A New Method to Compute Dynamic Slicing using Program Dependence Graph

Lipika Jha,
Department of Computer Science
and engineering,
Birla Institute of Technology,
Mesra, Ranchi (India)

K.S.Patnaik, PhD
Department of Computer Science
and engineering
Birla Institute of Technology
Mesra, Ranchi (India)

ABSTRACT

Program slicing is one of the techniques of program analysis that allows an analyst to automatically extract portions of programs relevant to the program being analyzed. It is an alternative approach to develop reusable components from existing software. It is a very important part of software development and maintenance. It is used in a number of applications such as program analysis, program debugging, reverse engineering, software testing, software maintenance, program understanding etc. In 1984, Weiser has introduced the concept of slicing. Earlier, static slices were used but now mainly dynamic slices are being used which further reduces the program size. In static slicing, only statically available information is used for computing slices whereas in dynamic slicing it includes all statements that affect the value of the variable occurrence for the given program inputs, not all statements that did affect its value. In this paper we have proposed a new method for computing dynamic slicing.

Keywords

Control flow graph, program dependence graph, dynamic slicing criteria, executable dynamic slicing, non-executable dynamic slicing, dynamic dependence graph, execution history

1. INTRODUCTION

Program slicing means reducing the given program to a minimal number of statements with respect to a given criteria S_c(n,v),where n is the variable and n the number of statement in the program. In other words finding all statements in a program that directly or indirectly affect the value of a variable occurrence is referred to as static slicing [5].Dynamic slicing should evaluate the variable occurrence identically to the original program for all test cases. Program slicing is used for a large number of computer applications such as debugging, maintenance, testing, etc. Slicing is concerned with finding all statements that could influence the value of the variable occurrence for any inputs. The size of a static slice may approach the original program, and the usefulness of a slice tends to diminish as the size of the slice increases. Therefore, in this paper we examine a dynamic slice, consisting only of statements that influence the value of a variable occurrence for specific program inputs. A dynamic slice of a program is constructed by analyzing an execution history of the program to discover data and control dependences.

This paper describes the earliest approach and the new approach to compute dynamic slicing. However, the emphasis of this paper is primarily of a theoretical rather than of a practical. The goal of this paper is to develop more precise dynamic slicing algorithm.

The rest of the paper is organized as follows. First section defines the basic concepts of types of slicing, control and data dependence and different graph used for computing slicing. Next defines related work. Then defines new method to compute dynamic slicing and finally future work.

2. BASIC CONCEPTS

Some basic terms and notations related to slicing in the following sections.

Types of Slices.

Static Slice. The slice which is computed for a general set of variables is called static slices i.e., static slices are the slices of all the values of the variables involved in the program.

Dynamic Slicing. It includes all statements that affect the value of the variable occurrence for the given program inputs, not all statements that did affect its value. Dynamic slicing criterion consist of a triple (n, V, I) where I is an input to the program.

Quasi Slicing. It is a hybrid of Static and Dynamic Slicing. Static slicing is computed during compile time, without having any information about the input variables of the program. Dynamic slicing analyses the code by giving input to the program. It is constructed at runtime with respect to a particular input. In Quasi slicing the value of some variables are fixed and the program is analyzed while the value of other variables vary. The behavior of the original program is not changed with respect to the slicing criterion.

Conditioned Slicing. It is a technique to compute program slices with respect to a subset of program executions.It is an extension of the static slicing. It includes the set of values in which the program is to executed, allowing the programmer to specify, not only the variables of interests, but also the initial conditions of interest. Any statements, which we know it will not execute, may be omitted afterwards. This shows the initial awareness of knowledge about the condition in which the program is to be executed and also has the advantage that it allows for additional simplification during slice construction.

Backward Slicing. It includes all parts of the program that might have influenced the variable at the statement under

consideration. The backward approach can be used in locating the bug by examining all previously executed statements with respect to a variable v at statement n, where n is the statement no. where error is found.

Forward Slicing. Contains all those statements of P which might be influenced by the variable.

Amorphous slice. All approaches to slicing discussed so far have been 'syntax preserving', That is, they are constructed by the sole transformation of statement deletion. The statements which remain in the slice are therefore a syntactic subset of the original program from which the slice was constructed. Amorphous slices are constructed using any program transformation which simplifies the program and which preserves the effect of the program with respect to the slicing criteria.

Dependency. Each statement of a code is dependent on other statement in some way, this is known as dependency. It is of two types:

Data Dependency. When a statement or a variable is dependent on some other statement for some data it is known as data dependency.

Control Dependency. When the execution of a statement is dependent on some other statement it is called as control dependency.

Visualisation of Slices. Visualisation of slices is a very efficient technique for analysing understanding and developing slices. It is done in following ways:

Control Flow Graph. It is simple representation of control flows and thus the flow in which statements are executed [10].It is used for data flow analysis. It consists of nodes, directed edges, unique exit node STOP and unique entry node START. It is an intermediate representation for slicing.

Program Dependence Graph. The program dependence graph of a program has one node for each simple statement and one node for each control predicate expression. It has two types of directed edges: data dependence edges and control dependence edges.

2.1 Dynamic Slicing

It is a technique for program debugging and understanding. The concept of dynamic program slicing was first introduced by Korel and Laski [5].It includes all statements that affect variable occurrence for the given program inputs, not all statements that did affect its value. It consists of a triple (n,V,I) where I is an input to the program. In static slicing only statically available information is used for computing slices. Whereas in dynamic slicing all possible inputs is used for computing slices. By taking a particular program execution in consideration, dynamic slicing may significantly reduce the size of the slice as compared to static slicing. Most of the existing dynamic slicing techniques have been proposed for sequential programs [6 and 8]. Two major types of dynamic slicing have been proposed: executable dynamic slicing and non-executable dynamic slicing. An executable dynamic slice is a set of statements that can be executed and it preserves a value of a variable of interest. On the other hand, a non-executable slice is a set of statements that influence the variable of interest and, it cannot be executed.

Executable Dynamic Program Slicing. A dynamic slicing criterion of program P executed on program input x is a triplet C=(x, y,q) where y is a variable at execution position q. An executable dynamic slice of program P on slicing criterion C is any syntactically correct and executable program P' that is obtained from P by deleting zero or more statements, and when executed on program input x produces an execution trace T^x for which there exists the corresponding execution position q' such that the value of y in T^x equals the value of y' in T^x. A dynamic slice P' preserves the value of y for a given program input x. The goal in dynamic slicing is to find the slice with the minimal number of statements, but, in general, this goal may not be achievable. However, it is possible to determine a safe approximation of the dynamic slice that preserves the computation of the value of a variable of interest.

Non-executable dynamic program slicing. For a given slicing criterion C=(x,y_q), a non-executable dynamic slice contains statements that "influence" the variable of interest y_q during program execution on input x. Non-executable dynamic slices cannot be executed. Most of the existing methods of computation of dynamic slice use the notion of data and control dependencies to compute non-executable dynamic program slices.

3. REVIEW OF RELATED WORK

Weiser first introduced the idea of slicing in 1984 [5]. He introduced the idea of static slicing using control flow graph. The major disadvantage of his approach was that each slice was computed from beginning i.e., during computation of slices nothing was saved or stored for future use. Then Ottenstein introduced the idea of PDG (program dependency graph) and used it to compute intraprocedural slices [10]. Horwitz took his idea further to SDG (System Dependency Graph) and computed interprocedural slices.

However static slice reduce the size of the program but it was not precise one. The concept of dynamic slicing was introduced by the Korel and Laski using Weiser CFG [6]. The method used by Korel and Laski becomes useless when there are many loops in the program. The transaction history becomes very long and difficult to find the dependence relation. For the first time Agrawal and Horgan used dependence graphs to compute dynamic slices. They also introduced the idea of precise dynamic slices and proposed DDG (Dynamic Dependency Graph) for computing precise dynamic slices. In this a new node is created for each executed node and its associated nodes. Mund proposed the concept of stable and unstable edges and use them to create dynamic slices. They further improved their algorithm and proposed an edge marking and unmarking algorithm and also node marking and unmarking algorithm. They proved that their algorithms are better than others in terms of precision, time complexity and space complexity. Most of these algorithms calculate backward slices. Much of the literature on program slicing is concerned with improving the algorithms of slicing keeping in mind reduction of the size of the slice and improvement the efficiency of computation. All the works focus on computation of precise dependence information and the accuracy of the computed slices.

Here in this method program dependency graph is used to compute dynamic slicing. In this PDG is independent of the input value, if the slicing variable is same the number of dynamic slicing can be computed with different input value.

4. NEW METHOD

Steps to compute dynamic slicing

Step 1: Construct PDG (Program Dependency Graph) of the program.

Step 2: Compute static slicing of the program using vertex reach ability problem. Mark the nodes as it reach.

Step 3: Execute the sliced program by taking any input value and store the logical value of the conditional statement and loop statement and the number of times as the statement executes

Step 4: Create the modify PDG based on static slicing computed using PDG.

Step 5: Start from the marked vertex and check whether it is conditional or loop statement, depending upon its logical value left or right vertex will be included in dynamic slice.

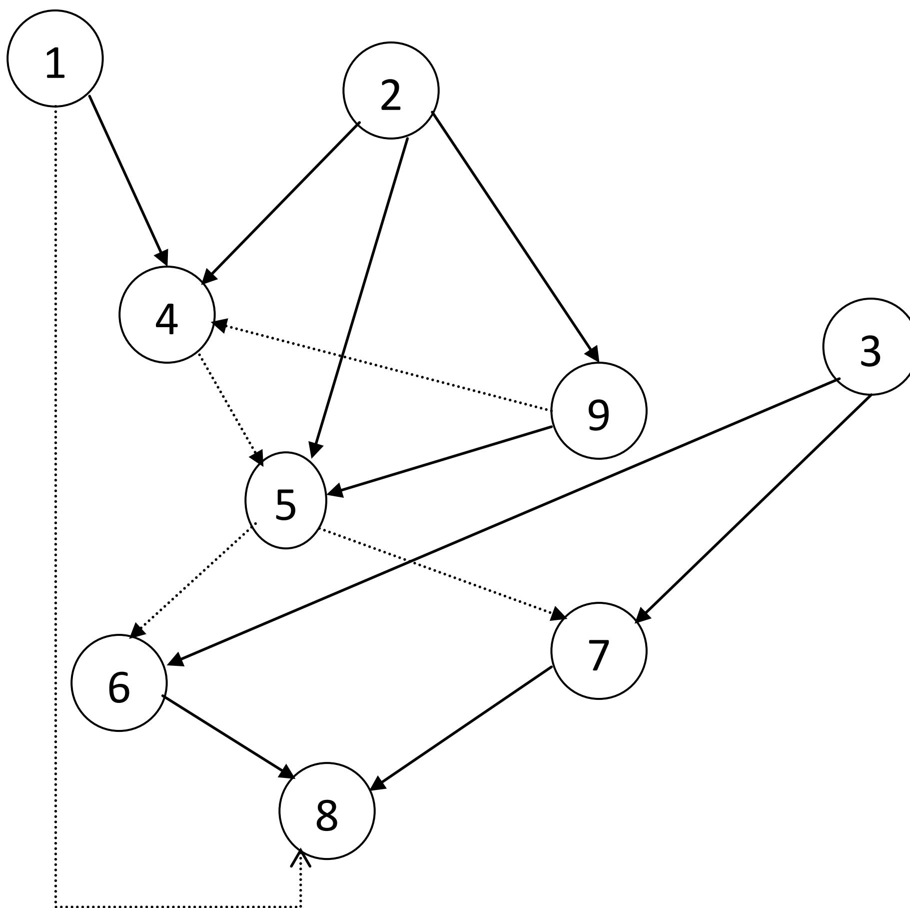

Figure 1: PDG of Example 1

Dynamic Slicing Criteria $S_c(8,c,5)$

- Compute the static slicing $S_c(8,c)$ using vertex reachability problem.
- Static slicing of program 1 will be {1,2,3,4,5,6,7,8,9}
- Execute the sliced program and store the logical value.

Example 1:

```
integer a,b,c;
1.read(a)
2.b = 1
3.c = 4
4.while(b <= a) do
5. if((b mod 2) > 0) then
6. c = c + 9
else
7. c = 10
8.write(c)
9.b = b + 1
endwhile
```

a) If a=2 then

First iteration
statement 4: true,1

statement 5:false,1

Second iteration
statement 4: true,2
statement 5:true,2

Third iteration
statement 4:false

- Modify PDG to reduce its size.

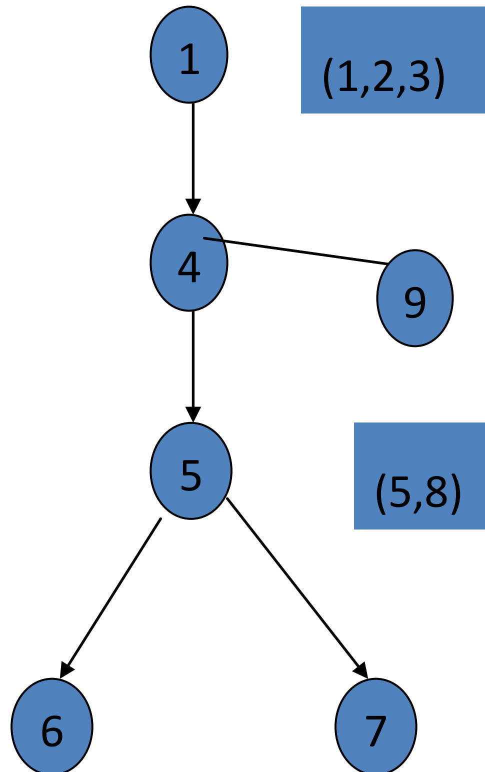

Figure 2: Modified PDG of Example 1

Start computing the slice from first vertex if it is loop control statement and iteration is more than one include current vertex and right vertex in the slice and if it is conditional control statement and its logical value is true include it in the slice and go to left vertex else right vertex or else go to the next vertex. Continue till it further cannot traverse the vertex of the modified graph.

If a=2 then

Include vertex1 in slice it is not a control statement go to next vertex 4.Include vertex 4 in slice it is loop control statement and its logical value is true and number of statement is more than once include both left(5) and right(9) vertex in slice and go to left vertex(5). Vertex 5 is conditional statement and its logical value is true go to left vertex.

dslice={1,2,3,4,5,7,8,9}

If a=0 then

Include vertex1 in slice it is not a control statement go to next vertex 4.Include vertex4 in slice it is loop control statement and its logical value is false it cannot be further traverse. So dynamic slice will be

dslice={1,2,3,4}.

If a=1 then

Include vertex1 in slice it is not a control statement go to next vertex 4.Include vertex 4 in slice it is loop control statement and its logical value is true and number of iteration is one include left(5) vertex in slice and go to left vertex(5). Vertex 5 is conditional statement and its logical value is false go to right vertex.

dslice={1,2,3,4,5,7,8}.

Example program 2:

integer m,a,i,b,x,y,z;

1.read(m);

2.a = 0;

3.i = 1;

4.b = 2;

5.while(i <= m) do

6.read(x);

7.if(x <= 0) then

8.y = x + 5;

else

9.y= x - 5;

10.z= y + 4;

11.if(z>0) then

12.a= a + z;

else

13.b= a + 5;

14.i= i + 1;

endwhile

15.write(a);

16.write(b);

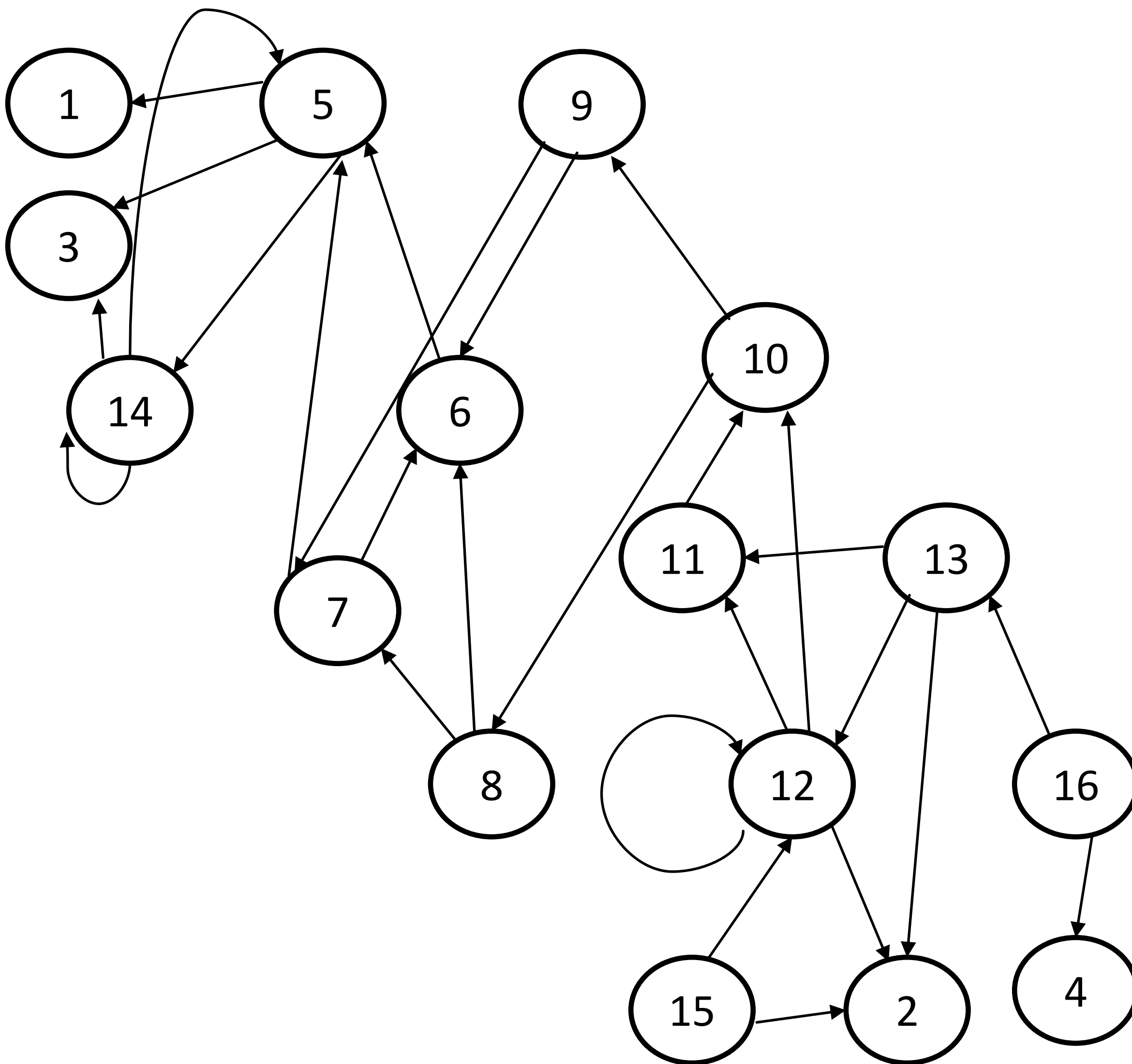

Figure 3 : PDG of Example 2

Dynamic Slicing Criteria $S_c(15,a,m,x)$

- Compute the static slicing $S_c(15,a)$ using vertex reachability problem.

- Static slicing of program 2 will be {1,2,3,,5,6,7,8,9,10,11,12,14,15}
- Execute the sliced program and store the logical value.

b) If m=2 &
 x=5 then

First iteration
statement 5: true,1
statement 7:true,1
x=-1
Second iteration
statement 5: true,2
statement 7:false,2

Third iteration
statement 5:false

- Modify PDG to reduce its size.

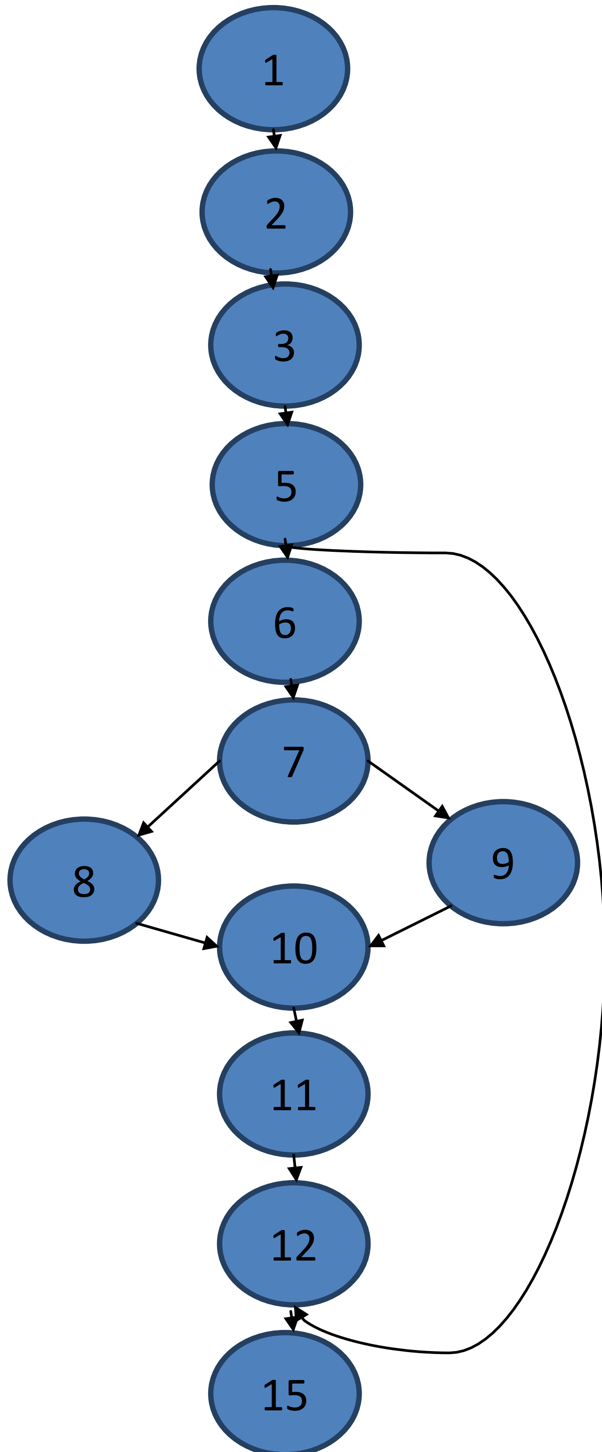

Figure 4: Modified PDG of Example 2

Start computing the slice from first vertex if it is loop control statement and iteration is more than one include current vertex and right vertex in the slice and if it is conditional control statement and its logical value is true include it in the slice and go to left vertex else right vertex or else go to the next vertex. Continue till it further cannot traverse the vertex of the modified graph.

If m=2 & x=5
Follow the steps for computing dynamic slicing.
dslice={1,2,3,,5,6,7,9,10,11,12,15}

If m=2 &x=-1
dslice={1,2,3,,5,6,7,8,10,11,12,15}

If m=-1
dslice={1,2,3,,5,15}

5. COMPARATIVE ANALYSIS

The above proposed method is more efficient for the input values which have the same slicing criteria. But for the different slicing criteria the different modified PDG will be created. Whereas in earlier approach for dynamic slicing the separate DDG is created for each input value independent of the slicing criteria. The proposed method is very useful for performing all possible test cases in same criteria using only one PDG.
Tabular representation for comparative analysis using the above two examples:

	No. of PDG in old method	No. of PDG in new method
Example1	3	1
Example2	3	1

From the above tabular representation we can conclude that the method which takes more no. of PDG requires more time and space to compute the slices.

6. CONCLUSION

Since in this we have to create a new PDG and modified PDG if the slicing variable changes which consume a lot of space and time. Thus, there is a lot of scope for further development w.r.t space and time complexity.

7. ACKNOWLEDGEMENT

I wish to convey my sincere gratitude and appreciation to each and every person who helped me in writing this paper. I am grateful to my institution, Birla Institute of Technology and my colleagues. I would especially like to thank Dr. K. S. Patnaik, my guide for his advice and guidance.

8. REFERENCES

[1] David W. Binkley and Keith Brian Gallagher (1996), "Program Slicing", Advances in Computers, Volume 43, page 1-45.

[2] G. B. Mund, R. Mall, S. Sarkar (2002), "An efficient dynamic program slicing technique", Department of Computer Science and Engineering, IIT Kharagpur, pages 23-132.

[3] G. B. Mund, R. Mall, S. Sarkar (2003), "Computation of intraprocedural dynamic program slices", Department of Computer Science and Engineering, IIT Kharagpur, pages 123-132.

[4] M. Weiser (1982), “Programmers use slices when debugging”, Communication of the ACM 25,

pages 446-452.

[5] M. Weiser (1984), “Program Slicing”. IEEE Transactions on Software Engineering, pages 352-357.

[6] B. Korel, S. Laski (1988) “Dynamic Program Slicing”, Information Processing letters, pages 155-163.

[7] K Ottenstein and L. Ottenstein (1984), “The Program Dependence Graph in Software Development Environment”, Procedings of the ACM SIGSOFT/SIGPLAN Software Engineering environments, SIGPLAN Notices, pages 177-184.

[8] H. Agrawal and J. Horgan (1990) “Dynamic Program Slicing”, Proceedings of ACM SIGPLAN 90 conference on Programming Language Design and implementation, pages 246-256.

[9] Korel B. and Yalamanchili S. (1994), “Forward Derivation of Dynamic Slices”. Proceedings of the International Symposium on Software Testing and Analysis, pages 66-79.

[10] Korel B. (1995), “Computation of dynamic slices for programs with arbitrary control-flow”. The 2nd International Workshop on Automated and Algorithmic Debugging, St. Malo, France, pages 1-41.

[11] Korel B. (1997), “Computation of dynamic slices for unstructured programs”. IEEE Transactions on Software Engineering, pages 17-34.

[12] Korel B. and Rilling J. (1997), “Dynamic Program Slicing in Understanding of Program Execution”. Proceedings of the 5th International Workshop on Program Comprehension, pages 80-90.

[13] Korel B. and Rilling J. (1997), “Application of Dynamic Slicing in Program Debugging”. Third International Workshop on Automated Debugging.

[14] Shimomura T. (1992), “The program slicing technique and its application to testing, debugging and maintenance”. Journal of IPS of Japan, pages 1078-1086.

[15] Tip F. (1995), “A survey of program slicing techniques. Journal of Programming Languages”, pages 121-189.