A Proposed Java Static Slicing Approach

Article · July 2018

DOI: 10.11591/ijeecs.v11.i1.pp308-317

4 authors:

Basem ghazi Alokush
Al-Zaytoonah University of Jordan
3 PUBLICATIONS 7 CITATIONS
SEE PROFILE

Mohammad Abdallah
Al-Zaytoonah University of Jordan
22 PUBLICATIONS 47 CITATIONS
SEE PROFILE

Mustafa al Rifaee
Al-Zaytoonah University of Jordan
10 PUBLICATIONS 16 CITATIONS
SEE PROFILE

Mosa Salah
Al-Zaytoonah University of Jordan
13 PUBLICATIONS 17 CITATIONS
SEE PROFILE

Indonesian Journal of Electrical Engineering and Computer Science
Vol. 11, No. 1, July 2018, pp. 308~317
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v11.i1.pp308-317

A Proposed Java Static Slicing Approach

Basem Alokush, Mohammad Abdallah, Mustafa Alrifaee, Mosa Salah
Al-Zaytoonah University of Jordan, Amman, Jordan

Article Info

Article history:

Received Nov 6, 2017
Revised Jan 8, 2018
Accepted Mar 17, 2018

Keywords:

Direct and indirect dependency
Forward and backward slicing
Static slicing

ABSTRACT

Program slicing is to abstract a part of source code depending on the point of interest. It used widely in maintenance, debugging and testing. There are many slicing techniques such as static, dynamic, and amorphous. In this paper, we choose to develop a new approach applying static slicing on Java programs. The new approach simplifies the data dependency using arrays. A new Tool called Java Multi-Slicing Tool (JavaMST) has been introduced to apply this approach.
JavaMST presents new ways to slice any simple java code segment, it allows you to extract the variables and its direct and indirect dependencies from the code, using backward, forward or both slicing techniques to produce the needed code. This tool is a simple tool designed to deal with simple java code segments. JavaMST can be run under any operating system and does not require a specialized platforms or plug-ins. Therefore, it is useful to be used for educational purposes.

Corresponding Author:

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

1. INTRODUCTION

Program slicing is an abstracting method extracts a piece of code, from a program, that focus on a subset of the program behavior or variable [1]. In 1979, Weiser [2] introduced the program slicing method as a way to abstract a variable-related piece of code preserving the behavior and syntax. The static slicing means that all possible sliced variable values and program execution will be taken into account [3]. A sliced variable is the point of interest, which is called as a slice. The static slicing produces as a result of a sub-program depending on the slice and deletes all irrelevant statements [1].

In static slicing, the first step is to identify the point of interest. A point of interest is the variable to be sliced (V) and the line number that the variable is placed (L), this called slicing criteria for the slice S(V, L) [4]. A Static slice can be executable or non-executable [5]. The executable slice means that the slice that has been produced can be compiled and run. On the other hand, the slice that is non-executable cannot be compiled, but still useful to understand the slice effect.

In static slicing, there are many different types, the frequently used types are Backward, Forward, Conditioned, Decomposition, Amorphous, and Quasi Slicing [6].

Program slice is computed by analyzing dependence relations between program statements. At the early era of program slicing, Weiser [2] used the flow graph technique to analyze the program to produce the slice. He drew graphs and write equations to explain the slicing techniques. Followed by Ottenstein and Ottenstein [7] where they used data dependence graph as a technique for program slicing.

Subsequently, new program slicing techniques have been introduced such as Dynamic Slicing and Condition Slicing using data and control flow diagrams [8, 9].

Other researchers, had used data dependencies computations to introduce the incremental slicing, which is a static slicing “that computes a slice in several steps, by incorporating additional types of data dependencies at each step” [10].

In [11] the researchers introduced the program slicing for Object-Oriented Programs, Java precisely, using the data dependence analysis. Their proposed method has based on dynamic data dependence analysis and static control dependence analysis. Where in [12] they used the dependency graph structure to capture the data from the source code.

A tool to produce a lightweight slice using program and system dependency graph to produce a list of line numbers, dependent variables, aliases, and function calls that are part of the slice for all variables both local and global for the entire system. The method is implemented as a tool [13, 14]

Chen and Xu [15] proposed an approach to slice approach the Object-Oriented programs written in Java. They produced an approach that is distinguished data members for different objects. Furthermore, they developed a model to produce a class slicing, object slicing in addition to statement slicing for Java programs.

Another way to look at the static slicing is to look at the slicing semantic and notations. In [16] the researchers identifies and categorizes different notions of a program slice available in the literature as well as several new notions.

In addition to previously proposed tools, there are off-the-shelf tools that run static slicing For Java programs. WALA [17] includes a slicer, based on context-sensitive tabulation of reachability in the system dependence graph. It also has been expanding the set of WALA tools implemented in JavaScript.

Indus [18] 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 [18, 19] 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 an effective tool for simplifying program understanding, program analysis, program debugging and testing.

Recently static slicing has been used in Object-Oriented regression testing [20]. Moreover, new static slicing techniques were introduced in web application slicing [21], machine code [22], software robustness measurement [23, 24], and Java programs quality [25, 26] and other applications [27].

The problem: The recent program slicing tools are using a particular type of programs, such as Kavari only runs as a plugin in eclipse version 3.2. Therefore, it is not easy to deal with them especially for slicing a piece of a program.

The Proposed Solution: In this paper, we have proposed a new technique to slice Java programs, using an algorithm that checks the dependency of variables directly and indirectly.

2. JAVA MULTI-SLICING TOOL (JAVAMST)

JavaMST; Java Multi-Slicing Tool, can run on any operating system or platform. JavaMST main purpose is to slice the equations from any Java file or text file. It allows slice the file using backward, forward, or both slicing techniques. It determines all the variables, their direct dependencies, and indirect dependencies, and then produces the needed code. In previous work [28] we introduce the JavaBST, which is a technique only to backward slice the Java programs. In this paper, the static, backward, and forward slicing was introduced for Java programming language.

The JavaMST algorithm is divided into five main stages; Stage 1; define the path of the code file. Stage 2; determines for all the variables, its direct dependents. Stage 3; determines for each variable, its indirect dependents. Stage 4; select slicing criteria (which variable, at which line, and slicing technique). Stage 5: produce the result code and copy it to a new file.

To build the JavaMST algorithm, we created two arrays the first called resultCode, which store the lines number and the code lines after the slicing, and the second array called allVariables, which contains objects of a class called Var.

The Var class contains:

1) Name of the variable.
2) Lines (a list contains the lines where this variable is declared or assigned a value).
3) Direct dependents (an array which contains the variables and their lines number, which current variable depends on directly).
4) Indirect dependents (an array that contains the variables and their lines number which current variable depends on indirectly).

To clarify the algorithm, we will use the following example as shown in Figure 1:

```
1	int a;
2	float b;
3	double c = 5;
4	float d;
5	int f = 12;
6	a = 20;
7	c = a + c;
8	b = b * c;
9	c = b + (a / 4);
10	d = c – 3;
11	f = d % 2;
12	b = b + d;
```

Figure 1. Code to be Sliced

First Stage; Define the path of the code file (the text file or the Java file that contains the code).

AllVariables array

Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
a	1	Null	Null		
	6				

Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
b	2	c	8		
	8	b	12		
	12	d	12		

Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
c	3	a	7		
	7	c	7		
	9	b	9		
		a	9		

Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
d	4	c	10		
	10				

Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
f	5	d	11		
	11				

Figure 2. All Variables Array (Direct Dependents)

Second Stage; Find all the variables and its direct dependents.

```
While not end of file
{
Read line.
If the current line is a declaration of a variable then
    {
    Create an object for this variable let us call it (V).
    Add (V) to the allVariables array.
    Add the current line number to the lines list of (V).
    }
If the current line contains an equation then
    {
      Get the variable name before the equal sign (=), let us call it (BV).
      Get from the AllVariables array the object with a name like (BV), let's call this object (BO).
      Add the current line to the lines list of (BO).

      For each variable after the equal sign, do the following:
      {
        Get the variable name, let's call it (AV).
        Add (AV) and current line number as a direct dependent of (BO).
      }
    }
}
```

the second stage will produce an array called (allVariables), shown in *Figure 2*.

Third Stage; Find all the indirect dependents for all the variables.

```
While not end of AllVariables array
{
  Get next Object, let us call it (V).
  Get the variable name of (V), let us call it (VN)
  Get the first line number in the lines list of (V), let us call that line number (VL).
  If ((VN) and (VL) not in the indirect dependent list of (V) ) then
    {
    Add (VN) and (VL) as indirect dependent of (V)
    }
  For each variable in the direct dependent list of (V) do the following:
  {
    Get the variable name, let us call it (D).
    Get the (D) line number, let us call it (DL).
    Get from AllVariables array the object of (D), let us call it (OD).
    Get the first line number in the lines list of (OD), let us call that line number (L).
    If ((L) < (DL))
    {
      Get from the original code file the code line that exists at the line (L), let us call that code line CodeL.
      Get the variable name from CodeL, let us call it (VT).
      If ((VT) and (L) not in the indirect dependent list of (V) ) then
      {
      Add (VT) and (L) as indirect dependent of (V)
      }
    }
    For all the elements stored in (OD) direct dependent list  do the following:
 {
      Get next line number from (OD) direct dependent list, let us call it (L).
      If ((L) < (DL))
      {
        Get the variable name of (OD), let us call it (VT)
        If ((VT) and (L) not in the indirect dependent list of (V) ) then
        {
        Add (VT) and (L) as indirect dependent of (V)
```

```
          }
        }
      }
    }
  }

Flage newInDriect = true
//newInDirect is a flage will be set to true if any new //indirect depedent is add to any object in the allVariables //array).

While (newInDriect = true)
{
  newInDriect = false
  While not end of AllVariables array
  {
  Get next Object, let us call it (V).

  For each variable in the indirect dependent list of (V) do the following:
  {
    Get the variable name from indirect list, let us call it (I).
    Get the (I) line number, let us call it (IL).
    Get from AllVariables array the object of (I), let us call it (OI).
    For all the elements stored in (OI) indirect dependent list  do the following:
 {
      Get next line number from (OI) indirect dependent list, let us call it (L).
      If ((L) < (IL))
      {
        Get the variable name of (OI) indirect list, let us call it (VT)
        If ((VT) and (L) not in the indirect dependent list of (V) ) then
        {
        Add (VT) and (L) as indirect dependent of (V)
        newInDriect = true
        }
      }
    }
  }
  }
}
```

The third stage will update the allVariablesarray, as shown in *Figure 3*.

Fourth Stage; Select the variable (V) and its line (L), then the slicing criteria.
Get the object of the selected variable (V) from the all variables array.

```
If the slicing criteria is backward slicing then
{
For all the lines numbers stored in the lines list of the selected object (V) do the following:
  {
  If the current line number in lines list is less or equal than (L) and the line  number is not in resultCode
  array then
  {
    Get from the original code file the code line which exists at the current line number.
    Add the line number and the code line to the resultCode array.
  }
}

For the all  the variables in the indirect dependent list of the selected variable (V) do the following:
{
  If the current line number of the indirect dependent variable is less than or equal (L) and the line number is
  not in resultCode array then
  {
    Get from the original code file the code line which exists at the current line number.
    Add the line number and the code line to the resultCode array.
    }
  }
}
Else if the slicing criteria is forward slicing then
{
For all the lines numbers stored in the lines list of the selected object (V) do the following:
  {
  If the current line number in lines list is greater than or equal (L) and the line  number is not in resultCode
  array then
  {
    Get from the original code file the code line which exists at the current line number.
    Add the line number and the code line to the resultCode array.
  }
}

For the all  the variables in the indirect dependent  list of the selected variable (V) do the following:
{
  If the current line number of the indirect dependent  variable is greater than or equal (L) and the line
  number is not in resultCode array then
  {
    Get from the original code file the code line which exists at the current line number.
    Add the line number and the code line to the resultCode array.
    }
  }
}
Else if the slicing criteria is total slicing then
{
For all the lines numbers stored in the lines list of the selected object (V) do the following:
{
  If the current line  number is not in resultCode array then
  {
    Get from the original code file the code line which exists at the current line number.
    Add the line number and the code line to the resultCode array.
    }
  }

For the all  the variables in the indirect dependent  list of the selected variable (V) do the following:
{
  If the current line number of the indirect dependent variable is not in resultCode array then
  {
    Get from the original code file the code line which exists at the current line number.
    Add the line number and the code line to the resultCode array.
    }
  }
}
```

AllVariables array

Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
a	1	Null	Null	a	1
	6				
Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
b	2	c	8	b	2
	8	b	12	c	3
	12	d	12	c	7
				a	1
				a	6
				d	4
				d	10
				c	9
Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
c	3	a	7	c	3
	7	c	7	a	1
	9	b	9	a	6
		a	9	b	2
				b	8
Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
d	4	c	10	d	4
	10			c	3
				c	9
				b	2
				a	1
				b	8
				c	7
				a	6
Name	Lines	Direct		Indirect	
		Name	Line	Name	Line
f	5	d	11	f	5
	11			d	4
				d	10
				c	3
				c	9
				b	2
				a	1
				b	8
				c	7
				a	6

Figure 3. All Variables Array (Direct and Indirect Dependents)

To clarify fourth stage; three examples are given below:
Example 1 : Suppose that we selected the variable "b" at line 8 and backward slicing then the result Code array will be :

Table 1. Backward Slicing (b,8)

Line number	Code line
2	float b;
8	b = b * c;
3	double c = 5;
7	c = a + c;
1	int a;
6	a = 20;

Example 2 : Suppose that we selected the variable "b" at line 8 and forward slicing then the result Code array will be :

Table 2. Forward Slicing (b,8)

Line number	Code line
8	b = b * c;
12	b = b + d;
10	d = c – 3;
9	c = b + (a / 4);

Example 3: Suppose that we selected the variable "b" at line 8 and total slicing then the result Code array will be :

Table 3. Total Slicing (b,8)

Line number	Code line
2	float b;
8	b = b * c;
12	b = b + d;
3	double c = 5;
7	c = a + c;
1	int a;
6	a = 20;
4	float d;
10	d = c – 3;
9	c = b + (a / 4);

Fifth Stage; Order the resultCode array according to the line number in ascending order, then export it to a new file.

```
Sort resultCode array in ascending order according to the lines numbers.
Create new JavaFile
While not end of resultCode array
 {
   Write the code line in the new file
 }
```

To clarify the fifth stage, the results, from the three examples in the fourth stage, will be used:
The results of example 1, will be after sorting:

Table 4. Example 1 Sorted

Result Code array	
Line number	Code line
1	int a;
2	float b;
3	double c = 5;
6	a = 20;
7	c = a + c;
8	b = b * c;

The results of example 2, will be after sorting:

Table 5. Example 2 Sorted

resultCode array	
Line number	Code line
8	b = b * c;
9	c = b + (a / 4);
10	d = c – 3;
12	b = b + d;

The results of example 3, will be after sorting:

Table 6. Example 3 Sorted

resultCode array	
Line number	Code line
1	int a;
2	float b;
3	double c = 5;
4	float d;
6	a = 20;
7	c = a + c;
8	b = b * c;
9	c = b + (a / 4);
10	d = c – 3;
12	b = b + d;

After that, the resultCode will be exported to a new text file.

3. EVALUATION

JavaMST has succeeded to introduce a new slicing approach that supports three types of slicing techniques; forward, backward, and both slicing techniques. JavaMST does not need to be plugged in or work on a certain platform, such as in Indus/Kaveri or WALA.

Moreover, JavaMST 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.

JavaMST produces a slightly small output that specifies the variable and its slices (reference variables) with line numbers. It also categorizes the slice dependency into directly or indirectly dependent. On the other side, Kaveri plug-in, it only colored the slide with yellow without showing the dependencies of it.

Since JavaMST is a simple and user-friendly slicing tool, it can be useful for educational purposes, to introduce the slicing technique to students or fresh researchers.

However, JavaMST is only sliced simple code with simple equations; it is still in the development stage. Also, it still not able to slice Object-Oriented program.

4. CONCLUSION

JavaMST is a multi-slicing tool, which reads a java code from a text file, extracts the variables, defined their direct and indirect dependencies and depending on the selected slicing criteria, and it produces a list of variables and their slices.

The main advantage of JavaMST that can slice an un-compiled Java program or a piece of code. However, still too simple to slice Object-Oriented programs and complex Java code.

REFERENCES

[1] M. Weiser, "Program slicing," presented at the *Proceedings of the 5th international conference on Software engineering*, San Diego, California, USA, 1981.

[2] M. Weiser, "Program slices: formal, psychological, and practical investigations of an automatic program abstraction method," PhD, The University of Michigan, Michigan, 1979.

[3] X. Baowen, Q. Ju, Z. Xiaofang, W. Zhongqiang, and C. Lin, "A brief survey of program slicing," vol. 30, pp. 1-36, 2005.

[4] M. Harman and R. Hierons, "An Overview of program slicing," *software focus,* vol. 2, pp. 85-92, 2001.

[5] F. John, G. Ramalingam, and T. Frank, "Parametric program slicing," presented at the Proceedings of *the 22nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, San Francisco, California, United States, 1995.

[6] N. Sasirekha, A. E. Robert, and D. M. Hemalatha, "Program slicing techniques and its applications," *International Journal of Software Engineering & Applications (IJSEA),* vol. 2, pp. 50-64, 2011.

[7] K. J. Ottenstein and L. M. Ottenstein, "The program dependence graph in a software development environment," *SIGPLAN Not.*, vol. 19, pp. 177-184, 1984.

[8] D. Binkley, S. Danicic, T. Gyim\, \#243, thy, M. Harman, *et al.*, "A formalisation of the relationship between forms of program slicing," *Sci. Comput. Program.*, vol. 62, pp. 228-252, 2006.

[9] Santosh Kumar Pani and G.B.Mund, "Property Based Dynamic Slicing of Object Oriented Programs," *International Journal of Software Engineering and Technology (IJSET),* vol. 1, pp. 69-82, 2016.

[10] A. Orso, S. Sinha, and M. J. Harrold, "Incremental Slicing Based on Data-Dependences Types," *presented at the Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01),* 2001.

[11] F. Ohata, K. Hirose, M. Fujii, and K. Inoue, "A Slicing Method for Object-Oriented Programs Using Lightweight Dynamic Information," presented at the *Proceedings of the Eighth Asia-Pacific on Software Engineering Conference*, 2001.

[12] N. F. Rodrigues and L. S. Barbosa, "Slicing for architectural analysis," *Sci. Comput. Program.*, vol. 75, pp. 828-847, 2010.

[13] C. D. Newman, T. Sage, M. L. Collard, H. W. Alomari, and J. I. Maletic, "srcSlice: a tool for efficient static forward slicing," presented at the *Proceedings of the 38th International Conference on Software Engineering Companion*, Austin, Texas, 2016.

[14] H. W. Alomari, M. L. Collard, J. I. Maletic, N. Alhindawi, and O. Meqdadi, "srcSlice: very efficient and scalable forward static slicing," *J. Softw. Evol. Process*, vol. 26, pp. 931-961, 2014.

[15] B. Xu and Z. Chen, "Dependence analysis for recursive java programs," *SIGPLAN Not.*, vol. 36, pp. 70-76, 2001.

[16] G. A. Venkatesh, "The semantic approach to program slicing," vol. 26, ed: ACM, 1991, pp. 107-119.

[17] T. J. W. L. f. Analysis, "WALA," 1.3.5 ed, 2013.

[18] G. Jayaraman and V. P. Ranganath, "Indus/Kaveri," ed, 2004.

[19] R. Venkatesh Prasad and H. John, "Slicing concurrent Java programs using Indus and Kaveri," vol. 9, ed: Springer-Verlag, 2007, pp. 489-504.

[20] S. Panda, "Regression Testing of Object-Oriented Software based on Program Slicing," 2016.

[21] M. Sahu and D. P. Mohapatra, "Forward Dynamic Slicing of Web Applications," *SIGSOFT Softw. Eng.* Notes, vol. 41, pp. 1-7, 2016.

[22] V. Srinivasan and T. Reps, "An improved algorithm for slicing machine code," *SIGPLAN Not.*, vol. 51, pp. 378-393, 2016.

[23] M. Abdallah and H. Tamimi, "Clauser: Clause Slicing Tool for C Programs," *International Journal of Software Engineering and Its Applications*, vol. 10, pp. 49-56, 2016.

[24] A. M. Abdalla, M. M. Abdallah, and M. I. Salah, "ABrief PROGRAM ROBUSTNESS SURVEY," International *Journal of Software Engineering and Applications*, vol. 8, pp. 1-10, 2017.

[25] S. Panda and D. P. Mohapatra, "ACCo: a novel approach to measure cohesion using hierarchical slicing of Java programs," *Innov. Syst. Softw. Eng.*, vol. 11, pp. 243-260, 2015.

[26] K. S. Patnaik and P. Jha, "Proposed Metrics for Process Capability Analysis in Improving Software Quality: An Empirical Study," *International Journal of Software Engineering and Technology (IJSET)*, vol. 1, pp. 152-164, 2016.

[27] J. Rashid, W. Mehmood, and M. W. Nisar, "A Survey of Model Comparison Strategies and Techniques in Model Driven Engineering," *International Journal of Software Engineering and Technology (IJSET)*, vol. 1, pp. 165-176, 2016.

[28] M. Abdallah, B. Alokush, M. Alrefaee, M. Salah, R. Bader, and K. Awad, "JavaBST: Java backward slicing tool," presented at *the 8th International Conference on Information Technology (ICIT) Amman*, Jordan, 2017.