Expansion of Application Scope and Addition of a Function for Operations into BWDM to Generate Test Cases from VDM++ Specification

Takafumi Muto[*], Tetsuro Katayama[*], Yoshihiro Kita†,
Hisaaki Yamaba[*], Kentaro Aburada[*], 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: 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

Abstract

Generating test cases from the VDM++ specification to eliminate ambiguity in the specification 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 extends BWDM to solve three problems. In addition, we conducted a comparison experiment with manual test case generation and confirmed a time saving of about 17 minutes.

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

1. Introduction

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

On the other hand, software testing is also necessary in design using formal methods, but manually generating test cases is labor-intensive and time-consuming. Therefore, we have developed BWDM, which is an automatic test case generation tool for VDM++ specifications, in our laboratory[2,3]. 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 is not very useful because it has the following three problems about its narrow scope of application.

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

Therefore, in order 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*

To solve the problem that the existing BWDM does not support conditional expressions for invariant conditions, pre-conditions, and post-conditions set in the definition, we extend the Syntax Analyzer and Test Suite Generator of BWDM.

The Syntax Analyzer is modified to obtain inputConditions, which store the conditional expressions

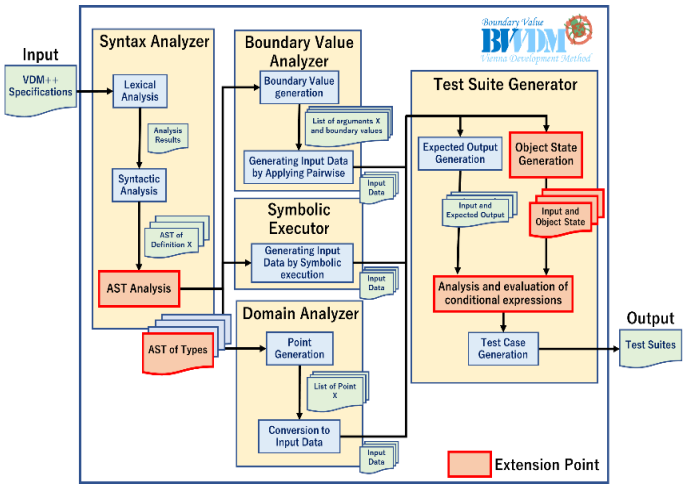

Fig. 1. The structure of the extended BWDM

needed to determine the input data, and outputConditions, which are the conditional expressions needed to determine the expected output. inputConditions store pre-conditions and invariant conditions of argument types, 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 Blocks*

To solve the problem that the existing BWDM does not support typedef blocks, 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.

Table 1. The expected states 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 types definition is false
	The value is outside the range of the argument type

2.3. *Addition of a Function to Generate Test Cases for Object States*

To solve the problem that the existing BWDM cannot generate test cases for operation definitions that manipulate object states, we add a function to generate test cases for object states.

The VDM++ specification includes invariant conditions for classes and types, and pre-conditions and post-conditions for operation and function definitions. The VDM++ specification includes invariant conditions for classes and types, and pre-conditions and post-conditions set in operation and function definitions. The test cases for object states to be generated in this paper are test cases that use these conditions to generate the expected output of whether the state of the object 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 case generation. Table 1 shows the expected states and the conditions corresponding to the states.

In generating the test case, 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.

3. Application Example

In this chapter, we confirm that the extended BWDM works correctly by using application examples. An example of the VDM++ specification is shown in List 1, and the output of applying it as input to the extended BWDM is shown in List 2.

List 1. Example of VDM++ specification

```
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
```

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

In List 2, we have generated a test case that inputs 10 and 11 for the argument tickets. From this, we can confirm that we can obtain the boundary value from the pre-condition "tickets <= 10" in List 1. The output is "Undefined Action" when the pre-condition is not satisfied, "Failure" when the invariant and post-conditions are not satisfied, and "Normal" when all the conditions are satisfied. We can confirm that each conditional expression is evaluated correctly.

3.2. *Confirmation of Support Type Definition Blocks*

In the "PayWithCardsAndCoupons" operation shown in List 1, the argument type of "amount" is "yen", which is defined in the type definition. In List 2, we have confirmed that we can generate a test case for the boundary value of the nat type, which is the actual type of "yen".

3.3. *Confirmation of Addition of a Function to Generate Test Cases for Object States*

In the "PayWithCardsAndCoupon" operation of List 1, 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 state of the object is "Failure". In the test case No. 19 in List 2, 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.

List 2. Output when List 1 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 Object States
No.1 : 4294967295 4294967295 -> Undefinde Action
(- Omission -)
No.11 : 0 0 -> Normal
No.12 : -1 0 -> Undefinde Action
No.13 : 4294967295 -1 -> Undefinde Action
No.14 : 4294967294 -1 -> Undefinde Action
No.15 : 0 -1 -> Undefinde Action
No.16 : -1 -1 -> Undefinde Action
No.17 : 4294967295 10 -> Undefinde Action
No.18 : 4294967294 10 -> Failure
No.19 : 0 10 -> Failure
No.20 : -1 10 -> Undefinde Action
No.21 : 4294967295 11 -> Undefinde Action
No.22 : 4294967294 11 -> Undefinde Action
No.23 : 0 11 -> Undefinde Action
No.24 : -1 11 -> Undefinde Action
```

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 Blocks*

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.

4.3. *Evaluation on Addition of a Function to Generate Test Cases for Object States*

We have confirmed that the extended BWDM can generate test cases for object states. As a result, the extended BWDM can generate test cases for operation definitions that manipulate the state of objects, which the existing BWDM cannot generate. Therefore, we can say that the usefulness of BWDM has been improved by adding the ability to generate test cases for object manipulation to BWDM.

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

We compared and verified the test case generation time for the object states of the extended BWDM with that of the manual case. The target VDM++ specification is the specification in List 1. The results of the comparative verification are shown in Table 2.

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.

As shown in Table 2, 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.

In the function to generate test cases for object states 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.

Table 2. Comparison of test case generation time by the extended BWDM and manual test case generation time for the specification in List 1

	Time
Average of 5 subjects	17m19s
BWDM	1.4s

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, as a result of comparing and verifying the time required to generate test cases manually, we were able to confirm that the time required to generate test cases using the extended BWDM was reduced by about 17 minutes. In addition, human errors were observed in the manual test case generation, and 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 types other than integer types
- Support for conditional expressions that refer to the value after the operation

6. References

1. Overture Project. Manuals. http://overturetool.org/documentation/manuals.html. Accessed: 2021-12-13.
2. 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.
3. 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.

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.