Volume 4, Issue 4, April 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Slicing of Object-Oriented JDBC Application Program

Khushbu Gupta*, Abhishek Ray
School Of Computer Engineering
KIIT university Bhubaneswar, India

Abstract— Debugging an object-oriented programming software is always challenging as it has to handle two aspects, the program state and data state. In this paper, we have proposed an algorithm for an object-oriented software that handles the dependencies which arises in both program state and data state. We have used Graph Coloring approach and extended an existing algorithm called Contradictory Graph Coloring Algorithm for computing dynamic slice of object-oriented JDBC application. We have named our algorithm as Extended Contradictory Graph Coloring Algorithm (ECGCA).

Keywords— Program slicing; database slicing; dynamic slicing; object-oriented programs; JDBC application; Extended Contradictory Graph Coloring Algorithm (ECGCA).

I. INTRODUCTION

Program slicing is a technique which has many applications in various software engineering activities such as program debugging, testing, maintenance etc. It is a practical disintegration process that excludes program modules that are insignificant to a particular computation process based on a criterion known as the slicing criterion. The slicing criterion <s, v> is a pair where s is the program point of interest or location of statement and v is the set of variables which are used or defined at s. A slice <s, v> is a collection of executable statements which are extracted from program p that are relevant to the program point of interest. Majority of research on slicing has been done on languages like C, C++, in which only the program state has been considered.

The full behaviour of program is defined in two forms program state and database state [8]. Debugging an object-oriented programming software is always challenging as it has to handle two aspects, the OOP state and data state. To debug software, slicing can be a good & established technique. Many researchers [10, 12, 15, 16, 14, 9, 3, 33, 32, 6, 28, 26, 11, 13, 23] have worked on the slicing of object-oriented programs and some work has also been found in slicing of query languages [19, 8, 4]. To the best of our knowledge no algorithm is available which addresses slicing of OOP which supports query handling in the database. Our objective is to develop an algorithm to compute the dynamic slice of JDBC application program using proper intermediate representation.

Organization of The Paper is, section 2 provides the basic concepts used in rest of the paper. Here we have discussed some definitions and described some intermediate program representation concepts, which are used in slicing techniques and used later in our algorithms. Section 3 provides a brief literature review. Section 4 provides our proposed work which follows intermediate representation of the JDBC program by System Dependence Graph and then proposed algorithm. Section 6 presents the concluding remarks, with scope for further research work.

II. BASIC CONCEPTS

A. *Dynamic slicing*

A dynamic slice can be constructed with respect to only one execution of any program. The statements that have no significance with the slicing criteria on some specific input is not included in this type of slicing. Due to the run-time handling of arrays and pointer variables, dynamic slicing algorithms are easier than static algorithm and result in more exact slices [6]. Dynamic slice is execution specific and relatively smaller, so it is helpful in debugging. The difference between static and dynamic slicing is that dynamic slicing assumes fixed input for a program, whereas static slicing does not make assumptions regarding the input. A dynamic slicing criterion specifies the input, and distinguishes between different occurrences of a statement in the execution history; typically, it consists of triple input occurrence of a statement variable.

B. *Introduction to database slicing*

Database Slicing is mainly done with the help of PDG i.e Program Dependence Graph. Slicing in databases is considered as a big issue when managing a large database becomes difficult. These kind of constraints lets us go in the area where we can find more efficient way of slicing those databases which find difficult to extract the significant information.

In the traditional Program Dependence Graph of any program P, if a slice of program P has to be found out, it should contain all nodes of the PDG from which a statement s should be reachable by control dependency or data dependency. The algorithm which uses this PDG fails to find slice when it has to deal with those programs which access database state. Hence, this can be said that the control dependencies are appropriate but when it comes to the data dependencies, the Program Dependence Graph considers only1form of state, i.e. program state. Thus, PDG is considered as partially

complete since program consists of two types of state, one state is program state and another state is database state. Thus, to figure out appropriate slices over database state, the PDG has to be extended to consider the absent dependencies. The first are those caused due to the connection between program state and database state. The other are those dependencies which occur between the pairs of statements that both can manipulate the database state.

If Program Dependence Graph is extended with these two types of dependencies, more accurate slices are computed for both kinds of state. So, for this reason DOPDGs i.e. Database Oriented Program Dependence Graph are been introduced [8]. These concepts are defined in following section.

C. ProgramDatabase (PD) Dependencies

It is a kind of data dependency in which the interaction between the program state and database state result into these dependencies [8]. This can be defined as follows:

A database statement *d* is programdatabase dependent upon the statement *s*, when there is some variable *v* such that:

- The variable *v* is provided as an input to the database statement *d*,
- The variable *v* is defined by the statement *s*
- There should be a *v*-definition free path from statement s to the database statement *d*.

Thus, the set of all these types of dependencies that are calculated for a particular program is known as programdatabase dependency graph (PDDG).

D. DatabaseDatabase (DD) Dependencies

This type of data dependency occurs when those pair of statements which both can manipulate database state are interacted with each other. It occupy that situation, when the one database statement's execution is affecting the behaviour of other database statement which is executed after the first one. This dependency is significant in calculating the appropriate slice for a program variable which is involved in a database command, and they are also useful for calculating this type of slice [8].

III. RELATED WORK

In 2004, Willmor et al. [8] have shown that existing algorithms don't consider some of the semantics of a database state, and hence proposed two more forms of dependency. These new forms of dependency help the slicer to consider the program's nature that is taking place fully within the database state (DD-dependencies) and also the interrelation between the program state and the database state (PD-dependencies). For this, they have used Database oriented Program Dependence Graph (DOPDG), which is an extension of PDG, in which additional dependences have been taken into consideration which were a kind of drawback in traditional PDG.

In 2013, Halder et al. [19] defined syntax-based DOPDG by expressing the conditions of DD and PD-Dependences in denotational form. They also refined syntax-based DOPDGs into semantics-based abstract DOPDGs by combining the notions of

- semantic relevancy of statements,
- semantic data dependence, and
- conditional dependences .

These contributions lead to an abstract program slicing algorithm for programs embedding SQL statements that strictly improves with respect to the literature.

In 1988, Laski et al. [6] have introduced dynamic slicing. In their work they said that slice depends upon input data and generated at the time of execution which is different from the static slicing [30] and therefore it is called as dynamic slicing. The main objective of dynamic slicing is similar to that of static slicing, that is to look for those statements which are responsible for program error. Then in 1990, Agrawal et al. [3] used Program Dependence Graph to represent algorithm to find dynamic slices. They have proposed a proper approach based on the graph called Dynamic Dependence Graph (DDG) which they have used as an intermediate representation. Zhao et al. in 1998 [33] extended the DDG of Agrawal [3] and represented different dynamic dependencies between statement instances for a specific execution of program and hence they named their graph as dynamic object-oriented dependence graph (DODG). Song [26] have developed a method which computes forward slice of an object-oriented program dynamically. Their method uses Dynamic Object Relationship Diagram (DORD) as representation [26]. Xu et al. [32] has used object program dependence graph (OPDG) to dynamically slice object-oriented programs. Wang et al. [28] proposed an algorithm which operates on compact byte code traces for Java programs and slices the program dynamically. Mohapatra et al. [23] have proposed a novel approach to slice object-oriented programs dynamically. They have presented an edge marking dynamic slicing (EMDS) algorithm and used Extended System Dependence Graph (ESDG) as an intermediate representation which works on marking and unmarking of edges during execution. Later, Mohapatra et al. [13] again proposed another algorithm which also works on the same concept of marking and unmarking but instead of edges they have made use of nodes to dynamically compute slice. The algorithm proposed here was called Node Marking Dynamic Slicing (NMDS) algorithm.

Barpanda et al [4] in his thesis work proposed a novel approach for object-oriented programs slicing through graph coloring technique. He has proposed an algorithm known as Contradictory graph colouring algorithm (CGCA).

IV. PROPOSED WORK

In the previous work of object-oriented program slicing, the algorithms proposed by different authors for computing slices of object-oriented programs are not concerned about the background processing of data that means these slicing algorithms are not handling the instructions involved in processing of the backend database. With this motivation, we propose a dynamic slicing algorithm, which is an extension of an existing algorithm Contradictory Graph Coloring

Algorithm (CGCA) for object-oriented programs [4]. We first statically construct the system dependence graph (SDG) [4] as the intermediate representation of the JDBC application program. Then graph coloring technique is used on SDG to compute the slice of an object-oriented program. Graph coloring is a method which is used for coloring the nodes so that, two nodes which shares the same edge will not have the same color. This is referred as node coloring problem [4]. Thus, according to the definition, the coloring of a graph G = (N,E) is a mapping c:N → C), where C is a set of colors, such that if (N,X) ∈ E, then c(N) ≠ c(X) [4]. The minimum number of colors needed to color a graph is known as chromatic number of graph, which is symbolically χ(G) [34].

The algorithm depends upon the following assumptions :

- ✓ Erase the condition that " two vertices sharing the same edge will not have the same color".
- ✓ The chromatic number of the graph, i.e. χ(G)=1.

The primary cause for which we have contradicted the constraints of graph coloring algorithm is that the statements that corresponds to two nodes which shares the same edge or two edges which shares the same node can be present in the slice. Thus, only one color is sufficient to color the nodes instead of using multiple color. The concept used for coloring the nodes which are in slice produced is State Restriction and is defined as follows.

State restriction, ($\restriction$) :- Universal view of state restriction given by Binkley et al. [5] is as follows:

Let σ is the state & V is set of variables, then $\sigma \restriction$ V restricts σ such that it is defined only for variables of V.

$$(\sigma \restriction V) = \begin{cases} \sigma x & if\ x \in V \\ \perp & otherwise \end{cases} \quad \textbf{(1)}$$

Since, the dynamic slice of an object-oriented program is to be computed, the issues has to be considered that may be occured due to control dependency or data dependency. Thus, to occupy the appropriate state restriction, original definition [5] have been modified in [4] as:

"Let 'σ' is the given state and V be the set of variables then 'σ' is state restricted by V if there exists either control dependence or data dependence". In the notation of the theory of program projection, it may be written as:

$$(\sigma \restriction V)cd \| dd = \begin{cases} \sigma x & if\ x \in V \\ \perp & otherwise \end{cases} \quad \textbf{(2)}$$

Keeping PD-dependencies in consideration in the slicing of an object-oriented program which handles database, original definition [5] of state restriction have been modified as:

"If 'σ' be the given state and V be the set of variables then 'σ' is state restricted by V when there exists either programdatabase dependence". In the notation of the theory of program projection, it may be written as:

$$(\sigma \restriction V)pd = \begin{cases} \sigma x & if\ x \in V \\ \perp & otherwise \end{cases} \quad \textbf{(3)}$$

V. INTERMEDIATE REPRESENTATION

The system dependence graph (SDG) of the program is used as the intermediate representation. In Fig. 1, we have shown the example program which is written in java and performs the operation of inserting records in the database table. The system dependence graph (SDG) is constructed statically and it consists of several elements where the circles or nodes represent the statements of the program and the edges represent the dependencies between the nodes. The program performs the function of inserting record into the database table using PreparedStatement interface. Use of PreparedStatement interface in the JDBC application is that the DB engine will prepare a query plan for the SQL command only once; the same prepared query plan will be executed repeatedly at db so that the overhead at the db is reduced. This can help us in efficient working of the algorithm. Since, the slice of an object-oriented program for a JDBC application is to be computed dynamically, the issues has to be considered that can be occured due to control dependency (cd) or data dependency (dd) or programdatabase dependency (pd).

Using the concept of state restriction ECGC Algorithm is presented as follows:

When a node is figured out to state restrict the slicing criterion depending upon the occurrence of the type of dependency mentioned, then it is colored,

Extended Contradictory Graph Coloring Algorithm

Input: SDG with set of nodes ($n_1, n_2, n_3, \ldots, n_m$), n_c be the node which represent the slicing criterion , current state is σ of slicing criterion, V($x_1, x_2, \ldots, x_j$) is the set of variables.

Output: Colored nodes representing the slice on the SDG.

- Draw SDG once.
- Set the value χ(SDG)=1 since we make use of only one color for computing the slice.
- Traverse backward through SDG

while (traNode ≤ n_m)
{
if (($\sigma \restriction x_j)_{cd}$ for n_r) , then
color node nr which is reached from nc while traversal
else if (($\sigma \restriction x_j)_{dd}$ for all $x_j \in$ V), then
{

if reached node n_r is colored, then
go to Stmt1
else
color reached node n_r
Stmt1: traNode = +1;
}
else if($(\sigma \uparrow x_j)_{pd} \forall x_j \in V$) , then
{
if reached node n_r is colored, then go to Stmt2
else
color reached node n_r.
Stmt2: traNode = +1;
}
}

- Capture the nodes those are colored at the time of traversal of graph showing the slice.

```
import java.sql.*;
import java.io.*;
public class PreparedInserEx {
1.public static void main(String[] args) {
        2.boolean repeat = true;
        3.while(repeat)          {
        4.try    {
        5. DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
         6. Connection con=DriverManager.getConnection("jdbc:odbc:oradsn","S YS TEM","khushi");
         7.PreparedStatement pstmt = con.prepareStatement("insert into emp values(?,?,?)");
        8. DataInputStream stdin = new DataInputStream(System.in);
        9. while(true)    {
                10.  System.out.println("Enter Employee No.");
                11. int eno = Integer.parseInt(stdin.readLine());
                12. System.out.println("Enter Employee Name");
                13. String ename = stdin.readLine();
                14. System.out.println("Enter Employee salary");
                15. Float sal = Float.parseFloat(stdin.readLine());
                    16.  pstmt.setInt(1,eno);
                    17. pstmt.setString(2,ename);
                    18. pstmt.setFloat(3,sal);
                    19. int count = pstmt.executeUpdate();
                        20.      if (count>0) {
                        21.              System.out.println("Record Insert Successfully");
                                 }
                        22.      else     {
                        23.              System.out.println("Record Insertion Failed");
                                 }
                        24.      System.out.println("Do you want to insert another  record(YES/NO)");
                        25.      String choice = stdin.readLine();
                        26.      if(!choice.equalsIgnoreCase("yes"))
                        27.              break;
                }
                        28.     repeat = false;
}
                29.  catch(Exception e)          {
                        30. System.out.println("Exception Raised Trying Again");
                                 }       }       }
```

Figure 1 : An example program : Program P

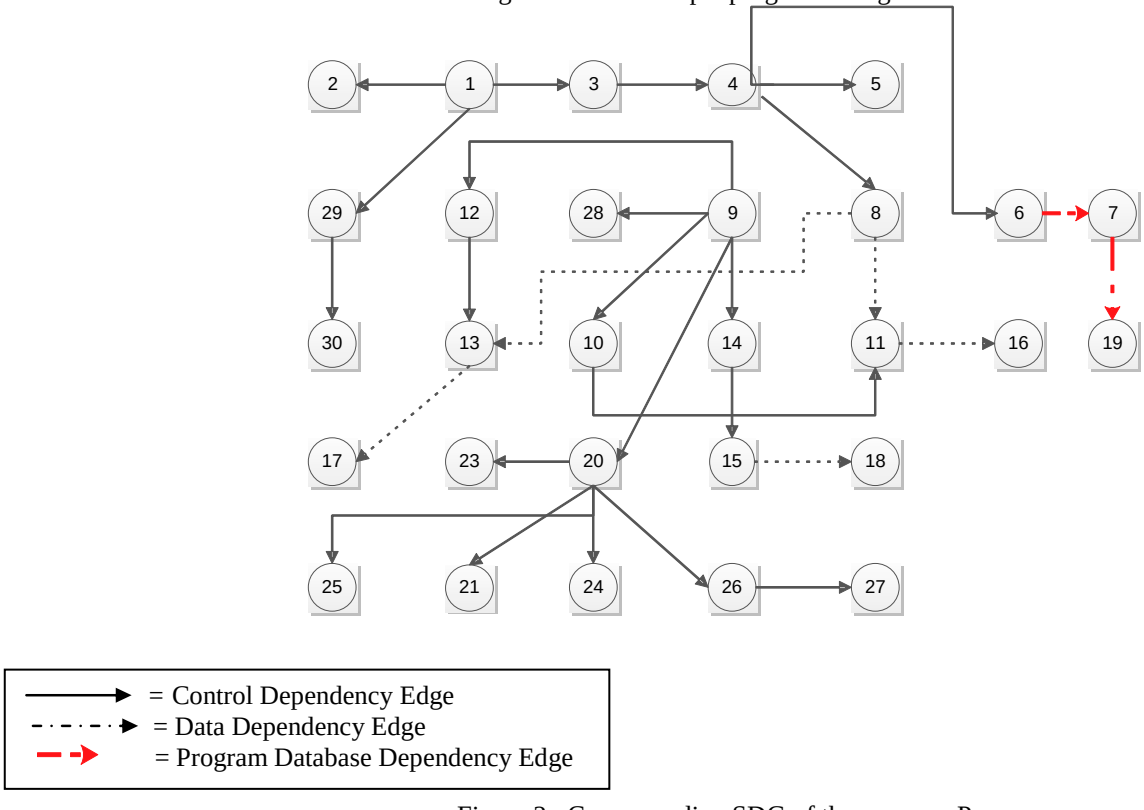

Figure 2 : Corresponding SDG of the program P

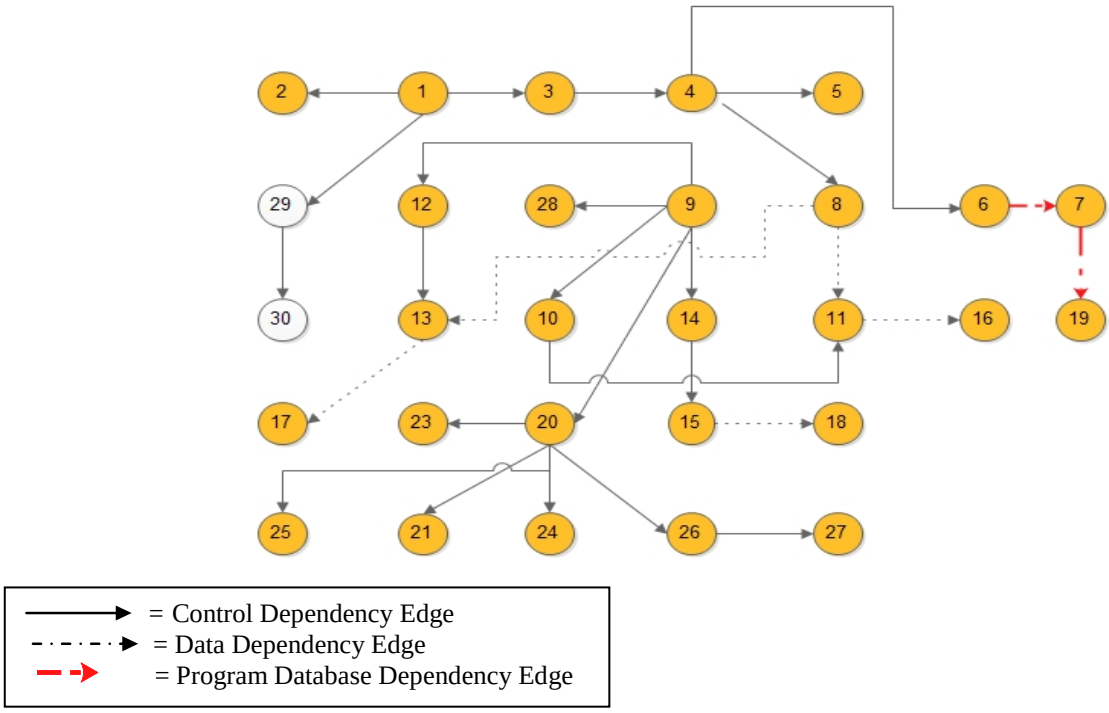

Figure 3 : Updated SDG of the program P, colored nodes showing the slice.

A. Working of algorithm :

In this section we discuss the working of the proposed algorithm. Fig. 1 shows a sample program which depicts insertion of user defined records using PreparedStatement interface into the database table. For the given program, we first construct the SDG. Fig. 2 shows the SDG for the sample program shown in Fig. 1. During the execution of the program, SDG will be updated. In a SDG the node which represents slicing point is considered as the first node for traversing and if the node is found to contain any type of dependency or state restrict the slicing criterion then it is colored. The procedure is continued for every iteration of while loop till each & every node is traversed. As stated in the algorithm, if any node is found to be already colored then it need not be colored again even if there is state restriction with the criterion. Fig. 3 shows an updated graph in which colored nodes represent the slice of the program shown in Fig.1 with respect to the slicing criterion <20,count> and input value count>1.

VI. CONCLUSIONS

In this paper we have proposed an algorithm named Extended Contradictory Graph Coloring Algorithm for computing dynamic slices of object-oriented JDBC application program. Our approach is an extension of Contradictory Graph Coloring Algorithm. Firstly, we have proposed an intermediate representation in the form of a graph to represent the object-oriented JDBC application program. Then, we have proposed ECGCA algorithm to compute the dynamic slices of object-oriented JDBC application. Currently we have not tested our proposed algorithm. Further we would like to implement the algorithm by developing a slicing tool.

REFERENCES

[1] Program slicing. http://www.shannarasite.org/project/ProgramSlicing/www.reportse2.html.
[2] A. Goel, A. R. C., and Mitra, T. Compactly representing parallel program executions. p. 191-202.
[3] Agrawal, H., and Horgan, J. Dynamic program slicing. In Proceedings of the ACM SIGPLAN90 Conference on Programmimg Languages Design and Implementation, SIGPLAN Notices, Analysis and Veri_cation, volume 25 (1990), pp. 246-256.
[4] Barpanda, S. S. A graph coloring approach to dynamic slicing of object-oriented programs, july 2010.
[5] Binkley, D., D. S. G. T. H. M. K. A., and Korel, B. A formalisation of the relationship between forms of program slicing. p. 228-252.
[6] B.Korel, and J.Laski. Dynamic program slicing. In . Information Processing Letters 29 (1988).
[7] Canfora, G., C. A., and A., L. Conditioned program slicing. pp. 595-607.
[8] D Willmor, S. M. E., and Shao, J. Program slicing in the presence of a database state. In Proceedings of the 20th International Conference on Software Maintenance (2004), pp. 448-452.
[9] Durga Prasad Mohapatra, Rajib Mall, R. K. An overview of slicing techniques for object-oriented programs. pp. 253-277.
[10] Durga Prasad Mohapatra, R. M., and Kumar, R. Dynamic slicing of concurrent object oriented programs. p. 283-290.

[11] Durga Prasad Mohapatra, R. M., and Kumar, R. An edge marking dynamic slicing technique for object-oriented programs. p. 60-65.
[12] Durga Prasad Mohapatra, R. M., and Kumar, R. An efficient technique for dynamic slicing of concurrent java programs. p. 255-262.
[13] Durga Prasad Mohapatra, R. M., and Kumar, R. A node marking dynamic slicing technique for object-oriented programs. p. 1 15.
[14] Durga Prasad Mohapatra, R. M., and Kumar, R. A novel approach for dynamic slicing of distributed object-oriented programs. p. 304-309.
[15] Durga Prasad Mohapatra, R. M., and Kumar, R. A novel method for computing dynamic slices of concurrent c++ programs. p. 744-750.
[16] Durga Prasad Mohapatra, R. M., and Kumar, R. Computing dynamic slices of concurrent object-oriented programs.
[17] F. Ohata, K. Hirose, M. F., and Inoue, K. A slicing method for object-oriented programs using dynamic light weight information.
[18] Gupta, R., and Soffa, M. L. A framework for generalized slicing. Tech. rep., University of Pittsburgh, 1992.
[19] Halder, R., and Cortesi, A. Abstract program slicing of database query languages. In Proceedings of the the 28th Symposium On Applied Computing, pp. 838-845.
[20] Harrold, M. J., and Rothermel, G. Syntax-directed construction of program dependence graphs. Tech. rep., May 1996.
[21] Horwitz, S., R. T., and Binkley, D. Interprocedural slicing using dependence graphs. p. 26-61.
[22] J.Rilling, and B.Karanth. A hybrid program slicing framework. pp. 12-23.
[23] Mohapatra, D. P. Dynamic slicing of object oriented programs. Master's thesis, Indian Institute of Technology, Kharagpur, India, 2005.
[24] Mund.G., M., and S.Sarkar. Computation of intra-procedural dynamic program slices. pp. 499-512.
[25] Ottenstein, K. J., and Ottenstein, L. M. The program dependence graph in a software development environment. pp. 177-184.
[26] Song, Y., and Huynh, D. Forward dynamic object-oriented program slicing.
[27] Tip, F. A survey of program slicing techniques. Tech. rep., 1994.
[28] Wang, T., and Choudhury, A. R. Using compressed bytecode traces for slicing java programs. p. 512-521.
[29] Weiser, M. Program slicing : formal, psychological, and practical investigations of an automatic program abstraction method, 1979.
[30] Weiser, M. Programmers use slices when debugging. p. 446 -452.
[31] Weiser, M. Programmers use slices with debugging. pp. 446{452.
[32] Xu, B., and Chen, Z. Dynamic slicing object-oriented programs for debugging. p. 115-122.
[33] Zhao, J. Dynamic slicing of object-oriented programs. Tech. rep., Information Processing Society of Japan, may 1998.
[34] Deo, N., *Graph Theory With Applications to Engineering and Computer Science*. Prentice-Hall, 1974.

