Ultra Scientist Vol. 26(2)A, 123-130 (2014). ISSN 2231-3478 (Print)
www.ultrascientist.org 2319-8052 (Online)

Program Slicing for Composite Data using Finite State Machine

K. THIAGARAJAN[1], J.KAVITHA[2] and PONNAMMAL NATARAJAN[3]

[1]Professor, Department of Mathematics in SBM College of Engineering and Technology, Dindigul, Tamil Nadu (INDIA)
email : vidhyamannan@yahoo.com
[2]Department of Mathematics in SBM College of Engineering and Technology, Dindigul, Tamil Nadu (INDIA)
Email: manokavi.j@gmail.com
[3]R&D, Advisor in Rajalakshmi Engineering College, Affiliated to Anna University Chennai, Tamil Nadu (INDIA)

(Acceptance Date 9th June, 2014)

Abstract

In the software development life cycle, the identification of errors/bugs plays an important role, as the end product should be bug free. This can be achieved through the concept of program slicing. Generally, program slice has a wider spectrum of applications that include debugging, testing, maintenance, code understanding, complexity measurement, security etc., There are two different categories of program slicing namely, static slicing and dynamic slicing which can be identified using connected graph and its structural arrangements. Existing slicing techniques were implemented over intra procedural non composite data [only which doesn't include arrays, structure, union etc,]. In the proposed work static program slicing concepts is applied over array of elements using finite state machine (FSM) method. Here, backward slicing techniques are applied to get quality output.

Key words: Slicing, Slice, Program Dependence Graph, Finite automata, Finite State Machine.

I. Introduction

Program slicing[1] is a program analysis technique that reduces a program to those statements that are relevant for a particular context or computation. Alternatively it is a technology that decomposes a program and extracts the portion of interest with respect to certain criterion that reduces the complexity of control flow. There are two types of slicing

namely static and dynamic slicing.

A. Static slicing :

Static program slice comprises of those program statements that affect the value of a variable at some program point of interest which is referred as slicing criterion. It is otherwise known as a static algorithm only uses statically available information that does not care about the path of execution in the program. Here all possible executions of the program are taken into account. A static slice preserves a projection of the semantics of the original program for all possible inputs[2,3].

B. Dynamic slicing :

A dynamic slice[5,6] consists of only those statements that actually affect the value of a variable at a program point for a given execution. It computes those statements which influence the value of a variable occurrence for a specific program. It requires the path to be same in the original program and in the slice. A dynamic slice is constructed with respect to only one execution of the program (iteration number is taken into account). A dynamic slice preserves the effect of the program for a fixed input. Dynamic slicing requires the path to be same in the original program and in the slice[8].

C. Concept of slice :

A slice is constituted by the statements that affect the value of the program with respect to the given variable occurrence. The various statements are statements using variables, expressions and assignments and control flow statements. It is an executable portion of the original program whose behavior is, under the same input, indistinguishable from that of the original program on a given variable 'V' at point 'P' in the program. Weiser defines a program slice with respect to slicing criterion that consists of program point 's' and a subset of program variables 'v' is now called executable backward static slice. Executable means the slice is required to be an executable program, backward means the direction edges are traversed when a slice is computed using a directed graph and static means they are computed as the solution to a static analysis problem (*i.e.*, without considering the program's input).

D. Computation of program slice :

(i) An iterative algorithm, computes the slice as set of data-flow equations

(ii) Representing the program as a graph, where vertices represent expressions and edges represent different types of inter-dependencies is also used to compute the slice as a simple graph reachability problem. This include identification of all basic components and analysis of all kinds of dependence relationships between these components in object-oriented programs, construction of Object Oriented Program Dependence Graph (OPDG), slice program dependence graph based on graph reachability slicing algorithms and get sliced program by projecting sliced program dependence graph to source program.

A forward[7] slice cosists of all program statements that are affected by a given point in the program whereas a backward slice consists of all program statements that affect a given point in the program[9].

E. Interprocedural Slicing :

(i) Slicing across procedures complicates the situation due to the necessity of translating and passing the criteria into and out of calling and called procedures. When a procedure 'P' calls a procedure 'Q' at statement 'I', the active criteria must first be translated into the context of 'Q' and then reversed once 'Q' has been sliced. Summary edges represent the transitive dependences due to procedure calls. Slices are computed by doing a two phase traversing on the System Dependence Graph. Here the main problem is the effective computation of summary edges[11].

F. Intra procedural Slicing :

In a program P, the slicing criterion is derived and the slicing is done inside the procedures located in 'P'. The slicing criterion is fully depend on the Program 'P' itself [10].

G. Static Backward Slicing :

A normal (static backward) slice of a program is defined using a set of variable identifiers (V) and a point in the program or line number (i). For example, the slicing criterion ({a, b}, 30) will result in a slice which only has code that affects the variables a and b up to line 30 in the program.

H. Conditioned Slicing :

Conditioned Slicing bridges the gap between static and dynamic slicing. Conditioned Slicing is relevant to a set of initial states rather than just one, as is the case with dynamic slicing. A conditioned slice can therefore act as a typical dynamic slice if just one initial state is specified. At the other end of the scale, a conditioned slice can act as a static slice if all possible initial states are specified[11].

I. Amorphous Slicing :

Amorphous slicing involves applying changes to the code syntax while preserving the semantics (meaning). This approach can help in understanding certain aspects of the code.

J. Forward Slicing :

In backward slicing, lines of code that do not affect the slicing criterion are removed. With forward slicing, lines that are not AFFECTED by the slicing criterion are removed. It looks forward in the code, relative to the line number specified in the criterion, instead of backwards[5].

K. Finite state machine :

A finite state machine (FSM) or finite state automaton or simply a state machine, is a model of behavior composed of a finite number of states, transitions between those states, and actions. A finite state machine is an abstract model of a machine with a primitive internal memory.

Outline of Paper :

Section 1 presents the overview of the static slicing concepts, its types and techniques Section 2 presents architectural design and flow diagram of the proposed system. Section 3 presents the algorithm for static slicing of array of elements. Section 4 presents program dependency graph. Section 5 presents

preliminaries on finite State machine. Finally, Section 6 presents conclusions and future work.

II. Architecture of the Proposed System

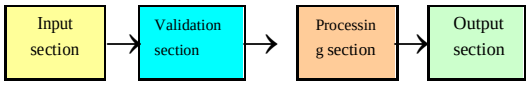

Fig. 1 Architectural Design of the proposed System

The block diagram describes the function performed by all aspects. Here the block diagram is divided into four sections namely, input section, validation section, processing section and output section. Input section reads actual array size, array element and current array size. For example, the actual array size is 10, array element also 10 and current array size is n=5. In validation section, all the input values which is read in input section are checked. If any discrepancies occur then make a correction. In processing section, if the input is valid, then calculation is done directly to get the exact result. Otherwise, calculation is done after updating the input. Finally, Output section displays the result (it may be correct or incorrect with respect to current array size).

This architecture diagram describes the overall flow of this paper. Initially, array size (n) and m number of array elements are read from the user. Then preprocessing such as smoothing is done on all these inputs. Then the inputs are validated to check whether the

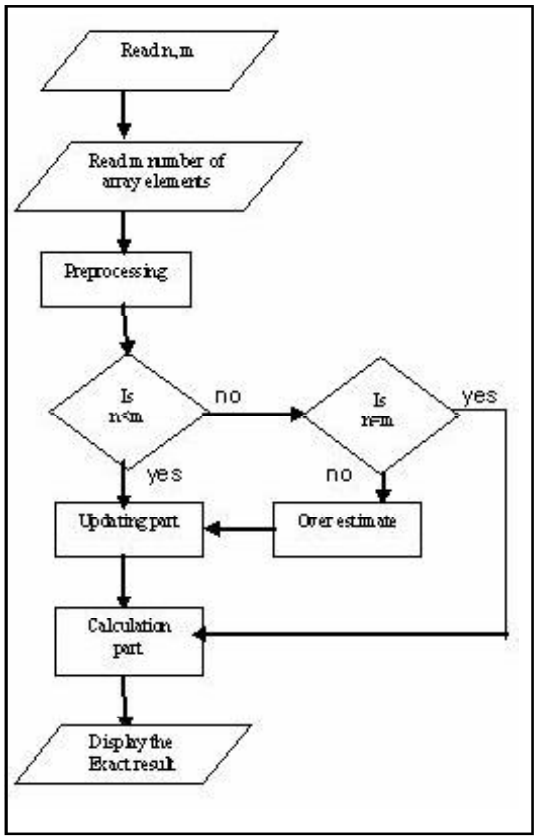

Fig. 2 Flow Diagram of the Proposed System

input is suitable for processing. Finally, the inputs are processed based on the size of the array and the number of array elements given as an input. If n = m, then it produces correct output, otherwise a correction is made based on the input. If n<m then, the input values of array elements are updated and then processing is done to get the correct output. If n>m then it implies an over estimate, i.e., the array size is greater than actual array size. Therefore, array size is updated until it is equal to number of array elements given as an input[10].

III. Algorithm

Input: Array of n elements

Output: static slicing of array elements

Other Variable: total, average

```
for each EX in the array
{
if (EX=AS)
{
calculate total; calculate average;
}
else if (EX<AS)
{
calculate erroneous result // but the total and
average is incorrect
}
else // (EX>AS)
make the necessary updation;
}
```

IV. Program Dependence Graph

A program dependence graph is a suitable internal program representation for monolithic programs, for the purpose of carrying out certain software engineering operations such as slicing and computation of program metrics.

Original Program:

```
1. #include<stdio.h>
2. int main()
3. {
4. int n,i,tot;
5. float avg;
6. int a[]={10,20,30,40,50,60,70,80,90,100};
7. printf("enter the value of n");
8. scanf("%d",&n);
9. tot=0;
10. for(i=1;i<=n;i++)
11. tot=tot+a[i];
12. avg=tot/n;
13. printf("%d\n%f ",tot,avg);
14. }
```

Static Slicing Criterion
Slicing criterion: (a[],n)

Which statements have a direct or indirect effect on variable n.

Static slice with respect to (a[10],n)

```
4. int n,i,tot;
5. float avg;
6. int a[]={10,20,30,40,50,60,70,80,90,100};
7. tot=0;
8. for(i=1;i<=n;i++)
9. tot=tot+a[i];
10. avg=tot/n;
11. printf("%d\n%f",tot,avg);
```

Nomenclature
EX is an element index
AS is a array size

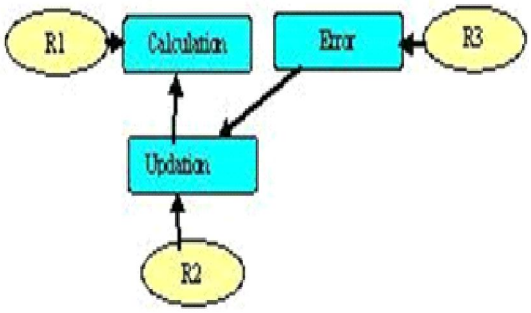

Fig. 3. Program dependency graph

Where

- R1= [n=m]
- R2= [n<m]
- R3= [n>m]

V. Finite State Machine Priliminaries

Slicing methodology may be expressed as the finite state machine (finite automaton), however it can be shown that both NFA and DFA have the same capacity. The following describes the definition various basic finite automata definitions. Finite automata recognize regular languages only. It receives as a string usually from an input tape. It delivers no output at all except an indication of whether the input is acceptable or not.

Finite state machine

A finite state machine (finite automata) is a mathematical model of how[12-13] computer functions while running a program. It consists of finite set of states and a set of transitions from state to state that occurs on input symbols from alphabet ". There are two types of finite automata namely Deterministic Finite Automata (DFA), Non Deterministic Finite Automata (NFA).

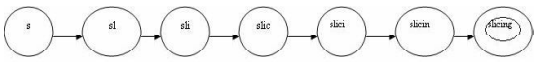

Fig. 4 FSM: recognizing a string "slicing"

A string x is said to be accepted by DFA or NFA M=(Q, Σ, δ, q_0,F), if δ (q_0, x) = P for some P in F. let us take an example of finite automata Q={q_o,q_1}, Σ={0,1},F={q_o}, δ is a transition function. The transition table is follows.

Table I
Input Transition Table

State	Input Symbol	
	0	1
q_o	{q_o,q_2}	{q_1}
q_1	{q_o}	{q_o,q_1}

The equivalent Finite state machine model is as follows

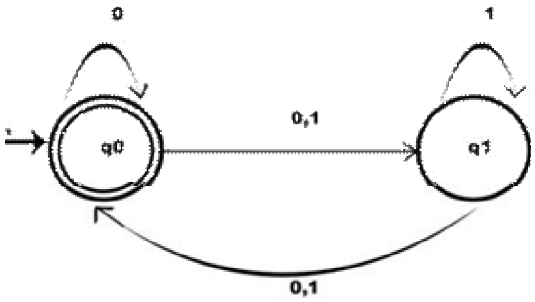

Fig. 5. Finite state machine model for table 1

Static slicing through DFA

Generally, we specify the slicing concepts in the following way.

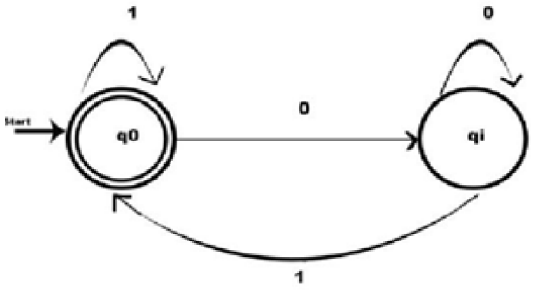

Fig. 6. DFA of Static Slicing

Where i=1, 2…

Example:

M= (Q, Σ , δ , q_0 , F)

Q= {q_0, q_1, q_2} Σ={0,1}

Table II
Input Transition Table

State	Input Symbol	
	0	1
q_0	{q_1,q_2}	{q_0}
q_1	{q_1}	{q_o}
q_2	{q_2}	{q_1}

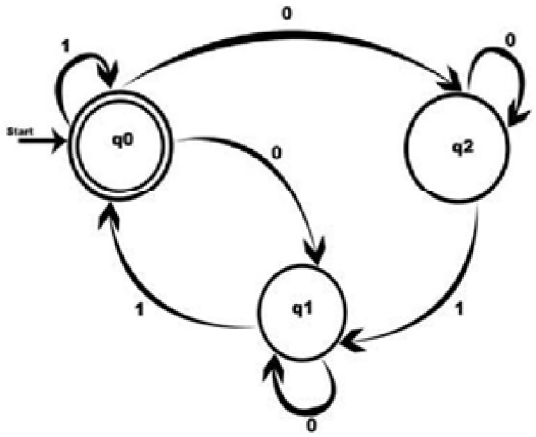

Fig. 7. Finite state machine model for table 2.

Transition Function

δ (q_0, 1) = {q_0}

δ (q_0, 0) = {q_3}

δ (q_1, 1) = {q_1}

δ (q_1, 0) = {q_0}

δ (q_2, 1) = {q_2}

δ (q_2, 1) = {q_1}

VI. Conclusion And Future Work :

Existing system have expressed the slicing concepts in the form of flow diagram. In the proposed system automaton explanations for approaching static and dynamic slicing have been suggested. Finite State Machine (FSM) gives full model of static program slicing, which accepts possible state (1) and declines impossible state (0). This paper the slicing techniques, were implemented over intra procedural composite data type array. Future enhancement would be to implement composite data such as structure, union etc, with the help of different mathematical techniques via automata theory[11-13].

References

1. Tracy Hall and Paul Wernick, “Program Slicing Metrics and Evolvability: an Initial Study”, Proceedings of the 2005 IEEE International Workshop on Software Evolvability (Software-Evolvability’05) 0-7695-2460-5/05,2005.
2. Heng Lu, Heng Lu, T.H. Tse, “Static Slicing for Pervasive Programs”, Proceedings of the Sixth International Conference on Quality Software (QSIC’06) 0-7695-2718-3/06, 2006.
3. Kai Pan, Sunghun Kim, E. James Whitehead, Jr, “Bug Classification Using Program Slicing Metrics”, Proceedings of the Sixth IEEE International Workshop on

Source Code Analysis and Manipulation (SCAM'06) 0-7695-2353-6/06,2006.

4. David Binkley Nicolas Gold, Mark Harman, Zheng Li and Kiarash Mahdavi, "An Empirical Study of Executable Concept Slice Size", Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06) 0-7695-2719-1/ 06,2006.
5. A´ rpa´d Besze´des, Tama´s Gergely and Tibor Gyimo´thy, "Graph-Less Dynamic Dependence-Based Dynamic Slicing Algorithms", Proceedings of the Sixth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM'06) 0-7695-2353-6/06, 2006.
6. Yancheng Wang, Bixin Li, Xufang Gong, "An Extension to Robustness Slicing Algorithm Based on Dynamic Array", Proceedings of the Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06) 0-7695-2611-X/ 06,2006.
7. Deji Fatiregun Mark Harman Robert M. Hierons, "Search–based amorphous slicing", Proceedings of the 12th Working Conference on Reverse Engineering
8. (WCRE'05)
9. Prof. J. T.Lallchandani Dr. Rajib Mall, "Computation of dynamic slices for object-oriented concurrent programs", Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)
10. Hai Huan Wei-Tek Tsai Raymond Paul, "Proof Slicing with Application to Model Checking Web Services", Proceedings of the Eighth IEEE International
11. Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)
12. Weiser M, "Program slicing", Proc. ICSE 1981, San Diego, California, Mar. 9–12 1981, pp.439– 449.
13. Bernard M.Moret, The Theory of Computation, Perarson education,First Indian reprint, (2002).