テストケース自動生成ツール BWDM の現状と課題

立山　博基 a)・片山　徹郎 b)

Current Status and Issues of Test Cases Automatic Generation Tool BWDM

Hiroki TACHIYAMA, Tetsuro KATAYAMA

Abstract

For software development using Formal Methods, we have developed a prototype of the boundary value test case automatic generation tool BWDM. The main two topics of our tool are (1) automatically generation of test cases and (2) boundary value analysis. Our tool improves the efficiency of software testing process in using VDM++ that is one of the Formal Methods. In this research, we show the structure of our tool, implemented functions, application example, evaluation of the usefulness, relative research, and future issues.

Keywords: Software Testing, Boundary Value Analysis, Formal Methods, VDM++

1. はじめに

ソフトウェアへバグが混入する原因の 1 つとして、上流工程のソフトウェア設計段階において、自然言語を一般的に用いていることが挙げられる。自然言語は元来、曖昧さを含んでいる [1]。そのため、プログラマが、仕様書上の表記を、仕様の作成者が本来意図していない意味で捉えてしまうことが起こる。実装者が、本来の仕様書の意図から外れた認識に基づいて実装を行った結果、ソフトウェアにバグが混入されてしまう。

この問題を解決するための 1 つの方法として、形式手法 [2] (Formal Methods) を用いた上流工程でのソフトウェア設計が挙げられる。形式手法を用いた開発では、まず、数理論理学を基盤とした形式仕様記述言語 (Formal Specification Language) により、開発対象が持つ特性を仕様として記述する。数理論理学を基にしているため、自然言語を用いた開発と異なり、定理証明や機械的な検査を用いて、記述した内容が正しいことを数学的に証明することが可能である。つまり、自然言語の持つ曖昧さを排除した、厳密な仕様を作成することが可能となる。

a)　工学専攻 機械・情報コース大学院生

b)　情報システム工学科准教授

一方で、自然言語もしくは形式手法を用いた設計、いずれにしても、実装を行った後は、作成したソフトウェアに対してテストを行う必要がある。テストを行うためには、テストケースの設計作業が必要であるが、人手によるテストケースの設計には手間と時間がかかる。そのため、テストケースの設計作業を効率よく行うことで、テスト工程を効率化できる。また、バグが潜みうる箇所に的を絞ったテストケースの設計を行うことも、テスト実施の効率化のために重要である。

そこで本研究では、形式手法を用いたソフトウェア開発における、テスト工程の作業効率化を目的として、境界値テストケース自動生成ツール BWDM(Boundary Value/Vienna Develop Method) の試作を行った。本稿では、この BWDM の現状と課題について述べる。BWDM は、VDM++ 仕様を対象として境界値分析を行い、境界値テストを実施するためのテストケースを自動生成する。本稿では、「境界値テストを行う為のテストケース」の意味で、「境界値テストケース」という呼称を用いる。境界値テストケースは、入力データとそれに対する期待出力データによって構成する。入力データとして、VDM 仕様中の関数定義に対して境界値分析を行い、抽出した境界値、及び引数型の最小値と最大値を用いる。

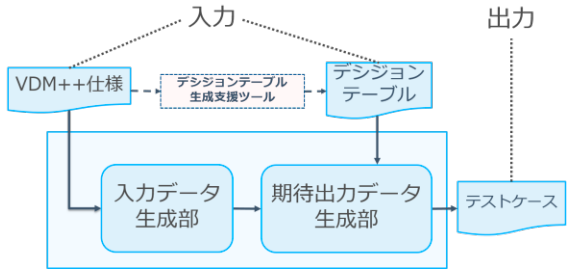

図 1.　BWDM の処理の流れ

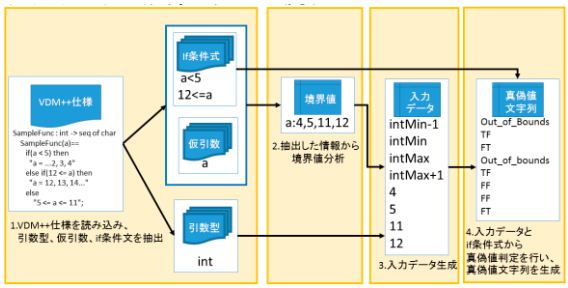

図 2.　入力データ生成部の処理の流れ

2　境界値テストケース自動生成ツール BWDM

本章では、本研究で試作したツールBWDM(Boundary Value/Vienna Development Method) について説明する。ツール名のWはValueとViennaの2つのVを意味する。BWDMは、VDM++仕様において記述した関数定義の境界値分析を行い、境界値テストケースを自動生成する。

図1に、BWDMの処理の流れを示す。BWDMは、入力データ生成部と期待出力データ生成部の2つの処理部で構成している。なお、期待出力データ生成には、本研究室の西川、吉川らが開発した、VDM++を用いたデシジョンテーブル生成支援ツール[3,4] で生成したデシジョンテーブルを利用する。BWDMに対する入力はVDM++仕様、及びデシジョンテーブルである。ユーザーは、事前にVDM++仕様の記述、及びデシジョンテーブル生成支援ツールによるデシジョンテーブルの生成を行っておく。BWDM は、入力であるVDM++仕様に記述した関数定義を境界値分析し、生成した境界値テストケースをtxtファイルとして出力する。

2.1　入力データ生成部の処理の流れ

入力データ生成部の処理の流れを、図2に示す。入力データ生成部では、入力としてVDM++仕様が記述されたファイルを受け取り、境界値テストケースの入力データを生成する。生成した入力データは、期待出力データ生成部で用いる。

2.2　期待出力データ生成部の処理の流れ

期待出力データ生成部の処理の流れを、図3に示す。期待出力データ生成部では、入力データ生成部で生成した入力データを、VDM++仕様中の関数に入力した場合の期待出

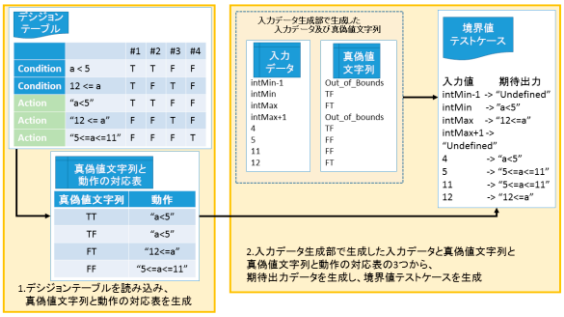

図 3.　期待出力データ生成部の処理の流れ

```
class Mix

functions

混在仕様 : nat * int -> seq of char
  混在仕様(arg1, arg2) ==
    if(arg1 mod 2 = 0) then
      if(0 <= arg2) then
        "arg1:偶数、arg2:正の数"
      else
        "arg1:偶数、arg2:負の数"
    else
      if(arg2 < 0) then
        "arg1:奇数、arg2:負の数"
      else
        "arg1:奇数、arg2:正の数";

end Mix
```

図 4.　不等式と剰余式が混在した仕様

力データを生成する。また、入力データと期待出力データを併せて、境界値テストケースの出力も行う。

3　適用例

本稿で試作したBWDMが正しく動作することを検証するため、不等式と剰余式が混在した仕様をBWDMに適用し、以下の2つの項目を確認する。

- VDM++仕様から境界値を正しく抽出していること
- 入力データと期待出力データを正しく境界値テストケースとして出力していること

不等式と剰余式が混在した仕様を、図4に示す。この仕様は、VDM++仕様を用いて、第一引数が偶数であるか否かと、第二引数が正の数か負の数であるかを判定している。

引数の個数:2

引数型: 第1引数:nat 第2引数:int

テストケースNo.	入力データ		-->	期待出力データ
No.1	natMin-1	intMin-1	-->	Undefined Action
No.2	natMin-1	intMin	-->	Undefined Action
No.3	natMin-1	intMax	-->	Undefined Action
No.4	natMin-1	intMax+1	-->	Undefined Action
No.5	natMin-1	0	-->	Undefined Action
No.6	natMin-1	-1	-->	Undefined Action
No.7	natMin	intMin-1	-->	Undefined Action
No.8	natMin	intMin	-->	arg1:偶数、arg2:負の数
No.9	natMin	intMax	-->	arg1:偶数、arg2:正の数
No.10	natMin	intMax+1	-->	Undefined Action
No.11	natMin	0	-->	arg1:偶数、arg2:正の数
No.12	natMin	-1	-->	arg1:偶数、arg2:負の数
No.13	natMax	intMin-1	-->	Undefined Action
No.14	natMax	intMin	-->	arg1:奇数、arg2:負の数
No.15	natMax	intMax	-->	arg1:奇数、arg2:正の数
No.16	natMax	intMax+1	-->	Undefined Action
No.17	natMax	0	-->	arg1:奇数、arg2:正の数
No.18	natMax	-1	-->	arg1:奇数、arg2:負の数
No.19	natMax+1	intMin-1	-->	Undefined Action
No.20	natMax+1	intMin	-->	Undefined Action
No.21	natMax+1	intMax	-->	Undefined Action
No.22	natMax+1	intMax+1	-->	Undefined Action
No.23	natMax+1	0	-->	Undefined Action
No.24	natMax+1	-1	-->	Undefined Action
No.25	2	intMin-1	-->	Undefined Action
No.26	2	intMin	-->	arg1:偶数、arg2:負の数
No.27	2	intMax	-->	arg1:偶数、arg2:正の数
No.28	2	intMax+1	-->	Undefined Action
No.29	2	0	-->	arg1:偶数、arg2:正の数
No.30	2	-1	-->	arg1:偶数、arg2:負の数
No.31	1	intMin-1	-->	Undefined Action
No.32	1	intMin	-->	arg1:奇数、arg2:負の数
No.33	1	intMax	-->	arg1:奇数、arg2:正の数
No.34	1	intMax+1	-->	Undefined Action
No.35	1	0	-->	arg1:奇数、arg2:正の数
No.36	1	-1	-->	arg1:奇数、arg2:負の数
No.37	3	intMin-1	-->	Undefined Action
No.38	3	intMin	-->	arg1:奇数、arg2:負の数
No.39	3	intMax	-->	arg1:奇数、arg2:正の数
No.40	3	intMax+1	-->	Undefined Action
No.41	3	0	-->	arg1:奇数、arg2:正の数
No.42	3	-1	-->	arg1:奇数、arg2:負の数

図 5. 不等式と剰余式が混在する仕様から生成したテストケース

図4の仕様をBWDMに適用した結果、生成したテストケースを、図5に示す。

図4中の仕様の引数*arg1*から生成する境界値は、*natMin-1, natMin, natMax, natMax+1, 1, 2, 3*である。また、引数*arg2*から生成する境界値は、*intMin-1, intMin, intMax, intMax+1, -1, 0*である。BWDMは、2つの変数それぞれの境界値の組み合わせ全てを入力データとして生成する。よって、この仕様から生成するテストケースは7＊6＝42(件) である。図5から、テストケースはNo.1からNo.42まで生成していることを確認できる。BWDMが、剰余式と不等式が混在した仕様から、正しくテストケースを出力していることがわかる。すなわち、「VDM++仕様から境界値を正しく抽出していること」と「入力データと期待出力データとを正しく境界値テストケースとして出力していること」を確認できる。

表1. テストケース生成にかかった時間

	仕様1	仕様2	仕様3
1回目	325	316	6277
2回目	283	389	5321
3回目	500	371	6236
4回目	291	334	5070
5回目	269	371	5700
平均	334	356	5720

4 考察

本研究では、形式手法を用いたソフトウェア開発における、テスト工程の効率化を目的として、境界値テストケース自動生成ツールBWDMの試作を行った。BWDMは、テストケース自動生成によるテストケース設計作業の効率化、及びVDM++仕様を基に境界値分析を行うことにより、ソフトウェア中のバグの発生しやすい箇所を重点的にテストケース設計を行うことで、テスト実施の効率化を目的としている。以下では、本研究で試作したBWDMについて考察する。

4.1 有用性の評価

BWDMの有用性について考察するために、if条件式の数が違う3つの仕様をBWDMに入力し、テストケースを生成するまでの時間を計測する。表1は、時間を計測した結果である。3つの仕様内には、不等式のif条件式を、仕様1に5個、仕様2に10個、仕様3に15個、それぞれ記述した。実行時間の計測は、JavaのSystem.nanoTimeメソッド[5]を用いて行った。時間の単位はミリ秒であり、それぞれの仕様に対して5回ずつ計測を行い、その平均を取った。なお、計測した時間は、BWDMへ入力としてVDM++仕様とデシジョンテーブルを与え、テストケースを出力し終えるまでの時間である。VDM++仕様やデシジョンテーブルを用意する時間は、今回は考慮していない。

それぞれ仕様毎の結果は、仕様1と仕様2が0.5秒未満、仕様3が6秒未満であった。各仕様毎のデシジョンテーブル上の規則数(デシジョンテーブルの列数) はそれぞれ、仕様1が32、仕様2が1024、仕様3が1048576である。これらは、if条件式内の真偽値によって、関数が取り得る状態の数を表す。仕様内に事前条件等の制約条件が記述されている場合、実際に関数が取り得る状態の数は、この数字より少なくなる。しかし、最大でこれだけの状態を取り得る関数に対して、人手によるテストケースの設計を行うことは、手間と時間がかかる。

それに対してBWDMは、条件数15までの仕様であれば、数秒で境界値テストケースを自動生成することが可能である。以上のことから、VDM++仕様から実装したソフトウェアをテストする際に、BWDMは有用であると言える。

```
if(a mod 5 = 0) then
  if(a > 92) then
    “95, 100, 105, ...” ・・・ 未実行
  else
    “..., 80, 85, 90”     ・・・ 5の場合実行
else
  “others”          ・・・4, 6, 92, 93の場合実行
  現状抽出するテストデータ：4, 5, 6, 92, 93
```

図 5. 不等式と剰余式が混在する仕様から生成したテストケース

4.2 既存の研究との比較

VDM++仕様からテストケースを自動生成法に関する研究の1つに、馬場氏の研究がある[6]。馬場氏の研究では、VDM仕様記述からコーナーケースに対するテストを行うための、テストドライバの雛形の作成を行うツールを開発した。形式手法を用いたソフトウェア開発におけるプロセス中の、単体テストにおけるテストを支援する。

馬場氏のツールは、現状、VDM-SL仕様内の操作定義をテストケース作成の対象としている。そのため、VDM++を対象に、関数定義からテストケースを作成するBWDMとは異なる。また、馬場氏のツールから出力されるテストケースは、入力データとして記述内の条件式を表示するものである。これに対して、BWDMは条件式から生成した、具体的な数値の入力データを確認することが可能である。そのため、BWDMの出力を使ってそのままテストを実行できることが、BWDM の利点であるといえる。

また、VDM++仕様記述をテストするためのテストケース自動生成ツールとして、TOBIAS[7]がある。TOBIASは、TOBIASに入力されたテストパターンに従い、テストケースを自動生成する。テストパターンとは、テスト設計者が正規表現を用いて、出力するテストケースを定義したものである。

TOBIASとBWDMは、どちらもVDM++を用いたソフトウェア開発のテスト段階を支援する。TOBIASは、VDM++で記述された仕様そのもののテスト支援を行う。これに対して、BWDMはVDM++で記述された仕様から実際に実装を行ったソフトウェアへのテストを支援する点で、異なっている。

5 BWDM の課題

BWDM の実装や考察を行う中で明らかになった、BWDM の課題を以下に示す。

- 仕様記述上に現れない境界値への対応

仕様記述上に具体的な数値としては現れないが、複数の条件式が関係しあうことによって発生する境界値があり、現状のBWDMはそれらを入力データとして生成できない。

表2. 図6の出力と満たすべきif条件式

出力	満たすべきif条件式
“95, 100, 105”	a mod 5 = 0 and a > 92
“…, 80, 85, 90”	a mod 5 = 0 and a <= 92
“others”	a mod 5 != 0

- BWDMの適用可能範囲の拡大

現状、BWMDは、関数定義内のif条件式の不等式と剰余式のみに対応しており、if条件式に複数の変数が存在する場合、境界値の抽出を行えない。また、引数に関しては、整数型にのみ対応しており、かつ、引数個が2個以下でないと、VDM++仕様を適用できない。条件式の抽出の際は、andとorで繋がった複合条件式を単純な条件式へと分解する処理を行っていない。これらにより、適用可能なVDM++仕様の範囲に制限がある。

- さまざまな環境への対応

現状のBWDMの型境界値生成機能は、実際の開発に用いられる多様な環境(bit数や開発言語) に対応できていない。

- デシジョンテーブル生成支援ツールとの依存関係

BWDMは現状、テストケース生成に必要なデシジョンテーブルの生成にデシジョンテーブル生成支援ツールを用いているため、依存関係が発生している。

これらのうち、仕様記述上に現れない境界値への対応について、具体的な解決案を示す。図 6 に、実際に問題が起こる if 式の一例を示す。この例では、if 式がネスト構造をとっており、*a mod 5 = 0* という条件式を持つ if 式の処理中でさらに *a > 92* という条件判定を行っている。現状の BWDM の入力データ生成処理部では、これら 2 つの if 条件式に対して境界値分析を行い、入力データとして *4, 5, 6, 92, 93* を生成する。しかし、これら 5 つの入力データのみの場合、3 行目の”*95, 100, 105*”が実行されないため、図 6 の if 式中の全ての出力をテストすることができない。

この問題を解決するため BWDM に、現状の if 条件式からの入力データ生成に加え、各出力行について、その行を実行するために満たすべき if 条件式を満たす整数値を入力データとして生成する処理を追加する。

表 2 に、図 6 において各出力行とその行に至るために満たすべき if 条件式を示す。変数 a について、*a mod 5 = 0* かつ *a > 92*、*a mod 5 = 0* かつ *a <= 92*、*a mod 5 != 0* がそれぞれ真となるような整数値 3 つを入力データとして生成する。

以上の処理を BWDM に実装することで、if 式中の全ての出力に対するテストケース生成が可能になる。

6 おわりに

本研究では、形式手法を用いたソフトウェア開発におけるテスト工程の効率化を目的として、VDM++仕様を対象とする境界値分析を基にしたテストケース自動生成ツールBWDMの試作を行った。適用例として、不等式と剰余式の混在したVDM++仕様を適用した結果、VDM++仕様から境界値を正しく抽出していることを確認した。また、入力データと期待出力データを正しく境界値テストケースとして出力していることを確認した。すなわち、BWDMが正しく動作することを確認した。さらに、BWDMに、if条件式を最大15個含んだ3種類のVDM++仕様を適用した結果、約6秒以内に境界値テストケースを生成した。そのため、BWDMがVDM++仕様を用いたテストケース設計作業の効率化に有用であることを示した。

以上のことから、BWDMを用いることで、形式手法を用いたソフトウェア開発において、テスト工程の作業効率化が見込まれる。また、VDM++仕様を基にしたテスト設計、及びテスト工程を行うことによって、生産性の向上、ならびに、開発したソフトウェアの高品質化も見込まれる。

以下に、今後の課題を示す。

- 仕様記述上に現れない境界値の対応
- BWDMの適用範囲の拡大
- さまざまな環境への対応
- デシジョンテーブル生成支援ツールとの依存関係

参考文献

1) なぜ形式手法か - IPA 独立行政法人 情報処理推進機構,http://sec.ipa.go.jp/users/seminar/seminar_tokyo_20130918-1.pdf, 2017/2/14 アクセス
2) 荒木啓二郎, 張漢明, プログラム仕様記述論, オーム社, 2002 年
3) 吉川祐基, 片山徹郎, VDM++を用いたデシジョンテーブルツールの改良, 宮崎大学 工学部 情報システム工学科 平成 25 年度卒業論文, 2014 年
4) 西川拳太, 片山徹郎, VDM++を用いたデシジョンテーブル生成支援について, 平成 25 年度 電機関係学会九州支部連合大会, 2013 年
5) クラス System, https://docs.oracle.com/javase/jp/6/api/java/lang/System.html, 2017/2/14 アクセス
6) 馬場勇輔, VDM 記述からのテストデータ生成法について, http://aofa.csce.kyushu-u.ac.jp/program-2015-12.php, 2017/2/14 アクセス
7) Olivier Maury, Yves Ledru, Pierre Boundatron, and Lydie du Bousquet, Using TOBIAS for the automatic generation of VDM test cases, http://vasco.imag.fr/Tobias/Papers/vdm02tobias.pdf, 2017/2/14 アクセス