Journal of Robotics, Networking and Artificial Life, Vol. 1, No. 2 (September 2014), 111-115

Proposal of a Visualizing Method of Data Transitions to Support Debugging for Java Programs

Tetsuro Katayama[*†], Hiroto Nakamura[†], Yoshihiro Kita[‡], Hisaaki Yamaba[†] and Naonobu Okazaki[†]

[†]*Faculty of Engineering, University of Miyazaki, Miyazaki 889-2192, Japan*

[‡]*Security Center, Kanagawa Institute of Technology, Kanagawa 243-0292, Japan*

[*]Corresponding author, E-mail: kat@cs.miyazaki-u.ac.jp
Tel: +81-985-58-7586, Fax: +81-985-58-7586

Abstract

It takes much time to find the cause of a bug in debugging of programs. Finding the cause of a bug needs to comprehend a flow and data transitions in executing programs. It is difficult to grasp behavior in executing the programs whose behavior is unexpected by a bug. We propose a visualizing method of data transitions to support debugging for Java programs in order to improve efficiency of debugging by supporting to find the cause of a bug. We have implemented TVIS in order to confirm efficiency of the proposed method. The data transitions diagram is the most characteristic function of TVIS which shows the data transitions in executing programs as a table. It can show visually abnormal behavior: no data renewed at all, data abnormally renewed, and so on. Because abnormal behavior is detected in the data transitions diagram at first glance, it is useful for programmers in finding the cause of a bug. This paper shows that the method can support to find the cause of a bug.

Keywords: programming, program slicing, visualization, debug, dynamic analysis, syntax analysis

1. Introduction

It takes much time to find the cause of a bug in debugging of programs [1]. Finding the cause of a bug needs to comprehend a flow and data transitions in executing programs. It is difficult to grasp behavior in executing the programs whose behavior is unexpected by a bug. Additionally, many programs are complicated for including a number of loops and branches. In particular, unskilled programmers need more time for this work.

Thin slicing [2] is a technique to find the cause of a bug, but sometimes its information is insufficient. Thin slicing is a kind of program slicing [3], and can analyze data transitions without superfluous information by restricting abstraction of slice to data transitions only. However, enough information to find the cause of a bug cannot be gotten because states and renewals timing of data which is not selected as slice are not shown.

This paper proposes a visualizing method of data transitions to support debugging for Java programs in order to improve efficiency of debugging by supporting to find the cause of a bug. The method supports to grasp behavior in executing programs because visualizes renewals and states of each data.

This paper has implemented TVIS which is tool to visualize data transitions for Java programs to confirm efficiency of the proposed method. Main functions of TVIS are the data transitions diagram, the renewal history table, and the slicing function. In particular, the data transitions diagram is the most characteristic function which shows the data transitions in executing programs as a table. Therefore, the method shows abnormal behavior and data renewals for understanding behavior of executing programs.

Published by Atlantis Press

This paper visualizes the programs which include a bug in order to confirm that TVIS can support to find the cause of a bug.

2. Data Transitions

The data transitions in this paper show the flow of variable renewals in executing a program. It expresses when and what value of each variable is renewed. Programmers can grasp behavior of a program because they can prefigure the behavior of each variable at arbitrary timing in the program execution by understanding the data transitions. Hence, finding the cause of a bug becomes easier by comparing the difference between the behavior which programmers expect and the actual behavior, if they grasp the data transitions of the program which includes a bug.

However, it is difficult to grasp data transitions of the program which includes a bug. The reason is that the program which includes a bug doesn't behave as programmer's expectation, in addition they doesn't know the cause of a bug.

3. Visualization

In order to confirm efficiency of the proposed method, TVIS (transitions visualization), which is tool to support debugging, has been implemented. It can visualize data transitions. Fig.1 shows the process of visualizing data transitions. The contents of the process is as follows.

I. TVIS statically analyses structure information of the program in order to choose the point which probes to do dynamic analysis are inserted in.
TVIS does syntax analysis of the source code which is visualized and analyses the result of it for investigating structure information of the program. Structure information of a program is information which includes the following data: locations of declaring variables, locations of renewals, ranges of each variable scope, locations of loops, and so on.

II. TVIS generates the source code which includes probes, and then outputs the result of dynamic analysis of the program by executing it.

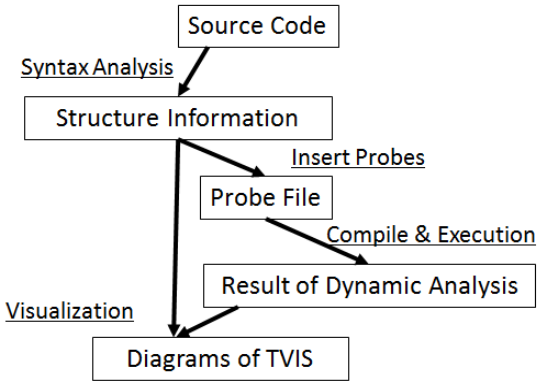

Fig. 1:The process of visualizing data transitions.

TVIS generates the probe file which is inserted probes in the source code to get information of renewals of variables and loops from structure information. TVIS outputs the result of dynamic analysis by executing the probe file after compiling it.

III. TVIS visualizes data transitions by using the result of dynamic analysis and the structure information of a program.
The result of dynamic analysis has information in each process in executing the program: declaration and renewals of variables, and loop. TVIS analyses data transitions which occur in each process by using the result of dynamic analysis, and shows them on the window.

Main functions of TVIS are the data transitions diagram, the renewal history table, and the slicing function. Fig.2 shows an example of the window of TVIS. The data transitions diagram is on the upper right of the window, and the renewal history table is on the green area, and slices list as the result of the slicing function is on the blue area. TVIS can support to grasp behavior of executing programs because it shows data transitions by using these functions.

The data transitions diagram is the most characteristic function which shows renewals of each variable and its timing. It is a table and indicates the number of iterations of each loop in a lateral direction and indicates the line number of data renewals in a vertical direction, and shows renewal value of variables in each loop. It is possible to understand the line and the loop where each renewal exists by using the data transitions diagram. For example, the area which is surrounded by the red frame in Fig.2 shows that *Loop1* is repeated four

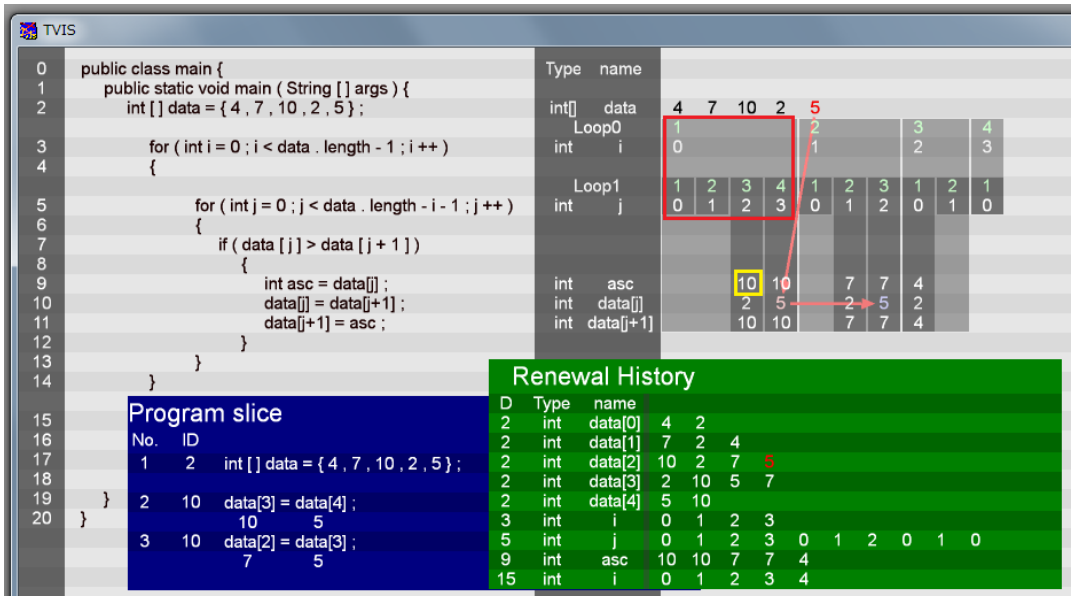

Fig. 2: An example of the window of TVIS.

times in the first loop of *Loop0*. The area which is surrounded by the yellow frame in Fig.2 shows that the value of variable *asc* is assigned into 10 in the third loop of *Loop1* in the first loop of *Loop0*. Hence, the data transitions diagram can show the flow of renewals of each data. Moreover, it is possible to grasp renewals and flow of the whole by using the table form, it is easy to understand states of the other data when a suspicious state occurs.

In addition, TVIS shows the data transitions arrow which shows relations between each renewal of variables as the red arrow on the data transitions diagram. The relations mean ones between state of a variable and the state of other variables which is used for the renewal of its state. After a programmer selects a state by clicking a value on the data transitions diagram, TVIS draws a red arrow from the state which influenced its renewal to the state which was selected. When a suspicious state occurs, finding the cause of it becomes easy by using the data transitions arrow.

The renewal history table shows how many renewals of each variable in executing programs happen and what its value is after renewals. It indicates the number of renewals in a lateral direction and indicates a variable name in a vertical direction. It can show information about whether abnormalities of values of variables after renewals and the number of the renewals happen.

In addition, TVIS can perform thin slicing to arbitrary states of each variable as an ancillary function. When the value on the data transitions diagram or the renewal history table is clicked, TVIS performs thin slicing, and shows the result of slicing on the blue area of the window. TVIS can perform thin slicing at one click, and it becomes easy to perform the slicing to not only the final state but also an arbitrary state of each variable by using the data transitions diagram or the renewal history table. Therefore, analysis of data transitions can be performed without superfluous slices.

4. Example

Two programs which include a bug are visualized by adapting them to TVIS, to confirm efficiency of the proposed method. They are bubble sort programs in Java and include different bugs.

The bubble sort program in Fig.3 sets a wrong condition of a loop. The correct loop condition in the 5th line in Fig.3 is not "*j* < *data.length-j-1*" but "*j* < *data.length-i-1*". This mistake that programmers confuse variables of similar names happens often, but they hardly notice existence of it. The program has returned the array which is identical with the original array because of this mistake.

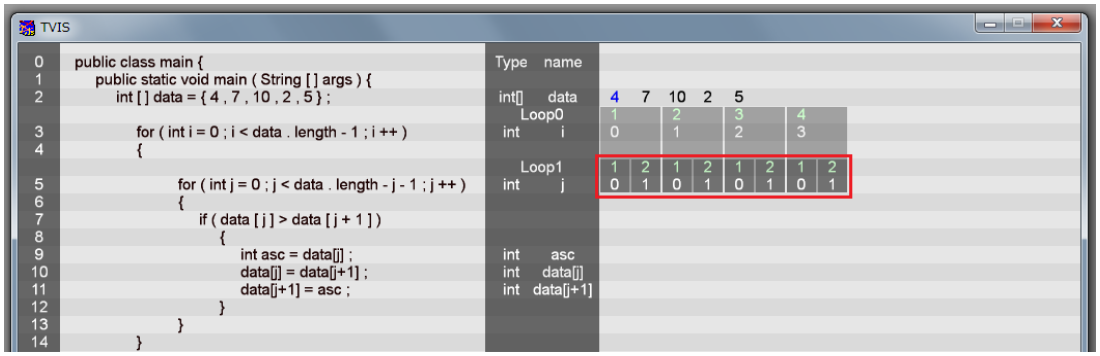

Fig. 3:An example A: Visualizing the program which includes a bug.

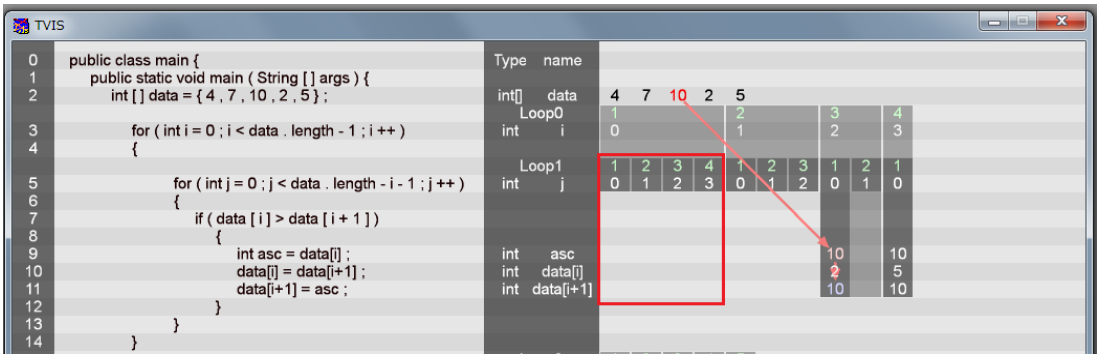

Fig. 4:An example B: Visualizing the program which includes another bug.

The data transitions diagram on Fig.3 shows abnormal behavior of exchanging elements of the array from 9th line to 11th line and *Loop1* which starts at the 5th line. The area which is surrounded by the red frame shows that all iterations in *Loop1* repeated only two times and variable *j as the loop counter* didn't become more than two. The correct behavior of *Loop1* dwindles the number of iterations of the loop as shown Fig.2. Hence, we understand that the loop condition of *Loop1* is suspicious.

The bubble sort program in Fig.4 contains variables assigned to a wrong value. The variable which is used from 7th line to 11th line on Fig.4 is not variable *i* but variable *j*. This mistake occurred by merely confusing variables, but it is hardly noticed because multiple lines are wrong in succession. The program finishes process without executing until completion of sorting because of this mistake.

The data transitions diagram on Fig.4 shows abnormal behavior to exchange elements of the array. The area which is surrounded by the red frame shows that Loop1 repeated four times in the first loop of *Loop0* which starts at the 3rd line. Here, *if* statement at the 7th line, was expected to compare all elements of the array, but it never exchanged them. Hence, we understand that *if* statement at the 7th line is suspicious.

From the above two results of visualization, we have confirmed that programmers can get useful information to find the cause of a bug and grasp behavior of the program included the bug by using visualization of data transitions of TVIS.

5. Evaluation

This paper has implemented TVIS which supports debugging for visualizing data transitions, in order to confirm efficiency of the proposed method by visualizing the program which includes bugs. It explains the difference by comparing the method with conventional methods below.

Dynamic slicing [4] which includes thin slicing is a technique to analyze behavior in executing a program by extracting processes which relate to generation of a selected state as slice. We cannot get useful information if not we appositely decide a condition of slicing, because slicing cannot show information which is not selected as slice. In case of Fig.3, it is difficult to get

useful information, because slice except initialization processes is not gotten even if elements of the array are selected as a condition of slicing. However, TVIS can show useful information for programmers to grasp behavior of a program, because it shows the situation of each variable and loop by the diagram.

Breakpoint is one of the most frequently used methods for debugging [5]. Breakpoint has the problem which programmers need experience of programming to decide a location of breakpoint. Nevertheless, it is a useful method of finding the cause of a bug, and also the method which automatically generates breakpoint has reported [6]. When programmers found an abnormal process, it is difficult to understand correctly the situation of execution with only information of a point. TVIS becomes easy to grasp behavior of a program, when programmers found the abnormal process, because TVIS shows them to information of variables or loops which concern it. In a large program, efficiency of TVIS may lower in comparison with breakpoint, because information of the diagram is liable to increase.

An applicable range of programs which TVIS can visualize is small in comparison with the tool [7] which can visualize a large program by a diagram with a high abstraction level or tool [8] which can visualize multithreaded programs. TVIS, which generates a diagram of low abstraction level, may make gigantic a diagram because data to visualize even a program of moderate size may become huge [9]. Therefore, the variety of programs which TVIS can visualize may have a limit. Accordingly, TVIS need reduce spaces to show information by improving the expression format of the diagram. An applicable range of programs which TVIS can visualize is enlarged by focusing object to visualize on only an important object.

6. Conclusion

This paper has proposed the visualizing method of data transitions to support debugging for Java programs in order to improve efficiency of debugging by supporting to find the cause of a bug. We have implemented TVIS which supports debugging and visualizes data transitions, in order to show efficiency of the proposed method. We visualize programs which include a bug by using TVIS, and show that TVIS can support to grasp behavior of executing programs and to find the cause of a bug.

Visualization of the proposed method can give the information which is not gotten by using conventional methods for analyzing data transitions. The data transitions diagram can show visually abnormal behavior: abnormal data renewals and so on. Also, it is easy to understanding states of other variables when abnormal behavior is found.

Therefore, the proposed method can support to find the cause of a bug by visualizing data transitions, and improve efficiency of debugging for Java programs. The method reduces time to grasp behavior in executing a program for programmers by showing data transitions to them. In particular, it is useful for unskilled programmers.

The future issues are as follows.

(1) Improving the format for expressing values of variables.
(2) Introducing the localization in visualization.

References

1. Roger S. Pressman, Software Engineering A Practitioner's Approach 5thEdition, McGraw-Hill Science (2001).
2. M. Sridharan, S. J. Fink, R. Bodik, Thin slicing, In Proc. the 2007 ACM SIGPLAN Conference on PLDI (2007), pp.112-122.
3. Mark Weiser, Programmers Use Slices When Debugging, Communications of the ACM Vol.25 (1982), pp. 446-452.
4. H Agrawal, JR Horgan, Dynamic Program Slicing, SIGPLAN Notices, Vol.25, No.6 (1990), pp.246-256.
5. G. C. Murphy et al, How Are Java Software Developers Using the Eclipse IDE?, IEEE Software, Vol.23, No.4 (2006), pp.76-83.
6. Cheng Zhang et al, Automated Breakpoint Generation for Debugging, JOURNAL OF SOFTWARE, Vol.8, No.3 (2013), pp.603-616.
7. Steven P. Reiss, Guy Eddon, From the Concrete to the Abstract: Visual Representations of Program Execution, DMS 2005 (2005), pp.315-320.
8. Jan Lönnberg et al, Java replay for dependence-based debugging, PADTAD '11 (2011), pp.15-25.
9. W. De Pauw et al, Execution patterns in object-oriented visualization, In Proc. 4th COOTS (1998), pp. 219-234.