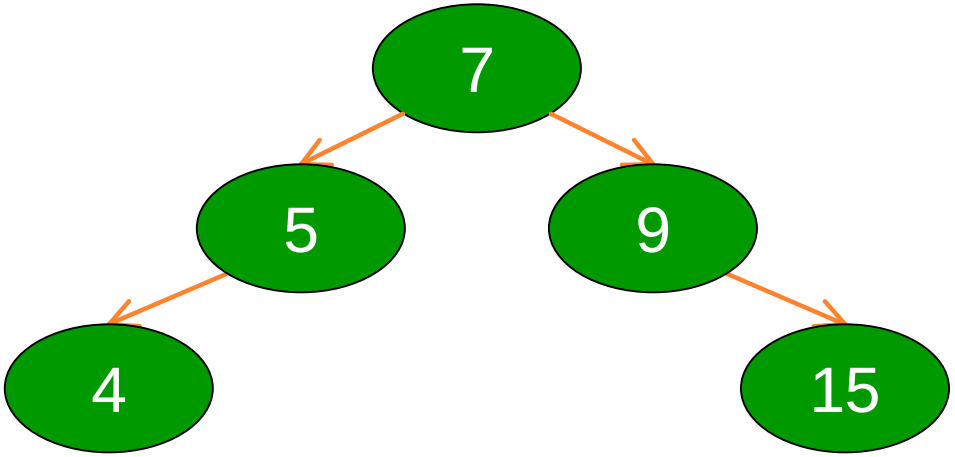
```
# filter odd [1; 2; 3; 4; 5];;
- : int list = [1; 3; 5]
```

課題3

- 木 (α tree) を受け取り深さ優先探索で、全要素を並べたリストを生成する関数 dfs: α tree → α list を定義せよ。

課題3 (例)

```
# dfs(Node(7,
    Node(5,Node(4,Leaf,Leaf),Leaf),
    Node(9,Leaf,Node(15,Leaf,Leaf))));;
- : int list = [7; 5; 4; 9; 15]
```



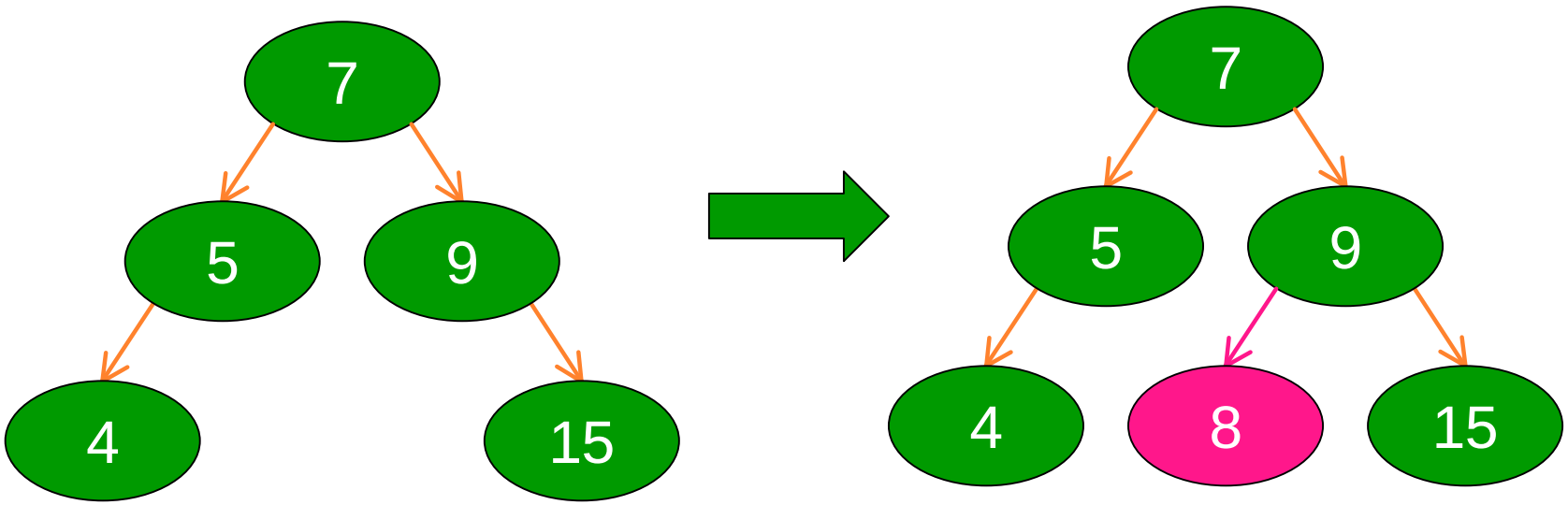
課題4 (optional)

- 課題3 (例) の木は、左の枝に含まれる要素はノードの値より小さく、右の枝に含まれる要素はノードの大きくなっている。このような木を2分探索木という。
- 2分探索木と新たな要素を与えられて、この要素を追加した木を返す関数 add_bst : α → α tree → α tree を定義せよ。

課題4 (例1)

```
# add_bst 8 (Node(7,
   Node(5,Node(4,Leaf,Leaf),Leaf),
   Node(9,Leaf,Node(15,Leaf,Leaf))));;
- : int tree =
Node
 (7, Node (5, Node (4, Leaf, Leaf),
              Leaf),
   Node (9, Node (8, Leaf, Leaf),
              Node (15, Leaf, Leaf)))
```

課題4 (例2)



課題の提出方法

- To: ml-report@yl.is.s.u-tokyo.ac.jp
- Subject: Report 2 *xxxxx* (**学生証番号**)

- **〆切**: 2000年6月19日 月曜日