VDM++ 言語入門セミナー　演習問題

(Revision : 0.3 – 2019 年 1 月 23 日)

佐原　伸

概要

「対象を如何にモデル化するか？」のモデル作成例。

目次

1 CRUD_MAP

写像の CRUD ライブラリー。

```
class
CRUD_MAP
functions
public
  CreateMap1[@Dom, @Rng] : map @Dom to @Rng * @Dom * @Rng -> map @Dom to @Rng
  CreateMap1 (m, d, e) ==
    m munion {d |-> e}
  pre  d not in set dom m
  post RESULT = m munion {d |-> e} ;
public
  ReadMapRestricToDom[@Dom, @Rng] : map @Dom to @Rng * set of @Dom -> map @Dom to @Rng
  ReadMapRestricToDom (m, sd) ==
    if sd = {}
    then m
    else sd <:  m;
public
  ReadMapRestricToRng[@Dom, @Rng] : map @Dom to @Rng * set of @Rng -> map @Dom to @Rng
  ReadMapRestricToRng (m, sr) ==
    if sr = {}
    then m
    else m :> sr;
public
  UpdateMap[@Dom, @Rng] : map @Dom to @Rng * @Dom * @Rng -> map @Dom to @Rng
  UpdateMap (m, d, e) ==
    m ++ {d |-> e};
public
  DeleteMapRestricToDom[@Dom, @Rng] : map @Dom to @Rng * set of @Dom -> map @Dom to @Rng
  DeleteMapRestricToDom (m, sd) ==
    sd <-:  m;
public
  DeleteMapRestricToRng[@Dom, @Rng] : map @Dom to @Rng * set of @Rng -> map @Dom to @Rng
  DeleteMapRestricToRng (m, sr) ==
    m :-> sr
end
CRUD_MAP
```

.....

2　実行可能な仕様 (図書館 1)

実行不可能な仕様で、事後条件・事前条件・不変条件などを整理した時点で、構文チェック・型チェックが可能なので、静的な検証は行うことができる。

実際のシステム開発では、日本語による仕様の欠陥より少ないとはいえ、VDM++ の実行不可能な陰仕様には多数の欠陥が残る。そこで、実行可能な仕様を作成することにする。

図書館システムの例では、下記のように、関数や操作の本体に、集合と写像を扱う演算子や、内包表現を使うことにより、容易に実行可能な陽仕様とすることができる。

関数や操作の本体の VDM++ による記述は、陰仕様で定義された「仕様」を検証するための記述である。したがって、設計時や実装時にこの記述をそのまま使う必要はない。「仕様」は、あくまでも陰仕様で記述された部分である。

ここで、スーパークラスの図書館 RD は、図書館のドメイン知識を持つクラスであり、後述する。

```
class
図書館 1 is subclass of 図書館 RD
values
public
  v 最大蔵書数 = 10000;
public
  v 最大貸出数 = 3
```

2.1　型定義

```
types
  public 蔵書 = map 蔵書 ID to 本
  inv w 蔵書 == card dom w 蔵書 <= v 最大蔵書数
```

2.2　インスタンス変数定義

```
instance variables
  private i 蔵書 : 蔵書 := { |-> };
  private i 貸出 : 貸出 := { |-> };
  inv 蔵書に存在する (dom merge rng i 貸出,i 蔵書)
```

2.3 操作定義

2.3.1 蔵書を追加する

```
operations
public
  蔵書を追加する : 蔵書 ID * 本 ==> ()
  蔵書を追加する (a 蔵書 ID, a 本) ==
    i 蔵書 := 図書館 RD‘蔵書を追加する (i 蔵書, a 蔵書 ID, a 本);
```

2.3.2 蔵書を削除する

```
public
  蔵書を削除する : set of 蔵書 ID ==> ()
  蔵書を削除する (anID 集合) ==
    i 蔵書 := 図書館 RD‘蔵書を削除する (i 蔵書, anID 集合)
  pre 貸出に存在しない (anID 集合, i 貸出) ;
```

2.3.3 蔵書を削除する

```
public
  蔵書を削除する : 本 ==> ()
  蔵書を削除する (a 本) ==
    i 蔵書 := 図書館 RD‘蔵書を削除する (i 蔵書, a 本)
  pre 貸出に存在しない (a 本, i 貸出) ;
```

2.3.4 題名と著者と分野で本を検索する

ここでは、写像の内包式 で題名と著者と分野のいずれかに一致する蔵書を検索結果として返している。内包式は集合や列に対しても形式は少し異なるが存在し、ある条件を満たした写像、集合、列のデータを得るのによく使う。

```
public
```

```
  本を検索する : ( 題名 | 著者 | 分野) ==> 蔵書
  本を検索する (a 検索キー) ==
    return {id |-> i 蔵書 (id) | id in set dom i 蔵書 &
                    (i 蔵書 (id).f 題名 = a 検索キー or
                     i 蔵書 (id).f 著者 = a 検索キー or
                     a 検索キー in set i 蔵書 (id).f 分野集合)}
  pre  検索キーが空でない (a 検索キー)
  post  forall book in set rng RESULT &
            (book.f 題名 = a 検索キー or
             book.f 著者 = a 検索キー or
             a 検索キー in set book.f 分野集合) ;
```

2.3.5 本を貸す

```
public
  本を貸す : 蔵書 * 利用者 ==> ()
  本を貸す (a 貸出本, a 利用者) ==
    i 貸出 := 図書館 RD‘本を貸す (i 貸出, a 貸出本, a 利用者)
  pre  貸出可能である (a 利用者, a 貸出本, i 貸出, i 蔵書, v 最大貸出数) ;
```

2.3.6 本を返す

```
public
  本を返す : set of 蔵書 ID * 利用者 ==> ()
  本を返す (anID 集合, a 利用者) ==
    i 貸出 := 図書館 RD‘本を返す (i 貸出, anID 集合, a 利用者)
  pre  貸出に存在する (anID 集合, i 貸出) ;
```

2.3.7 インスタンス変数のアクセッサ

以下の操作は、主に回帰テストケースを記述するのに便利になるよう作成した。

```
public
  蔵書を得る : () ==> 蔵書
  蔵書を得る () ==
    return i 蔵書;
public
```

```
  貸出を得る : () ==> 貸出
  貸出を得る () ==
    return i 貸出
..........................................................................................
..........................................................................................
end
図書館 1
..........................................................................................
```

3　図書館要求辞書

図書館のドメイン知識を持つ。

図書館に関する要求辞書の役割も持つ。クラス名に付いている RD は、Requirement Dictionary の略である。

実行不可能な陰仕様を記述した図書館 0 クラスで定義されていた、関数群に対応した関数群を持つ。

```
class
図書館 RD
```

3.1　型定義

```
types
  public 蔵書 = map 蔵書 ID to 本;
public
  本 :: f 題名 : 題名
       f 著者 : 著者
       f 分野集合 : set of 分野;
  public 題名 = seq of char;
  public 著者 = seq of char;
  public 分野 = seq of char;
  public 蔵書 ID = token;
  public 職員 = token;
  public 利用者 = token;
  public 貸出 = inmap 利用者 to 蔵書
```

3.2　関数定義

3.2.1　蔵書を追加する

```
functions
public
  蔵書を追加する : 蔵書 * 蔵書 ID * 本 -> 蔵書
  蔵書を追加する (a 蔵書, a 蔵書 ID, a 本) ==
    CRUD_MAP`CreateMap1[蔵書 ID, 本] (a 蔵書, a 蔵書 ID, a 本);
```

3.2.2 蔵書を削除する

```
public
  蔵書を削除する : 蔵書 * set of 蔵書 ID -> 蔵書
  蔵書を削除する (a 蔵書, anID 集合) ==
    CRUD_MAP`DeleteMapRestricToDom[蔵書 ID, 本] (a 蔵書, anID 集合);
```

3.2.3 蔵書を削除する

```
public
  蔵書を削除する : 蔵書 * 本 -> 蔵書
  蔵書を削除する (a 蔵書, a 本) ==
    CRUD_MAP`DeleteMapRestricToRng[蔵書 ID, 本] (a 蔵書, {a 本 });
```

3.2.4 本を貸す

利用者がすでに本を借りている場合と、初めて借りる場合で、処理が異なる。
すでに借りている場合は、貸出の写像を上書きし、まだ借りていない場合は、貸出の写像に追加する。
初めて借りる場合は、単純に利用者と貸出本の写を、従来の貸出写像に munion 演算子を使って併合する。

```
public
  本を貸す : 貸出 * 蔵書 * 利用者 -> 貸出
  本を貸す (a 貸出, a 貸出本, a 利用者) ==
    if a 利用者 in set dom a 貸出
    then CRUD_MAP`UpdateMap[利用者, 蔵書] (a 貸出, a 利用者, a 貸出 (a 利用者) munion a 貸出本)
    else CRUD_MAP`CreateMap1[利用者, 蔵書] (a 貸出, a 利用者, a 貸出本);
```

3.2.5 本を返す

貸出情報から、返却する蔵書に関するデータを削除する。
貸出が残っている利用者の蔵書は貸出に上書きし、貸出が残っていない利用者の情報は貸出から削除する。

```
public
```

```
  本を返す : 貸出 * set of 蔵書 ID * 利用者 -> 貸出
  本を返す (a 貸出, anID 集合, a 利用者) ==
    let w 利用者への貸出本 new = CRUD_MAP`DeleteMapRestricToDom[蔵書 ID, 本] (a 貸出 (a 利用者
), anID 集合) in
    if w 利用者への貸出本 new = { |-> }
    then CRUD_MAP`DeleteMapRestricToDom[利用者, 蔵書] (a 貸出, {a 利用者 })
    else CRUD_MAP`UpdateMap[利用者, 蔵書] (a 貸出, a 利用者, w 利用者への貸出本 new);
```

3.2.6 蔵書の状態に関する関数

```
public
  蔵書に存在する : set of 蔵書 ID * 蔵書 +> bool
  蔵書に存在する (anID 集合, a 図書館蔵書) ==
    if anID 集合 = {}
    then true
    else  exists id in set anID 集合 & id in set dom a 図書館蔵書;
```

```
public
  貸出に存在する : set of 蔵書 ID * 貸出 +> bool
  貸出に存在する (anID 集合, a 貸出) ==
    let w 蔵書 = merge rng a 貸出 in
     exists id in set anID 集合 & id in set dom w 蔵書;
```

```
public
  貸出に存在する : 本 * 貸出 +> bool
  貸出に存在する (a 本, a 貸出) ==
    let w 蔵書 = merge rng a 貸出 in
    a 本 in set rng w 蔵書;
```

```
public
  貸出に存在しない : set of 蔵書 ID * 貸出 +> bool
  貸出に存在しない (anID 集合, a 貸出) ==
    not 貸出に存在する (anID 集合, a 貸出);
```

```
public
```

```
  貸出に存在しない : 本 * 貸出 +> bool
  貸出に存在しない (a 本, a 貸出) ==
    not 貸出に存在する (a 本, a 貸出);
.....
.....
public
  貸出可能である : 利用者 * 蔵書 * 貸出 * 蔵書 * nat1 +> bool
  貸出可能である (a 利用者, a 貸出本, a 貸出, a 図書館蔵書, a 最大貸出数) ==
    蔵書に存在する (dom a 貸出本, a 図書館蔵書) and
    貸出に存在しない (dom a 貸出本, a 貸出) and
    最大貸出数を超えていない (a 利用者, a 貸出本, a 貸出, a 最大貸出数);
.....
```

3.2.7 最大貸出数を超えていない

```
.....
public
  最大貸出数を超えていない : 利用者 * 蔵書 * 貸出 * nat1 +> bool
  最大貸出数を超えていない (a 利用者, a 貸出本, a 貸出, a 最大貸出数) ==
    if a 利用者 not in set dom a 貸出
    then card dom a 貸出本 <= a 最大貸出数
    else card dom a 貸出 (a 利用者) + card dom a 貸出本 <= a 最大貸出数;
.....
```

3.2.8 蔵書の検索に関する関数

```
.....
public
  検索キーが空でない : 題名 | 著者 | 分野 +> bool
  検索キーが空でない (a 検索キー) ==
    a 検索キー <> ""
.....
.....
end
図書館 RD
.....
```

4　参考文献、索引

VDM++[2] は、1970 年代中頃に IBM ウィーン研究所で開発された VDM-SL[1] を拡張し、さらにオブジェクト指向拡張した形式仕様記述言語である。

VDM++ の教科書としては [3] がある。

VDM++ を開発現場で実践的に使う場合の解説が [4] にある。

参考文献

[1] Kyushu University. VDM-SL 言語マニュアル. Kyushu University, 第 2.0 版, 2016. Revised for VDMTools V9.0.2.

[2] Kyushu University. VDMTools VDM++ 言語マニュアル. Kyushu University, 第 2.0 版, 2016. Revised for VDMTools V9.0.2.

[3] ジョン・フィッツジェラルド, ピーター・ゴルム・ラーセン, ポール・マッカージー, ニコ・プラット, マーセル・バーホフ (著), , 酒匂寛 (訳). VDM++ によるオブジェクト指向システムの高品質設計と検証. IT architects' archive. 翔泳社, 2010.

[4] 佐原伸. 形式手法の技術講座—ソフトウェアトラブルを予防する. ソフトリサーチセンター, 2008.

索引