Train Fare System Specification by VDM++

Shin Sahara

Hosei University

Abstract

This is an example of requirement specification for calculation of train fare. This example uses operations, and set of records. This example includes type invariants, instance variable invariants, post-conditions, preconditions, and a simple regression test too. There is another test mechanism called ”combinatorial test”.

Contents

1 Fare

I'm a train fare in requirement specification layer.

```
class
Fare
types
  public Station = seq of char
  inv wStation == wStation <> "";
public
  FareRecord::fDeparture : Station
              fArrival : Station
              fFare :- nat
  inv fr == fr.fDeparture <> fr.fArrival
instance variables
  sFareSet : set of FareRecord := {};

operations
public
  Fare : set of FareRecord ==> Fare
  Fare (aFareSet) ==
    (   sFareSet := aFareSet;
        return self
    ) ;
```

1.1 Calculate_fare

Calculate fare between 2 stations from set of FareRecord.

```
public
  Calculate_fare : Station * Station ==> nat
  Calculate_fare (aDeparture, anArrival) ==
    let  wFareRecord in set sFareSet be st
             {aDeparture, anArrival} = {wFareRecord.fDeparture, wFareRecord.fArrival} in
    return wFareRecord.fFare
  pre  exists1 wFareRecord in set sFareSet &
          {aDeparture, anArrival} = {wFareRecord.fDeparture, wFareRecord.fArrival}
  post  exists1 wFareRecord in set sFareSet &
           {aDeparture, anArrival} = {wFareRecord.fDeparture, wFareRecord.fArrival} and
           RESULT = wFareRecord.fFare ;
```

1.2 AppendFareRecord

Append FareRecord to the instance variable sFareSet.
There is an error processing example.

```
public
```

```
  AppendFareRecord : FareRecord ==> ()
  AppendFareRecord (aFareRecord) ==
    sFareSet := sFareSet union {aFareRecord}
  pre  aFareRecord not in set sFareSet
  post sFareSet = sFareSet~ union {aFareRecord}
end
Fare
```

....

Test Suite : vdm.tc
Class : Fare

Name	**#Calls**	**Coverage**
Fare‘Fare	2	√
Fare‘Calculate_fare	3	√
Fare‘AppendFareRecord	2	√
Total Coverage		**100%**

2 Test

I'm a regression test case of train fare.
I take care of some error case.

```
class
Test
instance variables
  public sFare : Fare := new Fare ({
                             mk_Fare`FareRecord ("Tokyo", "Shinagawa", 220),
                             mk_Fare`FareRecord ("Tokyo", "Shinjuku", 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 ()],
        testResults = makeOrderMap (testcases) in
    return mk_ ("The result of regression test = ", forall i in set inds testcases & testcase
public
  t1 : () ==> bool
  t1 () ==
    return sFare.Calculate_fare ("Tokyo", "Shinagawa") = 220;
public
  t2 : () ==> bool
  t2 () ==
    return sFare.Calculate_fare ("Tokyo", "Shinjuku") = 180;
public
  t3 : () ==> bool
  t3 () ==
    (   sFare.AppendFareRecord(mk_Fare`FareRecord ("Shinjuku", "Shinagawa", 190));
        return sFare.Calculate_fare ("Shinjuku", "Shinagawa") = 190
    ) ;
```

2.1 t4 – testing error processing

FareRecord is duplicated, so `<DuplicateFareRecord>` exception have to occur.

```
public
  t4 : () ==> bool
  t4 () ==
    (   trap <RuntimeError> with  return truein
        (   sFare.AppendFareRecord(mk_Fare‘FareRecord("Shinjuku","Shinagawa",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	78%
Test‘run	1	√
Test‘makeOrderMap	1	√
Total Coverage		**95%**

3 UseFare1 – Combinatorial test

I am a combinatoial test of calculating fare.

```
class
UseFare1 is subclass of Fare
values
public
  vFare = new Fare ({
                       mk_Fare`FareRecord ("Tokyo", "Shinagawa", 220),
                       mk_Fare`FareRecord ("Tokyo", "Shinjuku", 180),
                       mk_Fare`FareRecord ("Shinjuku", "Tokyo", 280)})
traces
T1 :
  let s1 in set {"Tokyo", "Shinagawa", "Shinjuku"} in
  let s2 in set {"Tokyo", "Shinagawa", "Shinjuku"} in
  vFare.Calculate_fare (s1, s2)
  ;
end
UseFare1
```

4 Reference、Index

VDM++[1] is a formal specification description language that extended VDM-SL[2] developed by IBM Vienna Research Center in the mid-1970 and further object oriented extension.

References

[1] Kyushu University. *VDMTools VDM++ Language Manual.* Kyushu University, 2.0 edition, 2016.

[2] Kyushu University. *VDMTools VDM-SL Language Manual.* Kyushu University, 2.0 edition, 2016.

Index