運賃計算システム VDM++ 仕様

佐原　伸

法政大学
情報科学研究科

概要

写像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。

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
types
  public 金額 = nat;
  public 区間 = set of 駅;
  public 駅 = seq of char
  inv s == s <> "";
  public 運賃表 = map 区間 to 金額
instance variables
  s 運賃表 : 運賃表 := { |-> };

operations
public
  運賃 : 運賃表 ==> 運賃
  運賃 (a 運賃表) ==
    (   s 運賃表 := a 運賃表;
        return self
    );
```

1.1　運賃を得る

区間から金額への写像である s 運賃表から、区間の運賃を計算する。区間が運賃表に存在しない場合、**<運賃表に存在しない区間>**例外を発生させる。

```
public
  運賃を得る : 区間 ==> 金額
  運賃を得る (a 区間) ==
    if a 区間 not in set dom s 運賃表
    then exit <運賃表に存在しない区間>
    else return s 運賃表 (a 区間)
  post RESULT = s 運賃表 (a 区間);
```

1.2 運賃表に追加する

区間から金額への写像である s 運賃表に、運賃表を追加する。

要求仕様レベルのエラー処理を行っている例である。運賃表に区間がすでにある場合は、**<運賃データの重複>**例外を発生させる。

```
public
  運賃表に追加する : 運賃表 ==> ()
  運賃表に追加する (a 運賃表) ==
    if dom a 運賃表 subset dom s 運賃表
    then exit <運賃データの重複>
    else s 運賃表 := s 運賃表 munion a 運賃表;
public
  運賃表を得る : () ==> 運賃表
  運賃表を得る () ==
    return s 運賃表
end
運賃
```

Test Suite : vdm.tc

Class : 運賃

Name	#Calls	Coverage
運賃 ‘運賃	2	√
運賃 ‘運賃を得る	4	√
運賃 ‘運賃表を得る	2	√
運賃 ‘運賃表に追加する	2	√
Total Coverage		**100%**

2 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
Test
instance variables
  public s運賃 : 運賃 := new 運賃({{"東京","品川"} |-> 220,{"東京","新宿"} |-> 180});

functions
public static
  makeOrderMap : seq of bool +> map nat to bool
  makeOrderMap(s) ==
    {i |-> s(i) | i in set inds s}
operations
public
  run : () ==> seq of char * bool * map nat to bool
  run() ==
    let testcases = [t1(),t2(),t3(),t4(),t5(),t6(),t7()],
        testResults = makeOrderMap(testcases) in
    return mk_("The result of regression test = ",
                forall i in set inds testcases & testcases(i),testResults);
public
  t1 : () ==> bool
  t1() ==
    return s運賃.運賃を得る({"東京","品川"}) = 220;
public
  t2 : () ==> bool
  t2() ==
    return s運賃.運賃を得る({"東京","新宿"}) = 180;
public
  t3 : () ==> bool
  t3() ==
    (   s運賃.運賃表に追加する({{"新宿","品川"} |-> 190});
        return s運賃.運賃を得る({"新宿","品川"}) = 190
    );
```

2.1 運賃データの重複例外処理をテストするテストケース t4

運賃レコードが t3 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
  t4 : () ==> bool
  t4() ==
    (   trap <運賃データの重複> with  return truein
        (   s 運賃.運賃表に追加する ({{"新宿","品川"} |-> 290});
            return false
        )
    );
public
  t5 : () ==> bool
  t5() ==
    (   def w 運賃表 = s 運賃.運賃表を得る();
            w 許容運賃 = 200 in
        return {w 区間 |-> w 運賃表(w 区間) | w 区間 in set dom w 運賃表 & w 運賃表(w 区間
) < w 許容運賃 } =
                {{"東京","新宿"} |-> 180,{"新宿","品川"} |-> 190}
    );
```

2.2 運賃表に存在しない区間例外処理をテストするテストケース t6

運賃表に該当区間が存在しないため、例外<運賃表に存在しない区間>が発生することを確認している。

```
public
  t6 : () ==> bool
  t6() ==
    (   trap <運賃表に存在しない区間> with  return truein
        (   def - = s 運賃.運賃を得る ({"大阪","京都"}) = 190 in
            return false
        )
    );
```

2.3 RuntimeError 例外処理をテストするテストケース t7

実行時エラー「Run-Time Error 67: 重複要素は異なる値を持ちます」が発生することを確認している。

```
public
  t7 : () ==> bool
  t7() ==
    (   trap <RuntimeError> with  return truein
        (   def - = s 運賃.運賃表を得る() munion {{"東京","品川"} |-> 320} in
            return false
        )
    )
end
Test
```

Test Suite : vdm.tc

Class : Test

Name	#Calls	Coverage
Test‘t1	1	√
Test‘t2	1	√
Test‘t3	1	√
Test‘t4	1	80%
Test‘t5	1	√
Test‘t6	1	76%
Test‘t7	1	83%
Test‘run	1	√
Test‘makeOrderMap	1	√
Total Coverage		**92%**

3　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
UseFare2 is subclass of 運賃
values

  v 運賃 = new 運賃 ({{"東京", "品川"} |-> 220, {"東京", "新宿"} |-> 180})
traces
T1 :
  let s1 in set {"東京", "品川", "新宿", "大阪"} in
  let s2 in set {"東京", "品川", "新宿", "京都"} in
  v 運賃.運賃を得る ({s1, s2})
  ;
end
UseFare2
```

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。

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