ML 演習第 1 回 課題

後藤, 飯塚, 前田

平成 19 年 6 月 5 日

課題 4 (Optional)

関数の作成と関数の適用ができれば, それを用いて自然数を表現できることが知られている. この課題では自然数 n を “ f と z を受けとって, f を z に n 回適用した結果を返す関数 ”で表現することにする. 但し, 自然数 0 は `fun f z -> z` で表現することにする.

以下では OCaml の式 e に対して, その式が表現する自然数を [[e]] と表すことにする. この方法では [[`fun f z -> z`]] = 0 となる.

1. [[x]] = n なる表現 x を受けとって, [[x]] = n + 1 なる表現 x を返す関数 succ を定義せよ.
2. [[x]] = n なる表現 x と [[y]] = m なる表現 y を受けとって, [[z]] = $n+m$ なる表現 z を返す関数 plus を定義せよ.
3. [[x]] = n なる表現 x と [[y]] = m なる表現 y を受けとって, [[z]] = $n*m$ なる表現 z を返す関数 mult を定義せよ.
4. [[x]] = n なる表現 x と [[y]] = m なる表現 y を受けとって, [[z]] = n^m なる表現 z を返す関数 exp を定義せよ.

ヒント 以下のような関数を使うとデバッグしやすいだろう.

```
let check p = p (fun x -> x + 1) 0
```

この関数は表現 p を受けとって [[p]] を返す関数である. 自然数 n が “ f と z を受けとって, f を z に n 回適用した結果を返す関数 ”で表現されているわ けだから, f と z として `fun x -> x + 1` と 0 を渡してやると 0 に n 回 1 を加えた結果, すなわち n が返ってくるわけである.

関数 check を使うと, 実行結果は次のようになるはずである.

```
# let zero = fun f z -> z;; (* zero を定義 *)
val zero : 'a -> 'b -> 'b = <fun>
```

```
# check (succ (succ zero));; (* 0 + 1 + 1 *)
- : int = 2
# check (plus (succ zero) (succ (succ zero)));; (* 1 + 2 *)
- : int = 3
# check (mult (succ (succ zero)) (succ (succ zero)));; (* 2 * 2 *)
- : int = 4
# check (exp (succ (succ zero))
(succ (succ (succ zero))));; (* the 3rd power of 2 *)
- : int = 8
```