

## Proposal of a Supporting Method to Generate a Decision Table from the Formal Specification

Tetsuro Katayama<sup>\*,†</sup>, Kenta Nishikawa<sup>‡</sup>, Yoshihiro Kita<sup>‡</sup>, Hisaaki Yamaba<sup>‡</sup> and Naonobu Okazaki<sup>‡</sup>

<sup>†</sup>*Faculty of Engineering, University of Miyazaki, Miyazaki 889-2192, Japan*

<sup>‡</sup>*Security Center, Kanagawa Institute of Technology, Kanagawa 243-0292, Japan*

<sup>\*</sup>Corresponding author, E-mail: kat@cs.miyazaki-u.ac.jp

Tel: +81-985-58-7586, Fax: +81-985-58-7586

### 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 decision table from the formal specification in order to improve efficiency of the test design with formal methods. We have implemented a supporting tool to generate a decision table. It automatically generates a skeleton 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. Introduction

In recent years, the software quality cannot be maintained with the conventional software development methods because the system becomes large scale and high performance. At the same time, effect of defects in the system becomes one of the major social problems with the economy and life [1].

Hence, the software quality becomes more important. A demand for reliability and safety of the system is becoming increasingly.

In general, many defects are embedded in the upstream process of the software development. As one reason of the above, each step in the software development process moves to the next step with specifications included ambiguous description. Therefore, specifications should be written strictly. As a means for writing specifications strictly, formal

methods [2] are proposed. The formal methods are a means for using strict specifications in each step in the software development process. They express the system with a specification description language based on mathematical logic. Using the formal methods can remove defects or ambiguity of the specifications. They attract attention as a means to improve software quality.

By the way, as one of the test design techniques, the decision table [3] is proposed in the testing process of the software cycle. The decision table uses a matrix divided the logical relationships in specifications into items of conditions and actions. However, it takes much time and effort to extract test items and understand contents written on specifications in designing manually the decision table. It is no exception even if you write strict specifications with formal methods.

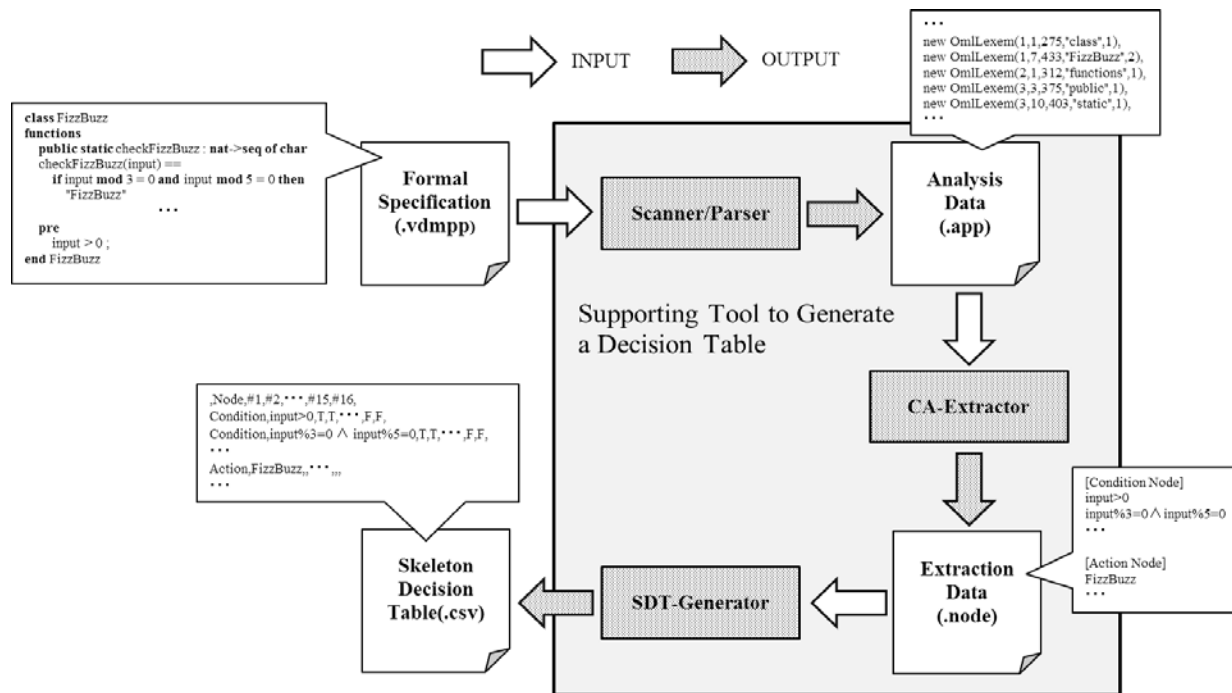


Fig. 1: The flow of supporting method to generate a decision table from the formal specification.

This paper proposes a supporting method to generate a decision table from the formal specification in order to improve efficiency of the test design with formal methods. This paper uses the formal specification description language VDM++ which is the lightweight formal methods VDM (Vienna Development Method) to write the formal specification. The proposed method supports to generate a decision table by extracting the logical relationship of the conditions and actions from a specification written with VDM++. We have implemented a supporting tool to generate a decision table. It automatically generates a skeleton decision table from the formal specification. By using the tool, it is considered that the efficiency of the test design is improved.

Here, the skeleton decision table which the tool generates has condition items, action items, and combinations of truth-values for the condition items. This means that a user must add to write truth-values in action items of the generated skeleton decision table to complete the decision table.

## 2. Supporting Method to Generate a Decision Table from the Formal Specification

Fig.1 shows the flow of the proposed method. The proposed method consists of three parts as follows.

- (i) Scanner/Parser
- (ii) CA-Extractor
- (iii) SDT-Generator

We explain the flow of the proposed method as follows.

- (1) A user prepares a formal specification after it is finished the syntax checking and type checking.
- (2) A user runs the supporting tool to generate a decision table after the user specified a formal specification.
- (3) The supporting tool to generate a decision table runs Scanner/Parser, which uses a formal specification specified by the user as an input.
- (4) Scanner/Parser generates the analysis data by analyzing the formal specification.
- (5) CA-Extractor generates the extraction data extracted conditions and actions of the formal specification from the analysis data by generated Scanner/Parser.
- (6) SDT-Generator generates a skeleton decision table from the extraction data by generated CA-Extractor, and output it as a csv file.

We explain each part as follows.

### 2.1. Scanner/Parser

Scanner/Parser is a VDM++ analyzer for Overture implemented by Marcel Verhoef [4]. Scanner/Parser

reads the formal specification specified by a user. Scanner/Parser generates the analysis data (.app) from the formal specification.

The analysis data has analysis information (e.g., token, token ID, and abstract syntax tree) needed by CA-Extractor..

Table 2:Extraction processes of conditions and actions.

Extraction of conditions
<ul style="list-style-type: none"> <li>Conditions extraction processes are pre-conditions expressions, post-conditions expressions and if-then-else expressions.</li> <li>To extract conditions, CA-Extractor extracts between from a start token (if, else, pre, post) to an end token (then, ;) as a condition. e.g., if (elseif) conditionA then <math>\rightarrow</math> conditionA pre conditionB ; <math>\rightarrow</math> conditionB</li> </ul>
Extraction of actions
<ul style="list-style-type: none"> <li>Actions extraction processes are if-then-else expressions.</li> <li>To extract actions, CA-Extractor extracts just after a start token (then or else) as an action. e.g., else actionA <math>\rightarrow</math> actionA</li> </ul>

Table 1:An example output of the skeleton decision table.

	Node	#1	#2	...	#15	#16
Condition	A>0	T	T	...	F	F
Condition	B=true	T	T	...	F	F
...	...	...	...	...	...	...
Action	NAT	null	null	null	null	null
...	...	...	...	...	...	...

## 2.2. CA-Extractor

CA-Extractor extracts conditions and actions from the analysis data generated by Scanner/Parser, and writes them to extraction data (.node). CA-Extractor extracts conditions and actions only from pre-conditions expressions, post-conditions expressions and if-then-else expressions.

Table 1 shows extraction processes of conditions and actions.

## 2.3. SDT-Generator

SDT-Generator generates a skeleton decision table from the extraction data by generated CA-Extractor, and outputs it as a csv file. Output format of a skeleton decision table is adopted CSV format because it is versatile.

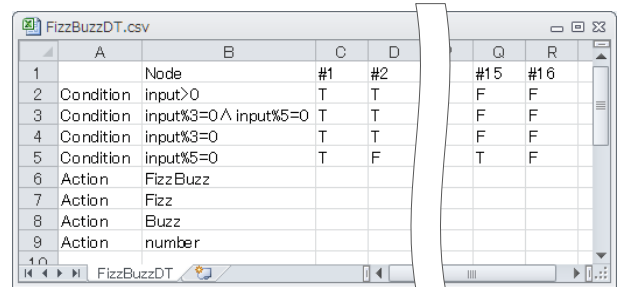
Table 2 shows an example of a skeleton decision table which SDT-Generator outputs. SDT-Generator

```

1 class FizzBuzz
2 functions
3   public static checkFizzBuzz : nat->seq of char
4   checkFizzBuzz(input) ==
5     if input mod 3 = 0 and input mod 5 = 0 then
6       "FizzBuzz"
7     elseif input mod 3 = 0 then
8       "Fizz"
9     elseif input mod 5 = 0 then
10      "Buzz"
11    else
12      "number"
13  pre
14    input > 0 ;
15 end FizzBuzz

```

Fig. 2:The formal specification of FizzBuzz.



A	B	#1	#2	#15	#16
Node	input>0	T	T	F	F
Condition	input%3=0 ^ input%5=0	T	T	F	F
Condition	input%3=0	T	T	F	F
Condition	input%5=0	T	F	T	F
Action	Fizz Buzz				
Action	Fizz				
Action	Buzz				
Action	number				

Fig. 3:A part of the skeleton decision table.

describes condition items and action items on the second column of a skeleton decision table. SDT-Generator describes truth-values in condition items on and after the third column of a skeleton decision table. The skeleton decision table of truth-values in action items is empty, because the supporting tool to generate a decision table cannot generate truth-values in action items.

## 3. Application Example

We have implemented a supporting tool to generate a decision table, in order to realize the proposed method. It supports the test design. It automatically generates a skeleton decision table from the formal specification.

We confirm that it works properly by adapting it to an example. Specifically, we confirm two items as follows.

- To extract conditions and actions from a formal specification
- To generate truth-values in condition items

Fig.2 shows the formal specification which is used as an application example. It is a specification of FizzBuzz, which is described in VDM++. FizzBuzz is a group word game. Players take turns to count incrementally, replacing any number divisible by three

with the word "fizz", and any number divisible by five with the word "buzz".

The formal specification of FizzBuzz in Fig.2 is described a pre-condition (i.e., in the 13th and 14th line) which means a value of the input is more than zero.

Fig.3 shows a part of the skeleton decision table generated by the supporting tool, and it is the result of applying the formal specification to the supporting tool.

We have confirmed that the skeleton decision table in Fig.3 is satisfied (i) and (ii). Therefore, we have confirmed that the supporting tool to generate a decision table works properly.

## 4. Discussion

This paper has proposed a supporting method to generate a decision table from the formal specification in order to improve efficiency of the test design with formal methods. We have implemented a supporting tool to generate a decision table, in order to realize the proposed method. It automatically generates a skeleton decision table from the formal specification. By using the tool, it is considered that the efficiency of the test design is improved.

We discuss our proposed method in this chapter.

### 4.1. Problems of the supporting tool to generate a decision table

We show problems of the supporting tool to generate a decision table as follows.

- Generating truth-values in action items

The supporting tool to generate a decision table cannot automatically generate truth-values in action items. This means that a user must add to write truth-values in action items of the skeleton decision table, in order to complete the decision table. If condition items increases, it takes much time and effort to describe manually truth-values in action items.

- Supporting to each syntax: cases expressions, for loop, and while loop, and so on

There is a limit to the formal specification applicable to the supporting tool to generate a decision table. Specifically, the supporting tool does not support to all syntax in VDM++. Hence, targets of conditions and actions extracted from a formal specification are pre-conditions expressions, post-conditions expressions and if-then-else expressions.

### 4.2. Evaluation of the proposed method

We confirm usefulness of the proposed method by using the generated skeleton decision table in chapter 3. First, we have completed the decision table by describing truth-values in action items of the skeleton decision table by hand. Second, we have implemented FizzBuzz in Java from the formal specification used as an application example in chapter 3.

We have tested FizzBuzz program by using the completed decision table, and at the result, we have been able to test all combinations of conditions and actions in FizzBuzz. Therefore, we have confirmed the usefulness of the proposed method.

### 4.3. Discussion about the supporting tool to generate a decision table

The supporting tool to generate a decision table extracts conditions and actions from a formal specification, and automatically generates a skeleton decision table. It is able to reduce the time and effort when testers manually design the testing.

Few researches of test design from the formal specification are reported, and the method is not well established. In addition, some tools to automatically generate a decision table are proposed [5, 6], but no tool to support the test design from a formal specification such as our proposed method.

By using the supporting tool to generate a decision table, it is considered that the efficiency of the test design from a formal specification is improved.

## 5. Conclusions

This paper proposed a supporting method to generate a decision table from the formal specification in order to improve efficiency of the test design with formal methods. We have implemented a supporting tool to generate a decision table, in order to realize the proposed method. The supporting tool generates a skeleton decision table from the formal specification by extracting conditions and actions of the formal specification.

We have confirmed that it generates the skeleton decision table and truth-value in conditions and actions of the skeleton decision table. By using the supporting tool to generate a decision table, it is considered that the efficiency of the test design from a formal specification is improved.

Future issues are as follows.

- Generating truth-values in action items
- Supporting to each syntax: cases expressions, for loop, and while loop, and so on
- Comparing with a decision table described by hand

## **References**

1. John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, Marcel Verhoef (2005), Validated Designs for Object-Oriented Systems, Springer.
2. Shin Nakajima (2007), Formal Methods as Software Engineering Tools (in Japanese), NII Technical Report, NII-2007-007J.
3. ISO 5806, Specification of single-hit decision tables.
4. A Scanner/Parser for the Overture Toolset, <http://overturetool.hosting.west.nl/twiki/bin/view/Main/OvertureParser/> (accessed November 29, 2013).
5. CEGTest, <http://softest.jp/tools/CEGTest/> (accessed November 29, 2013).
6. PictMaster, <http://sourceforge.jp/projects/pictmaster/> (accessed November 29, 2013).