VDM++仕様を対象としたテストケース自動生成ツール BWDM への PICT の適用

平木場　風太 a) ・片山　徹郎 b)

Application of PICT into BWDM which is a Test Case Generation Tool for the VDM++ Specification

Futa HIRAKOBA, Tetsuro KATAYAMA

Abstract

In recent years, specifications using specification language become more important. It is necessary to test the developed software, but it takes much time and effort to design test cases manually. So, we developed BWDM(Boundary Value Vienna Develop Method). BWDM is a test case generation tool for the VDM++ specification. However, the existing BWDM could cause a combinatorial explosion of the generated test cases. There is a pairwise testing as an effective testing method to reduce the total number of the combinations because the pairwise testing only generates test cases which satisfy combinations of two pairs. We apply the pairwise testing into BWDM. In applying the pairwise testing, we use PICT (Pairwise Independent Combinatorial Testing Tool) developed by Microsoft Corporation. However, BWDM cannot call PICT library directly. Hence, we have developed PICT-wrapper. It is an interface to connect PICT and BWDM. We extend BWDM in that PICT-wrapper is embedded. The extended BWDM eliminate the possibility of the combinatorial explosion.

Keywords: software testing, boundary value analysis, pairwise testing, VDM++, PICT

1.　はじめに

社会におけるソフトウェアの重要性は高まっており、形式仕様記述言語を用いた仕様記述が重要視されてきている [1]。作成したソフトウェアにはテストが必要であるが、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、VDM++仕様を対象としたテストケース自動生成を目的としたツール BWDM(Boundary Value/Vienna Develop Method) [2,3] を既に開発した。

既存の BWDM では VDM++仕様を対象として、境界値分析と記号実行を行い、境界値テストと、if-then-else 式の構造認識に基づいたテストを実施するためのテストケースを自動生成する。しかし、既存の BWDM は境界値分析時に全ての組合せを用いてテストケースを生成するため、組合せ爆発を起こす可能性がある。

組合せテストの総数を減らした上で、効果的なテストを行う方法としてペアワイズ法がある。ソフトウェアについて、3 つ以上の因子の組合せで発生する欠陥は、ほとんど存在しないことが知られている [4]。したがって、組合せテストは 2 つの因子の組合せに対して効果的である。（3 つ以上の因子で発生している欠陥を見つけるには、他の方法でテストする必要がある。）

a)　工学専攻 機械・情報系コース 大学院生

b)　情報システム工学科 教授

そして、2 つの因子のみの組合せをテストする手法のことをペアワイズ法と呼ぶ。

そこで本稿では、BWDM が組合せ爆発を起こす可能性の排除を目的として、BWDM によるテストケース生成にペアワイズ法を適用し、BWDM を拡張する。

本稿では、ペアワイズ法を適用するに当たって、Microsoft 社が開発した PICT(Pairwise Independent Combinatorial Testing tool) [5] を利用する。PICT は CLI ツールであるが、API（PICT ライブラリと呼称する）も提供しており、C++から利用できる。しかし、既存の BWDM は Java で記述していることから、PICT ライブラリを呼び出すことができない。そこで、PICT と BWDM を接続するためのインタフェースとして、PICT ラッパーを開発した。 そして、既存の BWDM に PICT ラッパーを埋め込むことで、BWDM を拡張した。

2.　既存の BWDM

既存の BWDM には以下の機能がある。

- 記号実行によるテストケース生成 [2]
- 境界値分析によるテストケース生成 [3]

記号実行によって生成したテストケースは、全ての実行

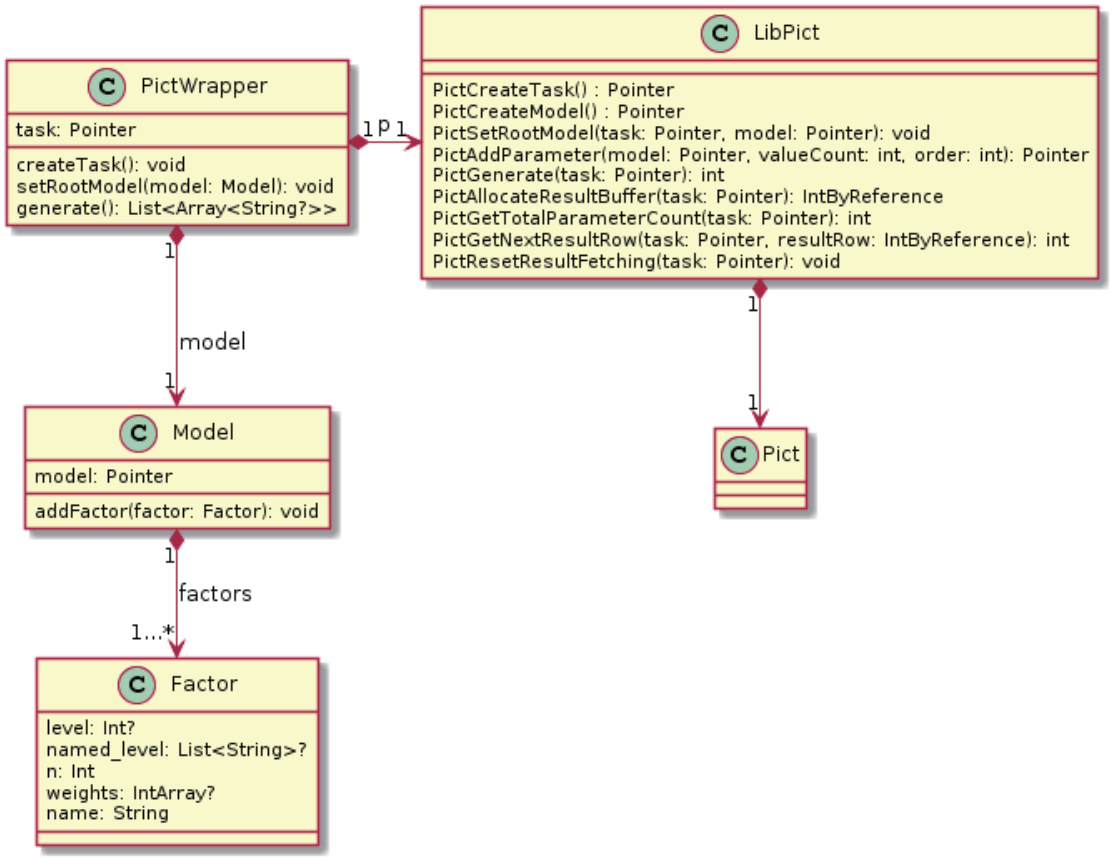

図 1: PICT ラッパーのクラス図.

フローを網羅できることが期待できる。境界値分析によって生成したテストケースは、境界値テストに使用することができる。既存の BWDM を使用することにより、VDM++ 仕様を用いたソフトウェア開発効率を改善できる。

しかし、既存の BWDM には問題がある。境界値分析によるテストケース生成において、生成したテストケース数は因子が取り得るそれぞれの値の数を掛け合わせることにより決定する。たとえば、（6、6、2、4、5、7）の因子の場合、既存の BWDM は、6×6×2×4×5×7＝10,080 のテストケースを生成する。したがって、組合せ爆発を起こす可能性がある。本稿では、この問題を解決するために、既存の BWDM を拡張する。

3. BWDM の拡張

本稿では PICT ラッパーを開発した。PICT ラッパーは PICT と BWDM を接続するためのインタフェースである。そして、既存の BWDM に PICT ラッパーを埋め込むことで、BWDM を拡張した。詳細を以下に示す。

3.1 PICT ラッパー

PICTはCLIツールであるが、API（PICTライブラリと呼称する）も提供しており、C++から利用できる。しかし、既存のBWDMはJavaで記述していることから、PICTライブラリを呼び出すことができない。そのため、BWDMの拡張の準備として、JNA(Java Native Access) [6] を利用し、Javaから呼び出すことのできるPICTライブラリ（PICTラッパーと呼称する）を作成した。

図1に、PICTラッパーのクラス図を示す。それぞれのクラスの説明を、以下に示す。

- クラスPict
  Microsoft 社が開発したC++で記述されたPICTライブラリである。
- クラスLibPict
  JNAを用いてJavaで記述したPICTライブラリのインタフェースである。メソッド名は全てPICTライブラリの持つ関数名と同じである。
  主に使用するPICTの関数を、以下に示す。
  - PictAddParameter － PICTへの因子と水準の登録
  - PictGenerate － 組合せデータの生成
  - PictGetNextResultRow － 生成データの取得
- クラスPictWrapper
  PICTを操作するためのクラスである。Kotlin [7] で記述する。以下の機能を持つ。
  - createTask - Taskの生成と初期化をする。Taskは、PICTの組合せ生成処理の最小単位である。PICTライブラリにおけるPictCreateTaskに相当する。
  - setRootModel － TaskにModelの登録をする。Modelは因子の集合である。PICTライブラリにおけるPictSetRootModelに相当する。
  - generate - ペアワイズ法を適用した組合せの生

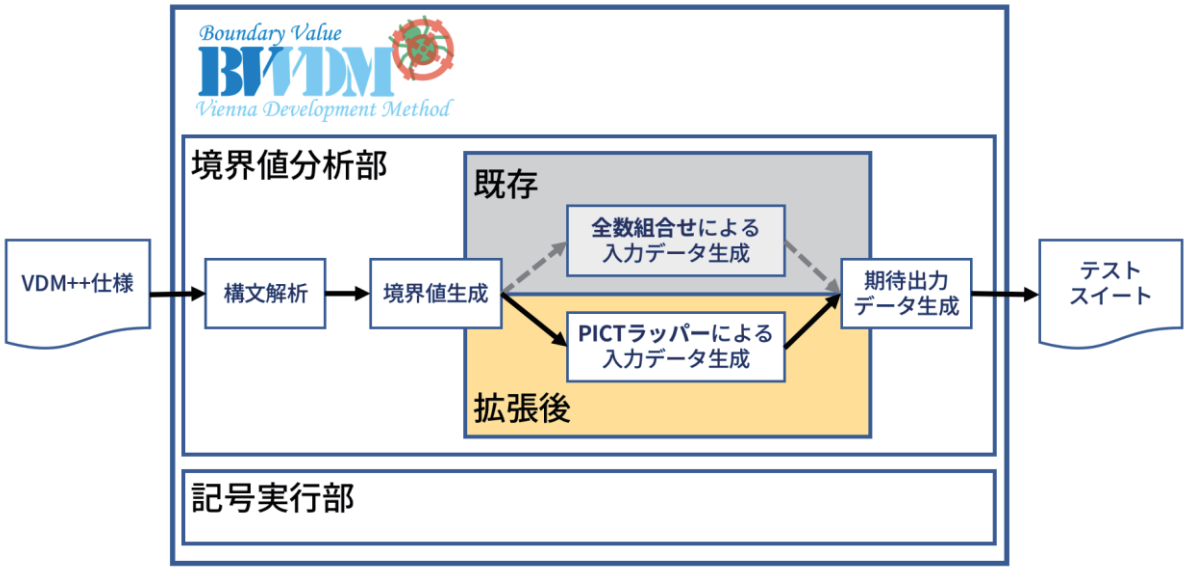

図 2: 拡張後の BWDM の処理の流れ.

成をする。PICTライブラリにおけるPictGenerateに相当する。

- クラスModel
  因子の集合を保持するためのクラスである。Kotlinで記述する。
  - コンストラクタ - PICTライブラリにおけるModelを生成する。PictCreateModelに相当する。
  - addFactor - 因子(Factor)をModelに登録する。PICTライブラリにおけるPictAddParameterに相当する。
- クラスFactor
  因子を表すクラスである。Kotlinで記述する。保持する情報を下記に示す。
  - level - 水準
  - named_level - 因子の取り得る値の集合
  - n - 最低限組合せるペア数(デフォルトで2)
  - weights - 因子の取り得る値の重みの集合
  - name - 因子の名前

  メンバ変数nの値を変えることにより、その因子については、n個の組合せを網羅する入力データを作成することができる。

3.2 ペアワイズ法の適用

本稿で拡張したBWDMの処理の流れを、図2に示す。境界値分析部ではまず、テストケースの入力データとして、VDM++仕様内の引数毎における、不等式、剰余式などに合わせた境界値、及び型の最小値・最大値の境界値を、それぞれ抽出する。既存のBWDMでは、引数毎に生成した境界値の全ての組合せを生成し、境界値テストの入力データとしていた。

拡張後のBWDMでは、全ての組合せを生成するのではなく、3.1節で作成したPICTラッパーを用いてペアワイズ法を適用し、入力データ生成を行う。具体的には、境界値分析で得た因子が取り得る値をPICTラッパーに入力する。

境界値分析後の境界値データを受けとったPICTラッパーの入力データ生成アルゴリズムを、以下に示す。また、このアルゴリズムを用いたPICTラッパーの処理の流れを、図3に示す。

A) 因子と、因子の取り得る値を元に、クラスFactorのインスタンスを因子の数だけ生成する。
B) クラスModelのaddFactorメソッドを用いて、PictAddParameter関数を呼び出し、PICTに因子と因子ごとの水準を登録する。
C) クラスPictWrapperのgenerateメソッドを用いて、ペアワイズ法を適用した組合せデータのリストを生成する。組合せデータは文字列型の配列のリストである。詳細の処理を以下に示す。
   (ア) PictGenerate関数を用いて、PICTにペアワイズ法を適用した組合せデータを生成させる。
   (イ) PictGetNextResultRow関数を用いて(ア)で生成した組合せデータを1件取得する。取得したデータは、因子が取り得るパラメータ群のインデックスとなる。因子の数だけ(イ)の処理を繰り返す。
   (ウ) (イ)で取得したデータのインデックスに該当するパラメータを用いて、組合せデータのリストを生成する。
D) C)で生成したリストを、PICTラッパーの出力データとする。

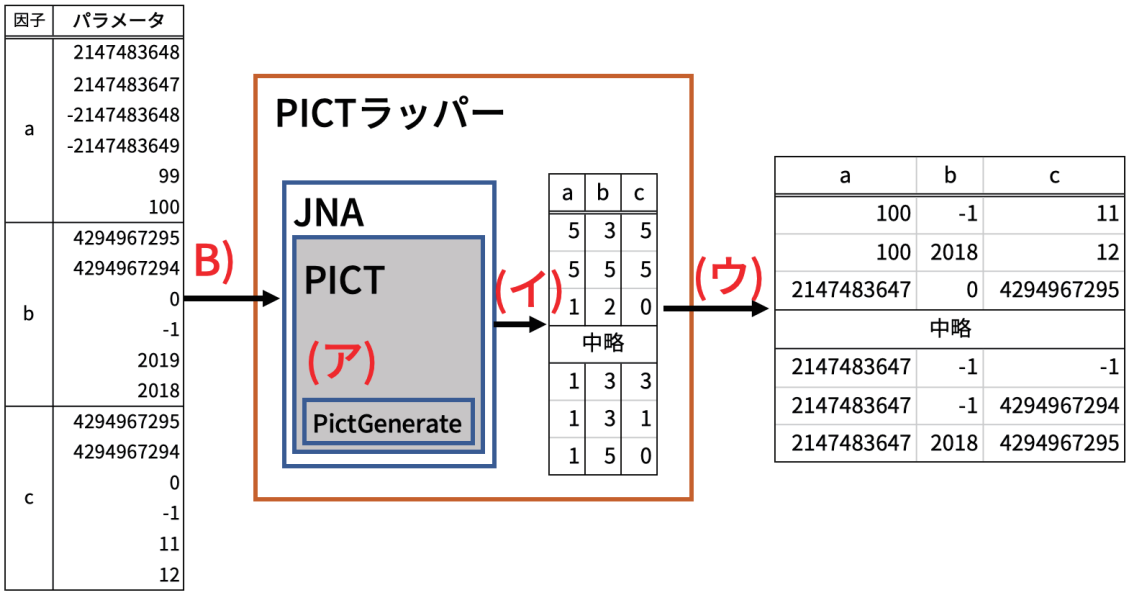

図 3: PICT ラッパーの処理の流れ.

表 1: 適用結果.

No	入力			期待出力
1	100	-1	11	Undefined Action
2	100	2,018	12	“a は 100 以上かつ c は 12 以上”
3	2,147,483,647	0	4,294,967,295	Undefined Action
~	~	~	~	~
38	2,147,483,647	-1	-1	Undefined Action
39	2,147,483,647	-1	4,294,967,294	Undefined Action
40	2,147,483,647	2,018	4,294,967,295	Undefined Action

```
class sampleClass

functions

sampleFunction : int*nat*nat -> seq of char
sampleFunction(a, b, c) ==
    if(a < 100) then
      if(b > 2018) then
        "aは100未満かつbは2018より大きい"
      else
        "aは100未満かつbは2018以下"
    elseif(c < 12) then
      "aは100以上かつcは12未満"
    else
      "aは100以上かつcは12以上";

end sampleClass
```

図 4: 因子が 3、水準が(6, 6, 6)の関数を持つ VDM++仕様.

4. 適用例

本稿で拡張したBWDMが正しく動作することを検証するため、拡張したBWDMにVDM++仕様を適用した。適用結果を、表1に示す。適用した、因子が3、境界値分析後の水準がそれぞれ(6, 6, 6)の関数のVDM++仕様を、図4に示す。

この適用例から、既存のBWDMでは216個生成していたテストケースを、拡張後のBWDMでは40個の生成に抑えており、かつ、40個のテストケースは、2個の因子のペアの組合せをすべて網羅できていることが確認できた。すなわち、拡張したBWDMが、VDM++仕様から、ペアワイズ法を適用した境界値テストケースを正しく出力できていることが確認できた。

5. 評価

本稿で拡張したBWDMが、既存のBWDMに比べて、テストケース総数を削減できることを確認する。膨大な数のテストケースを生成するために、因子が7、境界値分析後

表 2: 生成結果の比較.

	生成テストケース数	実行時間（秒）
既存の BWDM	663,552	6.767146152
拡張後の BWDM	78	0.88973152

```
class ProblemClass

  functions

  problemFunction : nat*nat*nat*nat*nat*nat*nat -> seq of char
    problemFunction(a, b, c, d, e, f, g) ==
      if(a > 4) then
        if(b mod 10 = 3) then
          if(c < 13) then
            if(b>11) then
              "a>4 and b>11 and c<13"
            else
              if(e<4) then
                "e<4"
              else
                if(g<11) then
                  "g<11"
                else
                  "a>4 and b>10 and c<13"
          else
            if(d>10) then
              "d>10"
            else
              if(e<10) then
                "e<10"
              elseif(f>10) then
                "f>10"
              else
                "a>4 and b>10 and c>=13"
        else
            "a>4 and b<=10"
      else
        "a<=4";

end ProblemClass
```

図 5: 因子が 7、水準が(6, 8, 6, 8, 8, 6, 6)の関数を持つ VDM++仕様.

の水準がそれぞれ(6, 8, 6, 8, 8, 6, 6)の関数を持つVDM++仕様を、既存のBWDMと拡張後のBWDMにそれぞれ適用する。生成結果の比較を、表2に示す。また、適用したVDM++仕様を図5に示す。実行環境は、macOS 10.13.6(CPU: Intel Core i5 2.3GHz, RAM: 16GB)である。比較に用いる式を、以下に示す。

$$削減率(\%) = \frac{A - B}{A} \times 100 \quad (1)$$

ここで、
A: 既存のBWDMによって生成したテストケース総数
B: 拡張したBWDMによって生成したテストケース総数
とする。
表2および式(1)より、生成テストケース数を$(663552 - 78)/663552 \times 100 = 99.98(\%)$削減できた。既存のBWDMでは、膨大な数のテストケースを生成したが、拡張後のBWDMでは、実用的な数のテストケースを生成した。したがって、拡張後のBWDMは、境界値分析結果から生成するテストケース数が組合せ爆発を起こす可能性を排除できたと言える。また、表2から、テストケース生成時間についても短縮できた。以上から、拡張後のBWDMは実用性が高いと言える。

6. おわりに

本稿では、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 における、境界値分析結果から生成するテストケース数が組合せ爆発を起こす可能性の排除を目的として、BWDM の拡張を行った。拡張した BWDM は、VDM++仕様に対して境界値分析を行い、ペアワイズ法を適用し、テストケースを自動生成する。これにより、境界値分析結果から生成するテストケース数が組合せ爆発を起こす可能性を排除することができ、テスト工程の作業効率化を見込めると考えられる。さらに、拡張後の BWDM は実用性が高いと言える。

以下に、拡張した BWDM の今後の課題を示す。

- 適用可能なVDM++構文の対応範囲の拡張

  BWDMは適用可能なVDM++構文の対応範囲が狭いため、BWDMに適用することができないVDM++仕様が存在する。より多くのVDM++仕様を適用可能にし、BWDMの実用性を向上させるために、VDM++構文の対応範囲を拡大していく必要がある。これについては、以下の項目を考えている。

  - 整数型以外の型への対応

    BWDMは、整数を表すint型しか対応できていない。すなわち、実数を表すreal型、有理数を表すrat型、複数の型から構成する合成型などの、VDM++の多くの型に未対応である。これについては、VDM++仕様を静的解析する際に、型情報を読み込み、境界値分析時に、読み込んだ型情報から境界値の生成処理を切り替えることによって、対応可能であると考える。
  - 関数定義以外の定義部への対応

    BWDMはVDM++仕様の関数定義部のみから境界値の抽出を行っている。これについては、操作定義部や、型定義部などの、使用上の他の定義部に対しても静的解析を行い、それらの定義に対する新たな境界値分析手法を考案することで、対応可能と考える。

- 制約条件の設定への対応

  拡張したBWDMは、PICTライブラリの機能の一部のみしか使用できない。例えば、拡張したBWDMでは、特定の組合せの制約を設定できない。特定の組合せとは、必ずテストしなければならない組合せ、または、テストする必要がない組合せを意味する。

これについては、PICTラッパーを拡張し、PICTライブラリの制約条件を設定する機能に対応することと、VDM++仕様の事前条件(pre句)を静的解析し、解析した条件式から生成した組合せを、PICTラッパーに入力することで、対応可能と考える。

- ドメインテストに基づいたテストケースの自動生成

同値分割法、境界値分析において、複数の変数同士に依存関係がある場合、それを考慮したテストをドメインテストと呼ぶ[8]。BWDMはドメインテストに基づいたテストケースを自動生成することができない。ドメインテストを自動で行う手法として、丹野氏、張氏が提案した、ドメインテスト技法に基づく網羅的なテストデータ自動生成手法[9]がある。この手法では、仕様書を元に、設計モデルと呼ばれる、入力変数とドメインの定義を人手で作成する。そして、記述した設計モデルから、ドメインテストに基づいたテストケースを自動生成する。そのため、設計モデルを人手で作成しなければならないという手間が残る。

この手法を、BWDMに適用することで、ドメインテストに基づいたテストケースの自動生成が可能と考える。また、BWDMが対象とするVDM++は形式仕様であることから、設計モデルを自動生成することが可能であり、設計モデルを人手で作成しなければならないという手間がなくなる。

参考文献

1) Anne E. Haxthausen: An introduction to formal methods for the development of safety-critical application, DTU Informatics Technical University of Denmark, pp.1-32, 2010.
2) 立山 博基, 片山 徹郎: VDM++仕様を用いたテストケース自動生成ツール BWDM における if-then-else 式の構造認識手法の提案, ソフトウェアエンジニアリングシンポジウム2017論文集, pp.130-137, 2017.
3) Hiroki Tachiyama, Tetsuro Katayama, et al.: Prototype of Test Cases Automatic Generation Tool BWDM Based on Boundary Value Analysis with VDM++, International Conference on Artificial Life and Robotics, pp.275-278, 2017.
4) D. Richard Kuhn, Dolores R. Wallace and Albert M. Gallo: Software fault interactions and implications for software testing, IEEE Transactions on Software Engineering, Vol. 30, pp.418-421, 2004.
5) Microsoft Corp.: Microsoft/pict Pairwise Independent Combinatorial Testing, https://github.com/Microsoft/pict, last accessed on 2019/02/15.
6) java-native-access/jna Java Native Access, https://github.com/java-native-access/jna, last accessed on 2019/02/15.
7) JetBrains: Kotlin Programming Language, https://kotlinlang.org/, last accessed on 2019/02/15.
8) Lee Copeland, 宗 雅彦: はじめて学ぶソフトウェアのテスト技法, 日経BP社, 2005.
9) 丹野 治門, 張 暁晶: ドメインテスト技法に基づく 網羅的なテストデータ自動生成手法の提案, 研究報告ソフトウェア工学 (SE), Vol.2014-SE-186, No.6, pp.1-8, 2014.