A theoretical analysis of automatic inspection of the control flow of computer programs

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
The authors have proposed an automated inspection on the control flow of computer programs, but its theoretical analysis has not been reported by other researchers yet. The automatic inspection on the control flow of a computer program is to check whether instructions are executed in intended order while executing the target program. The authors developed this automatic inspection by tracing the inconsistency between the execution of the target program and the general flowchart while using an interpreter to trace the flow. In this paper, after theoretically confirming each process of the automatic inspection algorithm, the following three points are discussed regarding the equivalence of the flow chart control flow and the program control flow. (1) The number of control flow paths in the flowchart is shown to be less than or equal to the number of program control flow paths when the program is coded correctly. (2) The necessary and sufficient condition for the input set selecting the control flow path of target-P which does not satisfy any control flow in GF to become empty set is that the macro statement becomes allocation statement for any input value, provided that the branches described in the flowchart are correctly coded. 3) we discuss the equivalence of the flow chart control flow and the program control flow.

Key words
software testing, program inspection, control flow, general flow chart, detailed flow chart

1. Introduction

The "inspection of control flow" (Kimura, 1976) in this study is an inspection whether instruction is executed in intended order while executing the target program (hereafter target-P). Target-P for this inspection system is written in assembler language and its instruction execution order is determined by input data. Normally, target-P is an object program written in machine language expressed in binary codes. Since directly handling machine language is difficult, the assembler language, which has a one-to-one correspondence with machine language and is expressed by alphanumeric characters and symbols, was used for target-P in this study.

The authors proposed an automatic inspection of the control flow (Funase et al., 2020), but the theoretical analysis has not yet been done. The proposed automatic inspection is executed while decoding each instruction of target-P by an interpreter, and the control flow is inspected while checking absolute address and label of each branch instruction. The control flows of target-P and its general flowchart are determined by input data. In addition, the control flow of the general flowchart is treated as text, and confirmation whether there is any contradiction in the control flow of target-P is conducted while it is being executed. A contradiction detected shows an existence of a logical error.

This study conducts a theoretical analysis on the automatic inspection to prove the equivalence of the control flow of the general flowchart and that of target-P (Myers, 1979; National Institute of Standards and Technology, 1990; IT media, 2011; TTSP, 2014; International Software Testing Qualifications Board, 2018; Japan Software Testing Qualifications Board, 2018; Software testing, 2020; TOPPERS, n.d.).

2. Overview of the automated inspection

The automatic inspection proposed in Funase et al. (2020) uses an interpreter to execute target-P and at the same time trace it to check whether it passes through the intended branch point. The outline is shown in Figure 1. A general flow chart (GF), written in documentation or mathematical formulas, is labeled with intended branch points and destinations and provided as a text-based flow chart. A graph containing only nodes and arcs of GF's branch points is called a general flowchart graph (GFG). In GFG, GF's labels are attached to its branch point nodes, and destination labels are attached to the end point (node) of the arc, which is a directed line connecting the nodes of the start point and the end point. In this study, GFG is referred to as the "test text (correct answer information)".

Next, consider target-P coded with GF and a detailed flowchart (DF) corresponding with target-P. DF is a flowchart in which each instruction is assigned to one block and the instruction is written in the block as it is. In the example shown in Figure 1, three types of errors are assumed to occur during coding. A detailed flow chart graph (DFG) is a graph similar to GFG and describes these logical errors. The same label is used in the corresponding place for target-P, DF, GFG, DFG and GF. The automatic inspection proposed in Funase et al. (2020) checks whether the DFG's control flow satisfies the GFG's control flow, both of which determined by input data.

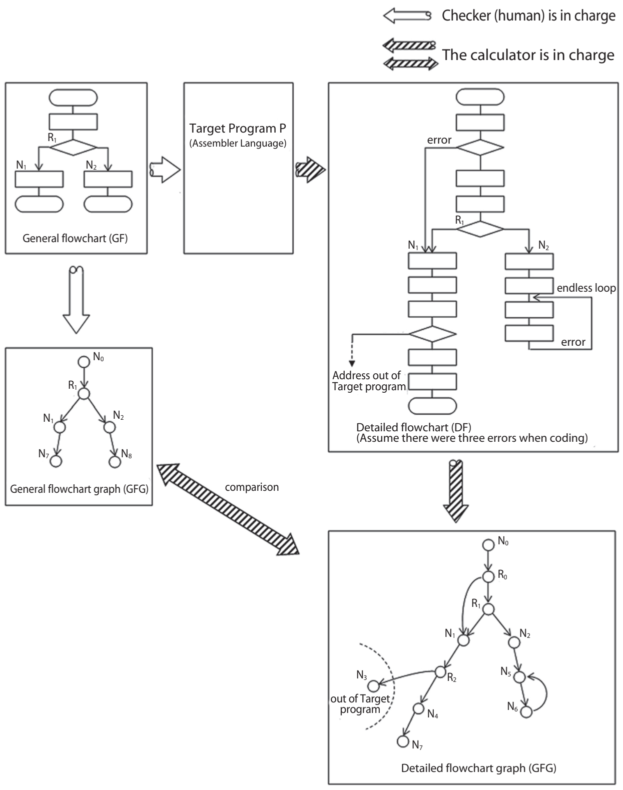
Checker (human) is in charge
The calculator is in charge
General flowchart (GF)
Target Program P (Assembler Language)
Detailed flowchart (DF) (Assume there were three errors when coding)
error
endless loop
Address out of Target program
General flowchart graph (GFG)
comparison
out of Target program
Detailed flowchart graph (GFG)

Figure 1: Overview of the inspection

"Satisfy" means the both flows do not contradict with each other. In the example of Figure 1, the first error is detected by reaching N_1 without passing through R_1. The second error is detected by coming out of the absolute address in target-P. The third error is determined to have fallen into an infinite loop by repeating the same series of paths multiple times for a predetermined threshold time.

3. Theoretical analysis of the automated inspection algorithm

3.1 Input data and statement

The control flow of GF is determined by input data, but the control flow of the corresponding target-P is not limited to one. Although description in GF can be not detailed, it must be expressed strictly in target-P. Also, there are many cases where there are multiple control flows in target-P.

In this study, it is assumed that target-P has m input variables $\chi_1, \chi_2, ..., \chi_m$. The set of data which χ_i can take is described as D (χ_i). ID is a set of possible values of input data (n-term set) and it is described as

$$ID = D(\chi_1) \times D(\chi_2) \times \cdots \times D(\chi_m)$$

where × represents a direct product.

A subset of ID is PIDt (t = 1, 2, · · ·, s) which selects the t-th control flow of target-P. However, for any two subsets,

$$PIDi \cap PID_j = \varphi$$

and

$$ID = PID_1 \cup PID_2 \cup \cdots \cup PID_s$$

GID_j is input data to select the j-th flow of GF and it is described as

$$GID_j = PID_{j1} \cup PID_{j2} \cup \cdots \cup PID_{je}$$

These ID subsets may contain control flows that cause logical errors.

In this study, the assembler language is composed of the following instructions:

- Start statement
- Assignment statement
- Test statement
- Macro statement
- Stop statement

Test statements and macro statements function as branch instructions. Although macro statements can be test statements or assignment statements, they are usually used as the latter. The former occurs when target-P crashed and unintended data is written in the areas of macro statement. The data written in this way may become test statement. The other instructions are statements described in Manna (1974).

3.2 Control flow

Normally, the control flow of a program indicates the execution order of instructions by a series of addresses, and is a totally ordered set of addresses representing the execution order of instructions from the start point to the endpoint. This study focuses on the branch point (branch instruction). The control flow is described as a totally ordered set of the pairs between labels attached to the branch point (branch instruction) and those attached to the instruction to be executed after the branch point (branch instruction). Some labels are described as absolute addresses to shorten the description of control flow.

[Definition 1] Path Cj of j-th control flow in GFG is represented by a totally ordered set of labels from start to end points as follows:

$$C_j = ((R_{j1}, N_{j1}), (R_{j2}, N_{j2}), \cdots, (R_{jej}, N_{jej}))$$
$$\text{for } j = 1,2, \cdots, d_1$$

where

R_{ji} represents a label attached to a branch point.

N_{ji} represents a label attached to the beginning of the process to be executed after R_{ji} labeled branch point.

e_j represents the number of branch points in path C_j.

[Definition 2] Path H_t of the t^{-th} control flow in target-P is represented by a totally ordered set of absolute addresses from start to the end points as follows:

$H_t = ((R^*_{t1}, N^*_{t1}), (R^*_{t2}, N^*_{t2}), \cdots, (R^*_{tft}, N^*_{tft}))$
for $t = 1,2, \cdots, d_2$

where

R^*_{ti} represents an absolute address where the branch instruction is stored.

N^*_{ti} represents an absolute address which stores the instruction to be executed after the instruction in the absolute address R^*_{ti}.

f_t represents the number of branch instructions in path H_t.

[Definition 3] The absolute address storing the instruction is obtained from the label attached to the instruction of target-P to create a correspondence table between the label and the absolute address. By using this correspondence table, the label of C_j is changed to the absolute address to create path C^*_j, which is represented by a totally ordered set of absolute addresses as follows:

$C^*_j = ((R^+_{j1}, N^+_{j1}), (R^+_{j2}, N^+_{j2}), \cdots, (R^+_{jej}, N^+_{jej}))$
for $j = 1,2, \cdots, d_1$

where

R^+_{ji} represents an absolute address where the instruction with the corresponding R_{ji} of target-P is stored.

N^+_{ji} represents an absolute address where the instruction with the corresponding Nji of target-P is stored.

[Definition 4] Control symbol * is inserted at the start and end of C^*_j, and between (R^+_{ji}, N^+_{ji}) and (R^+_{ji+1}, N^+_{ji+1}) for each i. The control flow path C'_j with endmark@ at the end is expressed as follows:

$C'_j = (*, (R^+_{j1}, N^+_{j1}), *, (R^+_{j2}, N^+_{j2}), *, \cdots, *, (R^+_{jej}, N^+_{jej}), *, @)$
for $j = 1,2, \cdots, d_1$

3.3 Automated inspection algorithm

The algorithm of the automatic inspection proposed by Funase et al. (2020) is outlined in an error detection process shown in Figure 2.

<1> First, a GF is given, and target-P is coded according to the GF.

<2> Next, a GFG is created from the GF.

<3> C_j is selected from input data.

<4> C^*_j is created by changing the labels of C_j to absolute addresses according to target-P.

<5> The operations of <2>, <3>, and <4> are automated so that C^*_j is determined by input data.

<6> Text C'_j is created to check whether target-P follows the intended node path.

<7> Target-P is controlled by DF.

<8> DFG is created from DF.

<9> The flow of DFG is controlled from input data.

<10> Upon execution, H_t is selected according to the value of input data.

<11> A check is conducted to see whether H_t satisfies C'_j.

<12> Check steps in <11> is a learning system, where text information increases proportional to the number of checks. This allows detailed checks on the same control flow when the same PID_t input data is taken. C''_j is the text for a complete check to see if it passes through the same control flow.

<13> If the same data in PID_t is input to subsets PID_1, PID_2, $\cdots$, PID_s, complete confirmation can be conducted to see if it passes same control flow.

4. Equivalence of flow chart control flow and program control flow

4.1 The number of the control flow path

Proposition (A): The number d_1 of the control flow path set $\{C_j\}$ of GF is equal to or less than the number d_2 of the control flow path set $\{H_t\}$ of target-P when it is correctly coded.

The reason is as follows. Since target-P codes GF correctly, the set $\{H_t\}$ satisfies the control flow path set $\{C^*_j\}$, in which labels of the set $\{C_j\}$ are changed to absolute addresses. Here, "satisfy" means that for any C^*_j, there exists an H_t which contains the same totally ordered set as the path. In other words, for each C^*_j, there is always an H_t with the same absolute address sequence on the path. In short, there is no contradiction. From this, if there is no macro statement in target-P, the branch point is only the test statement, and

$\{C^*_j \mid j = 1,2, \cdots, d_1\} \equiv \{H_t \mid t = 1,2, \cdots, d_2\}$

Therefore,
$d_1 = d_2$

If target-P contains macro statements, these may become branch points, so the number of control flow path sets $\{H_t\}$ increases. Also, if target-P contains a branch point not listed in GF, the number of Ht increases. From these considerations, proposition (A) holds true.

4.2 Condition for input set to become empty set

Proposition (B): The necessary and sufficient condition for the input set selecting the control flow path of target-P which does not satisfy any control flow in GF to become empty set is that the macro statement becomes allocation

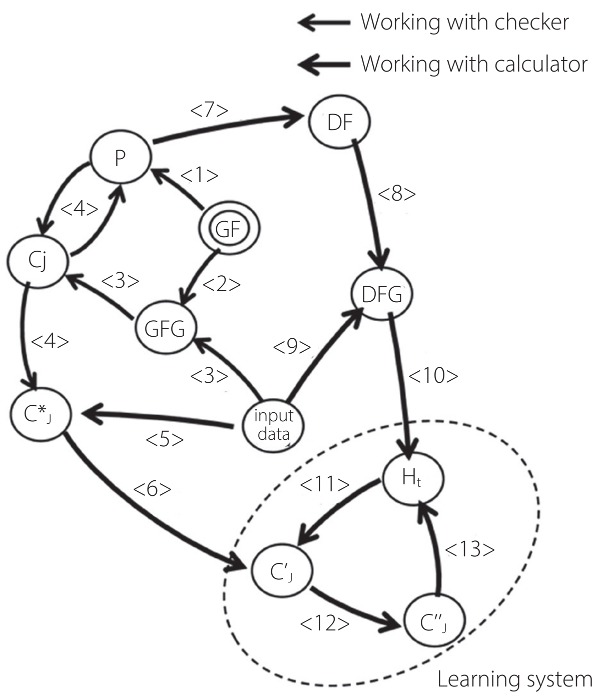

Figure 2: Error detection process

statement for any input value. However, it is assumed that the branches described in GF are coded correctly.

The reason is as follows. When there is no control flow of target-P that does not satisfy the control flow of the general flowchart, that is, when coding is performed correctly, a macro statement becomes an allocation statement regardless of the value of the input data. This is because a test statement is unintended by the programmer and is a source of error. On the contrary, when a macro statement becomes an allocation statement regardless of the value of the input data, since the branch described in GF is assumed to be coded correctly, there exists a control flow of target which satisfies the control flow of GF.

Therefore, proposition (B) holds true.

4.3 The equivalence of the control flows

Finally, the equivalence of the control flow of GF and that of target-P is considered. In this case, the texts to be compared are GF and {GID_j}. Target-P and DF have a one-to-one correspondence and are equivalent, but GF and target-P are not. If coded correctly, target-P can be generated from GF (GF→target-P). But opposite is not true and GF cannot be uniquely created from target-P because GF depends on how it is expressed and its degree of outline.

For each data, if DFG, which is generated from target-P, satisfies GFG, which is generated from GF, that is, when there is no contradiction, DFG and GFG are judged be equivalent in terms of control flow in this study. In this automatic inspection, this equivalence is inspected, and an error is detected by finding a contradiction. There are three types of errors that can be detected as shown in Figure 1. Errors are detected by tracing the control flow at runtime to check whether the above three types of errors have occurred. However, to show that there are no errors related to the control flow, it is necessary to try all the data, which is practically impossible.

5. Conclusion

This automatic inspection system proposed in Funase et.al. (2020) is developed by inputting GF and {GID_j} as text and by checking whether the DFG which is generated from target-P satisfies the GFG which is generated from GF. Since this automatic inspection system generally cannot test all the data of each GID_j, it cannot be shown that there is no error in the control flow. However, an error in the control flow for the input data being executed can be detected. This system is very effective because users are more interested in the results of the process currently being executed than in the future result. As such, this system is best fitted to detect errors related to control flow and Funase et al. (2020) reports these errors accounted for approximately 80 % of all errors.

References

Funase, S., Shimauchi, T., and Kimura, H. (2020). A proposal for run-time checks on command execution order of software program. *Studies in Science and Technology*, Vol. 9, No. 2, 149-152.

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)

Manna, Z. (1974). *Mathematical theory of computation*. McGraw-Hill.

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

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.

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

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

(Received: January 29, 2021; Accepted: March 11, 2021)