『プログラム言語論』 演習課題 No.2 の解答例

亀山

課題 1. (MiniML プログラミング)

以下の関数を MiniML で定義せよ。

- 整数上の割り算を行なう関数 div: ただし、整数上の割り算は、たとえば、(div 10 3) = 3 のように、余りを切り捨てて、商の整数部分を取りだすものである。また、割る数は、正の整数であることとする。

  解答例. まず「正の整数を正の整数で割る」場合だけを定義してみる。

```
let rec div1 m =
  let x = fst m in
  let y = snd m in
  if y > x then 0
  else  (div1 (x + (-1)*y, y)) + 1
```

  次に、これを使って、「任意の整数を正の整数で割る」div を定義する。

```
let div m =
  let x = fst m in
  let y = snd m in
  if x+1 > 0 then
     div1 (x,y)
  else
     (div1 (x * (-1),y)) * (-1)
```

  ここで、x+1>0 というのは $x \geq 0$ を MiniML 言語で表したものである。x が負の数のときは、dvi1(-x,y) を計算し、その結果にマイナスの記号をつけたものを返す。div は再帰関数ではないので、rec をつける必要はない。(つけてもよい。)

  以下はテストである。

```
let rec div1 m = ...
in let div m = ...
in ((div (10,3),div (-10,3)), (div (3,5), div (-3,5))) ;;
==>
# Main.run "enshu2.ml";;
Main.run "enshu2.ml";;
miniML interpreter Version 4.3 (2013/05/16):
((3,-3),(0,0))
- : string = "ok"
```

- 与えられた整数が素数であるかどうかを判定する関数 isprime.

  まず、正の整数 x が正の整数 y で割り切れるかどうかを判定する関数 isdivided を定義する。

```
let isdivided m =
  let x = fst m in
  let y = snd m in
  if div(x,y) * y = x then
     true
  else false
```

なお、この関数は、以下のように定義してもよい。

```
let isdivided m =
  let x = fst m in
  let y = snd m in
    div(x,y) * y = x
```

さて、isdivided があれば、「n が素数」とは「n が 2 以上 n-1 以下の全ての数で割り切れない」ということなので比較的容易に実装できる。

```
let rec isprime1 m =
  let x = fst m in
  let y = snd m in
  if x = y then
     true
  else if isdivided(x,y) then
     false
  else
     isprime1 (x, y+1)
in let isprime n =
   isprime1 (n,2)
```

実行例:

```
let rec div1 m =
  ...
in let div m =
  ...
in let isdivided m =
  ...
in let rec isprime1 m =
  ...
in let isprime n =
   isprime1 (n,2)
in
  (isprime 2,
  (isprime 3,
  (isprime 4,
```

```
  (isprime 5,
  (isprime 6,
  (isprime 7,
  (isprime 8,
  (isprime 9,
   isprime 10))))))))
;;
==>
(true,(true,(false,(true,(false,(true,(false,(false,false)))))))
```

- 余力があれば、「n が与えられると、n 番目の素数を返す関数 prime を定義せよ。

  これも補助関数として、prime1 を定義する。prime1 は引数が整数のペア (x,y) で、「x 以上の整数の中で、y 番目の素数」を返す関数である。たとえば、prime1 (4,3) = 11 である。

```
let rec prime1 m =
  let x = fst m in
  let y = snd m in
  if isprime x then
     if y = 1 then x
     else prime1 (x+1,y-1)
  else
     prime1 (x+1,y)
in let prime n =
  prime1 (2,n)
```

課題 2. (MiniML のモード)

MiniML の 0 から 5 までのモードを調べよ。

- 動的束縛か、静的束縛か。
- 値呼びか、名前呼びか、必要呼びか。
- 値呼びの場合は、関数適用 (f a) において、関数部分 f を評価してから引数 a を評価するか、引数 a を評価してら、関数部分 f を評価するか。

答え: テスト方法は、C 言語のときと同様である。
動的束縛か、静的束縛かは、以下のプログラムを走らせれば区別できる。

```
let x = 10 in
let f y = x + y in
let x = 20 in
  f 30  ;;
==>  40  (静的束縛; モード 0,2,5)
==>  50  (動的束縛; モード 1,3,4)
```

値呼び、名前呼び、必要呼びは、 以下のプログラムを走らせれば区別できる。(以下の表示では、関数の返り値は省略した。)

```
let f x = (x,x)  in
let g y = 10 in
 (f (print 10),
  g (print 20)) ;;
==>  (値呼び; モード 2,4)
10
20
==>  (名前呼び; モード 0,1)
10
10
==>  (必要呼び; モード 3,5)
10
```

関数適用において、関数から先に計算するか引数から先に計算するかは、miniML では固定 (モードによって変更できない) が、以下のプログラムで順番を知ることはできる。

```
(let y = (print 1) in
     fun x ->
       let y = (print 2) in
          10)
(let y = (print 3) in
     20) ;;
==>
1
3
2  (値呼びの、すべてのモード)
```

つまり、1. 関数部分、2. 引数, 3. 関数本体の順番で計算されている。

問. また、余力があれば、 OCaml 言語がどうか (動的束縛かどうか、等) を、実例を作って試しなさい。OCaml での print は、`print_string "abc";;` などを用いるとよい。

OCaml で以下のようなテストをすればよい。

```
let print x = print_string (string_of_int x) in
(let _ = (print 1) in
     fun x ->
       let _ = (print 2) in
          10)
(let _ = (print 3) in
     20)
;;
==>
312- : int = 10
```

つまり、1. 引数、2. 関数部分, 3. 関数本体の順番で計算されている。

補足: OCaml が引数を先に計算しているのは、処理系の都合 (スタックにデータを積む順番の都合上、引数を先に積んだ方が都合よいため) である。