ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 3, March 2013

Application of Formal Analysis Techniques for Monolithic Application Source Code Analysis

Asfa Praveen[1], Shamimul Qamar[2], Shahanawaj Ahamad[3]

Ph.D. (Computer Sc.) Research Scholar, Faculty of Science & Technology, Shri Venkateshwara University, Gajraula, (U.P.), India[1]

Professor of Electronics & Computer Engineering, Noida Institute of Engineering & Technology, Greater Noida, (U.P.), India[2]

Assistant Professor, Dept. of Comp. Sc. & Software Engg, College of Computer Sc. & Engineering, University of Ha'il, K.S.A[3]

Abstract: For the development of monolithic legacy applications to fulfil the updating requirements and further maintenance work, the programs of legacy has to be analyses thoroughly with many aspects and to achieve many intended objectives as SOA based migration of monolithic legacy software though service orientation of program and its related environment with services. In this direction, the legacy source code analysis is one the beginning task after assets assessment, this paper is intended to present some of the applied techniques for analysis of program features and source code, with maintaining the aspect of design recovery.

Keywords: Monolithic Application, Program Analysis, Legacy, Source Code, Concept Analysis.

I. INTRODUCTION

Many issues make monolithic application source code too difficult to understand and maintenance tasks hard to perform. The work presented here combines three techniques, with the goal to achieve on their entire strengths and overcoming their shortcomings. Program representation formalism which is called Lattice of Concept Slices and program modularization techniques is to separate statements in a code fragment as per the concept implemented. The goal of applications of analysis techniques is to achieve modularization because modules are self-contained, free from any side effect and the duplicate code is less. A domain concept is a structural pattern, uses of a variable, call to a method, regular expression matching on variable naming etc in analysis. This modularization procedure is illustrated with the help of an example C program. Analysis techniques such as concept assignment, formal concept analysis, and program slicing have largely been applied and used by reengineers for program modularization. The study has undertaken for their role in analysis and way of implementations.

II. CONCEPT ASSIGNMENT

Biggerstaff [1] presented the Concept Assignment problem for the identification of human oriented domain concepts for assigning them to implementation oriented source code within a program. There are two methods for identifying domain concepts: (i) the structural analysis (ii) the probable reasoning.

Parsing techniques is the basis for structural analysis and the domain concept is defined as a structural pattern, and is based on use of variables, calls to methods etc. The source code is parsed to match the signature of the pattern then matching lines of source code are considered to be incorporated part of domain concept. The atomic concepts are recognized in early stage first and then concepts are identified.

Probable reasoning is based on informal information, heuristics, thumb rules, weight of build up proof and so forth, many examples based studies have shown that systems of probable reasoning is based on concept assignment [2] with domain model which works as an adaptive observer [1]. The process uses a knowledge base that contains a list of domain concepts implemented in the program and their indicators. The indicators can be identifiers, keywords, comments, regular expressions etc. In the hypothesis generation stage the source code is taken as input and scanned through to generate hypotheses of domain concepts and based on the knowledge base. The hypotheses are then sorted by the indicator position in the source code. In the segmentation stage the sorted hypotheses are analysed to group them into segments using an unsupervised competitive learning neural network. The output of the stage is a collection of segments each containing a number of hypotheses. In the concept binding stage the segments hypotheses are analysed to identify the most evident concept. The

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 3, March 2013

segments are then labelled with their corresponding domain concepts. Code segments corresponding to the domain concepts are candidates for modules.

III. PROGRAM SLICING

Slicing as originally described by Weiser [3] is an abstraction of a program based on a particular behaviour. A slice is defined to be an executable subset of the original program that preserves the original behaviour of the program with respect to a slicing criteria <*P*, *V*>, which is a given variable *V* at a given program point *P*. The slice will consist of all the statements of the program that may affect the value of *V* at point *P*. The original slicing algorithm was based on statement deletion using data flow analysis. More widely used algorithms [4], [9] work on the Dependence Graph of the program. First, a program dependence graph (PDG) [5], [6] is created for the program at hand. Some additional nodes are inserted at the start of the PDG to correspond to the initial definitions of all variables used in the program without first being defined and at the end to correspond to the final uses of all the variables. The algorithm starts by traversing the PDG from the node to the program point *P* and then traces back to all the nodes that has a direct or indirect control or data flow dependency on this node. All the visited nodes are marked. All the unmarked nodes are deleted. The program has resulting PDG is the computed slice. This type of slicing is known as static intra-procedural slicing. [7], [8] gives a comprehensive list of all the slicing variations and techniques. Slicing has the advantage that the slices are self contained and executable by themselves. But the problem of slicing is that the decomposition is done based on very fine-grained program variables instead of domain concepts. Modularization based on slicing may result into modules that contain a significant amount of duplicated code because of overlapping control flows. Moreover, even though each of the decompositions is self-contained, if the duplicated code modifies global program resources it may cause significant and undesirable side effects when deployed in separate modules.

IV. FORMAL CONCEPT ANALYSIS

Formal Concept Analysis is a mathematical tool used for identifying groupings of objects that have common attributes and representing them in a lattice structure to show the generalization specialization relationship among the groups. Concept analysis starts with a context (*O*, *A*, *R*), a binary relation *R* between a set of objects *O* and their attributes *A*. A concept *C* (*E, I)* is a maximal collection of objects *E* (the extent) sharing common attributes *I* (the intent). A concept *C1* (*E1, I1*) is a sub-concept of another concept *C2* (*E2, I2*) if *E1* $\subseteq$ *E2* or equivalently *I2* $\subseteq$ *I1*. The sub-concept relation is a partial order relationship that forms a lattice over the set of the concepts, each of the nodes of the lattice being a concept. For the infimum of the lattice the intent is empty and the extent contains all the objects, whereas for the supremum the intent contains all the attributes and the extent is empty. Concepts in the lattice are then grouped together depending on the relationships among Concept analysis has been used as a data analysis method in other disciplines for a while.

In software engineering its applications include program understanding, automatic modularization of legacy [12], detection of configuration interference, class hierarchy transformation [10] and, source code restructuring [11]. In modularization, instead of decomposition concept analysis is used to identify grouping of program elements into modules. For example it is used to group together subroutines and global data structures into ADTs for object-oriented migration. As a result this technique is not directly applicable to the type modularization which is the main interested.

V. LATTICE OF CONCEPT SLICES

Program representation formalism is proposed now that is called the Lattice of Concept Slices. Based on this representation, modularization techniques are proposed in the next section. The goal is to achieve a modularization from monolithic such that each module implements preferably a single domain concept, each module is self-contained, there is minimal duplication in code and there is no side effect among modules The formation of the lattice is a three-stage process (i) domain concept identification, (ii) computation of concept slices and finally, (iii) building and analysing the lattice.

VI. APPLICATION FOR IDENTIFICATION OF DOMAIN CONCEPTS

The first step is the identification of domain concepts in the program. This can be done using exhaustive concept assignment techniques. A simpler approach of structural and informal analysis of the source code is applied. In the approach the developer provides a list of domain concepts that are taken from the functional specifications of the system and are implemented in the given program then associates such domain concepts with one or more program elements such as variables and structural idioms in the source code.

The associations may be based on the *use* or *def* of a particular data type or variable, call to a procedure or method, a particular variable passed as parameter in a call,

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 3, March 2013

expressions matching on identifier naming or comments etc. These associations cannot be by no means exhaustive and only serve as a starting point of the analysis. In addition, the software engineer identifies some statements as key statements [13] that are believed to contribute the most in the computation of that domain concept.

```
1: #include <stdio.h>
2: #define YES 1
3: #define NO 2
4: void main()
5: {
6: int nl = 0;
7: int nw = 0;
8: int nc = 0;
9: int inword = NO;
10: int c = getchar();
11: while (c!=EOF)
12: {
13: char ch = (char) c;
14: nc = nc + 1;
15: if (ch=='\n')
16: nl = nl + 1;
17: if (ch==' ' || ch=='\n' || ch=='\t')
18: inword = NO;
19: else if (inword == NO)
20: {
21: inword = YES;
22: nw = nw + 1;}
23: c = getchar();
}
24: printf("%d \n", nl);
25: printf("%d \n", nw);
26: printf("%d \n", nc);
}
```

Fig. 1: Line count program

Some domain concepts can be identified automatically based on a set of general criteria. The rationale behind the criteria is that any information being sent outside from the program or any change in the internal state that is externally visible are information that will be used by other parts of the program and hence are candidates for being part of a domain concept. Such criteria can be the identifiers such as return parameters of a function or method, modified formal parameters that have been called by reference, variables in output/print statements, global variables or class attributes been modified. These identifiers are candidates for domain concepts and the statements that modify these identifiers will be considered as part of the corresponding domain concept. The software engineer may accept or reject the suggestions made automatically.

The outcome of this step is a set of domain concepts and associated with each of them is a set of program statements that implement the concept, where some of the statements are marked as key statements. Each of the concepts is a candidate to form a possible module associated statements will comprise the statements for the module. As an illustration of the technique consider Fig. 1 that illustrates a simple line count program taken from [15] that counts the number of lines, words and characters in a text file and attempting to modularize *main* function. The function outputs the calculation results of three variables – *nl*, *nw* and *nc* statement 24, 25 and 26 respectively. Hence the automatic identification technique suggests the possible presence of three domain concepts corresponding to these three variables. The software engineer confirms the suggestion and names the domain concepts as *Lines*, *Words* and *Chars* respectively. The *nl* variable is being computed in statement number 6 and 16. Statement 6 is the declaration and initialization and does not directly contribute to the computation of *nl*, whereas Statement 16 is the place where the main computation is being done. In this respect the *Lines* domain concept consists of statement 6, 16 and 24, where statement 16 is the key statement.

Table 1 shows the list of domain concepts identified in this step and the statements associated with each of them. In addition to the domain knowledge used by the software engineer to collect the significant variables that are believed to be associated with a specific domain concept, other semi-automated techniques can be also used. These include data mining, cohesion metrics, and data usage analysis [14].

Table 1: The Domain Concepts

Domain Concepts	Statements
Lines	6,16,24
Words	7,22,25
Chars	8,14,26

VII. CONCLUSION

Monolithic application updating is an issue from various aspects; this paper presented implementation reviews upon the very stable and applied source code analysis techniques, program slicing, concept assignment, formal concept analysis. The work has presented an example case for application of them with implications, which have suggested and shown how these techniques can be incorporated for maintaining and restructuring of monolithic legacy program.

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 2, Issue 3, March 2013

REFERENCES

[1] Ted J. Biggerstaff, Bharat G. Mitbander and Dallas Webstar; "The Concept Assignment Problem in Program Understanding', Proceedings of the 15th International Conference on Software Engineering. May 1993.
[2] Nicolas Gold and Keith Bennett; "Hypothesis-based Concept Assignment in Software Maintenance",IEEE Proceedings on Software. August 2002.
[3] Mark Weiser; "Program Slicing", IEEE Transactions on Software Engineering, July 1984.
[4] Susan Horwitz, Thomas Reps and David Binkley; "Inter-procedural Slicing using Program Dependence Graphs", ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 12 Issue 1, January 1990.
[5] Jeanne Ferrante, Karl J. Ottenstein and Joe D. Warren; "The Program Dependence Graph and Its Use in Optimization", ACM Transactions on Programming Languages and Systems, July 1987.
[6] Susan Horwitz and Thomas Reps; "The Use of Program Dependence Graphs in Software Engineering", Proceedings of the 14th International Conference on Software Engineering, May 1992.
[7] Frank Tip; "A Survey on Program Slicing Techniques", Journal of programming languages, 1995.
[8] Andera De Lucia; "Program Slicing: Methods and Application", Proceedings of First IEEE International Workshop on Source Code Analysis and Manipulation, November 2001.
[9] K.J Ottenstein and L.M. Ottenstein; "The Program Dependence Graph in a Software Development Environment", Proceedings of the ACM Software Engineering Symposium on Practical Software Development Environments. April, 1984.
[10] G. Snelting; "Software Reengineering based on Concept Lattices", Proceedings of the Fourth European Conference on Software Maintenance and Reengineering, March 2000.
[11] G. Antoniol, G. Casazza, M. di Penta and E. Merlo; "A method to re-organize legacy systems via concept analysis", Proceedings. 9th International Workshop on Program Comprehension, May 2001.
[12] M. Siff and T. Reps; "Identifying Modules via Concept Analysis", IEEE Transactions on Software Engineering, Volume 25 Issue 6. November 1999.
[13]Mark Harman, Nicolas Gold, Rob Hierons and Dave Binkley; "Code Extraction Algorithms which Unify Slicing and Concept Assignment" Proceedings of Ninth Working Conference on Reverse Engineering, October 2002.
[14] K. Sartipi, K. Kontogiannis; "A User-assisted Approach to Component Clustering", In Journal ofSoftware Maintenance: Research and Practice.
[15] Keith B. Gallagher and James R. Lyle; "Using Program Slicing in Software Maintenance", IEEE Transactions on Software Engineering, Volume 17 Issue 8, August 1991.

BIOGRAPHIES

Asfa Praveen has six years of experience with good practical, academic and research projects exposures after completion of three years Master of Computer Applications (M.C.A.) degree in year 2007 from Punjab Technical University, Jalandhar with very good grades; Advanced 'A' level (P.G.) Diploma in Computer Science in year 2003 from Department of Electronics, Ministry of I.T., Govt. of India; Oracle Certified Professional (O.C.P.) Examination in year 2003 from Oracle Corporation, U.S.A.; she is currently pursuing Ph.D. in Computer Science from Faculty of Science & Technology of Shri Venkateshwara University, Gajraula, (U.P.), her area of research includes Service Oriented Reengineering of Monolithic Legacy Software.

Prof. (Dr.) Shamimul Qamar has sixteen years of wide experience in research, academics and administration, held various positions as Director, Professor, Consultants in universities and engineering colleges after completion of Ph.D. in Electronics and Computer Engineering from Indian Institute of Technology (I.I.T.) Roorkee with excellent grade; he has completed B.Sc. from Ch. Charan Singh University, Meerut; Bachelor of Engineering (B.E.) in Electronics & Communication Engineering from Madan Mohan Malviya Engineering College, Gorakhpur in the year 1996; M.Tech. (Information & Communication Systems) from Aligarh Muslim University, Aligarh; he has published more than 35 research papers in his credits and supervised many master projects and Ph.D. thesis; currently he is designated as Professor of Electronics and Computer Engineering in Noida Institute of Engineering and Technology, Mahamaya Technical University, Noida, U.P. (Delhi-N.C.R.), India.

Dr. Shahanawaj Ahamad is an active academician and researcher in the field of Computer Science, Software Reverse Engineering with twelve years of research and academic experience including five years in abroad, working with College of Computer Science & Engineering of University of Ha'il, K.S.A., before joining UoH he has worked with King Saud University, Al-Khraj University of K.S.A. and Shobhit University, Meerut (Delhi-NCR) and Uttar Pradesh Technical University of INDIA as HoD-I.T., Assistant Professor etc. He is professional member of British Computer Society, U.K., senior member of Computer Society of India, including membership of various national and international academic and research organizations, member of research journal editorial board and reviewer. He is currently working on Service-Oriented Migration, Multi Agent System Reverse Engineering, published more than twenty five research articles in his credit in national and international journals and conference proceedings. He holds M.Tech. followed by Ph.D. in Computer Science with specialization in Software Engineering from Jamia Millia Islamia Central University, New Delhi, India. He has supervised many bachelor projects, master and Ph.D. dissertations.