運賃計算システム VDM++ 仕様

佐原　伸

法政大学大学院
情報科学研究科

概要

レコードの集合と操作を使った運賃計算の例。

事前条件を無くし、エラー処理で例外を発生させ、回帰テストケースで例外を処理することも行っている。

型やインスタンス変数の不変条件、事後条件、事前条件、簡易回帰テストの例も含んでいる。

組み合わせテスト機能は、まだ、プロトタイプのため表示がおかしいし、説明はしないが、本モデルのテストには使用した。

目次

1 運賃クラス

要求仕様レベルの運賃を表す。

```
class
運賃 is subclass of 運賃RD
instance variables
  s運賃集合 : set of 運賃レコード := {};

operations
public
  運賃 : set of 運賃レコード ==> 運賃
  運賃(a運賃集合) ==
    (   s運賃集合 := a運賃集合;
        return self
    );
```

1.1 運賃を得る

運賃レコードの集合である s 運賃集合から、2 駅間の運賃を計算する。

```
public
  運賃を得る : 駅 * 駅 ==> nat
  運賃を得る(a駅1, a駅2) ==
    let w運賃レコード = 運賃レコードを得る(s運賃集合, a駅1, a駅2) in
    return w運賃レコード.f運賃;
```

1.2 運賃レコードを追加する

運賃レコードの集合である s 運賃集合に、運賃レコードを追加する。

要求仕様レベルのエラー処理を行っている例である。a 運賃レコードが、すでに s 運賃集合にある場合は、**<運賃データの重複>**例外を発生させる。

```
public
```

```
  運賃レコードを追加する : 運賃レコード ==> ()
  運賃レコードを追加する (a 運賃レコード) ==
    let w 運賃集合 = 運賃集合を得る () in
    if a 運賃レコード in set w 運賃集合
    then exit <運賃データの重複>
    else s 運賃集合 := 運賃 RD`運賃レコードを追加する (w 運賃集合, a 運賃レコード);
```

1.3 運賃レコードを更新する

運賃レコードを更新する

```
public
  運賃レコードを更新する : 運賃レコード * 運賃レコード ==> set of 運賃レコード
  運賃レコードを更新する (anOld, aNew) ==
    運賃 RD`運賃レコードを更新する (s 運賃集合, anOld, aNew) ;
```

1.4 運賃集合を得る

運賃集合を得る。

```
public
  運賃集合を得る : () ==> set of 運賃レコード
  運賃集合を得る () ==
    return 運賃 RD`運賃集合を得る (s 運賃集合)
end
運賃
```

Test Suite : vdm.tc

Class : 運賃

Name	#Calls	Coverage
運賃`運賃	1	√
運賃`運賃を得る	0	0%
運賃`運賃集合を得る	0	0%
運賃`運賃レコードを更新する	0	0%
運賃`運賃レコードを追加する	0	0%
Total Coverage		**17%**

2 運賃要求辞書

運賃のドメイン知識を持つ。

運賃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。

```
class
運賃 RD
```

2.1 型定義

```
types
  public 駅 = seq of char
  inv w 駅 == w 駅 <> "";
public
  運賃レコード::f 駅 1 : 駅
                f 駅 2 : 駅
                f 運賃 :- nat
  inv w 運賃レコード == w 運賃レコード.f 駅 1 <> w 運賃レコード.f 駅 2
```

2.2 関数定義

2.3 運賃レコードを得る

運賃レコードの集合から、2 駅間の運賃レコードを得る。

```
functions
public
  運賃レコードを得る : set of 運賃レコード * 駅 * 駅 -> 運賃レコード
  運賃レコードを得る (a 運賃レコード集合, a 駅 1, a 駅 2) ==
    CRUD_SET`ReadFromSet[運賃レコード](a 運賃レコード集合, mk_運賃レコード (a 駅 1, a 駅 2, 0));
```

2.4 運賃レコード集合を得る

運賃レコード集合から、条件を満たす運賃レコード集合を得る。

```
public
```

```
  運賃レコード集合を得る : ( 運賃レコード -> bool) -> set of 運賃レコード -> set of 運賃レコード
  運賃レコード集合を得る (a レコード選択関数)(a 運賃レコード集合) ==
    CRUD_SET`ReadSets[運賃レコード] (a レコード選択関数) (a 運賃レコード集合);
```

2.5　運賃レコードを追加する

```
  運賃レコードを追加する。
```

```
public
  運賃レコードを追加する : set of 運賃レコード * 運賃レコード -> set of 運賃レコード
  運賃レコードを追加する (sr,r) ==
    CRUD_SET`CreateSet1[運賃レコード] (sr,r);
```

2.6　運賃レコードを追加する

```
  運賃レコードを追加する。
```

```
public
  運賃レコードを追加する : set of 運賃レコード * set of 運賃レコード -> set of 運賃レコード
  運賃レコードを追加する (sr1,sr2) ==
    CRUD_SET`CreateSet[運賃レコード] (sr1,sr2);
```

2.7　運賃レコードを更新する

```
  運賃レコードを更新する
```

```
public
  運賃レコードを更新する : set of 運賃レコード * 運賃レコード * 運賃レコード -> set of 運賃レコード
  運賃レコードを更新する (sr,anOld,aNew) ==
    CRUD_SET`UpdateSet[運賃レコード] (sr,anOld,aNew);
```

2.8　運賃集合を得る

```
  運賃集合を得る。
```

```
public
  運賃集合を得る : set of 運賃レコード -> set of 運賃レコード
  運賃集合を得る (sr) ==
    CRUD_SET`ReadSets[運賃レコード](lambda - : 運賃レコード & true)(sr)
.....
.....
end
```

運賃 RD

.....

Test Suite : vdm.tc

Class : 図書館 RD

Name	#Calls	Coverage
Total Coverage		**1%**

3 CRUD_SET

集合の CRUD 関数ライブラリー。

```
class
CRUD_SET
functions
public
  CreateSet1[@Elem] : set of @Elem * @Elem -> set of @Elem
  CreateSet1(s,e) ==
    s union {e}
  pre  e not in set s
  post RESULT = s union {e} ;
public
  CreateSet[@Elem] : set of @Elem * set of @Elem -> set of @Elem
  CreateSet(s,as) ==
    s union as
  pre  as inter s = {}
  post RESULT = s union as ;
public
  ReadSets[@Elem] : (@Elem -> bool) -> set of @Elem -> set of @Elem
  ReadSets(f)(s) ==
    {e | e in set s & f(e)}
  pre  s <> {}
  post RESULT = {e | e in set s & f(e)} ;
public
  ReadFromSet[@Elem] : set of @Elem * @Elem -> @Elem
  ReadFromSet(s,e) ==
    let  es in set s be st  es = e in
    es
  pre  e in set s
  post RESULT = e ;
public
  UpdateSet[@Elem] : set of @Elem * @Elem * @Elem -> set of @Elem
  UpdateSet(s,de,ae) ==
    let ws = DeleteFromSet[@Elem](s,de) in
    CreateSet1[@Elem](ws,ae)
  pre  de in set s
  post ae in set s ;
```

```
public
  DeleteFromSet[@Elem] : set of @Elem * @Elem -> set of @Elem
  DeleteFromSet(s,e) ==
    s \ {e}
  pre  e in set s
  post RESULT = s \ {e}
end
CRUD_SET
```

4 Test クラス

運賃の回帰テストケースである。
エラーケースを、一部考慮している。

```
.....................................................................................
class
MyTestCase is subclass of TestCase
instance variables
  public s運賃 : 運賃 := new 運賃({
                        mk_運賃‘運賃レコード("東京","品川",220),
                        mk_運賃‘運賃レコード("東京","新宿",180),
                        mk_運賃‘運賃レコード("東京","四谷",150),
                        mk_運賃‘運賃レコード("四谷","新宿",130),
                        mk_運賃‘運賃レコード("新宿","品川",190),
                        mk_運賃‘運賃レコード("品川","新宿",170)});

operations
public
```

```
  test01 : () ==> ()
  test01() ==
    (   assertTrue("test01 東京・品川の運賃がおかしい。",
              s 運賃.運賃を得る("東京","品川") = 220);
        assertTrue("test01 東京・品川の運賃がおかしい。",
              s 運賃.運賃を得る("東京","新宿") = 180);
        assertTrue("test01 東京・四谷の運賃がおかしい。",
              s 運賃.運賃を得る("東京","四谷") = 150);
        assertTrue("test01 東京・品川の運賃がおかしい。",
              s 運賃.運賃を得る("新宿","品川") = 190);
        assertTrue("test01 東京・品川の運賃がおかしい。",
              s 運賃.運賃を得る("新宿","品川") = 190);
        s 運賃.運賃レコードを追加する(mk_運賃‘運賃レコード("東京","大阪",13000));
        assertTrue("test01 東京・大阪の運賃がおかしい。",
              s 運賃.運賃を得る("東京","大阪") = 13000);
        assertTrue("test01 1000 円以上の運賃レコードの抽出がおかしい。",
              let w 運賃集合 = s 運賃.運賃集合を得る(),
                  r 運賃集合 = s 運賃.運賃レコード集合を得る(lambda e : 運賃‘運賃レコード & e.f 運賃 > 1000)(w 運賃集合) in
              r 運賃集合 = {mk_運賃‘運賃レコード("東京","大阪",13000)});
        assertTrue("test01 東京・大阪の運賃がおかしい。",
              s 運賃.運賃を得る("東京","大阪") = 13000);
        assertTrue("test01 東京・大阪の運賃がおかしい。",
              let w 運賃集合 = s 運賃.運賃レコードを更新する(mk_運賃‘運賃レコード("新 宿","品 川",190),mk_運賃‘運賃レコード("新宿","品川",290)) in
              mk_運賃‘運賃レコード("新宿","品川",290) in set w 運賃集合)
    );
```

4.1 エラー処理をテストするテストケース t4

運賃レコードが testE01 で追加したものと重複したため、例外<**運賃データの重複**>が発生することを確認している。

```
public
  testE01 : () ==> bool
  testE01() ==
    (   trap <運賃データの重複> with  return truein
```

```
        s 運賃.運賃レコードを追加する (mk_運賃 ‘運賃レコード ("新宿", "品川", 290)) ;
        return false
    )
end
MyTestCase
```

Test Suite : vdm.tc

Class : MyTestCase

Name	#Calls	Coverage
MyTestCase‘test01	0	0%
MyTestCase‘testE01	0	0%
Total Coverage		**0%**

5 組み合わせテストケース UseFare1

運賃の組み合わせテストケースである。

組み合わせテスト自体は、まだプロトタイプであるため説明しない。

```
class
UseFare1 is subclass of 運賃
values
public
  v 運賃 = new 運賃 ({
                    mk_運賃 ‘運賃レコード ("東京","品川",220),
                    mk_運賃 ‘運賃レコード ("東京","新宿",180),
                    mk_運賃 ‘運賃レコード ("新宿","品川",190),
                    mk_運賃 ‘運賃レコード ("品川","新宿",170)})
traces
T1 :
  let s1 in set {"東京","新宿","品川"} in
  let s2 in set {"新宿","東京","品川"} in
  v 運賃.運賃を得る (s1,s2)
  ;
end
UseFare1
```

6 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。

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.

索引