International Journal of Computer Applications (0975 – 8887)
Volume 56– No.14, October 2012

Evaluation of Flow Graph and Dependence Graphs for Program Representation

Vinay Arora
Thapar University,Patiala
CSED

Rajesh Kumar Bhatia
DCR Uni. of Sc. Tech, Murthal
CSED

Maninder Singh
Thapar University, Patiala
CSED

ABSTRACT
Graphical methods offer the structural icon of the system that facilitates testing the logical progress of the program. A control flow graph describes the sequence in which the instructions of a program will get executed. PDG represents a program as a graph where statements and predicate expressions can be characterized by the nodes. The System Dependence Graph (SDG) is an extension of the Program Dependence Graph (PDG) and represents a program that consists of multiple procedures and involves procedural calls. An assessment of flow graphs & dependence graphs can be performed on the basis of properties like control dependence, data dependence, transitive dependence, flow sensitivity, parameter passing etc.

General Terms
Flow graph, dependence graph.

Keywords
Control flow graph, program dependence graph, system dependence graph.

1. INTRODUCTION
A graph G = (N, E) is defined as a finite set of nodes N and a finite set of edges E. The graphical methods offer the structural icon of the system that facilitates testing the logical progress of the program [1]. Flow graph indicates the flow of control between statements present in a program whereas a dependence graph symbolizes program features and dependencies between many objects. There are many graphical representations such as Data Flow Graph (DFG), Program Dependence Graph (PDG), System Dependence Graph (SDG), Extended System Dependence Graph (ESDG), Call-based Object-Oriented System Dependence Graph (COSDG), etc. Section 2 provides a brief review of various graph based approaches followed for program representation. Section 3 provides the assessment of flow graphs & dependence graphs on the basis of properties like control & data dependence, transitive dependence, flow sensitivity, parameter passing etc. Section 4 presents the conclusion for comparison between various graphical representations. Section 5 outlines the future scope.

2. PROGRAM REPRESENTATION USING GRAPHS

2.1 Control Flow Graph
A control flow graph is a directed graph where nodes correspond to the basic blocks (set of statements in a program) and the edges represent control flow paths [2]. For example, in Fig 1, blocks (nodes) are labeled such that block bi corresponds to node ni. An edge (i, j) connecting basic blocks bi and bj implies that control can go from block bi to block bj [4].

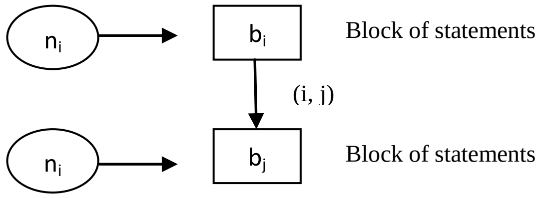

Fig 1 Block/Node representing set of statements [5]

Most of the programs are constructed with the three types of constructs namely sequence, selection and iteration. Fig 2 summarizes how the CFG for these three types of constructs can be drawn. The CFG representation of the sequence and decision types of statements is straightforward. For the Iteration type constructs such as the while construct, the loop condition is tested only at the beginning of the loop and therefore control flows from the last statement of the loop to the top of the loop [6].

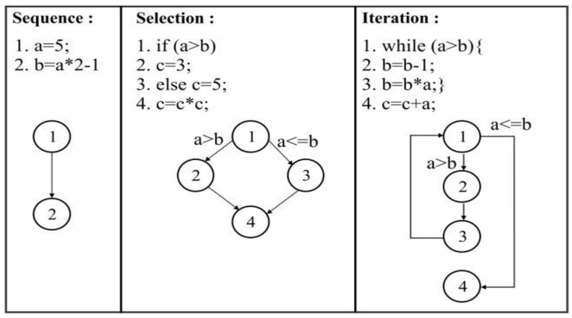

Fig 2 CFG for sequence, selection and iteration construct [5]

For programs written in Pascal Frankl et al introduced control flow and data flow testing criteria and also defined a new family of adequacy criteria called feasible data flow testing criteria which has been derived from the data flow testing criteria [7]. To find subsets of nodes and edges in a flow graph branch coverage and testing of programs has been proposed by Agrawal [8]. The author has introduced dominator relationships among super blocks which can be used to identify a subset of the super blocks and these techniques reduce object code size, runtime overhead and cost of coverage testing of programs. Dominator relationships were represented using Block Dominator Graph and Edge Dominator Graphs. An algorithm to construct Global Dominator Graph has also been presented which shows dominator relationships among mega blocks at inter-procedural level. Global dominator graph is the combination of block dominator graphs. Inter-procedural jump statements can also be handled using this graph.

An Event Graph is an extension of control flow graph in which interactions can be represented between the program units such as procedures and functions. The Event Inter Actions Graph (EIAG) is used as a model for concurrent programs and it constitutes an Event Graphs and various Interactions. A Class Specification Implementation Graph (CSIG) is a graphical representation which shows a class from two distinct perspectives, namely the class as specified and the class as implemented. In this graph, each class method can be represented by two control flow graphs where one graph visualizes at control flow as specified and other graph at control flow as implemented. Beydada and Gruhn presented the application of CSIGs in regression testing. The control flow graphs are the main constituents of a CSIG and therefore testing techniques implemented on control flow graphs can also be combined with CSIGs with some modifications.

For capturing the semantic inter-relations of aspect-related interactions for AspectJ software a Java Interclass Graph (JIG) is a new control flow representation. A JIG contains a CFG for each method which is internal to the set of classes [9]. Each call site is expanded into a call node and a return node where call node is linked with the entry node of the called method. There is a path edge between the call node and the return node to represent the path through the called method.

2.2 Program Dependence Graph

The PDG represents a program as a graph where statements and predicate expressions can be characterized by the nodes. The edges incident on to a node represent data values on which the node's operations depend and the control conditions on which the execution of the operations depends [10, 11]. A PDG can represent both control dependence as well as data dependence in a single graph.

For statements X and Y in a program, if X is control dependent on Y then there must be at least two paths out of Y. In this, one path always causes X to be executed and the other path may result in X not being executed. A data dependence exists between statements X and Y in a program if X defines a variable v, Y uses v and there is a path from X to Y in the program on which v is not defined [12, 13].

A program dependence graph contains a flow dependency edge from vertex v1 to vertex v2, iff all of the following conditions hold [10, 11]:

- v1 is a vertex that defines variable x and v2 is a vertex that uses x.
- Control that reach v2 after v1 through an execution path in which there is no intervening definition of x.

Consider the program given in Fig 3 where the code fragment is used to calculate the factorial of a number [14]. The execution of statements 11 and 12 is dependent on the control predicate at statement 9. The statement 11 is data dependent on statements 7, 8, 12 and itself. Fig 4 represents the corresponding Program Dependence Graph of the program given in Fig 3, where control dependence edges are represented as bold lines and data dependence edges are represented by light colored regular lines.

```
1.   class Factorial
2.   {
3.     public static void main(String args[ ])
4.     {
5.       int fact;
6.       int n;
7.         n =4;
8.         fact = 1;
9.           while (n != 0)
10.            {
11.              fact = fact * n;
12.              n = n-1;
13.            }
14.     }
15. }
```

Fig 3 A Sample Java Program to calculate factorial of a number [14]

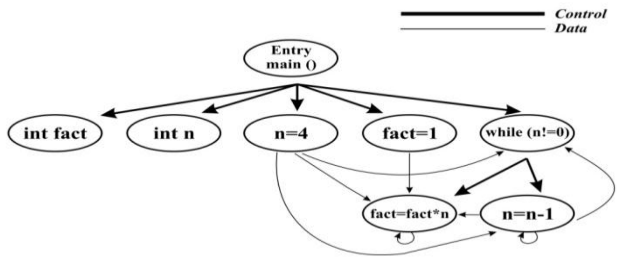

Fig 4 The Program Dependence Graph corresponding to the program in Fig 3 [14]

Rothermel and Harrold implemented PDG for regression testing in object-oriented software. The researchers presented an algorithm that constructs class dependence graphs (ClDG) for classes and application programs. The researchers used these graphs to determine which tests can cause a modified class to produce different output than the original [12, 15]. But the researchers did not considered polymorphism and dynamic binding in their approach. There were also few other graphs that extended the features of PDG for program slicing such as Object Program Dependence Graph (OPDG) [16], Dynamic Object Program Dependence Graph(DOPDG) [16], Efficient Dynamic Object Program Dependence Graph (EDOPDG) [17] and so on.

2.3 System Dependence Graph

The System Dependence Graph (SDG) is an extension of the Program Dependence Graph (PDG) and represents a program that consists of multiple procedures and involves procedural calls. SDG models a language in which parameters are passed by value and where a complete system consists of a single (main) program and a collection of auxiliary procedures [11]. Each Procedure Dependence Graph contains an entry vertex that represents entry into the procedure. To model parameter passing, SDG associates each procedure entry vertex with formal-parameter vertices namely a formal-in vertex for each formal parameter of the procedure and a formal-out vertex for each formal parameter that may be modified by the procedure [18]. SDG associates each call-site in a procedure with a call vertex and a set of actual parameter vertices with 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. A call edge connects a call vertex to the entry vertex of the called procedure's dependence graph. Parameter-in and parameter-out edges represent parameter passing. Parameter-in edges connect actual-in and formal-in vertices and parameter-out edges connect formal-out and actual-out vertices [15].

```
public static void main(String args[])
{
int i = 1;
int sum = 0;
while (i<11) {
        sum = add(sum, i);
        i = add(i, 1); }
System.out.println("sum = \n" + sum);
System.out.println("i =\n" +  i);
}
static int add(int a, int b)
{
return (a+b);
}
```

Fig 5 An Example Program [19]

Fig 5 depicts a program to find sum of numbers from 1 to 10. This program uses two methods namely main( ) and add( ) [19]. Fig 6 shows the System Dependence Graph of this program representing the flow within two methods. As the Program Dependence Graph can only represent the flow in a single procedure but System Dependence Graph is able to represent multiple procedures.

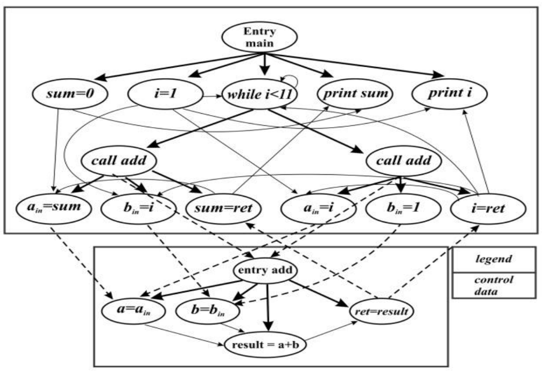

Fig 6 The System Dependence Graph of the example program in Fig 5 [19]

Horwitz et al. presented SDG for inter-procedural slicing and applied context-free grammer for creating SDG [11]. They presented all dependency relationships using SDG and PDG. Larson et al. extended the SDG of Horwitz et al. to signify Object-Oriented programs [15]. They had built 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 CIDG is represented by a procedure dependence graph. The CIDG construction expands each method entry by adding formal-in and formal-out vertices similarly as procedure dependence graphs. Liang et al. presented an SDG for object-oriented software that is more precise and efficient than previous approaches [18]. Based on this new SDG, they introduced the concept of object slicing and an algorithm to implement this concept. Mohapatra et al. presented a technique for dynamic slicing of Object-Oriented programs, which extends the System Dependence Graph (SDG) [20]. The graph is known as Extended System Dependence Graph (ESDG) that handles the features of object oriented programs such as polymorphism, inheritance etc. Their algorithm is named as Edge Marking Dynamic Slicing (EMDS) because it is based on marking and unmarking the edges of the ESDG. Zhao presented a Java-based graph that encapsulates the benefits offered by the earlier approaches of SDG. The Graph was named as Java System Dependence Graph (JSDG) and it enables the representation of Java-specific features such as interfaces, packages and single inheritance [21]. Walkinshaw et al. extended this Java-based graph that is known as Java System Dependence Graph (JSysDG) [22]. A JSysDG is a multi-graph that maps out the control and data dependencies among the statements of a Java program. Xi et al. presented an approach of Coarse-grained Dynamic Slice for Java Program [23]. This technique uses AspectJ code tracing tactic to gather method execution traces, which comprises information of method calls. Dynamic Java System Dependence Graph (DJSDG) is used for the intermediate representation and the slice computation has also been implemented on this graph.

3. ASSESSMENT OF FLOW GRAPH & DEPENDENCE GRAPHS

From the given literature, it has been appraised that each and every graphical method for program representation presented above supports some unique features/parameters. On the basis of parameters like procedural call, slicing, sensitivity, exception handling, test case generation etc. the comparative analysis of Control Flow Graph (CFG), Program Dependence Graph (PDG) and System Dependence Graph (SDG) has been done to better understand the usage of these graphs [5]. Control Flow Graph (CFG) can be taken as base graph for various other representations like BDG, EDG, EG, CCFG, etc. Similarly, Program Dependence Graph (PDG) can be taken as base graph for ClDG, OPDG, DOPDG, etc. System Dependence Graph extends the features of PDG and can be taken as base graph for other graphs like ESDG, JSDG, OSDG, etc.

4. CONCLUSION

From the given literature this has been listed out that system dependence graph is a feature rich representation as compare to flow graph that supports features like control, data and transitive dependence, single & multiple procedure, inter & intra procedure calls, multiple types of edges, slicing, context sensitivity, inheritance & polymorphism, test case generation and parameter passing. Whereas flow graph be deficient in representing data & transitive dependence, multiple procedures, inter & intra procedure calls, multiple types of edges, slicing, context sensitivity, inheritance & polymorphism etc.

5. FUTURE SCOPE

System dependence graph can be extended further for incorporating the features like exception handling & flow sensitivity.

Table 1: Comparison of CFG, PDG and SDG (y/n denotes presence/absence of the feature in the corresponding graph) [5]

S No.	Parameters	CFG	PDG	SDG
1.	Control Dependency [2,4,6,7,10,13,24]	**y**	**Y**	**y**
2.	Data Dependency [10,13,18,24]	n	**Y**	**y**
3.	Transitive Dependency [11,14,15,18]	n	N	**y**
4.	Single Procedure [2,4,10,11,12,15]	**y**	**Y**	**y**
5.	Multiple Procedures [2,10,11,12,25]	n	N	**y**
6.	Intra-procedural Calls [10,11,13,14,15]	n	**Y**	**y**
7.	Inter-procedural Calls [10,11,13,14,15,16,25]	n	N	**y**
8.	Multiple types of Edges [4,8,10,13,25]	n	**Y**	**y**
9.	Slicing [10,11,12,13,14,25]	n	**Y**	**Y**
10.	Flow-Sensitive [4,25]	**y**	**Y**	N
11.	Context-Sensitive [4,25]	n	N	**Y**
12.	Inheritance & Polymorphism [10,15,16,24,26]	n	N	**Y**
13.	Dynamic Binding [12,13,14,15,16,18]	n	N	N
14.	Test Case Generation [2,7,10,11,13]	**y**	**Y**	**Y**
15.	Exception Handling [25,26]	n	**Y**	N
16.	Parameter Passing [14,15,21,24,27]	n	N	**Y**

Table 2: Description of Graphs shown in Fig 7 [5]

S No.	Acronym	Abbreviation	Description
1.	CFG	Control Flow Graph	Flow between nodes & edges
2.	BDG	Block Dominator Graph	Dominator relationship among blocks
3.	EDG	Edge Dominator Graph	Dominator relationship among edges
4.	GDG	Global Dominator Graph	BDG + EDG + Inter-procedural level
5.	EG	Event Graph	CFG + Interaction between procedures
6.	EIAG	Event InterActions Graph	EG + Interaction for concurrent programs
7.	CCFG	Class Control Flow Graph	CFG + Call Graph between classes
8.	ICCFG	Inter-Class Control Flow Graph	CFG + Inter-class relationship
9.	CSIG	Class Specification Implementation Graph	CFG for each class method
10.	JIG	Java Inter-class Graph	Inter relationship of AspectJ programs
11.	PDG	Program Dependence Graph	Control + Data dependencies for single procedure
12.	CDS	Control Dependence Sub-graph	Control dependencies for single procedure
13,	DDS	Control Dependence Sub-graph	Data dependencies for single procedure
14.	OPDG	Object Program Dependence Graph	PDG + Object-Oriented Features
15.	DOPDG	Dynamic Object Program Dependence Graph	OPDG + Dynamic Slicing
16.	EDOPDG	Efficient Dynamic Object Program Dependence Graph	DOPDG + few modifications
17.	ClDG	Class Dependence Graph	Set of PDGs + Inter-procedural calls within class
18.	SDG	System Dependence Graph	Set of PDGs + Interprocedural calls for whole sys.
19.	ESDG	Extended System Dependence Graph	Extends SDG by representing a class with ClDG
20.	JSDG	Java System Dependence Graph	SDG + Interfaces& Packages in Java
21.	JSysDG	Java System Dependence Graph	JSDG_ few modifications
22.	DJSDG	Dynamic Java System Dependence Graph	JSDG + Dynamic Slicing
23.	COSDG	Call-based Object-oriented System Dependence Graph	Dependencies + Flow + CallGraph + Inherited Call + Polymorphic calls

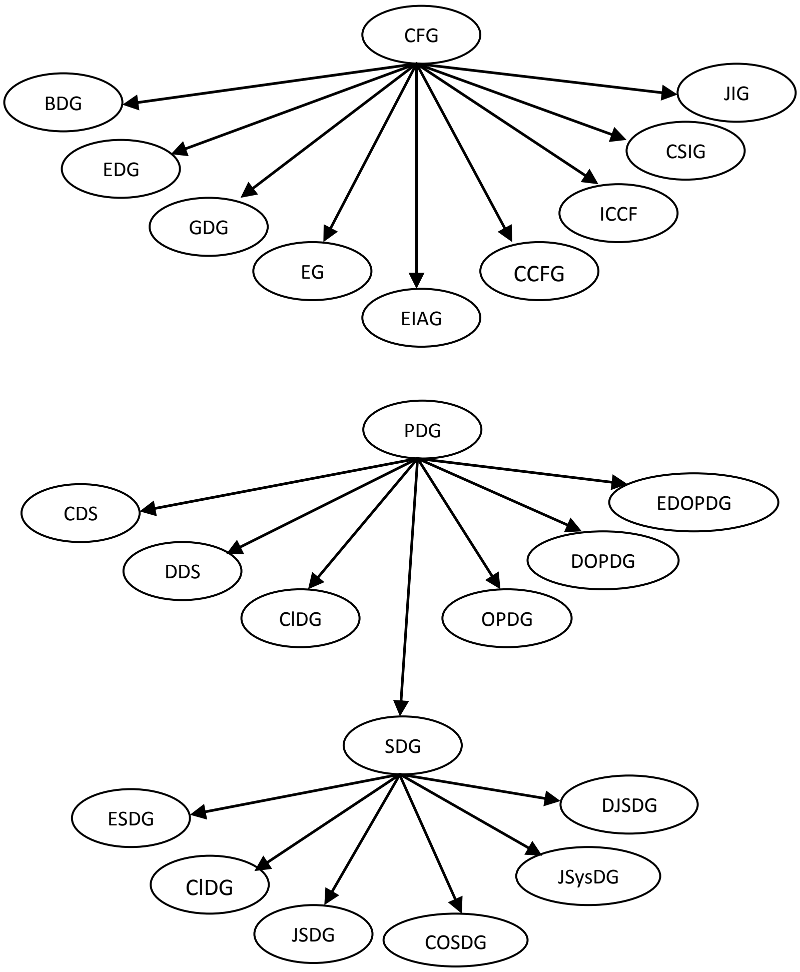

Fig 7: CFG, PDG, SDG and their sub-graphs [5]

6. REFERENCES

[1] M. E. Paige, "On partitioning Program Graphs", IEEE Transactions on Software Engineering, vol. se-3, no. 6, pp. 386-393, November 1977.

[2] Frances E. Allen, "Control Flow Analysis", In Proceedings of a Symposium on Compiler optimization, ACM SIGPLAN Notices, vol. 5, July 1970.

[3] Shveta Verma, Vinay Arora, "Survey on Graphical Methods for Test Case Generation", International Journal of Mobile & Adhoc Network (IJMAN), vol. 2, issue 2, pp. 257-264, May 2012

[4] Robert Gold, "Control Flow Graph and Code Coverage", Int. J. Appl. Math.Computer Sci., vol. 20, no. 4, pp. 739–749,2010.

[5] M.E. Thesis on "Comparative Analysis of Flow Graph and Dependence Graphs", Shveta Verma, Vinay Arora, Thapar University, Patiala.

[6] R. Mall, Fundamentals of Software Engineering, Prentice Hall of India.

[7] P. G. Frankl and E. J. Weyuker, "An Applicable Family of Data Flow Testing Criteria", IEEE Transactions on Software Engineering, vol. 14, no. 10, October 1988.

[8] HiralalAgrawal, "Dominators, Super Blocks and Program Coverage", In Proceedings of the 21st ACM SIGPLAN-SIGACT symposium on Principles of programming language, 1994.

[9] G. Xu and A. Rountev, "Regression Test Selection for AspectJ Software", In 29th International Conference on Software Engineering, pp. 65-74, 2007.

[10] J. Ferrante, J. Worren and K. Ottenstein, "The Program Dependence Graph and its use in optimization", In ACM Transactions on Programming Languages and Systems, vol. 9, no. 3, pp. 319-349, July 1987.

[11] S. Horwitz, T. Reps and D. Binkley, "Interprocedural slicing using dependence graphs," In ACM Transactions on Programming Languages and Systems, vol. 12, pp. 26-60, January 1990.

[12] G. Rothermel and M. J. Harrold, "Selecting regression tests for Object-Oriented Software", In Proceedings of International Conference on Software Maintenance, pp. 14-25, IEEE transactions, 1994.

[13] K. Tewary and M. J. Harrold, "Fault Modeling using the Program Dependence Graph", In Proceedings of 5th International Symposium on Software Reliability Engineering, pp. 126-135, IEEE Transactions, 1994.

[14] P. E. Livadas andS.Croll, "System Dependence Graphs Based on Parse Treesand their Use in Software Maintenance", IEEE Transactions, 2005.

[15] L. Larsen and M. J. Harrold, "Slicing object-oriented software", In 18th International Conference on Software Engineering, pp. 495–505, Mar. 1996.

[16] B. Xu, Z. Chen and H. Yang, "Dynamic slicing object-oriented programs for debugging", In the Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation, pp. 115 – 122, 2002.

[17] S.Park, "Efficient Dynamic Slicing of Object Oriented Programs", Dept. of R&D, Korea Micro System, pp.143-721, January2003.

[18] D. Liang and M.J. Harrold, "Slicing objects using System Dependence Graphs", International Conference on Software Maintenance, pp. 358-367, November 1998.

[19] TONG Chun Yin under the supervision of Dr. LO Eric Chi Lik and Mr. LUK Ming Hay , " Java System Dependence graph API ", Department of computing, The Hong Kong polytechnic University, 2010, Available at: http://www.comp.polyu.edu.hk/~csllo/teaching/SDGAPI (accessed 25/4/2012).

[20] D.P.Mohapatra, R.Mall, and R.Kumar, "A node marking dynamic slicing technique for object-oriented programs", In Proceedings of Workshop on Software Development and Architecture, Bangalore, pp.1 – 15, January 2004.

[21] J. Zhao, "Applying program dependence analysis toJava software," in Proc. Workshop on Software Engineeringand Database Systems, (Taiwan), pp. 162–169, December 1998.

[22] Neil Walkinshaw,Marc Roper, Murray Wood, "The Java System Dependence Graph", Proceedings of the Third IEEE International Workshop on Source Code Analysis and Manipulation, 2003.

[23] Liu Xi, Miao Li, Zhao Dan, Li Wei, "An approach of coarse-grained dynamic slices for Java programs", IEEE 3rd International Conference on Communication Software and Networks, pp.670 – 674, May 2011.

[24] E S F Najumudheen, R. Mall, D. Samanta, "A Dependence representation for coverage testing of object-oriented programs", Journal of Object Oriented Technology (JOT), Vol. 9, No. 4, pp. 1-23, 2010.

[25] C. Hammer, J. Krinkle and G. Snelting, "Information Flow Control for Java based on Path Conditions in Dependence Graphs", In Proceedings of IEEE International Symposium on Secure Software Engineering, Virgina, USA, 2007.

[26] V. Martena, A. Orso and M. Pezze, "Interclass Testing of object oriented software", In Proceedings of 8th IEEE International Conference on Engineering of Complex Computer Systems, pp. 135-144, 2002.

[27] E S F Najumudheen, R. Mall, D. Samanta, "A Dependence Graph-based Test Coverage Analysis technique for Object-Oriented programs", In 6th International Conference on Information Technology: New Generations, IEEE, 2009.