VDM++仕様を対象にしたテストケース自動生成ツールBWDMにおけるif式の構造認識に基づいたテストケース生成手法の提案

立山 博基
宮崎大学大学院工学研究科
tachiyama@earth.cs.miyazaki-u.ac.jp

片山 徹郎
宮崎大学工学教育研究部
kat@cs.miyazaki-u.ac.jp

要旨

ソフトウェア開発において、形式手法を用いることで、仕様に曖昧さが含まれることを防ぐことができる。我々は、形式手法 *VDM* 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、テストケース自動生成ツール *BWDM* を試作した。既存の *BWDM* の問題点として、*VDM* 仕様中の *if* 式からテストケース生成を行う際に、ネストや *else if* などの *if* 式同士の構造を認識していない点が挙げられる。そのため、そのような構造の中にある戻り値に対するテストケース生成を行えない。本稿では、この *BWDM* における、テストケース生成処理の問題点を解決するために、*if* 式の構造認識に基づいたテストケース生成手法の提案を行う。提案手法を *VDM* 仕様に適用した結果、従来の *BWDM* ではテストケース生成を行えなかった仕様中の戻り値に対する、テストケースの生成が可能になることを確認した。そのため、*BWDM* に提案手法を実装することで、テストケース生成処理の問題点を解決し、*BWDM* の有用性が向上すると言える。

1. はじめに

ソフトウェアは年々大規模化かつ高機能化を続け、その結果、ソフトウェア上のバグが社会にもたらす影響も近年では甚大なものとなっている [1]。

ソフトウェアに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 [2]。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、仕様書の本来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 つの手段として、形式手法 (Formal Methods)[3] が提案されている。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。

一方で、自然言語もしくは形式手法を用いた設計のいずれにしても、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。

このような問題とソフトウェアテストの必要性を踏まえ、形式仕様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 [4, 5, 6]。BWDM は、形式仕様記述言語 VDM++で記述した仕様とデシジョンテーブルを入力とし、VDM++仕様を基に境界値分析を行い、テストケースを自動生成する。デシジョンテーブルとは、ソフトウェアの入力に対する出力を表形式でまとめたものである [7]。ソフトウェアに対してテストを行う際に、

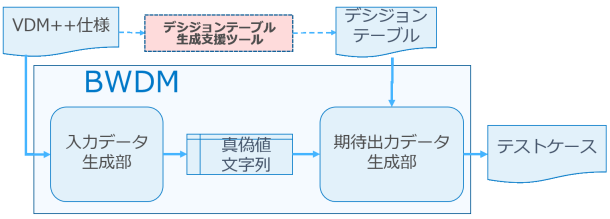

図 1. BWDM の構成

ソフトウェアの期待出力を確認する際に有用である。

既存の BWDM の問題点として、VDM++仕様中の if 式からテストケース生成を行う際に、ネストや else if などの if 式同士の構造を認識していない点が挙げられる。そのため、そのような構造の中にある戻り値に対するテストケース生成を行えない。

そこで本稿では、この問題の解決を目的として、if 式の構造認識に基づいたテストケース生成手法を提案する。具体的には、if 式の構造を認識し、条件式を生成し、それを基にテストケースを生成することによって、既存の問題点の解決を目指す。

2. テストケース自動生成ツール BWDM

本章では、既存のテストケース自動生成ツール BWDM の構成、機能、処理、入出力、及び、問題点について記す。BWDM とは、Boundary Value と Vienna Development Method の頭字語である。W は Value と Vienna の 2 つの V を意味する。

2.1. BWDM の構成・機能・処理

図 1 に、BWDM の構成を示す。BWDM は入力データ生成部と期待出力データ生成部によって構成しており、それぞれの役割は以下の通りである。

- 入力データ生成部
  - VDM++仕様読み込み
  - 構文解析
  - データ抽出
  - 境界値分析
  - 入力データ生成

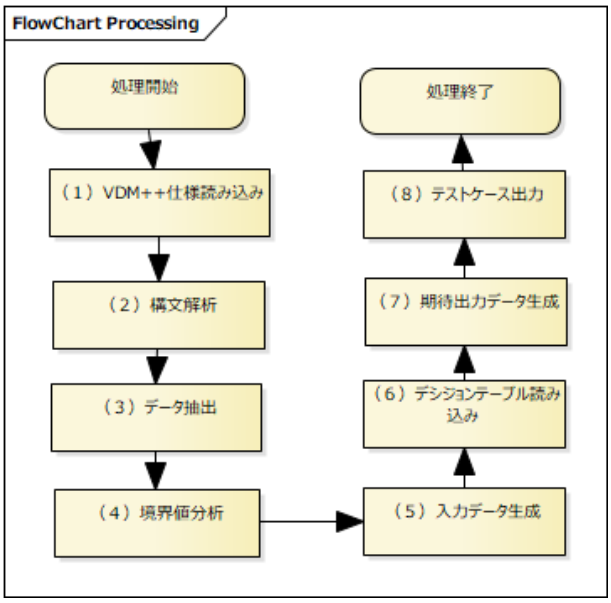

図 2. BWDM の処理の流れ

- 期待出力データ生成部
  - デシジョンテーブル読み込み
  - 期待出力データ導出
  - テストケース生成

BWDM が持つ機能は、以下の通りである。

- VDM++仕様の構文解析及び情報の抽出
- 抽出した情報を基にした境界値分析
- 境界値分析を行った結果を基にした入力データ生成
- デシジョンテーブルを基にした入力データに対応する期待出力データの導出
- テストケースの出力 (入力データ＋期待出力データ)

図 2 に、BWDM の処理のフローチャートを示す。処理は以下の流れで行われる。

1. 指定された VDM++仕様ファイルを読み込む
2. VDM++仕様ファイルを構文解析する
3. 関数定義中の引数型と if 式中の条件式に関する情報を抽出する
4. 抽出した情報に対して境界値分析を行う

```
1 class SampleClass
2
3 functions
4
5 sampleFunction : int*nat − > seq of char
6   sampleFunction(a, b) ==
7   if(a <= 10) then
8     if(-1 > a) then
9       “a <= 10 and -1 > a”
10    else if(b > 0) then
11      “a <= 10 and !(-1 > a) and b > 0”
12    else
13      “a <= 10 and !(-1 > a) and !(b > 0)”
14  else
15    if(b <= -3) then
16      “!(a <= 10) and b <= -3”
17    else if(a < 20) then
18      “!(a <= 10) and !(b <= -3) and a < 20”
19    else
20      “!(a <= 10) and !(b <= -3) and !(a < 20)”;
21
22 end SampleClass
```

図 3. 入力例：VDM++仕様

5. 境界値分析の結果から入力データを生成する
6. デシジョンテーブルを読み込む
7. 入力データとデシジョンテーブルから期待出力データを導出する
8. 入力データと期待出力データをテストケースとして併せて出力する

構文解析は、Nick Battle 氏の開発した VDM 静的解析ツール VDMJ を用いて実現している [8]。境界値分析は、抽出した引数型、if 式中の条件式のそれぞれに対して行い、入力データを生成する。

BWDM は期待出力の導出を、入力データとデシジョンテーブルの照らし合わせによって行う。この際に使用するデシジョンテーブルは、本研究室で開発したデシジョンテーブル生成支援ツール [9] によって生成したものである。最終的に、テストケースを csv(comma-separated values) 形式でファイルに書き出す。

引数の個数:2				
引数型:	第1引数:int	第2引数:nat		
テストケースNo.	入力データ		-->	期待出力データ
No.1	intMin−1	natMin−1	-->	Undefined Action
No.2	intMin−1	natMin	-->	Undefined Action
No.3	intMin−1	natMax	-->	Undefined Action
No.4	intMin−1	natMax+1	-->	Undefined Action
No.5	intMin−1	1	-->	Undefined Action
No.6	intMin−1	−3	-->	Undefined Action
No.7	intMin−1	−2	-->	Undefined Action
No.8	intMin	natMin−1	-->	Undefined Action
No.9	intMin	natMin	-->	aは10以下で、bは0以下です
No.10	intMin	natMax	-->	aは10以下で、bは0より大きいです
No.11	intMin	natMax+1	-->	Undefined Action
No.12	intMin	1	-->	aは10以下で、bは0より大きいです
No.13	intMin	−3	-->	aは10以下で、bは0以下です
No.14	intMin	−2	-->	aは10以下で、bは0以下です
No.15	intMax	natMin−1	-->	Undefined Action
No.16	intMax	natMin	-->	aは20以上で、bは−3より大きいです
No.17	intMax	natMax	-->	aは20以上で、bは−3より大きいです
No.18	intMax	natMax+1	-->	Undefined Action
No.19	intMax	1	-->	aは20以上で、bは−3より大きいです
No.20	intMax	−3	-->	aは10より大きく、bは−3以下です
No.21	intMax	−2	-->	aは20以上で、bは−3より大きいです
No.22	intMax+1	natMin−1	-->	Undefined Action
No.23	intMax+1	natMin	-->	Undefined Action
No.24	intMax+1	natMax	-->	Undefined Action
No.25	intMax+1	natMax+1	-->	Undefined Action
No.26	intMax+1	1	-->	Undefined Action
No.27	intMax+1	−3	-->	Undefined Action
No.28	intMax+1	−2	-->	Undefined Action
No.29	10	natMin−1	-->	Undefined Action
No.30	10	natMin	-->	aは10以下で、bは0以下です
No.31	10	natMax	-->	aは10以下で、bは0より大きいです
No.32	10	natMax+1	-->	Undefined Action
No.33	10	1	-->	aは10以下で、bは0より大きいです
No.34	10	−3	-->	aは10以下で、bは0以下です
No.35	10	−2	-->	aは10以下で、bは0以下です
No.36	11	natMin−1	-->	Undefined Action
No.37	11	natMin	-->	aは10より大きく20未満で、bは−3より大きいです
No.38	11	natMax	-->	aは10より大きく20未満で、bは−3より大きいです
No.39	11	natMax+1	-->	Undefined Action
No.40	11	1	-->	aは10より大きく20未満で、bは−3より大きいです
No.41	11	−3	-->	aは10より大きく、bは−3以下です
No.42	11	−2	-->	aは10より大きく20未満で、bは−3より大きいです
No.43	19	natMin−1	-->	Undefined Action
No.44	19	natMin	-->	aは10より大きく20未満で、bは−3より大きいです
No.45	19	natMax	-->	aは10より大きく20未満で、bは−3より大きいです
No.46	19	natMax+1	-->	Undefined Action
No.47	19	1	-->	aは10より大きく20未満で、bは−3より大きいです
No.48	19	−3	-->	aは10より大きく、bは−3以下です
No.49	19	−2	-->	aは10より大きく20未満で、bは−3より大きいです
No.50	20	natMin−1	-->	Undefined Action
No.51	20	natMin	-->	aは20以上で、bは−3より大きいです
No.52	20	natMax	-->	aは20以上で、bは−3より大きいです
No.53	20	natMax+1	-->	Undefined Action
No.54	20	1	-->	aは20以上で、bは−3より大きいです
No.55	20	−3	-->	aは10より大きく、bは−3以下です
No.56	20	−2	-->	aは20以上で、bは−3より大きいです

図 4. 出力例：テストケース

BWDM は現状、整数値の入力データ生成のみの対応であるため、以降の説明で登場する変数の型は全て整数型であるとする。また、テストケース生成処理は、関数定義中の引数型と if 式のみを対象に行う。

2.2. BWDM の入出力

BWDM の入出力に関して、VDM++仕様ファイルを図 3 に、テストケースファイルを図 4 に、それぞれ示す。

入出力に関係するファイルは以下である。

- 入力
  - VDM++仕様ファイル (テキスト形式)

```
1 class ProblemClass
2
3 functions
4
5 problemFunction : nat -> seq of char
6   problemFunction(a) ==
7     if(a mod 4 = 0) then
8       if(a > 92) then
9         “a mod 4 = 0 and a > 92”
10       else
11         “a mod 4 = 0 and !(a > 92)”
12     else
13       “!(a mod 4 = 0)”;
14
15 end ProblemClass
```

図 5. 現状の課題：ネストした if 式

  - デシジョンテーブルファイル (csv 形式)

- 出力
  - テストケースファイル (csv 形式)

図 4 の入力データ中に存在する $intMin, intMin - 1, intMax, intMax + 1$ などは、図 3 中の関数定義における引数型 (int) に対して境界値分析を行い、生成した入力データである。それぞれ、「引数型 int の最小値 (Minimum)」、「引数型 int 型の最小値より 1 小さい値」、「引数型 int の最大値 (Maximum)」「引数型 int の最大値より 1 大きい値 (Maximum)」を表す。引数型の範囲外の値 (*Min-1, *Max+1) については、桁溢れにより、プログラムの動作が予測不可能であるため、期待出力データは Undefined Action としている。

2.3. BWDM の問題点

既存の BWDM は、構文解析により VDM++仕様中の if 式の抽出を行っている。しかし、ネスト構造や else if などの構造は無視している。それらの構造を持つ if 式が仕様中に存在する場合も、個々の if 式として抽出し、それら 1 つ 1 つに対して境界値分析を行い、入力データを生成している。そのため、複数の if 式が関わりあった構造の中にある戻り値に対する入力データの生成は行えない。

引数の個数:1			
引数型:	第1引数:nat		
テストケースNo. 入力データ --> 期待出力データ			
No.1	natMin-1	-->	Undefined Action
No.2	natMin	-->	a mod 4 = 0 and !(a > 92)
No.3	natMax	-->	!(a mod 4 = 0)
No.4	natMax+1	-->	Undefined Action
No.5	4	-->	a mod 4 = 0 and !(a > 92)
No.6	3	-->	!(a mod 4 = 0)
No.7	5	-->	!(a mod 4 = 0)
No.8	93	-->	!(a mod 4 = 0)
No.9	92	-->	a mod 4 = 0 and !(a > 92)

図 6. 図 5 から BWDM で生成したテストケース

図 5 と図 6 に、現状、問題となる if 式、及び、その if 式から生成したテストケースを示す。図 5 には、a mod 4 = 0 と $a > 92$ の整数値 a に関する 2 つの if 式がネストしており、戻り値は文字列である。文字列中には、その戻り値を返す際に a が満たすべき条件式を表している。

図 6 の期待出力データ中に戻り値 “a mod 4 = 0 and a > 92” に対するテストケースが存在しないことが確認できる。BWDM は境界値分析による入力データ生成処理により、1 つ目の a mod 4 = 0 からは 3, 4, 5、2 つ目の $a > 92$ からは 92, 93 を入力データとして生成するが、それらの中には、“a mod 4 = 0 and a > 92” を出力する入力データは存在していない。戻り値 “a mod 4 = 0 and a > 92” を出力するためには、a mod 4 = 0 and $a > 92$ を満たす入力データを生成する必要がある。

このような if 式による構造関係を認識し、その先の戻り値に対する入力データを生成できるように、BWDM を改良する必要がある。

3. if 式の構造認識に基づいたテストケース生成手法について

本章では、2.3 節で示した問題点を改良するための手法、及び、BWDM 上でその処理を行うための改良方法について述べる。

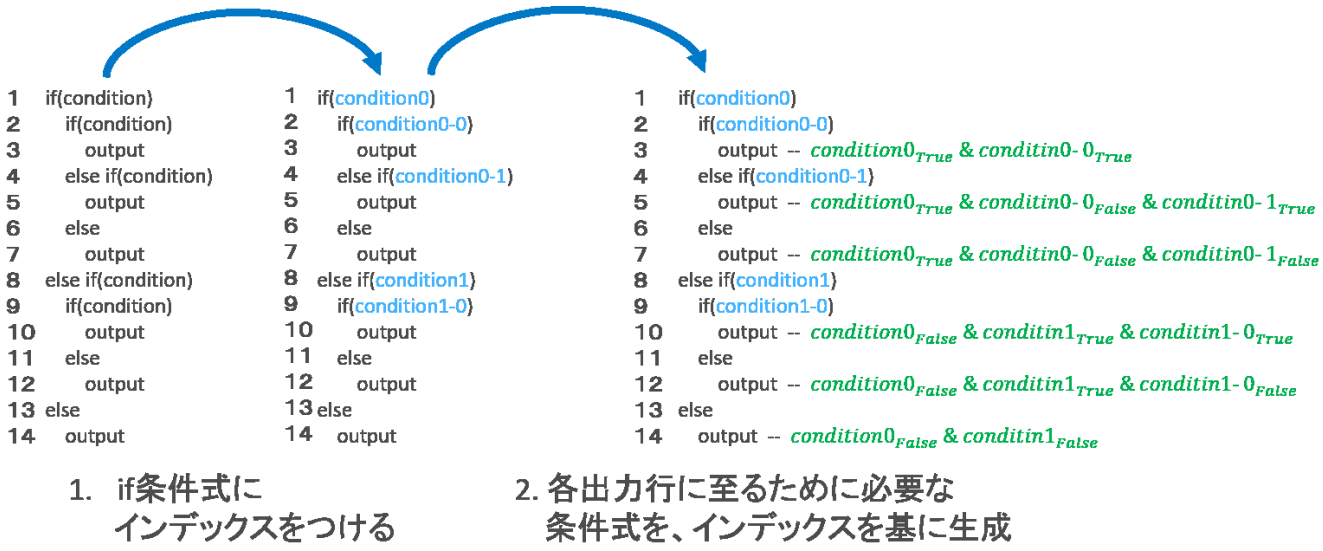

図 7. if 式へのインデックス付け及び満たすべき条件式の生成

表 1. 図 5 における戻り値及びその際満たすべき条件式と真偽値

戻り値	条件式
“a mod 4 = 0 and a > 92”	a mod 4 = 0, a > 92
“a mod 4 = 0 and !(a > 92)”	a mod 4 = 0, a <= 92
“!(a mod 4 = 0)”	a mod 4 != 0

3.1. 提案手法

if 式の各出力行へ至るためには、各出力行へ至る条件式を満たす入力データを生成すればよい。表 1 に、2.3 節で用いた図 5 において、各出力行へ至るために満たすべき条件式を示す。

1 つ目と 2 つ目の出力においては、if 条件式がネストしているため、満たすべき if 条件式は and で繋がったものとなる。また、else 節の後にある出力は、記述してある if 条件式を否定した条件式を満たす必要がある。

次節から、提案手法を実現するための具体的な処理の流れを述べる。

3.2. if 式の構造の認識及び満たすべき if 条件式の生成

提案手法の例を、図 7 に示す。この例を基に、提案手法の説明を行う。手順は、if 式に対するインデックス付けと条件式の生成の 2 つである。

1. if 式へのインデックス付け (図 7 中の手順 1.)

   if 式の構造を認識するために、構文解析を行った後に、各 if 式へインデックスを付ける。インデックスにより、ネストや else if などの構造と、各戻り値に対する条件式を確認できるようにする。図 7 中には、if 条件式が 5 つ存在している。ネスト構造の中に存在している条件式 (図 7 中の $condition0-0, condition0-1, condition1-0$) は、親の条件式 (図 7 中の $condition0, condition1$) のインデックスの数字の後に、更にインデックスとして数字を割り振る。

2. 条件式の生成 (図 7 中の手順 2.)

   各出力行へ至る入力データについて、if 条件式のインデックスを辿りながら、入力データが満たすべき条件式を生成する。インデックスを辿る方法は複数の手法が考えられるが、今回は各戻り値から条件式を辿る。各戻り値ごとに、if 式のインデックスを利用して、その戻り値に至るために満たすべき条件式を導出する。

引数の個数:1			
引数型:	第1引数:nat		
テストケースNo. 入力データ --> 期待出力データ			
No.1	3424	-->	"a mod 4 = 0 and a > 92"
No.2	52	-->	"a mod 4 = 0 and !(a > 92)"
No.3	1217	-->	"!(a mod 4 = 0)

図 8. 図 5 に提案手法を適用して生成したテストケース

5 行目の output の出力に必要な条件式を具体例とすると、まず、4 行目の $condition0-1$ を満たす必要がある。更に、$condition0-1$ は else if の条件式であるため、2 行目の $condition0-0$ は否定する必要がある。そして、1 行目と 2 行目で if 式がネストしているため、1 行目の $condition0$ は満たす必要がある。

そのため、必要となる条件式は、$condition0-1_{True}$ かつ $condition0-0_{False}$かつ $condition0_{True}$ となる。

なお、ここでの $condition*_{True}, condition*_{False}$ とは、$a > 92$ という if 条件式を例にすると、$condition_{True}$ が $a > 92$、$condition_{False}$ は、$\overline{condition_{True}}$ の $a <= 92$ となる。

3.3. 生成した条件式を満たす入力データの生成

満たすべき条件式を生成した後に、その条件式を満たす整数値を生成する。乱数により整数値を生成し、整数値が条件式を満たせば、入力データとして採用する。

条件式を満たす整数が存在しない場合、入力データの生成は不可能である。そのため、入力データ生成に時間制限を設け、一定時間が経過した場合は入力データ生成処理を終了する。

3.4. 提案手法によるテストケース

図 8 に、2.3 節の図 5 に対して、提案手法を適用することで生成できるテストケースを示す。提案手法により新たに生成できるテストケース中に、期待出力データとして “a mod 4 = 0 and a > 92” を出力しているテストケースを確認できる。

よって、提案手法による処理を今後 BWDM に実装することで、if 式構造の中にある戻り値に対するテストケースを生成できる。

4. 考察

4.1. 提案手法の有用性について

本稿では、VDM++仕様を基にしたテストケース自動生成ツール BWDM において、if 式同士の構造の中にある戻り値に対するテストケース生成を行えない問題を解決するために、if 式の構造認識に基づいたテストケース生成手法を提案した。

提案手法は、まず、VDM++仕様を構文解析した後に、関数定義中に登場する if 式にインデックス付けを行った。この際、ネストの中にある if 式については、その親のインデックスに、更にインデックスを加える形とすることで、ネストした if 式の構造を表現した。そして、付与したインデックスを基に、各出力行へ至るための条件式を生成した。最後に、生成した条件式を満たす入力データを乱数によって生成した。

今回提案した手法により、if 式同士の構造の中にある戻り値に対するテストケースを生成できることを確認できた。よって、既存の BWDM のテストケース生成処理に、今回提案したテストケース生成処理を追加で実装することで、if 式同士の構造の中にある戻り値に対するテストケース生成を行えない問題を解決できる。

4.2. 関連研究

記号実行 (Symbolic Execution)[10] を用いることで、今回の提案手法同様に、if 式が入れ子になった場合も全ての戻り値に対してテストケース生成を行った事例として、宮本らの研究 [11] と、高松らの研究 [12] がある。記号実行は、各戻り値に対する制約条件 (条件式) を導出する。

ソースコード中の表明 (Assersion) の動的な自動生成手法は広く研究されているが、その問題点として、最終的に生成する表明は、手法中で対象プログラムの実行データを取得する際に用いられるテストケースに依存するということが知られている。宮本らはこのテストケー

ス依存問題を改善するために、記号実行などの手法を用いたテストケース生成手法を提案した。記号実行を用いて、対象ソースコード内の全ての戻り値に対して、その制約条件 (条件式) を導出し、その条件を満たす引数を生成している。

また、オブジェクト指向プログラミングにおけるソフトウェアテストを行うには、テスト対象のインスタンス生成・状態の変更などの前準備のメソッド実行を、テストケース内で行う必要がある。そのようなテストケース自動生成ツールの 1 つに Seeker[13] があり、高松らは Seeker の機能拡張を行った。Seeker では、記号実行と具体的な値を組み合わせた動的記号実行 (Dynamic Symbolic Execution) を用いて、ソースコード中の条件式を基に、全ての戻り値に対する引数の組を生成している。

本研究で提案する、if 式の構造認識に基づいたテストケース生成手法は、インデックスを用いて戻り値から条件式を導出するため、記号実行と比較して、探索空間を削減できる。そのため、計算効率の面で、記号実行に比べて優位性が有ると言える。

また、VDM 仕様を基にしたテストケースの自動生成法に関する研究の 1 つに、馬場らの研究がある [14]。馬場らの研究では、VDM 仕様記述からコーナーケースに対するテストを行うことを目的として、テストドライバの雛形の作成を行うツールを開発した。馬場らのツールは、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。

馬場らのツールは、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、馬場ら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[15] 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. おわりに

本稿では、VDM++を基にしたテストケース自動生成ツール BWDM における、if 式同士の構造の中にある戻り値に対するテストケース生成を行えない問題を解決するために、if 式の構造認識に基づくテストケース生成手法を提案した。

提案手法ではまず、VDM++仕様を構文解析した後に、if 条件式にインデックスを付与し、そのインデックスを基に、各出力行へ至るための条件式を生成し最終的に、それらの条件式を満たす入力データを乱数によって生成した。

提案手法により、既存の BWDM では生成できなかった、if 式による構造の中にある戻り値に対するテストケースの生成が可能なことを確認した。故に、既存の BWDM のテストケース生成処理に、今回提案したテストケース生成処理を追加で実装することで、if 式同士の構造の中にある戻り値に対するテストケース生成を行えない問題を解決できる。

以上のことから、今回の提案手法に基づいて BWDM へ入力データ生成処理を実装することで、既存の BWDM の問題点を解決し、BWDM の更なる利便性の向上に繋がると言える。

以下に、BWDM の今後の課題を示す。

- デッドコードへの対応

  戻り値に対する条件式を満たす整数が存在しない場合、戻り値はデッドコードである。この場合、入力データ生成は行えないため、本研究では入力データ生成処理を一定時間で終了する。充足不能な条件式に対しては、入力データ生成処理を行わず、ユーザーにデッドコードを指摘する機能が必要である。

- 未対応の定義部や構文、演算子への対応

  操作定義、型定義などの定義部へは現在未対応である。また、事前条件や事後条件、let 式など

の VDM++に存在する多くの構文にも、既存の BWDM は未対応である。これらの構文からテストケースを生成する手法を考案し、BWDM に実装することで、BWDM の有用性が更に向上すると考えている。

- 整数型以外への対応

  既存の BWDM は、整数型以外の、実数型や合成型には未対応である。構文解析の際に、整数型以外の型の情報を抽出し、境界値分析及び入力データ生成処理を実装することで、これらのテストケースの生成が可能になると考えている。

- 両辺が変数である if 条件式への対応

  if 条件式中の両辺が変数である場合、既存の BWDM は境界値分析を行うことができないため、入力データ生成を行えない。このような if 条件式から境界値を抽出する処理を BWDM に実装することで、BWDM の適用範囲の更なる拡大を見込める。

参考文献

[1] 失敗事例データベース, 失敗事例 – みずほファイナンシャルグループ大規模システム障害, http://condezine.jp/article/detail/2364, 2017/3/8 アクセス

[2] IPA 独立行政法人 情報処理推進機構, なぜ形式手法か, http://sec.ipa.go.jp/users/seminar/seminar_tokyo_20150912-02.pdf, 2017/3/8 アクセス

[3] 荒木啓二郎, 張漢明, プログラム仕様記述論, オーム社, 2002

[4] 立山博基, 片山徹郎, VDM++仕様を基にした境界値テストケース自動生成ツール BWDM の試作について, 平成 28 年度 電気・情報関係学会 九州支部連合大会, p.90, 2016

[5] 立山博基, 片山徹郎, VDM++仕様に対する境界値分析を用いたテストケース自動生成, JaSST'16 九州, p.32, 2016

[6] Hiroki Tachiyama, Tetsuro Katayama, Yoshihiro Kita, Hisaaki Yamaba, and Naonobu Okazaki, *Prototype of Test Cases Automatic Generation Tool BWDM Based on Boundary Value Analysis with VDM++*, The 2017 International Conference on Artificial Life and Robotics(ICAROB 2017), pp.275-278, 2017

[7] SQuBOK 策定部会, ソフトウェア品質知識体系ガイド 第 2 版, オーム社, 2014

[8] Nick Battle GitHub - nickbattle/vdmj, https://github.com/nickbattle/vdmj, 2017/3/8 アクセス

[9] 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ツールについて, 平成 25 年度 電気関係学会 九州支部連合大会, p.381, 2013

[10] James C. King, *Symbolic Execution and Program Testing*, Communications of The ACM Vol.19, No.7, pp.385-394, DOI: 10.1145/360248.360252 1976

[11] 宮本敬三, 堀直哉, 岡野浩三, 楠本真二, Daikon 生成表明改善のためのテストケース自動生成手法とその評価実験, コンピュータソフトウェア, Vol.28, No.4, pp.306-317, https://www.jstage.jst.go.jp/article/jssst/28/4/28_4_4_306/_article/-char/ja/ 2010

[12] 高松宏樹, 佐藤晴彦, 小山聡, 栗原正仁, 動的記号実行によるメソッドの複雑度を考慮したテストケース自動生成, 情報処理学会研究報告 Vol.2014-SE-185, No.27, NAID: 110009803966 2014

[13] Suresh Thummalapenta, Tao Xie, Nikolai Tillmann, Jonathan de Halleux, Zhendong Su, *Synthesizing Method Sequences for High-Coverage Testing*, SIGPLAN Notices Vol.46, No.10, pp.189-206, DOI: 10.1145/2076021.2048083 2011

[14] 馬場勇輔, 荒木啓二郎, 日下部茂, 大森洋一, 形式仕様記述のプロパティベーステストへの活用, 情報処理学会 火の国シンポジウム予稿集, https://www.ipsj-kyushu.jp/page/ronbun/hinokuni/1005/5C/5C-1.pdf, 2017/3/11 アクセス

[15] Olivier Maury, Yves Ledru, Pierre Bontron and Lydie du Bousquet, *Using TOBIAS*

for the automatic geneeration of VDM test cases, Laboratoire Logiciels, Systemes Reseaux – IMAG, http://vasco.imag.fr/Tobias/Papers/vdm02tobias.pdf, 2017/5/10 アクセス