型システム入門メモ

maton

第 18 章　事例：命令的オブジェクト

18.6　単純なクラス

演習 18.6.1. [推奨，⋆⋆]　fullref 検査器は使ってません ...。あと、「λ」をうまく表示できなかったので、代わりに「\」と書いてます。

```
decCounterClass =
  \r:CounterRep.
    let super = resetCounterClass r in
      {get   = super.get,
       inc   = super.inc,
       reset = super.reset,
       dec   = \_:Unit. r.x:=pred(!(r.x))};
```

演習 18.6.2. [⋆⋆ ↛]

新しい構文形式 (t を項であるとする)

$$\begin{aligned} t ::= & \ ... \\ & \ t \text{ with } t \end{aligned}$$

新しい評価規則 (l, r をレコードラベル , v, w を値であるとする)

(1)

$$\left\{l_i = v_i \ ^{i \in 1..n}\right\} \text{ with } \{\} \longrightarrow \left\{l_i = v_i \ ^{i \in 1..n}\right\}$$

(2)

$$\begin{aligned} & \left\{l_i = v_i \ ^{i \in 1..j-1}, l_j = v_j, l_k = v_k \ ^{k \in j+1..n}\right\} \text{ with } \left\{l_j = {v'}_j, r_h = w_h \ ^{h \in 1..m}\right\} \\ \longrightarrow & \left\{l_i = v_i \ ^{i \in 1..j-1}, l_j = {v'}_j, l_k = v_k \ ^{k \in j+1..n}\right\} \text{ with } \left\{r_h = w_h \ ^{h \in 1..m}\right\} \end{aligned}$$

(3)

$$\left\{l_i = v_i \ ^{i\in 1..n}\right\} \text{ with } \left\{r_j = w_j, r_h = w_h \ ^{h\in 1..m}\right\}$$
$$\longrightarrow \left\{l_i = v_i \ ^{i\in 1..n}, r_j = w_j\right\} \text{ with } \left\{r_h = w_h \ ^{h\in 1..m}\right\}$$

(4)

$$\frac{t_1 \longrightarrow t_1'}{t_1 \text{ with } t_2 \longrightarrow t_1' \text{ with } t_2}$$

(5)

$$\frac{t_2 \longrightarrow t_2'}{v_1 \text{ with } t_2 \longrightarrow v_1 \text{ with } t_2'}$$

新しい型付け規則 ($T_1 \vee T_2$ は p.161 で見た合併型)

$$\frac{\Gamma \vdash t_1 : T_1 \qquad \Gamma \vdash t_2 : T_2}{\Gamma \vdash t_1 \text{ with } t_2 : T_1 \vee T_2}$$

評価規則の基本的な考え方は、マージソートにおけるマージに由来している。すなわち、with の右辺のレコードを先頭から見ていき、右辺のレコードの先頭要素が

(1) 空ならば、マージ完了である

(2) 左辺のレコードに含まれていれば、オーバーライドする

(3) 左辺のレコードに含まれていなければ、左辺に加える

という操作を、それぞれ 1 ステップで実行する。(4), (5) の評価規則は、with の両辺が値になっていない場合の評価順序を定めている。

18.7 インスタンス変数の付加

演習 18.7.1. [推奨，⋆⋆] やはり fullref 検査器は使ってません。

```
DoubleBackupCounter =
  {get:Unit->Nat, inc:Unit->Unit,
   reset:Unit->Unit,backup:Unit->Unit,
   reset2:Unit->Unit,backup2:Unit->Unit};

DoubleBackupCounterRep =
```

```
  {x: Ref Nat, b: Ref Nat, b2: Ref Nat};

doubleBackupCounterClass =
  \r:DoubleBackupCounterRep.
    let super = backupCounterClass r in
      {get     = super.get,
       inc     = super.inc,
       reset   = \_:Unit. r.x:=!(r.b),
       backup  = \_:Unit. r.b:=!(r.x),
       reset2  = \_:Unit. r.x:=!(r.b2),
       backup2 = \_:Unit. r.b2:=!(r.x)};
```

このとき型は、

- doubleBackupCounterClass : DoubleBackupCounterRep -> DoubleBackupCounter

のようになる。関数型は反変なので、doubleBackupCounterClass は backupCounterClass の部分型になっていない ... ？

なお、演習 18.6.2. で定めた with 構文を使うと、doubleBackupCounterClass は以下のようになる。

```
doubleBackupCounterClass =
  \r:DoubleBackupCounterRep.
    let super = backupCounterClass r in
      super with {reset2  = \_:Unit. r.x:=!(r.b2),
                  backup2 = \_:Unit. r.b2:=!(r.x)};
```

18.11 オープンな再帰と評価順序

演習 18.11.1. [推奨，⋆⋆⋆] 以下、解答では断りなく with 構文を用いる。

(1) get の呼び出しを数えられるようにする。カウンタは共有で。

```
instrCounterClass =
  \r:InstrConterRep.
```

```
  \self:Unit -> InstrCounter.
    \_:Unit.
      let super = setCounterClass r self unit in
        super with
        {get = \i:Nat. (r.a:=succ(!(r.a)); super.get i),
         set = \i:Nat. (r.a:=succ(!(r.a)); super.set i),
         accesses = \_:Unit. !(r.a)};
```

(2) reset を持つサブクラスを定義。InstrConterRep を使いまわす。(ResetInstrCounter の型は省略。)

```
resetInstrCounterClass =
  \r:InstrConterRep.
  \self:Unit -> ResetInstrCounter.
    \_:Unit.
      let super = instrCounterClass r self unit in
        super with {reset = \_:Unit. r.x:=1};
```

(3) backup を持つサブクラスを定義。(BackupInstrCounter の型は省略。)

```
BackupInstrCounterRep =
  {x: Ref Nat, a: Ref Nat, b: Ref Nat};

backupInstrCounterClass =
  \r:BackupInstrConterRep.
  \self:Unit -> BackupInstrCounter.
    \_:Unit.
      let super = instrCounterClass r self unit in
        super with
        {reset = \_:Unit. r.x:=!(r.b),
         backup = \_:Unit. r.b:=!(r.x)};
```

18.13 要点のまとめ

演習 18.13.1. [⋆⋆⋆] オブジェクト同一性を実現するには、new 関数が呼び出される度に異

なるオブジェクト ID をオブジェクトに付与するのが良いだろう。オブジェクト ID の表現についてはここでは議論しない[*1]。あらゆる new 関数は、内部に ID を発行するためのストアを持ち、new 関数を呼び出す度にオブジェクトに ID を付与してストアを更新する。

この機構を実装するには、すべてのクラスが ID のストアと ID の更新メソッドを持つクラスを継承するようにすればよい。これは Java における `Object` クラスに類似するアプローチと言える。

1 セキュリティを気にしなければ通し番号でいいだろう