TELKOMNIKA Telecommunication, Computing, Electronics and Control
Vol. 18, No. 1, February 2020, pp. 311~316
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v18i1.12655

A proposed java forward slicing approach

Rana Bader[1], Basem Alokush[2], Mohammad Abdallah[3], Khalil Awad[4], Amir Ngah[5]
[1,2,3,4]Al-Zaytoonah University of Jordan, Jordan
[5]School of Informatics and Applied Mathematics, Universiti Malaysia Terengganu, Malaysia

Article Info

Article history:

Received Mar 13, 2019
Revised Jun 19, 2019
Accepted Jul 2, 2019

Keywords:

Direct and indirect dependency
Forward and backward slicing
Static slicing

ABSTRACT

Many organization, programmers, and researchers need to debug, test and make maintenance for a segment of their source code to improve their system. Program slicing is one of the best techniques to do so. There are many slicing techniques available to solve such problems such as static slicing, dynamic slicing, and amorphous slicing. In our paper, we decided to develop a tool that supports many slicing techniques. Our proposed tool provides new flexible ways to process simple segments of Java code, and it generates needed slicing according to the user needs, our tool will provide the user with direct and indirect dependencies for each variable in the code segments. This tool can work under various operating systems and does not need particular environments. Thus, our tool is helpful in many aspects such as debugging, testing, education, and many other elements.

Corresponding Author:

Mohammad Abdallah,
Al-Zaytoonah University of Jordan,
Amman, Jordan.
Email: m.abdallah@zuj.edu.jo

1. INTRODUCTION

Slicing methods focus on extracting code segments from the original program code. These extracted segments depend on certain variables, methods, or behavior [1]. Program slicing methods are used as a way to abstract variable related code form the original program without changing the behavior or the syntax of the original program, in 1979, Weiser [2, 3] was one of the firsts who introduced program slicing. In static slicing, all the possible sliced variable values and the execution of the program are taken into account [1]. Certain variables in the original code are the main interest for some users, and these variables can be called slices. Static slicing generates a sub-program according to the needed slice variable, this subprogram contains all the code related to the slice variable, and any unrelated code is ignored [2, 3].

The first step in static slicing, is to decide what is our slicing criteria, I mean which variable to be sliced and in which line number, let the sliced variable called (V), and located in the line number (L), so are slicing criteria is S (V, L) [2, 3]. Static slicing can generate executable slice which can be compiled and run alone without the original code, and also static slicing can generate none executable slice which cannot be compiled or run, but helpful to determine the effect of a slice [4, 5]. Static slicing introduced may different slicing types, such as conditioned, forward, backward, amorphous, decomposition, clause and quasi slicing. These types are the most frequently used types [6-11].

Program slicing is done by finding the relations within program statements, then analyzing these relations dependency. When program slicing was introduced early, Weiser [2] produced slices by analyzing the flow graph technique. He used graphs and wrote equations to demonstrate the slicing techniques. After Weiser, Ottenstein and Ottenstein [12, 13], also used graphs to determine data dependencies as slicing

technique. Due to the success of graphs dependences analysis, more new techniques in program slicing were introduced, such as condition slicing using data and control flow diagrams and dynamic slicing [14]. Other researchers introduced incremental slicing, they used data dependences computations to generate a slice. Incremental slicing is a static slicing "that computes a slice in several steps, by incorporating additional types of data dependencies at each step" [15].

Data dependence analysis is used by many researchers for object-oriented programs slicing, precisely Java [10, 16]. Researchers used static control dependence analysis and dynamic data dependence analysis as a proposed method, where they used the dependency graph structure to generate slices from the source code [11]. Later, several researchers [17-19] have designed a slicing approach for object-oriented programs written in Java. Their approach can identify data members from different objects. Moreover, they created a model that can generate class slicing, statement slicing and object slicing for java programs. Slicing semantic and notations can be considered as another way to look at static slicing, and researchers used slicing semantic and notation [15] to extract identifiers and categories different notions of a program slice available in the literature as well as several new notions.

For Java programs, many proposed tools can do static slicing, For example: watson libraries for analysis (WALA) [20] contains a slicer, based on context-sensitive tabulation of reachability in the system dependence graph. A set of WALA tools were implemented in JavaScript for more expanding. Indus [21] is an effort to provide a collection of program analyses and transformations implemented in Java to customize and adapt Java programs. It is intended to serve as an umbrella for static analyses such as points-to analysis, escape analysis, and dependence analyses, transformations such as program slicing and program specialization via partial evaluation, and any software module that delivers the analyses/transformations into a particular application or platform. Kaveri [21, 22] is an Eclipse plug-in front-end for the Indus Java slicer. It utilizes the Indus program slicer to calculate slices of Java programs and then displays the results visually in the editor. The purpose of this project is to create a useful tool for simplifying program understanding, program analysis, program debugging and testing. Recently static slicing has been used in object-oriented regression testing [23]. Moreover, new static slicing techniques were introduced in web application slicing [24], machine code [25], software robustness measurement [26, 27], and Java programs quality [28-30].

2. JAVA FORWARD SLICING TOOL (JAVA FST)

The forward slicing is a technique that is focus on the effect of a variable in a particular line number on the further code (forward) code of a program [31]. In this research, the proposed tool (JavaFST) was built using Java code to apply a forward slicing technique on any Java program. In this tool, the concerned programmer (user) can write any simple, i.e., without classes and functions, Java program as an input and then have the output as a forward sliced program depending on the slicing criteria that the programmer determines. The main contribution in the proposed tool is that it enables the programmer to input any number of lines in the program with any number of variables, by using a linked list structure instead of then using array structure in previous works.

The JavaFST mechanism, in brief, will be as follows: At first, the programmer insert the java program code (the input of JavaFST), then JavaFST will extract a list of all variables in the program. Secondly: the programmer will be asked to choose the slicing criteria; the variable to be slicing and the line number of that variable. Finally: JavaFST will display all the statements starting from the selected line in the program which has a direct or indirect dependency on the variable chosen as a forward sliced program. The JavaFST algorithm will as follow, and the diagrams in Figure 1 and Figure 2 explains the JavaFST mechanisms. Figure 1 shows how the variables are determined and prepared to be sliced. In Figure 2 the slicing process of each variable is done. The following steps explain the mechanisem:

Input: A simple Java program starting after "main" method.

Output:

a. A list with all variables in the program and the weight for each variable, where variable weight means the number of direct and indirect dependencies on it.
b. A sliced statements which are a segment from the main program (presented as a sliced list) depending on a specific variable determined by the user
   - Step 1: Declare list (prog) to store program statements and the line number of the statement.
   - Step 2: Declare another list (var) to store all the variables in the program, each entry in this list contains the variable name, the weight of the variable and a list (var_lines) to store all number of lines where there is the dependency on this variable.
   - Step 3: Input a simple java program with any number of lines and any number of variables.
   - Step 4: Add each statement to the list (prog), and each variable (v) to the list (var).

- Step 5: while (not the end of the list (prog))
- Step 6: for each variable (v) in list (var)
- Step 7: if (v) is found in the right side of an assignment statement '=' OR if the variable in the right side of the assignment affected by the variable (v) ➔ add 1 to variable (v) weight and store the line number of this statement in the (var_lines) list for this variable
- Step 8: else skip to the next statement
- Step 9: end while
- Step 10: Determine the variable you want to slice the program on it, if found ➔ go to step 11, else go to step 13.
- Step 11: Determine the line number you want to start slicing from it, if found ➔ go to step 12, else go to step 13.
- Step 12: Start slicing from the line number you determine depending on the chosen variable, and hence all the statements which their lines number were in the list (var_lines) will be stored into the sliced list (output), then go to step 14.
- Step 13: There is an error in the requested variable OR the line number; an error message will be displayed to notify that error.
- Step 14: end

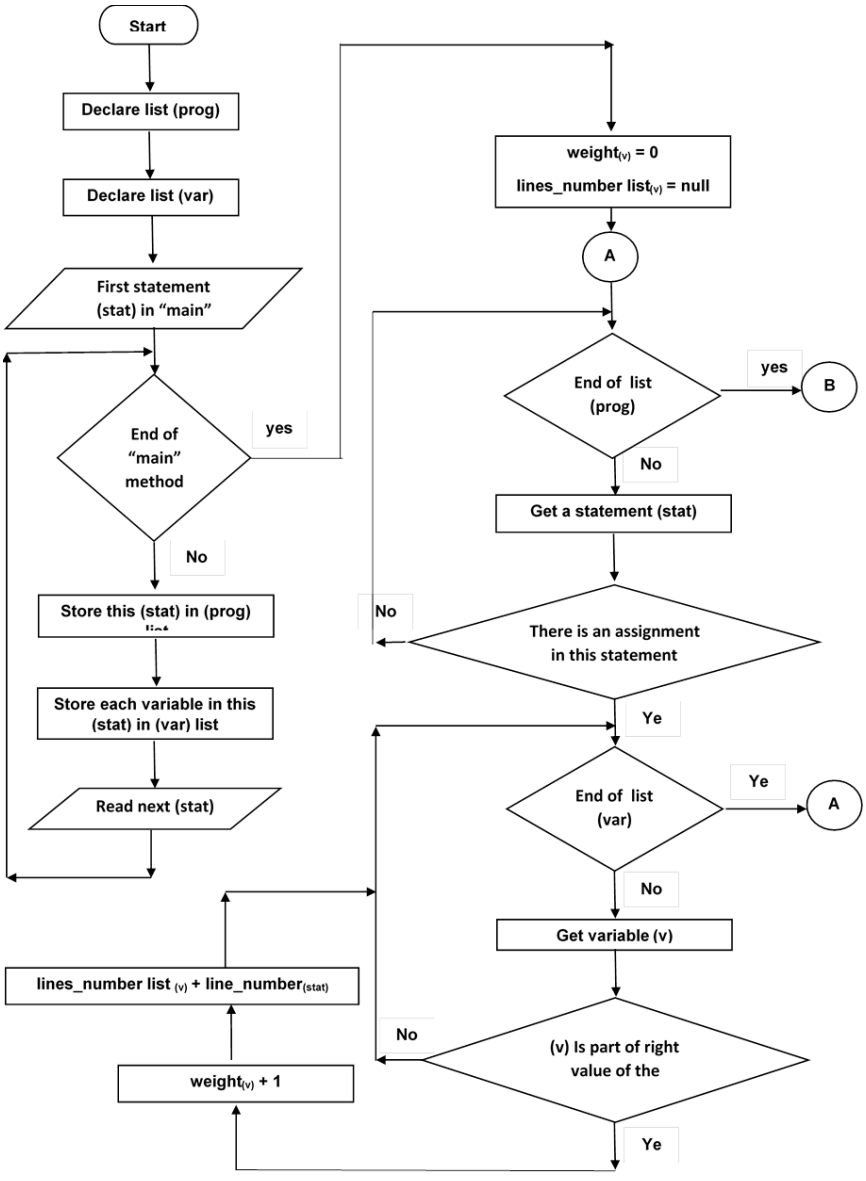

Figure 1. JavaFST variable determination mechanism

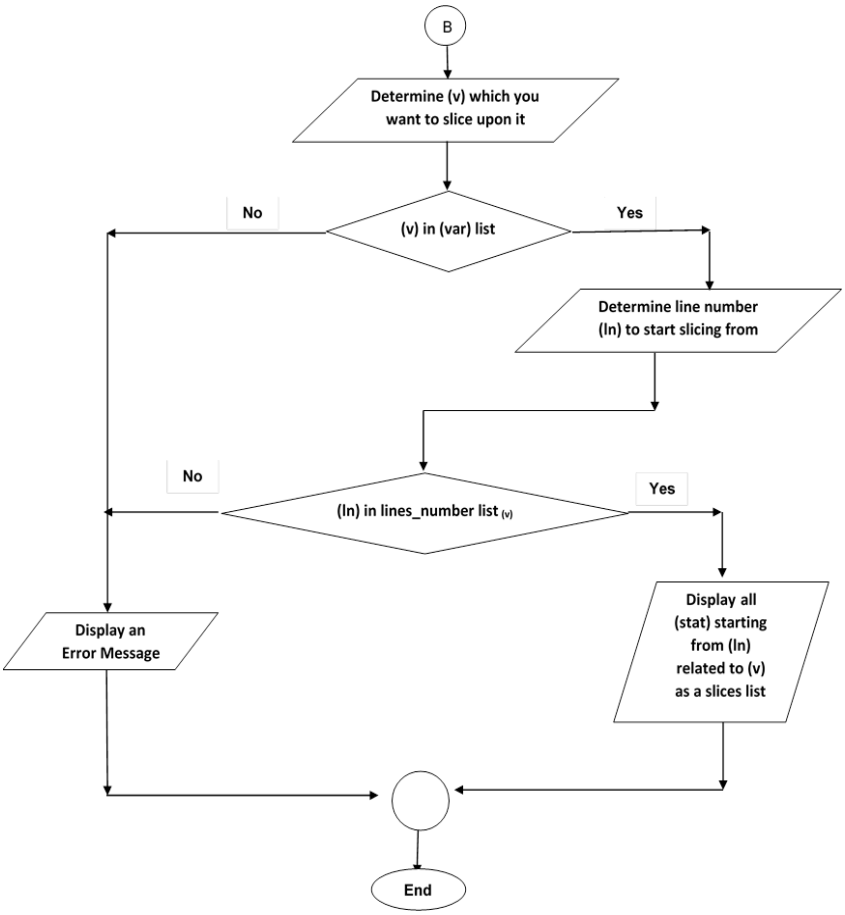

Figure 2. JavaFST slicing mechanism

3. TEST CASE AND EVALUATION

In the example below, a simple Java application code will be entered, and it has numbers of variables with different data type as shown in Figure 2. When the program end is reached, the code will display all the variables declared and used in the program. Then it will ask you to choose the variable that you want to slice on it as seen in Figure 3. Finally, Figure 4 shows the sliced code which depends on the selected variable in Figure 3.

```
Enter your simple program. You should start after main method...
1       int x=10, y=5;
2       int z=6;
3       int sum;
4       double avg;
5       sum = x+y+z;
6       avg = sum/3;
7       z = x*2;
8       int a = y+8;
9      double c;
10     c = avg + a;
# … used to indicate the program end
Number of lines: 10
```

Figure 2. Java code to be sliced

Variables in the program are:

Variable Name	Weight
x	4
y	4
z	3
sum	2
avg	1
a	1
c	0

Figure 3. The list of variables

```
Enter the variable name which you want to slice upon it:
x
Enter the line number where you want to start slicing from it:
6

The sliced code:
avg = sum/3;
z = x*2;
c = avg + a;
```

Figure 4. The sliced code

Java FST has succeeded to introduce a new slicing approach that supports forward slicing technique using LinkedList data structure. As a result, it gives the programmer the flexibility to slice as many as variables as required. JavaFST also does not need to be plugged in or work on a particular platform, such as in Indus/Kaveri or WALA. Moreover, JavaFST read a Java code stored in a text file or a text file that does not need to be compiled nor run before slicing. Therefore, more flexibility and time saving that can be useful for a testing piece of code. However, the JavaFST is still unmuture enough the slice an advanced Java programs code, i.e., contains classes and functions. Therefore, it can be used for educational purposes or simple analysis applications.

4. CONCLUSION

JavaFST is a forward slicing tool, which reads a java code from a text file, extracts the variables, and run a forward slicing on the selected slicing criteria, and it produces a complete slice for the chosen variable on a specific line. The main advantage of JavaFST that can slice an un-compiled Java program or a piece of code. Moreover, it was built to use a LinkedList to store the variable, which gives a dynamic space that allows the programmer to save as many as variable required, however, still in early stage and not able slice object-oriented programs and complex Java code.

REFERENCES

[1] S. Neelamegam, et al., "Program Slicing Techniques and its Applications," *International Journal of Software Engineering & Applications (IJSEA),* vol. 2, no. 3, pp. 50-64, 2011.
[2] M. Weiser, "Program slices: formal, psychological, and practical investigations of an automatic program abstraction method," Dissertation for the degree of Doctor of Philosophy (Computer and Communication Sciences), The University of Michigan, Michigan, 1979.
[3] M. Weiser, "Program slicing," *Proceedings of the 5th international conference on Software engineering, San Diego, California,* United States, pp. 439–449, 1981.
[4] K. Gallagher and J. R. Lyle, "Using program slicing in software maintenance," *Software Engineering, IEEE Transactions on*, vol. 17, no. 8, pp. 751-761, 1991.
[5] A. Ngah and S. A. Selamat, "A Brief Survey of Program Slicing," *International Symposium on Research in Innovation and Sustainability 2014 (ISoRIS '14),* Malacca, Malaysia, pp. 1467-1470, 2014.
[6] F. Tip, "A survey of Program Slicing Techniques," *Journal of Programming Languages*, vol. 3, pp. 121-189, 1995.
[7] D. Binkley and K. Gallagher, "Program Slicing," *Advances in Computers*, vol. 43, pp. 1-50, 1996.
[8] M. Harman and K. B. Gallagher, "Program slicing," Information and Software Technology, vol. 40, pp. 577-581, 1998.

[9] D. Binkley, et al., "An Empirical Study of Amorphous Slicing as a Program Comprehension Support Tool," *Presented at the Proceedings of the 8th International Workshop on Program Comprehension*, 2000.
[10] M. Abdallah, "A Weighted Grid for Measuring Program Robustness," PhD, Computer Science, Durham University, 2012.
[11] M. Abdallah and H. Tamimi, "Clauser: Clause Slicing Tool for C Programs," *International Journal of Software Engineering and Its Applications*, vol. 10, no. 3, pp. 49-56, 2016.
[12] K. Awad, et al., "A Proposed Forward Clause Slicing Application," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 13, no. 1, pp. 1-6, 2019.
[13] K. J. Ottenstein and L. M. Ottenstein, "The program dependence graph in a software development environment," *SIGPLAN Not.*, vol. 19, no. 5, pp. 177-184, 1984.
[14] F. Jeanne, et al., "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 9, no. 3, pp. 319-349, 1987.
[15] B. Korel and J. Laski, "Dynamic program slicing," *Information Processing Letters*, vol. 29, no. 3, pp. 155-163, 1988.
[16] A. Orso, et al., "Incremental Slicing Based on Data-Dependences Types," *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, 2001.
[17] M. Abdallah, et al., "JavaBST: Java backward slicing tool," *2017 8th International Conference on Information Technology (ICIT)*, pp. 614-618, 2017.
[18] X. Baowen, et al., "A brief survey of program slicing," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 2, pp. 1-36, 2005.
[19] B. Alokush, et al., "A Proposed Java Static Slicing Approach," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 11, no. 1, pp. 308-317, 2018.
[20] A. Ngah and S. A. Selamat, "Using Object to Slice Java Program," *Journal of Engineering and Applied Sciences*, vol. 13, no. 6, pp. 1320-1325, 2018.
[21] W. L. f. A. (WALA), "Watson Libraries for Analysis (WALA)," [Online], Available: http://wala.sourceforge.net, 2015.
[22] G. Jayaraman, et al., "Kaveri: Delivering the Indus Java Program Slicer to Eclipse," *Cerioli M. (eds) Fundamental Approaches to Software Engineering. FASE 2005. Lecture Notes in Computer Science*, vol. 3442. Springer, Berlin, Heidelberg, 2005.
[23] R. V. Prasad and H. John, "Slicing concurrent Java programs using Indus and Kaveri," *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 5, pp. 489-504, 2007.
[24] S. Panda, "Regression Testing of Object-Oriented Software based on Program Slicing," Thesis for the degree of Doctor of Philosophy in Computer Science and Engineering, National Institute of Technology Rourkela, 2016.
[25] M. Sahu and D. P. Mohapatra, "Forward Dynamic Slicing of Web Applications," *ACM SIGSOFT Software Engineering Notes*, vol. 41, no. 3, pp. 1-7, 2016.
[26] V. Srinivasan and T. Reps, "An improved algorithm for slicing machine code," *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 378-393, 2016, doi: https://doi.org/10.1145/2983990.2984003.
[27] M. Abdallah and H. A. Tamimi, "Clauser: Clause Slicing Tool for C Programs," *International Journal of Software Engineering and Its Applications*, vol. 10, no. 3, pp. 49-56, 2016.
[28] A. M. Abdalla, et al., "A Brief Program Robustness Survey," *International Journal of Software Engineering and Applications*, vol. 8, no. 1, pp. 1-10, 2017.
[29] S. Panda and D. P. Mohapatra, "ACCo: a novel approach to measure cohesion using hierarchical slicing of Java programs," *Innovations in Systems and Software Engineering*, vol. 11, no. 4, pp. 243-260, 2015.
[30] N. AlAbwaini, et al., "Using Program Slicing to Detect the Dead Code," *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, pp. 230-233, 2018.
[31] M. Abdallah and M. M. Al-rifaee, "Towards a new framework of program quality measurement based on programming language standards," *International Journal of Engineering & Technology*, vol. 7, no. 2.3, pp. 1-3, 2018.
[32] C. D. Newman, et al., "srcSlice: A Tool for Efficient Static Forward Slicing," *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pp. 621-624, 2016.