SUGISAKA MASANORI

Journal of Robotics, Networking and Artificial Life
Vol. 9(3); December (2022), pp. 255–262
ISSN (Online):2352-6386; ISSN (Print):2405-9021
https://alife-robotics.org/jrnal.html

Research Article

Expansion of Application Scope and Addition of a Function for Operations into BWDM which is an Automatic Test Cases Generation Tool for VDM++ Specification

Takafumi Muto[1], Tetsuro Katayama[1], Yoshihiro Kita[2], Hisaaki Yamaba[1], Kentaro Aburada[1], Naonobu Okazaki[1]
[1]*Department of Computer Science and Systems Engineering, Faculty of Engineering, University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192 Japan*
[2]*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*

ARTICLE INFO

Article History
Received 25 November, 2021
Accepted 14 September 2022

Keywords
Software testing
Formal methods
Test cases
VDM++
Automatic generation.

ABSTRACT

The use of the formal specification description language VDM++ in software design can eliminate ambiguity in the specification. However, software testing after implementation is necessary even if the design uses VDM++, but manually generating test cases is labor-intensive and time-consuming. Therefore, our laboratory developed BWDM, which is an automatic test case generation tool for VDM++ specifications. However, BWDM is not very useful because it has three problems about its narrow scope of application. This paper solves the three problems and improves the usefulness of BWDM by expanding the scope of application of VDM++ definitions and adding a function to generate test cases for object states. In addition, we conducted a comparison experiment with manual test case generation and confirmed that BWDM can reduce work time.

1. Introduction

One of the methods to eliminate the ambiguity of specifications in software design is to use formal methods for software design[1]. One of the formal specification description languages is VDM++[2].

On the other hand, software testing after implementation is necessary for either design using natural language or formal methods, but manually generating test cases is labor-intensive and time-consuming. Therefore, we have developed BWDM, which is an automatic test cases generation tool for VDM++ specifications, in our laboratory[3],[4]. BWDM automatically generates test cases that can be used to perform boundary value testing, domain analysis testing, and testing based on structure recognition of if-then-else expressions.

However, BWDM has the following three problems.

- It does not support conditional expressions for invariant conditions and pre-conditions and post-conditions.
- It does not support type definition block.
- It is not possible to generate test cases for operation definitions that manipulate the object state.

Corresponding author's E-mail: muto@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

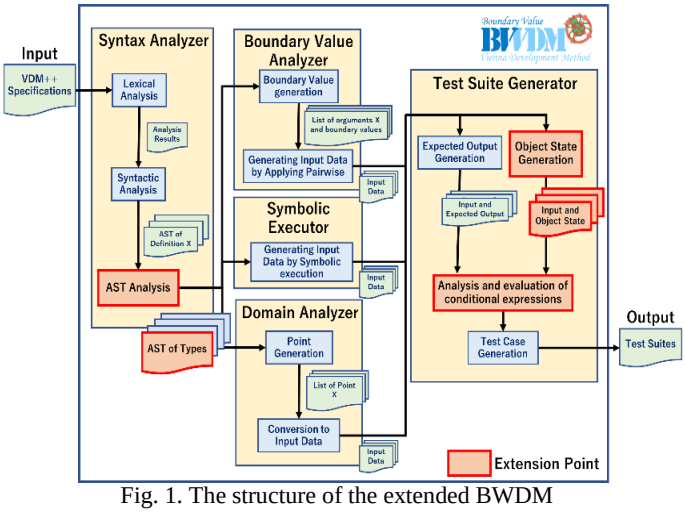

Fig. 1. The structure of the extended BWDM

Because of these three problems, the VDM++ specification targeted by the existing BWDM has many limitations and is not highly useful. Therefore, to improve the usefulness of BWDM, this paper extends BWDM to solve the above three problems.

2. The Extended BWDM

The structure of the extended BWDM is shown in Fig. 1.

2.1. *Analysis and Evaluation of Each Conditional Expressions in the Definition*

Existing BWDM has the problem that it does not support conditional expressions for invariant and pre-conditions and post-conditions. An example of a VDM++ specification including pre-conditions is shown in List 1.

In List 1, the pre-condition "point <= 100" is defined, but the existing BWDM cannot generate test cases with 100 and 101 as inputs, which are the boundary values of the pre-condition. In addition, when a value more than 100 is used as input, the pre-condition is not satisfied, so a test case with "Undefined Actions" as the expected output should be generated, but the existing BWDM sets the expected output as "A" and generates an incorrect test case.

To solve this problem, we extend the Syntax Analyzer and Test Suite Generator of BWDM.

The Syntax Analyzer is modified to obtain inputConditions, which store the conditional expressions needed to determine the input data, and outputConditions,

List 1. Example of a VDM++ specification including pre-conditions

```
class GradeEvaluation
 functions
  evaluateGrades : nat -> seq of char
  evaluateGrades (point) ==
   if(point >= 60) then
    if(point >= 70) then
     if(point >= 80) then
      if(point >= 90) then
       "A"
      else
       "B"
     else
      "C"
    else
     "D"
   else
    "F"
   pre point <= 100;
end GradeEvaluation
```

which are the conditional expressions needed to determine the expected output. inputConditions store pre-conditions and invariant conditions of argument type, and outputConditions store post-conditions. The inputConditions are passed to the Boundary Value Analyzer to obtain the input data.

The Test Suite Generator adds a process to evaluate the conditional expressions of inputConditions and outputConditions during the process of generating the expected output. If the conditional expression is false, it means that the conditional expression set in the definition is not satisfied at the beginning or end of the process, so "Undefined Action" is set as an expected output. If the conditional expression is true, the expected output is the same as the existing BWDM.

2.2. *Support Type Definition Block*

Existing BWDM has the problem that BWDM does not support type definition block. An example of a VDM++ specification including types is shown in List 2.

In List 2, the "MONTH" type is defined and the argument type of the "determineQuarter" function is of type MONTH. The existing BWDM does not support type definitions, so it cannot determine what type the "month" argument is and cannot generate a test case that

List 2. Example of a VDM++ specification including types

```
class Quarter
types
  public MONTH = nat1
  inv m == m <= 12;
functions
  determineQuarter : MONTH -> seq of char
  determineQuarter (month) ==
    if(month <= 3) then
      "Q1"
    else
      if(month <= 6) then
        "Q2"
      else
        if(month <= 9) then
          "Q3"
        else
          "Q4";
end Quarter
```

has as input the boundary value of the type and the boundary value of the invariant condition.

To solve this problem, the Syntax Analyzer is modified.

In the extended BWDM, the Syntax Analyzer keeps an abstract syntax tree of type definitions when it performs abstract parsing. When a type definition is used in each definition block, the type definition is converted to the actual type based on this abstract syntax tree. Furthermore, during conversion, conditional expressions for invariant conditions are added to inputConditions if the type to be converted is an argument type of an operation definition or function definition, or to outputConditions if it is an instance variable type.

2.3. *Addition of a Function to "Generate Test Cases for the Object State"*

Existing BWDM has the problem that it is not possible to generate test cases for operation definitions that manipulate the object state, it is not possible to generate test cases for specifications that manipulate instance variables. In other words, the existing BWDM only targets specifications in which the object state is always constant.

An example of a VDM++ specification including operation definitions that manipulate the object state is shown in List 3. In List 3, there is "payWithCardsAndCoupons" operation that manipulates

List 3. Example of a VDM++ specification including operation definitions that manipulate the object state

```
class Payment
types
  public yen = nat;
values
  cardUsageLimit: yen = 100000;
instance variables
  coupon: nat := 8;
  cardUsageAmount: yen := 0;
  inv cardUsageAmount <= cardUsageLimit;
operations
  payWithCardsAndCoupons: yen * nat ==> ()
  payWithCardsAndCoupons(amount, tickets) ==
     (cardUsageAmount :=
      cardUsageAmount + (amount - amount * (tickets
* 0.1));
      coupon := coupon - tickets)
  pre tickets <= 10
  post coupon~ = coupon + tickets;
functions
end Payment
```

the instance variable "coupon" and the instance variable "cardUsageAmount". When this VDM++ specification is applied to an existing BWDM, it does not generate a test case for the "payWithCardsAndCoupons" operation.

To solve this problem, we add a function to generate "Test Cases for the Object State".

The VDM++ specification sets invariant conditions for class and type definitions. In addition, it sets pre-conditions and post-conditions for operations definitions and functions definitions. The "Test Cases for the Object State" to be added in this paper is test cases that use these conditions to generate the expected output of whether the object state after the operation is "Normal" or "Failure", or whether there is an error in the input. If there is an error in the input, the expected output is set to "Undefined Action" as in the existing test cases generation. Table 1 shows the expected state and the conditions corresponding to the state.

In generating the test cases, the Test Suite Generator obtains the object state after the operation by using an arithmetic expression to be assigned to the instance variable and the value of the instance variable. After obtaining the object state after the operation, all conditional expressions stored in "inputConditions" are evaluated. If the result is false, "Undefined Action" is set

Table 1. The expected state and the conditions corresponding to the states

expected state	conditions
Normal	All conditionals are true
Failure	The invariant condition of the instance variables definition is false.
	Post-condition is false
	The value of the instance variable after the operation is outside the range of the type.
Undefined Action	Pre-condition is false
	The invariant condition of the type definition is false
	The value is outside the range of the argument type

as the expected output. Otherwise, then all conditional expressions stored in "outputConditions" are evaluated by applying the object state after the operation. If the result is false, "Failure" is set as the expected output. Otherwise, that is both "inputConditions" and "outputConditions" are true, "Normal" is the expected output.

3. Application Example

In this chapter, we confirm that the extended BWDM works correctly by using application examples.

3.1. *Confirmation of the Analysis and Evaluation of Each Conditional Expression in the Definition.*

To confirm that the pre-conditions have been supported, List 4 shows the output when the VDM++ specification in List 1 is applied to the extended BWDM.

In test cases No. 5 and 6 in List 4, we can confirm that they generate test cases with the pre-condition boundary values 100 and 101 as inputs. In test case No. 6, the expected output is "Undefined Action", which means that the test case evaluates that the pre-condition is not satisfied.

In addition, we confirmed that it can generate test cases for each conditional expression for the VDM++ specification including post-conditions and invariant conditions.

3.2. *Confirmation of Support Type Definition Block*

List 4. Output when List 1 is applied to the extended BWDM

```
Function Name : evaluateGrades
Argument Type : point:nat
Return Type : seq of (char)
Number of Test Cases : 19 cases(BVA:14/SE:5)

Boundary Values for Each Argument
point : 4294967295 4294967294 0 -1 100 101 60 59
70 69 80 79 90 89

Test Cases for Boundary Value Analysis
No.1 : 4294967295 − > Undefined Action
No.2 : 4294967294 − > Undefined Action
No.3 : 0 − > "F"
No.4 : -1 − > Undefined Action
No.5 : 100 − > "A"
No.6 : 101 − > Undefined Action
No.7 : 60 − > "D"
No.8 : 59 − > "F"
No.9 : 70 − > "C"
No.10 : 69 − > "D"
No.11 : 80 − > "B"
No.12 : 79 − > "C"
No.13 : 90 − > "A"
No.14 : 89 − > "B"
```

To confirm that the type definitions are supported, List 5 shows the output when the VDM++ specification in List 2 is applied to the extended BWDM.

Looking at the Argument Type part in List 5, we can confirm that the "month" argument is determined to be of type nat1. In test cases No. 5 and 6, we can confirm that the test cases are generated with 12 and 13 as inputs, which are the boundary values of the invariant condition "m <= 12" defined in the type definition "MONTH".

3.3. *Confirmation of Addition of a Function to Generate "Test Cases for the Object State"*

To confirm that test cases are correctly generated for operation definitions that manipulate the object state, List 6 shows the output when the VDM++ specification in List 3 is applied to the extended BWDM.

In the "payWithCardsAndCoupon" operation of List 3, if 0 and 10 are used as inputs, the value of the instance variable "coupon" after the operation will be -2. Since "coupon" is of type nat and the conditional expression "0 <= coupon" stored in outputConditions is false, the expected object state is "Failure". In test case

List 5. Output when List 2 is applied to the extended BWDM

```
Function Name : determineQuarter
Argument Type : month:nat1
Return Type : seq of (char)
Number of Test Cases : 16 cases(BVA:12/SE:4)

Boundary Values for Each Argument
month : 4294967296 4294967295 1 0 12 13 3 4 6 7 9
10

Test Cases for Boundary Value Analysis
No.1 : 4294967296 − > Undefined Action
No.2 : 4294967295 − > Undefined Action
No.3 : 1 -> “Q1”
No.4 : 0 − > Undefined Action
No.5 : 12 − > “Q4”
No.6 : 13 − > Undefined Action
(- Omission -)
```

No. 19 in List 6, the expected object state is "Failure" with 0 and 10 as inputs, so it can be confirmed that the test cases for the object state can be generated appropriately.

4. Discussion

4.1. *Evaluation on the Analysis and Evaluation of Each Conditional Expression in the Definition.*

We have confirmed that the extended BWDM can generate test cases corresponding to the conditional expressions in the invariant, pre-condition, and post-condition. This enables the generation of test cases corresponding to the conditional expressions set in the definitions, thus extending the range of applications of BWDM. Therefore, we can say that the usefulness of BWDM has been improved.

4.2. *Evaluation on Support Type Definition Block*

We have confirmed that the extended BWDM can generate test cases for VDM++ specifications using type definitions. As a result, the extended BWDM can generate test cases corresponding to the definitions described in the type definition block, and the application range of BWDM has been extended. Therefore, we can say that the usefulness of BWDM has been improved.

List 6. Output when List 3 is applied to the extended BWDM

```
Function Name : payWithCardsAndCoupons
Argument Type : amount:nat tickets:nat
Return Type : ()
Number of Test Cases : 24 cases

Boundary Values for Each Argument
amount : 4294967295 4294967294 0 -1
tickets : 4294967295 4294967294 0 -1 10 11

Test Cases for the Object State
(- Omission -)
No.11 : 0 0 -> Normal
No.12 : -1 0 -> Undefined Action
No.13 : 4294967295 -1 -> Undefined Action
No.14 : 4294967294 -1 -> Undefined Action
No.15 : 0 -1 -> Undefined Action
No.16 : -1 -1 -> Undefined Action
No.17 : 4294967295 10 -> Undefined Action
No.18 : 4294967294 10 -> Failure
No.19 : 0 10 -> Failure
No.20 : -1 10 -> Undefined Action
No.21 : 4294967295 11 -> Undefined Action
No.22 : 4294967294 11 -> Undefined Action
No.23 : 0 11 -> Undefined Action
No.24 : -1 11 -> Undefined Action
```

4.3. *Evaluation on Addition of a Function to Generate "Test Cases for the Object State"*

We have confirmed that the extended BWDM can generate "Test Cases for the Object State". As a result, the extended BWDM can generate test cases for operation definitions that manipulate the object state, which the existing BWDM cannot generate. Therefore, we can say that the usefulness of BWDM has been improved by the addition of a function to generate "Test Cases for the Object State" into BWDM.

4.4. *Comparison and Verification with Manual Test Cases Generation*

We compared and verified the generation time for "Test Cases for the Object States" of the extended BWDM with the manual case. The target VDM++ specification is the 20-line specification in List 3. The results of the comparative verification are shown in Table 2.

Table 2. Comparison of test case generation time by the extended BWDM and manual test case generation time for the specification in List 3

	Generation time
Average of 5 subjects	17m19s
BWDM	1.4s

Manual verification was conducted by a total of five people, two graduate students, and three fourth-year undergraduates, and the time required to finish writing all the necessary test cases was measured. If the test cases were inaccurate, we pointed out the mistakes, and the time measurement ended when the subjects wrote the correct test cases.

In the verification of the extended BWDM, the time required to run the extended BWDM on the command line and generate test cases was measured. The computer used for the verification was Windows 10 Pro OS, 3.6GHz Intel Core i7 CPU, and 16GB memory.

As shown in Table 2, for the 20-line VDM++ specification, the time required to generate test cases using the extended BWDM was reduced by about 17 minutes compared to generating test cases manually. In addition, human error was observed in the manual test case generation, such as incorrectly calculating the value of instance variables after the operation.

In the function to generate "Test Cases for the Object State" added in this paper, it was confirmed that the time required for test case generation, which was a feature of the existing BWDM, could be reduced and that human errors could be eliminated. Therefore, we can say that the usefulness of BWDM has been improved.

4.5. *Related Works*

A systematic literature review of research on automatic test case generation from requirement specifications is presented by Ahmad Mustafa et al[5]. They identify and discuss 30 primary studies on requirements-based generation from 410 studies. One of the significant lessons learned here is that most software testing errors can be attributed to errors in natural language requirements. Discovering ambiguities and incompleteness in natural language is difficult and is one of the key problems in natural language requirements.

In addition, most approaches to test case generation focus on UML as input for a generation[6],[7]. But it is impossible to capture all of the developed attributes of the system from the UML.

In contrast, BWDM uses the formal specification description language VDM++ as input for test case generation, a language with a clear and rigorous meaning based on mathematical logic that eliminates the ambiguities and incompleteness of natural language. Therefore, BWDM does not have the issues in test case generation from requirement specifications discussed by Ahmad Mustafa et al.

Aamer Nadeem et al. proposed a method of automatic test case generation for VDM++ specifications[8]. The method determines input data using pre-conditions and invariant conditions described in the instance variable definition block. The pre-conditions and invariant conditions are each equivalence partitioned, and a representative value is determined at random from the valid input domain as input data. In addition, a test sequence is generated by preparing a test descriptor defined as a valid test sequence. After generating the input data and test sequences, the two are combined to generate test cases. Here, test descriptors must be manually prepared.

In contrast, BWDM can generate test cases with only a VDM++ specification. In addition, test cases generated by BWDM can be used for boundary value testing and domain analysis testing. Furthermore, test cases with symbolic execution can be expected to test execution flows that cannot be covered by boundary value analysis.

5. Conclusion

In this paper, to improve the usefulness of BWDM, it has been extended to solve the three problems.

An example of the application to the extended BWDM is shown, and it is confirmed that the above three problems have been solved.

Furthermore, because of comparing and verifying the time required to generate "Test Cases for the Object State" manually, we could confirm a reduction of about 17 minutes by using the extended BWDM for the 20-line VDM++ specification used for the verification. In addition, human errors were observed in the manual test

case generation, but it was confirmed that human errors could be eliminated in the test case generation by using the extended BWDM.

From the above, the BWDM extended in this paper can be said to have improved its usefulness.

The following is a list of future tasks.

- Support for type other than integer type
- Support for conditional expressions that refer to the value after the operation

6. References

1. Flemming Nielson, Hanne Riis Nielson: Formal Methods: An Appetizer. Springer, 2019.
2. Overture Project. Manuals. http://overturetool.org/documentation/manuals.html. Accessed: 2021-12-13.
3. H. Tachiyama, T. Katayama, T. Oda. Automated Generation of Decision Table and Boundary values from VDM++ Specification. The 15th Overture Workshop: New Capa-bilities and Applications for Model-based Systems Engineering Technical Report Series, No. CS-TR-1513-2017, pp. 89-103, 2017.
4. 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.
5. Ahmad Mustafa, Wan M. N. Wan-Kadir, Noraini Ibrahim, Muhammad Arif Shah. Automated Test Case Generation from Requirements: A Systematic Literature Review. Computers, Materials & Continua, vol. 67, no.2, pp. 1819-1833, 2021.
6. M. Lafi, T. Alrawashed, A. M. Hammad. Automated Test Cases Generation From Requirements Specification, International Conference on Information Technology (ICIT), pp. 852-857, 2021.
7. Mauricio Rocha, Adenilso Simão, Thiago Sousa. Model-based test case generation from UML sequence diagrams using extended finite state machines. Software Quality Journal, Volume 29, Issue 3, pp.597–627, 2021.
8. Aamer Nadeem, Muhammad Jaffar-Ur-Rehman. Automated Test Case Generation from IFAD VDM++ Specifications. SEPADS 05: 4th WSEAS International Conference on Software Engineering, Parallel & Distributed Systems, No.28, pp.1-7, 2005.

Authors Introduction

Takafumi Muto

Takafumi Muto received the Bachelor's degree in engineering (computer science and systems engineering) from the University of Miyazaki, Japan in 2021. He is currently a Master's student in Graduate School of Engineering at the University of Miyazaki, Japan. His research interests software testing, software quality, and formal method.

Tetsuro Katayama

Tetsuro Katayama 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.

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.

Hisaaki Yamaba

Hisaaki Yamaba 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.

Kentaro Aburada

Kentaro Aburada 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.

Naonobu Okazaki

Naonobu Okazaki 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.