プログラム言語論

亀山幸義

筑波大学 情報科学類

No. 3

目次

注意事項

この資料では、見やすさのため，記号 → を使っているが、これは->という連続した 2 文字のことである。

```
let f = fun x → x + 1;;
==> val f : int → int = <fun>
```

資料 No.3 に掲載する OCaml プログラムは、coins 計算機の以下の場所にある。

`/home/prof/kam/plm/ex3.ml`

このファイルでは、`let _ = ...` という構文を使っているが、あまり気にしなくてよい。また、ファイル中にはエラーを起こす式があるので、`#use "ex3.ml";;` とやってファイル全体を実行しようとしても、最後まで実行されない可能性がある。

目次

いろいろな種類のデータ: 真理値

真理値の操作:

```
let f1 x y = (x mod y) == 0;;
f1 5 3;; (* = false *)
f1 6 3;; (* = true *)

let f2 x y = (f1 x y) && (x != y) && (y > 1);;
f2 9 3;; (* = true *)
f2 9 1;; (* = false *)

let f3 x y = (f1 x y) || (f1 y x);;
f3 2 8;; (* = true *)
```

いろいろな種類のデータ: 整数値

整数データの操作:

```
let g1 x = 3 + x / 2 * 3;;
g1  5 ;;   (* = 9 *)
g1 -5 ;;   (* エラー *)
g1 (-5);;  (* = -3 *)
max_int;;  (* = 整数の最大値 *)
```

割り算は、整数上の (小数点以下を切り捨てる) 割り算である。
「-」の記号は、通常は 2 項演算子 (a-b) を表すので、負の数を表す時は括弧が必要。

いろいろな種類のデータ: 浮動小数点数

浮動小数点数 (実数) の操作:

```
let h1 x y = x *. 3.14 -. y;;
h1 5 3;;   (* エラー *)
h1 2. 3.14;; (* = 3.14 *)

float 2;;          (* = 2. *)
floor 3.14;;       (* = 3. *)
truncate 3.14;;    (* = 3 *)
```

浮動小数点数と整数は異なる種類なので、まぜて使えない。
浮動小数点数に対する演算名には「.」がつく。

いろいろな種類のデータ: 文字列

```
"ab" ^ "cd" ^ "ef";; (* = "abcdef" *)
string_of_int (3 * 5);; (* = "15" *)
string_of_float (sqrt 3.);;
(* = "1.73205080757" *)

print_endline ("a" ^ (string_of_int 3) ^ "b");;
(* a3b が印刷される。返り値は ()  *)
```

目次

複合的なデータの構成

関数型言語はデータを構成する手段を豊富に持つ。なぜなら、

- 小さな関数を組合せて大きなプログラムを作るスタイルで、
- 関数が返せるもの (返り値) は 1 つだけ。

データの構成方法: 組

組 (tuple) の構成と操作:

```
let mkpair x y = (x,y);;
let get_x (x,y) = x;;
let rotate (x,y) =
  let c = cos (Float.pi /. 4.) in
  let s = sin (Float.pi /. 4.) in
  (x*.c-.y*.s, x*.c+.y*.s);;
rotate (1.5, 2.3);;
  (* 反時計回りに π/4 回転した点 *)
let dist (x,y) = sqrt (x*.x +. y*.y);;
dist (1.,2.);; (* 原点までの距離 *)
```

データの構成方法: リスト (1)

組は、決まった個数の集まりで、要素は違う種類でもよい。
リストは、不定個の集まりで、要素は同じ種類。

```
let swap (x,y) = (y,x);;
swap ("pi", 3.14);;
(* = (3.14, "pi") *)

let list1 = [10; 20; 30];;
List.hd list1;; (* = 10 *)
List.tl list1;; (* = [20; 30] *)
3 :: list1;; (* = [3; 10; 20; 30] *)
1.2 :: (2.5 :: []);; (* = [1.2; 2.5] *)
1 :: (2.5 :: []);; (* エラー *)
```

データの構成方法: リスト (2)

以下の 2 つは (内部的には) まったく同じ。

```
[10; 20; 30]
10 :: (20 :: (30 :: []))
```

さらに、:: は右結合するので、括弧を省略してもよい。つまり、上記 2 つは以下のものと同じである。

```
10 :: 20 :: 30 :: []
```

このスライドの知識は、次ページの「パターンマッチ」のときに利用する。(どんなリストも、空リストか、`h::t` のパターンのどちらかである。)

データの構成方法: リスト (3)

リストを使うためには: パターンマッチ

```
let f1 x =
  match x with
| [] → 0
| h::t → h;;

f1 [];; (* = 0 *)
f1 [10; 20];; (* = 10 *)

let rec f2 x =
  match x with
| [] → 0
| h::t → 1 + (f2 t);;

f2 [10; 20; 30];; (* = 3 *)
f2 ["ab"; "cdefg"];; (* = 2 *)
```

データの構成方法: リスト (4)

リストを使うためには: パターンマッチ

```
let rec f3 x =
  match x with
| [] → 1
| h::t → h * (f3 t);;
f3 [3; 5; 2];; (* = 30 *)

let rec f4 x y =
  match x with
| [] → y
| h::t → h :: (f4 t y);;
f4 [3; 5] [−1; 4];; (* = [3; 5; −1; 4] *)
```

データの構成方法: リスト (5)

より一般的なパターンマッチ (この授業ではあまり使わない):

```
let g1 x =
  match x with
| [] → 0
| 1::t → 10
| 2::(5::t) → 20
| _ → 30  ;;

g1 [1; 2];;      (* = 10 *)
g1 [2; 2; 2];;   (* = 30 *)
g1 [2; 5; 3];;   (* = 20 *)
g1 [3; 2];;      (* = 30 *)
```

データの構成方法: リスト (6)

リストの要素の 2 乗和:

```
let rec sqrsum lst =
  match lst with
| [] → 0
| h::t → h * h + sqrsum t;;

sqrsum [10; −1; 5];;      (* = 126 *)
```

データの構成方法: リスト (7)

リストの最大値:

```
let rec maxlist lst =
  match lst with
| [] → min_int
| h::t → let m = maxlist t in
           max h m ;;
maxlist [];;   (* = −4611686018427387904 *)
maxlist [10; −100; 5];;     (* = 10 *)
```

末尾再帰で同じものを書くと:

```
let rec maxlist_sub lst n =
  match lst with
| [] → n
| h::t →
   maxlist_sub t (max h n) ;;
let maxlist lst =
  match lst with
| [] → min_int
| h::t → maxlist_sub t h ;;
```

データの構成方法: リスト (8)

リストに関する再帰関数をもっと楽に書こう。

```
let list1 = [10; 20; 30];;
let g1 x = x + 2;;
let g2 x = sqrt (float x);;

List.map g1 list1;;
(* = [12; 22; 32] *)
List.map g2 list1;;
(* = [3.1...; 4.4...; 5.4...] *)

let list2 = [3.14; 5.8; -2.5];;
let g3 x = (truncate x) * 2;;
List.map g3 list2;;
(* = [6; 10; -4] *)

List.map g3 list1;; (* エラー *)
List.map g2 list2;; (* エラー *)
```

データの構成方法: リスト (9)

リストに関する再帰関数をもっと楽に書こう。

```
let list1 = [10; 20; 30];;
let h1 x y = x + y * 2;;
let h2 x y = x * 3 + y;;
List.fold_left h1 5 list1;;
(* = h1(h1(h1(5,10),20),30) = 125 *)
List.fold_left h2 7 list1;;
(* = h2(h2(h2(7,10),20),30) = 369 *)
```

リストは基本的で重要なデータ構造:

- 数値計算では配列，記号計算ではリスト.
- リスト処理の基本は，パターンマッチと再帰.
- リスト処理を一気に記述する関数群: map,fold など

複数の引数を持つ関数

複数の引数を持つ関数の書き方は 2 つある:

```
(* 化された書き方 curry *)
let f1 x y = x * 2 + y;;
f1 3 5;; (* = 11 *)

(* 化されていない書き方 curry *)
let f2 (x,y) = x * 2 + y;;
f2 (3,5);; (* = 11 *)
```

まぜると危険:

```
f1 (3,5);; (* エラー *)
f2 3 5;;   (* エラー *)
```

OCaml では通常、f1 の書き方 (curry 化された方) を使う。

第 2 週演習問題 (前半)

以下のものを，OCaml 言語で実装せよ.

- 関数 $f1$: 整数のリストと整数 n を引数として、リストの n 番目の要素を返す。(リストの長さが n 未満のときは、0 を返す。)
- 関数 $f2$: 整数のリスト 2 つを引数として、それらを「ベクトル」と見なしたときの「内積」を返す。例: `f2 [1;2;3] [5;-3;3] = 8`
  2 つのリストの長さが異なるときは、短い方に合わせる。例: `f2 [1;2;3] [5;-3;3;4,1] = 8`
- 関数 $f3$: 整数のリストに対して、その平均値を四捨五入した整数値を返す。(長さ 0 のリストに対しては、*min_int* を返す。与えられた整数が偶数かどうか判定するためには、(n mod 2 = 0) とするとよい。)
- 関数 $f4$: 整数のリストに対して、その中央値 (median) を四捨五入した整数値を返す。(長さ 0 のリストに対しては、*min_int* を返す。)
  例: `f4 [100;0;103;101;102] = 101`, `f4 [2;5] = 4`