関数スタイルによる機能形式仕様のテスト支援に関する考察

2012SE085 春日井勇貴
指導教員：張漢明

1 はじめに

プログラムのテストを行う際，仕様記述は重要なものであり，仕様の妥当性確認や設計と、プログラムの正当性確認の基準は，テストでは欠かせないものである．また，機能仕様の妥当性確認を行う際，VDM-SL で記述された機能仕様があることと，テストを実行するということが前提である [1]．しかし，テストケースを手動で作成する際，どのようなテストケースがあるのか，またどのような結果が見込まれるのかを全て考えなくてはならず，期待通りではない場合，再度テストケースを作る等，テストケースの作成には手間がかかるという問題点がある [2]．本研究の目的は，関数スタイルの VDM-SL 機能仕様の妥当性を確認するためのテストの支援を行うことである．そのためには、テストケースの書き方を考え、テストケースの自動実行ができるように、テスト実行環境の整備を行う必要がある．

2 背景技術

2.1 VDM-SL

VDM (Vienna Development Method) は 1970 年にウィーン研究所で開発されたウィーン定義言語 (Vienna Definition Language = VDL) に端を発する、長い研究の流れから生み出されたものの 1 つである．VDM-SL は、VDM のフォーマルな形式記述言語のひとつであり、ISO/IEC 1318-1 により標準化された．VDM-SL には，VDMPad, Overture, VDMTools の 3 種類のツールがある．特に VDMPad は，Overture, VDMTools と比べると，容易に利用できるという利点がある．本研究は，この VDMPad を利用して研究を行う．

2.2 関数スタイルの VDM-SL 機能仕様

図 1 VDM-SL 機能仕様の定義

状態の変化を関数で表現したものが，『関数スタイル』である．図 1 のように，入力の前状態から，手続き的な関数を記述して操作を行うことで，後状態が出力される．定義として，操作名と操作型の記述を行い，入力と前状態の事前条件，事後条件を入力する．関数スタイルでは、操作を入力と前状態の組から、出力と後状態の組への関数として記述する．テストが容易に行えるようにするには，テストを自動実行する関数を用意しなくてはならない [3]．

3 研究のアプローチ

本研究では、以下の観点より研究を進めた．

(1) 関数適用の手続き化

関数を用いてシナリオを実行するためには，以下のように関数適用の連続となってしまう．

```
関数 4(対象，関数 3(対象，関数 2(引数 2，関数 1(引数 1，目的となる物))
```

普通に関数を適用させると，テストケースに間違いがあった場合は間違いに気づきにくい，操作を追加した際も関数適用の連続となる等の問題点が生じる．

```
関数 1(引数 1)，関数 2(引数 2)，関数 3(対象)，関数 4(対象)
```

上記のように関数適用を手続き的に扱うことで，どのような操作を行うのかがわかりやすくなる．また、操作を追加しても手間がかからない．

(2) テストケースの書き方

テストケースは，関数の適用を手続き的に扱えるようにする．操作列，初期値，期待値を構成し，期待値の作成支援を行う．

(3) テストケースの実行

テストケースを実行するため，自動実行する関数を作成する必要がある．詳細については後述する．

(4) ツールの利用

VDM-SL のテストを行うため，ツールとして，VDMPad を利用する．VDMPad は，Overture や VDMTools といったツールと比べると，簡易に利用できるという利点がある．

(5) 自動販売機の事例に適用

本研究で得られたことを自動販売機の事例に適用して，テスト支援に関する考察を行う．

4 VDM-SL のテスト実行環境の整備

4.1 関数適用を操作列の実行で表現

関数適用を操作列の実行で表現するには、以下の定義を行う．

```
types
  操作型 = 状態型 -> 状態型;
  操作列型 = seq of (操作型);
```

また、操作列を実行できるようにするには、以下の関数を用意する．

```
functions
  操作列を実行:操作列型 * 状態型 -> 状態型
  操作列を実行(操作列，現在の状態) ==
```

```
if 操作列 = []
then 現在の状態
else 操作列を実行 (tl 操作列, (hd 操作列)(現在の状態));
```

4.2 テストケースの書き方

テストケースを手続き的に扱うことで，テストケースを容易に実行することが可能になる．テストケースは、以下の様式で用意する．

```
テストケース型::
    操作列: 操作列型
    初期値: 状態型
    期待値: 状態型;
```

4.3 テストケースの実行

テストケースの実行を行うための関数として，以下の関数を用意する．

テストケースを調べる

```
functions
 テストケースを調べる:テストケース型 -> bool
 テストケースを調べる (mk_テストケース型
 (操作列, 初期値, 期待値)) ==
  操作列を実行 (操作列, 初期値) = 期待値;
```

この関数は，テストケースが正しいのかどうかの妥当性確認を行う．指定したテストケースが期待値と同じになる場合は true を返し，期待値と異なる場合は false を返す．

テストケース列を調べる

```
functions
 テストケース列を調べる:seq of テストケース型 -> seq of bool
 テストケース列を調べる (s) ==
  if s = []
  then []
  else [テストケースを調べる (hd s)] ^ テストケース列を調べる
(tl s);
```

予めテストケース名を列挙したテストケース列を用意し，各テストケースが期待値通りになるかどうかを確認する．結果は true または false で返された列になる．

5 考察

5.1 事例：自動販売機

```
functions
  販売ボタン押下操作:販売ボタン型 -> 操作型
  販売ボタン押下操作 (販売ボタン) ==
   lambda obj:
    自動販売機型 & 販売ボタン押下 (販売ボタン, obj);
```

自動販売機の事例に適用すると，操作は上記のようになる．自動販売機には，販売ボタン押下，貨幣投入，返金ボタン押下の 3 つの操作があり，それぞれ操作列型から操作型へ関数を返している．テストケース及びテストケース列は以下のように記述し，操作列では自動販売機の操作を，期待値では商品の在庫や金庫の変数をそれぞれ定義する．

```
values
  操作列_102 = [
    貨幣を投入操作 (100), 貨幣を投入操作 (100),
    販売ボタン押下操作 (販売ボタン_0),
    販売ボタン押下操作 (販売ボタン_0)
  ];

  期待値_102 =
    出力設定 (商品_0, 0,
    在庫更新 (商品_0, -2,
    金庫更新 (200,
    自動販売機_0)));

  テストケース_102 = mk_テストケース型 (
    操作列_102,
    自動販売機_0,
    期待値_102);
```

5.2 テストケースの期待値作成支援

また，テストケースを調べるために，期待値を作成しなければならない．以下のように定義し，テストケースの期待値の作成支援を行う．

```
functions
  在庫更新:商品型 * int * 自動販売機型 -> 自動販売機型
  在庫更新 (商品, num, 自動販売機) ==
    mu(自動販売機,
      在庫 |-> 自動販売機.在庫 ++
      {商品 |-> 自動販売機.在庫 (商品) + num});

  金庫更新:int * 自動販売機型 -> 自動販売機型
  金庫更新 (num, 自動販売機) ==
    mu(自動販売機,
      金庫 |-> 自動販売機.金庫 + num);

  出力設定: [商品型] * [金額型] * 自動販売機型 -> 自動販売機型
  出力設定 (商品, 金額, 自動販売機) ==
    mu(自動販売機,
      出力 |-> mk_出力型 (商品, 金額));
```

商品排出による在庫の減少，投入された貨幣の金庫への追加といった変数も，上記のように定義しておけば，変数を全て定義しなくて済む．

6 おわりに

状態遷移テストと同値クラステストのテストが容易になるよう，関数スタイルによる機能形式仕様のテスト支援に関する考察を行ってきた．

テストを実行するための関数と操作型の定義を行い，テストケースの期待値の作成を支援したり，テストケースを手続き的に扱うことで，テストを容易になり，問題点が解決されるということがわかる．

参考文献

[1] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef, *Validated Designs for Object-oriented Systems*, Springer, 2005.

[2] 玉井哲雄:『ソフトウェア工学の基礎』．岩波書店, 2005.

[3] 荒木啓二郎，張漢明:『プログラム仕様記述論』，オーム社，2002.