関数型言語によるアルゴリズム入門

Jacques Garrigue, 2005 年 10 月 12 日

1 Objective Caml 入門

起動

```
$ ocaml
        Objective Caml version 3.08.3
```

値と変数の定義・型

```
# let x = 1 ;;                                          (* let は定義 *)
val x : int = 1
# x = 2 ;;
- : bool = false                                        (* '=' だけだと等号になる *)
# let x = "hello" ;;
val x : string = "hello"
# let x = ref 1 ;;                                      (* 変数の定義 *)
val x : int ref = {contents = 1}
# !x ;;                                                 (* 変数の読み出し *)
- : int = 1
# x := 2 ;;                                             (* 変数の書き込み *)
- : unit = ()
# !x ;;
- : int = 2
# let y = 1 ;;
val y : int = 1
```

関数の定義と値の可視範囲

```
# let f (z : int) =
    !x + y + z ;;
val f : int -> int = <fun>                              (* '->' は関数の型を表す *)
# f 0;;
- : int = 3
# x := 5 ;;
- : unit = ()
# f 0;;
- : int = 6                                             (* x の値を変えると結果が変わる *)
# let y = 12 ;;
val y : int = 12
# f 0;;
- : int = 6                                             (* 値を再定義しても影響はない *)
```

多引数の関数

```
# let p (x : int) (y : int) =                                    (* 引数を並べるだけ *)
    2 * x - y * y ;;
val p : int -> int -> int = <fun>                                  (* '->' が増える *)
# p 3 4;;                                                        (* 適用のときも並べるだけ *)
- : int = -10
```

実数と演算子

```
# let pi = 3.1416 ;;
val pi : float = 3.1416
# let twopi = 2. * pi ;;
This expression has type float but is here used with type int
# let twopi = 2. *. pi ;;                                        (* 実数の演算子は整数と違う! *)
val twopi : float = 6.2832
# let twopi = (float 2) *. pi;;                                  (* 整数を実数に変えるのに関数が要る *)
val twopi : float = 6.2832
```

高階関数と型推論

```
# let mean (f : float -> float) (x1 : float) (x2 : float) =
    (f x1 +. f x2) /. 2. ;;
val mean : (float -> float) -> float -> float -> float = <fun>
# fun x -> x ** 3. ;;                                            (* 関数も値である *)
- : float -> float = <fun>
# mean (fun x -> x ** 3.) 3. 4. ;;
- : float = 45.5
# let mean f x1 x2 = (f x1 +. f x2) /. 2. ;;                     (* 型を書かなくても OK *)
val mean : ('a -> float) -> 'a -> 'a -> float = <fun>
```

配列と反復

```
# let arr = [| 2; 3; 5; 7; 11 |] ;;
val arr : int array = [|2; 3; 5; 7; 11|]
# arr.(0);;                                                      (* 添字は 0 から *)
- : int = 2
# let sum (arr : int array) =
    let r = ref 0 in                                             (* 変数を作る *)
    for i = 0 to Array.length arr - 1 do
      r := !r + arr.(i)
    done;
    !r;;                                                         (* 変数の値を返す *)
val sum : int array -> int = <fun>
# sum arr;;
- : int = 28
```

```
# Array.iter;;                                          (* for ループより安全な関数 *)
- : ('a -> unit) -> 'a array -> unit = <fun>
# let sum (arr : int array) =
    let r = ref 0 in
    Array.iter (fun x -> r := !r + x) arr;                    (* しかも短かい! *)
    !r ;;
val sum : int array -> int = <fun>
# let local (x0 : 'a) (f : 'a ref -> unit) =
    let r = ref x0 in f r; !r ;;
val local : 'a -> ('a ref -> unit) -> 'a = <fun>
# let sum (arr : int array) =
    local 0 (fun r -> Array.iter (fun x -> r := !r + x) arr)
  ;;                                                        (* さらに短かい *)
val sum : int array -> int = <fun>
```

型の種類

ここまで出てきた型のまとめ

`int`	整数	$-2 \cdot 10^{30} \sim 2 \cdot 10^{30} - 1$ (多元では 10^{62})
`bool`	真偽値	`true` または `false`
`string`	文字列	C 言語と同じ
t `ref`	変数 (参照型)	t 型の値への参照 (変更可能)
`unit`	単位型	`()` だけ．C 言語の `void` に似ている
`float`	実数	C 言語の `double` と同じ
t `->` u	関数型	t 型から u 型への関数
t_1 `->` ... `->` t_n `->` u	多引数関数型	$t_1, \ldots, t_n$ 型から u 型への関数
`'a`, `'b`,...	型変数	関数適用時に具体的な型が選べる
t `array`	配列	t 型の値を要素とする配列

実習課題 (1 回目)

添付資料の「Emacs で OCaml を使う」を元に環境を整え，以下の課題をこなす.

1. 上の例を入力してみたり，変更したりで，それを理解する．分からないことを聞く.
2. `Array.iter` と同じ型を持ち，同じ動きをする関数 `iter` を定義せよ.
3. f，ϵ，x が与えられたとき，以下の $f'(x)$ を計算する関数 `derive` を定義せよ.

$$f'(x) = \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$$

   期待している型は次のとおり.

```
val derive : (float -> float) -> float -> float -> float
```

   ある関数 `f` が与えらたとき，次のように `f'` を定義すると，`f'` はどんな型になりますか？

```
let f' = derive f (1e-5);;
```

4. 台形式によって関数 `f` の積分を計算する関数 `integ` を定義せよ． 式は

$$Int(f, N, x, x') = \frac{x' - x}{N} \sum_{k=0}^{N-1} \frac{f(x_k) + f(x_{k+1})}{2} \quad \text{where } x_k = \frac{(N-k)x + kx'}{N}$$

   期待している型は次のとおり.

```
val integ : (float -> float) -> int -> float -> float -> float
```

明日までに 感想や質問および (2)-(4) のできた分だけを `computer-lecture-2005-aw4@math.nagoya-u.ac.jp` 宛に送って下さい.