宣言型プログラム論

200911434 青木大祐

平成 24 年 9 月 25 日

問題 4.1

```
(* 集合の交わり *)
let rec inter l1 l2 =
  match l1 with
    | [] -> []
    | head::rest -> List.filter (fun x -> head = x) l2 @ inter rest l2
;;

inter [3; 1; 2] [2; 3];;

(* ある値と対にする *)
let pair v l =
  List.map (fun x -> (v, x)) l
;;

pair 1 ["A"; "B"; "C"];;

(* 直積 *)
let rec prod l1 l2 =
  match l1 with
    | [] -> []
    | head::rest -> pair head l2 @ prod rest l2
;;

prod [1; 2; 3] ["A"; "B"];;
```

```
        val inter : 'a list -> 'a list -> 'a list = <fun>
#   - : int list = [3; 2]

#         val pair : 'a -> 'b list -> ('a * 'b) list = <fun>
#   - : (int * string) list = [(1, "A"); (1, "B"); (1, "C")]

#             val prod : 'a list -> 'b list -> ('a * 'b) list = <fun>
#   - : (int * string) list =
[(1, "A"); (1, "B"); (2, "A"); (2, "B"); (3, "A"); (3, "B")]
```

問題 4.2

```
(* リストのリストを連結 *)
let flatten l =
  if l = [] then
    []
  else
    List.fold_right (fun x flat -> x @ flat) l [];;

flatten [[1; 2]; []; [3]];;

(* 要素がリストにあるか調べる *)
let exists f l =
  List.fold_left (fun flag x -> if not flag then f x else flag) false l;;

exists (fun x -> x > 1) [0; 3];;
```

```
        val flatten : 'a list list -> 'a list = <fun>
#   - : int list = [1; 2; 3]
#     val exists : ('a -> bool) -> 'a list -> bool = <fun>
#   - : bool = true
#
```

問題 4.3

```
type 'a btree =
    Lf
  | Br of 'a * 'a btree * 'a btree;;

(* 新しい要素を追加 *)
let rec add elt tree =
  match tree with
    | Br (value, left, right) ->
        if elt = value then
          Br (value, left, right)
        else if elt < value then
          Br (value, add elt left, right)
        else
          Br (value, left, add elt right)
    | Lf -> Br (elt, Lf, Lf);;

let tr = add 3 (add 1 (add 2 Lf));;

(* 最小の要素を返す *)
let rec min_elt tree =
  match tree with
    | Br (value, Lf, _) -> value
    | Br (value, left,_) -> min_elt left
    | Lf -> failwith "must not be reached";;

min_elt tr;;

(* 要素を削除 *)
let rec remove elt tree =
  match tree with
    | Br (v, Lf, Lf) ->
        if elt = v then
          Lf
        else
          tree
    | Br (v, left, Lf) ->
        if elt = v then
          left
        else
          Br (v, remove elt left, Lf)
    | Br (v, Lf, right) ->
        if elt = v then
          right
        else
          Br (v, Lf, remove elt right)
    | Br (v, left, right) ->
        if elt = v then
          let root = min_elt right in
          Br (root, left, remove root right)
        else if elt < v then
          Br (v, remove elt left, right)
        else
          Br (v, left, remove elt right)
    | Lf -> Lf;;

remove 3 tr;
```

```
    type 'a btree = Lf | Br of 'a * 'a btree * 'a btree
#       val add : 'a -> 'a btree -> 'a btree = <fun>
#       val tr : int btree = Br (2, Br (1, Lf, Lf), Br (3, Lf, Lf))
#       val min_elt : 'a btree -> 'a = <fun>
#   - : int = 1
#       val remove : 'a -> 'a btree -> 'a btree = <fun>
#   - : int btree = Br (2, Br (1, Lf, Lf), Lf)
#
```

問題 4.4

```
type 'a ftree = FBr of 'a * 'a ftree list;;

(* 木の深さを返す *)
let rec fdepth tree =
  let FBr (_, trlist) = tree in
  (List.fold_left (fun d tree -> max d (fdepth tree)) 0 trlist) + 1;;

fdepth (FBr (1, [FBr (2,[]); FBr (3, [FBr (4, [])])]));;

(* 先順で走査 *)
let rec fpreorder tree =
  let FBr (v, trlist) = tree in
  v :: List.fold_left (fun l tree -> l @ (fpreorder tree)) [] trlist;;

fpreorder (FBr (1, [FBr (2, [FBr (3, []); FBr (4, [])]); FBr (5, [])]));;
```

```
    type 'a ftree = FBr of 'a * 'a ftree list
#         val fdepth : 'a ftree -> int = <fun>
#   - : int = 3
#         val fpreorder : 'a ftree -> 'a list = <fun>
#   - : int list = [1; 2; 3; 4; 5]
#
```

問題 4.5

```
let rec split l =
  match l with
    | [] -> ([], [])
    | [x] -> ([x], [])
    | x1::x2::rest ->
        let (x1rest, x2rest) = split rest in
        (x1::x1rest, x2::x2rest);;

split [1; 2; 3; 4; 5; 6;];;


let rec merge func l1 l2 =
  match (l1, l2) with
    | ([], _) -> l2
    | (_, []) -> l1
    | (head1::rest1, head2::rest2) ->
        if func head1 head2 then
          head1::(merge func rest1 l2)
        else
          head2::(merge func l1 rest2);;


merge (fun x y -> x <= y) [1; 3; 5] [2;3;4];;

let rec msort f l =
  match l with
    | [] -> []
    | [x] -> [x]
    | _ -> let (half1, half2) = split l in
           merge f (msort f half1) (msort f half2);;

msort (fun x y -> x <= y) [5; 3; 1; 7; 6; 4];;
```

```
        val split : 'a list -> 'a list * 'a list = <fun>
#   - : int list * int list = ([1; 3; 5], [2; 4; 6])
#       val merge : ('a -> 'a -> bool) -> 'a list -> 'a list -> 'a list = <fun>
#     - : int list = [1; 2; 3; 3; 4; 5]
#       val msort : ('a -> 'a -> bool) -> 'a list -> 'a list = <fun>
#   - : int list = [1; 3; 4; 5; 6; 7]
#
```