ATLANTIS PRESS Journal of Robotics, Networking and Artificial Life, Vol. 4, No. 2 (September 2017) 110–113

BWDM: Test Cases Automatic Generation Tool Based on Boundary Value Analysis with VDM++

Tetsuro Katayama[*], Hiroki Tachiyama[*], Yoshihiro Kita[†], Hisaaki Yamaba[*], Kentaro Aburada[*], and Naonobu Okazaki[*]
[*]*University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192 Japan*
[†]*Tokyo University of Technology, 1404-1 Katakura, Hachioji, 192-0914 Japan*
E-mail: kat@cs.miyazaki-u.ac.jp, tachiyama@earth.cs.miyazaki-u.ac.jp, kitayshr@stf.teu.ac.jp, yamaba@cs.miyazaki-u.ac.jp, aburada@cs.miyazaki-u.ac.jp, oka@cs.miyazaki-u.ac.jp

Abstract

For software development using Formal Methods, we have developed a prototype of the boundary value test case automatic generation tool BWDM. The main two topics of our tool are (1) automatically generation of test cases and (2) boundary value analysis. Our tool improves the efficiency of software testing process in using VDM++ that is one of the Formal Methods. In this research, we show the composition of our tool, application example, evaluation of the usefulness, relative research, and future issues.

Keywords: Software Testing, Boundary Value Analysis, Formal Methods, VDM++

1. Introduction

A cause of coming into bugs in software is using natural languages in process of software design. Natural languages are ambiguous. And this causes misunderstanding of developers in designing documents. Finally, they implement software based on the misunderstanding, and the software will have bugs.

As one method to solve this problem, Formal Methods are suggested that used in upstream process of software development. In developing process of using formal methods, specification is described what characteristic of development target, by using not natural languages but formal specification languages what based on mathematical logic. For that reason, unlike conventional development, specification can be proved its correctness of description contents by arithmetical theorem proofs, mechanical inspection, etc. In other words, it is possible to write strict specification document which is excluded ambiguous description.

On the other hand, either way designed using natural languages or formal specification languages, it needs software testing after implementation of software. In order to test software, it is necessary to design test cases, but it takes much time and effort to design them manually. Therefore, by efficiently designing test cases, it is possible to improve the efficiency of the testing process. It is also important to design test cases focused on places where bugs can be hidden, in order to improve the efficiency of executing of software testing.

Therefore, in this research, we develop a prototype of test case automatic generation tool BWDM (Boundary Value/Vienna Development Method) for the purpose of improving the efficiency of the test process in software development using formal methods. We use Boundary Value Analysis[1] for test cases designing. We use the "boundary value test case" what means the test case designed based on boundary value analysis.

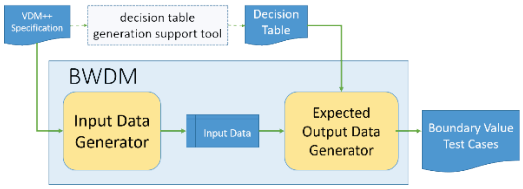

Fig. 1. The flow of BWDM processing

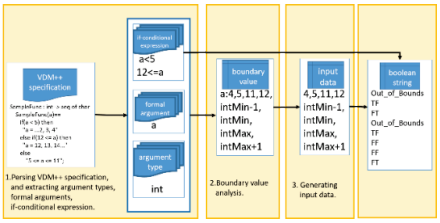

Fig. 2. The flow of input data generator processing

2. Boundary Value Test Case Automatic Generation Tool BWDM

BWDM analyzes boundary values of function definitions that described in VDM++ specification, and automatically generates boundary value test cases. Figure 1 shows the flow of BWDM processing. Note that VDM is a formal method devised at the IBM Vienna Research Institute in the 1970s, and VDM++ is an object-oriented extension of VDM-SL[2] which is formal specification language of VDM.

BWDM consists of input data generator and expected output data generator. In process of expected output data generator, the decision table is used, what is generated by decision table generation support tool developed in our laboratory[3]. The inputs into BWDM are VDM++ specification and decision table. Here, we implement BWDM in Java language.

2.1. *Process of input data generator*

Figure 2 shows the flow of processing of input data generator. The input is VDM++ specification file. And this unit generates input data based on boundary value analysis.

2.2. *Process of expected output generator*

Figure 3 shows the flow of processing of expected output data generator. This unit generates expected output data when input data is input to a function that is

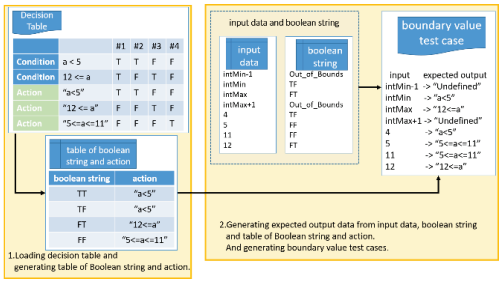

Fig. 3. The flow of expected output data generator processing

written in the VDM++ specification. And it outputs test cases that consists of input data and expected output data.

3. Application Example

We apply the formal specification that mixed inequality expression and surplus expression to BWDM. And we check 3 items.

(i) BWDM is operated correctly.
(ii) The boundary value is correctly extracted from the VDM++ specification.
(iii) BWDM outputs the input data and the expected output data correctly as the boundary value test case.

Figure 4 shows formal specification that we apply to BWDM. This specification judges whether the first argument is even or odd, and whether the second argument is a positive number or a negative number. Figure 5 shows the generated test cases by applying the formal specification of Figure 4 to BWDM.

The boundary values generated from the argument arg1 of the specification in Fig. 4 are natMin-1, natMin, natMax, natMax+1, 1, 2, and 3. And from the argument arg2, intMin-1, intMin, intMax, intMax+1, -1, and 0 are generated. Here, “natMin-1” means a value which is one less than the minimum value of the natural type. BWDM generates all combinations of boundary values of each of the two variables as input data. Therefore, the number of test cases generated from this specification is 7 * 6 = 42 (cases). By Fig. 5, we can check test cases from No. 1 to No. 42. From this, it is clear that BWDM outputs correct test cases from the formal specification. Also, we can check that both of extracting boundary values from VDM++ specification and outputting boundary value test cases that consist both of input data and expected output data are correct as well.

```
class MixSpecification

functions

  mix: nat * int -> seq of char
    mix( arg1 , arg2 ) ==
      if( arg1 mod 2 = 0) then
        if (0 <= arg2 ) then
          " arg1:even arg2:positive"
        else
          " arg1:even arg2:negative"
      else
        if( arg2 < 0) then
          " arg1:odd arg2:negative"
        else
          " arg1:odd arg2:positive";

end MixSpecification
```

Fig. 4. The formal specification that mixed inequality expression and surplus expression

The number of arguments:2

argument type:	argument1:nat	argument2:int		
No.	input data		-->	expected output data
No.1	natMin-1	intMin-1	-->	Undefined Action
No.2	natMin-1	intMin	-->	Undefined Action
No.3	natMin-1	intMax	-->	Undefined Action
No.4	natMin-1	intMax+1	-->	Undefined Action
No.5	natMin-1	0	-->	Undefined Action
No.6	natMin-1	-1	-->	Undefined Action
No.7	natMin	intMin-1	-->	Undefined Action
No.8	natMin	intMin	-->	arg1:even arg2:negative
No.9	natMin	intMax	-->	arg1:even arg2:positive
No.10	natMin	intMax+1	-->	Undefined Action
No.11	natMin	0	-->	arg1:even arg2:positive
No.12	natMin	-1	-->	arg1:even arg2:negative
No.13	natMax	intMin-1	-->	Undefined Action
No.14	natMax	intMin	-->	arg1:odd arg2:negative
No.15	natMax	intMax	-->	arg1:odd arg2:positive
No.16	natMax	intMax+1	-->	Undefined Action
No.17	natMax	0	-->	arg1:odd arg2:positive
No.18	natMax	-1	-->	arg1:odd arg2:negative
No.19	natMax+1	intMin-1	-->	Undefined Action
No.20	natMax+1	intMin	-->	Undefined Action
No.21	natMax+1	intMax	-->	Undefined Action
No.22	natMax+1	intMax+1	-->	Undefined Action
No.23	natMax+1	0	-->	Undefined Action
No.24	natMax+1	-1	-->	Undefined Action
No.25	2	intMin-1	-->	Undefined Action
No.26	2	intMin	-->	arg1:even arg2:negative
No.27	2	intMax	-->	arg1:even arg2:positive
No.28	2	intMax+1	-->	Undefined Action
No.29	2	0	-->	arg1:even arg2:positive
No.30	2	-1	-->	arg1:even arg2:negative
No.31	1	intMin-1	-->	Undefined Action
No.32	1	intMin	-->	arg1:odd arg2:negative
No.33	1	intMax	-->	arg1:odd arg2:positive
No.34	1	intMax+1	-->	Undefined Action
No.35	1	0	-->	arg1:odd arg2:positive
No.36	1	-1	-->	arg1:odd arg2:negative
No.37	3	intMin-1	-->	Undefined Action
No.38	3	intMin	-->	arg1:odd arg2:negative
No.39	3	intMax	-->	arg1:odd arg2:positive
No.40	3	intMax+1	-->	Undefined Action
No.41	3	0	-->	arg1:odd arg2:positive
No.42	3	-1	-->	arg1:odd arg2:negative

Fig. 5. The test cases generated by applying the formal specification of Figure 4 to BWDM

4. Discussion

In this research, for the purpose of improving the efficiency of the test process in software development using formal methods, we have developed a prototype of boundary value test case automatic generation tool BWDM. The inputs to BWDM are VDM++ specification and decision table, and the test case automatic generation and the boundary value analysis to the function definition in the specification are performed. Therefore, this tool improves the efficiency both of designing of test case and conducting of testing. In the followings, we consider BWDM.

4.1. *Evaluation of usefulness*

To consider BWDM usefulness, we used 3 specifications. We measured the time to generate test cases. Table 1 shows conclusion of measuring. Within the three specification, the if-conditional expressions of inequalities are described. The specification 1 has 5 if-conditional expressions, specification 2 has 10 expressions and specification 3 has 15 expressions. To measure execution time, we used System.nanoTime[4] method of Java. We measured the time that is to give the VDM++ specification and the decision table as an input to the BWDM and to finish outputting the test cases. The unit of time was millisecond, and measurements were made five times for each specification, and we calculated the average of 5 measuring as well. Here, we don't consider the time to prepare the VDM++ specification and the decision table.

For the average time in each specification, specification 1 and specification 2 were less than 0.5 seconds, and specification 3 was less than seven seconds. The number of rules on the decision table for specification 1 (number of columns in the decision table) is 32, specification 2 is 1024 and specification 3 is 1048576. These represent the number of states that the function can take, depending on the Boolean value in the if-conditional expression. When constraint conditions such as preconditions are described in the specification, the number of states actually taken by the function is less than this number. However, it is troublesome and time consuming to manually design test cases for functions that can have as many states as possible. On the other hand, BWDM can automatically generate boundary value test cases in several seconds if the specification is up to condition number 15.

Hence, we think that BWDM is usefulness when testing software that implemented from VDM++ specification.

Table 1. Generation time of test cases from three specifications

Times	Specification1	Specification2	Specification3
1st time	325	316	6277
2nd time	283	389	5321
3rd time	500	371	6236
4th time	291	334	5070
5th time	269	371	5700
Average	334	356	5720

4.2. *Related research*

TOBIAS[5] is a test cases automatic generation tool for testing the VDM++ specification. TOBIAS automatically generates test cases according to a test pattern input to TOBIAS. It is defined by test designer using a regular expression to output a test case.

Both TOBIAS and BWDM support the testing phase of software development using VDM++. TOBIAS supports testing the VDM++ specification. In contract, BWMD supports testing actually implemented software.

This is what the difference of between two tools. Therefore, when you need to test implemented software, BWDM is superior.

5. Conclusion

In this research, we have developed a prototype of test cases automatic generation tool BWDM based on boundary value analysis targeting the VDM++ specification with the aim of streamlining the testing process in software development using formal method.

We applied the VDM++ specification that mixed inequality expression and surplus expression into BWDM, and our tool analyzed boundary value of the specification correctly. Finally, BWDM outputs test cases that consist both of input data and expected output data correctly. Furthermore, as a result of applying three kinds of VDM++ specifications including up to 15 if-conditional expressions to BWDM, it generated test cases within about seven seconds. Therefore, BWDM improves the efficiency of designing test cases.

In conclusion, by using BWDM, it is expected to improve the efficiency of the test process in software development using formal methods. Moreover, by conducting test design and test process based on the VDM++ specification, improvement both of productivity and quality of developed software is expected.

Future works are as follows.

- Generation of boundary values not appearing in the specification description

  Although it does not appear as a concrete numerical value in the specification description, there are boundary values that occur when two or more if-conditional expressions are related, and the current BWDM cannot generate those values as input data. It is necessary to modify the current process to generate boundary values.
- Responding to various environments

  Current BWDM, boundary value analysis of argument types of function definition cannot cope with various environments (bit number and development language) used for actual development. It is necessary to implement a function to allow users to set information about test cases what they want.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 24220001.

References

1. Blake Neate, Boundary value analysis, http://www.cs.swan.ac.uk/~csmarkus/CS339/docs.htm, (accessed June 26, 2017).
2. International Organization for Standardization, ISO/IEC JTC 1/SC 22/WG, Information technology--Programming languages, their environments and system software interfaces--Vienna Development Method--Specification Language--Part 1: Base language (1996)
3. 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*, 222-225 (2014)
4. Java System class, https://docs.oracle.com/javase/jp/6/api/java/lang/System.html, (accessed June 26, 2017).
5. Olivier Maury, Yves Ledru, Pierre Bountron and Lydie du Bousquet, Using TOBIAS for the automatic generation of VDM test cases, http://vasco.imag.fr/Tobias/Papers/vdm02tobias.pdf, (accessed June 26, 2017).