Journal of Robotics, Networking and Artificial Life, Vol. 2, No. 3 (December 2015), 205-208

# Prototype of a Decision Table Generation Tool from the Formal Specification

**Tetsuro Katayama[*], Kenta Nishikawa[*], Yoshihiro Kita[†]**
**Hisaaki Yamaba[*], Kentaro Aburada[‡] and Naonobu Okazaki[*]**
[*]*University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192 Japan*
[†]*Kanagawa Institute of Technology, 1030 Shimo-ogino, Kanagawa, 243-0292 Japan*
[‡]*Oita National College of Technology, 1666 Maki, Oita, 870-0152 Japan*
*E-mail: kat@cs.miyazaki-u.ac.jp, nishikawa@earth.cs.miyazaki-u.ac.jp, kita@earth.cs.miyazaki-u.ac.jp, yamaba@cs.miyazaki-u.ac.jp, aburada@oita-ct.ac.jp, oka@cs.miyazaki-u.ac.jp*

**Abstract**

This research has implemented a prototype of a decision table generation tool from the specification (the formal specification) described in a formal specification language. This paper uses the formal specification description language VDM++ which is the lightweight formal methods VDM (Vienna Development Method) to write the formal specification. We applied some general specifications to the prototype, in order to evaluate its usefulness. As a result, the prototype has improved the efficiency in test design with formal methods.

*Keywords*: automatic generation, decision table, formal method, formal specification, VDM++, test design

## 1. Introduction

In recent years, the software quality cannot be maintained with the conventional software development methods because software 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.

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

In general, many defects are embedded in the upstream process of the software development.[1] 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 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 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 design the decision table. It is needed to extract test items and understand contents written on specifications. It is no exception even if you write strict specifications with formal methods.

This research has implemented a prototype of a decision table generation tool from the specification (the formal specification) described in a formal specification language, in order to improve efficiency of the test design with formal methods.[4] This prototype generates a decision table from the formal specifications, and displays it. This paper uses the formal specification description language VDM++ which is the lightweight

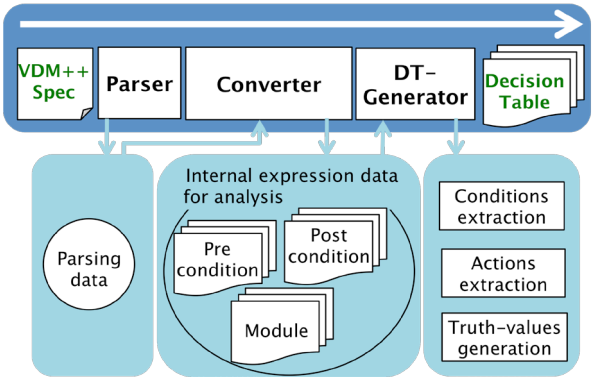


Fig. 1. A process to generate a decision table of the prototype.

Table 1. Extraction rules.

| Pattern of condition extraction | Pattern of action extraction |
|---|---|
| if "condition" then<br>elseif "condition" then | then "action" elseif<br>then "action" else<br>else "action" if<br>else "action" elseif<br>else "action" else<br>else "action" EOF |
| cases "condition" -> | -> "action" cases<br>others "action" EOF |
| pre "condition" post<br>pre "condition" EOF<br>post "condition" EOF | EOF(End Of File) |

formal methods VDM (Vienna Development Method) to write the formal specification.

## 2. Process to Generate a Decision Table of the Prototype

Fig. 1 shows a process to generate a decision table of the prototype. The prototype consists of three parts: Parser, Converter, and DT-Generator. We use a parser of Overture Toolset as a Parser.[5]

First, Parser reads a VDM++ specification inputted by a user, parses VDM++ specification, and outputs a parsing data. The parsing data has an abstract syntax tree and tokens.

Next, Converter converts the parsing data into an internal expression data for analysis by a module unit. The internal expression data for analysis is a data converted an abstract syntax tree of a parsing data into information suitable for analysis such as the division of a module or the correspondence of "if" and "else".

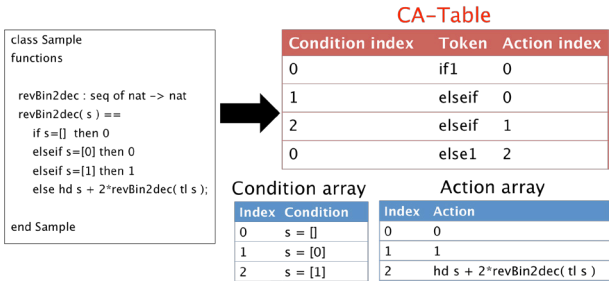


Fig. 2. An example of CA-Table.

Then, DT-Generator extracts conditions and actions from an internal expression data for analysis. After, DT-Generator stores conditions and actions in an array of String type. Table 1 shows extraction rules of conditions and actions. DT-Generator makes CA-Table, when DT-Generator extracts conditions and actions. Fig. 2 shows an example of CA-Table. CA-Table is a table which is correspondence of conditions and actions. CA-Table is three columns of condition index, token, and action index. DT-Generator generates truth-values based on this CA-Table.

We show truth-values generating process as follows.

(i) Make an array to store truth-values
(ii) Select the first column of this array
(iii) Select a row of CA-Table
(iv) Compare a token of the selected row of CA-Table
  (a) If this token matches "if", "elseif", or "cases"
    (I) Store "Y" into the condition index row of this column, then store "N" from the next column to the last column
    (II) Store "X" into the action index row of this column
  (b) If this token matches "else", or "others"
    (I) Store "N" into the condition index row of this column, then store "-" from the next column to the last column
    (II) Store "X" into the action index row of this column
(v) If there is a row that we have not yet selected, we select the next column of this array and return to third step. Otherwise truth-values is filled

Finally, DT-Generator generates a decision table from conditions, actions, and truth-values.

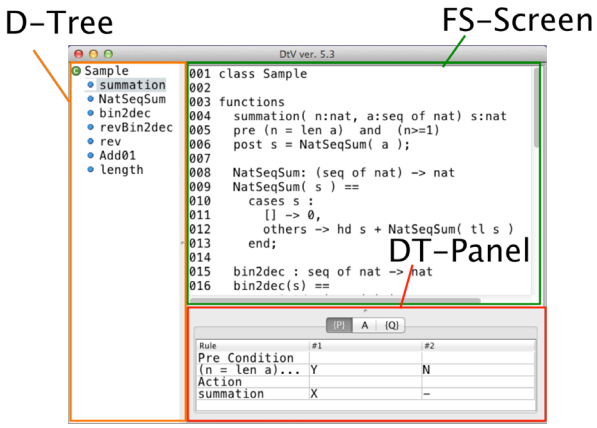


Fig. 3. Overview of the prototype.

## 3. Overview of the Prototype and Application Example

Fig. 3 shows overview of the prototype. The prototype consists of three displays: D-Tree, FS-Screen, and DT-Panel. FS-Screen displays a VDM++ specification inputted by a user. DT-Panel consists of three tabs: P-Tab, A-Tab, and Q-Tab. P-Tab displays a decision table of preconditions. A-Tab displays a decision table of a module. Q-Tab displays a decision table of post conditions. D-Tree displays a list of the definition names of the VDM++ specification. D-Tree redraws a decision table, when a user selects any the definition names of a VDM++ specification.

We confirm that this prototype works properly by adapting it to an example. Fig. 4 shows an example formal specification. It stores a positive integer X in a binary expression array b. Here, the highest-order digit must become one.

Fig. 5 shows the application example results. These results shows that this prototype extracts conditions and actions from a specification. Also, these results shows that this prototype generates truth-values. Therefore, we have confirmed that this prototype works properly.

## 4. Discussion

This research has implemented a prototype of a decision table generation tool from the formal specification, in order to improve efficiency of the test design with

```
class Dec2bin
functions

 dec2bin( X : nat ) b : seq of nat
 pre  X >= 1

 post  (forall i in set inds b & b(i)=1 or b(i)=0)  and
      b(1) = 1  and
      X = NatSeqSum( [ b(i)*2**(len b - i) | i in set inds b ] );

 NatSeqSum : seq of nat -> nat
 NatSeqSum( s ) ==
  cases s :
   [] -> 0,
   others -> hd s + NatSeqSum( tl s )
  end;

end Dec2bin
```

Fig. 4. The formal specification of converting decimal to binary.

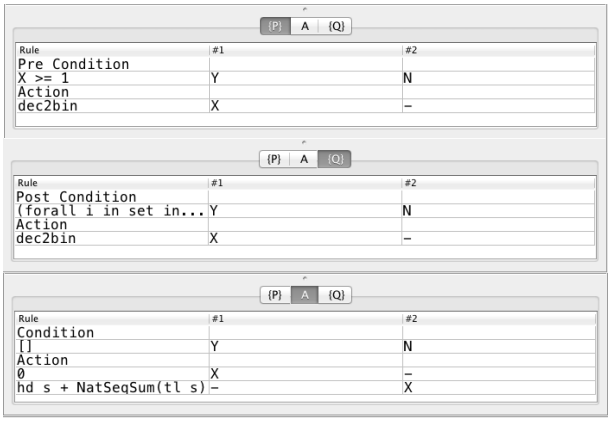


Fig. 5. The application example results.

formal methods. This prototype generates a decision table from the formal specifications, and displays it.

We discuss our prototype in this chapter.

### 4.1. *Evaluation of the usefulness*

We confirm the usefulness of this prototype by using the examinees.

Specifically, we apply three specifications, which is ways of combination of truth-values of the conditions are different. Then, we measure the time of examinees

Table 2. The measured results.

| Formal Specifications (ways of combination of truth-values of the conditions) | Examinee A (sec) | Examinee B (sec) | Examinee C (sec) | Examinee D (sec) | Examinee E (sec) | Average (sec) | Prototype (sec) |
|---|---|---|---|---|---|---|---|
| Specification A (4) | 252 | 213 | 169 | 240 | 134 | 216 | 0.012 |
| Specification B (16) | 405 | 559 | 499 | 435 | 557 | 498 | 0.016 |
| Specification C (256) | 1131 | 1292 | 1194 | 1859 | 1397 | 1629 | 0.02 |

and the prototype, which is required until completion for the decision table, and compares it.

Table 2 shows the measured results. By using the tool, we could automatically generate a decision table which has 256 ways of combination of truth-values of the conditions in about 20 milliseconds.

That is, we have confirmed the usefulness of this prototype.

### 4.2. *Related work*

Few researches of test design from the formal specification are reported[6], and the method is not well established.

Also, CEGTest[7] is a tool supporting the generation of the decision table. CEGTest automatically generates from a cause effect graph created by a user. However, a user must make a cause effect graph created, manually. Therefore, it takes much time and effort. It is needed to extract test items and understand contents written on the formal specification.

In addition, some test tools that inputs the formal specification are proposed[8, 9], but those tools do not support the generation of the decision table from a formal specification such as our prototype. In contrast, our prototype can automatically get a decision table from the formal specification inputted by a user.

## 5. Conclusions

This research has implemented a prototype of a decision table generation tool from the formal specification, in order to improve efficiency of the test design with formal methods. This prototype generates a decision table from the formal specifications, and displays it.

We have confirmed that our prototype extracts conditions and actions from the formal specification. Also, we confirmed that the prototype generates truth-values.

By using the tool, we could automatically generate a decision table which has 256 ways of combination of truth-values of the conditions in about 20 milliseconds.

Future issues are as follows.

- The usefulness improvement of the prototype
- Application to large-scale system specifications
- Automatic generation of test data
- Expansion to other test design techniques

## Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 24220001.

## References

1. G. Tassey, The Economic Impacts of Inadequate Infrastructure for Software Testing, National Institute of Standards and Technology, Planning Report 02-3 (2002).
2. John Fitzgerald, Peter Gorm Larsen, Paul Mukherjee, Nico Plat, Marcel Verhoef, Validated Designs for Object-Oriented Systems, Springer (2005).
3. ISO 5806, Specification of single-hit decision tables.
4. Kenta Nishikawa, Tetsuro Katayama, Yoshihiro Kita, Hisaaki Yamaba and Naonobu Okazaki, Proposal of a Supporting Method to Generate a Decision Table from the Formal Specification, *International Conference on Artificial Life and Robotics*, (2014) 222-225.
5. A Scanner/Parser for the Overture Toolset, http://overturetool.hosting.west.nl/twiki/bin/view/Main/OvertureParser/ (accessed October 30, 2014).
6. Jeremy Dick, Alain Faiver, Automating the Generation and Sequencing of Test Cases from Model-Based Specifications, *FME'93: Industrial-Strength Formal Methods*, Lecture Notes in Computer Science, **670** (1993) 268-284.
7. CEGTest, http://softest.jp/tools/CEGTest/ (accessed October 30, 2014).
8. Y. Ledru, L. du Bousquet, O. Maury, and P. Bontron, Filtering TOBIAS combinatorial test suites, *Fundamental Approaches to Software Engineering*, (2004) 281-294.
9. Adriana Sucena Santos, Combinatorial Test Automation Support for VDM++, *VDM/Overture Workshop*, (2008) 45-53.