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
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
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 in set s 運賃集合 be st
            {a 駅 1, a 駅 2} = {w 運賃レコード.f 駅 1, w 運賃レコード.f 駅 2} in
   return w 運賃レコード.f 運賃
 pre  {a 駅 1, a 駅 2} in set {{e.f 駅 1, e.f 駅 2} | e in set s 運賃集合 }
```

```
  post  exists1 w 運賃レコード in set s 運賃集合 &
          {a 駅 1, a 駅 2} = {w 運賃レコード.f 駅 1, w 運賃レコード.f 駅 2} and
          RESULT = 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) ==
    s 運賃集合 := s 運賃集合 union {a 運賃レコード }
  pre  a 運賃レコード not in set s 運賃集合
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	3	√
運賃`運賃レコードを追加する	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({
                        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)});

operations
public
  run : () ==> bool * seq of bool
  run() ==
    let testcases = [t1(),t2(),t3(),t4()] in
    return mk_(forall i in set inds testcases & testcases(i),testcases);
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.運賃レコードを追加する(mk_運賃‘運賃レコード("新宿","品川",190));
        return s運賃.運賃を得る("新宿","品川") = 190
    );
```

2.1 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 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
Test
```

Test Suite : vdm.tc

Class : Test

Name	#Calls	Coverage
Test‘t1	1	√
Test‘t2	1	√
Test‘t3	1	√
Test‘t4	1	64%
Test‘run	1	10%
Total Coverage		**74%**

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
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)})
traces
T1 :
  let s1 in set {"東京", "品川", "新宿"} in
  let s2 in set {"東京", "品川", "新宿"} in
  v 運賃.運賃を得る (s1, s2)
  ;
end
UseFare1
```

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

索引