A proposal for run-time checks on command execution order of software program

Shinichi Funase (Faculty of Production Systems Engineering and Sciences, Komatsu University, shinichi.funase@komatsu-u.ac.jp)
Toshihiko Shimauchi (Faculty of Intercultural Communication, Komatsu University, toshihiko.shimauchi@komatsu-u.ac.jp)
Haruhiko Kimura (Faculty of Production Systems Engineering and Sciences, Komatsu University, haruhiko.kimura@komatsu-u.ac.jp)

Abstract

In this paper, we proposed a method to automatically perform "inspection of control flow" of a computer program at runtime. "Inspection of control flow" is an inspection of whether the commands of the program under inspection are executed in the intended order. This inspection system targets programs that are written in assembler language and whose command execution order is determined based on input data. Normally, the program to be executed is an object program written in machine language. However, it is difficult to directly handle the machine language expressed in binary numbers. Therefore, the assembler language that has a one-to-one correspondence with the machine language and is expressed by alphanumeric characters and symbols is used for this paper. The proposed system used a learning system to automatically increase the amount of text information. The performance of this system was compared to a conventional inspection method from the input/output relationship. The proposed system detected 80% of the errors detected by the input / output relationship method.

Key words

software testing, program inspection, control flow, logic errors, debugging

1. Introduction

Software testing is the process of finding behaviors or defects (bugs) that are not specified in a computer program. Behaviors and defects that are not in the specifications are usually called logic errors. Debugging is the process of fixing defects in a program found in software tests. Successful software testing means that the test finds defects, passes all specified test items, or reaches specified quality goals. For some software development, the target quality requires to pass all the specified test items. For example, the OS and programming languages stipulate conformance tests to check if they meet the specifications. Software testing can show the existence of defects but cannot prove the non-existence of defects. The work of guaranteeing that software does not have unspecified behavior is called proof, and there are systems designed for proof and languages fit for proof, but the actual proof was conducted by using mathematical methods such as Euclidean algorithm. (Kimura, 1976; Myers, 1979; NIST 1990; IT Media, 2011; ISTQB, 2018, JSTQB, 2018, Wikipedia, 2020, TOPPERS, n.d.)

The software test proposed in this paper falls into the category of dynamic software testing where the program inspection is automatically performed simultaneously with the execution of the program. The bugs targeted in the software test of this paper are bugs related to control flow (Kimura, 1976).

2. Overview of the proposed inspection method

The "inspection of control flow" conducted in this paper is a process to inspect whether or not the program under test has executed the intended series of instructions. The program to be inspected P (hereafter target-P) is written in assembler language (used in FACOM 230-15), and the sequence of instruction execution of target-P is determined by the input data.

The outline of this inspection is as follows. First, target-P is passed through the interpreter to be decrypted and executed by the interpreter. Target-P will not be translated into machine language, but each instruction is deciphered by the interpreter, and the interpreter executes each instruction. Specifically, the instruction to be executed for target-P is called into the interpreter, the type of instruction is deciphered, and the function of that instruction is executed by the interpreter. Then, the next instruction is called, and its decoding and execution is repeated in the same manner. In this process, a series of labels (address) which target-P should pass is retrieved to check whether the sequence follows the specified path.

3. Program for inspection test and preparation of proposed inspection

First, target-P is created by following the general flowchart. A general flow chart is a schematic design and clearly describes the outline and flow of each work. In the general flowchart used in this paper, labels are added at the branch points and at the beginning of each work. The flowchart also constitutes a tree. These labels in the general flowchart are used for target-P. A detailed flowchart is a flowchart of target-P, and each block corresponds to each instruction of the program. Figure 1 shows an example of creating general and detailed flowcharts.

The general flowchart in Figure 1 shows a simple process

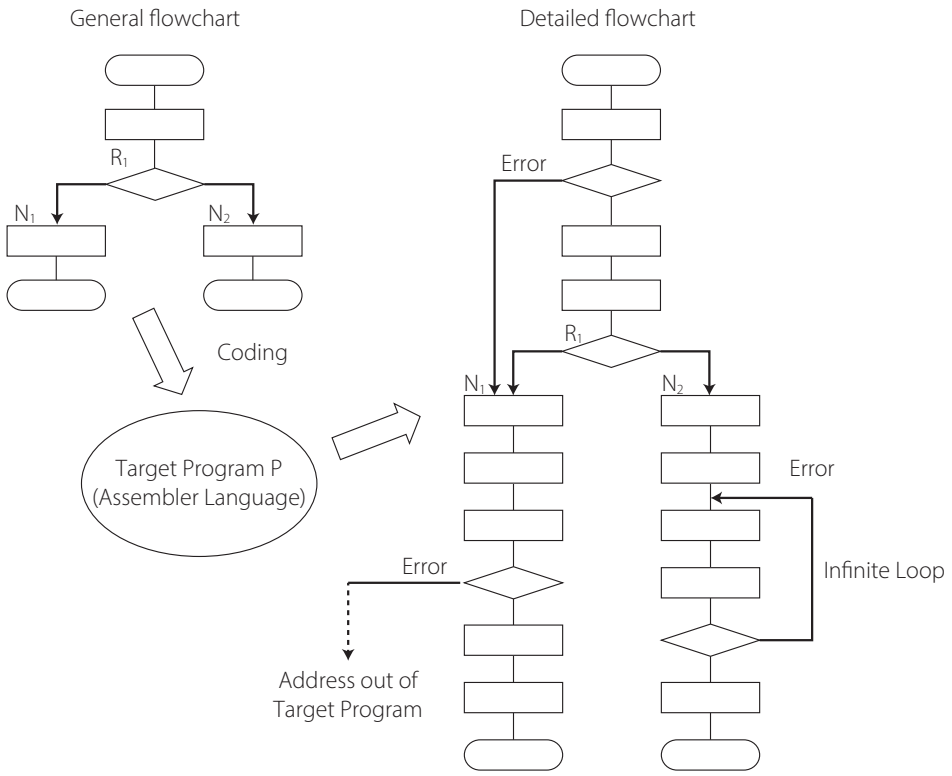

Figure 1: General and detailed flowcharts

that is divided into a process with label N_1 and that with label N2 at the branch point R_1. Target-P is coded based on this general flowchart. The detail flowchart is an easy-to-understand version of target-P. In this example, there are three errors. The first does not pass through the branch point R_1. The second is to go outside of the program. The third is an infinite loop with iterative processing.

A preparation for the proposed inspection requires a creation of a correspondence table for retrieving from input data a series of labels (addresses) that target-P must pass. Figure 2 shows an example of a control flow of input data based on a correspondence table. This series of labels is a marker to show the main flow of control described in the general flowchart.

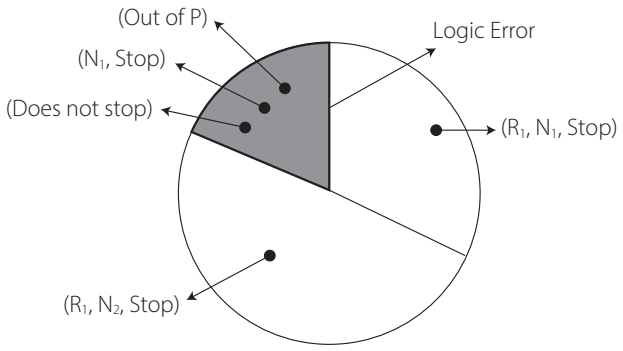

Figure 2: Control flow of input data

Note: The shaded area corresponds with the three errors shown in Figure 1.

4. Types of logic errors covered in this study

Following four logic errors are targeted in this paper:

(1) The control flow goes out of target-P.
(2) The control flow does not pass through the intended series of branch points and labels.
(3) In the learned control flow, the control follows different branches even though the input data is the same.
(4) The control flow doesn't stop.

The logic error in (3) occurs when target-P is partially destroyed in advance.

5. Flow of proposed inspection methods

(1) Target-P is input to the interpreter.
(2) The interpreter decodes the first instruction of target-P, and the processing of the same function as the instruction is executed in the interpreter. However, if it is an input instruction, it reads the input data and also retrieves a series of labels that target-P must pass through.
(3) The next instruction of target-P is retrieved, decrypted, and the processing of the same function as that instruction is executed in the interpreter, but while confirming that P passes through a series of labels that must be passed. This operation is repeated until a stop command

is issued, an address outside P is reached, or an infinite loop is reached. However, for the infinite loop, the elapsed time for threshold shall be set in advance.

(4) In addition, a learning function is added to this system. This function memorizes a pair of input data and a sequence (branch point and/or jump destination address) that P passed through and checks whether the same input data to target-P leads to the same sequence (branch point and/or jump destination address) stored as a pair in the memory. This learning function confirms whether logic error (3) is occurred or not. However, the data used is limited to the initial execution period to save memory.

6. Experiment

A part of the communication software (1,500 words routine) was used for target-P in the experiment with two methods. The first method employed a conventional method to check the existence of a logic error based on the relationship between the input data and the resulting output data. The second method is the proposed method described in 5. After running 40,000 steps, the first method detected five logic errors while the second method detected four. The logic error not detected in the second method was an error not affecting adversely on control flow. The proposed inspection method can be judged to have necessary error detecting capability.

The feature of this system allows the inspection to be carried out simultaneously with the execution of the program, which shortens the inspection time. Therefore, when comparing the inspection times of this system and checking input/output relationship under human intervention, this system is significantly faster. However, running target-P with this automatic error check system requires significantly longer time compared to running the program without the system. Another advantage of this system compared to conventional input-output relationship checking is that it allows to locate approximate steps where logic errors occur.

7. Results and Discussion

The following was found from the statistical data collected during the inspection period. Commands listed in Tables 1 through 3 are those of assembler language used in FACOM230-15.

Table 1 shows the probability of commands to be executed in target-P organized by category. Table 2 shows the probability of commands in detail. The branch commands are executed in 0.29 probability, meaning the collation is performed every three steps on average.

Table 1: Probability of command types

Category	Probability
Address reference command	0.81
N designation command	0.15
Special command	0.03
Others	0.01

Table 2: Probability of command

Command	Probability
J	0.20
L	0.20
T	0.16
SOB	0.08
MIS	0.07
MDS	0.07
LB	0.07
Others	0.15

The total number of corrections for each instruction by the proposed system is shown in Table 3.

Table 3: The number of corrections for each command

Command	Correction
L	12
LB	8
J	5
T	5
MA	4
SZ	2
MB	1
EOR	1
MDS	1

8. Conclusion

In this paper, an automatic inspection method focusing on the control flow was proposed. The proposed system used a learning system to automatically increase the amount of text information. In addition, the performance of this system was compared to a conventional inspection method from the input/output relationship by using a part of the communication software as target-P.

The result of the study showed that this system detected 80 % of the errors detected by the input/output relationship method. The system allows inspection to be conducted simultaneously with the execution of target-P, shortening the time required for inspection and improving the accuracy of the test. Furthermore, the system shows approximate location where the logic error occurs. From the above, the effectiveness of this system was established.

The logic errors that can be inspected by this inspection system are as follows.

- It is possible to inspect whether the label or address series intended by target-P is passed or not while letting target-P perform the actual work.
- The system can detect a case of control flow change when some input data are selected and executed from the input data area which should select the same control flow.

Although the system has these advantages, it has several limitations. One of them is that this system is intended for testing a program which determines the correct control flow path from input data to text

References

International Software Testing Qualifications Board (2018). Certified tester foundation level syllabus 2018v3-1. https://castb.org/wp-content/uploads/2020/01/ISTQB-CTFL_Syllabus_2018_V3.1.pdf (Accessed on December 10, 2020).

IT Media (2011). "coverage criteria.", Information system glossary. http://www.itmedia.co.jp/im/articles/1111/07/news142.html (Accessed on December 10, 2020). (in Japanese)

Japan Software Testing Qualifications Board (2018). Foundation level syllabus, version 2018.J03. http://jstqb.jp/syllabus.html#syllabus_foundation (Accessed on December 10, 2020).

Kimura, H. (1976). A study of software reliability. Master thesis of Tohoku University. (in Japanese)

Myers, G. J. (1979). *The art of software testing*. New York: John Wiley and Sons.

National Institute of Standards and Technology (1990). PCTS:151-2, POSIX Test Suite (POSIX 1990 version). http://www.itl.nist.gov/div897/ctg/posix_form.htm (Accessed on December 10, 2020).

TOPPERS (n.d.) TTSP. https://www.toppers.jp/ttsp.html (Accessed on December 10, 2020). (in Japanese)

Wikipedia (2020). Software testing (Accessed on December 10, 2020).

(Received: December 16, 2020; Accepted: December 25, 2020)