『プログラム言語論』 宿題の解答例
(2013/06/13 出題, 亀山)

OCaml で、いくつかのプログラム (項) の型付けを行なうものであり、これは、OCaml で実行してみればよい。

```
% ocaml
        OCaml version 4.00.1

# let f x y = x (x (y + 10));;
val f : (int -> int) -> int -> int = <fun>

# let g x = (fst x) + (snd x) + 10;;
val g : int * int -> int = <fun>

# let i_comb x = x ;;
val i_comb : 'a -> 'a = <fun>

# let k_comb x y = x;;
val k_comb : 'a -> 'b -> 'a = <fun>

# let s_comb x y z = (x z) (y z);;
val s_comb : ('a -> 'b -> 'c) -> ('a -> 'b) -> 'a -> 'c = <fun>

# let b_comb x y z = x (y z);;
val b_comb : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
```

若干の解説 1.

関数 f は、int->int 型の x をもらうと関数を返す、という高階関数である。f が返す関数は、int 型の y をもらうと、int 型を返す、という (普通の) 関数である。

全体として、f は (int->int) -> (int->int) という型になる。なお、f の定義は、MiniML では

```
let f x =
  fun y -> x (x (y + 10))
in ...
```

と書く必要がある。このように書くと、f が高階関数であることが一目瞭然である。

この f は、1 引数をもらって 1 つの値を返す関数であるが、ちょっと違う見方をすると、「2 引数をもらう関数」と見なすこともできる。つまり、

```
let g z = z + 20 in
  f g 30
```

というように書くと (これは (f g) 30 というように括弧付けされるので、) g (g (30 + 10)) という計算がなされ、きちんと意図通り、80 が帰ってくる。

このようにして、関数型言語では、1 引数の高階関数 (返すものが、また関数になっている) により、2 引数関数相当のものを表現することができる。(また、3 引数以上も同様である。) 逆に、2 引数関数を、1 引数の高階関数で表現することもでき、こちらの方向を「カリー化」と言う。

若干の解説 2. 上記のプログラムで、x_comb という名前のものは「コンビネータ」と呼ばれるラムダ式である。たとえば、k_comb は K コンビネータとよばれ、$\lambda x.x$ を表す。コンビネータの理論は、ラムダ計算の理論とともに研究されてきており、なんと、驚くべきことに、上記の S コンビネータと K コンビネータのたった 2 つだけを使って、すべてのラムダ式を表現することができることが知られている。(ただし、自由変数を含むラムダ式の場合は、それと同じだけの変数が必要である。) ラムダ計算が「チューリング完全」であることを思いだすと、なんと、S と K だけで、「チューリング完全」な計算モデル (プログラム言語) が構成できてしまうのである。とんでもなくシンプルかつエレガントな世界である。(とはいえ、自然数や、if-then-else や再帰呼びだしなど、すべてのプログラム言語の機能を S と K だけで表現していたら、膨大な長さとなってしまい、機械語より読みにくくなるので、現実にはそんなことはしない。ここに書いたのは、あくまで「理論的にはそうできる」ということである。)

若干の解説 3. 話がすっかり脱線したが、多相性との関連でいえば、i_comb i_comb というプログラムが型がつくことが、ちょっとした驚きである。

```
let i_comb x = x in
   i_comb i_comb ;;
```

というのは、普通の型つきプログラム言語では、x x というプログラムは、うまく型がつかないからである。(左の x が $A \to B$ という型だとすると、右の x は A 型になり、しかし、$A \to B$ と A は、型として一致するようなことは決してないので、通常は型がつきそうもない。)

なぜ、ML 系言語では、このような不思議なプログラムが型がつくかといえば、鍵は「多相性」である。つまり、i_comb i_comb という式では、左の i_comb は $(\alpha \to \alpha) \to (\alpha \to \alpha)$ という型をもち、右の i_comb は $\alpha \to \alpha$ という型をもつことによって、全体として型が整合している。このように、「使われるところによって、異なる型を持つことができる」というのが、多相性 (特に parametric polymorphism) の強みである。