Report from Dagstuhl Seminar 14352

Next Generation Static Software Analysis Tools

Edited by

Patrick Cousot[1], Daniel Kroening[2], and Carsten Sinz[3]

1 **ENS – Paris, FR,** pcousot@cims.nyu.edu
2 **University of Oxford, GB,** kroening@cs.ox.ac.uk
3 **KIT – Karlsruhe Institute of Technology, DE,** carsten.sinz@kit.edu

Abstract

There has been tremendous progress in static software analysis over the last years with, for example, refined abstract interpretation methods, the advent of fast decision procedures like SAT and SMT solvers, new approaches like software (bounded) model checking or CEGAR, or new problem encodings. We are now close to integrating these techniques into every programmer's toolbox.

The aim of the seminar was to bring together developers of software analysis tools and algorithms, including researchers working on the underlying decision procedures (e. g., SMT solvers), and people who are interested in applying these techniques (e. g. in the automotive or avionics industry).

The seminar offered the unique chance, by assembling the leading experts in these areas, to make a big step ahead towards new, more powerful tools for static software analysis.

Current (academic) tools still suffer from some shortcomings:

- Tools are not yet robust enough or support only a subset of a programming language's features.
- Scalability to large software packages is not yet sufficient.
- There is a lack of standardized property specification and environment modeling constructs, which makes exchange of analysis results more complicated than necessary.
- Differing interpretations of programming language semantics by different tools lead to limited trust in analysis results.
- Moreover, a comprehensive benchmark collection to compare and evaluate tools is missing.

Besides these application-oriented questions, further, more fundamental questions have also been topics of the seminar:

- What are the right logics for program verification, bug finding and software analysis? How can we handle universal quantification? And how to model main memory and complex data structures?
- Which decision procedures are most suitable for static software analysis? How can different procedures be combined? Which optimizations to general-purpose decision procedures (SAT/SMT/QBF) are possible in the context of software analysis?

Seminar August 24–29, 2014 – http://www.dagstuhl.de/14352
1998 ACM Subject Classification D.2.4 Software/Program Verification, F.3.1 Specifying and Verifying and Reasoning about Programs
Keywords and phrases Software quality, Bug finding, Verification, Decision procedures, SMT/SAT solvers
Digital Object Identifier 10.4230/DagRep.4.8.107
Edited in cooperation with Christoph Gladisch

Next Generation Static Software Analysis Tools, *Dagstuhl Reports*, Vol. 4, Issue 8, pp. 107–125
Editors: Patrick Cousot, Daniel Kroening, and Carsten Sinz

Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Executive Summary

Patrick Cousot
Klaus Havelund
Daniel Kroening
Carsten Sinz
Christoph Gladisch

Software errors are still a widespread plague. They manifest themselves, e. g., in program crashes, malfunction, incorrect behavior, or security vulnerabilities. Even software that has been in use for decades and has been deployed to millions of users (e. g., the compression library zlib) still contains flaws that are revealed only now and have to be fixed. Both in academia and industry considerable effort has been undertaken to develop tools and methodologies to obtain fault-free software. Nowadays, static analysis tools, which search for program errors without running the software, have reached a state where they are, in some industries (e. g., the automotive or avionics industry), already part of the standard software development and quality assurance process (with tools and companies like, e. g., Polyspace, Coverity, KlocWork, AbsInt, or Astrée). And although these tools can help finding residual errors more quickly, they still suffer from some shortcomings:

- Lack in precision. For a certain fraction of program locations in the source code it cannot be decided whether there is an error or not. Such "undecided cases" require (often time-consuming) manual rework, limiting the value of such tools.
- Due to the manual effort required, static software analysis tools have not yet made their way to mainstream software development (besides industries, where software reliability is indispensable and considerable amounts of time and money are spent on quality assurance).

Over the last years, software analysis tools based on abstract interpretation have been refined and tools based on new core formalisms, such as model checking, have gained traction, mainly in the form of two key methods: counterexample-guided abstraction refinement (CEGAR), and bounded model checking (BMC). The success of these new tools was, to a substantial part, enabled by the enormous progress that was made on the underlying logical decision procedures (SAT and SMT solvers). New software analysis tools based on these techniques come with considerably improved precision (less false positives), but they are still not competitive with tools based on abstract interpretation with respect to scalability. Also, they are rarely used in industrial software development projects so far.

With this seminar we believe that we were able to stimulate further progress in this field by intensifying the collaboration between (a) researchers on new static software analysis tools, (b) scientists working on improved high-performance decision procedures, and (c) practitioners, who know what is needed in industry and which kind of software analysis tools are accepted by developers and which are not.

The Dagstuhl Seminar was attended by participants from both industry and academia. It included presentations on a wide range of topics such as:

- Recent trends in static analysis, consisting of new algorithms and implementation techniques.
- New decision procedures for software analysis, for example, to analyze programs with complex data structures.

- Industrial case studies: What are the problems industrial users of static analysis tools are facing?
- Experience reports and statements on current challenges.

The first day of the seminar started with an introduction round, in which each participant shortly presented his research interests. As the seminar was held concurrently with a second, closely related Dagstuhl Seminar on "Decision Procedures and Abstract Interpretation" (14351), the introductory session was held jointly by both seminars. Four overview talks were also organized jointly by both seminars, and were given by Thomas Reps, Patrick Cousot, Vijay Ganesh, and Francesco Logozzo.

There was also a tool demonstration session on Thursday afternoon, in which seven tools were presented (15 minutes each).

In further talks of the seminar young as well as senior researchers presented on-going and completed work. Tool developers and participants from industry reflected on current challenges in the realm of software analysis.

The seminar was concluded with a panel discussion about the current challenges of static software analysis for industrial application (see Sec. 5 for an extended exposition of the panel discussion).

We expect that with this Dagstuhl Seminar we were able to make a step forward towards bringing static software analysis tools to every programmer's workbench, and therefore, ultimately, improve software quality in general.

2 Table of Contents

3 Overview of Talks

3.1 Most Overlooked Static Analysis Pitfalls

Roberto Bagnara (BUGSENG & University of Parma, IT)

Quality software requires complex verification activities. Such activities cannot be practically and reliably performed without extensive use of tools. Poor-quality static analysis tools either result into higher costs of the verification process or fail their goal altogether, delivering a false sense of security instead of the promised quality for the developed software. There are well-engineered static analysis tools that are based on obsolete technology, to the point of ignoring 30+ years of research in software verification. There also are theoretically-sophisticated tools that fall short of their objectives due to poor engineering. In this talk I will illustrate what I believe are important pitfalls of the design of static analysis tools, drawing on the experience of the BUGSENG's team that developed the ECLAIR software verification platform.

3.2 CPAchecker: A Flexible Framework for Software Verification

Dirk Beyer (Universität Passau, DE)

Main reference D. Beyer, M. E. Keremoglu, "CPAchecker: A Tool for Configurable Software Verification," in Proc. of the 23rd Int'l Conf. on Computer Aided Verification (CAV'11), LNCS, Vol. 6806, pp. 184–190, Springer, 2011; pre-print available from author's webpage.
URL http://dx.doi.org/10.1007/978-3-642-22110-1_16
URL http://www.sosy-lab.org/~dbeyer/Publications/2011-CAV.CPAchecker_A_Tool_for_Configurable_Software_Verification.pdf

CPAchecker is a tool and framework that aims at easy integration of new verification components. It is based on configurable program analysis, a concept for implementing different approaches from data-flow analysis, abstract interpretation, and software model-checking in one uniform software framework. Every abstract domain, together with the corresponding operations, implements the interface of configurable program analysis (CPA). The main algorithm is configurable to perform a fixed-point analysis on arbitrary combinations of existing CPAs.

In software verification, it takes a considerable amount of effort to convert a verification idea into actual experimental results– we aim at accelerating this process. We hope that researchers and practitioners find it convenient and productive to implement new verification ideas and algorithms using this flexible and easy-to-extend platform, and that it advances the field by making it easier to perform practical experiments.

The tool is implemented in Java and runs as command-line tool or as Eclipse plug-in. CPAchecker has existing CPAs for several abstract domains already, including predicates, explicit values, octagons, and BDDs. The tool integrates CEGAR, lazy abstraction refinement, interpolation, boolean predicate abstraction, large-block encoding, bounded model checking, generation of error witnesses with test values, and several SMT solvers in a modular and flexible design. CPAchecker is publicly available under the open-source license Apache 2. The tool won several medals in competitions on software verification.

http://cpachecker.sosy-lab.org/

3.3 Abstract Interpretation: "Scene-Setting Talk"

Patrick Cousot (ENS – Paris, FR)

We unify static analysis by extrapolation (widening) with static analysis by interpolation to prove a given program specification. This unification is done in the theory of abstract interpretation using dual-narrowing. We show that narrowing and dual-narrowing are equivalent up to the exchange of their parameters. This yields new ideas for narrowing based on Craig interpolation. This unification is also possible by understanding that interpolation can be done in arbitrary abstract domains, not only logical ones. We show that an increasing iterative static analysis using extrapolation of successive iterates by widening followed by a decreasing iterative static analysis using interpolation of successive iterates by narrowing (both bounded by the specification) can be further improved by a increasing iterative static analysis using interpolation of iterates with the specification by dual-narrowing until reaching a fixpoint and checking whether it is inductive for the specification.

3.4 Abstracting Induction by Extrapolation and Interpolation

Patrick Cousot (ENS – Paris, FR)

We unify static analysis by extrapolation (widening) with static analysis by interpolation to prove a given program specification. This unification is done in the theory of abstract interpretation using dual-narrowing. We show that narrowing and dual-narrowing are equivalent up to the exchange of their parameters. This yields new ideas for narrowing based on Craig interpolation. This unification is also possible by understanding that interpolation can be done in arbitrary abstract domains, not only logical ones. We show that an increasing iterative static analysis using extrapolation of successive iterates by widening followed by a decreasing iterative static analysis using interpolation of successive iterates by narrowing (both bounded by the specification) can be further improved by a increasing iterative static analysis using interpolation of iterates with the specification by dual-narrowing until reaching a fixpoint and checking whether it is inductive for the specification.

3.5 Path-sensitive static analysis using trace hashing

Tomasz Dudziak (Universität des Saarlandes, DE)

Path-sensitivity can significantly improve precision of static program analysis but due to the inherently exponential number of possible control flow histories it often requires manual tweaking of parameters and annotations. I propose an approach based on hashing of control flow paths that can dynamically adapt to available resources. By careful construction of the hash function it can exploit additional assumptions about the nature of path-sensitive properties. Additionally, due to its randomized nature it introduces a new trade-off between cost of the analysis and probability of proving properties of interest.

3.6 An algebraic approach for inferring and using symmetries in rule-based models

Jerôme Feret (ENS – Paris, FR)

Symmetries arise naturally in rule-based models, and under various forms. Besides automorphisms between site graphs, which are usually built within the semantics, symmetries can take the form of pairs of sites having the same capabilities of interactions, of some protein variants behaving exactly the same way, or of some linear, planar, or 3D molecular complexes which could be see modulo permutations of their axis and/or mirror-image symmetries.

In this paper, we propose a unifying handling of symmetries in Kappa. We follow an algebraic approach, that is based on the single pushout semantics of Kappa. We model classes of symmetries as finite groups of transformations between site graphs, which are compatible with the notion of embedding (that is to say that it is always possible to restrict a symmetry that is applied with the co- domain of an embedding to the domain of this embedding) and we provide some assumptions that ensure that symmetries are compatible with pushouts. Then, we characterize when a set of rules is symmetric with respect to a group of symmetries and, in such a case, we give sufficient conditions so that this group of symmetries induces a forward bisimulation and/or a backward bisimulation over the population semantics.

3.7 Bounded Verification with TACO: Symmetry-breaking + tight field bounds

Marcelo Frias (University of Buenos Aires, AR)

Main reference J. P. Galeotti, N.Rosner, C. G. Lopez Pombo, M. F. Frias, "TACO: Efficient SAT-Based Bounded Verification Using Symmetry Breaking and Tight Bounds ," IEEE Trans. on Software Engineering, 39(9):1283–1307, 2013.
URL http://dx.doi.org/10.1109/TSE.2013.15

I will discuss work developed in my group on bounded verification of JML-annotated Java code, and how appropriate symmetry breaking predicates and bounds on the semantics of class fields allow us to improve the performance of TACO.

3.8 Impact of Community Structure on SAT Solver Performance

Vijay Ganesh (University of Waterloo, CA)

Joint work of Newsham, Zack; Ganesh, Vijay; Fischmeister, Sebastian; Audemard, Gilles; Simon, Laurent
Main reference Z. Newsham, V. Ganesh, S. Fischmeister, G. Audemard, L. Simon, "Impact of Community Structure on SAT Solver Performance," in Proc. of the 17th Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT'14), LNCS, Vol. 8561, pp. 252–268, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-09284-3_20

Modern CDCL SAT solvers routinely solve very large industrial SAT instances in relatively short periods of time. It is clear that these solvers somehow exploit the structure of real-world instances. However, to-date there have been few results that precisely characterize this structure. In this paper, we provide evidence that the community structure of real-world

SAT instances is correlated with the running time of CDCL SAT solvers. It has been known for some time that real-world SAT instances, viewed as graphs, have natural communities in them. A community is a sub-graph of the graph of a SAT instance, such that this sub-graph has more internal edges than outgoing to the rest of the graph. The community structure of a graph is often characterized by a quality metric called Q. Intuitively, a graph with high-quality community structure (high Q) is easily separable into smaller communities, while the one with low Q is not. We provide three results based on empirical data which show that community structure of real-world industrial instances is a better predictor of the running time of CDCL solvers than other commonly considered factors such as variables and clauses. First, we show that there is a strong correlation between the Q value and Literal Block Distance metric of quality of conflict clauses used in clause-deletion policies in Glucose-like solvers. Second, using regression analysis, we show that the the number of communities and the Q value of the graph of real-world SAT instances is more predictive of the running time of CDCL solvers than traditional metrics like number of variables or clauses. Finally, we show that randomly-generated SAT instances with 0.05 <= Q <= 0.13 are dramatically harder to solve for CDCL solvers than otherwise.

3.9 Using a deductive verification environment for verification, bug finding, specification, and all that

Christoph Gladisch (KIT – Karlsruher Institut für Technologie, DE)

Joint work of Gladisch, Christoph; Shmuel Tyszberowicz; Bernhard Beckert; Ferruccio Damiani; Mana Taghdiri; Mattias Ulbrich; Tianhai Liu; Aboubakr Achraf El Ghazi; Daniel Grunwald

The boundaries between program analysis techniques such as static analysis, deductive verification, abstract interpretation, and model checking are overlapping. Deductive verification technology can, for instance, be used as a framework for the other techniques. In this talk deductive verification technology is taken as a basis and a set of techniques that were developed on the KeY platform are presented such as fault detection, model generation for quantified formulas, test generation, and specification techniques.

References

1 Daniel Grunwald and Christoph Gladisch and Tianhai Liu and Mana Taghdiri and Shmuel Tyszberowicz. *Generating JML Specifications from Alloy Expressions*. 10th Haifa Verification Conference (HVC), Israel, 2014
2 Aboubakr Achraf El Ghazi and Mattias Ulbrich and Christoph Gladisch and Shmuel Tyszberowicz and Mana Taghdiri. *JKelloy: A Proof Assistant for Relational Specifications of Java Programs*. NASA Formal Methods – 6th International Symposium, NFM 2014, Houston, TX, USA, 2014.
3 Wolfgang Ahrendt and Bernhard Beckert and Daniel Bruns and Richard Bubel and Christoph Gladisch and Sarah Grebing and Reiner Hähnle and Martin Hentschel and Vladimir Klebanov and Wojciech Mostowski and Christoph Scheben and Peter Schmitt and Mattias Ulbrich. *The KeY Platform for Verification and Analysis of Java Programs* 6th Working Conference on Verified Software: Theories, Tools and Experiments 2014, Vienna, Austria, 2014.

4 Christoph Gladisch and Shmuel Tyszberowicz. *Specifying a Linked Data Structure in JML for Formal Verification and Runtime Checking*. Brazilian Symposium on Formal Methods (SBMF), Brasilia, Brasil, 2013
5 Christoph Gladisch. *Model Generation for Quantified Formulas with Application to Test Data Generation*. International Journal on Software Tools for Technology Transfer (STTT), Volume 14, Number 4, 2012.
6 Christoph Gladisch. *Verification-based Software-fault Detection*. KIT Scientific Publishing, Karlsruhe, 2011.

3.10 Static Analysis of Energy Consumption

Manuel Hermenegildo (IMDEA Software – Madrid, ES)

Joint work of López, Pedro; Haemmerlé, Remy; Hermenegildo, Manuel V.; Klemen, Maximiliano; Liqat, Umer; Serrano, Alejandro; Eder, Kerstin; Georgiou, Kiryakos; Kerrison, Steve
Main reference U. Liqat, S. Kerrison, A. Serrano, K. Georgiou, P. López-Garcia, N. Grech, M. V. Hermenegildo, K. Eder, "Energy Consumption Analysis of Programs based on XMOS ISA-Level Models," in Proc. of the 23rd Int'l Symp. on Logic-Based Program Synthesis and Transformation (LOPSTR'13), to appear; pre-print available from author's webpage.
URL http://clip.dia.fi.upm.es/papers/isa-energy-lopstr13-final.pdf

Energy consumption is a major concern in data centers and high-performance computing, and there is also an increased demand for energy savings in devices which operate on batteries and other limited power sources, such as implantable/portable medical devices, sensors, or mobile phones. Beyond the advances in hardware power efficiency, significant additional energy savings can be achieved by improving the software. Static inference of the energy consumed by programs during execution is instrumental in this task, having important applications in the optimization and verification of such consumption by programs, and in general in energy-aware software development. At the same time it is an area that presents a number of interesting challenges.

We present an approach to the inference and verification of upper- and lower-bounds on the energy consumption of programs, as well as some current results from our tools. The bounds we infer and check are functions of the sizes of the input data to the program. Our tools are based on translating the program to a block-based intermediate representation, expressed as horn clauses, deriving cost equations, and finding upper- and lower-bound cost solutions. We also present some recent improvements to resource bounds inference, including casting the cost analysis more fully within abstract interpretation frameworks and using sized shapes as data abstractions. The energy analysis makes use of ISA- and LLVM-level models of the cost of instructions or sequences of instructions. The inferred bounds compare well to measurements on the hardware and open up new avenues for future research and application.

See the Dagstuhl Seminar slides (http://www.dagstuhl.de/mat/Files/14/14352/14352.HermenegildoManuel.Slides.pdf) for a full classified bibliography.

References

1 U. Liqat, S. Kerrison, A. Serrano, K. Georgiou, P. López-Garcia, N. Grech, M.V. Hermenegildo, K. Eder. Energy Consumption Analysis of Programs based on XMOS ISA-Level Models. In *Pre-proceedings of the 23rd International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 13)*, September 2013.

2 A. Serrano, P. López-Garcia, M. Hermenegildo. Resource Usage Analysis of Logic Programs via Abstract Interpretation Using Sized Types. In *Theory and Practice of Logic Programming, 30th Int'l Conference on Logic Programming (ICLP'14) Special Issue*, Vol. 14, Num. 4–5, pages 739–754, Cambridge U. Press, 2014.

3.11 Insides and Insights of Commercial Program Analysis

Ralf Huuck (NICTA – Sydney, AU)

Joint work of Huuck, Ralf; Cassez, Franck; Fehnker, Ansgar

In this work we give an overview of the technologies underpinning our commercial C/C++ program analyzer Goanna and we share some of the experiences in applying these technologies to large industrial code bases. In particular, we highlight the core technologies of model checking, abstract interpretation and SMT-based automatic trace refinement as well as their interplay within the Goanna tool. We present some commercial experiences ranging from runtime metrics to tool comparison, and we highlight some of the challenges that we have been facing and as well as opportunities ahead.

3.12 Steps towards usable verification

Francesco Logozzo (Microsoft Research – Redmond, US)

Main reference M. Fähndrich, F. Logozzo, "Static contract checking with Abstract Interpretation," in Proc. of the 2010 Int'l Conf. on Formal Verification of Object-oriented Software (FoVeOOS'10), LNCS, Vol. 6528, pp. 10–30, Springer, 2010.
URL http://dx.doi.org/10.1007/978-3-642-18070-5_2

We describe our experience with the CodeContracts static checker (cccheck), probably the most successful verification tool out there: The analyzer has been downloaded over 150K times, and it is used in Microsoft product groups.

The cccheck is based on abstract interpretation, and it does not use any out-of- the-box SMT solver. In the talk I explain the rationale for this decision. Briefly, abstract interpretation allows us to have a very fine grain control on the precision/cost ration and it provides a level of automation unmatched by other approaches (e. g., for the inference of loop invariants, the suggestion of code fixes, or the generation of *sound* contracts). Furthermore, we avoid all kind of problems that come from using external generic tools, as, e. g., timeouts, non-monotonicity of the analysis, randomizations, non-determinism, etc.

3.13 Viper – Verification Infrastructure for Permission-based Reasoning

Peter Mueller (ETH Zürich, CH)

Joint work of Juhasz, Uri; Kassios, Ioannis; Müller, Peter; Novacek, Milos; Schwerhoff, Malte; Summers, Alexander

Main reference U. Juhasz, I. T. Kassios, P. Müller, M. Novacek, M. Schwerhoff, A. J. Summers, "Viper: A Verification Infrastructure for Permission-Based Reasoning," Technical Report, ETH Zürich, 2014.

URL http://pm.inf.ethz.ch/publications/getpdf.php?bibname=Own&id=JKMNSS14.pdf

The automation of verification techniques based on first-order logic specifications has benefited greatly from verification infrastructures such as Boogie and Why. These offer an intermediate language that can express diverse language features and verification techniques, as well as back-end tools such as verification condition generators.

However, these infrastructures are not well suited for verification techniques based on separation logic and other permission logics, because they do not provide direct support for permissions and because existing tools for these logics often prefer symbolic execution over verification condition generation. Consequently, tool support for these logics is typically developed independently for each technique, dramatically increasing the burden of developing automatic tools for permission-based verification.

In this talk, we present a verification infrastructure whose intermediate language supports an expressive permission model natively. We provide tool support, including two back-end verifiers, one based on symbolic execution, and one on verification condition generation; this facilitates experimenting with the two prevailing techniques in automated verification. Various existing verification techniques can be implemented via this infrastructure, alleviating much of the burden of building permission-based verifiers, and allowing the developers of higher-level techniques to focus their efforts at the appropriate level of abstraction.

3.14 Static Analysis Modulo Theory

Andreas Podelski (Universität Freiburg, DE)

A recent approach to static analysis can be described in analogy with SMT solving. Satisfiability here corresponds to the existence of an error path in the program, or: unsatisfiability corresponds to the emptiness of an automaton. Each time the tool finds an error path, i. e., a word accepted by the automaton, it analyzes the word in the theory of the data domain. If the word is infeasible, it learns a new automaton which rejects the word (and many others). It then adds the new automaton to the intersection of the already existing automata. We can extend the approach from sequential to recursive, parallel or unboundedly parallel programs. The emptiness check always amounts to a static analysis over a "theory-free" abstract domain.

3.15 Static Analysis Blind Spots in Automotive Systems Development

Hendrik Post (Robert Bosch GmbH – Stuttgart, DE)

Sound Static Analysis has become a mature and powerful technique for source code analysis. It is now time to stop and to contemplate whether scalability is still the most important problem or whether industrial applications lack other non-trivial contributions beyond improving performance. In this talk, we give an overview about interests and blockers for static analysis from an industrial perspective. Based on these findings, we give input for the workshop discussions.

3.16 Construction of modular abstract domains for heterogeneous properties

Xavier Rival (ENS – Paris, FR)

In this talk, we study the construction of shape-numeric static analysers. We set up an abstract interpretation framework that allows to reason about simultaneous shape-numeric properties by combining shape and numeric abstractions into a modular, expressive abstract domain. Such a modular structure is highly desirable to make its formalisation, proof and implementation easier to perform and to get correct. Furthermore, we extend this modular abstract domains so as to combine different memory abstractions, for better scalability and greater expressiveness. This framework is implemented in the MemCAD static analyser.

3.17 Efficiently Intertwining Widening with Narrowing

Helmut Seidl (TU München, DE)

Joint work of Seidl, Helmut; Apinis, Kalmer; Vojdani, Vesal
Main reference K. Apinis, H. Seidl, V. Vojdani, "How to combine widening and narrowing for non-monotonic systems of equations," in Proc. of the 34th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI'13), pp. 377–386, ACM 2013.
URL http://dx.doi.org/10.1145/2491956.2462190

Non-trivial analysis problems require posets with infinite ascending and descending chains. In order to compute reasonably precise post-fixpoints of the resulting systems of equations, Cousot and Cousot have suggested accelerated fixpoint iteration by means of widening and narrowing.

The strict separation into phases, however, may unnecessarily give up precision that cannot be recovered later. While widening is also applicable if equations are non-monotonic, this is no longer the case for narrowing. A narrowing iteration to improve a given post-fixpoint, additionally, must assume that all right-hand sides are monotonic. The latter assumption, though, is not met in presence of widening. It is also not met by equation

systems corresponding to context-sensitive interprocedural analysis, possibly combining context-sensitive analysis of local information with flow-insensitive analysis of globals.

As a remedy, we present a novel operator that combines a given widening operator with a given narrowing operator. We present adapted versions of round-robin as well as of worklist iteration, local and side-effecting solving algorithms for the combined operator and prove that the resulting solvers always return sound results and are guaranteed to terminate for monotonic systems whenever only finitely many unknowns are encountered.

3.18 Automating Software Analysis at Large Scale

Michael Tautschnig (Queen Mary University of London, GB)

Main reference Michael Tautschnig, "Automating Software Analysis at Large Scale," in Proc. of the 2014 Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS'14), to appear.

Software model checking tools promise to deliver genuine traces to errors, and sometimes even proofs of their absence. As static analysers, they do not require concrete execution of programs, which may be even more beneficial when targeting new platforms. Academic research focusses on improving scalability, yet largely disregards practical technical challenges to make tools cope with real-world code. The Debian/GNU Linux distribution proved to provide a perfect basis for experimenting with those tools. Initial experiments lead to a number of improvements in tools, but also more than 500 bug reports.

3.19 Automatic Inference of Ranking Functions by Abstract Interpretation

Caterina Urban (ENS – Paris, FR)

We present a family of parameterized abstract domains for proving termination of imperative programs by abstract interpretation. The domains automatically synthesize piecewise-defined lexicographic ranking functions and infer sufficient preconditions for program termination. The abstract domains are parameterized by a numerical abstract domain for state partitioning and a numerical abstract domain for ranking functions. This parameterization allows to easily tune the trade-off between precision and cost of the analysis. We describe instantiations of these domains with intervals, octagons, polyhedra and affine functions. We have implemented a prototype static analyzer for proving conditional termination of programs written in (a subset of) C and, using experimental evidence, we show that it is competitive with the state of the art and performs well on a wide variety of benchmarks.

3.20 Byte-Precise Verification of Low-Level List Manipulation

Tomas Vojnar (Technical University of Brno, CZ)

Main reference K. Dudka, P. Peringer, T. Vojnar, "Byte-Precise Verification of Low-Level List Manipulation," Brno University of Technology, Technical Report, No. FIT-TR-2012-04, 2013.
URL http://www.fit.vutbr.cz/~vojnar/Publications/FIT-TR-2012-04.pdf

We propose a new approach to shape analysis of programs with linked lists that use low-level memory operations. Such operations include pointer arithmetic, safe usage of invalid pointers, block operations with memory, reinterpretation of the memory contents, address alignment, etc. Our approach is based on a new representation of sets of heaps, which is to some degree inspired by works on separation logic with higher-order list predicates, but it is graph-based and uses a more fine-grained (byte-precise) memory model in order to support the various low-level memory operations. The approach was implemented in the Predator tool and successfully validated on multiple non-trivial case studies that are beyond the capabilities of other current fully automated shape analysis tools.

The result is a joint work with Kamil Dudka and Petr Peringer. The work was originally published at SAS'13. The Predator tool is available here: http://www.fit.vutbr.cz/research/groups/verifit/tools/predator/

3.21 System LAV and Automated Evaluation of Students' Programs

Milena Vujosevic-Janicic (University of Belgrade, RS)

Joint work of Vujosevic-Janicic, Milena; Kuncak, Viktor;
Main reference M. Vujošević-Janičić, V. Kuncak, "Development and Evaluation of LAV: An SMT-Based Error Finding Platform," in Proc. of the 4th Int'l Conf. on Verified Software: Theories, Tools, Experiments (VSTTE'12), LNCS, Vol. 7152, pp. 98–113, Springer, 2012.
URL http://dx.doi.org/10.1007/978-3-642-27705-4_9

In this talk, we give a short overview of a software verification tool LAV [1], we present challenges and experiences in applying verification techniques in automated evaluation of students' programs [2] and discuss our ongoing work on regression verification of students' programs.

LAV is an open-source tool for statically verifying program assertions and locating bugs such as buffer overflows, pointer errors and division by zero. It integrates into the popular LLVM infrastructure for compilation and analysis. Combining symbolic execution and SAT encoding of program's behaviour, LAV generates polynomial-size verification conditions for loop-free code, while for modelling loops it can use both under- or over- approximation techniques. Generated verification conditions are passed to one of the several SMT solvers: Boolector, MathSAT, Yices, and Z3.

Software verification tools are not commonly applied in automated evaluation of students' programs, although precise and reliable automated grading techniques are of big importance for both classical and on-line programming courses. We ran LAV on a corpus of students' programs, observed advantages and challenges of using verification in this context and we showed that verification techniques can significantly improve the automated grading process. LAV outperformed (concerning time, bugs found and false alarms) black-box fuzzing

techniques that are commonly used for bug finding in students' programs and successfully met all the specific requirements posed by evaluation process.

Our ongoing research focuses on functional correctness of small-sized programs written by students at introductory courses. We explore automatic assessment of functional correctness by regression verification (where the specification of a student's program is given as a teacher's program). Although this problem is undecidable in general, regression verification can give useful results and enhance automated grading process in some cases.

References

1 M. Vujošević-Janičić and V. Kuncak. Development and Evaluation of LAV: An SMT-Based Error Finding Platform. In *VSTTE*, volume 7152 of *LNCS*, pages 98–113. Springer, 2012.
2 M. Vujošević-Janičić, M. Nikolić, D. Tošić, and V. Kuncak. Software verification and graph similarity for automated evaluation of students' assignments. *Information and Software Technology*, 55(6):1004–1016, 2013. *Elsevier*.

4 Tool Demonstrations

4.1 aiT Worst-Case Execution Time Analysis

Christian Ferdinand (AbsInt – Saarbrücken, DE)

AiT WCET Analyzers statically compute tight bounds for the worst-case execution time (WCET) of tasks in real-time systems. They directly analyze binary executables and take the intrinsic cache and pipeline behavior into account.

4.2 The Goanna Static Analyzer

Ralf Huuck (NICTA – Sydney, AU)

We present some of the features and capabilities of our source code analyser Goanna. In particular, we show the IDE integration and usage in Visual Studio. This includes analysis features such as interprocedural tracing, selection for various coding standards and the bug management dashboard. Furthermore, we explain the Linux command line interface and demonstrate some exemplary bug finding capabilities. Finally, we present a number of benchmark detection results and open questions for future work.

4.3 Cccheck/Clousot

Francesco Logozzo (Microsoft Research – Redmond, US)

I demo cccheck, the popular abstract interpretation-based verifier for .NET. The demo includes:

1. finding bugs in C# programs;
2. provide *automatic* code fixes for such bugs;
3. show how Clousot infers contracts and it is also able to prove that a method computes the max of an array.

4.4 LLBMC: The Low-Level Bounded Model Checker

Carsten Sinz (KIT – Karlsruher Institut für Technologie, DE)

URL http://llbmc.org

We present LLBMC, the low-level bounded model checker. LLBMC implements a bounded model checking algorithm complemented by a rewriting approach to simplify verification conditions and prove simple properties. LLBMC is fully automatic and requires minimal preparation efforts and user interaction. It supports all C constructs, including not so common features such as bitfields. LLBMC models memory accesses (heap, stack, global variables) with high precision and is thus able to find hard-to-detect memory access errors like heap or stack buffer overflows. LLBMC can also uncover errors due to uninitialized variables or other sources of non-deterministic behavior. Due to its precise analysis, LLBMC produces almost no false alarms (false positives).

We demonstrate the features of LLBMC on three examples: the first is on checking equivalence of two programs containing many bit-wise logical operations; the second explains LLBMC's precise modeling of memory on a program where writing to memory has an unexpected effect on the control flow; the third example presents LLBMC's ability to derive lambda-expressions for loops with array updates.

For further information on LLBMC see http://llbmc.org. An evaluation / academic version can also be downloaded from this URL.

4.5 FuncTion

Caterina Urban (ENS – Paris, FR)

We present FuncTion, a research prototype static analyzer able to infer piecewise-defined ranking functions for programs written in (a subset of) C language. In particular, we present FuncTion's web interface and we demonstrate the features and capabilities of the analyzer by means of a few exemplary programs.

4.6 Predator: A Shape Analyzer Based on Symbolic Memory Graphs

Tomas Vojnar (Technical University of Brno, CZ)

URL http://www.fit.vutbr.cz/research/groups/verifit/tools/predator

Predator is a shape analyzer that uses the abstract domain of symbolic memory graphs (SMGs) in order to support various forms of low-level memory manipulation commonly used in optimized C code. Predator is implemented as a GCC (GNU Compiler Collection) plug-in. Predator is freely available at http://www.fit.vutbr.cz/research/groups/verifit/tools/predator.

5 Discussion Session

5.1 Discussion on "The current limitations of static analysis tools"

On the last day of the seminar a discussion session was held involving people from both academia and industry. The discussion focussed on the current limitations of static analysis tools and how to overcome them.

More specifically, we asked participants to think about:

- How can we improve the usability of static analysis tools and bring them to more users?
- What is the best way to combine algorithms and tools? Do we need a standardized exchange format?
- What is the "best" language for specifying properties and the environment, in which a program is run?
- What has to be done to bring static analysis tools to new fields such as security and privacy?

During the discussion it was observed that the most notable problems of current tools are:

- Annotations and specifications, which are essential to obtain precise analysis results and fewer false positives, are not standardized and can often not be exchanged between tools.
- There is a lack in detailed comparisons between static analysis tools, which makes it more difficult for a possible user to decide which tool to apply.
- Static analysis tools and compilers are not sufficiently integrated.

It was also argued that standardized specification and annotation languages exist, but are insufficient and thus not frequently used. It was also brought forward that even though there are benchmarks and competitions for analysis tools, it is still hard for a non-expert to decide which tool is most appropriate for a specific purpose. More work is needed to comparatively describe features and strengths of individual tools.

Participants

- Roberto Bagnara
BUGSENG & University of Parma, IT
- Dirk Beyer
Universität Passau, DE
- Mehdi Bouaziz
ENS, Paris, FR
- Patrick Cousot
ENS, Paris, FR
- Tomasz Dudziak
Universität des Saarlandes, DE
- David Faragó
KIT – Karlsruher Institut für Technologie, DE
- Christian Ferdinand
AbsInt, Saarbrücken, DE
- Jérôme Feret
ENS, Paris, FR
- Marcelo Frias
University of Buenos Aires, AR
- Vijay Ganesh
University of Waterloo, CA
- Roberto Giacobazzi
University of Verona, IT
- Christoph Gladisch
KIT – Karlsruher Institut für Technologie, DE
- Udo Gleich
Daimler Research, Ulm, DE
- Manuel Hermenegildo
IMDEA Software, Madrid, ES
- Ralf Huuck
NICTA, Sydney, AU
- Daniel Kroening
University of Oxford, GB
- K. Rustan M. Leino
Microsoft Res., Redmond, US
- Francesco Logozzo
Microsoft Res., Redmond, US
- Peter Müller
ETH Zürich, CH
- Filip Niksic
MPI-SWS, Kaiserslautern, DE
- Andreas Podelski
Universität Freiburg, DE
- Hendrik Post
Robert Bosch GmbH – Stuttgart, DE
- Francesco Ranzato
University of Padova, IT
- Xavier Rival
ENS, Paris, FR
- Helmut Seidl
TU München, DE
- Carsten Sinz
KIT – Karlsruher Institut für Technologie, DE
- Michael Tautschnig
Queen Mary University of London, GB
- Shmuel Tyszberowicz
Academic College of Tel Aviv Yaffo, IL
- Caterina Urban
ENS, Paris, FR
- Tomas Vojnar
Technical University of Brno, CZ
- Milena Vujosevic-Janicic
University of Belgrade, RS
- Reinhard Wilhelm
Universität des Saarlandes, DE