Practical Application of Program Slicing and Its Use in Software Testing and Maintenance

Ph.D. Thesis Booklet

Ákos Hajnal

Research Supervisors:

István Forgács, Ph.D.

László Zsolt Varga, Ph.D.

Eötvös Loránd University (ELTE), Faculty of Informatics (IK)

Doctoral School in Informatics (András Benczúr, D.Sc.)

Foundations and Methodology of Informatics (János Demetrovics, D.Sc.)

Computer and Automation Research Institute

Hungarian Academy of Sciences (MTA SZTAKI)

Budapest, Hungary, 2012

Introduction

COBOL is often thought as an old-fashioned programming language which is of little importance by now. However, the fact is that several billion lines of COBOL codes are actively used today and COBOL is still the dominant language for business applications. In 1997, it was estimated that as much as 80 percent of the world's computer code ran on COBOL, and there were 240 billion lines of COBOL code in use. Although the role of COBOL has been slightly decreased during the past decades, its dominance is still expected to last over the next ten years as well [Binkley 2007].

COBOL applications are often very large, many of them consist of more than 1,000,000 lines of code, and even applications over 10,000,000 lines are not considered unusually large. Many of the legacy systems are more than 30–40 years old, whose maintenance is very labor-intensive and costly task. The lack of proper documentation, ad-hoc maintenance activities over such long lifetimes, and the poor logical structure of programs can make maintenance very difficult. What is more, there is a huge risk involved in transforming and modernizing such applications, which companies are typically unwilling to undertake.

Program slicing is a potentially useful analysis for aiding such maintenance activities. The concept of program slicing was proposed by Mark Weiser [1984] that extends data-flow analysis by accommodating control dependences (effects of data-flow on control). Program slicing is a technique for simplifying programs by focusing on selected aspects of semantics. The process of slicing "deletes" those parts of the program that can be determined to have no effect upon the semantics of interest, called the *slicing criterion*. As program slices are typically much smaller than the whole program, they can be more easily understood or maintained. Since Weiser's classical "backward static program slicing" motivated to aid debugging, other forms and computations have been evolved such as dynamic slicing, quasi-static slicing, conditioned slicing, amorphous slicing, hybrid slicing, and relevant slicing. Slicing has found its applications in different areas of software engineering including software testing, software maintenance, program comprehension, re- and reverse engineering, and program integration.

The dissertation was motivated by the high and actual demand for supporting maintenance of legacy COBOL systems and the potential in using various applications of program slicing. COBOL has been fallen out of the focus of the program slicing research so far, and as it

turned out, existing methods are inefficient in performing these tasks. The work followed aimed at developing a new static program slicing method that addresses the challenges raise at slicing industrial-scale COBOL codes. The resulting novel approach presented in the dissertation had been successfully applied to – but not limited to – COBOL, and induced developing further related techniques that may help in making program slicing more practical.

Main Contributions

The new scientific results of the dissertation can be summarized in the following five theses.

Thesis 1. *I have analyzed existing static program slicing techniques with respect to their applicability to large-size, legacy COBOL codes, and concluded that previous methods are impractical due to their prohibitive time or space requirements; practical use of program slicing on industrial-scale codes requires a demand-driven approach.*

Precision of the computed program slices is crucial at slicing industrial-scale codes, as overly large slices, containing too much false positives are useless for the users. Imprecision in static analysis may derive from different sources. It can be due to *infeasible* program paths (for which there exists no input that forces the execution of the selected path), or the use of dynamic constructs or pointers (which take their particular values at run-time), which problems are impossible to solve in general. It can be decided that, however, whether a selected program path is *realizable*, i.e., on procedure exits, it always returns to the site of the most recent call. There are studies investigating whether considering the calling-context has significant affect on the slice sizes, and these show that inaccurate slices due to following non-realizable paths can be several times larger than the ones that consider realizable program paths; what is more, the computation of these extra large slices may take more time, which makes imprecise solutions impractical for slicing large-size programs.

There are two fundamental approaches to accounting for the calling-context problem. One is based on explicitly maintaining the call stack; the other is based on a two-pass traversal over the *system dependence graphs* (SDGs) [Horwitz et al. 1990]. The first approach however may cause the re-analysis of procedures several times for different call stacks; moreover, it suffers combinatorial explosion in the case of recursion due to the infinite

number of possible call stacks. Using SDGs, precise static slices can be calculated by performing graph reachability due to summary edges computed in advance. The method is often considered as the quasi-standard technique for precise interprocedural slicing.

However, the construction of the SDG can be very expensive in the case of large-size codes. To build the SDG we need to perform exhaustive data-flow analysis in every procedure to discover all data dependences, and because of the large number of global variables, which is common in COBOL (the principal program structuring mechanism is the `PERFORM` statement used to call parameterless procedures), we must add extra parameter vertices at each procedure entry node and call site. For comparison, the control flow graph representation of one of the investigated program systems contained a total of 210,965 nodes, whereas SDG required introducing more than 85 million extra parameter vertices just to represent parameter passing via globals. The computation of all the summary edges can also be very expensive; it may take several hours (occasionally, even days); furthermore, in the case of COBOL's complex instructions, for each statement that assigns values to more (even several hundreds) of variables we need to create multiple nodes in such a way that each contains at most one definition (required by graph reachability used by SDG-based slicing).

In conclusion, in spite of the accuracy of the SDG-based approach, it also has prohibitive time and space requirements in the case of industrial-scale COBOL codes, which would require a demand-driven approach.

These results are shown in Chapter 1 of the dissertation; related publications of the author are: [1, 3, 5, 6].

Thesis 2. *I have developed a novel demand-driven static program slicing technique based on token propagation over control flow graphs, which computes accurate program slices with respect to realizable program paths. I proved correctness and completeness of the method.*

The basic idea of the proposed method is to explore definition-use chains by propagating *tokens* over the control flow graph. A token is sort of reaching definition information (in forward slicing) associated with a definition, or definitions of the same variable, respectively. The token propagation starts from the node of the slicing criterion, and the token created for the initial definition is propagated to successor nodes iteratively along definition-clear paths with respect to the defined variable. Those nodes that are reached by this token and contain use of the defined variable are marked as "in the slice" (definition-use pairs); definitions

influenced by the uses induce new token propagations from these nodes to explore indirect dependences (definition-use chains).

Without context information the propagation would traverse all possible program paths, including non-realizable ones. To avoid it the method uses *backtrack indices* in tokens to control propagations at procedure exits.

The token propagation method can be extended to accommodate control dependences by introducing *control tokens*, which are created at predicate nodes and propagated along control edges. Nodes reached by control tokens are marked as in the slice; definitions in control dependent nodes start new token propagations to reveal indirect dependences. The method is applicable to compute forward, backward, data-flow and full slices, respectively.

The presented algorithm is based on control flow graphs, which are more easily adaptable to accommodate different programming language constructs, and attains the accuracy of the SDG approach at avoiding its preprocessing requirements by computing summary edges on-demand.

These results are shown in Chapter 2 of the dissertation; related publications of the author are: [1, 3].

Thesis 3. *I implemented the proposed demand-driven technique and evaluated its performance on industrial-scale COBOL codes. I concluded that the method is capable of efficiently calculating precise program slices of reasonable size; longer computation times result in overly large slices uninterpretable for human users.*

To evaluate the presented slicing approach a prototype of the slicing algorithm has been implemented and evaluated on a large COBOL system (consisting of over 1.2 million lines of code). A considerably large set of test cases has been selected randomly on which both slice computation times (data-flow and full slicing in both directions) and slice sizes were measured. The objective was to evaluate how fast slices can be calculated “from scratch”.

The empirical results show that the method is indeed able to compute accurate program slices quickly, whereas longer computation times always result in large slices. This characteristic makes the proposed approach suitable for application in interactive contexts.

These results are shown in Chapter 3 of the dissertation; related publication of the author is: [1].

Thesis 4. *I have developed a novel "slice explainer" technique to aid slice comprehension. The algorithm is applicable to calculate reasoning about computed slice elements in the form of actual dependence chains at low cost, which is confirmed by empirical results.*

Larger program slices, but even slices containing only some tens of program instructions can be very difficult to understand. A method called the "reason-why algorithm" is proposed to reason about slice elements by determining an actual dependence chain – augmented with control information – from the slicing criterion to the chosen slice element. Without such a tool, verification or comprehension of the resulting program slice requires considerable expertise and time.

These results are shown in Chapter 4 of the dissertation; related publication of the author is: [2].

Thesis 5. *I proposed possible improvements for the algorithm design, and showed that by applying these techniques, time or space efficiency of the token propagation-based slicing can be further increased. I have developed modified algorithms to determine definition-use graphs, program dices and chops.*

In the case of large-size codes, the number of tokens to be propagated and stored can be very high. For this reason, different time-space tradeoffs and alternatives for the algorithm design are proposed.

The number of tokens to be stored can be reduced by calculating the topological sorting of procedures and processing them in postorder. The number of tokens to be propagated can be reduced by pre-computing the so called GREF-GMOD-KILL sets that can be exploited to filter unnecessary token propagations in advance. Subsequent slicing tasks can be sped up by reusing the results of previous calculations. It is also noted that the token propagation method can be adapted to calculate flow edges, which can potentially further reduce the number of required token propagations.

A modified algorithm is introduced capable of constructing definition-use graphs that can aid program comprehension and data-flow based testing. It is also shown that the token propagation method can be adapted to calculate slicing variants called dicing and chopping.

These results are shown in Chapter 5 of the dissertation; related publication of the author is: [3].

Scientific Publications

Author's Publications on the Topic of the Thesis

Journals

[1] Ákos Hajnal, István Forgács. 2012. A demand-driven approach to slicing legacy COBOL systems. JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS, 24(1), pp. 67–82.

Impact factor: 0.844, independent citations: 1

[2] Ákos Hajnal, István Forgács. 2012. Understanding program slices. ACTA CYBERNERTICA (to appear).

Conferences

[3] Ákos Hajnal, István Forgács. 2002. A precise demand-driven def-use chaining algorithm. In: 6th European Conference on Software Maintenance and Reengineering (CSMR'2002), IEEE Computer Society, pp. 77–86.

Independent citations: 2

[4] Ákos Hajnal, István Forgács. 1998. An applicable test data generation algorithm for domain errors. In: 1998 ACM/SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '98), ACM New York, pp. 63–72. SOFTWARE ENGINEERING NOTES, 23(2), ACM New York, pp. 63–72.

Independent citations: 12

[5] István Forgács, Ákos Hajnal. 1998. Automated test data generation to solve the Y2k problem. In: 2nd International Software Quality Week Europe 1998 (QWE'98), p. 10 (paper 2S).

[6] István Forgács, Ákos Hajnal, Éva Takács. 1998. Regression slicing and its use in regression testing. In: 22nd Annual International Computer Software and Applications Conference (COMPSAC'98), IEEE Computer Society, pp. 464–469.

Independent citations: 1

Other

[7] István Forgács, Ákos Hajnal. 2011. Forráskód-analízis: Olvassunk a sorok között! COMPUTERWORLD SZÁMÍTÁSTECHNIKA, XLII:(41), pp. 10–12 (in Hungarian).

[8] István Forgács, Ákos Hajnal. 1997. Szoftver tesztelés. Jegyzet Szoftver minőség és tesztelés tárgyhoz, BME, ELTE Doktori Iskola, 1997/1998-as tanév, p. 41 (in Hungarian).

Further Publications

Journals

[9] David Isern, Antonio Moreno, David Sánchez, Ákos Hajnal, Gianfranco Pedone, László Zsolt Varga. 2011. Agent-based execution of personalised home care treatments. APPLIED INTELLIGENCE, 34(2), pp. 155–180.

Impact factor: 0.849, independent citations: 2

Book Chapters

[10] Ákos Hajnal, Tamás Kifor, Gergely Lukácsy, László Zsolt Varga. 2009. Web services as XML data sources in enterprise information integration. In: Services and Business Computing Solutions with XML: Applications for Quality Management and Best Processes (P. Hung, Ed.), IGI Global, pp. 82–97. Enterprise Information Systems: Concepts, Methodologies, Tools and Applications, 2011, Hershey PA: Information Science Reference, pp. 972–985.

[11] Ákos Hajnal, Antonio Moreno, Gianfranco Pedone, David Riano, László Zsolt Varga. 2008. Formalizing and leveraging domain knowledge in the K4CARE home care platform. In: Semantic knowledge management: An ontology-based framework (A. Zilli, E Damiani, P. Ceravolo, A. Corallo, G. Elia, Eds.), Information Science Reference, pp. 279–302.

[12] László Zsolt Varga, Ákos Hajnal, Zsolt Werner. 2005. The WSDL2Agent tool. In: Software Agent-Based Applications, Platforms and Development Kits (R. Unland, M. Klusch, M. Calisti, Eds.), Whitestein Series in Software Agent Technologies, Birkhauser Basel, pp. 197–223.

Conferences

[13] Tamás Kifor, Tibor Gottdank, Ákos Hajnal, Péter Baranyi, Brúnó Korondi, Péter Korondi. 2011. Smartphone emotions based on human-dog interaction. In: 2nd International Conference on Cognitive Infocommunications: CogInfoCom 2011, IEEE Computer Society, pp. 1–6.

[14] Tamás Kifor, Tibor Gottdank, Ákos Hajnal, Csanád Szabó, András Róka, Brúnó Korondi, Péter Korondi. 2011. EthoPhone, human-dog interaction inspired affective computing for smartphone. In: Proceedings IEEE/ASME International Conference on Advanced Intelligent Mechatronics, IEEE Computer Society, pp. 542–547.

[15] Ákos Hajnal, David Isern, Antonio Moreno, Gianfranco Pedone, László Zsolt Varga. 2007. The role of knowledge in designing an agent platform for home care. In: 2nd International Conference on Knowledge Management in Organizations (KMO), pp. 16–26.

Independent citations: 2

[16] Ákos Hajnal, David Isern, Antonio Moreno, Gianfranco Pedone, László Zsolt Varga 2007. Knowledge driven architecture for home care. In: Multi-agent systems and applications V: 5th International Central and Eastern European Conference on Multi-agent Systems (CEEMAS 2007), pp. 173–182. LECTURE NOTES IN ARTIFICIAL INTELLIGENCE, Vol. 4696, Springer-Verlag, pp. 173–182.

Independent citations: 11

[17] Ákos Hajnal, Gianfranco Pedone, László Zsolt Varga. 2007. Ontology-driven agent code generation for home care in Protégé. In: 10th International Protégé Conference: Budapest, Hungary, pp. 91–93.

Independent citations: 4

[18] Ákos Hajnal, Tamás Kifor, Gianfranco Pedone, László Zsolt Varga. 2007. Benefits of provenance in home care. In: Healthgrid 2007. STUDIES IN HEALTH TECHNOLOGY AND INFORMATICS, Vol. 126: From genes to personalized healthcare: grid solutions for the life sciences (N. Jacq, Y. Legré, H. Muller, I. Blanquer, V. Breton, D. Hausser, V. Hernández, T. Solomonides, M. Hofman-Apitius, Eds.), IOS Press, pp. 330–337.

Independent citations: 1

[19] László Zsolt Varga, Ákos Hajnal, Zsolt Werner. 2004. An agent based approach for migrating web services to semantic web services. In: 11th International Conference on Artificial Intelligence: Methodology, Systems (AIMSA 2004). LECTURE NOTES IN COMPUTER SCIENCE, Vol. 3192, Springer-Verlag, pp. 371–380.

Independent citations: 12

[20] László Zsolt Varga, Ákos Hajnal. 2003. Engineering web service invocations from agent systems. In: 3rd International/Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2003). LECTURE NOTES IN COMPUTER SCIENCE, Vol. 2691, Springer-Verlag, pp. 626–636.

Independent citations: 15

Other

[21] Jonathan Dale, Ákos Hajnal, Martin Kernland, László Zsolt Varga. 2003. Integrating web services into Agentcities. Agentcities Technical Recommendation Document (actf-rec-00006).

Independent citations: 10

Summary

	articles	impact factor	ind. citations
journal	3	1.693	3
book chapter	3	–	0
conference	12	–	60
other	3	–	10
total	21	1.693	73

References

BINKLEY, D. 2007. Source code analysis: A road map. In *FOSE '07: 2007 Future of Software Engineering*, 104–119.

HORWITZ, S., REPS, T., AND BINKLEY, D. 1990. Interprocedural slicing using dependence graphs. In *ACM Transactions on Programming Languages and Systems*, 12(1), 26–60.

WEISER, M. 1984. Program slicing. In *IEEE Transactions on Software Engineering*, 10(4), 352–357.