Extension to Support Types and Operation/Function Definitions in BWDM to Generate Test Case Tool from the VDM++ Specification

Shota Takakura*, Tetsuro Katayama*, Yoshihiro Kita†,
Hisaaki Yamaba*, Kentaro Aburada*, and Naonobu Okazaki*
**Department of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192 Japan*
†Department of Information Security, Faculty of Information Systems, Siebold Campus, University of Nagasaki 1-1-1 Manabino, Nagayo-cho, Nishi-Sonogi-gun, Nagasaki, 851-2195 Japan
E-mail: takakura@earth.cs.miyazaki-u.ac.jp, kat@cs.miyazaki-u.ac.jp, kita@sun.ac.jp, yamaba@cs.miyazaki-u.ac.jp, aburada@cs.miyazaki-u.ac.jp, oka@cs.miyazaki-u.ac.jp

Abstract

Generating test cases from the formal specification description VDM++, which is a method for disambiguating specifications, is time-consuming and labor intensive. Therefore, our laboratory has developed BWDM, a tool that automatically generates test cases from VDM++ specifications. However, existing BWDM has problems that it only supports integer types and cannot generate test cases for operation and function definitions including recursive structure. Therefore, in order to improve the usefulness of BWDM, this paper extends BWDM to solve the above problems. Consequently, it has confirmed that the use of extended BWDM can reduce the test case generation time compared to manual test case generation.

Keywords: software testing, formal methods, VDM++, automatic generation, test cases

1. Introduction

Today, our lives are supported by a lot of software. As a result, software has become larger and more complex, and software bugs have a greater impact on society. One of the causes of bugs in software is the use of natural language in the upstream stage of software development. This is because natural language contains ambiguity. One method to solve this problem is to design software using formal methods. VDM is one of the most common formal methods used for this purpose, and VDM++ is a formal specification description language used in VDM for object-oriented modeling [1].

On the other hand, software design using VDM++ requires software testing. However, it is time-consuming and labor-intensive to manually generate test cases from specifications written in VDM++. In addition, manual test case generation may cause leakage in testing. Therefore, our laboratory has developed BWDM [2], [3], an automatic test case generation tool for VDM++ specifications. However, existing BWDM has the problem that it only supports integer types and cannot generate test cases for operation and function definitions with recursive structure.

Therefore, to improve the usefulness of BWDM, it is extended as follows.

- Addition of a function to generate test cases for enumerated type
- Addition of a function to generate test cases for operations/functions with recursive structure

In this paper, Chapter 2 describes the existing BWDM, Chapter 3 describes the extended BWDM, Chapter 4 shows application examples, and Chapter 5 discusses and confirms the usefulness of BWDM. Chapter 6 is the conclusion.

2. BWDM

BWDM is a tool that automatically generates test cases using the VDM++ specification as input. As examples, Fig. 1 shows the VDM++ specification and Fig.2 shows the test cases generated from it by BWDM, respectively.

3. The Extended BWDM

This chapter describes the details of the extension of BWDM. Fig. 3 shows the structure of the extended BWDM.

```
Class calendar
functions
judgeLeapYear: nat -> seq of char
judgeLeapYear(year) ==
  if(year mod 4 = 0 and ((year mod 100 = 0) =>
    (year mod 400 = 0))) then
  "Leap Year"
  else "Normal Year "
end calendar
```

Fig. 1 Example of VDM++ specification

3.1. *Addition of a Function to Generate Test Cases for Enumerated Type*

To solve the problem of the existing BWDM only supporting specific types, we extend Syntax Analyzer and Symbolic Executor. Concretely, we extend BWDM to generate test cases for enumerated type consisting of a set of unique identifiers.

In the extended Syntax Analyzer, if a type definition of an enumerated type exists in the abstract syntax tree analyzing process, the corresponding type declaration is replaced with a list consisting of the declared name and values. If the substituted list exists in the syntax tree in the Symbolic Executor, JavaAPI [4] which generates enumerated type is used to generate constraints to be registered with the SMT solver [5] used in the propositional analyzer section. This additional process can generate input data for enumerated type by symbolic execution.

3.2. *Addition of a Function to Generate Test Cases for Operations/Functions with Recursive Structure*

To solve the problem of existing BWDM's inability to generate test cases for operation and function definitions with recursive structure, we extend Syntax Analyzer and Test Suite Generator.

In existing BWDM, when there is an operation/function call in test suite generation, test cases cannot be generated unless the parsing of the called operation/function is completed. This is the cause that the parsing of the operation/function including recursive calls is not completed and the abstract syntax tree cannot be generated.

Therefore, when there is an operation or function that has self-recursive calls, an upper limit is set on the number of recursive calls, and the abstract parse tree analysis process is modified so that it is executed after the parse tree analysis for all definitions is completed. This allows the test data generation to be performed after the parsing of the VDM++ specification given as input is completed. Additionally, a function is added that terminates the process if self-recursive calls are performed more than a limited number of times when recursively generating output data in the output data generation section in Test Suite Generator.

```
Function Name : judgeLeapYear
Argument Type : current_year : nat
Return Type : seq of (char)
Number of Test Cases : 17 cases(BVA:13 /SE:4)

Boundary Values for Each Argument
current_year : 4294967295 4294967294 0 -1 3 4
5 99 100 101 399 400 401

Test Cases for Boundary Value Analysis
No.1 : 4294967295 -> Undefined Action
No.2 : 4294967294 -> "Normal year"
No.3 : 0 -> "Leap year"
No.4 : -1 -> Undefined Action
No.5 : 3 -> "Normal year"
No.6 : 4 -> "Leap year"
No.7 : 5 -> "Normal year"
No.8 : 99 -> "Normal year"
No.9 : 100 -> "Normal year"
No.10 : 101 -> "Normal year"
No.11 : 399 -> "Normal year"
No.12 : 400 -> "Leap year"
No.13 : 401 -> "Normal year"
```

Fig. 2 Output when Fig.1 is applied to the BWDM

4. Application Examples

This chapter shows application examples to confirm that the extended BWDM works correctly.

4.1. *Confirmation that Test Cases can be Generated for Enumerated Type*

Fig. 4, shows the VDM++ specification using enumerated type used for verification, and Fig.5 shows the results of applying it to the extended BWDM, respectively.

From Fig. 4, it can see that in the "judgeLightColor" function, "public TraficLight = <BLUE>|<YELLOW>|<RED>;" is defined as the type. As output, Fig.5 shows that test cases can be generated for the definition using the enumerated type.

4.2. *Confirmation that Test Cases can be Generated for Definitions with Recursive Structure*

Fig. 6 shows the VDM++ specification using a definition with recursive structure used for verification, and Fig 7

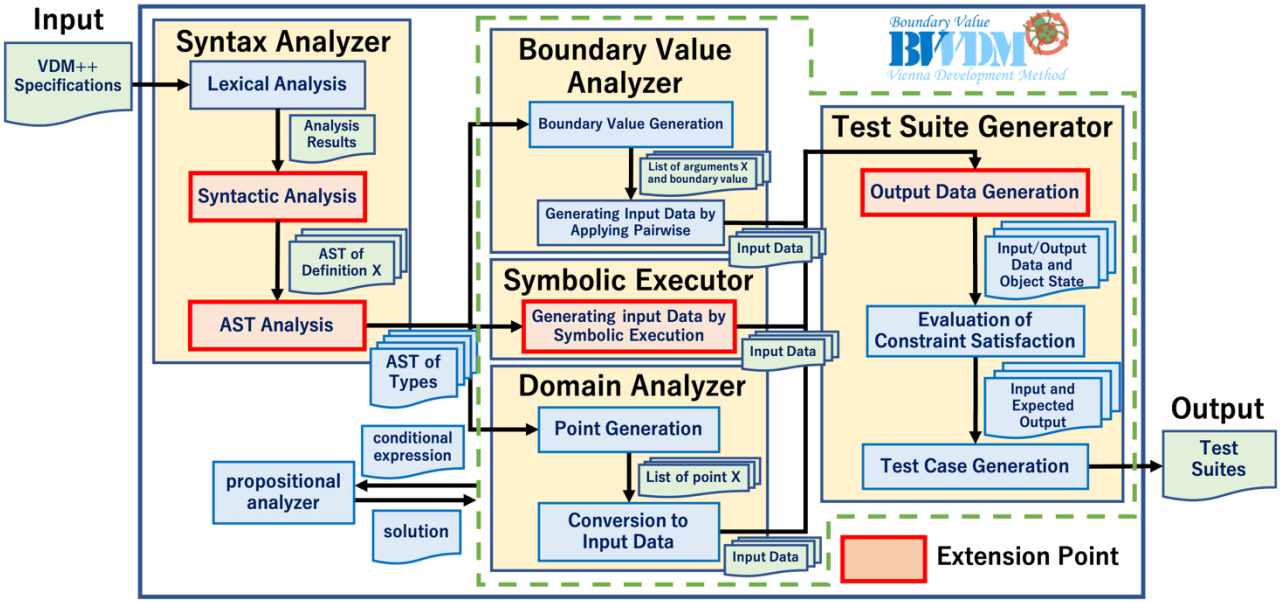

Fig. 3 The structure of the extended BWDM

```
Class judgeLightColor

Types
public TrafficLight = <BLUE> | <YELLOW> | <RED>;
functions
 judgeLightColor: TrafficLight ==> seq of char
  judgeLightCoulor (color) ==
    if color = <BLUE >
      "The color is blue. "
    if color = <YELLOW>
      "The color is yellow"
    if color = <RED>
      "The color is red."

end judgeLightColor
```

Fig. 4 VDM++ specification using enumerated type

shows the results of applying it to the extended BWDM, respectively.

From Fig. 6, it can see that in the "calcSum" function is defined with recursive structure. As output, Fig.7 shows that test cases can be generated for definition with recursive structure.

5. Discussion

5.1. *Evaluation of Test Cases Generation for Definitions using Enum Type*

As shown in Section 4.1, we have confirmed that the extended BWDM can generate test cases from definitions using enumerated type in the VDM++ specification of Fig. 4. Therefore, it can be said that this extension enables BWDM to generate test cases from definitions using

```
Function Name : judgeLightColor
Argument Type : color:
Return Type : seq of (char)

Test Cases by Symbolic Execution
No.1 : <BLUE> -> "The color is blue"
No.2 : <YELLOW> -> "The color is yellow"
No.3 : <RED> -> "The clolor is red"
```

Fig. 5 Output when Fig.4 is applied to the extended BWDM

enumerated type. Thus, the usefulness of BWDM has been improved.

5.2. *Evaluation of Test Case Generation for Definitions with Recursive Structure*

As shown in Section 4.2, we have confirmed that the extended BWDM can generate test cases from definition with recursive structure in the VDM++ specification of Fig. 6. Therefore, it can be said that this extension enables BWDM to generate test cases from definitions with recursive structure. Thus, the usefulness of BWDM has been improved.

5.3. *Comparison with Manual Test Case Generation Time*

Using the extended BWDM, we evaluate the test case generation time for the VDM++ specifications which include definitions with enumerated type and recursive structure, by comparing it to the human case. Two VDM++ specifications, Fig. 4 and Fig. 6, are used in the

```
Class calcSum

  factorial: nat ==> nat
    factorial (value) ==
      if value = 0 then
        0
      else value + factorial (value − 1)

end calcSum
```

Fig. 6 VDM++ specification with recursive structure

```
Function Name : calcSum
Argument Type : value:nat
Return Type : nat

Boundary Values for Each Argument
value: 4294967295 4294967294 0 −1

Test Cases of Boundary Values
No.1 : 4294967295 −> Undefined Action
No.2 : 4294967294 −> Undefined Action
No.3 : 0 −> 0
No.4 : −1 −> Undefined Action
```

Fig. 7 Output when Fig.6 is applied to the extended BWDM

experiment. The manual verification is performed by a total of five people, two graduate students and three fourth-year undergraduates in our laboratory. We measure the time required to generate sufficient test cases from the VDM++ specification, which are no leakage. The time measurement is terminated when the subjects have generated the correct test cases, and when the manually test cases includes any incorrect test cases, the mistakes are pointed out. The comparison results are shown in Table.1.

Table 1 shows that about 14 minutes are saved when using the extended BWDM to generate test cases compared to manual test case creation. In addition, human error has been observed in the manual test case creation.

In this research, we have confirmed that an extended BWDM that adds a test case generation function for definitions using enumerated type and recursive structure can reduce the time and human errors required for test case generation. Therefore, it can be said that the usefulness of BWDM has been improved.

Table.1 Comparison of test case generation time

	Time
Average of 10 subjects	14m29s
BWDM	2.6s

6. Conclusion

In order to improve the usefulness of BWDM, an automatic test case generation tool for the VDM++ specification, two extensions have been made to the existing BWDM to solve the problems that only supports integer types and that it cannot generate test cases for operation and function definitions with recursive structure.

The application examples of the extended BWDM are shown, and the test case generation function for definitions using enumerated type and definitions with recursive structure for expanding the range of supported types is confirmed. We have confirmed that the extended BWDM can reduce test case generation time by about 14 minutes compared to manually generating test cases from VDM++ specifications. In addition, we have confirmed that the extended BWDM can eliminate human errors.

Therefore, it can be said that the usefulness of BWDM has been improved by the extensions in this paper.

The future issues are the following.

- Support for types other than integer and enumerated type
- Test case generation for input values with more recursive calls
- Test case generation for mutually recursive functions

References

1. Overture Project. Manuals. http://overturetool.org/documentation/manuals.html. Accessed: 2023-12-13.
2. T. Katayama, F. Hirakoba, Y. Kita, H. Yamaba, K. Aburada, and N. Okazaki. Application of Pirwise Tsting into BWDM which is a Test Case Generation tool for the VDM++ Specification. Journal of Robotics, Networking and Artificial Life, Vol.6, No.3, pp.143-147, 2019.
3. T. Muto, T. Katayama, Y. Kita, H. Yamaba, K. Aburada, and N. Okazaki. Expansion of Application Scope and Addition of a Function for Operations into BWDM which is an Automatic Test Cases Generation Tool for VDM++ Specification. Journal of Robotics, Networking and Artificial, Vol.9. No.3, pp.255-262, 2022.
4. Z3:Packagecom.microsoft.z3. https://z3prover.github.io/api/html/namespacecom_1_1microsoft_1_1z3.html. Accessed: 2023-12-13.
5. Z3. https://github.com/Z3Prover/z3. Accessed: 2023-12-14.

Authors Introduction

Mr. Shota Takakura

Shota Takakura received the Bachelor's degree in engineering (computer science and systems engineering) from the University of Miyazaki, Japan in 2023. He is currently a Master's student in Graduate School of Engineering at the University of Miyazaki, Japan. His research interests include software testing, software quality, and formal method.

.

Dr. Tetsuro Katayama

He received a Ph.D. degree in engineering from Kyushu University, Fukuoka, Japan, in 1996. From 1996 to 2000, he has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2000 he has been an Associate Professor at the Faculty of Engineering, Miyazaki University, Japan. He is currently a Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include software testing and quality. He is a member of the IPSJ, IEICE, and JSSST.

Dr. Yoshihiro Kita

Yoshihiro Kita received a Ph.D. degree in systems engineering from the University of Miyazaki, Japan, in 2011. He is currently an Associate Professor with the Faculty of Information Systems, University of Nagasaki, Japan. His research interests include software testing and biometrics authentication.

Dr. Hisaaki Yamaba

He received the B.S. and M.S. degrees in chemical engineering from the Tokyo Institute of Technology, Japan, in 1988 and 1990, respectively, and the Ph D. degree in systems engineering from the University of Miyazaki, Japan in 2011. He is currently an Assistant Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include network security and user authentication. He is a member of SICE and SCEJ.

Dr. Kentaro Aburada

He received the B.S., M.S, and Ph.D. degrees in computer science and system engineering from the University of Miyazaki, Japan, in 2003, 2005, and 2009, respectively. He is currently an Associate Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include computer networks and security. He is a member of IPSJ and IEICE.

Dr. Naonobu Okazaki

He received his B.S, M.S., and Ph.D. degrees in electrical and communication engineering from Tohoku University, Japan, in 1986, 1988 and 1992, respectively. He joined the Information Technology Research and Development Center, Mitsubishi Electric Corporation in 1991. He is currently a Professor with the Faculty of Engineering, University of Miyazaki since 2002. His research interests include mobile network and network security. He is a member of IPSJ, IEICE and IEEE