ソフトウェア技法 第7回 201311367 佐藤良祐 2014年12月4日

課題 7.1

1

任意のリスト xs, ys, 任意の関数 f に対して

```
map f (xs @ ys) = (map f xs) @ (map f ys)
```

が成り立つことを示せ.

```
(1) xs = [] のとき
 map f ([] @ ys) = map f ys
                 = [] @ map f ys  (map の定義)

(2) xs のとき与えられた式が成り立つと仮定して x::xs のときを示す.
 map f ((x::xs) @ ys) = map f (x::(xs @ ys))
                      = f x :: map f (xs @ ys)
                      = f x :: (map f xs) @ (map f ys)
                      = (map f (x::xs)) @ (map f ys)
```

2

```
map f (rev l) = rev (map f l)
```

を証明せよ.

```
(1) l = [] のとき
map f (rev []) = map f [] = [] = rev[] = rev (map f [])

(2) l が成り立つと仮定して l = x :: xs を示す.
map f (rev (x::xs)) = map f (rev xs @ x)  (rev の定義)
                    = map f (rev xs) @ map f [x] (1 の証明より)
                    = map f (rev xs) @ [f x]     (map を適用した)
                    = rev (map f xs) @ [f x]     (仮定から)
                    = rev (f x :: (map f xs))    (rev の定義から)
                    = rev (map f (x::xs))        (map の定義から)
以上より成り立つことが示された
```

3

任意のリスト xs, ys, zs に対して

```
revapp (xs @ ys) zs = revapp ys (revapp xs zs)
```

が成り立つことを証明せよ.

```
(1) xs = [] のとき
revapp ([] @ ys) zs = revapp ys zs
                    = revapp ys ([] :: zs)
                    = revapp ys (revapp xs zs)

(2) xsのとき成り立つと仮定してx::xsが成り立つことを示す.
revapp ((x::xs)@ys) zs = revapp (xs @ ys) (x::zs)
                       = revapp ys (revapp xs (x::zs))
                       = revapp ys (revapp (x::xs zs))
```

4

任意の木tに対して

```
size (reflect t) = size t
```

を示せ.

```
(1) P(Lf)が成り立つとする.
  size (reflect Lf) = size (Lf) = 0
(2) t1, t2の場合を仮定してt = Br(v, t1, t2)が成り立つ時を示す.
  size (reflect t) = size (reflect Br(v, t1, t2))
                   = size (Br(v, reflect t2, reflect t1))
                   = 1 + size t2 + size t1
                   = size t
```

課題 7.2

```ocaml
(* 例外の定義 *)
exception Zero;;

(* 整数リストの積を求める ただし 0を見つけたら例外を発生させる *)
let rec preprod lst =
  match lst with
    [] -> 1
  | x::rest -> if x = 0 then raise Zero else x * preprod rest;;

(* 上の関数を用いて整数リストに積を求める関数 *)
let prod lst = try preprod lst with Zero -> 0;;
```

```
# preprod [1;2;3;4];;
- : int = 24
# preprod [0; 1; 2;];;
- : int = 0
# preprod [1;2;3;0];;
Exception: Zero.
# preprod [1;2;3];;
- : int = 6
```