問題 7

次のように定義されたデータ型 Nat (自然数) において、商と余りを求める関数 divide と累乗を計算する関数 expt を定義しなさい。

解答及び解説

まずは、型 Nat の割り算を考える。足し算 plus を考えたときは、型 Nat のことばを使って、その規則を次のように表現した。

$$\begin{cases} \text{自然数0とmを足すと、結果はmである。} \\ \text{自然数nとmの足し算は、自然数「nの前の数」とmを足した結果の次の数に等しい。} \end{cases}$$

同様に、割り算の規則を表現してみると、

自然数 n を m で割った商は、$n - m$ を m で割った商の次の数に等しい。

となる。このままプログラムにすれば、

```
div n m = Succ (div (minus n m) m)
```

となるが、自然数での引き算 minus は、部分関数であったことを思い出そう。

このことを考えると divide は、次のように定義することができる。

```
divide :: Nat -> Nat -> (Nat, Nat)
divide n m = divide' (minus n m) m n


divide' :: Maybe Nat -> Nat -> Nat -> (Nat, Nat)
divide' Nothing m n = (Zero, n)
divide' (Just l) m n = let (q, r) = divide' (minus l m) m l in (Succ q, r)
```

ところで、divide' の引き数は 2 つではなく、3 つになっている。それは、minus n m の結果が Nothing となるとき、つまり n から m が引けないときは、商は Zero であり、余りは n になる。そこで、divide' には、割る数 m と割られる数 minus n m の他に、余りとなる n を引き数として与える必要がある。

ところで、minus が Nothing となるのは、「$n < m$」のときである。なので、型 Nat を Ord クラスにしておけば、次のようなプログラムでも良い。

```
divide :: Nat -> Nat -> (Nat, Nat)
divide n m | n < m     = (Zero, n)
           | otherwise = let (q, r) = divide (minus n m) m in (Succ q, r)
```

次に、 自然数の累乗 (expt) を考える。累乗の規則は、次のようになる。

$$\begin{cases} \text{自然数nの0乗は、1である。} \\ \text{自然数nのm乗は、自然数nの「mの前の数」乗に、nを掛けた数に等しい。} \end{cases}$$

「m の前の数」を求める関数は無いため、プリント No.7 でやっているように、パターンを使って「の前の数」を求めるようにする。

```
expt :: Nat -> Nat -> Nat
expt n Zero = Succ Zero
expt n (Succ m) = times n (expt n m)
```