International Journal of Computer Applications Technology and Research
Volume 3– Issue 10, 634 - 639, 2014, ISSN: 2319–8656

A Comparative Analysis of Slicing for Structured Programs

LipikaJhaK.S.Patnaik
Department of Computer Science Department of Computer Science
and engineering, and engineering,
Birla Institute of Technology,Birla Institute of Technology,
Mesra, Ranchi, IndiaMesra, Ranchi,India

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.In this paper we have discuss the static and dynamic slicing andits comparison by taking number of examples.

Keywords: Slicing techniques, control and data dependence, data flow equation, control flow graph, program dependence graph

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 nonsequential sets of statement. Program is decomposed based on program analysis. A program is analyzed using data flow and control flow. 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 criterion.

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.

Program slicing describes a mechanism or tool which allows the automatic generation of a slice. All statements affecting or affected by the variables V in n mentioned in the slicing criterion becomes a part of the slice.

This paper gives the detail description of slicing techniques and comparison of different slicing. The paper is organized as follows. Section 2 defines some common slicing techniques.Section 3 defines static and dynamic slicing .Section 4 defines comparison of different slicing, section 5 presents conclusion andfinallyacknowledgement and references.

2. SLICING TECHNIQUE

The original concept of a program slice was introduced by Weiser[1].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.

The computation of a slicing is done using data dependence and control dependence. Data dependence and control dependence are defined in terms of the CFG of a program. A control flow graph is a graph which is interpreted as a program procedure. The nodes of a graph represent program statements and edges represent control flow transfers between statements.
A Control Flow Graph for program P is a graph in which each node is associated with a statement from P and the edges represent the flow of control in P. With each node n (i.e., each statement in the program) associate two sets: USE(n), the set of variables whose values are used at n, and DEF(n),the set of variables whose values are defined at n.

There are three main techniques to compute slice:

2.1Data dependence(DD) and control dependence(CD)

Data dependence and control dependence are defined in terms of the CFG of program.A statement *j* is data dependent on statement *i* if a value computed at *i* is used at *j* in some program execution. A data dependence may be defined as; there exists a variable x such that

(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.Node i is control-dependent on j in program P if
1.thereexists a path P from i to j such that j post dominates every node in P.
2. i is not post dominated by j.

S_C={m | n . n ϵ C and m→*n}.
Or
S_c = DD U CD

2.2Data Flow Equation

According to Weiser the slicing is computed by iteratively solving data flow equation.He defines slice as an executable program that is obtained from the original program by deleting zero or more statement. At least one slice exists for any criterion for any program that is the program itself.
The set of relevant variable $R^O_C(i)$ with respect to slicing criterion C=(p,V) is:

1. $R^O_C(i)$=V when i=p
2. $R^O_C(i)$= $(R^O_C(j)$-DEF(i))U (USE(i) if $R^O_C(j)\cap DEF(i)\neq\emptyset$)

The set of relevant statements to C denoted S^0_C,is defined as:
S_c^0={i | Def(i) ∩ R_c^0(j) ≠ ø ,i → ^{CFG} j}
S_c^0={3,4,9}
The set of conditional statements which control the execution of the statements in S^0_C,denoted B^0_C is defined as:

B^0_c={bϵG|Infl(b) ∩ S^0_C≠ ø }

Infl(5)={6,7,8}
The sliced program S_c is defined recursively on the set of variables and statements which have either direct or indirect influence on V. Starting from zero, the superscripts represent the level of recursion.

R_c^{i+1}(n)= R_c^i(n)$\cup_{b\in B_c^i} R^o_{b,\cup(b))}$(n)
B_c^{i+1}={bε G|INFL(b) ∩ $S_c^{i+1} \neq ø$ }
S_c^{i+1}={n ε G|DEF(n) ∩ R_c^{i+1}(j) ≠ ø}U B_c^i

The sliced program includes the conditional statements with an indirect influence on a slice,the control variables which are evaluated in the logical expression , and the statements which influence the control variables.

2.3Program 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 EXIT vertex.The edges of the PDG represent the control and flow dependence induced by the CFG.

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

It includes all the statements that affect variable v or affected by the variable at the point of interest (i.e., at the statement x). It is computed by finding consecutive sets of indirectly relevant statements, according to data and control dependencies. Static slicing criterion consists of a pair (n, V)where n is point of interest and V is a variable in a program based on which program will be sliced.

3.2 Dynamic Slicing

A dynamic program slice 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. In static slicing, only statically available information is used for computing slices.

4. COMPARISON USING DATA FLOW EQUATION

Example 1: Let us consider a program which computes the sum and product of first n numbers, using a single loop.

```
void main()
    1.  {int n;
    2.  cin>>n;
    3.  if (n>0)
    4.  int i=1;
    5.  int sum=o;
    6.  int product=1;
    7.  int k;
    8.    while(i<=n)
    9.    {cin>>k;
    10.     sum=sum+k;
    11.     product=product*k;
    12.      i=i+1;}
    13. cout<<sum;
    14. cout<<product;
        }
```

Static slicing: Slicing criterion C is (14, product)

Statement	USE()	DEF()	R^o_c	R_c^1
1		n	Ø	Ø
2		n	Ø	Ø
3	N		Ø	N
4		i	Ø	i,n
5		sum	Ø	i,n
6		product	Ø	product,i,n
7		k	product	product,i,n
8	i,n		product	product,i,n
9		k	product	product,i,n,
10	sum,k	sum	product, k	product,i,n,k
11	product, i	product	product, k	product,i,n,k
12	I	i	product	product,i,n
13	Sum		product	Product
14	Product		product	Product

S_C^0= {6,9,11}

B_C^0={3,8}
R_C^1(n)=R_C^0(n)$\cup_{b\in<3,8>} R^o_{b,\cup(b))}$(n)
$R^o_{3,n}$(3)=n $R^o_{3,n}$(14)=n $R^o_{3,n}$(13)=n
$R^o_{3,n}$(12)=n $R^o_{3,n}$(11)=n
$R^o_{3,n}$(10)=n $R^o_{3,n}$(9)=n $R^o_{3,n}$(8)=n
$R^o_{3,n}$(7)=n $R^o_{3,n}$(6)=n $R^o_{3,n}$(5)=n
$R^o_{3,n}$(4)=n $R^o_{3,n}$(2)= Ø $R^o_{3,n}$(1)= Ø
$R^o_{8,<i,n>}$(8)=i,n $R^o_{8,<i,n>}$(12)=i,n
$R^o_{8,<i,n>}$(11)=i,n $R^o_{8,<i,n>}$(10)=i,n
$R^o_{8,<i,n>}$(9)=i,n $R^o_{8,<i,n>}$(7)=i,n
$R^o_{8,<i,n>}$(6)=i,n $R^o_{8,<i,n>}$(5)=i,n
$R^o_{8,<i,n>}$(4)=i, $R^o_{8,<i,n>}$(3)= Ø
$R^o_{8,<i,n>}$(2)= Ø $R^o_{8,<i,n>}$(1)= Ø
S_C^1={12,11,9,6,4,2}
B_c^1={bε G|INFL(b) ∩ S_c^{i+1} ≠ ø }
B_c^1={3,8}
S_c^{i+1}={n ε G|DEF(n) ∩ R_c^{i+1}(j) ≠ ø}U B_c^i
S_c^1={2,3,4,6,8,9,11,12}

Dynamic slicing: Slicing criterion C:(14,product,n=2)

```
voidmain ()
1.  {int n;
2.  cin>>n;
3.  if (n>0)
4.  int i=1;
5.  int sum =1;
6.int product=1;
7.int k;
8[1].  i<=n
9[1].cin>>k;
10[1].   sum=sum+k;
11[1].product=product*k;
12[1].i=i+1;}
8[2].  i<=n
9[2].      cin>>k;
10[2].    sum=sum+k;
11[2].    product=product*k;
12[2].     i=i+1;
8[3].  i<=n
13. sum=sum+k;
14. cout<<product;
    }
```

DU:{(2,3),(2,8[1]),(2,8[2]),(2,8[3]),(4,8[1]),(4,12[1]),(12[1],8[2]))
(12[1],12[2]),(12[2],8[3]),(6,11[1]),(11[1],11[2]),(11[2],14),(9[1],11[1]),(9[2],11[2])

TC:
{3,4},{3,5},{3,6},{3,7}{3,8[1]},{(8[1],9[1]),(8[1],11[1]),(8[1],12[1]), (8[2],9[2]),(8[2],11[2]), (8[2],12[2]) }
IR: {(8[1],8[2]), (8[1],8[3]), (8[2],8[1]), (8[2],8[3]), (8[3],8[1]), (8[3],8[2])

S[0]={11[2]}
A[1]={11[1],9[2],8[2]}
S[1]= {11[2], 11[1], 9[2], 8[2]}
A[2]={6,9[1],8[1],2,8[3],12[1]}
S[2]={11[2], 11[1], 9[2], 8[2], 6,9[1],8[1],2,8[3],12[1]}
A[3]={4, 12[1],12[2]}
S[3]= {11[2], 11[1], 9[2], 8[2], 6,9[1],8[1],2,8[3], 4, 12[1],12[2]}

Example 2: Let us consider a program which computes the sum and product of first n numbers, using for loop.

```
void main()
1.  {int n;
2.  cin>>n;
3.  if (n>0)
4.  int sum=0;
5.  int product=1;
6.  int k;
7.   for(i=1;i<=n;i++)
8.   {cin>>k;
9.      sum=sum+k;
10.     product=product*k;
        }
11. cout<<sum;
12. cout<<product;
    }
```

Static slicing: Slicing criterion C is (12, product)

Statement	USE()	DEF()	R_c^0	R_c^1
1		n	Ø	I
2		n	Ø	I
3	N		Ø	i,n
4		sum	Ø	i,n
5		product	Ø	i,n
6		k	Product	product,i,n
7	i,n	i	Product	product,i,n
8		k	Product	product,i,n

				,
9	sum,k	sum	product, k	product,i,n ,k
10	product, k	product	product, k	product,i,n ,k
11	Sum		product	Product
12	Product		product	Product

S_C^0= {11,6,4}
B_C^0={7,8,9}
$R_{7,n}^o$(3)=n $R_{7,n}^o$(11)=product,n
$R_{7,n}^o$(10)=product,k,n
$R_{7,n}^o$(9)=product,k,n$R_{7,n}^o$(8)=product,n
$R_{7,n}^o$(7)=product,n $R_{7,n}^o$(6)=product,n
$R_{7,n}^o$(5)=n
$R_{3,n}^o$(4)=n $R_{7,n}^o$(2)= Ø $R_{3,n}^o$(1)= Ø
$R_{7,<i,n>}^o$(7)=i,n $R_{7,<i,n>}^o$(11)=i,n
$R_{7,<i,n>}^o$(10)=i,n$R_{7,<i,n>}^o$(9)=i,n
$R_{7,<i,n>}^o$(8)=i,n$R_{7,<i,n>}^o$(6)=i,n $R_{7,<i,n>}^o$(5)=i,n
$R_{7,<i,n>}^o$(4)=i,n $R_{7,<i,n>}^o$(3)=i,n
$R_{7,<i,n>}^o$(2)=i $R_{7,<i,n>}^o$(1)=i
S_C^1={10,8,7,5,2}
B_c^1={3,7}
S_c ={2,3,5,7,8,10}

Dynamic slicing: Slicing criterion C:(12,product,n=2)

```
void main ()
    1.  {int n;
    2.  cin>>n;
    3.  if (n>0)
    4.  int sum =1;
    5.  int product=1;
    6.  int k;
    7[1]. i=1; i<=n
    8[1].    cin>>k;
    9[1].   sum=sum+k;
    10[1].    product=product*k;
    7[2].  i<=n
    8[2].     cin>>k;
    9[2].    sum=sum+k;
    10[2].    product=product*k;
    7[3].  i<=n
    11. cout<< sum;
    12. cout<<product;
        }
```

DU: {(2,3),(2,7[1]),(2,7[2]),(2,7[3]),(4,9[1]),(9[1],9[2]), (9[2],11),(5,10[1]),(10[1],10[2]),(10[2],12),(8[1],9[1]),(8[1],10[1]), (8[2],9[2]), (8[2],10[2])}
TC:
{(3,4),(3,5),(3,6),(3,7[1]),(7[1],8[1]),(7[1],9[1]),(7[1],10[1]), (7[2],8[2]),(7[2],9[2]), (7[2],10[2]) }
IR: {(7[1],7[2]), (7[1],7[3]), (7[2],7[1]), (7[2],7[3]), (7[3],7[1]), (7[3],7[2])

S[0]= {10[2]}
A[1]= {10[1], 8[2], 7[2]}
S[1]= {10[2], 10[1], 8[2], 7[2]}
A[2]= {5, 8[1], 7[1], 2, 7[3],10[1]}
S[2]= {10[2], 10[1], 8[2], 7[2], 5, 7[1], 8[1], 2, 7[3]}
A[3]= {3}
S[3]= {10[2], 10[1], 8[2], 7[2], 5, 7[1], 8[1], 2, 7[3],3}

5. CONCLUSION

The concept of program slicing was originally defined by Weiser as a method for decomposing a program into pieces, i.e., slices. The static slice produced by his method is of the partially equivalent program type. His solution produces a static program slice based on solving data flow equations iteratively on the flow graph of the program. Here in this paper we have used the same concept for computing the static and dynamic slicing. We have taken the number of example and analyzed the slicing .As the number of control statement increases in a program it becomes difficult to compute the control dependent statement and then the relevant variable. In for loop we can either consider the different statement for the three section or we can consider a single statement. The output will be same the only difference is that it will be easy to compute from the different statement.

6. ACKNOWLEDGEMENT

I wish to convey my sincere gratitude and appreciation to each and every person who helped me in writing this paper. I am grateful to my institution, Birla Institute of Technology and my colleagues. I would especially like to thank Dr. K. S. Patnaik, my guide for his advice and guidance.

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

[5] M. Kamkar, "An overview and comparative classification of program slicing techniques," J. Systems Software, vol. 31, pp. 197–214, 1995.