International Symposium on Research in Innovation and Sustainability 2014 (ISoRIS '14) 15-16 October 2014, Malacca, Malaysia
Special Issue
Sci.Int.(Lahore),26(4),1467-1470,2014 ISSN 1013-5316; CODEN: SINTE 8

A BRIEF SURVEY OF PROGRAM SLICING

Amir Ngah, Siti Aminah Selamat
School of Informatics and Applied Mathematics
University Malaysia Terengganu
21030 Kuala Terengganu, Malaysia
Email: amirnma@umt.edu.my

ABSTRACT: *Program slicing is a decomposition technique that produces a new sub-program relevant to a particular computation. Program slicing was first introduced by Weiser in 1981 [27]. Since then, program slicing has grown and become an important research field in software engineering. This paper briefly describes the program representation, program slicing techniques and their applications.*

KEYWORDS: Program slicing, static slicing, dynamic slicing, conditioned slicing, decomposition slicing.

1.0 INTRODUCTION

Since Weiser's technique, program slicing has grown and become an important research field in software engineering. This fact was endorsed by Binkley and Gallagher [3], who stated that the number of citations for the paper by Weiser on program slicing increased significantly year by year. Recently, there are a number of papers that have done a survey on program slicing techniques and it applications [12, 9, 22, 25]. Since Weiser's first program slicing technique, many program slicing techniques have been introduced such as dynamic slicing [21, 1], forward slicing [2], decomposition slicing [15], interprocedural slicing [20], conditioned slicing [5], stop-list slicing [13], amorphous slicing [4], hybrid program slicing [24] and abstract slicing [19, 28].

Program slicing is a decomposition technique that produces a new sub-program relevant to a particular computation. The new sub-program is called a slice, and is an executable program that is produced from the original program with respect to the specified slicing criterion. Slicing criterion is a set of conditions used in the slicing computation to produce a slice. A basic slicing criterion uses two main parameters. They are the variable or a set of variables and the location of interest. This paper is organized in three main sections. The next section discusses the representation of programs or systems. This is followed by a discussion of program slicing techniques in the third section. The fourth section is about the applications of program slicing.

2.0 REPRESENTATION OF PROGRAM

There are three different representations used in different types of slicing such as control flow graphs, program dependence graph, and system dependence graph. A brief explanation of these representations is given below.

2.1 Control Flow Graph

Tip [26] states that Weiser's approach uses data flow and control flow dependences in order to compute a slice. A Control Flow Graph (CFG) is a representation of the program with the combination of nodes and edges from the start node to the end node. A CFG represents control dependencies of the program. An example of CFG is shown in Figure 1. Every statement in the program is represented by nodes. The flow from one node to another node is called an edge. Nodes 1 and 4 are called predicate nodes because they have more than one out going edge. A path is the flow from the start node (node 1) to the end node (node 7). Nodes 6 and 7 are non-branching statements which can be treated as one statement unit [10]. There are four unique paths through the CFG in Figure 1.

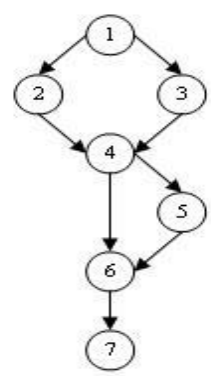

Figure 1: The Control Flow Graph

2.2 Program Dependence Graph

A Program Dependence Graph (PDG) is an intermediate representation of a program using a combination of data dependences and control dependences of the program [11, 20, 23]. Data dependences are used to represent data flow relations of the program. Control dependences represent control flow relationships of the program. Control dependences are derived from the CFG. For instance, in Figure 2, statement 7 is dependent on statement 3 because statement 7 has the use of the variable sum that depends on its definition at statement 3. The relation of both statements is called data dependence. Statement 5 and 7 show the relationship between statement and predicate. Statement 7 is dependent on statement 5 as a predicate. This dependence is called the control dependence.

```
(1)  read (n);
(2)  i := 1;
(3)  sum := 0;
(4)  product := 1;
(5)  while i <= n
(6)  {
(7)    sum := sum + 1;
(8)    product := product * i;
(9)    i ++;
(10) }
(11) write (sum);
(12) write (product);
```

Figure 2: The Program to be Sliced [26]

2.3 System Dependent Graph

Horwitz et al [20] have introduced the concept of System Dependence Graph (SDG). SDG is an extension of the PDG. It includes the PDG, which represents the main program of the system; procedure dependence graphs, which represent the procedures of the system; and some additional edges. There are two types of additional edges. These are the edges that represent direct dependences between a call site and the called procedure, and edges that represent transitive dependences due to calls. In SDG, transitive interprocedural flow dependences are represented by using heavy bold arcs. The call edges, parameter-in edges, and parameter-out edges which connect program and procedure dependence graphs together are represented by using dashed arrows.

3.0 PROGRAM SLICING TECHNIQUES

3.1 Static and Dynamic Slicing

The first program slicing technique by Weiser was based on static program analysis [27]. Weiser's program slices are called an executable static slice [3]. It is called an executable because the slices are an executable program and called static because the computation of slices is performed without considering the input of the program. Figure 2 shows a program which computes the value of variable sum and product if the input n is a positive number. A slice of this program with respect to the slicing criterion (product, 12) is all statements that are involved in the computation of the variable product at line 12. In other words, all statements that are involved in the computation of the variable sum have been excluded from the slice. The statements that are involve in this slice are 1, 2, 4, 5, 6, 8, 9, 10, and 12.

Korel and Laski [21] have proposed dynamic slicing as a counterpart of Weiser's static slicing technique. Their technique has considered the input values in the computation of slice. They introduced the concept of the trajectory which is the path that has actually been executed for some input. The concepts of data flow and control flow are used in order to produce Data-data (DD) and Test Control (TC) relations based on the trajectory. The DD relation is equivalent to the concept of *definition-use* (du) and the TC relation is based on control dependence. A dynamic slice can be computed by using the DD and TC relations. The main element in their technique is that they compute a slice based on a program execution (trajectory) not a CFG. Agrawal and Horgan [1] have also discussed dynamic slicing. They have introduced the concept of Dynamic Dependence Graph (DDG) that is based on the PDG. The only difference between them is that the DDG creates a separate node for each occurrence of a statement in the execution history. In other words, the number of nodes in the DDG is equal to the number of statements in the execution history including repeated statements.

3.2 Backward and Forward Slicing

Weiser's program slicing technique is also known as a backward slicing. It is known as Backward because the way edges are traversed using a dependent graph. Weiser's backward slicing computes slices using the data flow analysis that begins by tracing backward the possible statements that have influences on the variable of interest. For example, the slice for the program in Figure 2 with respect to the variable sum at line 11 is statements 1, 2, 3, 5, 6, 7, 9, 10 and 11. The computation of the slice starts at line 11 which is the use of the variable sum. From the use of this variable sum, the slice will be computed backward using the CFG. The last definition (def) of the variable sum is at line 7. From this line all related definition-uses are considered in the slice. Bergeretti and Carre [2] have introduced the notion of forward slicing. Forward slicing includes all statements that depend on the slicing criterion. Forward slice can be obtained from the PDG. Horwitz et al. [20] have computed forward slices for interprocedural program based on the SDG.

3.3 Conditioned Slicing

Conditioned program slicing was first introduced by Canfora et al. [5] and later modified as variants [17,8, 18, 7]. The conditioned program slicing forms a bridge between the static and dynamic analysis. The conditioned slicing criterion is a triple, (p, V, n) where p is some initial conditions of interest and (V, n) are the two elements of the static slicing criterion.

3.4 Stop-List Slicing

Early program slicing techniques required two parameters: a variable or a set of variables, and a program location of interest. All statements related to this slicing criterion are included in the program slice. Gallagher et al. [13] have introduced a new technique that has considered a third additional parameter in the slicing criterion. The third parameter is called stop-list and is a set of variables that are not of interest. The computation of a stop-list slice will exclude all statements that are related to these excluded variables by using the data-flow dependence analysis. In theory, this technique has the potential to reduce the size of slice compared to the traditional slicing techniques. The evaluation of this technique by Gallagher et al. [13] shows that the results are encouraging giving a large reduction in the slice size.

3.5 Decomposition Slicing

Gallagher and Lyle [15] have introduced the term decomposition slicing. The technique uses slicing to decompose a program directly into two parts, decomposition slice and complement. The decomposition slice is built for one variable and is the union of all slices taken at line numbers of the uses of the given variable. The calculation of these slices can use any independent slicing techniques. Therefore, the quality of the decomposition slice is dependent on the quality of the slice itself. The complement is the sub-program that remains after the decomposition slice is removed from the original program.

4.0 APPLICATIONS OF PROGRAM SLICING

4.1 Debugging

The original program slicing technique by Weiser was developed to aid debugging activities [27]. In debugging, the purpose is to identify errors that occur in the program. Program slicing techniques can assist the debugger to detect errors and the affected statements without considering the unrelated statements. Program slicing can minimize the size of the original program to the parts of interest based on the slicing criterion. The application of debugging has also motivated the introduction of dynamic slicing [16]. Dynamic slicing [1, 21] can offer a better assistant in debugging. It can produce a smaller slice compared to static slicing for a specific program input.

4.2 Program Comprehension

An early part of the software maintenance phase is program comprehension. Program slicing can be used to assist the program comprehension process. For instance, Canfora et al. [5] have used conditioned slicing in the context of program comprehension and reused existing software. Conditioned slicing enables the computation of refined code fragments implementing specific program behaviors. Binkley et al. [4] have used amorphous slicing for program comprehension.

4.3 Software Maintenance

Software maintenance is always dealing with changes. It determines whether a change at some parts of the program will affect the behavior of the other parts of the program. Program slicing can be used in order for the maintainer to concentrate only on the modified parts of the program. This can minimize the chances of introducing unexpected errors. Gallagher and Lyle [15] have introduced decomposition slicing that was used in a new software maintenance process model.

4.4 Software Testing

There are two main structural based testing techniques: control flow testing and data flow testing. Program slicing techniques are based on the manipulation of control flow and data flow graphs. The important part of software testing that applies program slicing techniques is regression testing. Slicing based testing techniques have been discussed in [14, 6].

5.0 CONCLUSION

This paper briefly explains the techniques of program slicing. Since Weiser's first technique, many program slicing techniques has been introduced such as dynamic slicing, forward slicing, and decomposition slicing, and conditioned slicing. This paper also classifies the slicing techniques into some applications such as debugging, software maintenance, program comprehension and regression testing.

ACKNOWLEDGEMENTS

This research is sponsor by Ministry of Education, Malaysia Government under Research Acculturation Collaborative Grant (RACE).

REFERENCES

[1] Hiralal Agrawal and Joseph Robert Horgan. Dynamic program slicing. In Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'90), pages 246–256, 1990.

[2] Jean-Francois Bergeretti and Bernard A. Carré. Information-flow and data-flow analysis of while-programs. ACM Transactions on Programming Languages and Systems (TOPLAS), 7(1):37–61, 1985.

[3] David Binkley and Keith Brian Gallagher. Program slicing. Advances in Computers, 43:1–50, 1996.

[4] David Binkley, Mark Harman, L. Ross Raszewski, and Christopher Smith. An empirical study of amorphous slicing as a program comprehension support tool. In Proceedings of the 8th International Workshop on Program Comprehension, pages 161–170, 2000.

[5] Gerardo Canfora, Aniello Cimitile, and Andrea De Lucia. Conditioned program slicing. Information & Software Technology, 40(11-12):595–607, 1998.

[6] Omar Chebaro, Nikolai Kosmatov, Alain Giorgetti, and Jacques Julliand. Program slicing enhances a verification technique combining static and dynamic analysis. In Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, pages 1284–1291, New York, NY, USA, 2012. ACM.

[7] Sebastian Danicic, Andrea De Lucia, and Mark Harman. Building executable union slices using condi- tioned slicing. In Proceedings of the International Workshop on Program Comprehension (IWPC'04), pages 89–99, 2004.

[8] Mohammed Daoudi, Lahcen Ouarbya, John Howroyd, Sebastian Danicic, Mark Harman, Chris Fox, and Martin P. Ward. Consus: A scalable approach to conditioned slicing. In Proceedings of the 9th Working Conference on Reverse Engineering (WCRE'02), pages 109–118, 2002.

[9] Chen Duanzhi. Program slicing. In Preceedings of the International Forum on Information Technology and Applications, pages 15–18, 2010.

[10] Elena Dubrova. Structural testing based on minimum kernels. In Proceedings of the Conference on Design, Automation and Test in Europe (DATE'05), pages 1168–1173. IEEE Computer Society, 2005.

[11] Jeanne Ferrante, Karl J. Ottenstein, and Joe D. Warren. The program dependence graph and its use in optimization. ACM Transactions on Programming Languages and Systems (TOPLAS), 9(3):319–349,1987.

[12] Keith Gallagher and David Binkley. Program slicing. In Proceedings of the Frontiers of Software Maintenance, pages 58–67, 2008.[13] Keith Gallagher, David Binkley, and Mark Harman. Stop-list slicing.

In Proceedings of the IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'06), pages 11–20, 2006.

[14] Keith Gallagher, Tracy Hall, and Sue Black. Reducing regression test size by exclusion. In Proceed- ings of the International Conference on Software Maintenance (ICSM'07), pages 154–163, 2007.

[15] Keith Brian Gallagher and James R. Lyle. Using program slicing in software maintenance. IEEE Transactions on Software Engineering, 17(8):751–761, 1991.

[16] Mark Harman and Robert M. Hierons. An overview of program slicing. Software Focus, 2(3):85–92,2001.

[17] Mark Harman, Robert M. Hierons, Chris Fox, Sebastian Danicic, and John Howroyd. Pre/post condi- tioned slicing. In Proceedings of the International Conference on Software Maintenance (ICSM'01), pages 138–147, 2001.

[18] Robert M. Hierons, Mark Harman, Chris Fox, Lahcen Ouarbya, and Mohammed Daoudi. Conditioned slicing supports partition testing. Software Testing, Verification and Reliability, 12(1):23–28, 2002.

[19] Hyoung Seok Hong, Insup Lee, and Oleg Sokolsky. Abstract slicing: a new approach to program slicing based on abstract interpretation and model checking. In Proceedings of the Fifth International Workshop on Source Code Analysis and Manipulation, pages 25–34, 2005.

[20] Susan Horwitz, Thomas W. Reps, and David Binkley. Interprocedural slicing using dependence graphs. ACM Transactions on Programming Languages and Systems (TOPLAS), 12(1):26–60, 1990.

[21] Bogdan Korel and Janusz W. Laski. Dynamic program slicing. Information Processing Letters, 29(3):155–163, 1988.

[22] Andrea De Lucia. Program slicing: Methods and applications. In Proceedings of the First IEEE International Workshop on Source Code Analysis and Manipulation, pages 142–149, 2001.

[23] Karl J. Ottenstein and Linda M. Ottenstein. The program dependence graph in a software development environment. In Proceedings of the 1st ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, pages 177–184, 1984.

[24] Juergen Rilling and Bhaskar Karanth. A hybrid program slicing framework. In Proceedings of the Fifth IEEE International Workshop on Source Code Analysis and Manipulation, pages 12–23, 2001.

[25] N. Sasirekha, A. Edwin Robert, and M. Hemalatha. Program slicing techniques and its applications. International Journal of Software Engineering and Applications (IJSEA), 2(3), 2011.

[26] Frank Tip. A survey of program slicing techniques. Journal of Programming Languages, 3:121 189,1995.

[27] Mark Weiser. Program slicing. In Proceedings of the International Conference on Software Engineer- ing (ICSE'81), pages 439–449, 1981.

[28] Damiano Zanardini. The semantics of abstract program slicing. In Proceedings of the Eighth IEEE International Working Conference on Source Code Analysis and Manipulation, pages 89–98, 2008.