Report from Dagstuhl Seminar 14351

Decision Procedures and Abstract Interpretation

Edited by

Daniel Kroening[1], Thomas W. Reps[2], Sanjit A. Seshia[3], and Aditya V. Thakur[4]

1 **University of Oxford, GB,** kroening@cs.ox.ac.uk
2 **University of Wisconsin-Madison, US,** reps@cs.wisc.edu
3 **University of California, Berkeley, US,** sseshia@eecs.berkeley.edu
4 **University of California, Berkeley, US,** thakur@berkeley.edu

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 14351 "Decision Procedures and Abstract Interpretation". The seminar brought together practitioners and reseachers in abstract interpretation and decision procedures. The meeting highlighted the connections between the two disciplines, and created new links between the two research communities. Joint activities were also conducted with the participants of Dagstuhl Seminar 14352 "Next Generation Static Software Analysis Tools", which was held concurrently.

Seminar August 24–29, 2014 – http://www.dagstuhl.de/14351
1998 ACM Subject Classification D.2.4 Software/Program Verification. F.3.1 Specifying and Verifying and Reasoning about Programs. F.3.2 Semantics of Programming Languages. F.4.1 Mathematical Logic
Keywords and phrases Program analysis, Abstract interpretation, Abstract domain, Fixed-point finding, Satisfiability checking, Satisfiability modulo theories, Decision procedures, Constraint programming
Digital Object Identifier 10.4230/DagRep.4.8.89

1 Executive Summary

Aditya Thakur

The seminar was successful in bringing the following two communities together:

- designers and implementors of abstract interpreters, and
- designers and implementors of decision procedures.

The abstract interpretation (AI) and decsion procedure (DP) communities have several interests in common. Tools created by each of these communities can be viewed as using symbolic techniques to explore the state state of a transition system. However, the respective repertoires of techniques used in the two disciplines are quite different, and each community has its own mindset and outlook. The seminar sought to capitalize on recent ideas that demonstrated new connections between the two disciplines, and, consequently, promote the cross-fertilization between the areas at a deep technical level.

The seminar had 27 participants from both the AI and DP communities. To keep pariticipants from both areas engaged during a session, the organizers refrained from filling a session only with talks focusing on a particular community. Instead, each session consisted of

talks by participants of both research communities. Furthermore, talks by young researchers were scheduled earlier in the week, which enabled them to get better feedback on their research.

The Dagstuhl Seminar 14352 "Next Generation Static Software Analysis Tools" was held concurrently with Dagstuhl Seminar 14351. There were a number of joint activities organized to foster interaction among participants of the two seminars:

- The first session on Monday was a joint session for participants of both seminars. In this session, all participants introduced themselves and briefly described their research interests. Furthermore, Patrick Cousot, an organizer for Seminar 14352, and Thomas Reps, an organizer for Seminar 14351, each gave a "scene-setting" talk.
- The Wednesday excursion to the steel mill and Egyptian exhibit was organized as a joint activity.
- A joint session was organized on Thursday afternoon. The talks in this session were given by participants of both seminars.
- The seating arrangement for the Friday dinner was organized so that participants from both seminars sat together.
- The schedule of talks for both seminars was shared with all participants. Hence, participants of one seminar were able to attend a specific talk in the other seminar, if they felt the talk was especially relevant.

Apart from the planned activities listed above, the week saw a lot of informal discussions among participants of these two seminars in the evenings.

The seminar also featured talks about two other research areas: constraint programming (CP) and machine learning (ML). The talks by Mine, Rueher, and Truchet highlighted the use of abstract interpretation in CP. The talks by Reps, Seshia, Sharma, and Thakur discussed the application of ML techniques, such as inductive learning, to problems in AI and DP. Both these sets of talks garnered interesting discussions about the connections among all these various reseach areas. Furthermore, this discussion indicates that future seminars should include even more researchers and practitioners from not just the AI and DP communities, but also the CP and ML communities.

2 Table of Contents

3 Overview of Talks

3.1 Symbolic Optimization with SMT Solvers

Aws Albarghouthi (University of Toronto, CA)

Joint work of Li, Yi; Albarghouthi, Aws; Kincaid, Zachary; Gurfinkel, Arie; Chechik, Marsha
Main reference Y. Li, A. Albarghouthi, Z. Kincaid, A. Gurfinkel, M. Chechik, "Symbolic Optimization with SMT Solvers," in Proc. of the 41st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'14), pp. 607–618, ACM, 2014.
URL http://dx.doi.org/10.1145/2535838.2535857

The rise in efficiency of Satisfiability Modulo Theories (SMT) solvers has created numerous uses for them in software verification, program synthesis, functional programming, refinement types, etc. In all of these applications, SMT solvers are used for generating satisfying assignments (e. g., a witness for a bug) or proving unsatisfiability/validity(e. g., proving that a subtyping relation holds). We are often interested in finding not just an arbitrary satisfying assignment, but one that optimizes (minimizes/maximizes) certain criteria. For example, we might be interested in detecting program executions that maximize energy usage (performance bugs), or synthesizing short programs that do not make expensive API calls. Unfortunately, none of the available SMT solvers offer such optimization capabilities.

In this talk, I describe SYMBA, an efficient SMT-based optimization algorithm for objective functions in the theory of linear real arithmetic (LRA). Given a formula Phi and an objective function t, SYMBA finds a satisfying assignment of Phi; that maximizes the value of t. SYMBA utilizes efficient SMT solvers as black boxes. As a result, it is easy to implement and it directly benefits from future advances in SMT solvers. Moreover, SYMBA can optimize a set of objective functions, reusing information between them to speed up the analysis. We have implemented SYMBA and evaluated it on a large number of optimization benchmarks drawn from program analysis tasks, namely, symbolic abstraction for a large family of numerical abstract domains. Our results indicate the power and efficiency of SYMBA in comparison with competing approaches, and highlight the importance of its multi-objective-function feature.

3.2 Spatial Interpolants

Joshua Berdine (Microsoft Research UK – Cambridge, GB)

Joint work of Albargouthi, Aws; Berdine, Josh; Cook, Byron; Kincaid Zachary

We propose SplInter, a new technique for proving safety properties of heap manipulating programs that marries (1) a new separation logic–based analysis for heap reasoning with (2) an interpolation-based technique for refining and strengthening heap shape invariants with data invariants. SplInter is property-directed, precise, and produces counterexample traces in case a property does not hold. Using the novel notion of spatial interpolants modulo theories, SplInter can infer complex invariants over general recursive predicates, e. g., of the form all data elements in a linked list are even or a binary tree is sorted. Furthermore, we treat interpolation as a black box, which gives us the freedom to encode data manipulation in whatever theory is suitable for the program at hand (e. g., bitvectors, arrays, or linear

arithmetic), so that our technique immediately benefits from any future advances in SMT solving and interpolation.

3.3 The Transformer Refinement Prover – A Work in (Lack of) Progress Talk

Martin Brain (University of Oxford, GB)

Over the past few years there have been a number of interesting papers showing that algorithms used for SAT solving (Stalmarck's, DPLL, CDCL) can be lifted to work over abstract domains. This raises the question of whether the architecture of a SAT solver can be lifted to produce a generic solver which can be parameterised with different abstract domains to produce a range of concrete decision procedures. The Transformer Refinement Prover (TRP) is an attempt to build such a tool. This talk discusses some of the issues engineering, architectural and conceptual that have been encountered during the construction.

3.4 A formal approach to the Analysis of Reliability Architectures

Alessandro Cimatti (Bruno Kessler Foundation – Trento, IT)

Joint work of Bozzano, Marco; Cimatti, Alessandro; Mattarei, Cristian
Main reference M. Bozzano, A. Cimatti, C. Mattarei, "Efficient Analysis of Reliability Architectures via Predicate Abstraction," in Proc. of the 9th Int'l Haifa Verification Conf. (HVC'13), LNCS, Vol. 8244, pp. 279–294, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-319-03077-7_19

The development of complex and critical systems calls for a rigorous and thorough evaluation of reliability aspects. Over the years, several methodologies have been introduced in order to aid the verification and analysis of such systems. Despite this fact, current technologies are still limited to specific architectures, without providing a generic evaluation of redundant system definitions.

In this talk, we present a novel approach able to assess the reliability of an arbitrary combinatorial redundant system. We rely on an expressive modeling language to represent a wide class of architectural solutions to be assessed. On such models, we provide a portfolio of automatic analysis techniques: we can produce a fault tree, that represents the conditions under which the system fails to produce a correct output; from it, we can extract a function over the components reliability, which represents the failure probability of the system. At its core, the approach relies on the logical formalism of equality and uninterpreted functions. Advanced automated reasoning techniques, in particular Satisfiability Modulo Theories decision procedures, and Predicate Abstraction, are suitably combined to achieve efficiency.

We carried out an extensive experimental evaluation of the proposed approach on a wide class of multi-stage redundant systems. We are able to obtain, in a fully automated manner, all the results that are manually obtained in previous works, and we cover a much wider class of architectures, demonstrating scalability for a large number of components, thus enabling the analysis of complex architectures of realistic size.

3.5 Lifting Satisfiability Procedures to Reachability Analysis

Vijay DSilva (Google – San Francisco, US)

Joint work of D'Silva, Vijay; Haller, Leopold; Kroening, Daniel
Main reference V. D'Silva, L. Haller, D. Kroening, "Abstract Conflict Driven Learning," in Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'13), pp. 143–154, ACM, 2013.
URL http://dx.doi.org/10.1145/2429069.2429087

There have been many attempts at lifting ideas from the satisfiability literature to program analysis. The intuition behind the abstract satisfaction approach is that the objects manipulated by satisfiability solvers satisfy similar axioms to abstract domains used in program analysis. An immediate consequence of identifying these axioms is that satisfiability procedures can be lifted to all structures satisfying these axioms. In this work, we show that the DPLL and Conflict Driven Clause Learning (CDCL) algorithms lift to certain families of lattices and transformers used for reachability analysis. The resulting reachability analyzers automatically refine fixed points using the notions of decisions and learning. This approach has been applied successfully to bound the error of floating point computations in embedded software.

References

1 Vijay D'Silva, Leopold Haller and Daniel Kroening. *Satisfiability Solvers are Abstract Interpreters*. Static Analysis Symposium, 2012
2 Vijay D'Silva, Leopold Haller and Daniel Kroening. *Abstract Conflict Driven Learning*. Principles of Programming Languages, 2013

3.6 Solving Exists/Forall Problems With SMT

Bruno Dutertre (SRI – Menlo Park, US)

Joint work of Dutertre, Bruno; Jovanovic, Dejan

We describe an algorithm for solving problems of the form exists x. for all y. P(x, y) by relying of two SMT or SAT solvers. One solver searches for candidate x while the other attempts to refute x by exhibiting a y for which P(x, y) is false. A key component of this algorithm is generalizing from a counterexample y. We describe generalization methods that work for different quantification domains, including a method based on model-guided virtual term substitution.

3.7 Predicate Abstraction in IC3

Alberto Griggio (Bruno Kessler Foundation – Trento, IT)

Joint work of Cimatti, Alessandro; Griggio, Alberto; Mover, Sergio; Tonetta, Stefano
Main reference A. Cimatti, A. Griggio, S. Mover, S. Tonetta, "IC3 Modulo Theories via Implicit Predicate Abstraction," in Proc. of the 20th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'14), LNCS, Vol. 8413, pp. 46–61, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-642-54862-8_4

We present a novel approach for generalizing the IC3 algorithm for invariant checking from finite-state to infinite-state transition systems, expressed over some background theories. The procedure is based on a tight integration of IC3 with Implicit (predicate) Abstraction, a technique that expresses abstract transitions without computing explicitly the abstract system and is incremental with respect to the addition of predicates. In this scenario, IC3 operates only at the Boolean level of the abstract state space, discovering inductive clauses over the abstraction predicates. Theory reasoning is confined within the underlying SMT solver, and applied transparently when performing satisfiability checks. When the current abstraction allows for a spurious counterexample, it is refined by discovering and adding a sufficient set of new predicates. Importantly, this can be done in a completely incremental manner, without discarding the clauses found in the previous search.

3.8 Property Directed Polyhedral Abstraction

Arie Gurfinkel (Carnegie Mellon University – Pittsburgh, US)

Joint work of Gurfinkel, Arie; Bjørner, Nikolaj

We show how to combine the benefits of Polyhedral Abstract Interpretation (poly-AI) with the flexibility of Property Directed Reachability (PDR) algorithms for computing safe inductive convex polyhedral invariants. We develop two algorithms that integrate Poly-AI with PDR and show their benefits on a prototype in Z3 using a preliminary evaluation. The algorithms mimic the traditional forward Kleene and a chaotic backward iterations, respectively. Our main contribution is to show how to replace the expensive convex hull and quantifier elimination computations, a major bottleneck in poly-AI, with a lazy property-directed algorithms based on interpolation and model-based projection. Our approach integrates seamlessly within the framework of PDR adapted to Linear Real Arithmetic, and allows to dynamically decide between computing convex and non-convex invariants as directed by the property.

3.9 The PAGAI static analyzer

Julien Henry (VERIMAG – Gières, FR)

Joint work of Henry, Julien; Monniaux, David; Moy, Matthieu

Pagai is a static analyzer based on combinations of abstract interpretation and SMT, that computes numerical invariants for LLVM bitcode. Abstract interpretation can be made more precise by distinguishing every paths inside loops for delaying least upper bounds, at the expense of an exponential blowup. SMT allows symbolic and sparse representation of sets of paths, and the fixpoint computation is guided by SMT queries. We present early but promising experimental results of PAGAI on the SV-Comp benchmarks. In a second part, we present a new approach to the estimation of Worst-Case execution time, by defining the problem as an instance of optimization modulo theory. Naive encodings of the problem into SMT lead to formulas intractable for any production- grade solver based on DPLL(T). We show that simple static pre-analysis of program fragments provide invariants that dramatically improve the efficiency of the SMTsolver on these examples.

3.10 Projection using Parametric Objectives

Jacob Howe (City University – London, GB)

Joint work of Howe, Jacob; King, Andy
Main reference J. M. Howe, A. King, "Polyhedral Analysis using Parametric Objectives," in Proc. of the 19th Int'l Symp. on Static Analysis (SAS'12), LNCS, Vol. 7460, pp. 41–57, Springer, 2012.
URL http://dx.doi.org/10.1007/978-3-642-33125-1_6

The abstract domain of polyhedra lies at the heart of many program analysis techniques. However, its operations can be expensive, precluding their application to polyhedra that involve many variables. This talk describes a new approach to computing polyhedral domain operations. The core of this approach is an algorithm to calculate variable elimination (projection) based on parametric linear programming. The algorithm enumerates only non-redundant inequalities of the projection space, hence permits anytime approximation of the output. Some preliminary data from experiments are included.

3.11 Abstract Conflict-Driven Learning

Daniel Kroening (University of Oxford, GB)

Joint work of D'Silva, Vijay; Haller, Leopold; Kroening, Daniel
Main reference V. D'Silva, L. Haller, D. Kroening, "Abstract Conflict Driven Learning," in Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL'13), pp. 143–154, ACM, 2013.)
URL http://dx.doi.org/10.1145/2429069.2429087

Modern satisfiability solvers implement an algorithm, called Conflict Driven Clause Learning, which combines search for a model with analysis of conflicts. We show that this algorithm can be generalised to solve the lattice-theoretic problem of determining if an additive

transformer on a Boolean lattice is always bottom. Our generalised procedure combines overapproximations of greatest fixed points with underapproximations of least fixed points to obtain more precise results than computing fixed points in isolation. We generalise implication graphs used in satisfiability solvers to derive underapproximate transformers from overapproximate ones. Our generalisation provides a new method for static analyzers that operate over non-distributive lattices to reason about properties that require disjunction.

3.12 A method to infer inductive numeric invariants inspired by Constraint Programming

Antoine Mine (ENS – Paris, FR)

Joint work of Mine, Antoine; Truchet, Charlotte; Sankaranarayanan, Sriram

In this talk, we suggest the idea of using algorithms inspired by Constraint Programming in order to infer inductive invariants on numeric programs. Similarly to Constraint Programming solvers on continuous domains, our algorithm approximates the problem from above, using decreasing iterations that may split, discard, and tighten axis-aligned boxes. If successful, the algorithm outputs a set of boxes that includes the initial states and is a post-fixpoint of the abstract semantic function of interest. Our work is very preliminary; many improvements still need to be performed to determine if the method is usable in practice, and in which contexts. Nevertheless, we show that a naive proof-of-concept implementation of our algorithm is already capable of inferring non-trivial inductive invariants that would otherwise require the use of relational or even non-linear abstract domains when using more traditional abstract interpretation iteration methods.

3.13 Automating Separation Logic with Trees and Data Using SMT Solvers

Ruzica Piskac (Yale University, US)

Joint work of Piskac, Ruzica; Wies, Thomas; Zufferey, Damien
Main reference R. Piskac, T. Wies, D. Zufferey, "Automating Separation Logic with Trees and Data," in Proc. of the 26th Int'l Conf. on Computer Aided Verification (CAV'14), LNCS, Vol. 8559, pp. 711–728, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-08867-9_47

Separation logic (SL) follows a discipline of local reasoning that mimics human intuition about how to prove the correctness of heap-manipulating programs. Central to this discipline is the frame rule, a Hoare logic proof rule that decomposes the global heap into a footprint, the region on which a program fragment operates, and a frame, the region that remains untouched by the program fragment. Automation of the frame rule involves the actual inference of the frame from SL assertions expressing the global heap and the footprint. In this talk, I present Grasshopper, a tool for compositional verification of heap-manipulating programs. What makes our tool unique is its decidable specification language, which supports mixing of assertions expressed in separation logic and first-order logic. We achieve this combination of specification languages through a translation to programs whose specifications are expressed in a decidable fragment of first-order logic. This logic is well-suited for automation using SMT solvers.

3.14 Verification with Recursive Functions

Régis Blanc (EPFL – Lausanne, CH)

Joint work of Blanc, Régis; Kuncak, Viktor; Suter, Philippe; Kneuss, Etienne
Main reference R. Blanc, V. Kuncak, E. Kneuss, P. Suter, "An overview of the Leon verification system: verification by translation to recursive functions," in Proc. of the 4th Workshop on Scala (SCALA'13), Article No. 1, ACM, 2013.
URL http://dx.doi.org/10.1145/2489837.2489838

We present the Leon system, a verifier for a subset of the Scala programming language. Along with several functional features of Scala, Leon supports imperative constructs such as mutations and loops, using a translation into recursive functional form. Both properties and programs in Leon are expressed in terms of user-defined functions. We discuss several techniques that led to an efficient semi-decision procedure for first-order constraints with recursive functions, which is the core solving engine of Leon. We illustrate the current capabilities of Leon on an interactive web interface.

3.15 Setting the Scene for "Decision Procedures and Abstract Interpretation"

Thomas W. Reps (University of Wisconsin – Madison, US)

Joint work of Reps, Thomas W.; Thakur, Aditya V.
Main reference T. W. Reps, A. V. Thakur, "Through the lens of abstraction," presentation at the 2014 High Confidence Software and Systems Conf. (HCSS'14); manuscript available from author's webpage.
URL http://www.cs.wisc.edu/wpis/papers/hcss14-final-version.pdf

This talk is intended as a "scene-setting" talk about Seminar 14351 for the benefit of the participants of both Seminars 14351 and 14352. It presents a somewhat personal view of the opportunity offered by the seminar, and concentrates mainly on two topics. The first is the use of logic to support abstract interpretation (i. e., for performing alpha on a set of states described by a formula, for applying the best transformer, for creating a representation of the best transformer, and for creating the reduced product of two or more values). The second is the use of abstract interpretation to support decision procedures better (e. g., by reverse-engineering existing decision procedures to identify uses of abstract domains, which allows them to be generalized by using more expressive abstract domains; and by using logic-based abstraction methods directly for unsatisfiability checking and validity checking).

The seminar is intended to expose members of two communities to each other, namely, (i) designers/implementers of abstract interpreters, and (ii) designers/implementers of decision procedures. One connection between the two communities is that the tools that are created by their respective members can be viewed as using symbolic techniques to explore a state space. However, the repertoires of techniques used in the two disciplines are quite different, and each has its own mindset and outlook. The ideas and methods presented in the talk demonstrate new connections between the two disciplines, and suggest that the time is ripe for cross-fertilization between them to occur.

3.16 On Suspicious Intervals for Floating-Point Number Programs

Michel Rueher (University of Nice, FR)

Joint work of Ponsini, Olivier ; Michel, Claude; Rueher, Michel
Main reference O. Ponsini, C. Michel, M. Rueher, "Verifying floating-point programs with constraint programming and abstract interpretation techniques," Automated Software Engineering, published online, 2014.
URL http://dx.doi.org/10.1007/s10515-014-0154-2

Programs with floating-point computations are often derived from mathematical models or designed with the semantics of the real numbers in mind. However, for a given input, the computed path with floating-point numbers may differ from the path corresponding to the same computation with real numbers. State-of-the-art tools compute an over- approximation of the error introduced by floating-point operations with respect to the same sequence of operations in an idealized semantics of real numbers. Thus, totally inappropriate behaviours of a program may be dreaded but the developer does not know whether these behaviours will actually occur, or not. That is why it is very important to estimate the accuracy of floating-point computations with respect to the same sequence of operations in an idealized semantics of real numbers. To tackle this problem, we will present some capabilities of CP techniques for: a) Computing tight approximations for value analysis; b) Identify suspicious values. The crux of the matter is the accuracy of the estimation of floating-point computations because a rough approximation may generate numerous false alarms. We show that a hybrid approach for value analysis of floating-point programs that combines abstract interpretation and CP techniques is more effective than static analyser and CP solvers, when used separately. Interestingly, the refutation capabilities of CP solvers over floating-point numbers and over real numbers can significantly improve the precision of the domains computed by abstract interpretation. When the approximation remains nevertheless too rough, CP techniques can also help to identify suspicious values, that is values for which the behaviour of the program over the floating-point numbers is significantly different from the behaviour one could expect over the real numbers. In other words, for verifying whether a program can actually produce values inside the part of the approximation over the floats that intersect with a forbidden interval.

3.17 Inferring Invariants by Strategy Iteration

Peter Schrammel (University of Oxford, GB)

Joint work of Daniel Monniaux; Schrammel, Peter
Main reference D. Monniaux, P. Schrammel, "Speeding Up Logico-Numerical Strategy Iteration," in Proc. of the 21st Int'l Symp. on Static Analysis (SAS'14), LNCS, Vol. 8723, pp. 253–267, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-10936-7_16

Template polyhedral analysis abstracts numerical variables inside a program by one polyhedron per control location, with a priori fixed directions for the faces. The strongest inductive invariant in such an abstract domain may be computed by a combination of strategy iteration and SMT solving. Unfortunately, the above approaches lead to unacceptable space and time costs if applied to a program whose control states have been partitioned according to predicates. We therefore propose a modification of the strategy iteration algorithm where the strategies are stored succinctly, and the linear programs to be solved at each iteration step are simplified according to an equivalence relation.

3.18 Solvers, Abstraction, and Inductive Learning

Sanjit A. Seshia (University of California – Berkeley, US)

Main reference S. A. Seshia, "Sciduction: Combining Induction, Deduction, and Structure for Verification and Synthesis," in Proc. of the 49th Annual Design Automation Conf. (DAC'12), pp. 356–365, ACM, 2012; pre-print available from author's webpage.
URL http://dx.doi.org/10.1145/2228360.2228425
URL http://www.eecs.berkeley.edu/~sseshia/pubs/b2hd-seshia-dac12.html

This talk seeks to make connections between three topics: decision procedures (SMT solvers), abstraction and abstract interpretation, and inductive learning (machine learning). There are three main messages in the talk. First, we make the point that many verification tasks are effectively solved through "reduction to synthesis". Examples include the generation of inductive invariants for proofs by induction, and the generation of abstract models for abstraction-based verification. Second, the resulting synthesis problems can be tackled through a combination of induction (learning from examples), deduction, and a structure hypothesis. An example is the counterexample-guided inductive synthesis (CEGIS) paradigm. We compare CEGIS with "traditional" machine learning algorithms. Finally, we pose some fundamental questions about the efficiency and convergence (termination) of CEGIS. Initial results are presented that draw from results in machine learning theory.

3.19 Data-Driven Invariant Inference

Rahul Sharma (Stanford University, US)

Joint work of Sharma, Rahul;Alex Aiken; Saurabh Gupta; Bharath Hariharan; Aditya Nori
Main reference R. Sharma, A. Aiken, "From Invariant Checking to Invariant Inference Using Randomized Search," in Proc. of the 26th Int'l Conf. on Computer Aided Verification (CAV'14), LNCS, Vol. 8559, pp. 88–105, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-08867-9_6

We discuss two applications that leverage concrete states to improve invariant inference. First, for many abstract interpretations, concrete states can help reduce the number of fixpoint iterations required to reach convergence. In the second application, concrete states guide a search based invariant inference engine. The main advantage of a search based procedure is the generality and we show how to retarget our procedure to infer invariants for many different domains.

3.20 Verification using Small and Short Worlds

Rohit Sinha (University of California – Berkeley, US)

Joint work of Sinha, Rohit; Sturton, Cynthia; Maniatis, Petros; Seshia, Sanjit; Wagner, David
Main reference R. Sinha, C. Sturton, P. Maniatis, S. A. Seshia, D. Wagner, "Verification with Small and Short Worlds," in Proc. of the 2012 IEEE Int'l Conf. on Formal Methods in Computer-Aided Design (FMCAD'12), pp. 68–77, IEEE, 2012.
URL http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6462557

We consider the verification of safety properties in systems with large arrays and data structures. Such systems are common at the low levels of software stacks; examples are

hypervisors and CPU emulators. The very large data structures in such systems (e. g., address-translation tables and other caches) make automated verification based on straightforward state-space exploration infeasible. We present S2W, a new abstraction-based model-checking methodology to facilitate automated verification of such systems. As a first step, inductive invariant checking is performed. If that fails, we compute an abstraction of the original system by precisely modeling only a subset of state variables while allowing the rest of the state to evolve arbitrarily at each step. This subset of the state constitutes a "small world" hypothesis, and is extracted from the property. Finally, we verify the safety property on the abstract model using bounded model checking. We ensure the verification is sound by first computing a bound on the reachability diameter of the abstract model. For this computation, we developed a set of heuristics that we term the "short world" approach.

3.21 More Algorithms for Symbolic Abstraction

Aditya Thakur (University of Wisconsin – Madison, US)

Joint work of Thakur, Aditya; Reps, Thomas; Breck, Jason

This talk describes two algorithms for performing *symbolic abstraction* [1]: Given a formula φ in a logic $\mathcal{L}$ and an abstract domain $\mathcal{A}$, the symbolic abstraction of φ is the strongest consequence of φ that is expressible in $\mathcal{A}$. Symbolic abstraction has a dual use: it can be used to compute abstract transformers in abstract interpretation, and to check unsatisfiability of a formula.

The talk presents the bilateral framework [2] for performing symbolic abstraction that maintains an over-approximation and under-approximation of the final answer. The algorithm performs symbolic abstraction by intelligently querying an SMT solver. The framework was instantiated to synthesize abstract transformers for machine- code analysis.

The next algorithm for symbolic abstraction is applicable to a new fragment of separation logic (SL) [3]. The algorithm works by performing a bottom-up evaluation of the formula using an abstract domain of shape graphs. This algorithm can be used to check unsatisfiability of an SL formula.

References

1 Reps, T., Sagiv, M., Yorsh, G. (2004). *Symbolic implementation of the best transformer.* In Verification, Model Checking, and Abstract Interpretation (pp. 252-266). Springer Berlin Heidelberg.
2 Thakur, A., Elder, M., Reps, T. (2012). *Bilateral algorithms for symbolic abstraction.* In Static Analysis (pp. 111-128). Springer Berlin Heidelberg.
3 Thakur, A., Breck, J., Reps, T. (2014). *Satisfiability modulo abstraction for separation logic with linked lists* . In Symposium on Model Checking of Software (SPIN) (pp. 58–67). ACM.

3.22 Abstract Domains for Constraint Programming

Charlotte Truchet (University of Nantes, FR)

Joint work of Truchet, Charlotte; Marie Pelleau; Antoine Miné; Frédéric Benhamou
Main reference M. Pelleau, A. Miné, C. Truchet, F. Benhamou, "A Constraint Solver Based on Abstract Domains," in Proc. of the 14th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'13), LNCS, Vol. 7737, pp. 434–454, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-35873-9_26

Constraint Programming (CP) in a domain of Artificial Intelligence that offers a variety of tools to model and solve hard combinatorial problems. Abstract Interpretation (AI) is a domain of Semantics that studies approximations of program traces in order to prove correctness properties. Although they are distinct scientific areas, these two domains have a lot in common. In both cases, we are interested in some set that is hard to compute or intractable: solution set in CP, concrete domain in AI. In both cases, instead of computing this set, we study some over-approximations of it: abstract domains in AI, consistent domains in CP. But the methods differ when the over-approximations are not good enough. CP has developed sophisticated algorithmic mechanisms to exactly solve the problem if the variables are discrete, or reach a given precision if they are continuous. In AI, the abstract domains themselves are refined, either by adding operators, or by increasing their expressivity. In the end, CP provides with solving methods that are very efficient on many NP problems, but are rather monolithic. For instance, they are unable to solve mixed problems with integer and real variables. AI analyzes huge programs using a lot of expressive abstract domains, but does not feature a notion of precision.

In this talk, we showed how to introduce the notion of abstract domain in CP using the example of the Octagons, which offer a good trade between efficiency and expressivity. Then we presented Absolute, a prototype constraint solver without constraints: it is built upon a library of abstract domains called Apron, by Miné and Jeannet. It, thus, naturally copes with mixed problems.

3.23 Ideal Abstractions

Thomas Wies (New York University, US)

Joint work of Damien Zufferey, Tom Henzinger
Main reference D. Zufferey, T. Wies, T. A. Henzinger, "Ideal Abstractions for Well-Structured Transition Systems," in Proc. of the 13th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI'12), LNCS, Vol. 7148, pp. 445–460, Springer, 2102.
URL http://dx.doi.org/10.1007/978-3-642-27940-9_29

Many concurrent infinite state systems can be seen as well-structured transition systems (WSTS). Examples include concurrent programs with shared-memory and dynamic thread creation, as well as distributed message passing systems in the actors framework. WSTS are an attractive class of systems for formal analysis because they admit generic decision procedures for important verification problems such as coverability. Unfortunately, these decision procedures often have very high complexity or provide termination guarantees only in special cases that are not of practical relevance. To obtain a practical analysis with more general termination guarantees, we propose an abstract interpretation that is inspired by decision procedures for the covering problem of WSTS. The abstract domain of our

analysis builds on the ideal completion of the well-quasi-ordered state space to obtain an efficient symbolic representation of infinite sets of states. A widening operator that mimics acceleration-based forward algorithms for computing covering sets ensures termination while controlling the loss of precision of the analysis. I will resent an instance of our analysis framework for the class of depth-bounded WSTS and its application to verifying progress properties of concurrent data structure implementations.

3.24 Parametric Program Analysis

Hongseok Yang (University of Oxford, GB)

Joint work of Hongseok Yang, Mayur Naik, Xin Zhang, Ravi Mangal, Radu Grigore, Ghila Castelnuovo, Mooly Sagiv

Recent years have seen the development of successful commercial programming tools based on static analysis technologies, which automatically verify intended properties of programs or find tricky bugs that are difficult to detect by testing techniques. One of the key reasons for this success is that these tools use clever strategies for abstracting programs—most details about a given program are abstracted away by these strategies, unless they are predicted to be crucial for proving a given property about the program or detecting a type of program errors of interest. Developing such a strategy is, however, nontrivial, and is currently done by a large amount of manual engineering efforts in most tool projects. Finding a good abstraction strategy automatically or even reducing these manual efforts involved in the development of such a strategy is considered one of the main open challenges in the area of program analysis.

In this talk, I will explain how I tried to address this challenge with colleagues in the past few years. During this time, we worked on parametric program analyses, where parameters for controlling the degree of program abstraction are expressed explicitly in the specification of the analyses. For those analyses, we developed algorithms for searching for a desired parameter value with respect to a given program and a given property, which use ideas from the neighbouring areas of program analysis such as testing, searching and optimisation. In my talk, I will describe the main ideas behind these algorithms without going into technical details. I will focus on intuitions about why and when these algorithms work. I will also talk briefly about a few lessons that I learnt while working on this problem.

3.25 A Simple and Scalable Static Analysis for Bound Analysis and Amortized Complexity Analysis

Florian Zuleger (TU Wien, AT)

Joint work of Sinn, Moritz; Zuleger, Florian; Veith, Helmut
Main reference M. Sinn, F. Zuleger, H. Veith, "A Simple and Scalable Static Analysis for Bound Analysis and Amortized Complexity Analysis," in Proc. of the 26th Int'l Conf. on Computer Aided Verification (CAV'14), LNCS, Vol. 8559, pp. 745–761, Springer, 2014.
URL http://dx.doi.org/10.1007/978-3-319-08867-9_50

We present the first scalable bound analysis that achieves amortized complexity analysis. In contrast to earlier work, our bound analysis is not based on general purpose reasoners such as abstract interpreters, software model checkers or computer algebra tools. Rather, we

derive bounds directly from abstract program models, which we obtain from programs by comparatively simple invariant generation and symbolic execution techniques. As a result, we obtain an analysis that is more predictable and more scalable than earlier approaches. We demonstrate by a thorough experimental evaluation that our analysis is fast and at the same time able to compute bounds for challenging loops in a large real-world benchmark.

Technically, our approach is based on lossy vector addition systems (VASS). Our bound analysis first computes a lexicographic ranking function that proves the termination of a VASS, and then derives a bound from this ranking function. Our methodology achieves amortized analysis based on a new insight how lexicographic ranking functions can be used for bound analysis.

Participants

- Aws Albarghouthi
University of Toronto, CA
- Joshua Berdine
Microsoft Research UK – Cambridge, GB
- Régis Blanc
EPFL – Lausanne, CH
- Martin Brain
University of Oxford, GB
- Jörg Brauer
Verified Systems International GmbH – Bremen, DE
- Alessandro Cimatti
Bruno Kessler Foundation – Trento, IT
- Vijay D'Silva
Google – San Francisco, US
- Bruno Dutertre
SRI – Menlo Park, US
- Alberto Griggio
Bruno Kessler Foundation – Trento, IT
- Arie Gurfinkel
Carnegie Mellon University, US
- Julien Henry
VERIMAG – Gières, FR
- Jacob Howe
City University – London, GB
- Daniel Kroening
University of Oxford, GB
- Akash Lal
Microsoft Research India – Bangalore, IN
- Antoine Miné
ENS – Paris, FR
- Ruzica Piskac
Yale University, US
- Thomas W. Reps
University of Wisconsin – Madison, US
- Michel Rueher
University of Nice, FR
- Peter Schrammel
University of Oxford, GB
- Sanjit A. Seshia
University of California – Berkeley, US
- Rahul Sharma
Stanford University, US
- Rohit Sinha
University of California – Berkeley, US
- Aditya Thakur
University of Wisconsin – Madison, US
- Charlotte Truchet
University of Nantes, FR
- Thomas Wies
New York University, US
- Hongseok Yang
University of Oxford, GB
- Florian Zuleger
TU Wien, AT