Static Slicing Technique with Algorithmic Approach

Sonam Agarwal¹, ArunPrakash Agarwal²
¹(M.Tech CSE, Amity University, Noida, India)
²(Assistant Professor, Amity University, Noida, India)

Abstract: In order to improve the accuracy of the static program slicing which is used to locate the faults that cause an exception, we propose a new algorithm which is stored when an exception occurred. The proposed approach consists of various steps and figure out those methods and statements that have not been executed. Then, these methods which are not executed will be ignored when building the system dependence graph. Finally, the accurate slice will be got by adopting the improved program slicing algorithm. And the result shows that using our approach the slice is 8 percent less than using the general static program slicing algorithm on average. One approach to improve the comprehension of programs is to reduce the amount of data to be observed and inspected. Programmers tend to focus on comprehended selected functions (outputs) and those parts of a program that are directly related to that particular function rather than all possible program functions. One approach is to utilize program slicing, a program decomposition technique that transforms a large program into a smaller one that contains only statements relevant to the computation of a selected function.

Keywords: Program slicing, Static slicing, Static slicing algorithm

I. Introduction

Program slicing is a decomposition technique to solve program analysis problem. Program slicing is used for slicing the large program into small and simple programs that contain only that statement that is necessary to complete any execution or execute any program without any error. It is widely used for program analysis, understanding, testing, debugging, metrics etc. The concept and principle of program slicing were first established by M. Weiser [6] in 1979, and after that many other slicing techniques has been implemented. Program slicing is a feasible method to restrict the focus of a task to specific sub-components of a program. The program slicing can be classified as static slicing and dynamic slicing. For a specific slicing criterion, the main difference between them is that the information they required. Static program slicing only needs static information which is the source code, while the other requires the entire execution trace which is corresponding to a specific program input. In other words, static slicing considers all possible executions but dynamic slicing only considers the execution of a specific input [16]. Therefore, dynamic slicing contains fewer statements and is more precise than the static one. The optimized dynamic slicing will take several minutes to perform the slicing and the preparing time is even longer [2]. With complex program and long execution traces, dynamic slicing is not appropriate.

Applying slicing technique to software architectures can benefit software development in two main ways [18]. The first one concerns maintenance of a component-based software. By using slicing tools on an architectural description, we can determine which components might be affected when a given component is modified. Second, architectural reuse can be facilitated. While reuse of code is important, reuse of software design and patterns are expected to offer greater productivity benefits and reliability enhancements [22].

II. Paper Work

In this paper we firstly write the simple program along with the sliced program. Sliced program is also called the simple slicing or static slicing of a program.

(a) Void main()
(b) {
(c) int i=0;
(d) int n;
(e) int sum=0;
(f) printf(“enter the value of n”);
(g) scanf("%d", &n);
(h) for(i=0; i<n; i++)
(i) {
(j) sum=sum+i;
(k) printf("the current value of i is", i);
(l) }
(m) printf("the sum is",sum);
(n) getch();
(o) }

Program: Simple Program

This program consists of three variables. Sohere we have three variables which can remove. Now we describe all the three conditions that satisfy the criteria[20]. After removing one by one each variable which program remains that is called sliced program or static sliced program.

Condition 1: Remove variable i
(a) Void main()
(b) {
(c)
(d) int n;
(e) int sum=0;
(f) printf("enter the value of n");
(g) scanf("%d",&n);
(h)
(i)
(j)
(k)
(l)
(m) printf("the sum is",sum);
(n) getch();
(o) }

Program: Static Sliced Program

Condition 2: Remove variable n
(a) Void main()
(b) {
(c)
(d)
(e)
(f)
(g)
(h)
(i) {
(j) sum=sum+i;
(k) printf("the current value of i is",i);
(l) }
(m) printf("the sum is",sum);
(n) getch();
(o) }

Program: Static Sliced Program

Condition 3: Remove variable sum
(a) Void main()
(b) {
(c)
(d)
(e)
(f) printf("enter the value of n");
(g) scanf("%d",&n);
(h) for(i=0; i<n; i++)
(i) {
(j)
(k) printf("the current value of i is",i);
Program: Static Sliced Program

Now after understanding the static slicing concept we describe a static slicing algorithm which is clearly understandable and gives the solution of simple program to the static sliced program. This algorithm is very simple and gives the precise solution of any problem. In order to extract a slice from a program, the dependencies between the statements must be computed first. The control flow graph (CFG) is a data structure which makes the control dependencies for each operation in a program explicit [3][4].

2.1 Algorithm

P: Simple Program
Ch: variable which we want to remove

```
Static_Slice(p,ch)
{
    int l;
    str s1="":
    1. Open program P in read mode
    2. Fetch the input variable Ch
    3. Find l i.e. length of program P
    4. while(!EOF)
        5. {
            6. for(i=0;i<l;i++)
                7. {
                    8. if(Ch==getc(p))
                        9. {
                            10. Delete the variable;
                            11. }
                        12. else
                            13. {
                                14. s1=s1+Ch;
                                15. }
                            16. }
                        17. }
        5. }
}

III. Conclusion

This paper presents general program slicing algorithms. The algorithms compute correct program slices for all language constructs found in major object-oriented programming languages, e.g., polymorphism, inheritance, late binding, exception handling, local and global variables. This paper provides the general static slicing algorithm computes correct and executable static slices. Static slicing methods have been proposed for maintenance and program understanding because this way certain parts of the program can be “sliced away” that are of no interest with respect to the slicing criterion.

The application of slicing in various areas like debugging, cohesion measurement, comprehension, maintenance and re-engineering and testing are highlighted. In this paper, we have proposed and validated a technique, which is an extension of an existing technique proposed by us in an analogous study. On carefully analyzing the code the behavior of techniques we observed that proposed technique gives better result.

IV. Future Scope

As part of our future work, we will expand our framework to extend the algorithmic support within our structure and derive new slicing related concepts, as well as new visualization techniques. We also plan to integrate forward slicing algorithm and backward slicing algorithm within our static slicing framework.

In future we plan to make a more robust implementation of the algorithms and to perform some measurements on real, industrial applications with real case studies. For this purpose we need to make the implementation more stable and improve the performance, especially regarding the slowdown of the instrumented code. Another on-going work is to implement the algorithm for the C++ language.
Static Slicing Technique With Algorithmic Approach

References