Dynamic Slicing - An Aid to Debugging for Binary Search Based Applications

Divanshi Priyadarshni Wangoo
ITM University, Department of Computer Science and Engineering
Sector-23A, Gurgaon-122002, Haryana, India
e-mail: divanshi.wangoo@gmail.com

Abstract—A variety of slicing techniques have been developed for different object-oriented programs. Dynamic slicing has a practicable impact in a variety of applications demanding less time, cost, maintenance and a better understanding of the source code by the developer. Binary searching has many applications ranging from file indexing to large data searching. Debugging being an important software engineering task becomes essential for applications based on binary searching. Making debugging a more methodical process of finding and reducing the number of bugs is at the heart of this process. Graph-based program representation is suitable as a basis for a range of software engineering applications. Dynamic slicing was specifically designed as an aid to debugging and can be used for searching the offending statements which caused the program error and bugs. Due to various existing relationships such as inheritance, polymorphism, dynamic binding, encapsulation, abstraction, etc, there is a need for dynamic slicing that aid interactive applications. Therefore dynamic slicing is a more useful and powerful technique for object-oriented programming applications. In this paper, our effort has been to implement dynamic slicing in binary search based applications.

Index Terms—System dependence Graphs (SDG), Debugging, Internal dependencies, Impact analysis, Concurrency, Dynamic Dependence Graphs (DDG)

I. INTRODUCTION

Object slicing is the process of inspecting statements in a program object by object suitable for applications such as program understanding, debugging, testing and reverse engineering [1]. Object-oriented slicing is done with the help of system dependence graphs (SDG) using an algorithm on the SDG to compute interprocedural slices. Object-oriented concepts such as classes, objects, inheritance, polymorphism, dynamic binding, message passing, abstraction, information hiding are prevalent in applications [2]. These also can be represented using the SDG’s for imperative language programs. The SDG represents the features of object-oriented programs, which includes the objects that are present in the languages such as C++ and Java.

A. Benefits of SDG

The main benefits of SDG are that it distinguishes data members for different objects, represents objects that are used as parameters or data members in other objects, and represents how polymorphism affects parameters and parameter binding [3]. Moreover the SDG technique for object slicing lets a user inspect statements in a slice object by object. Analyzing and representing software in terms of its internal dependencies is important for a variety of software engineering applications. Operations included amongst

DOI: 03.AETS.2013.3.131_34
© Association of Computer Electronics and Electrical Engineers, 2013
selected few are computation of program metrics, traceability, completeness, volatility; product metrics, quantitative measure- code metrics; lines of code (LOC), design metrics, process metrics or qualitative measure; design structure quality index; functional points (FP).

B. Impact Analysis

Analyzing programs is highly crucial to improve overall efficiency in software maintenance, according to Pressman (2005). Impact analysis is a process that predicts & determines the parts of a software system, which can be affected by the changes brought to the system [8]. Impact analysis is essential for identifying those parts of a program that require retesting; it reduces potential errors due to unknown change impacts and improves estimation of time and money required for maintenance [8]. There are two types of impact analysis; static and dynamic impact analysis. Static impact analysis depends on analysis of source code and dynamic impact analysis depends on a set of executions of the system.

II. Program Slicing

A program slice is a set of program statements and predicates that might affect the value of a program variable v that is defined or used at a program point p [3,4]. The slicing criterion is represented by <v, p>, where v is the program variable and p is the program point [3]. According to Pressman, for the maintenance of software applications, program understanding, debugging, regressing testing, and reverse engineering is essential. Thus slicing proves to be beneficial.

A. Categories of Program Slicing

Different applications require different types of slices, so program slicing has been categorized by taking into consideration the run-time environment, graph traversal and the program being from which slices are computed [4,5]. Slices can be backward or forward, static or dynamic, intra-procedural or inter-procedural [4].

Static Slicing and Dynamic Slicing: Static slicing techniques uses static analysis to derive slices i.e., the source code of the program is analyzed and the slices are computed for all possible input values [4]. Therefore static slices are conservative and contain more statements than necessary. Slices from the parent program are computed irrespective of any input. So, static slices are less useful as they contain most of the statements of the parent program [4,5].

Dynamic slicing makes use of the information about a particular execution of a program [4]. A dynamic slice with respect to a slicing criterion <s, V>, for a particular execution, contains those statements that actually affect the slicing criterion in the particular execution [4]. Therefore dynamic slices are usually smaller than static slices and are more useful in interactive applications such as program debugging testing and maintenance [4,5].

Backward Slicing and Forward Slicing: The Backward slice contains all parts of the program that might directly or indirectly affect the slicing criterion [4, 5]. Thus, a static backward slice provides the answer to the question “which statements affect the slicing criterion?”[4, 5]

A forward slice with respect to a slicing criterion<s, V> contains all parts of the program that might be affected by the variables in V used or defined at the program point s [4]. Thus a forward slice provides the answer to the question “which statements will be affected by the slicing criterion?”[4, 5]

Intra-Procedural Slicing and Inter-Procedural Slicing: Intra-procedural slicing computes slices within a single procedure and calls to other procedures are either not handled at all or handled conservatively [4,5]. For object-oriented programs inter-procedural slicing is more useful as practical object-oriented programs contain more than one method or procedure [4,5]. If a program consists of more than one procedure, inter-procedural slicing can be used to derive slices that span multiple procedures [4, 5].

III. System Dependence Graphs

A system dependence graph (SDG) contains one procedure dependence graph for each procedure used in the program [3]. A procedure dependence graph represents a procedure as a graph in which vertices are statements or predicate expressions [3]. There are data dependence edges which represents flow of data between statements or expressions. Control dependence edges are used to represent control conditions on which the execution of a statement or expression depends [3]. There is an entry vertex associated with each procedure dependence graph which represents entry into the procedure. For parameter passing, an SDG
associates each procedure entry vertex with formal-parameter vertices. A formal-in vertex is used for each formal parameter of the procedure and a formal-out vertex for each formal parameter that may be modified by the procedure [3, 6]. Each call site associated with an SDG of the procedure has a call vertex and a set of actual parameter vertices which are an actual-in vertex for each actual parameter at the call site and an actual-out vertex for each actual parameter that may be modified by the called procedure [3, 6]. Further an SDG connects procedure dependence graphs at call sites [3]. A call edge connects a call vertex to the entry vertex of the called procedure’s dependence graphs [3, 6]. Parameter edges are also used for parameter passing representation with a set of parameter-in and parameter-out edges connecting actual-in and formal-in vertices (formal-out and actual-out vertices). SDG also uses summary edges to explicitly represent the transitive flow of dependence across call sites caused by data dependences, control dependence, or both [3, 6]. A summary edge connects an actual-in vertex and an actual-out vertex if the value associated with the actual-in vertex may affect the value associated with the actual-out vertex [3, 6]. The illustration of the above SDG can be well understood from Fig.1 below and consequently from Fig.2 further.

A. SDG for Binary Search Program

Precondition: A sorted array

1. const int MAX=10;
2. class array

private:
3. int a[MAX];
4. int count;
public:
5. array();
6. void add(int item);
7. void search(int item);
};
8. count=0;
9. for(int i=0;i<MAX;i++)
10. a[i]=0;
11. void array::add(int item)
12. if(count<MAX)
13. a[count]=item;
14. count++;
15. else
16. cout<<"Array is full"<<endl;
17. void array::search(int num)
18. int mid,lower=0,upper=count-1,flag=1;
19. for(mid=(lower+upper)/2;lower<=upper;mid=(lower+upper)/2)
20. if(a[mid]==num)
21. break;
22. if(num>a[mid])
23. lower=mid+1;
24. else
25. upper=mid-1;
26. cout<<"Element is not present in the array";
27. if(flag)
28. flag=0;
29. cout<<"The number is at position"< num
30. main()
31. array a;
32. a.add(1);
33. a.add(2);
34. a.add(3);
35. a.add(4);
36. cout<<"Enter number to search";
37. cin>>num;
38. a.search(num);

Fig.1: Binary search program

280
A new form of slicing called dynamic slicing is dependent on input data and is generated during execution time analysis [4]. Researchers have presented algorithms for finding dynamic program slices using system dependence graphs with their respective program dependence graphs and class dependence graph. They have proposed dynamic slicing methods by marking nodes on a static program dependence graphs. To represent various dynamic dependencies between statement instances for a particular execution of an object-oriented program two types of dependence graphs were introduced- Dynamic Dependence Graphs (DDG) and Dynamic Object-Oriented Dependence Graphs (DODG) [4,7].

A DODG is an arc classified digraph \((V, A)\), where \(V\) is the multi-set of flow-graph vertices, and \(A\) is the set of arcs representing dynamic control dependencies and data dependencies between vertices [4].

Concurrency in object-oriented programs is a useful and important concept for applications requiring fast execution results with little or no programming bugs at all. Therefore it is necessary that concurrent object-oriented programs should have synchronized interactions among objects, multiple threads of control and a dynamically varying number of objects for simplicity and clear understanding of programs. Concurrency is implemented in the form of multithreaded programs and dynamic slices can be computed for such programs. Dynamic slicing techniques can also be applied to distributed programming applications spreading over a wide geographical region. Subsequently, remotely localized people working on such applications can make their work much easier [4]. Dynamic slicing, therefore, becomes an important aspect for the programmer to better understand the code that has been developed. Thus it allows the programmer to reduce the probability...
of an error or bug hindering the application program to run to the desired level with less cost and maintenance.

A. Dynamic Slicing For Binary Search Program

Binary searching is an effective searching technique which can be applied to large scale applications such as phonebook searching, e-dictionary, large directories that require an effective data structure searching technique. Binary search algorithm requires $O(\log_2 n)$ searches to search a whole binary tree and thus reduces searching time and promotes fast execution of the result. Many applications based on binary searching are fast, efficient and takes less time to implement.

Dynamic slicing can be applied to binary search based applications by computing a dynamic slice at a point in the program called the slice point which can be used to reduce the chances of the programmer of making errors at the beginning stages of the Software Development Lifecycle Model (SDLC). During the designing and implementation phases the programmer would have an easy hand of removing bugs before going to the testing phase of the SDLC.

A dynamic slice for the binary search program can be computed and a slice point can be made input by the programmer in case of programming errors like typecasting error occurring at the function calling and function definition stage, the programmer would be able to know the bug and would remove it at an early stage. Other common programming errors made the programmer such as memory out of bounds for arrays; infinite loop can be improved and removed at an early stage the dynamic slicing technique.

In the SDG of Fig.2 the binary search program for a sorted array, the slice point a is the point at which the function is defined and slice point "b" is the function calling point. If there is any typecasting error at these two slice points the error can be removed at a very initial phase of the software development which would help in reducing cost, time, money and maintenance of the software application developed providing a relief to the developer too. Therefore, including dynamic slicing techniques in the developing stage proves to be very efficient and effective and should be used in applications demanding such solutions.

V. CONCLUSION

A more precise implementation can be made for programs built on inheritance and thread concepts. For analysis of the flow from base class to derived class and from one thread to another (multithreading), as well parallel algorithms for slicing(slicing of distributed object-oriented programs), and dynamic slicing of concurrent object-oriented programs. By slicing objects in an object-oriented program, there is better support for debugging and program understanding for large scale programs. It increases program visibility, code understanding through easy recognition of control dependence edges, data dependence edges, program flow, procedure calls and parameter bindings.

ACKNOWLEDGEMENT

I would like to seize this opportunity to thank my guide Dr. Supriya P. Panda of CSE &IT at ITMU for her efforts of timely supervising and leading me through the way to accomplish this piece of work.

REFERENCES