形式仕様を用いたテスト設計時におけるデシジョンテーブル生成支援手法の提案

西川　拳太[a)]・片山　徹郎[b)]

Proposal of a Supporting Method to Generate a Decision Table in Test Design with the Formal Specification

Kenta NISHIKAWA, Tetsuro KATAYAMA

Abstract

In recent years, the software quality becomes more important because the system becomes large scale and high performance. In general, many defects are embedded in the upstream process of the software development. As one reason of the above, specifications include ambiguous description. As a means for writing specifications strictly, formal methods are proposed. By the way, as one of test design techniques, the decision table is proposed. However, it takes much time and effort to extract test items and understand contents written on specifications in designing manually the decision table. This paper proposes a supporting method to generate a template of a decision table from the formal specification written in VDM(Vienna Development Method)++ in order to improve efficiency of the test design with formal methods. We have implemented a supporting tool which automatically generates a template of a decision table from the formal specification. By using the tool, it is considered that the efficiency of the test design is improved.

Keywords: Formal method, VDM++, Test design, Decision table, Automatic generation

1. はじめに

近年、システムの大規模化、高機能化に伴い、従来の開発手法では、ソフトウェアの品質を維持できなくなってきている。同時に、システムの不具合が経済や生活に与える影響は大きな社会問題となっている[1)]。

このような背景から、ソフトウェアの品質がより重要視されるようになった。システムの信頼性・安全性に対する要求は、ますます高まってきている[2)]。

一般に、ソフトウェア開発の上流工程は多くの不具合を含んでいる。上記の理由の 1 つとして、仕様に曖昧な記述を含んだまま、次の工程へ移っていることが挙げられる[3)]。このことから、曖昧な記述を含んだ仕様を厳密に記述する必要がある。仕様を厳密に記述する手段として、形式手法[4)]がある。形式手法はソフトウェア開発の各工程で厳密な仕様を利用するための手段である。形式手法は数理論理学に基づく仕様記述言語を用いてシステムを表現する。形式手法を使うことにより、仕様の曖昧性または不備を除去できる。形式手法は、ソフトウェア品質向上のための一手段として注目されている。

一方で、ソフトウェアサイクルにおけるテストプロセスでは、テスト設計技法の 1 つに、デシジョンテーブル[5)]がある。デシジョンテーブルとは、仕様内の論理関係を条件と動作の項目に分けて、マトリクスで表現したものである。しかし、デシジョンテーブルを人手で作成するには、仕様記述内容の理解、および、テスト項目の抽出など手間と時間がかかる。これは、形式手法を用いて仕様を厳密に記述したからといって、例外ではない。

そこで本研究では、形式手法を用いたテスト設計時の作業効率化を目的として、形式仕様を用いたデシジョンテーブル生成支援手法を提案する。

具体的には、形式仕様記述言語 VDM(Vienna Development Method)++で記述した仕様から、条件と動作の項目を抽出することにより、デシジョンテーブルの生成を支援する。

我々は、本提案手法を実現するため、デシジョンテーブル生成支援ツールを開発する。デシジョンテーブル生成支援ツールは、VDM++で記述した形式仕様からデシジョンテーブルの雛形を CSV (Comma Separated Values)[6)]形式で自動生成する。ツールを使用することによって、テスト設計時の作業効率が改善すると考えられる。

なお、今回形式仕様記述言語として用いる VDM++は、1960 年代に開発されたライトウェイトな形式手法 VDM の形式仕様記述言語 VDM-SL[7)]を、オブジェクト指向に基づいたモデル化を扱えるように拡張した言語である[8)]。ま

a)情報システム工学専攻大学院生
b)情報システム工学科准教授

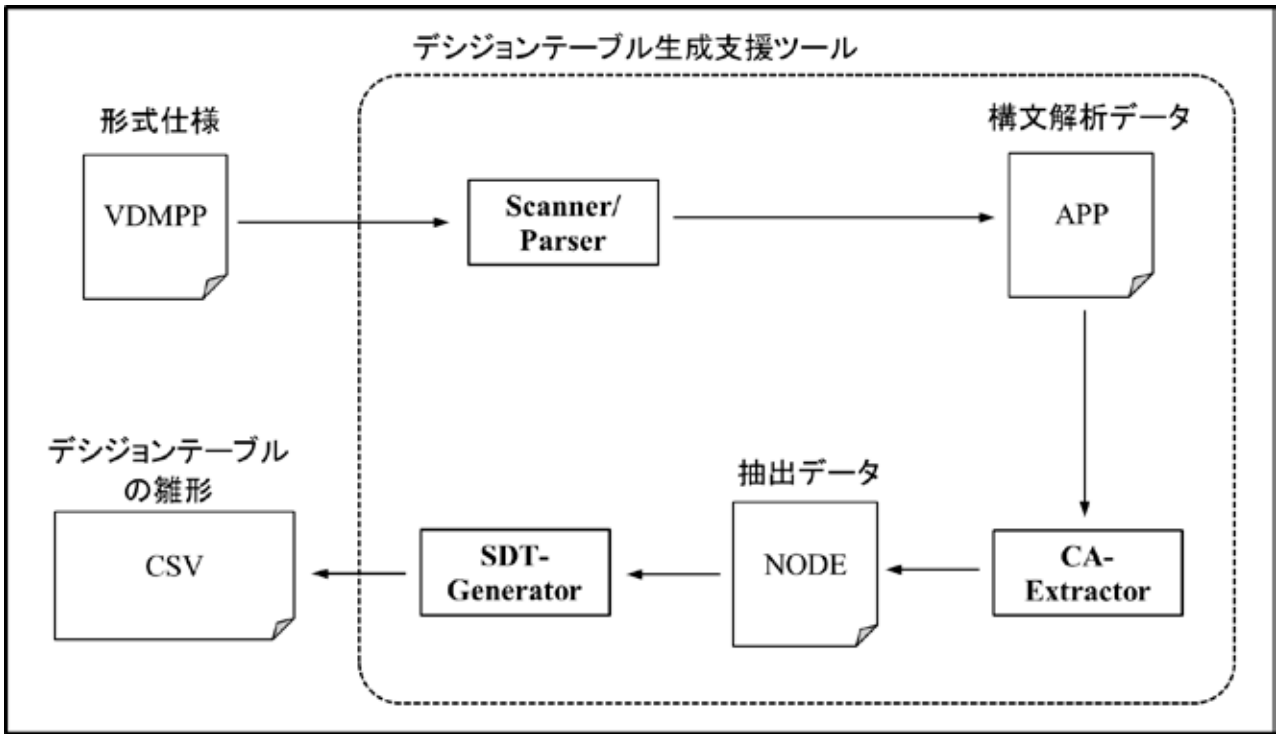

図 1. 形式仕様を用いたデシジョンテーブル生成支援手法の流れ.

```
            new OmlBinaryOperator(23,14,9),
            new OmlSymbolicLiteralExpression(new OmlNumericLiteral(0,14,10),14,10),
            14,
            9
           ),
           nil,
           14,
           9
          ),
          3,
          17
         ),
         3,
         17
        )
       ],2,1)
      ],
      false,
      1,
      1
     )
    ],0,0),
    [
     new OmlLexem(1,1,275,"class",1),
     new OmlLexem(1,7,433,"FizzBuzz",2),
     new OmlLexem(2,1,312,"functions",1),
```

図 2. 構文解析データの出力例.

た、ツールが自動生成するデシジョンテーブルの雛形は、条件と動作の項目、および、条件項目に真理値を生成したものである。したがって、ユーザはデシジョンテーブルを完成させるために、動作項目の真理値を、デシジョンテーブルの雛形に記述する必要がある。

2. 形式手法を用いたデシジョンテーブル生成支援手法

本提案手法の流れを、図 1 に示す。提案手法は、以下の 3 つの部で構成する。

i. Scanner/Parser
ii. CA-Extractor
iii. SDT-Generator

以下、提案手法の流れを説明する。

1. ユーザは、事前に構文および型チェックを済ませた形式仕様(VDM++ファイル)を用意する。
2. ユーザは、形式仕様を指定して、デシジョンテーブル生成支援ツールを実行する。
3. デシジョンテーブル生成支援ツールは、ユーザが指定した形式仕様を、Scanner/Parser の入力として実行する。
4. Scanner/Parser は、形式仕様を解析し、形式仕様の解析データを生成する。
5. CA-Extractor は、Scanner/Parser が生成した形式仕様の解析データから、形式仕様の条件と動作を抽出した抽出データを生成する。
6. SDT-Generator は、 CA-Extractor が生成した抽出データから、デシジョンテーブルの雛形を生成し、.csv ファイルとして出力する。

以下では、それぞれの部について説明する。

2.1 Scanner/Parser

Scanner/Parser は、Marcel Verhoef が開発した Overture のための VDM 構文解析器 [9]である。Scanner/Parser は、ユーザが指定した形式仕様を読み込む。Scanner/Parser は、読み込んだ形式仕様に対し構文解析を行い、その解析結果を構文解析データ(.app)として出力する。

構文解析データの出力例を、図 2 に示す。構文解析データには、以下の解析情報を保持している。

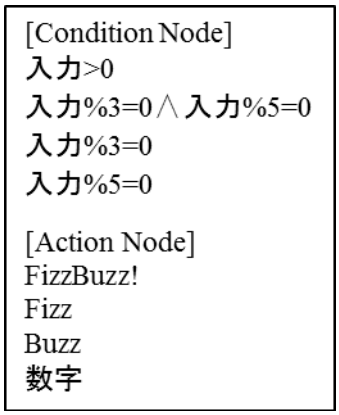

図 3. 抽出データの出力例.

- 抽象構文木
- 行番号
- インデックス番号
- 識別子
- トークン
- 判別子

抽象構文木は、仕様記述内の構文要素を、構文規則に従って木構造で表現したものである。行番号は、トークンが出現する行を示す。インデックス番号は、仕様記述内におけるトークンの開始位置を示す。識別子は、トークごとに割り当てられた固有の ID である。トークンは、仕様に記述された構文の最小単位である。判別子は、予約語やコメントなど、各トークンがどのドメインに属するかを示す。

これらの解析データは、CA-Extractor で仕様記述内の条件と動作を抽出する際に必要となる。

2.2 CA-Extractor(Condition Action-Extractor)

CA-Extractor は、Scanner/Parser が生成した構文解析データから条件と動作を抽出し、その条件と動作を抽出データ(.node)として出力する。

抽出データの出力例を、図 3 に示す。

これらの条件と動作の抽出データは、SDT-Generator でデシジョンテーブルの雛形を生成する際に必要となる。

CA-Extractor は条件と動作を、if-then-else 式、および、事前条件式と事後条件式から抽出する。

条件と動作の抽出処理を、表 1 に示す。

2.3 SDT-Generator(Skeleton Decision Table-Generator)

SDT-Generator は、CA-Extractor が出力した抽出データから、デシジョンテーブルの雛形を自動生成する。デシジョンテーブルの雛形の出力形式は、汎用性の高い CSV 形式を採用している。

デシジョンテーブルの雛形の出力例を、表 2 に示す。デシジョンテーブルの雛型には、条件と動作の項目を 2 列目に記述する。3 列目以降は、条件項目の真理値を記述する。

表 1. 条件と動作の抽出処理.

■ 条件の抽出
● if-then-else 式および事前条件式と事後条件式が対象である。 ● 条件開始トークン(if, elseif, pre, post)から、条件終了トークン(then, ;)までの間を、条件として抽出する。 e.g., if(elseif) 条件 A then → 条件 A pre 条件 B; → 条件 B
■ 動作の抽出
● if-then-else 式が対象である。 ● then または else のトークン開始直後を、動作として抽出する。 e.g., else 動作 A → 動作 A

表 2. デシジョンテーブルの雛形の出力例.

	ノード	#1	#2	…	#99	#100
条件	書籍	T	T	…	F	F
条件	セール	T	T	…	F	F
⋮	⋮	⋮	⋮	⋮	⋮	⋮
動作	割引	null	null	null	null	null
⋮	⋮	⋮	⋮	⋮	⋮	⋮

動作項目の真理値は生成不可能なため、デシジョンテーブルの雛形の動作項目の真理値は空である。

3. 適用例

我々は、本提案手法を実現するデシジョンテーブル生成支援ツールを開発した。デシジョンテーブル生成支援ツールは、テスト設計を支援する。ツールは、形式仕様を入力として、デシジョンテーブルの雛形を出力する。

開発したツールが正しく動作することを、適用例を用いて確認する。具体的には、以下の 2 つの項目を確認する。

i. 形式仕様から条件と動作を抽出していること
ii. 条件項目の真理値を生成していること

適用例に用いる形式仕様を、図 4 に示す。これは、VDM++を用いて記述した FizzBuzz の仕様である。FizzBuzz とは、英語圏で行われる言葉遊びである。入力した数値が 3 で割り切れるときは"Fizz"、5 で割り切れるときは"Buzz"、両方で割り切れる場合は"FizzBuzz!"と表示する。適用例に用いる形式仕様には、入力が 0 以上であることを意味する事前条件式(pre 入力> 0)を記述している。

開発したツールに形式仕様を適用した結果、ツールが生成したデシジョンテーブルの雛形の一部を、図 5 に示す。

デシジョンテーブルの雛形から「i. 形式仕様から条件と動作を抽出していること」と「ii. 条件項目の真理値を生成していること」を確認した。すなわち、開発したツール

```
class FizzBuzz
functions
  public static FizzBuzz関数 : nat->seq of char
  FizzBuzz関数(入力) ==
    if 入力 mod 3 = 0 and 入力 mod 5 = 0 then
      "FizzBuzz!"
    elseif 入力 mod 3 = 0 then
      "Fizz"
    elseif 入力 mod 5 = 0 then
      "Buzz"
    else
      "数字"
  pre
    入力 > 0 ;
end FizzBuzz
```

図 4. FizzBuzz の形式仕様.

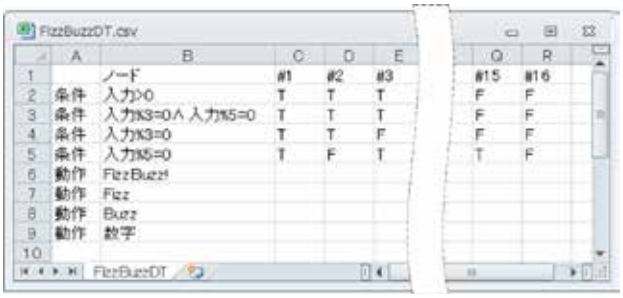

図 5. デシジョンテーブルの雛形の一部.

が正しく動作することを確認した。

4. 考察

本研究では、形式手法を用いたテスト設計時の作業効率化を目的として、形式仕様を用いたデシジョンテーブル生成支援手法を提案した。

我々は、本提案手法を実現するデシジョンテーブル生成支援ツールを開発した。開発したツールは、VDM++で記述した形式仕様から、デシジョンテーブルの雛形を自動生成する。ツールを使用することによって、テスト設計時の作業効率が改善すると考えられる。

本章では、提案手法について考察する。

4.1 提案手法に関する考察

本提案手法の有用性を、適用例でデシジョンテーブル生成支援ツールが生成したデシジョンテーブルの雛形を用いて確認する。

まず、デシジョンテーブルの雛形の動作項目に真理値を手動で記述し、デシジョンテーブルを完成させた。完成させたデシジョンテーブルの一部を、図 6 に示す。

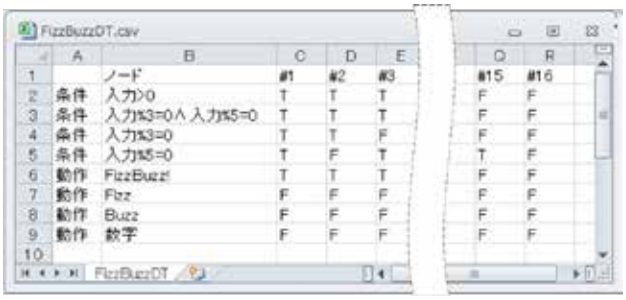

図 6. デシジョンテーブルの一部.

```
public class FizzBuzz {
  public static void main(String[] args) {
    byte b[] = new byte[100];
    try {
      System.in.read(b);
      int number = Integer.parseInt((new String(b)).trim());
      if(number<=0)throw new Exception("input is more than zero");
      if (number % 3 == 0 && number % 5 == 0) {
        System.out.println("FizzBuzz!");
      } else if (number % 3 == 0) {
        System.out.println("Fizz");
      } else if (number % 5 == 0) {
        System.out.println("Buzz");
      } else {
        System.out.println(String.valueOf(number));
      }
    } catch (Exception e) {
      System.out.println("Err: " + e);
    }
  }
}C
```

図 7. FizzBuzz のプログラム.

次に、適用例に用いた形式仕様から、FizzBuzz の仕様を、Java 言語を用いて実装した。Java で実装した FizzBuzz のプログラムを、図 7 に示す。

完成したデシジョンテーブルを用いて FizzBuzz プログラムをテストした。その結果、FizzBuzz 仕様内の条件と動作のすべての組合せをテストできた。すなわち、本提案手法の有用性を、確認できた。

4.2 関連研究

デシジョンテーブル生成支援ツールは、形式仕様から条件と動作を抽出し、デシジョンテーブルの雛形を CSV 形式で自動生成する。これによって、テスト担当者が手動でテスト設計を行う際の時間と手間を削減できる。

形式仕様からテスト設計を行った事例 [10]は未だ報告が少なく、その手法は十分に確立していない。

デシジョンテーブルの生成を支援するツールとして、CEGTest[11]がある。CEGTest は、ユーザが作成した原因結果グラフから、デシジョンテーブルを自動生成する。CEGTest は、デシジョンテーブルを自動で圧縮する機能を持つ。

CEGTest と本研究において開発したデシジョンテーブル生成支援ツールを比較した結果、明らかになった問題点を以下に示す。

- 動作項目の真理値が未記述である
- 集約可能な条件項目に対し、自動で圧縮する機能を持たない

しかし、CEGTest の入力となる原因結果グラフは、ユーザが仕様書から直接人手で作成しなければならない。そのため、デシジョンテーブルを作成する際に、形式仕様記述内容の理解、および、テスト項目の抽出など手間と時間がかかる、といった問題が発生する。これに対し、デシジョンテーブル生成支援ツールは、形式仕様の記述内容を理解する必要なく、VDM++で記述した仕様をツールの入力として指定することにより、デシジョンテーブルの雛形を得ることができる。

また、日本語で記述された仕様を基に、形式仕様を記述する場合、必ずしも動作が明確に定義されているとは限らない。そのため、デシジョンテーブルにおける動作項目の真理値が未記述であったとしても、本提案手法においては、デシジョンテーブルの雛形を自動生成することにより、ユーザは条件と動作の論理関係を意識しながら、仕様内の動作を記述していくことが可能となる。さらに、動作項目の真理値を記述する際に、テストすべき点を考慮しながら、ユーザによる任意の表の圧縮が可能である。

これらのことから、デシジョンテーブル生成支援ツールは、形式仕様を用いたテスト設計を行う際に、作業の効率化が期待できると考えられる。

5. おわりに

本研究では、形式手法を用いたテスト設計時の作業効率化を目的として、形式仕様を用いたデシジョンテーブル生成支援手法を提案した。

本提案手法を実現するため、我々はデシジョンテーブル生成支援ツールを開発した。開発したツールは、形式仕様から条件と動作を抽出することにより、デシジョンテーブルの雛形を CSV 形式で自動生成する。我々は、形式仕様から条件と動作を抽出することを確認した。また、デシジョンテーブルの雛形において、条件項目の真理値を生成していることを確認した。ツールを使用することによって、テスト設計時の作業効率が改善すると考えられる。

以下に、今後の課題を挙げる。

- 他のテスト設計技法への対応

本提案手法が支援するテスト設計技法は、デシジョンテーブルのみである。しかし、デシジョンテーブルのみを用いて、厳密な形式仕様からテスト設計を行うには限界がある。より効果的なテスト設計を行うためには、他のテスト設計技法への対応が必要となる。形式手法は、対象とするモデルの仕様を記述する際、集合の定義が可能であることから、今後は、モデルのドメインに着目したテスト設計技法である同値分割や境界値分析への支援手法を考えている。

- 動作項目に対する真理値の生成支援

開発したツールは、動作項目の真理値を自動生成できない。したがって、ユーザはデシジョンテーブルを完成させるため、動作項目の真理値を、デシジョンテーブルの雛形に直接、人手で記述しなければならない。条件項目の数が増える場合、動作項目の真理値を人手で記述するには、手間を要する。これは、構文解析データから条件と動作の項目を抽出する際、条件と動作の論理関係を、データとしてファイルに保持するなど、動作項目の真理値を自動生成する仕組みを考える必要がある。

- デシジョンテーブル生成支援ツールの効率性向上

本研究において開発したツールは、CUI(Character User Interface)で動作する。ツールが自動生成したデシジョンテーブルの雛形は、ファイルを開くことにより、その結果を確認できる。これは、ユーザが早急に結果を確認したい際には、効率性が低いと言える。今後は、ファイルの出力とは別に、コマンドライン上にツールの出力結果を表示する、あるいは、デシジョンテーブルを GUI(Graphical User Interface)で表現する、などの対応に取り組む。

- デシジョンテーブル生成支援ツールの有効性評価

本研究において開発したツールが、どの程度の割合で作業を効率化できるかについて、定量的な評価を行う必要がある。具体的には、開発したツールを用いて作成したデシジョンテーブルと、人手により作成したデシジョンテーブル、それぞれの完成までに要した時間を計測し、比較する実験を考えている。

- cases 式や for ループ文、while ループ文など、各構文への対応

デシジョンテーブル生成支援ツールに適用可能な形式仕様には、制限がある。具体的には、デシジョンテーブル生成支援ツールは、すべての VDM++構文に対応していない。そのため、形式仕様を用いた条件および動作の抽出は、if-then-else 式、および、事前条件式と事後条件式のみを対象としている。適用可能な形式仕様の範囲を広げるため、ツールの機能拡張を考える。

参考文献

1) John Fitzgerald, Peter Gorm Larsen, Paul Mukheerjee, Nico Plat, Marcel Verhoef: Validated Designs for Object-Oriented Systems, Springer, 2005.
2) 中島震: 形式手法入門 ロジックによるソフトウェア設計, オーム社, 2012.
3) 佐原伸: ～ソフトウェアトラブルを予防する～形式手法の技術講座, ソフト・リサーチ・センター, 2008.
4) 中島震: ソフトウェア工学の道具としての形式手法, NII-2007-007J, 2007.
5) ISO 5806: Specification of single-hit decision tables.
6) RFC4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files.
7) 荒木啓二郎, 張漢明: プログラム仕様記述論, オーム

社, 2002.
8) 石川冬樹: VDM++による形式仕様記述, 近代科学社, 2011.
9) A Scanner/Parser for the Overture Toolset: http://overturetool.hosting.west.nl/twiki/bin/view/Main/OvertureParser/ (2014/2/27 アクセス).
10) IPA/SEC: 厳密な仕様記述における形式手法成功事例調査報告書, http://www.ipa.go.jp/sec/reports/20130125.html (2014/2/27 アクセス).
11) CEGTest: http://softest.jp/tools/CEGTest/ (2014/2/27 アクセス).