

プログラミング演習B

ML編 第3回

2009/5/12 (コミ)

2009/5/13 (情報・知能)

住井

[http://www.kb.ecei.tohoku.ac.jp/
~sumii/class/proenb2009/ml3/](http://www.kb.ecei.tohoku.ac.jp/~sumii/class/proenb2009/ml3/)

今日のポイント

1. 局所定義
2. 高階関数 (higher-order functions)
整数や浮動小数点数と同じように、
関数も「値」として扱える
3. 多相関数 (polymorphic functions)
「どんな型についても使える」関数

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m1-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。

復習：変数・関数定義

変数定義

val 変数名 = 式

関数定義

fun 関数名 引数名₁ ... 引数名_n = 式

ポイント 1 : 局所定義

- **let 定義₁ in 式₁ end**
 - 定義₁は式₁の中でのみ使える
- **local 定義₁ in 定義₂ end**
 - 定義₁は定義₂の中でのみ使える
- 「その場だけ必要な」定義に用いる
- **let**と**local**は何が違うの? ⇒ **in**の中が違う
 - ◆ **let ... in ... end**は全体として式に、
local ... in ... endは全体として定義になる

例

```
- let val pi = 3.14 in
= pi * 10.0 * 10.0 end ;
val it = 314.0 : real
- local val pi = 3.14 in
= fun area r = pi * r * r end ;
val area = fn : real -> real
- area 10.0 ;
val it = 314.0 : real
- pi ;
stdIn:22.1-22.3 Error: unbound
variable or constructor: pi
```

課題 3. 1

以下の定義や式を順番に入力し、
結果を考察せよ。

1. `val pi = 3.0`
2. `let val pi = 3.14 in
pi * 10.0 * 10.0 end`
3. `local val pi = 3.14 in
fun area r = pi * r * r end`
4. `pi * 10.0 * 10.0`
5. `area 10.0`

ポイント2：高階関数

例題：

「浮動小数点数から浮動小数点数への関数 f を受け取って、それを微分した関数 f' を返す」という関数`diff`を書け。

- 微分は

$$f'(x) = \frac{f(x + 0.001) - f(x)}{0.001}$$

と近似せよ。

解答例

```
fun diff f = (* 関数fを引数として受け取る *)
  let
    fun f' x = (* 関数f'を定義 *)
      (f (x + 0.001) - f x) / 0.001
  in
    f' (* f'を結果として返す *)
  end
```

- このように「関数を引数として受け取る」あるいは「関数を結果として返す」関数を高階関数という

実行例

```
- fun diff f = ... ; (* 前のページと同じ *)  
val diff = fn : (real -> real) -> real ->  
    real  
- val g = diff Math.sin ;  
val g = fn : real -> real  
- g 0.0 ;  
val it = 0.9999998333333 : real  
- g 3.14 ;  
val it = ~0.9999999361387 : real  
- (diff Math.sqrt) 1.0 ; (* 括弧は省略可能 *)  
val it = 0.499875062461 : real
```

引数の型が関数型

返値の型も関数型

課題3.2

1. `Math.sqrt`, `Math.sin`, `Math.cos`, `Math.tan`, `Math.exp`, `Math.ln`などの関数について、先の`diff`を用いて微分を計算し、結果を確認せよ。
2. 浮動小数点数から浮動小数点数への適当な関数を自分で定義して、先の`diff`を用いて微分を計算し、結果を確認せよ。
 - 定義する関数によっては次頁に注意

ちょっと微妙な注意...

- SMLでは、+や*など一部の演算が、整数と浮動小数点数の両方についてオーバーロード（多重定義）されている
- しかし、ユーザが定義した関数はオーバーロードされない
 - 曖昧な場合は、デフォルトで整数が優先される
 - `fun square x = x * x ;`
`val square = fn : int -> int`
 - 浮動小数点数にしたい場合は「式 : 型」などの構文で型を指定する
 - `fun square (x : real) = x * x ;`
`val square = fn : real -> real`

課題 3. 3

整数から整数への関数 f に対し、

$$g(n) = f(n) - f(n-1)$$

なる関数 g のことを f の階差という。

f を引数として受け取り、 f の階差を結果として返す関数`delta`を書け。

課題3.4

整数から整数への関数 f と、非負整数 n を引数として受け取り、 $f(1) + f(2) + f(3) + \dots + f(n)$ を結果として返す関数`sigma`を書け。

- 下のようになれば良い

```
- fun square x = x * x ;  
val square = fn : int -> int  
- sigma square 10 ;  
val it = 385 : int
```

ヒント

- 前回の再帰関数sumを参照
- たとえば次のような形で書ける

```
fun sigma f n =  
  if n = 0 then 0 else  
    (* f(1)+f(2)+f(3)+...+f(n-1)  
       を求め、それにf(n)を加える *)
```

- 他の形で書いてもOK

課題3.5 (optional)

浮動小数点数から浮動小数点数への関数fを受け取って、fを0.0から1.0まで積分した結果を返す関数integralを書け。

- 積分は

$$\int_0^1 f(x)dx = \{f(0) + f(0.001) + f(0.002) + \dots + f(0.999)\} \times 0.001$$

と近似せよ。

- ヒント：「浮動小数点数xを受け取って、 $f(x) + f(x + 0.001) + f(x + 0.002) + \dots + f(0.999)$ を返す」再帰関数gを局所定義し、 $(g\ 0.0) * 0.001$ を返す。
- integral Math.expの値が1.72ぐらいになれば良い。

課題3.5の注意

- 浮動小数点数には誤差がありうるので、**=で比較してはいけない**。誤差も考慮して、<=などで比較すること。

- 0.1 + 0.1 + 0.1 = 0.3 ;

```
stdIn:17.1-17.22 Error: operator and  
operand don't agree [equality type  
required]
```

```
operator domain: ''Z * ''Z
```

```
operand:          real * real
```

```
in expression:
```

```
0.1 + 0.1 + 0.1 = 0.3
```

- 0.1 + 0.1 + 0.1 > 0.3 ;

```
val it = true : bool
```

ポイント3：多相関数

例題：

「関数 f と関数 g を受け取って、 f と g の合成関数を返す」という関数`compose`を書け。

解答例

```
- fun compose f g =  
=   let fun h x = g (f x) in  
=   h end ;
```

```
val compose =  
  fn : ('a -> 'b)    fの型  
      -> ('b -> 'c)    gの型  
      -> 'a -> 'c      hの型
```

'a, 'b, 'cは「何でも良い」 (型変数)

実行例

```
- (compose Math.exp Math.ln) 1.23 ;  
val it = 1.23 : real  
- fun square x = x * x ;  
val square = fn : int -> int  
- (compose square square) 10 ;  
val it = 10000 : int
```

このcomposeのように「どんな型についても使える」関数を多相関数という。

課題 3. 6

以下の関数は多相関数である。どのような型を持つか確認して考察せよ。また、実際に様々な型で使ってみよ。

1. `fun id x = x`
2. `fun first x y = x`
3. `fun second x y = y`
4. `fun twice f x = f (f x)`

課題3.7

課題3.4の関数sigmaと、課題3.3の関数deltaを、前出のcomposeで合成したら、どのような関数になるか。いくつかの例を実際に試して確認せよ。

- 合成の順番に注意すること

課題 3. 8 (optional)

1. 「関数 f と非負整数 n を受け取り、 f を n 回合成した関数を返す」という関数 `repeat` を書け。
2. 上述の `repeat` と前出の `diff` を使って、浮動小数点数から浮動小数点数への様々な関数の n 階微分 ($n \geq 2$) を求め、結果を確認せよ。