2006－SE－151（17）
2006／3／24

VDM におけるテスティング方式についての考察

田口研治*　寺本宏子**　本位田真一*
国立情報学研究所*　（株）CSK システムズ**

抄録　本論文は，形式仕様記述言語 VDM-SL におけるテスティングの論理的枠組について VDM-SL における陰・陽関数定義からの考察と，VDM-SL のオブジェクト指向への拡張である，VDM++ のテスティング・フレームワークについてのサーベイ結果を示す.

On the Testing Method in VDM

Kenji Taguchi*　Hiroko Teramoto**　Shinichi Honiden*
National Institute of Informatics*　CSK Systems Corp**

Abstract　This paper outlines the logical foundations of testing in VDM-SL via definitions of its implicit/explicit functions in VDM-SL, and shows the survey result on currently available testing frameworks for VDM++, an object-oriented extension of VDM-SL.

１. はじめに

形式手法の分野ではテスティングについては長い間無視されていた．主な理由としては，形式的な検証を定理証明により行うのが主流だったからである．ただし，現実的には定理証明器を用いて形式的に検証するのはコストがかかり，形式手法の現場への普及の大きな障害になっている．形式的証明は必ずしも操作的な正しさを保障するものではないので，動的なテストが必要になるなど，形式的証明が常に万全ではないという問題もある．近年においては，形式手法とテスティングを組み合わせることでテスティングのコストを低くする試みや,テスティングと形式手法を補完するものとして関連付ける試みがある（[10]）.

本論文においては,モデル(状態)に基づいた仕様記述言語である VDM-SL（[3], [12]）の形式仕様への，テスティングを行うことに対する論理的枠組みを考察することを主眼とする．それに加えて，VDM-SL のオブジェクト指向への拡張である VDM++ （[6]）に対する現在利用可能なテスティングのフレームワークについてのサーベイの結果を簡単に報告する.

VDM-SL（Vienna Development Methods-Specification Language）は IBM の Vienna 研究所におけるプログラム言語の形式的意味論の研究から始まった．ISO による標準化を経て，現在では言語仕様についてはほぼ確定している．VDM-SL は厳密な意味論に基づいた形式仕様記述言語であり，豊富なデータ型とそれ上の操作の提供，状

態遷移システムとしての仕様記述，論理学としては非定義の値を取り扱える三値論理 LPF (Logic of Partial Function)を用いる，という特徴を持つ．

VDM-SL においては 陰（implicit）定義（事前条件と事後条件により宣言的に定義）と陽（explicit）定義（実行可能）の両方が関数と操作に対して可能である．実行可能な仕様記述言語として VDM-SL を捉えると，陰定義は陽定義の形式的な仕様と考えられる．テスティングの対象になるのは，陽定義の関数であるが，もしその陰定義が利用可能ならば，テスティングはどのように行われるかを，論理的枠組みから本論文では考察する．

本論文の構成は，第２章において，論理的枠組みについての定式化を行い，第３章において，現在利用可能なテスティング・フレームワークについてのサーベイの結果を示す．最後に第４章においては，今後の研究課題について述べる．

２．　論理的枠組み

２．１．　陰定義と陽定義間の論理的関係

議論を簡単にするために，関数定義とデータ型だけを対象に議論を進める．本論文においては，論理記号，演算子は VDM に準拠した形で用いる．

【陰関数定義】

i_fun(x_1 : D_1, … , x_n : D_n) y : R
pre P(x_1, … , x_n)
post P'(x_1, … , x_n, y);

ここで， x_1, … , x_n は引数，D_1, … , D_n は引数のデータ型， y は返り値， R はそのデータ型とする． pre_i_fun(x_1, … , x_n) と post_i_fun(x_1, … , x_n, y) は事前条件，事後条件を表す述語論理式とする．

陽（explicit）関数は次の形をしている．

【陽関数定義】

e_fun : D_1 * … * D_n -> R
e_fun(x_1, … , x_n) = =
fun_body
pre P(x_1, … , x_n);

ここで，陽関数の事前条件を表す述語論理式を次のものとする．

pre_e_fun(x_1, … , x_n)

VDM においては，データ型にも不変式を定義できる．

【型定義】

D をユーザ定義のデータ型とする．

D = データ定義

inv d = = inv_D(d)

ここで，inv_D はデータ型 D 固有な不変式（述語論理式）を表す．

陰定義と陽定義との間の論理的関係は以下の証明責務により定義される．

【証明責務】

forall $<d_1, \cdots, d_n>$ in $D_1 * \cdots * D_n$ & forall i $(1 <= i <= n)$ &

pre_e_fun($d_1, \cdots, d_n$) and pre_i_fun($d_1, \cdots, d_n$) and inv_D_i (d_i)

=>

e_fun($d_1, \cdots, d_n$) in R and inv_R(e_fun($d_1, \cdots, d_n$)) and

post_i_fun($d_1, \cdots, d_n$, e_fun($d_1, \cdots, d_n$))

すなわち，引数が取りうる全ての値に対して，陽関数の事前条件式と陰関数の事前条件式が成立し，データ不変式を満すならば，陽関数の返り値は定義された返り値のデータ型であり，データ不変式と陰関数の事後条件式を満たす．

２．２．テスティングの定義

ここでは関数の陰・陽定義とテスティングとの関連を述べる．最初にテスティングの用語を整理する．

【テスティングの基本用語の整理】

陽関数 e_fun : $D_1 * \cdots * D_n$ -> R に対するテストデータとは下記のものである

テストデータの集合： TIn subset $D_1 * \cdots * D_n$

テスト結果の集合： TOut subset R

テストデータとテスト結果を結びつける関数： g : TIn -> TOut

テストする関数 e_fun に対して以下の条件が成立すれば，関数はテストを通過したと言う．

e_fun はテストを通過 <=> forall $<d_1, \cdots, d_n>$ in TIn & $e_fun(d_1, \cdots, d_n) = g(d_1, \cdots, d_n)$

要求分析に関する書類からテストデータを作成し，入出力だけを考慮するテスティングの方法はブラックボックス・テスティングと呼ばれる．ここでは，仕様が（正しい）陰定義として与えられているブラックボックス・テスティングの枠組みを考慮する．

２．３． テストデータの正しさと陰定義の論理的関係

実際のシステム開発においては，テストデータが間違っていることは往々にしてある．陰定義が（正しい）仕様として与えられている場合には，テストデータの正しさについて次のことが言える．

【テスト入力用データ集合の正しさ】

テストデータの集合 TIn は，陰定義に関して正しいテストデータである <=>

forall $<d_1, \cdots, d_n>$ in TIn & forall i (1<= i <= n) &

$inv_D_i(d_i)$ and $pre_i_fun(d_1, \ldots, d_n)$

すなわち，与えられたテストデータの集合 TIn における全てのデータが，データ不変式を満たしかつ事前条件（陰関数の事前条件式 pre_i_fun ）を満たすのならば，TIn は正しいテストデータである．

ここでは，テスト用の入力データが正しい仕様を満たしているかどうかだけに言及していることに注意する必要がある．

テスト結果については，[1] に述べられているように，事後条件はオラクルとして働く．すなわち，正しいテスト結果を判定する論理式として考えることが出来る．

【テスト出力用データ集合の正しさ】

forall $<d_1, \cdots, d_n>$ in TIn & $g(d_1, \cdots, d_n)$ in TOut and $inv_R(g(d_1, \cdots, d_n))$ and $post_i_fun(d_1, \cdots, d_n, g(d_1, \cdots, d_n))$

すなわち，テストデータ $<d_1, \ldots, d_n>$ とそれに対応するテスト出力 $g(d_1, \ldots, d_n)$ が与えられた場合，それらのデータは陰関数定義における，事後条件式 post_i_fun を満たす．

ここで定義された，テストの入出力に関する定義を結合すると次の定義が導かれる．

【テスト入出力の正しさと陰関数の関係】

forall $<d_1, \cdots, d_n>$ in TIn & forall i (1<= i <= n) &

$inv_D_i(d_i)$ and $pre_i_fun(d_1, \cdots, d_n)$ =>

$g(d_1, \cdots, d_n)$ in TOut and inv_R($g(d_1, \cdots, d_n)$) and
post_i_fun($d_1, \cdots, d_n, g(d_1, \cdots, d_n)$)

これは証明責務とほとんど同一の定義になっていることが分かる．

２．４．　仕様の誤りの検証

実際のシステム開発においては，正しいと思われた仕様に誤りがあることが後で判明することは往々にしてある．形式仕様記述言語の場合は，非形式的な要求仕様を形式仕様が忠実に反映していないことがある．形式手法の分野においては，いかに試行錯誤して正しい仕様を導くかということについては，ほとんど議論の対象にはなっていなかったが，非形式的な要求仕様からテストデータを生成することで，形式仕様の正しさに確信を持つことが出来る．例えば，テストデータについて確信はあるが，テストデータと正しい（と思われる）仕様とを前節に述べられたように試験した結果，正しい応答を得られない場合，仕様自身が誤っている可能性がある．

ここでの重要な点は，現実のシステム開発においてはテストデータが誤っている場合もあるし，仕様が間違っている場合もあることである．そのような場合の仕様の改良方法については，今後研究の余地があると思われる．

２．５．　境界分析（boundary analysis）

入力境界は，陰関数の事前条件式で与えられる．VDM-SL においては，通常のプログラミング言語において用いられない集合データ型も用いられるので，境界分析は個々のデータ型について考える必要がある．

２．６．　パーティショニング

テストデータ生成の際のデータ量を減らすために，テストデータの集まりを等値類に分割する方法をパーティショニング（partitioning）と言う．

【パーティショニング】

$P_1, \cdots, P_n$ は TIn のパーティショニング <=> P_1 union … union P_n = TIn and
forall i, j & 1 <= i, j <= n => P_i inter P_j = {}

パーティショニングは単なるテストデータとして生成するのではなく，構文として支援することが望ましい（[2]）．

例：

Group: Type of Partition

Partition: P_1

Condition: C_1 (e.g., x < 0)

Partition: P_2

Condition: C_2 (e.g., x = 0)

Partition: P_3

Condition: C_3 (x > 0) ;

このようにパーティショニングを構文的に支援することにより，事前条件式において不足している条件を容易に見つけられる可能性がある．

【パーティショニングと陰定義との論理的関連】

実際には，各パーティションは $P_i = \{ x \mid P_i(x) \}$ subset TIn の形のデータの集合として与えることが出来るので，次の関係が成立する．

forall <d_1, ..., d_n > in TIn & forall i,j (1 <= i <> j <= n) & P_i and P_j => false and P_1 or … or P_n<=> pre_i_fun(d_1, … , d_n)

３．　テスト・フレームワーク

VDM におけるテスティング・フレームワークとして VDM-SL をオブジェクト指向に拡張した VDM++に対する，VDMUnit（[6]）と関数型回帰テスト支援ライブラリ（[8],[9]）を取り上げる．

３．１．VDMUnit

VDMUnit は E. Gamma と K. Beck （[11]）による Java の単体テスト・フレームワークを VDM++ 上に簡易的に実現したものである．VDM++ には JUnit のようにテストスイートの自動生成といった機能は無いが，基本的なクラス構成（図１），利用方法は同じである．VDM++Toolbox （[13]）において利用することで，カバリッジ情報の出力などが可能である．TestClass クラスのソースを示す（図２）．

３．２．関数型回帰テスト支援ライブラリ

VDM++ に回帰テストのためのライブラリを実装したものが [9] である．単体テストと回帰テストのためのフレームワークをオブジェクト指向に基づいて行うと，コード数が多くなるので，クラスをレコード型で，テスト用の手続きは手続き呼び出しを用いることで実行速度の向上，テスト用ソース量の削減を図ったものである．基本的な単体テストの機能は VDMUnit と同等である．

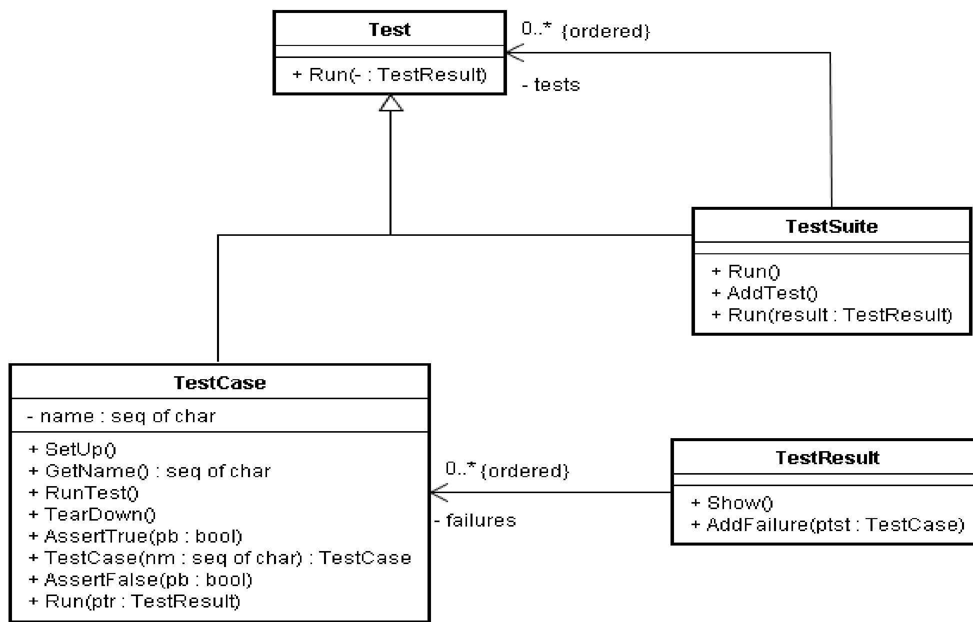

図１． VDMUnit のクラス構成

```
class TestCase is subclass of Test
protected AssertTrue : bool ==> ()
AssertTrue (b) == if not b then exit <FAILURE>;
public Run : TestResult ==> ()
    Run (tr) ==
      trap <FAILURE>
        with
          tr.AddFailure(self)
        in
          (SetUp();
           RunTest();
           TearDown()) ;
```

図２． VDMUnit における TestCase クラス

４． 結論

本論文においては，形式仕様記述言語 VDM-SL の仕様に対するテスティングの論理的枠組みを示すと同時に，現在利用可能なテスティング・フレームワークについてサーベイを行った．

説明した二つのフレームワークは VDM++ の実行可能な仕様に対して，従来のプログラムコードに対するテストと同一の技術を用いるという点において，特に形式仕様記述言

語であることによる特殊性というものが無い．それは陰定義が全く利用されていないのが主な理由である．今後は，テスティングと証明との関連についての理論的な基礎付け，有限状態機械を用いた VDM++ に対するテスト・フレームワークとその理論的背景についての調査，陰定義を用いることによりテストデータの整合性を検証する機能などを，今回調査したツールに実装することを検討する予定である．

尚，本論文の成果は，科学技術振興調整費新興分野人材養成「産学融合先端ソフトウェア技術者養成拠点の形成」におけるものである．

【参考文献】

[1] B. K. Aichernig, “Automated Black-Box Testing with VDM Oracles”, SAFECOMP 1999, pp250-259, LNCS 1698, Springer (1999)

[2] P. Madsen, “Unit Testing using Design by Contract and Equivalence Partitions”, XP 2003, pp425-426, LNCS 2675, Springer (2003)

[3] C. B. Jones, “Systematic Software Development using VDM” (2nd Ed), Prentice Hall (1990) (freely available at http://www.csr.ncl.ac.uk/vdm/ssdvdm.pdf.zip)

[4] S. Burton, J. Clark, J. McDermid, “Testing, Proof and Automation. An Integrated Approach”, International Workshop on Automated Program Analysis, Testing and Verification (2000)

[5] I. Burdonov, A. Kossatchev, A. Petrenko, D. Galter, “KVEST: Automated Generation of Test Suites from Formal Specifications”, FM’99 – Formal Methods: World Congress on Formal Methods in the Development of Computing Systems, pp608-621, LNCS 1708, Springer (1999)

[6] J. Fitzgerald, P. G. Larsen, et. al., “Validated Designs for Object-oriented Systems”, Springer (2005).

[7] A. A. Koptelov, V. V. Kuliamin, A. K. Petrenko, “Vdm++TesK: Testing of VDM++ programs”, unpublished

[8] 佐原伸, “VDM++ 基本ライブラリの作成”, ソフトウェア・シンポジウム (2004)

[9] 佐原伸, “VDM++ 関数型回帰テスト支援ライブラリ”, ソフトウェア・シンポジウム (2005)

[10] R.M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Luettgen, A.J. Simons, S. A. Vilkomir, M. R. Woodward, H. Zedan, “Working together: Formal Methods and Testing”, to appear in ACM Computing Surveys

[11] K. Beck, “JUnit Pocket Guide”, O’Reilly Media (2004)

[12] J. フィッツジェラルド, P. G. ラーセン, “ソフトウェア開発のモデル化技法”, 岩波書店 (2003)

[13] VDM++ Toolbox, CSK システムズ（株）