1 集合論の基礎

問題 1.1 上記の性質が，前述の順序対の集合による表現の上で，成り立つことを証明せよ．

証明. 順序対 $(a,\ b)$ を集合 $\{\{a\},\ \{a,\ b\}\}$ と取ることができるから

$$(a,\ b) = (a',\ b') \iff \{\{a\},\ \{a,\ b\}\} = \{\{a'\},\ \{a',\ b'\}\}$$

したがって，$(a,\ b) = (a',\ b')$ のとき

$$\{a\} \in \{\{a'\},\ \{a',\ b'\}\}\ \&\ \{a,\ b\} \in \{\{a'\},\ \{a',\ b'\}\}$$

を得る．$\{a\} \in \{\{a'\},\ \{a',\ b'\}\}$ より

$$\{a\} = \{a'\} \text{ or } \{a\} = \{a',\ b'\}$$

同様に，$\{a,\ b\} \in \{\{a'\},\ \{a',\ b'\}\}$ より

$$\{a,\ b\} = \{a'\} \text{ or } \{a,\ b\} = \{a',\ b'\}$$

(i) $\{a\} = \{a',\ b'\}$ のとき，$a = a' = b'$ である．このとき，$\{a,\ b\} = \{a'\}$ と $\{a,\ b\} = \{a',\ b'\}$ は同等である．$\{a,\ b\} = \{a',\ b'\}$ のときを考えて，$a = b = a' = b'$ を得る．したがって，$a = a'\ \&\ b = b'$ が成り立つ．

(ii) $\{a\} = \{a'\}$ のとき，$a = a'$ である．$\{a,\ b\} = \{a'\}$ のとき，$a = b = a'$ で，$\{a\} = \{a',\ b'\}$ より $\{b\} = \{b,\ b'\}$ となるから，$b = b'$ を得る．したがって，$a = a'\ \&\ b = b'$ が成り立つ．また，$\{a,\ b\} = \{a',\ b'\}$ のとき，$a = b'$ or $b = b'$ である．$a = b'$ のとき，$a = a' = b'$ となるから (i) と同等の議論で $a = a'\ \&\ b = b'$ が成り立つ．$b = b'$ のとき，まさに $a = a'\ \&\ b = b'$ が成り立つ．

(i)，(ii) より

$$(a,\ b) = (a',\ b') \Longrightarrow a = a'\ \&\ b = b'$$

逆に，$a = a'\ \&\ b = b' \Longrightarrow (a,\ b) = (a',\ b')$ が成り立つ．

$$\therefore\quad (a,\ b) = (a',\ b') \iff a = a'\ \&\ b = b'$$

■

問題 1.2 X と Y が集合であるとき，$X \rightharpoonup Y$ と $X \to Y$ はどちらも集合である．

解答. そうですね．

問題 1.3 $R \subseteqq X \times Y$ かつ $S \subseteqq Y \times Z$ かつ $T \subseteqq Z \times W$ であるとする．$T \circ (S \circ R) = (T \circ S) \circ R$ であること（すなわち，合成が**結合的** (associative) であること）を確かめよ．また，$R \circ Id_X = Id_Y \circ R = R$ であること（すなわち，恒等関数は合成操作に関する恒等元であること）を確かめよ．

証明. ■

2 入門：操作的意味論

問題 2.1 もしプログラミング言語 ML（OCaml, SML）や Haskell でプログラミングができるようであれば，**IMP** の構文をデータ型として定義せよ．さらに，構文要素 e_0，e_1 に対してその同一性 $e_0 \equiv e_1$ を判定するプログラムを書け．

解答. 純粋関数型言語 Haskell での記述例を示す．データ型とは Bool，Int，Char，Maybe など様々なものがある．例えば Bool 型は

data `Bool = False | True`

と定義されている．`Bool` は型の名前で `False` や `True` は値コンストラクタと呼ばれる．したがって，この型宣言は「`Bool` 型は `True` または `False` の値を取り得る」と読める．また，型名と値コンストラクタ名は大文字から始める．Haskell では **BNF**（Backus-Naur form）で書き下された構文規則をほぼそのまま **data** 型として定義することができる.（非常に便利）

IMP は定義は p.17 の通り．それを **data** 型を用いて定義すると次のようになる．

Listing 1: IMP の構文をデータ型として定義

```haskell
type N = Int

type Loc = String

data Aexp = Const N        -- n
          | Var Loc        -- X
          | Add Aexp Aexp  -- a0 + a1
          | Sub Aexp Aexp  -- a0 - a1
          | Mul Aexp Aexp  -- a0 * a1

data Bexp = Tru            -- true
          | Fal            -- false
          | Equ Aexp Aexp  -- a0 = a1
          | Le Aexp Aexp   -- a0 <= a1
          | Not Bexp       -- not b
          | And Bexp Bexp  -- b0 /\ b1
          | Or Bexp Bexp   -- b0 \/ b1

data Com = Skip              -- skip
         | Assign Loc Aexp   -- X := a
         | Seq Com Com       -- c0;c1
         | If Bexp Com Com   -- if b then c0 else c1
         | While Bexp Com    -- while b do c
```

構文要素の同一性を判定するプログラムは，次のように再帰的に定義することができる．

Listing 2: 構文要素の同一性を判断する関数

```haskell
aexp_equal :: Aexp -> Aexp -> Bool
aexp_equal (Const n1) (Const n2) = n1 == n2
aexp_equal (Var v1) (Var v2) = v1 == v2
aexp_equal (Add a11 a12) (Add a21 a22) = aexp_equal a11 a21 && aexp_equal a12 a22
aexp_equal (Sub a11 a12) (Sub a21 a22) = aexp_equal a11 a21 && aexp_equal a12 a22
aexp_equal (Mul a11 a12) (Mul a21 a22) = aexp_equal a11 a21 && aexp_equal a12 a22
aexp_equal _ _ = False

bexp_equal :: Bexp -> Bexp -> Bool
bexp_equal Tru Tru = True
bexp_equal Fal Fal = False
bexp_equal (Equ a11 a12) (Equ a21 a22) = aexp_equal a11 a21 && aexp_equal a12 a22
bexp_equal (Le a11 a12) (Le a21 a22) = aexp_equal a11 a21 && aexp_equal a12 a22
bexp_equal (Not b1) (Not b2) = bexp_equal b1 b2
bexp_equal (And b11 b12) (And b21 b22) = bexp_equal b11 b21 && bexp_equal b21 b22
bexp_equal (Or b11 b12) (Or b21 b22) = bexp_equal b11 b21 && bexp_equal b21 b22
bexp_equal _ _ =False
```

```
18
19 com_equal :: Com -> Com -> Bool
20 com_equal Skip Skip = True
21 com_equal (Assign l1 a1) (Assign l2 a2) = aexp_equal (Var l1)  (Var l2) && aexp_equal a1 a2
22 com_equal (Seq c11 c12) (Seq c21 c22) = com_equal c11 c21 && com_equal c12 c22
23 com_equal (If b1 c11 c12) (If b2 c21 c22) = bexp_equal b1 b2 && com_equal c11 c12 && com_equal c21 c22
24 com_equal (While b1 c1) (While b2 c2) = bexp_equal b1 b2 && com_equal c1 c2
25 com_equal _ _ = False
```

Listing 2 の 1 行目の

```
aexp_equal :: Aexp -> Aexp -> Bool
```

は，関数 `aexp_equal` の型宣言である．`Aexp` 型と `Aexp` 型を引数として `Bool` 型を返す関数である．Haskell には型推論という機能があり，関数の型宣言は書かなくてもよい．一方で，同じ純粋関数型言語の Idris は型宣言を書かなくてはならない.（余談）実行例を以下に示す.

```
ghci> com_equal (Assign "X" (Const 5)) (Assign "X" (Const 5))
True
ghci> aexp_equal (Add (Const 3) (Const 5)) (Add (Const 5) (Const 3))
False
```

問題 2.2 算術式の評価規則をプログラミングせよ．問題 2.1 で導入したデータ型を使うとよい．

解答.