Program Slicing Techniques and their Need in Aspect Oriented Programming

Preeeti Sikka
Ph. D. Scholar
Punjab Technical University
Jalandhar, Punjab, India

Kulwant Kaur
School of Information Technology
Apeejay Institute of Management
Technical Campus, Jalandhar

ABSTRACT
Analysis of various studies has proved the importance of applying program slicing on the source code while debugging, testing, quality assurance, software measurement and maintenance by extracting out the code irrelevant to criteria on behalf of which program slicing is applied, without affecting the relevancy in the information of the code. Program slicing techniques has been upgraded by its different types emerged to overcome the limitation of its previous types with different types of dependence graphs, criteria and on different types of programming paradigm with different types of tools. Paper highlights the effectiveness of hybrid slicing over static and dynamic slicing using case study for applying different types of slicing on same program. Paper also discusses the Aspect Oriented Programming and the areas where it has been proved better than object oriented programming and upholds the need for applying hybrid slicing on Aspect Oriented Program.

General Terms
Aspect Oriented Programming (AOP), Object Oriented Programming (OOP), Program Dependence Graph (PDG),

Keywords
Program Slicing, Static slicing, Dynamic slicing, Hybrid Slicing, Aspect Oriented Programming.

1. INTRODUCTION
'Program Slicing' as the name refers is to slice a problem to remove the part of a program without which the result of testing, maintenance, debugging would not be influenced and be the same as if slicing not applied but with lesser cost and time. Program Slicing applied for all general inputs, known as static slicing has the disadvantage that code may not be get reduced as expected. Dynamic Slicing comes into existence to overcome static slicing by slicing the program for specific input variable however in return pay memory to store the slice for different input variables. Hybrid Slicing takes the advantage of both static slicing and dynamic slicing. One over other different types of Slicing with their pros and cons are described in section 2. The pattern of the rest of the paper is as follows: Section 3 gives the case study of different types of slicing applied on same program with program dependence graph. With brief introduction of AOP Section 4 introduces two major areas in the programming where object oriented paradigm is lacking behind to be fit and will elaborate how Aspect Oriented approach enhances the Object oriented approach and will proposed the need of applying program slicing on Aspect oriented programs. Section 5 concludes the paper.

2. RELATED STUDY OF PROGRAM SLICING AND ITS TYPES
Static Slicing is the first type of slicing developed by Mark Weiser in 1988[1]. Static Slice of a program with respect to statement number 'x' will contain only those statements which are related to 'x' either by affecting the statement 'x' or getting affected by it resulting in reduction of the overall cost of software budget to test, debug or maintain a program due to reduction in lines of code. Static slicing is general for all set of inputs, length of code can be reduced more if slice could be obtained for specific input variable, and this idea was bought by Bodgan Korel and Janusz Laski in 1988 and termed as Dynamic slicing [2]. Dynamic slice being created for distinguished input variable is more précised over static slice and overcome the problems faced by static slicing in arrays and pointers of not knowing the result information about specific element of an array because of array being treated as a single variable in static approach. Dynamic data structures handle them more precisely [2]. Dynamic slicing can determine the value of array subscript during program execution allowing every element of an array to be treated as a separate variable resulting in more number of statements to be excluded after checking the influence of the specified element on the program. Pointer in program is another area where dynamic slicing is preferred over static slicing having the ability to create and manipulate dynamic variables needed to deal with pointers. Along with given advantages of working with specific input the main disadvantage is its run time overhead to collect the information traced in program execution for different provided inputs. Hybrid Slice integrates the dynamic information from a specific execution into a static slice analysis [3]. It uses dynamic information i.e. breakpoints and call history information into the static slice giving the reduced run time overhead than of dynamic slice. Another approach of slicing with reduced slice than static slice and less overhead than dynamic slice is Dependence cache slicing [4] which reduces the slice size than static slice especially in case of arrays and pointers . Dependence cache Slicing takes less cache size proportional to the number of variables instead of preparing caches for all as in static slice. Run time overhead of Dependence Cache slicing is proportional to the number of the variable access [4], reducing the run time overhead than the dynamic slice.

3. CASE STUDY: COMPARISION OF TYPES OF SLICING
Simple program of calculating the area of plane shapes based on the user's choice has been taken with line numbers (Figure 1) and shown the outputs with reduced line of code after applying different type of slicing on it (Figure 2 and Figure 3). Figure 2 has proved that there are instances when static slice size is approximately equal to the original size of code and

Figure 3 has proved that dynamic slice can reduce the slice size up to 50 %. Figure 3 has shown that applying the dynamic slice with specific input value, slice size can be reduced up to 50% than the size of static slice. Dynamic slice taken is for single input value ch=2. What for other input values? Dynamic slicing is to keep the trace for different slices created with different inputs. Need to keep different traces in memory results in high run time overhead. To overcome this issue Hybrid slicing has integrated the dynamic information into static slicing analysis [3]. Hybrid slice can be computed by applying breakpoints on program dependency graph. Program Dependence Graph (PDG), used to represent program slices [5] has been created to show the flow of data and control from one statement to another (Figure 4). Breakpoints are kept on statement no. 1,6,7,8,9,10. For mentioned program dynamic slice has to trace the slice for ch=1, ch=2, ch=3 and ch=4. To reduce this overhead hybrid Slicing first will execute the value of variable at breakpoint and Slice will only be created for the variable whose value is not as expected. Hybrid Slicing is proved to be better than dynamic slice and static slice as it reduces the memory overhead as compared to memory overhead in dynamic slicing and by predicting the control flow through breakpoints, higher accuracy is there as compared to prediction of control flow in static slicing.

```
1.  float area_sqr(float x)
2.  {
3.  return x*x;
4.  }
5.  float area_rect(float x, float y)
6.  {
7.  return x*y;
8.  }
9.  float area_cir(float x)
10. {
11. return 3.14*x;
12. }
13. void main ()
14. {
15. float len, br, s, r, ch ;
16. clrscr();
17. cout<<"Enter your choice/n";
18. cout<<"\n1. Area of rectangle";
19. cout<<"\n2. Area of Square";
20. cout<<"\n3. Area of circle";
21. cout<<"\n 4. Exit":
22. cin>>ch;
23. switch(ch)
24. {
25. case 1:cout<<"\n Enter the length and breadth of
    rectangle\n";
26. cin>>len>>br;
27. cout<<"Area of Rectangle is "<<Area_rect (len, br);
28. break;
29. case 2: cout<<"Enter the side of square";
30. cin>>s;
31. cout<<"Area of Square is "<< Area_sqr (s);
32. break;
33. case 3: cout<<"Enter the radius of circle";
34. cin>>rad;
35. cout<<"Area of Circle is "<<Area_cir (r);
36. break;
37. case 4: exit;
38. default: cout<<"Enter valid choice";
39. }
40. getch();
41. }
```

Figure 1: Program to Calculate the Area of Plane Shapes as per user choice.

```
1.  float area_sqr(float x)
2.  {
3.  return x*x;
4.  }
5.  float area_rect(float x, float y)
6.  {
7.  return x*y;
8.  }
9.  float area_cir(float x)
10. {
11. return 3.14*x;
12. }
13. void main ()
14. {
15. float len, br, s, r, ch;
16. clrscr();
17. ……..
22. cin>>ch;
23. switch(ch)
24. {
25. case 1:cout<<"\nEnter the length and breadth of
    rectangle\n" ;
26. cin>>len>>br;
27. cout<<"Area of Rectangle is "<<Area_rect (len, br);
28. break;
29. case 2: cout<<"Enter the side of square";
30. cin>>s;
31. cout<<"Area of Square is "<< Area_sqr (s);
32. break;
33. case 3: cout<<"Enter the radius of circle";
34. cin>>rad;
35. cout<<"Area of Circle is "<<Area_cir (r);
36. break;
37. case 4: exit;
38. default: cout<<"Enter valid choice";
39. }
40. getch();
41. }
```

Figure 2: Static Slicing- Slice of Program with respect to variable ch.

```
1.  float area_sqr (float x)
2.  {
3.  return x*x;
4.  }
……..
13. void main ()
14. {
……..
29. case 2: cout<<"Enter the side of square";
30. cin>>s;
31. cout<<"Area of Square  is "<< Area_sqr (s);
39. }
40. getch();
41. }
```

Figure 3: Dynamic Slicing- Slice of Program with respect to variable ch and input value = 2

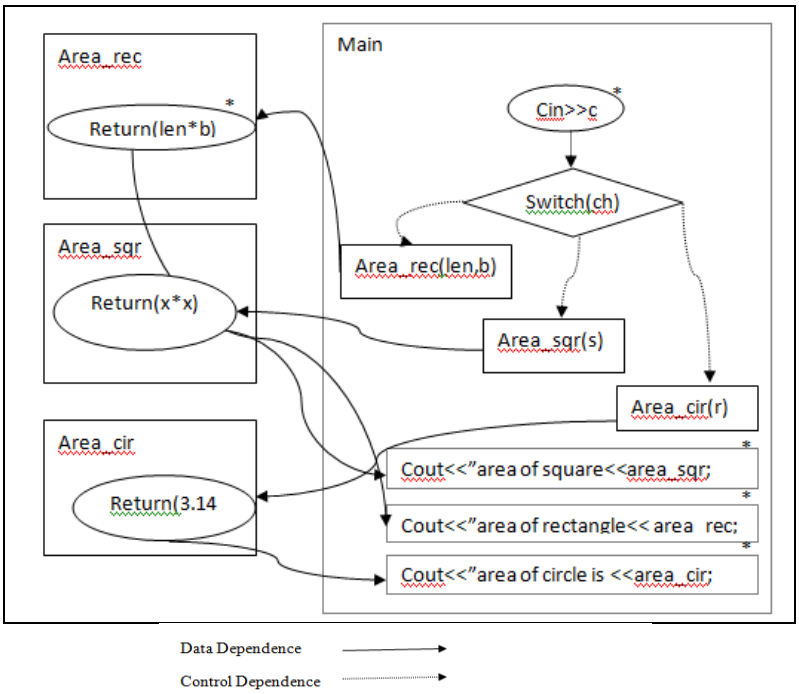

Figure 4: Program Dependence Graph with breakpoints applied

4. ASPECT ORIENTED PROGRAMMING

4.1 Deficiency in Object Oriented Programming

Basic principle of object oriented paradigm is the separate concern of every class but it is not happened actually. There are few codes that are cross cutting for an example code for exception handling that is common to all classes, if that code is put into class than it will violate the principle of OO that every class has its specific functionality [4] and if that code is not put inside any of the class then it results in tangled code which itself is a problem related to complexity and maintenance. The effectiveness of the program slicing for the programs with object oriented paradigm is discussed in the paper; analysis of program slicing effectiveness has also been done [6]. Another major issue with program slicing tools developed for object oriented language i.e. java is the difficulty to maintain the tools when language will get revised [7].

4.2 Need for program slicing on Aspect Oriented Programming

Aspect oriented approach solves the problem of tangled code by following the principle of Separation of Concerns [7]. Cross cutting concerns are linked to several parts of the program, AOP deals with them by putting them into a separate modular unit known as Aspects and has overcome the issue faced by Object oriented programs.

AOP implementation has been proved as both easy to understand and efficient [8] but due to the advanced features like aspects, join points, advice Aspect oriented programs are large, complex and difficult to analyze [9].If program slicing is applied to Aspect oriented program to convert the complex program to small slices then this only difficulty will get solved. Dynamic Slicing algorithm for aspect oriented program has been proposed [5] using dynamic aspect oriented dependence graph. Due to the feature of separation of concerns of AOP, developed tools can be easily maintained with newer versions of the respective language. AOP approach reduces the cost even more than the cost needed by

dependence cache slicing. There is need for tools to apply program slicing on Aspect Oriented program to overcome all the stated limitations and to reduce the software cost and time eminently.

Priority of different slicing techniques on behalf of the parameters defined is shown in the following table.

Table 1. Priorities as Low (L), Medium (M), High (H) of slicing techniques to be used on behalf of the parameters defined

	PS in OOP			**PS in AOP**
Parameters\Types of Slicing	**SS**	**DS**	**HS**	Any slicing Technique
Reduced Slice Size	L	H	M	Comparison will be same as in OOP amongst the slicing technique for given parameters
Less Runtime Overhead	H	L	M	
Capability of Dealing with Array/ Pointers	L	H	M	
Capability of Dealing with Inter procedural functions	L			H

SS: Static Slicing; DS: Dynamic Slicing; HS: Hybrid Slicing (HS)

5. CONCLUSION

Paper focuses on the pros and cons of different types of slicing and seen that with the time new type of slicing has added advantage over the previous type of slicing. Example of simple program has been taken to illustrate the difference of size of slice obtained by applying static and dynamic slicing.

Object Oriented approach being based on object fits with real world problems and it has been discussed that aspect oriented programming is better than the object oriented programming by improving the areas where object oriented approach is still lacking behind, Hybrid slicing has been analyzed as a good slicing technique to slice the program with lesser run time overhead and with higher precision. Paper establishes the idea of increasing the efficiency of Aspect oriented programming by providing the facility of hybrid slicing to it.

In future work, we would provide the features of hybrid slicing with other improved types of slicing to the aspect oriented programming to increase the efficiency of aspect oriented programs within lesser time and cost.

6. REFERENCES

[1] Weiser, M., "Program Slicing", IEEE Transactions on software engineering, Vol. 10, Issue 4, 1984, 352-357

[2] Korel, B. and Laski, J., "Dynamic Program Slicing", Information Processing Letters, Vol. 29, Issue 3, doi>10.1016/0020-0190(88)90054-3, 26 October 1988, 155-163.

[3] Gupta, R., Soffa, M.L., and Howard, J., "Hybrid slicing: Integrating Dynamic Information with static analysis", ACM Transactions on Software Engineering and Methodology, Vol. 6, Issue 4, October 1997.

[4] Takada, T., Ohata F., Inoue K., "Dependence-Cache Slicing: A Program Slicing Method Using Lightweight Dynamic Information", IWPC, 2002, 169-177

[5] Pan, K., Kim, S., E. J. Whitehead, Jr., "Bug Classification Using Program Slicing Metrics," Sixth IEEE International Workshop on Source Code Analysis and Manipulation, ISBN: 0-7695-2353-6, 2006, 31 – 42.

[6] Saleem, M., Hussain, R., Ismail, Y., Mohsin, S., "Cost Effective Software Engineering using Program Slicing Techniques", Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human, ACM New York, NY, USA ©2009, ISBN: 978-1-60558-710-3 doi>10.1145/1655925.1656065, 2009, 768-772.

[7] Ishio, T., Kusumoto, S., Inoue, K., "Program slicing tool for effective software evolution using aspect-oriented technique", Proceedings of the 6th International Workshop on Principles of Software Evolution, ISBN: 0-7695-1903-2, 1-2 Sept. 2003, 3-12.

[8] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier,J.M., Irwin, J., "Aspect Oriented Programming" , Published in proceedings of the European Conference on Object-Oriented Programming, Finland. Springer-Verlag LNCS 1241, Print ISBN: 978-3-540-63089-0; Online ISBN: 978-3-540-69127-3, June 1997.

[9] Mohapatra, D.P., Sahu, M., Kumar, R., and Mall, R., "Dynamic Slicing of Aspect-Oriented Programs", Informatica 32, 2008, 261–274.