形式仕様を用いたデシジョンテーブル生成手法の提案

西川 拳太[1]　片山 徹郎[1]　喜多 義弘[2]　山場 久昭[1]　岡崎 直宣[1]

概要：近年，ソフトウェアの品質がより重要視されるようになった．一般に，ソフトウェア開発の上流工程の成果物は多くの不具合を含んでいる．その理由の 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．仕様を厳密に記述する手段として，形式手法がある．一方，テスト設計技法の 1 つに，デシジョンテーブルがある．デシジョンテーブルの設計には，仕様記述内容の理解と論理関係の抽出に，手間と時間がかかるという問題がある．これは，形式手法を用いて仕様を厳密に記述した場合も，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手法を提案する．我々は，本提案手法を実現するデシジョンテーブル自動生成ツールを実装した．本ツールを使用することにより，条件の真理値の組み合わせが 2^{13}(8129) 通りのデシジョンテーブルに対し，0.7 秒程度の時間で自動生成できた．

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．

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 [1].

一般に，ソフトウェア開発の上流工程の成果物は，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．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 [2] 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．形式手法を用いることにより，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 1 手段として注目されている．

一方で，ソフトウェアライフサイクルにおけるテストフェーズでは，テスト設計技法の 1 つに，デシジョンテーブル [3] 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，条件と動作の抽出に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を生成する．

我々は，本提案手法を実現するため，デシジョンテーブル自動生成ツールを実装する．デシジョンテーブル自動生成ツールは，VDM++で記述した形式仕様を入力とし，生成したデシジョンテーブルを GUI(Graphical User Interface) 上に提示する．

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 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 [4].

2. デシジョンテーブル生成手法

デシジョンテーブル生成手法の流れを，**図 1** に示す．本提案手法は、Parser，D-Extractor，CAP-Extractor，DT-Generator の 4 つの部分から成る．以下，提案手法の処理手順を説明する．

(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

[1] 宮崎大学
University of Miyazaki
[2] 神奈川工科大学
Kanagawa Institute of Technology

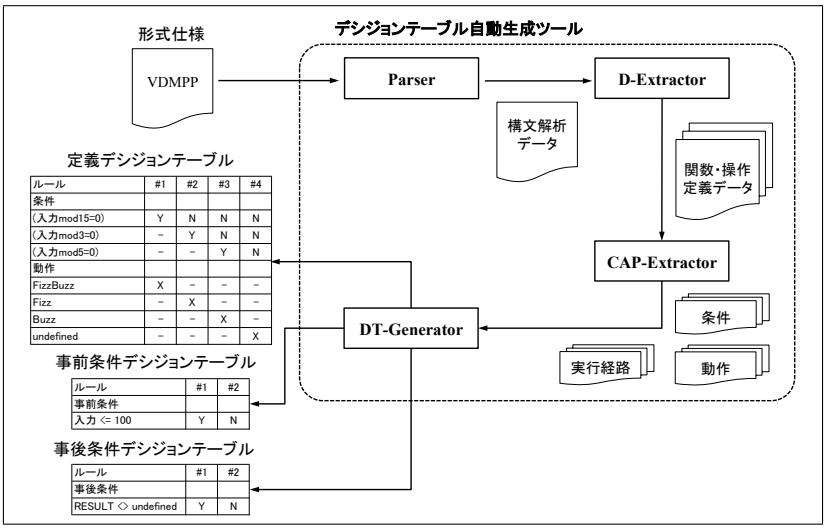

図 1　デシジョンテーブル生成手法の流れ

Fig. 1　A process flow of a method to generate a decision table.

自動生成ツールを実行する．

(3) デシジョンテーブル自動生成ツールは，ユーザが指定した形式仕様を，Parser の入力として実行する．

(4) Parser は，Overture Toolset の構文解析器 [5] を利用して形式仕様を構文解析し，構文解析データを出力する．

(5) D-Extractor は，Parser が出力した形式仕様の構文解析データから，形式仕様上に記述された関数と操作の定義を，それぞれ関数定義データおよび操作定義データとして抽出する．

(6) CAP-Extractor は，D-Extractor が抽出した関数定義データと操作定義データから，形式仕様の条件と動作，および，その実行経路を抽出データとして抽出する．

(7) DT-Generator は， CAP-Extractor が抽出した抽出データから，事前条件と事後条件および定義の 3 つのデシジョンテーブルを生成し，GUI 上に掲示する．

以下では，提案手法を構成している各部について，それぞれ述べる．なお，本提案手法における条件および動作の抽出は，if-then-else 構文と cases 構文，および，事前条件式と事後条件式のみを対象とする．今回，for や while，let be などの各構文および再帰呼出しは，適用可能な仕様記述の範囲外とする。

2.1　D-Extractor

D-Extractor は，Definition-Extractor の略であり，構文解析データから形式仕様記述内の関数と操作の定義を，それぞれ関数定義データおよび操作定義データとして抽出する．各定義ごとに抽出することにより，CAP-Extractor で条件と動作および実行経路のデータを抽出しやすくし，DT-Generator で各定義ごとにデシジョンテーブルを表示することができる．

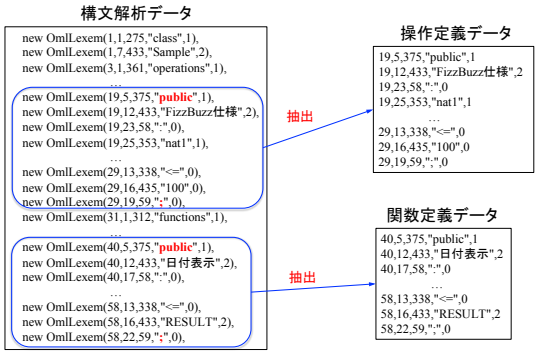

図 2　D-Extractor の流れ

Fig. 2　A process flow of the D-Extractor.

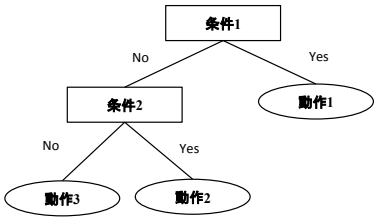

図 3　実行経路データの例

Fig. 3　An example of a execution path data.

D-Extractor の流れを，図 2 に示す．D-Extractor は，アクセス修飾子 (public, private, protected のいずれか) を開始トークンとして，各定義の終わりを示す “;” までを，それぞれ関数定義データまたは操作定義データとして抽出する．関数定義データおよび操作定義データは，CAP-Extractor で条件と動作および実行経路を抽出する際に必要となる．

2.2　CAP-Extractor

CAP-Extractor は，Condition Action Path-Extractor の略であり，D-Extractor が抽出した関数定義データと操作定義データから，条件と動作，および，その実行経路を，

表 1　条件と動作の抽出処理
Table 1　The extraction process of conditions and actions.

処理パターン	条件抽出規則	動作抽出規則
(if または elseif) 条件 then 動作 (elseif または else) else 動作 (if else ; pre post)	(if または elseif) から then まで –	then から (elseif または else) まで else から (if else ; pre post) のいずれかまで
(:または,) 条件 -> 動作 (, または end)	(:または,) から-> まで	-> から (, または end) まで
pre 条件 (; または post)	pre から (; または post) まで	–
post 条件 ;	post から; まで	–

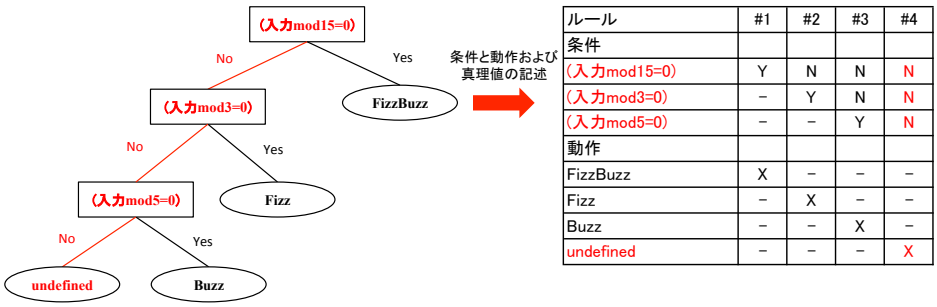

図 4　DT-Generator の流れ
Fig. 4　A process flow of the DT-Generator.

それぞれ抽出データとして抽出する．CAP-Extractor は，if-then-else 構文と cases 構文，および，事前条件式と事後条件式を抽出の対象とする．

実行経路データの例を，**図 3** に示す．実行経路データとは，各定義に記述した条件と動作の関係を木構造で表現したものであり，各ノードは条件名または動作名を持つ．条件名を持つノードは，“Yes” と “No” の 2 つの子ノードを持つ．

CAP-Extractor の抽出処理は，**表 1** に示した抽出規則に従い，該当したトークンを条件または動作と実行経路として抽出する．これらの条件と動作および実行経路の抽出データは，DT-Generator でデシジョンテーブルを生成する際に必要となる．

2.3　DT-Generator

DT-Generator は，Decision Table-Generator の略であり，CAP-Extractor が抽出した条件と動作および実行経路の抽出データから，事前条件と事後条件および定義の 3 つのデシジョンテーブルを自動生成する．定義デシジョンテーブルとは，関数または操作の定義から，事前条件と事後条件を除いた，条件と動作の論理関係の組み合わせをマトリクスで表現したものである．事前条件と事後条件および定義のデシジョンテーブルをそれぞれ分けて掲示することにより，複雑な仕様から生成される大規模なデシジョンテーブルの煩雑さを解消する．デシジョンテーブルは，各定義ごとに生成する．

DT-Generator の流れを，**図 4** に示す．デシジョンテーブルには，CAP-Extractor で抽出した条件と動作を 1 列目に記述する．2 列目以降は，実行経路のデータを元に各項目の真理値を記述する．

3.　デシジョンテーブル自動生成ツールの実装

2 章で提案したデシジョンテーブル生成手法に基づき，デシジョンテーブル自動生成ツールを実装する．デシジョンテーブル自動生成ツールは VDM++で記述した形式仕様を入力とし，出力としてデシジョンテーブルを生成する．

今回実装したデシジョンテーブル自動生成ツールの外観を，**図 5** に示す．以下，各パーツについて説明する．

(1) 定義選択ツリー
クラス名をルートノードに，各関数と操作の定義名を子ノードとして表示する．各子ノードの定義名を選択した際は，「定義デシジョンテーブルエリア」および「事前条件・事後条件デシジョンテーブルエリア」に，選択した定義のデシジョンテーブルを表示する．

(2) 形式仕様エリア
読み込んだ形式仕様を行番号付きで表示する．

(3) 定義デシジョンテーブルエリア
定義選択ツリーによって選択した定義名の定義デシジョンテーブルを表示する．

(4) 事前条件・事後条件デシジョンテーブルエリア
定義選択ツリーによって選択した定義名の事前条件デシジョンテーブルと事後条件デシジョンテーブルをタブ形式で表示する．

4.　適用例

今回実装したデシジョンテーブル自動生成ツールが正し

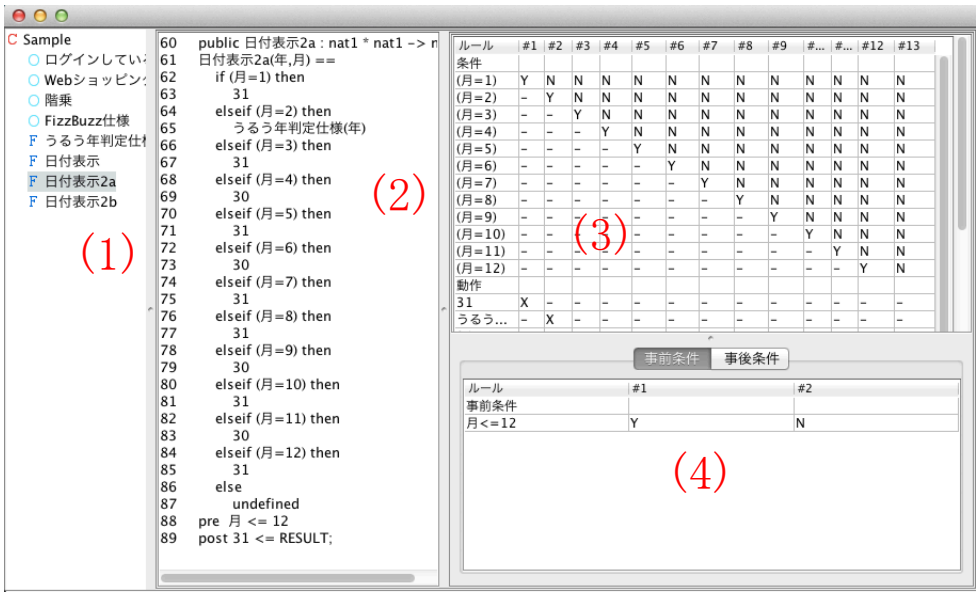

図 5　ツールの外観

Fig. 5　An overview of our tool.

```
class FizzBuzz
functions
    public FizzBuzz 仕様 : nat1 -> seq of char
    FizzBuzz 仕様 (入力) ==
        if (入力 mod 15=0) then
            ”FizzBuzz”
        elseif (入力 mod 3=0) then
            ”Fizz”
        elseif (入力 mod 5=0) then
            ”Buzz”
        else
            undefined
    pre 入力 <= 100;
end FizzBuzz
```

図 6　FizzBuzz の形式仕様

Fig. 6　A formal specification of the FizzBuzz.

ルール	#1	#2	#3	#4
条件				
(入力mod15=0)	Y	N	N	N
(入力mod3=0)	–	Y	N	N
(入力mod5=0)	–	–	Y	N
動作				
FizzBuzz	X	–	–	–
Fizz	–	X	–	–
Buzz	–	–	X	–
undefined	–	–	–	X

ルール	#1	#2
事前条件		
入力<=100	Y	N

図 7　ツールの出力結果

Fig. 7　An output result of our tool.

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 3 つの項目を確認する．

(1) 形式仕様から条件と動作を抽出していること
(2) 条件と動作の真理値を生成していること
(3) 生成したデシジョンテーブルから，仕様内の条件と動作のすべての組み合わせをテストできること

適用例に用いる形式仕様を，**図 6** 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 で割り切れる場合は “Fizz”，5 で割り切れる場合は “Buzz”，両方で割り切れる場合は “FizzBuzz” と表示し，それ以外は入力した数値を表示する．適用例に用いる形式仕様には，数値の表示の場合，未定義式 (undefined) として定義する．未定義式は，式の結果が定義されないことを明白に述べるために用いる．事前条件式には，入力が 100 以下であることを意味する式 (pre 入力 <= 100) を記述している．

実装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を，**図 7** に示す．図 7 のデシジョンテーブルから「(1) 形式仕様から条件と動作を抽出していること」と「(2) 条件と動作の真理値を生成していること」を確認した．

次に，FizzBuzz の形式仕様から，Java 言語を用いてプログラムを実装した．なお，実装の際に，形式仕様に記述した未定義式は，入力した数値を println メソッドを用いて表示する実装に，事前条件式は，入力が 100 以上である場合に “入力は 100 以下” のメッセージと共に，例外処理として実装した。

Java で実装した FizzBuzz プログラムを，**図 8** に示す．図 7 の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結果，「(3) 生成したデシジョンテーブルか

```
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>100)throw new Exception("入力は 100 以下");
            if (number % 15 == 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);
        }
    }
}
```

図 8　FizzBuzz プログラム
Fig. 8　A source code of the FizzBuzz program.

ら，仕様内の条件と動作のすべての組み合わせをテストできること」を確認した．

これらのことから，実装したツールが正しく動作することを確認した．

5. 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自動生成ツールを実装した．実装したツールは，VDM++で記述した形式仕様から，デシジョンテーブルを自動生成する．

本章では，提案手法について考察する．

5.1 デシジョンテーブル自動生成ツールの有用性

デシジョンテーブル自動生成ツールの有用性を，条件の真理値の組み合わせ数が異なる，以下の 3 つの形式仕様を用いて確認する．

(1) うるう年判定 (条件の真理値の組み合わせ：2^3 通り)
(2) FizzBuzz(条件の真理値の組み合わせ：2^4 通り)
(3) 干支表示 (条件の真理値の組み合わせ：2^{13} 通り)

具体的には，各形式仕様に対し，デシジョンテーブル自動生成ツールが生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，比較する実験を行う．

なお，FizzBuzz は 4 章で適用例に用いた形式仕様を，うるう年判定と干支表示は，それぞれ**図 9** と**図 10** に示す形式仕様を用いる．

実験の結果を，**表 2** に示す．表 2 に示すように，デシジョンテーブル自動生成ツールを用いた結果，条件の真理値の組み合わせが $2^3(8)$ から $2^4(16)$ 通りの仕様で約 0.6 秒程度，$2^{13}(8129)$ 通りの仕様でも，0.7 秒程度の時間でデシジョンテーブルを生成できたことがわかる．すなわち，デシジョンテーブル自動生成ツールの有用性を確認できた．

```
class LeapYear
operations
    public うるう年判定仕様 : nat1 ==> nat1
    うるう年判定仕様 (年) ==
        if (年 mod 4 = 0) then
            if (年 mod 100 = 0) then
                if (年 mod 400 = 0) then
                    29
                else
                    28
            else
                29
        else
            28;
end LeapYear
```

図 9　うるう年判定の形式仕様
Fig. 9　A formal specification of determining a leap year.

```
class Zodiac
functions
    public 干支表示 : nat1 -> seq of char
    干支表示 (入力) ==
        cases 入力 :
        1 -> "子",
        2 -> "丑",
        3 -> "寅",
        4 -> "卯",
        5 -> "辰",
        6 -> "巳",
        7 -> "午",
        8 -> "未",
        9 -> "申",
        10 -> "酉",
        11 -> "戌",
        12 -> "亥",
        others -> undefined
        end;
end Zodiac
```

図 10　干支表示の形式仕様
Fig. 10　A formal specification of displaying zodiac.

5.2 関連研究

デシジョンテーブル自動生成ツールは，VDM++で記述した形式仕様を入力とし，生成したデシジョンテーブルを GUI 上に提示する．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 [6], [7] は未だ報告が少なく，その手法は十分に確立していない．

また，デシジョンテーブルの生成を支援するツールとして，CEGTest[8] がある．CEGTest は，ユーザが作成した

表 2　計測時間 (sec)
Table 2　Measurement time(sec).

形式仕様 (条件の真理値の組み合わせの数)	人手によるデシジョンテーブルの作成	デシジョンテーブル自動生成ツール
うるう年判定 (2^3)	164	0.558
FizzBuzz(2^4)	137	0.563
干支表示 (2^{13})	329	0.713

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．

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，条件と動作の抽出に手間と時間がかかる，といった問題が発生する．さらに，形式仕様を入力としたテストツール [9], [10] はいくつか存在するものの，これらのツールは，どれもデシジョンテーブルの生成には対応していない．これに対し，デシジョンテーブル自動生成ツールは，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を自動で得ることができる．

6. 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自動生成ツールを実装した．実装したツールは，VDM++で記述した形式仕様を入力とし，生成したデシジョンテーブルを GUI 上に提示する．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において，条件と動作の真理値を生成していることを確認した．そして，生成したデシジョンテーブルから，仕様内の条件と動作のすべての組み合わせをテストすることができた．本ツールを使用することにより，条件の真理値の組み合わせが 2^{13}(8129) 通りのデシジョンテーブルに対し，0.7 秒程度の時間で自動生成できた．

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
  本提案手法が対応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を支援する手法を考えている．
- デシジョンテーブル自動生成ツールの実用性向上
  現状，デシジョンテーブル自動生成ツールは，for や while，let be などの各構文および再帰呼出しに対応しておらず，適用可能な形式仕様の範囲に制限がある．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 構文と cases 構文，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，実装したツールの機能拡張を行う必要がある．
- 大規模なシステムを対象とした形式仕様への適用
  今回，デシジョンテーブル自動生成ツールに適用した仕様は，条件の真理値の組み合わせが最大で 2^{13}(8192) 通りである．より大規模で複雑な仕様に対し，本ツールがどの程度の時間でデシジョンテーブルを生成できるのか，時間を計測し，評価する必要がある．
- テストデータの自動生成
  形式仕様を用いたテスト設計時の作業効率をより向上するため，ツールが自動生成したデシジョンテーブルと，形式仕様の記述を元にテストデータを作成し，テスト実施の効率化を図る．

参考文献

[1] John Fitzgerald, Peter Gorm Larsen, Paul Mukheerjee, Nico Plat and Marcel Verhoef: *Validated Designs for Object-Oriented Systems*, Springer (2005).
[2] 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).
[3] ISO 5806:1984, Specification of single-hit decision tables.
[4] 石川冬樹：VDM++による形式仕様記述，近代科学社 (2011).
[5] A Scanner/Parser for the Overture Toolset: 入手先 ⟨http://overturetool.hosting.west.nl/twiki/bin/view/Main/OvertureParser/⟩ (参照 2014-05-16).
[6] 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，入手先 ⟨http://www.ipa.go.jp/sec/reports/20130125.html⟩ (参照 2014-05-16).
[7] 羽田裕，和田圭司：VDM 仕様とテスト設計による仕様の問題発見に関する評価，入手先 ⟨http://www.juse-sqip.jp/workshop/seika/2011/attachs/SQiPII-1.pdf⟩ (参照 2014-05-16).
[8] CEGTest: 入手先 ⟨http://softest.jp/tools/CEGTest/⟩ (参照 2014-05-16).
[9] Y. Ledru, L. du Bousquet, O. Maury, and P. Bontron: *Filtering TOBIAS combinatorial test suites*, Fundamental Approaches to Software Engineering, pp.281-294 (2004).
[10] Adriana Sucena Santos: *Combinatorial Test Automation Support for VDM++*, VDM/Overture Workshop, pp.45-53 (2008).