第 11 章 単純な拡張（後半）

> テキストの解答要約はこんな感じで引用表現にする（引用じゃないけど）

演習 11.9.1. $[\star\star]$

- true を inl unit と定義する
- false を inr unit と定義する
- if t_1 then t_2 else t_3 を case t_1 of inl unit $\Rightarrow$ t_2 | inr unit $\Rightarrow$ t_3 と定義する

> Bool $\stackrel{\text{def}}{=}$ Unit + Unit を忘れずに。あとcase文ではunit値に限定せずに、任意の値を受け取る。

演習 11.11.1. $[\star\star]$

equal

```
equal = fix ff;
equal : Nat → Nat → Bool

ff = λie:Nat → Nat → Bool.
       λx:Nat.
         λy:Nat.
           if iszero x
             then if iszero y
               then true
               else false
             else if iszero y
               then false
               else ie (pred x) (pred y);
ff : (Nat → Nat → Bool) → Nat → Nat → Bool
```

plus

```
plus = fix gg;
plus : Nat → Nat → Nat

gg = λie:Nat → Nat → Nat.
       λx:Nat.
         λy:Nat.
           if iszero y
             then x
             else ie (succ x) (pred y);
gg : (Nat → Nat → Nat) → Nat → Nat → Nat
```

times

```
times = λx:Nat.
          λy:Nat.
            fix hh (plus x) 0 y;
times : Nat → Nat → Nat

hh = λie:(Nat → Nat) → Nat → Nat → Nat.
       λf:Nat → Nat.
         λx:Nat.
           λy:Nat.
             if iszero y
               then x
               else ie f (f x) (pred y);
hh : ((Nat → Nat) → Nat → Nat → Nat) → (Nat → Nat) → Nat → Nat → Nat
```

factorial

```
factorial = λx:Nat.
              λy:Nat.
                fix ii (times x) (succ 0) y;
factorial : Nat → Nat → Nat

ii = λie:(Nat → Nat) → Nat → Nat → Nat.
       λf:Nat → Nat.
         λx:Nat.
           λy:Nat.
             if iszero y
               then x
               else ie f (f x) (pred y);
ii : ((Nat → Nat) → Nat → Nat → Nat) → (Nat → Nat) → Nat → Nat → Nat
```

> equal以外はもっと簡単に書ける。例えばtimesなら、3 * 4 = 4 + ((3 - 1) * 4) = 4 + (4 + (2 - 1) * 4)) = 4 + (4 + (4 + (1 - 1) * 4))) = 4 + (4 + (4 + 0))) = 12
>
> 私の方法だと、3 * 4 = ie(3 + 0, 4 - 1) = ie(3 + 3 + 0, 3 - 1) = ie(3 + 3 + 3 + 0, 2 - 1) = ie(3 + 3 + 3 + 3 + 0, 1 - 1) = 12
>
> factorialって累乗じゃなくて階乗じゃん…

演習 11.11.2. [⋆]

```
letrec plus : Nat → Nat → Nat = λx:Nat. λy:Nat.
  if iszero x then 0 else succ (plus (pred x) y);

letrec times : Nat → Nat → Nat = λx:Nat. λy:Nat.
  if iszero x then 1 else plus y (times (pred x) y);

letrec factorial : Nat → Nat = λx:Nat.
  if iszero x then 1 else times x (factorial (pred x));
```

> 実際に使うときは letrec .. in という形式にしないとエラー。

演習 11.12.1. [⋆⋆⋆]

- 全部は大変なのでリストに関する場合分けだけにします…

進行

- 正しく型付けされた項は値であるか評価できる。

項 t の型付け導出の最後で場合分け

- T-Nil のとき、項 t は nil[T1] である。nil[T1] に適用できる評価規則はないが、値なのでOK
- T-Cons のとき、項 t は cons[T1] t1 t2 という形である。部分項 t1 t2 について場合分け（値であるか、評価できる）
  - t1 t2 が両方値→t も値なのでOK
  - t1 が評価可能、t2はどちらでも→E-Cons1 が適用できる
  - t1 が値、t2 が評価可能→E-Cons2 が適用できる
- T-Isnil のとき、項 t は isnil[T11] t1 という形である。部分項 t1 について場合分け
  - t1 が値 nil[T]→E-IsnilNil が適用できる
  - t1 が値 cons[T] v1 v2→E-IsnilCons が適用できる
  - t1 が評価可能→E-Isnil が適用できる
- T-Head のとき、項 t は head[T11] t1 という形である。部分項 t1 について場合分け
  - t1 が値 nil[T]→E-…あれ？
    - ということで、**進行は成り立たない**
    - 多分T-Tailでも成り立たないケースがでてくる

保存

- 正しく型付けされた項が評価されるとき、その項は元の型で正しく型付けされる。

項 t の型付け導出の最後で場合分け

- T-Nil のとき、項 t は nil[T1] である。評価されないのでOK
- T-Cons のとき、項 t は cons[T11] t1 t2 という形である。
  - t1 が評価されるとき→E-Cons1→cons[T11] t1' t2→T-Consで型付け
  - t2 が評価されるとき→E-Cons2→cons[T11] v1 t2→T-Consで型付け
- T-Isnilのとき、項 t は isnil[T11] t1 という形である。
  - t1 が評価されるとき→E-Isnil→isnil[T11] t1'→T-Isnilで型付け
  - T-Head, T-Tailも同様

演習 11.12.2. [⋆⋆]

T-Nil以外は前提部分の型付けがあるので問題ない。

T-Nilは項に型がないと型付けできない（Listの要素の型がわからない）。

（HaskellでもよくEmptyの型がわからないみたいなの出るよね…）