International Journal of Electrical Electronics & Computer Science Engineering
Special Issue - AET 2017 | E-ISSN : 2348-2273 | P-ISSN : 2454-1222
Available Online at www.ijeecse.com

Variants of Program Slicing and Its Applications

P. A. Tijare, Dr. P. R. Deshmukh
Computer Science & Engineering, Sipna College of Engineering & Technology, Amravati, India
pritishtijare@rediffmail.com, pr_deshmukh@yahoo.com

Abstract: ***Slicing is a procedure for simplifying characterization to a slicing criterion. The thought behind all methods to program slicing is to create the least complex program possible conceivable that keeps up the significance of the first program concerning the original program. In this paper we have introduced various forms of slicing criteria: static, dynamic, conditioned and amorphous slicing along with number of parameters that are to be considered while slicing of a program and discussed its few applications.***

Keywords: Slicing, static, dynamic, conditioned, amorphous.

I. INTRODUCTION

Program Slicing is a program analysis technique through which various program slicing applications including software engineering activities such as understanding of a program, debugging, testing, optimization, maintenance etc. are carried out. Program slicing is a strategy to limit the focus of a task to particular part of a program. It may also be utilized to find out the statements of a program that are significant to a given computation. Program slices can be computed automatically by statically analyzing the data and control flow of the program.

Program slicing was firstly introduced by Weiser as a first program analysis technique [1]. Depending upon the slicing criteria, program slicing slices the program into number of slices without disturbing the behavior of the program.

There are various types of slicing have been proposed in the literature. Some of them are static slicing, dynamic slicing [4], quasi static slicing [3], and simultaneous dynamic slicing [5], object oriented slicing and conditioned slicing [6].

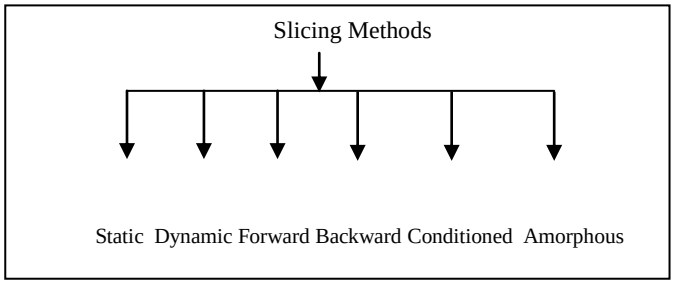

Figure 1: Various Types of Slicing Methods

There are various parameters that are to be considered while slicing of a program.

Table 1: Parameters for Program Slicing

Parameter	Meaning
Slicing Point	Point where the programmer is interested repeat the number of the time till the logic of the program is complete
Slicing Variable	Variable mentioned in the slicing criteria
Scope	Intra procedural or inter procedural
Slicing Direction	Forward or Backward
Abstraction Level	Procedure or Statement
Information Type	Static or Dynamic
Result	Equivalent to the program of few set of statements from the program

II. STATIC SLICING

The basic form of slicing is static slicing. It is created by avoiding those parts of the program which are irrelevant to the values present in the set of variables. The slices are computed according to Data Flow Dependence and Control Flow Dependences [2]. The static slices are calculated with respect to the slicing criteria. By considering the slicing criteria two type of slices can be constructed. Forward Slice and Backward Slice

Forward Slice:

Forward Slice in which those statements which are acting as a sub set of the program which are affected by the slicing criteria are considered. For computation of a forward slice of a program from a point P, forward reachability in the program dependence graph from node P is calculated.

Backward Slice:

In backward slice the subset of instruction that contribute to value of variables in criterion is considered. To compute a backward slice of a program from a point P,

backward reachability in the program dependence graph from node P is calculated. The technique used by Mark Weiser for slicing is known as Backward Slicing.

Working of Slicing

Consider C language code as shown in Figure 2. Assume we just think about the impact this part of code has on the variable c toward the finish of the program. We can build a regressive cut on c toward the finish of the program to concentrate consideration on this part. The slice of Figure 2 on c at the end of the program is shown in Figure 3. The line e = a; is not influencing the final value of c and this is the reason it is excluded in the slice. Somewhat more subtle is the way that the task c = b*2- a; likewise has no impact upon the last estimation of c, thus it too is excluded in the slice. Although this statement updates the value of c in the middle of the program, the value assigned is overridden by the last line, hence no effect of the first assignment.

```
a = 10;
b = 20;
c = b*2-a;
e = a;
c = a+b;
/* the slice point is the end of the program */
```

Figure 2: A Program Fragment to be Backward Sliced

```
a = 10;
b = 20;
c = a+b;
```

Figure 3: The Backward Slice of Figure 2

Consider part of the program in Figure 4. This program is computing the sum and the product of numbers from 1 to *n*. The sum is stored in the variable *add* and the product in the variable *prod*. The computation in *add* is correct, but that on *prod* is incorrect. Since the computation of *add* is correct we are not looking over there which only affects this variable, but to track the fault which causes the erroneous computation on *prod*. Slicing on *prod* is a starting point for debugging, because it removes computations which can be shown (statically) to leave *prod* unaffected.

The static slice on the final value of the variable *prod* is shown in Figure 5. As the slice is simpler than the original program in Figure 4, yet contains all the statements which could possibly affect the final (and incorrect) value of the variable *prod*, examining the slice will allow us to find the bug faster than examining the original. In this case, the bug lies in the initialization of the variable *prod*. The variable should be initialized to 1 and not to 0.

```
scanf("%d",&num);
add=0;
prod=0;
while (num>0)
{
add=add+num;
prod=prod*num;
num=num-1;
}
printf("%d%d",prod,add); }

/* the slice point is the end of the program */
```

Figure 4: A Program Fragment to be Sliced

```
scanf("%d",&num);
prod=0;
while (num>0)
{
prod=prod*num;
num=num-1;
}
```

```
prod=0;
```

Figure 6: A Dynamic Slice of Figure 4

III. DYNAMIC SLICING

Dynamic slicing concept is introduced by using the concept of Control Flow Graph. After debugging, when we executed program, presumably found that it produced an unexpected value. For example, the program in figure 4 might have been executed with the input value 0 for the variable *num*. Now, when we are executing the program in figure 4 with the value 0 we will find that the variable *prod* contains the wrong value. Instead of the static slice to locate the cause of this bug, it would make more sense to construct a slice which *exploited* the information available about the input which caused the program to go wrong. Such type of slice is called a dynamic slice [7]. We can build a dynamic slice for the variable *prod* towards the finish of the program on the input sequence 0. The dynamic slice developed with respect to this criterion is shown in Figure 6.

IV. CONDITIONED SLICING

Static and dynamic slicing represents two ends – either all about the input (dynamic) or noting about the input (static). So conditioned slicing can be use to bridge the

gap between two. Any slice that is developed with respect to condition, such approach to slicing is called the conditioned slicing [6,8]. The slices are conditioned by information about the condition through which the program is to be executed. Conditioned slicing addresses represents the type of problems maintainers face when task of understanding large legacy systems

V. AMORPHOUS SLICING

All above methods for slicing are syntax protecting. They are developed by transforming deletion of statements. All approaches to slicing discussed till now are `syntax preserving'. Therefore, those statements which remain in the slice are a syntactic subset of the original program from which the slice was build. In opposite, amorphous slices [9,10] are constructed using *any* program transformation which simplifies the program and maintains the impact of the program with respect to the slicing criterion. This syntactic opportunity allows amorphous slicing to perform greater simplification with the result that amorphous slices are never larger than their syntax-preserving counterparts. Frequently they are considerably smaller.

VI. INTERPRETATIONS

Method	Interpretation
Static Slicing	There is no effective role of the condition part of the criteria in the simplification process
Dynamic Slicing	The condition which defines values for each input to the program, is a conjunction of equalities
Forward Slice	Developed by removing statements which cannot be *affected by* the slicing criterion
Backward Slice	Developed by removing those parts of the program which have no *effect upon* the slicing criterion.
Conditioned Slice	Provides a suitable bridge between the two ends of static and dynamic slicing.
Amorphous Slice	Developed using any program transformation which simplifies the program and which maintain the effect of the program with respect to the slicing criterion.

VII. APPLICATIONS OF PROGRAM SLICING

Some of the applications of program slicing are:

Load Balancing

Program slicing can be utilized to break down a traditional program into significantly independent slices to assign to separate processors to achieve load balancing in parallel programming. [11].

Debugging

Finding a bug can be a troublesome undertaking when one is gone up against with a vast program. In such cases, program slicing is valuable since it can enable one to ignore may statements while endeavoring to confine the bug. In the event that a program processes a wrong an incentive for a variable. If a program computes an erroneous value for any variable, only those statements in its slice would contain the bug; all statements not in the slice can safely be ignored.

Software Maintenance and Re-engineering

Programming support is frequently trailed by re-designing efforts, whereby a framework is controlled to enhance it. Canfora at al. [12] acquainted conditioned all together with enable conditions to be utilized to extract code based on conditions. The thought is to isolate desired functionality so that it can be retrieving and reused.

VIII. CONCLUSION

This paper introduces four types of slicing criteria namely static, dynamic, conditioned and amorphous slicing. Program slicing can be utilized to break down a traditional program into independent sub parts to assign to separate processors to achieve better performance in load balancing in parallel programming. Also, we have discussed number of parameters that are to be considered while slicing of a program. This paper also focuses few applications of slicing.

IX. REFERENCES

[1] Mark Weiser, “Program slicing”, In Proceedings of the International Conference on Software Engineering (ICSE’81), pages 439–449, 1981.

[2] N.Sasirekha, A.Edwin Robert and Dr.M.Hemalatha, “Program Slicing Techniques And Its Applications”, International Journal of Software Engineering & Applications (IJSEA), Vol.2, No.3, July 2011 DOI : 10.5121 /ijsea.2011. 2304 50.

[3] G.A. Venkatesh, “The semantic approach to program slicing”, ACM SIGPLAN Notices, vol. 26, no. 6, 1991, pp. 107-119.

[4] B. Korel and J. Laski, (1990), "Dynamic slicing of computer programs", The Journal of Systems and Software, vol. 13, no. 3, pp. 187-195.

[5] R.J. Hall,(1995), "Automatic extraction of executable program subsets by simultaneous program slicing", Journal of Automated Software Engineering, vol. 2, no. 1, pp. 33-53.

[6] G. Canfora, A. Cimitile, and A. De Lucia,(1998), "Conditioned program slicing", Information and Software Technology, vol. 40, no. 11/12, pp. 595-607.

[7] B. Korel and J. Rilling. Dynamic program slicing methods. Information and Software Technology special issue on Program Slicing, 40(11-12), pages 647-659, December 1998.

[8] S. Danicic, C. J. Fox, M. Harman and R. M. Hierons. ConSIT: A Conditioned Program Slicer. IEEE International Conference on Software Maintenance (ICSM 2000), San Jose, California, USA, October, 2000. pages 216-226.

[9] M. Harman and S. Danicic. Amorphous Program Slicing. IEEE International Workshop on Program Comprehension (IWPC'97), Dearborn, Michigan, May 1997, pages 70-79.

[10] D. W. Binkley, M. Harman, L. R. Raszewski and C. Smith. An empirical study of amorphous slicing as a program comprehension support tool. 8th IEEE International Workshop on Program Comprehension (IWPC 2000), Limerick, Ireland, June, 2000. Pages 161-170.

[11] M.Weiser. 1983.Reconstructing sequential behavior from parallel behavior projections. Information Processing Letters 17(10):129–35.

[12] G. Canfora, A. Cimitile, A. De Lucia and G. A. Di Lucca.Software Salvaging Based on Conditions.IEEE International Conference on Software Maintenance (ICSM 1994), Victoria, Canada, September 1994, pages 424-433.