Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Science and Engineering

Elixir Comp. Sci. & Engg. 60 (2013) 16225-16227

ISSN: 2229-712X

Program slicing: A Review

Lipika Jha[1] and K.S. Patnaik[2]

[1]Birla Institute of Technology Extension Center Noida, A7 Sector-1,Noida, Uttar Pradesh, India.

[2]Birla Institute of Technology, Mesra, Ranchi, Jharkhand, India.

ARTICLE INFO

Article history:

Received: 4 May 2013;

Received in revised form:

17 June 2013;

Accepted: 5 July 2013;

Keywords

Slicing techniques,

Data dependence,

Control dependence,

Data flow equation,

Control flow graph.

ABSTRACT

Program Slicing is a method for automatically decomposing programs by analyzing their data flow and control flow. Slicing reduces the program to a minimal form called "slice" which still produces that behavior. Program slice singles out all statements that may have affected the value of a given variable at a specific program point. Slicing is useful in program debugging, program maintenance and other applications that involve understanding program behavior. This paper is a review paper on program slicing.

1. Introduction

Program slicing is one of the techniques of program analysis. It is an alternative approach to develop reusable components from existing software. To extract reusable functions from ill-structured programs we need a decomposition method which is able to group non sequential sets of statement. Program is decomposed based on program analysis. The decomposed program is called slice which is obtained by iteratively solving data flow equations based on a program flow graph. Program analysis uses program statement dependence information (i.e. data and control dependence) to identify parts of a program that influence or are influenced by a variable at particular point of interest is called the slicing criteria.

Slicing Criterion:

$$C = (n , V)$$

where n is a statement in program P and V is a variable in P .A slice S consists of all statements in program P that may affect the value of variable V at some point n.

This paper gives the overall knowledge of the program slicing. However, the emphasis of this paper is primarily of a theoretical rather than of a practical. The goal of this paper is to have the thorough knowledge of the program slicing.

The rest of the paper is organized as follows. Section 2 defines some common slicing techniques. Section 3 defines types and traversal of slicing. Section 4 defines slicing of object orient programs. In section 5 the different application of program slicing is explained. Finally section 6 includes references.

2. Slicing Technique

The original concept of a program slice was introduced by Weiser. He claims that a slice corresponds to the mental abstractions that people make when they are debugging a program. A slice S consists of all statements in program P that may affect the value of variable V at some point n.

Variables V at statements n can be affected by statements because:

– Statements which control the execution of n(Control Dependence)

– Statements which uses the V at n (Data Dependence)

The goal of slicing is to create a subprogram of the program (by eliminating some statements), such that the projection and the original program compute the same values for all variables in V at point n.

2.1 Data dependence and control dependence: Data dependence and control dependence are defined in terms of the CFG of a program.

Data Dependence: A statement *j* is data dependent on statement *i* if a value computed at (i) is used at *j* in some program execution.

(i) x ε DEF(i) and x ε USE(j)

or x ε DEF(i) and x ε DEF(j)

(ii) There exists a path from *i* to *j* without intervening definitions of x.

Control dependence information identifies the conditionals node that may affect execution of a node in the slice. In a CFG, a node j post-dominates a node i if and only if j is a member of any path from i to Stop.

2.2 Data Flow Equation: According to Weiser the slicing is computed by iteratively solving data flow equation.

The set of relevant variable $\mathbf{R}_c^0$ (i) with respect to slicing criteria C=(p,V) is:

1. $\mathbf{R}_c^0$ (i)=V when i=p
2. $\mathbf{R}_c^0$ (i)=($\mathbf{R}_c^0$ (j)-DEF(i))U

(USE(i) if $\mathbf{R}_c^0$ (j)∩DEF(i)≠ø)

The set of relevant statements to C denoted S^0_C, is defined as:

S^0_C={i | Def(i) ∩ $\mathbf{R}_c^0$ (j) ≠ ø ,i → ^{CFG} j}

$\mathbf{B}_c^0$ ={b| i ϵ Infl(b),i ϵ $\mathbf{S}_c^0$ }

The set of indirectly relevant statement R_c^{i+1}

$$\mathbf{R}_c^{i+1} (n)= \mathbf{R}_c^i (n) \mathbf{U}_{b \in B_c^0} \mathbf{R}_1(b,\mathbf{U} (b)))^{\dagger o} (n)$$

2.3 Program dependence graph: Program dependence graph is defined in terms of a program's control flow graph .The PDG includes the same set of vertices as the CFG, excluding the

Tele:

E-mail addresses: lipika_192@yahoo.co.in, ktosri@gmail.com

EXIT vertex. The edges of the PDG represent the control and flow dependence induced by the CFG. A program dependence graph contains a flow dependence edge from vertex v_1 ,to vertex v_2 iff all of the following hold:
(1) v_1 is a vertex that defines variable x.
(2) v_2 is a vertex that uses x.
(3) Control can reach v_2 after v_1 via an execution path along which there is no intervening definition of x. That is, there is a path in the standard control flow graph for the program by which the definition of x at v_1 reaches the use of x at v_2 .
A flow dependence that exists from vertex v_1 to vertex v_2 is denoted by $v_1 \rightarrow_f v_2$.
Flow dependences can be further classified as loop carried or loop independent.
A flow dependence $v_1 \rightarrow_f v_2$ is carried by loop L, denoted by $v_1 \rightarrow_{lc(L)} v_2$, if in addition to (l), (2), and (3) above, the following also hold:
(4) There is an execution path that both satisfies the conditions of (3) above and includes a backedge to the predicate of loop L.
(5) Both v_1 and v_2 are enclosed in loop L.
A program dependence graph contains a def-order dependence edge from vertex v_1 l to vertex v_2 iff all of the following hold:
(1) v_1 and v_2 both define the same variable.
(2) v_1 and v_2 are in the same branch of any conditional statement that encloses both of them.
(3) There exists a program component v_3 such that $v_1 \rightarrow_f v_3$ and $v_2 \, f \rightarrow v_3$.
(4) v_1 occurs to the left of v_2 in the program's abstract syntax tree.
The extraction of slices is based on data dependence and control dependence. A slice is directly obtained by a linear time walk backwards from some point in the graph, visiting all predecessors.

3. Types of Slicing

3.1 Static Slicing

The slice which is computed for a general set of variables are called static slices i.e., static slices are the slices for the whole range of values of the variables involved in the program.

3.2 Dynamic Slicing

It includes all statements that affect the value of the variable occurrence for the given program inputs, not all statements that did affect its value. Dynamic slicing criterion consist of a triple (n, V, I) where I is an input to the program.

3.3 Quasi Slicing

It is a hybrid of Static and Dynamic Slicing. Static slicing is examined during compile time, using no information about the input variables of the program. In Quasi slicing the value of some variables are fixed and the program is analyzed while the value of other variables vary. The behavior of the original program is not changed with respect to the slicing criterion.

3.4 Conditioned Slicing

The conditioned slicing criterion includes the set of states in which the program is to executed, allowing the programmer to specify, not only the variables of interests, but also the initial conditions of interest. Any statements, which we know it will not execute, may be omitted afterwards. This not only shows the initial awareness of knowledge about the condition in which the program is to be executed, but also has the advantage that it allows for additional simplification during slice construction. This characteristics is practically used in a tool-assisted form of analysis a program by cases (one per condition), and makes it attractive as a supporting technique for program comprehension.

3.5 Backward Slicing

It includes all parts of the program that might have influenced the variable at the statement under consideration. The backward approach can be used in locating the bug by examining all previously executed statements with respect to a variable v at statement n, where output of v is found incorrect at that point.

3.6 Forward Slicing

Contains all those statements of P which might be influenced by the variable.

3.7 Amorphous slice

All approaches to slicing discussed so far have been 'syntax preserving', That is, they are constructed by the sole transformation of statement deletion. The statements which remain in the slice are therefore a syntactic subset of the original program from which the slice was constructed. Amorphous slices are constructed using any program transformation which simplifies the program and which preserves the effect of the program with respect to the slicing criteria.

3.8 Inter-procedural Slicing

Interprocedural slicing is used to slice the program which contains more than one procedure. 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 procedure P calls procedure Q at statement i, the active criteria must first be translated into the context of Q and then recovered once Q has been sliced.

Weiser's approach for interprocedural static slicing involves three separate tasks.

- First, interprocedural *summary information* is computed, . For each procedure P, a set MOD(P) and USE(P) is computed. In both cases, the effects of procedures transitively called by P are taken into account.
- The effect of call-statements on the sets of relevant variables and statements are computed using the summary information. A call to procedure P is treated as a conditional assignment statement '**if** <SomePredicate> **then** MOD(P) := USE(P) where actual parameters are substituted for formal parameters.
- The third part is to generate new slicing criteria with respect to which intraprocedural slices are computed in step (ii). For each procedure P, new criteria are generated for

(i) procedures Q called by P-the criteria consists of all pairs (n_Q, V_Q), where n_Q is the last statement of Q and V_Q is the set of relevant variables in P in the scope of Q (formals are substituted for actuals).It is denoted by DOWN(n_Q, V_Q).

(ii) procedures R that call P-the new criteria consist of all pairs (N_R, V_R) such that N_R is a call to P in R, and V_R is the set of relevant variables at the first statement of P that is in the scope of R (actuals are substituted for formals).It is denoted by UP(N_R, V_R).

4. Slicing of Object-Oriented Programs

Larson and Harrold extended the SDG of Horwitz et al. to compute slicing of object-oriented programs. They have used Class Dependence Graphs (ClDG) for each class in an object-oriented program. A ClDG captures the control and data dependence relationships that can be determined about a class without knowledge of calling environments. Each method in a ClDG is represented by a procedure dependence graph. Each method has a method entry vertex that represents the entry into the method. A ClDG also contains a class entry vertex that is connected to the method entry vertex for each method in the class by a class member edge.

5. Applications of slicing

The main application of slicing was debugging, if a program computes an erroneous value for some variable x at some program point, the bug is likely to be found in the slice with respect to x at that point. A number of other applications are parallelization, program differencing and integration, software maintenance, testing, reverse engineering, and compiler tuning.

5.1 Debugging

Debugging can be a difficult task when one is confronted with a large program, and few clues regarding the location of a bug. Program slicing is useful for debugging, because it reduces the size of search program .If a program computes an erroneous value for a variable x at statement n ,then it assumes that error will be in any one of the statement above it which affect the value of the variable x that is the backward slicing. In this case, it is likely that the error occurs in the one of the statements in the slice. However, it need not always be the case. Forward slices are also useful for debugging. A forward slice show how a value computed at x is being used subsequently, and can help the programmer ensure that x establishes the invariants assumed by the later statement.

5.2 Testing

Slicing helps to decompose programs which in testing, it makes test work faster and more efficient. Slicing based on particular slice criteria. Through this the inter-related modules can be identified, which then can be tested separately without disturbing the rest of the program. Because program slicing helps in understanding programs by dividing it into slices, the task of testing can be allocated to a various testers.

5.3 Regression Testing

The aim of regression testing is to ensure that a change to a software system does not introduce new errors in the unchanged part of the program. Testing effort can be reduced if fewer tests cases are run on a simpler program. Program slicing can be used to partition test cases into those that need to be re-run, as they may have been affected by a change , and those that can be ignored ,as their behavior can be guaranteed to be unaffected by the change. A partition is formed by finding all affected statements: those statements whose backward slice includes a new or edited statement .This set can be efficiently computed as the forward slice taken with respect to the new and edited statements.

5.4 Software maintenance

Maintaining a large software system is a challenging task. Most programs spend 70% or more of their life time in the software maintenance phase where they are corrected and enhanced. Slicing, in the form of decomposition Slicing, reduces the effort required to maintain software. The decomposition slice, taken with respect to variable v from function f, is the union of the slices taken with respect to v at each definition of v and at the end of f.

5.5 Differencing

Program differencing is the task of analyzing an old and a new version of a program in order to determine the set of program components of the new version that represent syntactic and semantic changes. There are two related programs differencing problems:

1. Find all the components of two programs that have different behavior.
2. Produce a program that captures the semantic differences between two programs.

Similar components in both the old and new programs can be obtained by comparing the backward slices of the vertices in old and new's dependence graphs G_{old} and G_{new}. Components whose vertices in G_{old} and G_{new} have isomorphic slices have the same behavior in old and new; thus the set of vertices from G_{new} for which there is no vertex in G_{old} with an isomorphic slice safely approximates the set of components with different behavior. This set is a safe as it is guaranteed to contain all the components with different behavior. Second differencing problem is obtained by taking the backward slice with respect to the set of affected points.

5.7 Reverse Engineering

Reverse engineering concerns the problem of comprehending the current design of a program and the way this design differs from the original design. This involves abstracting out of the source code the design decision and rationale from the initial development and understanding the algorithms chosen.

Program slicing provides a toolset for this type of re-abstraction .For example, a program can be displayed as a lattice of slice s ordered by the is-a-slice-of relation. Comparing the original lattice and the lattice after maintenance can guide an engineer towards places where reverse engineering energy should be spent.

6. References

[1] M. Weiser, "Program Slicing," IEEE Transactions on Software Engineering, Vol. 16, No. 5, 1984, pp. 498-509.

[2] F. Tip, "A Survey of Program Slicing Techniques," Journal of Programming Languages, Vol. 3, No. 3, 1995, pp. 121-189.

[3] D. Binkley and K. B. Gallagher, "Program Slicing," Advances in Computers, Vol. 43, 1996, pp. 1-50.

[4] M. Weiser. Programmers use slices when debugging. *Communications of the ACM*, 25(7):446–452, 1982.

[5] B. Korel and J. Laski. Dynamic program slicing. *Information Processing Letters*, 29(3):155–163, 1988.

[6] K.B. Gallagher and J.R. Lyle. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, 17(8):751–761, 1991.