08161 Abstracts Collection
Scalable Program Analysis
— Dagstuhl Seminar —

Florian Martin[1], Hanne Riis Nielson[2], Claudio Riva[3] and Markus Schordan[4]

[1] AbsInt, Saarbrücken, DE
martin@absint.com
[2] DTU Lyngby, DK
riis@imm.dtu.dk
[3] Nokia Helsinki, FIN
claudio.riva@nokia.com
[4] TU Wien, AT
markus@complang.tuwien.ac.at

Abstract. From April 13 to April 18, 2008, the Dagstuhl Seminar 08161 "Scalable Program Analysis" was held in the International Conference and Research Center (IBFI), Schloss Dagstuhl. During the seminar, several participants presented their current research, and ongoing work and open problems were discussed. Abstracts of the presentations given during the seminar as well as abstracts of seminar results and ideas are put together in this paper. The first section describes the seminar topics and goals in general. Links to extended abstracts or full papers are provided, if available.

Keywords. Static analysis, security, pointer analysis, data flow analysis, error detection, concurrency

08161 Executive Summary – Scalable Program Analysis

Motivation and Introduction

As the volume of existing software in the industry grows at a rapid pace, the problems of understanding, maintaining, and developing software assume great significance. A strong support for analysis of programs is essential for a practical and meaningful solution to such problems. Static analysis tools can make a huge impact on how software is engineered but in an industrial context research must be properly balanced with a focus on deployment of analysis tools.

Our goal was to bring together researchers from academia and industry to discuss the strengths and weaknesses of state-of-the-art program analysis technology for industrial-sized software. To achieve that goal the seminar gathered 38 participants from 9 companies and 23 academic/research institutions.

Proceeding of the Seminar

The seminar started with two sessions on technological research challenges in industry and continued with more theoretical sessions, covering scalable shape and pointer analysis, concurrent program analysis, source-level and scalable instruction-level analysis, scalable path conditions, state-of-the-art techniques in abstract interpretation for scalability, and type systems. Overall we had 28 talks, with a good diversity in topics, showing the many research directions and applications of program analysis, also covering program synthesis with sketching, using machine learning for scalable analysis, analysis for architecture reconstruction, and data-flow analysis for multi-core architectures.

A high number of participants, about 50 %, had also implemented their analysis technique in a tool. Some participants had already expressed some interest in tool sessions before the seminar, but there were also concerns mentioned that tool sessions can be a real show-stopper if too lengthy. To encourage lively discussions, the tool sessions were preceded by discussion groups starting on Tuesday. Each group consisted of 4–7 people, of which at least two people had already indicated their interest in presenting a tool. Each group was asked to define a set of challenging questions to be asked about a program analysis tool. On the the next day, Wednesday, we selected in a one hour discussion session in a democratic process a subset of the proposed questions to be answered by every tool presenter on Wednesday/Friday.

The idea was that each presentation of a tool would start by answering those selected 7 questions, providing some basis for comparing the tools, and to create a common frame for each tool presentation. The presentations were kept to a minimum in time, about 15 mins, and the presenters were asked to focus on the most impressive analysis feature of the tool.

This format with preceding working groups worked out well, as it further increased the interest in the tool sessions. We eventually scheduled 15 tool & infrastructure presentations on Thursday and Friday.

The following 10 tools were presented:

- AiT (Florian Martin - AbsInt)
- ASTREÉ (Patrick Cousot - ENS - Paris)
- CodeSonar (David Melski - GrammaTech Inc.- Ithaca)
- Columbus (Arpad Beszedes - University of Szeged)
- EspC Concurrency Toolset (Jason Yue Yang - Microsoft Corp. - Redmond)
- Havoc (Thomas Ball - Microsoft Corp. - Redmond)
- Parfait (Cristina Cifuentes - Sun Microsystems Laboratories - Brisbane)
- PluggableTypes (Michael D. Ernst , MIT - Cambridge)
- SAFE (Eran Yahav - IBM TJ Watson Research Center - Hawthorne)
- Space Invader (Dino Distefano - Queen Mary College - London)

The presented 5 infrastructures were interesting to the participants because they provided a basis for building analysis tools:

- Bauhaus (Rainer Koschke - Universität Bremen)

– LLNL-ROSE (Daniel J. Quinlan, Lawrence Livermore National Laboratory)
– LLVM (Vikram Adve - Univ. of Illinois - Urbana)
– SATIrE (Markus Schordan, TU Vienna)
– WALA (Steve Fink, IBM TJ Watson Research Center - Hawthorne)

An interesting experiment during the seminar was that the entire C++ source code of the infrastructure LLNL-ROSE was used as test case for the Columbus tool and the analysis results were presented in a tool session on the next day.

The tool presentations were mixed, some were hands-on with live presentations, others gave an in-depth view on the tool's capabilities without actually running it. The presenters were asked to keep the presentation short, 10–15 minutes, and focus on some specific interesting feature of the tool. Overall, the format worked well in keeping the audience's attention with every tool. It also encouraged people from academia and industry to get into discussions after the sessions regarding possible future applications and extensions of the tools.

Fun and Art

On Wednesday evening the participants took a well deserved break and visited the nearby winery. A tour on the beautiful hills of the winery with an overview of the history and state-of-the-art of wine making preceded the dinner. Discussions about the selection of the picture that the seminar participants would try to donate a share of, started at the dinner in the winery. Eventually Werner Rauber's picture "Rotkohl 2" ("Red Cabbage") was chosen, and 2 shares were donated by the participants. Since the juice of red cabbage can be used as a home-made pH indicator, turning red in acid and blue in basic solutions, it was considered to by a good analogy to having an indicator for the tradeoff between precision and run-time of an analysis.

Achievements of the Seminar

The seminar showed how broad the field of program analysis has grown over the years. Traditionally used in optimizing compilers, program analysis has turned into a major discipline with techniques and commercial tools supporting understanding, maintaining, and engineering of software. It often turned out that an in-depth discussion of scalability requires further investigations. The scalability of analysis techniques is a major issue as the size of software systems is rapidly growing and the automatic analysis of those systems is becoming yet more important in future. Many questions were raised about scalability - to address the scale of today's systems, analyses will have to be run as parallel programs in future, posing themselves as problem of being scalable on multiple cores, but also whether it can be applied to multiple parts of a system which may differ in structural properties of the code or even in used programming languages.

Raising awareness about the many faces of scalability is the major achievement of the seminar. As the seminar progressed, increasingly more questions about scalability were raised, mostly asking for more extensive evaluations of the methods & tools in future. Discussions about the need and development of specific benchmarks for scalability were started and agreed to be continued past the seminar by different groups of the seminar. Carefully systematically designed sets of test cases, accompanied by test cases from industry, was considered a good setting for evaluating scalability.

Participation Statistics

The seminar was attended by 38 people from 11 countries, which were: Australia (1), Austria (4), Denmark (1), Finland (1), France (4), Germany (7), Great Britain (4), Hungary (1), Ireland (1), Korea (1), U.S.A. (13).

From Academia/Research Institutions 26 people were attending the seminar, whereas from industry 12 people participated, representing 9 companies, which were: AbsInt (Saarbrücken), Airbus (Toulouse), Berner & Mattner Systemtechnik (Berlin), Google Inc. (Mountain View), Grammatech Inc. (Ithaca), IBM Research Center (Hawthorne), Microsoft (Redmond), Nokia (Helsinki), Robert Bosch GmbH (Stuttgart), Sun Microsystems Lab (Brisbane).

Conclusion

The Dagstuhl seminar on “Scalable Program Analysis” was a tremendous success with many fruitful discussions and new questions being raised. Several connections between industry and academia were formed and showed all signs that they will find their continuation after the seminar. The seminar also showed that program analysis and the question about scalability is cross-cutting many different communities. This was also reflected by the diversity of the techniques presented in talks and the tools as well. The cooperation of industry and academia, as being encouraged by many funding programs these days, will further help both sides, to focus on new methods for addressing, characterizing, and comparing scalability.

Systematic Testing of Concurrent Programs with CHESS

Thomas Ball (Microsoft Corp. - Redmond, US)

In this talk, I'll present work by Madan Musuvathi and Shaz Qadeer of my group on radically improving how we test multi-threaded concurrent programs. Using ideas from direct model checking of executables, they have created an automated tool called CHESS that systematically explores the thread schedules of a concurrent program. CHESS incorporates several novel algorithms including iterative context bounding, which prioritizes the search to schedules with fewer context switches first, and fair stateless model checking, which guarantees that the tool will correctly handle programs that depend on fair scheduling to terminate and will find all livelocks in finite state programs. I will demonstrate a version of CHESS I have created for .NET programs and talk about our vision for making debugging of concurrent programs a first-class activity supported by all levels of the software stack.

Keywords: Concurrency, threading, model checking, testing, debugging

Full Paper:
http://research.microsoft.com/projects/chess/

Value Flow Graph Analysis with SATIrE

Gergö Barany (TU Wien, AT)

Partial redundancy elimination is a common program optimization that attempts to improve execution time by removing superfluous computations from a program. There are two well-known classes of such techniques: syntactic and semantic methods. While semantic optimization is more powerful, traditional algorithms based on SSA from are complicated, heuristic in nature, and unable to perform certain useful optimizations. The value flow graph is a syntactic program representation modeling semantic equivalences; it allows the combination of simple syntactic partial redundancy elimination with a powerful semantic analysis. This yields an optimization that is computationally optimal and simpler than traditional semantic methods.

This talk discusses partial redundancy elimination using the value flow graph. A source-to-source optimizer for C++ was implemented using the SATIrE program analysis and transformation system. Two tools integrated in SATIrE were used in the implementation: ROSE is a framework for arbitrary analyses and source-to-source transformations of C++ programs, PAG is a tool for generating data flow analyzers from functional specifications.

Keywords: Partial redundancy elimination, value flow analysis, source-to-source optimization

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1570

Architecture reconstruction and quality measurement in telecom software - an experience report

Arpad Beszedes (University of Szeged, HU)

We present the source code analysis toolset Columbus, which has been developed at the Software Engineering Department of the University of Szeged, Hungary with cooperation of the Nokia Research Center (the owner of rights is FrontEndART Ltd., a spin-off company of the university). Columbus is used as part of our methodology for quality assurance of IT systems, specifically quality assessment based on source code, and architecture reconstruction, flaw detection. Our industrial partners utilizing this technology range from telecommunications software to financial systems, and other types of applications. Furthermore, we participate in quality assurance of different open source projects as well.

One of our most important international partners is Nokia, with which many-years R&D cooperation is maintained in various fields including source code analysis, compiler optimization and embedded systems development in open source environment. During the period of 1998-2001 the basic Columbus technology has been developed, while in consecutive projects between 2005-2007 we performed the analysis of different Nokia software, including proprietary and open source as well, for architecture reconstruction and quality assessment. In this talk, an overview of the technology will be given and expreiences will be reported about this particular industrial application.

Keywords: Source code analysis, quality assessment, architecture reconstruction, Columbus

Joint work of: Beszedes, Arpad; Riva, Claudio

Dependence Cluster Causes

Dave Binkley (Loyola College - Baltimore, US)

A dependence cluster is a maximal set of program components that all depend upon one another. For small programs, programmers as well as static-analysis tools can overcome the negative effects of large dependence clusters. However, this ability diminished as program size increases. Thus, the existence of large dependence clusters presents a serious challenge to the scalability of modern software. Recent ongoing work into the existence and causes of dependence clusters is presented. A better understanding of clusters and their causes is a precursor to the construction of more informed analysis tools and ideally the eventual breaking or proactive avoidance of large dependence clusters.

Keywords: Data Dependence, Control Dependence, Slice, Cluster

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1571

Average Case Analysis of Some Elimination-Based Data-Flow Analysis Algorithms

Johann Blieberger (TU Wien, AT)

The average case of some elimination-based data-flow analysis algorithms is analyzed in a mathematical way. Besides this allows for comparing the timing behavior of the algorithms, it also provides insights into how relevant the underlying statistics are when compared to practical settings.

Keywords: Average case analysis, elimination-based data-flow analysis algorithms, reducible flow graphs

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1572

Synthesis with Sketching and the Counterexample-Guided Inductive Synthesis Algorithm

Rastislav Bodik (Univ. California - Berkeley, US)

The talk will explore how a programmer can (i) encode his insight into a synthesizer and (ii) how to turn a verifier into a synthesizer.

The talk is an overview of the project on program synthesis with sketching: http://sketch.cs.berkeley.edu/wiki/

Keywords: Program synthesis, verification, language design

Parfait - Designing a Scalable Bug Checker

Cristina Cifuentes (Sun Microsystems Laboratories - Brisbane, AU)

We present the design of Parfait, a static layered program analysis framework for bug checking, designed for scalability and precision by improving false positive rates and scale to millions of lines of code. The Parfait framework is inherently parallelizable and makes use of demand driven analyses.

In this paper we provide an example of several layers of analyses for buffer overflow, summarize our initial implementation for C, and provide preliminary results.

Results are quantified in terms of correctly-reported, false positive and false negative rates against the NIST SAMATE synthetic benchmarks for C code.

Keywords: Static analysis, demand driven, parallelizable

Joint work of: Cifuentes, Cristina; Scholz, Bernhard

Extended Abstract: http://drops.dagstuhl.de/opus/volltexte/2008/1573

See also: Proceedings of the ACM SIGPLAN Static Analysis Workshop (SAW), 12 June 2008

A Scalable Technique for Characterizing the Usage of Temporaries in Framework-intensive Java Applications

Bruno Dufour (Rutgers Univ. - Piscataway, US)

Framework-intensive applications (e.g. Web applications) heavily use temporary data structures, often resulting in performance bottlenecks. Blended program analysis enables practical, effective analysis of large framework-based Java applications for performance understanding. Blended analysis combines a dynamic representation of the program calling structure, with a static analysis applied to a region of that calling structure with observed performance problems. We present an optimized blended escape analysis to approximate object lifetimes and thus, to identify these temporary data structures and their uses. Empirical results show that this optimized analysis on average prunes 37% of the basic blocks in our benchmarks, and achieves a speedup of up to 8 times compared to the original analysis. Newly defined metrics quantify key properties of temporary data structures and their uses. A detailed empirical evaluation offers the first characterization of temporaries in framework-intensive applications. The results show that temporary data structures can include up to 12 distinct object types and can traverse through as many as 14 method invocations before being captured.

Keywords: Java, performance understanding, blended analysis

Joint work of: Dufour, Bruno; Ryder, Barbara; Sevitsky, Gary

Scalable pluggable types

Michael D. Ernst (MIT - Cambridge, US)

A type system is valuable only if it helps developers to find and prevent problems in their programs. Progress in type systems has been hampered by a lack of realistic evaluation: many interesting type systems have been proposed without a demonstration of benefits in practice. The difficulty of building a robust and scalable implementation of a type system may be part of the reason for the lack of experimentation.

In order to help the research community to progress, and in particular to help us evaluate our own proposed type systems, we have built a framework for implementing custom (pluggable) type systems in the context of the Java

language, as plug-ins to a Java compiler. Using the framework, a type system designer can implement a new type system just by overriding a few methods, or for simple type systems entirely declaratively. We have also designed extensions to the Java syntax and classfile format that are planned for inclusion in Java 7. These are crucial to the usability and backward compatibility of a pluggable type system.

We have used our framework to build a set of type checkers for Java and to evaluate the type systems they embody. Our checkers have been run on multiple programs of more than 200 KLOC, and on many smaller ones. The experiments in some cases validated our design and in other cases pointed out weaknesses. These insights would have been impossible to achieve without using the type systems in a realistic setting.

Full Paper:
http://groups.csail.mit.edu/pag/jsr308/

Snugglebug

Steve Fink (IBM TJ Watson Research Center - Hawthorne, US)

We describe a new project at IBM Research applying program analysis to improving software quality for large Java applications. We are building a tool that infers partial specifications and involves the user in a specification feedback loop, in order to acquire richer specifications to drive bug finding and test case generation. The tool relies on symbolic underapproximate analysis to derive candidate specifications, find bugs, and generate JUnit tests as concrete witnesses. This talk will give an overview of our approach and discuss work-in-progress with aggressive interprocedural symbolic analysis and test case generation respecting object-oriented APIs.

Scalable Analyses via Machine Learning: Predicting Memory Dependencies Precisely

Lars Gesellensetter (TU Berlin, DE)

Program analysis tackles the problem of predicting the behavior or certain properties of the considered program code. The challenge lies in determining the dynamic run-time behavior statically at compile time.

While in rare cases it is possible to determine exact dynamic properties already statically, in many cases, e.g., in analyzing memory dependencies, one can only find imprecise information.

To overcome this problem, we look at the dynamic run-time behavior and use Machine Learning (ML) techniques to learn the relationship between static program features and dynamic run-time behavior.

ML yields highly scalable predictors, which are safely applicable when erroneous predictions merely have an impact on program optimality but not on correctness. Once trained, these predictors can be used in a static compiler.

In this extended abstract, I present our approach to mitigate the impact of the memory gap using ML techniques and speculative optimizations.

The memory gap denotes the fact that over the last decade, computer performance is increasingly dominated by memory speed, which did not manage to keep pace with the ever increasing CPU rates. We consider novel speculative optimization techniques of memory accesses to reduce their effective latency. We trained predictors to learn the memory dependencies of a given pair of accesses, and use the result in our optimization to decide about the profitability of a given optimization step.

Keywords: Program Analysis, Memory Dependencies, Speculative Optimization, Machine Learning

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1574

Automatic inference of Java's non-null annotations

Thomas Jensen (IRISA - Rennes, FR)

We present a semantics-based automatic null pointer analysis for inferring non-null annotations of fields in object-oriented programs. The analysis is formulated for a minimalistic OO language and is expressed as a constraint-based abstract interpretation of the program which for each field of a class infers whether the field is definitely non-null or possibly null after object initialization. The analysis is proved correct with respect to an operational semantics of the minimalistic OO language. This correctness proof has been machine checked using the Coq proof assistant. Experiments with a prototype implementation for Java byte code show that the analysis is capable of analysing large code bases such as Soot.

Data-Flow Analysis for Multi-Core Computing Systems: A Reminder to Reverse Data-Flow Analysis

Jens Knoop (TU Wien, AT)

The increasing demands for highly performant, proven correct, easily maintainable, extensible programs together with the continuous growth of real-world programs strengthen the pressure for powerful and scalable program analyses for program development and code generation. Multi-core computing systems offer new chances for enhancing the scalability of program analyses, if the additional computing power offered by these systems can be used effectively. This, however, poses new challenges on the analysis side. In principle, it requires program analyses which can be easily parallelized and mapped to multi-core architectures. In this paper we remind to reverse data-flow analysis, which has been introduced and investigated in the context of demand-driven data-flow analysis, as one such class of program analyses which is particularly suitable for this.

Keywords: Multi-core computing systems, scalable program analysis, reverse data-flow analysis, demand-driven data-flow analysis

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1575

Program Analysis on Binaries in Industrial Practice

Florian Martin (AbsInt - Saarbrücken, DE)

AbsInt is active in the area of program analysis on binary executable code for since years. The analyzers are used widely in area of safety critical systems like in the avionics and automotive area.

The talk will present the challanges and some solutions which enabled the analysis of large real systems.

Scalable instruction-level analysis for two security applications

Stephen McCamant (MIT - Cambridge, US)

Commonly static analyses are sound by construction, and other aspects of their design trade off between precision and scalability; but that is not the only possible approach. I'll describe two analyses that were designed from the start to scale to large and complex programs by doing a limited amount of work at the instruction level. Of course, care is still needed to make them sound, and there are still trade offs involving precision or related goals. Though they share many design principles with classic static analysis, the application areas are also a bit different: one is a static rewriting to add runtime sandboxing checks, and the other is a mainly dynamic analysis (with some static aspects) for quantitatively measuring information flow. (Papers about these tools appeared/will appear in Usenix Security 2006 and PLDI 2008 respectively.)

Keywords: Binary analysis, software-based fault isolation, secure information flow

Full Paper:
http://groups.csail.mit.edu/pag/pubs/pittsfield-usenix2006-abstract.html

Full Paper:
http://groups.csail.mit.edu/pag/pubs/secret-max-flow-pldi2008-abstract.html

See also: ‘Evaluating SFI for a CISC Architecture’ by Stephen McCamant and Greg Morrisett. In 15th USENIX Security Symposium, (Vancouver, BC, Canada), August 2-4, 2006.; ‘Quantitative Information Flow as Network Flow Capacity’ by Stephen McCamant and Michael D. Ernst. In Proceedings of the ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation, (Tucson, AZ, USA), June 9-11, 2008.

Towards Distributed Memory Parallel Program Analysis

Daniel J. Quinlan (LLNL - Livermore, US)

Our work presents a parallel attribute evaluation for distributed memory parallel computer architectures where previously only shared memory parallel support for this technique has been developed. Attribute evaluation is a part of how attribute grammars are used for program analysis within modern compilers. Within this work, we have extended ROSE, a open compiler infrastructure, with a distributed memory parallel attribute evaluation mechanism to support user defined global program analysis required for some forms of security analysis which can not be addresses by a file by file view of large scale applications. As a result, user defined security analyzes may now run in parallel without the user having to specify the way data is communicated between processors. The automation of communication enables an extensible open-source parallel program analysis infrastructure.

Keywords: Parallel computing, attribute evaluation, program analysis

Joint work of: Quinlan, Daniel J.; Barany, Gergö; Panas, Thomas

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1568

Source-To-Source Analysis with SATIrE: an Example Revisited

Markus Schordan (TU Wien, AT)

Source-to-source analysis aims at supporting the reuse of analysis results similar to code reuse. The reuse of program code is a common technique which attempts to save time and costs by reducing redundant work. We want to avoid re-analyzing parts of a software system, such as library code. In the ideal case the analysis results are directly associated with the program itself. Source-to-source analysis supports this through program annotations. Further more, to get the best out of available software analysis tools, we aim at enabling the combination of the analysis results of different tools. In order to allow this, tools must be able to process another tool’s analysis results. This enables numerous applications such as automatic annotation of interfaces, testing of analyses by checking

the results of an analysis against provided annotations, domain aware analysis by utilizing domain-specific program annotations, and making analysis results persistent as annotations in source code.

The design of the Static Analysis Tool Integration Engine (SATIrE) allows to map source code annotations to its intermediate program representation as well as generating source code annotations from analysis results that are attached to the intermediate representation. The technical challenges are the design of the analysis information annotation language, the bidirectional propagation of the analysis information through different phases of the internal translation processes, and the combination of the different analyses through the plug-in mechanism. In its current version SATIrE targets C/C++ programs. In this paper we present the approach of source-to-source analysis and show in a detailed example analysis how we support this approach in SATIrE.

Keywords: Source-to-source analysis, ARAL, Annotation Language

Full Paper: http://drops.dagstuhl.de/opus/volltexte/2008/1569

Full Paper:
http://www.complang.tuwien.ac.at/markus/satire

Scalable Path Conditions In Dependence Graphs

Gregor Snelting (Universität Karlsruhe, DE)

Path Conditions are necessary conditions for information flow between two program points. They are useful for eg software reengineering or software security analysis. Path conditions are generated from program dependence graphs. Naive generation of path conditions does however not scale, in particular if it is supposed to be context sensitive and object sensitive.

The talk recalls fundamentals of path conditions, presents algorithmic techniques (based on interval analysis in PDGs and BDDs) to tame complexity, describes precise path conditions for dynamic dispatch, and discusses examples and case studies.

Keywords: Program dependence graphm path condition, security analysis, information flow control

See also: G. Snelting, T. Robschink, J. Krinke: Efficient Path Conditions in Dependence Graphs for Software Safety Analysis. ACM Transactions on Software Engineering and Methodology 15(4), October 2006, pp. 410 - 457 .

Airbus' Experience in static Analysis

Jean Souyris (Airbus - Toulouse, FR)

Airbus' centre of competence "Avionics and simulation products" uses static analyzers on certified avionics programs and plans to use more on the next generation of avionics software products. This has been possible thanks to its involvement in research projects together with labs like Pr. Cousot's one at Ecole normale superieure de Paris.

Keywords: Avionics - Certification - Abstract Interpretation

Static Program Analyses at Berner & Mattner

Joachim Wegener (Berner & Mattner Systemtechnik - Berlin, DE)

Thr presentation will give an overview on typical static analysis projects performed for customers and the tools used. Experiences with existing tools will be described and existing weaknesses explained.

Joint work of: Wegener, Joachim; Schmidt, Michael; Sthamer, Harmen

QVM: An Efficient Runtime for Detecting Defects in Deployed Systems

Eran Yahav (IBM TJ Watson Research Center - Hawthorne, US)

Coping with software defects that occur in the post deployment stage is a challenging problem: bugs may occur only when the systems uses a specific configuration and only under certain usage scenarios. Nevertheless, halting production systems until the bug is tracked and fixed is often impossible.

Thus, developers have to try to reproduce the bug in laboratory conditions. Often the reproduction of the bug consists of the lion share of the debugging effort.

In this paper we suggest an approach to address the aforementioned problem by using a specialized runtime environment (QVM, for Quality Virtual Machine). QVM efficiently detects defects by continuously monitoring the executions of the production system. QVM enables the efficient checking of violations of user-specified correctness properties, e.g., typestate safety properties, Java assertions, and heap properties pertaining to ownership.

QVM is markedly different from existing techniques for continuous monitoring by using a novel overhead manager which enforces a user-specified overhead budget for quality checks. Existing tools for error detection in the field usually disrupt the operation of the deployed system. QVM, on the other hand, provides a balanced trade off between the cost of the monitoring process and the maintenance of sufficient accuracy for detecting defects. Specifically, the overhead cost of using QVM instead of a standard JVM, is low enough to be acceptable in production environments.

We implemented QVM on top of IBM's production Java Virtual Machine (J9) and used it to detect and fix various errors in real-world applications.

Keywords: Virtual machine, dynamic analysis, overhead management, software quality

Joint work of: Yahav, Eran; Vechev, Martin; Arnold, Matt

Verifying Dereference Safety via Expanding-Scope Analysis

Eran Yahav (IBM TJ Watson Research Center - Hawthorne, US)

This paper addresses the challenging problem of verifying the safety of pointer dereferences in real Java programs. We provide an automatic approach to this problem based on a sound interprocedural analysis. We present a staged expanding-scope algorithm for interprocedural abstract interpretation, which invokes sound analysis with partial programs of increasing scope. This algorithm achieves many benefits typical of whole-program interprocedural analysis, but scales to large programs by limiting analysis to small program fragments. To address cases where the static analysis of program fragments fails to prove safety, the analysis also suggests possible annotations which, if a user accepts, ensure the desired properties.

Experimental evaluation on a number of Java programs shows that we are able to verify 90 % of all dereferences soundly and automatically, and further reduce the number of remaining dereferences using non-nullness annotations.

Keywords: Null dereference, verification, abstract interpretation

Joint work of: Loginov, Alexey; Yahav, Eran; Chandra, Satish; Fink, Stephen; Rinetzky, Noam; Nanda, Mangala Gowri

SAFE / SAFE Mining / SALSA

Eran Yahav (IBM TJ Watson Research Center - Hawthorne, US)

This presentation summarizes the work of many people on a number of related projects in IBM Research. In particular, this is a brief summary of our papers from ISSTA'06, ISSTA'07 and ISSTA'08.

Keywords: Typestate, verification, abstract interpretation, aliasing

Full Paper:
http://www.research.ibm.com/people/e/eyahav/publications.html

Scalable Shape Analysis For Systems Code

Hongseok Yang (Queen Mary College - London, GB)

Pointer safety faults in device drivers are the number one cause of crashes in operating systems code. In principle, shape analysis tools can be used to prove the absence of this type of error. In practice, however, shape analysis is not used due to the unacceptable mixture of scalability and precision provided by existing tools. In this talk, I will describe a new join operation for the separation domain which aggressively abstracts information for scalability yet does not lead to false error reports. The operator is a critical piece of a new shape analysis tool that provides an acceptable mixture of scalability and precision for industrial application.

Experiments with our tool on whole Windows and Linux device drivers (firewire, pci-driver, cdrom, md, etc.) represent the first working application of shape analysis to whole industrial programs—and the beginning of the end for the largest problem plaguing the correctness of systems code.

This is joint work with Oukseh Lee, Josh Berdine, Cristiano Calcagno, Byron Cook, Dino Distefano and Peter O'Hearn.

Keywords: Shape Analysis, Program Verification, Abstract Interpretation, Separation Logic

Joint work of: Yang, Hongseok; Lee, Oukseh; Berdine, Josh; Calcagno, Cristiano; Cook, Byron; Distefano, Dino; O'Hearn, Peter

Taming Win32 Threads with Static Analysis

Jason Yue Yang (Microsoft Corp. - Redmond, US)

The battle against concurrency bugs poses a serious challenge across the software industry. To help developers tackle concurrency issues, we have developed a concurrency toolset based on static analysis technologies, comprising (1) an annotation language Concurrency SAL, 2) concurrency checkers EspC and Global EspC, which employ single-function and cross-function analyses respectively, and 3) an annotation inference tool CSALInfer. In this talk, we describe how to use this toolset to detect a variety of concurrency bugs including deadlocks, race conditions, Win32 locking errors, and atomicity violations.

Programmer-assisted program analyses

Daniel von Dincklage (Google Inc. - Mountain View, US)

Modern object-oriented languages have complex features such as dynamic class loading that make analyses difficult.

In practice programmers rarely use these complex features in their full generality. As a consequence, analyses that would produce precise results with the author's intended semantics are often inapplicable since the analyses need to handle the full generality of these features.

A programmer can sidestep this problem by placing annotations into his program: Annotations communicate the intended semantics to the analyses and enable analyses that would otherwise be imprecise.

In practice, using annotations is difficult because the programmer must guess (i) which annotations to place and (ii) where to place the annotations. Thus, it is unlikely that he will guess correctly and place an annotation that actually makes an analysis more preicse.

We present a system that makes it feasible for a programmer to improve the precision of analyses and thus speed up his program by placing annotations. Our system first recommends which analyses the programmer should enable and then guides the programmer towards which annotations he should place to enable many optimizations.

Keywords: Annotation, optimization, analysis