Report from Dagstuhl Seminar 19062

Bringing CP, SAT and SMT together: Next Challenges in Constraint Solving

Edited by

Sébastien Bardin[1], Nikolaj S. Bjørner[2], and Cristian Cadar[3]

1 **CEA LIST, FR,** sebastien.bardin@cea.fr
2 **Microsoft Research – Redmond, US,** nbjorner@microsoft.com
3 **Imperial College London, GB,** c.cadar@imperial.ac.uk

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 19062 *"Bringing CP, SAT and SMT together: Next Challenges in Constraint Solving"*, whose main goals were to bring together leading researchers in the different subfields of automated reasoning and constraint solving, foster greater communication between these communities and exchange ideas about new research directions.

Constraint solving is at the heart of several key technologies, including program analysis, testing, formal methods, compilers, security analysis, optimization, and AI. During the last two decades, constraint solving has been highly successful and transformative: on the one hand, SAT/SMT solvers have seen a significant performance improvement with a concomitant impact on software engineering, formal methods and security; on the other hand, CP solvers have also seen a dramatic performance improvement, with deep impact in AI and optimization. These successes bring new applications together with new challenges, not yet met by any current technology.

The seminar brought together researchers from SAT, SMT and CP along with application researchers in order to foster cross-fertilization of ideas, deepen interactions, identify the best ways to serve the application fields and in turn help improve the solvers for specific domains.

Seminar February 3–6, 2019 – http://www.dagstuhl.de/19062

2012 ACM Subject Classification Theory of computation → Logic, Theory of computation → Automated reasoning, Mathematics of computing → Solvers, Theory of computation → Constraint and logic programming, Hardware → Theorem proving and SAT solving, Software and its engineering → Formal methods, Software and its engineering → Software verification, Hardware → Functional verification

Keywords and phrases Automated Decision Procedures, Constraint Programming, SAT, SMT

Digital Object Identifier 10.4230/DagRep.9.2.27

1 Executive Summary

Sébastien Bardin (CEA LIST, FR)
Nikolaj S. Bjørner (Microsoft Research – Redmond, US)
Cristian Cadar (Imperial College London, GB)
Vijay Ganesh (University of Waterloo, CA)

The scattered landscape of constraint solving. Constraint solving is at the heart of several key technologies, including program analysis, testing, formal methods, compilers, security analysis, optimization, and AI. During the last two decades, constraint solving has been

Bringing CP, SAT and SMT together: Next Challenges in Constraint Solving, *Dagstuhl Reports*, Vol. 9, Issue 2, pp. 27–47
Editors: Sébastien Bardin, Nikolaj S. Bjørner, and Cristian Cadar
Dagstuhl Reports
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

highly successful and transformative: on the one hand, SAT/SMT solvers have seen a significant performance improvement with a concomitant impact on software engineering, formal methods and security; on the other hand, CP solvers have also seen a dramatic performance improvement, with deep impact in AI and optimization.

These successes bring new applications together and new challenges: some fundamental constraints still lack efficient reasoning (e.g., floating-point arithmetic); quantifiers are rarely taken into account; current approaches focus essentially on satisfiability and/or validity while some applications would benefit from queries such as optimization or model counting. While each of the SAT, SMT and CP communities has made progress on some of these problems, no approach is able to tackle them all. Moreover, while historically strongly connected, the SAT/SMT communities have had minimal interactions with the CP community over the recent years.

Goals. The aim of this seminar was to reunify the Constraint Solving landscape and identify the next big challenges together with promising approaches. The seminar brought together researchers from SAT, SMT and CP along with applications researchers in order to foster cross-fertilization of ideas, deepen interactions, identify the best ways to serve the application fields and in turn help improve the solvers for specific usages.

An overview of constraint solving.

- **CP.** *Constraint Programming* [1] focuses on finding a solution (satisfiability) or a best solution (optimization) to constraint problems seen as sets of atomic constraints over arbitrary domains. Traditionally, CP is interested in problems defined over finite-domain variables (typically: bounded integers), yet a lot of work has also been devoted to infinite domains such as real numbers. The basic scheme of CP approaches (in the finite setting) consists in exploring the search tree of all partial valuations of the problem until a solution is found, or all possible valuations have been explored. At each step, *propagation* allows to refine further the admissible values for yet-unlabeled variables and, once no more propagation is possible, *labeling* assigns a value to a yet-unlabeled variable (yielding a backtrack point) and then propagation takes place against this, etc. CP has been highly successful in AI-related domains such as planning or scheduling, and promising applications to program verification have emerged recently.

  Strong points: advance propagation techniques based on the key notion of arc-consistency; specific reasoning, especially for finite-domain theories (e.g. floats, bounded arithmetic, bitvectors); queries beyond satisfiability, e.g. optimization

- **SAT.** While the seminal DPLL procedure [3] follows mostly the procedure described above for CP but specialized to the Boolean case[1], the true miracle of SAT comes from its modern version [2], where conflict-driven learning allows significant driven-by-need pruning of the search space—making the technique equally good at finding solutions or proving there is none. Many more improvements have been explored over the years, with carefully tuned propagation, data structures and branching heuristics. DPLL-style SAT solvers are at the core of hardware design and verification tools, and they have shown unreasonable efficiency on very large industrial problems.

  Strong points: conflict-driven clause learning methods; efficient search/propagate procedure, with optimized branching and look-ahead.

[1] Seeing CP as a generalization of SAT is also possible.

- **SMT.** *Satisfiability Modulo Theory* [4] extends SAT by considering the satisfiability problem over combinations of first-order theories, for examples formulas involving complex boolean structure plus uninterpreted functions, arrays and linear arithmetics. While first restricted to the unquantified case, the technique has been extended with partial support for quantifiers. The core of SMT techniques is the combination of efficient theory-dedicated conjunctive-only decision procedures (typically through the Nelson-Oppen combination framework) together with their lifting to the general (disjunctive) case thanks to the DPLL(T) framework, where a DPLL-style SAT solver works in interplay with theory solvers. SMT problems arise naturally in software analysis, where programs are built over combinations of basic data types. Hence, SMT solvers are naturally at the heart of most modern software verification technologies.

  Strong points: first-order decision procedures, including theories over infinite domains; elegant combinations of solvers; partial handling of quantifiers.

Research questions. The seminar allows to highlight several key challenges to current constraint solving techniques. They have been discussed during the meeting from different research perspectives.

- Hard-to-handle data types: several common data types and associated theories are still not managed in an efficient-enough way, typically finite-but-large domains such as modular arithmetic, bounded arithmetic with non-linear operations, floating-point arithmetic or bitvector constraints deeply mixing arithmetic and bit-level reasoning, sets with cardinality, strings with size, etc.
- Quantifiers: quantifiers can be added to SMT solvers but often at the price of losing model generation, while there is some support for finite quantification in SAT and CP but at the price of a significant drop in performance; yet, quantifiers are useful in practice (initial state, pre/post-conditions, summaries, etc.);
- Beyond satisfiability: while the first applications of constraint solving were concerned with finding solutions or proving validity / infeasibility, new applications bring new types of queries, such a optimization, soft constraints, solution counting, over-approximating sets of solutions, etc.
- New trade-offs between learning and propagation: while the SAT community seems to have reached a sweet spot on this question (with efforts put on a posteriori learning rather than on a priori propagation), the issue is not settled yet for SMT and CP, and may be theory and/or application dependent.

Potential synergies. We have also identified the following potential synergies between CP, SAT and SMT, and expect strong interactions around these points in a near future:

- CP researchers have advanced propagation techniques, domain-dedicated reasoning and (deep) constraint combination. SAT and SMT researchers can learn from that.
- SAT researchers have significantly advanced branching heuristics, look-ahead and conflict-clause learning methods. CP and SMT researchers can learn from that.
- SMT researchers have focused on theory solvers and well-defined solver combinations. How can we do "lightweight" theory integration in SAT/CP solvers that trade off generality for cheaper and focused implementation of theories aimed at very specific applications? SAT and CP researchers can take advantage of these points.
- How can we better serve the needs of applications researchers? Application researchers can tell solver designers about which of these features (and combinations thereof) they would like the most in a single solver.
- Finally, an important question is how do we leverage machine learning in these contexts. The experience of the SAT community may bring here some answers.

Outcome. The main goal of this Dagstuhl seminar was to bring together leading researchers in the different subfields of automated reasoning and constraint solving, foster greater communication between these communities and discuss new research directions.

The seminar had 28 participants from Australia, Austria, France, Germany, Finland, Italy, Spain, Sweden, Switzerland, United Kingdom and United States, from both academia, research laboratories and the industry. More importantly, the participants represented several different communities, with the topics of the talks and discussions reflecting these diverse interests in both solving technologies (CP, SAT, SMT), challenges (floating-point constraints, quantifiers, etc.) and application domains (testing, verification, security, compilation, commercialization, among others).

It was the first time such an *inclusive* meeting was held, bringing together leading researchers from SAT/SMT (typical interest: formal verification), CP (typical interest: optimization) and applications (typical interest: testing, verification, security). All participants agreed the event was fruitful, and we expect to see more collaborations between SAT/SMT and CP in a near future.

References

1 R. Dechter. *Constraint processing*. Elsevier Morgan Kaufmann, 2003.
2 L. Zhang, C. F. Madigan, M. H. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *International Conference on Computer-aided design*. IEEE Press, 2001.
3 M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 1962.
4 C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*. 2009.

2 Table of Contents

3 Tutorials

3.1 The State-of-the-Art in SAT Solving: Search, Simplify, Prove!

Armin Biere (Johannes Kepler Universität Linz, AT)

This tutorial covers algorithmic aspects of conflict-driven clause learning and important extensions developed in recent years, including decision heuristics, restart schemes, various pre- and inprocessing techniques as well as proof generation and checking. Programmatic incremental usage of SAT solvers is discussed next. The talk closes with an overview on the state-of-the-art in parallel SAT solving and open challenges in general.

The speaker contributed to the core technology of modern SAT solving, has developed 12 SAT solvers since 1999, which won 36 medals including 16 gold medals in international SAT competitions.

3.2 SMT – Basics and Recent Trends

Nikolaj S. Bjørner (Microsoft Research – Redmond, US)

The tutorial provides a refresher on Satisfiability Modulo Theories, SMT [1, 2]. SMT is used for a branch of automatic theorem proving that integrates satisfiability search with specialized procedures for theories of relevance. We describe the basic architecture of SMT solvers as a combination of propositional SAT search with integrated theory reasoning. The theories described in the tutorial includes arithmetic reasoning, floating point reasoning, strings, and domains found in CP solvers. The tutorial highlights recent advances in arithmetic reasoning, such as reducing non-linear integer arithmetic solving to linear integer arithmetic by introducing tangent lemmas and other properties of linear arithmetic formulas. The approach complements a technique based on reductions to cylindric algebraic decomposition. A number of other recent advances around reducing integer feasibility using strengthened linear real constraints, and centering search around policy iteration are also discussed. The second part of the tutorial describes uses of SMT solving. First from the view of which functionality is available in SMT solvers as a service to applications, second from the perspective of a set of timely applications of SMT solvers, including network verification, quantum compilation, smart contract verification, trusted financial software, DNN analysis, axiomatic economics, and uses of strings and regular expression reasoning.

References

1 C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In *Handbook of Satisfiability*. 2009.
2 Leonardo de Moura and Nikolaj Bjørner. Satisfiability Modulo Theories: Introduction and Applications. CACM, September 2011

3.3 The State of the Art in CP Solving: Infer, Relax, Search!

Pierre Flener (Uppsala University, SE)

I explain the declarative structure-based high-level modelling and the composition of specialised algorithms within the satisfaction and optimisation solvers of constraint programming (CP) technology, whether they work by systematic search or local search or a hybrid thereof, aided by a lot of inference and, to a lesser extent, by relaxation. If desired, the solving process can be parametrised by a user-provided search strategy, either by choice among predefined ones or programmatically. A lot of important extensions have been developed in recent years, including preprocessing, symmetry handling, autonomous search, hybridisation with SAT, etc.

3.4 Learning in Constraint Programming

Peter J. Stuckey (The University of Melbourne, AU)

Joint work of Peter J. Stuckey, Olga Ohrimenko, Michael Codish, Thibaut Feydy, Geoffrey Chu, Graeme Gange
Main reference Olga Ohrimenko, Peter J. Stuckey, Michael Codish: "Propagation via lazy clause generation", Constraints, Vol. 14(3), pp. 357–391, 2009.
URL https://doi.org/10.1007/s10601-008-9064-x

Nogood learning is a powerful mechanism for improving combinatorial search, by remembering what went wrong in the past, we can avoid making the same mistakes in the future. Nogood learning originated in the CP community, but had its first major impact in the SAT community where nogood learning SAT solvers are now universal. Modern CP solvers make use of the same learning mechanisms as in SAT, but there are many different features that arise in making them effective. In this talk I describe the nitty-gritty details about how CP solvers use nogood learning to improve performance. We discuss atomic constraints versus Booleans, integer variable (*theory*) propagators, lazy literal generation, structure based extended resolution, lifting explanations, and theory propagators in CP.

References

1 Thibaut Feydy, Peter J. Stuckey: Lazy Clause Generation Reengineered. CP 2009. Springer, 2009
2 Andreas Schutt, Thibaut Feydy, Peter J. Stuckey, Mark G. Wallace: Explaining the cumulative propagator. Constraints 16(3): 250-282 (2011)

4 Overview of Talks

4.1 Redundancy in Clausal Proofs and Satisfaction Driven Clause Learning

Armin Biere (Johannes Kepler Universität Linz, AT)

Joint work of Marijn J. H. Heule, Benjamin Kiesl, Armin Biere

We discuss recent notions of redundancy in clausal propositional proof system, including variants of blocked clauses, resolution asymmetric tautologies (RAT), as well as our new redundancy notion of propagation redundancy (PR). These concepts can be used to obtain short clausal proofs of hard combinatorial problems. We further proposed a new SAT solving paradigm, called satisfaction driven clause learning (SDCL), which can generate such proofs automatically.

References

1 Marijn J. H. Heule, Benjamin Kiesl, Armin Biere: Short Proofs Without New Variables. CADE 2017
2 Marijn J. H. Heule, Benjamin Kiesl, Martina Seidl, Armin Biere: PRuning Through Satisfaction. Haifa Verification Conference 2017

4.2 Bigly solving with Z3

Nikolaj S. Bjørner (Microsoft Research – Redmond, US)

The talk describes solving hard scheduling constraints using distributed SAT solving. Using the method of cube-and conquer, Z3 is distributed on hundreds of CPUs in Azure. We describe these solving methods that leverage cloud infrastructure.

4.3 COLIBRI: CP for FP (and BV)

François Bobot (CEA LIST, FR) and Sébastien Bardin (CEA LIST, FR)

Joint work of Sébastien Bardin, François Bobot, Zakaria Chihani, Bruno Marre

At first sight people misunderstand floating point numbers as reals. Then with more experiences their counter-intuitive behavior are patent. However sometimes they still behave like reals. We are going to see how a CP approach is able to prove these kind of assertions. We are going to see also how bitvectors are handled in a CP way.

We presented an efficient Constraint Programming approach to the SMTLIB theory of quantifier-free floating-point arithmetic (QF-FP). We rely on dense interreduction between many domain representations to greatly reduce the search space. We compare our tool to current state-of-the-art SMT solvers and show that it is consistently better on large problems involving non-linear arithmetic operations (for which bit-blasting techniques tend to scale

badly). We also briefly present results on the theory of bit-vectors (QF-BV) following the same high-level CP philosophy. These results emphasize the importance of the conservation of the high-level structure of the original problems, compared with standard bitblasting approaches.

References

1 Zakaria Chihani, Bruno Marre, François Bobot, Sébastien Bardin: Sharpening Constraint Programming Approaches for Bit-Vector Theory. CPAIOR 2017. Springer, 2017
2 François Bobot, Zakaria Chihani and Bruno Marre: Real Behavior of Floating Point. SMT Workshop 2017
3 Sébastien Bardin, Philippe Herrmann, Florian Perroud: An Alternative to SAT-based Approaches for Bit-Vectors. TACAS 2010. Springer, 2010

4.4 Replayable Symbolic Execution

Frank Busse (Imperial College London, GB)

Joint work of Frank Busse, Cristian Cadar

Symbolic execution is a dynamic program analysis technique that heavily relies on constraint solving. Constraint solving is computationally expensive, and many symbolic execution engines employ query caches to reduce solving time. Still, most of the execution time is spent solving constraints. Even worse, current engines are not able to reuse solver results and have to re-compute all results in every run and for every new version of a program under test. We present *Replayable Symbolic Execution*, a lightweight technique that persistently stores and re-uses solver results to significantly reduce the solving time in subsequent executions. Additionally, we give an overview of the distribution of solving times for individual queries in symbolic execution runs on real-world software. Symbolic execution often generates queries that are challenging for modern solvers, but the majority of queries is easily solvable. However, this majority accumulates to a substantial amount of solving time, and we argue that developers of SMT solvers should also consider this use case and optimise for short start-up phases.

4.5 Just Fuzz-it: An Unconventional Approach to SMT Solving

Cristian Cadar (Imperial College London, GB)

Joint work of Daniel Liew, Alaistair Donaldson, Cristian Cadar, J. Ryan Stinnett

In this ongoing work, we investigate the use of coverage-guided fuzzing as a means of proving satisfiability of SMT formulas over finite variable domains, with specific application to floating-point constraints. We show how an SMT formula can be encoded as a program containing a location that is reachable if and only if the program's input corresponds to a satisfying assignment to the formula. A coverage-guided fuzzer can then be used to search for such an assignment via a test input that covers the location. We have implemented this idea in a tool, Just Fuzz-it Solver (JFS), and we present a large experimental evaluation showing

that JFS is both competitive with and complementary to state-of-the-art SMT solvers with respect to solving floating-point constraints, and that the coverage-guided approach of JFS provides significant benefit over naive fuzzing in the floating-point domain. Applied in a portfolio manner, the JFS approach thus has the potential to complement traditional SMT solvers for program analysis tasks that involve reasoning about floating-point constraints.

A publication is coming up and will be posted at https://srg.doc.ic.ac.uk/publications/ JFS is publicly available at https://github.com/delcypher/jfs/

4.6 Greybox Fuzzing with Cost-Directed Input Prediction

Maria Christakis (MPI-SWS – Kaiserslautern, DE)

Joint work of Valentin Wüstholz, Maria Christakis
Main reference Valentin Wüstholz, Maria Christakis: "Learning Inputs in Greybox Fuzzing", CoRR, Vol. abs/1807.07875, 2018.
URL https://arxiv.org/abs/1807.07875

Greybox fuzzing is a lightweight testing approach that effectively detects bugs and security vulnerabilities. However, greybox fuzzers randomly mutate program inputs to exercise new paths; this makes it challenging to cover code that is guarded by narrow checks, which are satisfied by no more than a few input values.

In this work, we present a technique that extends greybox fuzzing with a method for predicting new inputs based on costs computed along already explored program executions. The new inputs are predicted such that they guide exploration toward optimal executions, which minimize a certain cost, for instance, the cost of covering a new path or revealing a vulnerability. We have evaluated our technique and compared it to standard greybox fuzzing on real-world benchmarks. In comparison, our technique detects significantly more bugs, often orders-of-magnitude faster.

4.7 Better Bit Blasting in Yices 2

Bruno Dutertre (SRI – Menlo Park, US)

Joint work of Dejan Jovanovic, Stéphane Graham-Lengrand, Jorge A. Navas

We present recent developments in solving bit-vector problems in Yices using the standard *bit-blasting* method, which amounts to converting a bit-vector SMT problem into SAT. This is work in progress. New techniques discussed include: better use of information available at the SMT level to guide preprocessing and variable elimination in the SAT solver, and the use of cut-sweeping as a preprocessing and in-processing technique.

4.8 SMT for Binary-Level Security Analysis

Benjamin Farinier (CEA LIST, FR)

Joint work of Sébastien Bardin, Richard Bonichon, Robin David, Matthieu Lemerre, Marie-Laure Potet, Benjamin Farinier

Program verification is an undeniable success of formal methods. Driven by progress in Satisfiability Modulo Theories, it led to the development of several tools for automatic bugs search. However, these decision procedures remain unsuitable when looking for vulnerabilities. Indeed, not all the bugs are vulnerabilities, and being able to distinguish them requires the resolution of formulas of appreciably larger size, but also belonging to more expressive logics which are poorly supported by current solvers. In this presentation, I will explain why the search for vulnerabilities leads to such formulas, then I will present two of our results on their resolution.

References

1 Benjamin Farinier, Sébastien Bardin, Richard Bonichon, Marie-Laure Potet: Model Generation for Quantified Formulas: A Taint-Based Approach. CAV 2018. Springer, 2018
2 Benjamin Farinier, Robin David, Sébastien Bardin, Matthieu Lemerre: Arrays Made Simpler: An Efficient, Scalable and Thorough Preprocessing. LPAR 2018. Springer, 2018

4.9 SMT-Based Exploit Generation: Past, Present and Future

Sean Heelan (University of Oxford, GB)

SMT solvers are at the core of the most popular approaches to exploit generation. In this talk I will first briefly outline the exploit generation problem and then explain how the existing state-of-the-art leverages SMT solvers to address it. Finally, I will give an overview of an entirely new approach to exploit generation that addresses some of the most significant limitations of existing systems, while still being tightly coupled with SMT solving technology.

4.10 Symbolic Pointers

Timotej Kapus (Imperial College London, GB)

Joint work of Timotej Kapus, Cristian Cadar

Symbolic execution is an effective technique for exploring paths in a program and reasoning about all possible values on those paths. However, the technique still struggles with code that uses complex heap data structures, in which a pointer is allowed to refer to more than one memory object. In this talk I present and discuss three ways symbolic execution can handle such cases:

1. Symbolic execution forks execution into multiple states, one for each object to which the pointer could refer, this can lead to major state explosion.

2. Instead of forking the whole symbolic execution, the constraints of each potential fork can be grouped together in a big disjunction, however SMT solver often struggle more with disjunctions.
3. All the memory can be grouped together into a single objetct thus avoiding the problem, but makes the constraints too large for real programs.

4.11 Symbol Elimination and Vampire

Laura Kovács (TU Wien, AT)

Joint work of Laura Kovacs, Andrei Voronkov, Simon Robillard, Evgeny Kotelnikov, Bernhard Gleiss

We overview the symbol elimination method for using first-order theorem proving in software analysis and verification. Symbol elimination exploits consequence finding in saturation-based theorem proving and generates logical consequences of an input set S of formulas such that these consequences are using only a subset of the input symbols from S. To make symbol elimination practical and scalable for program analysis, we use symbol elimination in the first-order theories of various data structures, imposing the challenge of reasoning with both theories and quantifiers. The talk will overview recent recent developments on symbol elimination and its use within our Vampire theorem prover, and report on our experiments applying symbol elimination for generating loop invariants and proving program loops correct.

References

1 Bernhard Gleiss, Laura Kovács, Simon Robillard: Loop Analysis by Quantification over Iterations. LPAR 2018
2 Laura Kovács, Andrei Voronkov: Finding Loop Invariants for Programs over Arrays Using a Theorem Prover. FASE 2009
3 Evgenii Kotelnikov, Laura Kovács, Andrei Voronkov: A FOOLish Encoding of the Next State Relations of Imperative Programs. IJCAR 2018

4.12 Floating-point Program Verification with CP Technology

Laurent Michel (University of Connecticut – Storrs, US)

Joint work of Heytem Zitoun, Claude Michel, Michel Rueher, Laurent Michel
Main reference Heytem Zitoun, Claude Michel, Michel Rueher, Laurent Michel: "Search Strategies for Floating Point Constraint Systems", in Proc. of the Principles and Practice of Constraint Programming – 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 – September 1, 2017, Proceedings, Lecture Notes in Computer Science, Vol. 10416, pp. 707–722, Springer, 2017.
URL https://doi.org/10.1007/978-3-319-66158-2_45

The ability to verify critical software is a key issue in embedded and cyber physical systems typical of automotive, aeronautics or aerospace industries. Bounded model checking and constraint programming approaches search for counter-examples that exhibit property violations. The search of such counter-examples is a long, tedious and costly task, especially for programs performing floating point computations. Existing search strategies are dedicated to finite domains and, to a lesser extent, to continuous domains. In this talk, we outline how CP can be used to this end and how critical novel search strategies are to floating point constraints. Empirical results help position this work with respect to state-of-the-art SAT and SMT solvers applied to the same task.

4.13 Experiences with a Little Combinatorial-optimization Startup

Robert Nieuwenhuis (UPC – Barcelona, ES)

Experiences with a little combinatorial-optimization startup. Some of the technical and practical challenges we encountered are discussed. We also describe three of the very different real-world problems we have attacked, and discuss desirable improvements in solver technology.

4.14 On Division Versus Saturation in Cutting Planes

Jakob Nordström (KTH Royal Institute of Technology – Stockholm, SE)

Joint work of Stephan Gocht, Amir Yehudayoff, Jakob Nordström

The conflict-driven clause learning (CDCL) paradigm has revolutionized SAT solving over the last two decades. Extending this approach to pseudo-Boolean (PB) solvers doing 0-1 linear programming holds the promise of further exponential improvements in theory, but intriguingly such gains have not materialized in practice. Also intriguingly, the most popular PB extensions of CDCL have not employed the standard cutting planes method with division, but have instead used the saturation rule saying that no variable coefficient needs to be larger than the maximum contribution that the inequality can require from this variable. To the best of our knowledge, there has been no study comparing the strengths of division and saturation in PB solving.

In this work, we show that cutting planes with division can be exponentially stronger than cutting planes with saturation, even when all linear combinations of inequalities are required to cancel variables (as in PB conflict analysis). In the other direction we do not obtain an exponential separation, but we show that the number of division steps needed to simulate a single saturation step can be exponential in the bitsize of the coefficients involved. We also perform some experiments on crafted benchmarks to see to what extent these theoretical phenomena can be observed in actual solvers. Our conclusions are that a careful combination of division and saturation seems to be crucial to harness more of the power of the cutting planes method in PB solvers.

References

1 Marc Vinyals, Jan Elffers, Jesús Giráldez-Cru, Stephan Gocht, Jakob Nordström: In Between Resolution and Cutting Planes: A Study of Proof Systems for Pseudo-Boolean SAT Solving. SAT 2018
2 Jan Elffers, Jesús Giráldez-Cru, Jakob Nordström, Marc Vinyals: Using Combinatorial Benchmarks to Probe the Reasoning Power of Pseudo-Boolean Solvers. SAT 2018

4.15 Continuous Constraint Solving

Marie Pelleau (Laboratoire I3S – Sophia Antipolis, FR)

Joint work of Marie Pelleau, Antoine Miné, Charlotte Truchet, Frédéric Benhamou
Main reference Marie Pelleau, Antoine Miné, Charlotte Truchet, Frédéric Benhamou: "A Constraint Solver Based on Abstract Domains", in Proc. of the Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings, Lecture Notes in Computer Science, Vol. 7737, pp. 434–454, Springer, 2013.
URL http://dx.doi.org/10.1007/978-3-642-35873-9_26

Constraint Programming generally deals with discrete variables. In this short talk I will summarize some of the techniques used to deal with numerical problems containing continuous variables.

4.16 Reusing Solutions Modulo Theories

Mauro Pezzè (University of Lugano, CH)

Joint work of Andrea Aquino, Giovanni Denaro, Mauro Pezzè
Main reference Andrea Aquino, Giovanni Denaro, Mauro Pezzè: "Heuristically matching solution spaces of arithmetic formulas to efficiently reuse solutions", in Proc. of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017, pp. 427–437, IEEE / ACM, 2017.
URL http://dx.doi.org/10.1109/ICSE.2017.46

This talk presents an approach for reusing formula solutions for both satisfiability and unsatisfiability proofs in order to reduce the impact of Satisfiability Modulo Theories (SMT) solvers on the scalability of symbolic program analysis.

SMT solvers can efficiently handle huge expressions in relevant logic theories, but they still represent a main bottleneck to the scalability of symbolic analyses, like symbolic execution and symbolic model checking. Reusing proofs of formulas solved during former analysis sessions can reduce the amount of invocations of SMT solvers, thus mitigating the impact of SMT solvers on symbolic program analysis. Yet, early approaches to reuse formula solutions exploit equivalence and inclusion relations among structurally similar formulas, and are strongly tighten to the specific target logics.

In this talk, I present an original approach that reuses both satisfiability and unsatisfiability proofs shared among many different formulas – beyond the standard cases of equivalent or related-by-implication formulas. The approach straightforwardly generalises across multiple logics. The technique is based on the original concept of distance between formulas, which heuristically approximates the likelihood of formulas to share either satisfiability or unsatisfiability proofs.

4.17 Interpolation in SMT

Tanja Schindler (Universität Freiburg, DE)

Joint work of Jochen Hoenicke, Tanja Schindler
Main reference Jochen Hoenicke, Tanja Schindler: "Efficient Interpolation for the Theory of Arrays", in Proc. of the Automated Reasoning – 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Lecture Notes in Computer Science, Vol. 10900, pp. 549–565, Springer, 2018.
URL https://doi.org/10.1007/978-3-319-94205-6_36

Craig interpolants are used to derive invariants in interpolation-based model checking. The interpolants can be generated from proofs of unsatisfiability provided by an SMT solver. In the talk we discuss some approaches to produce interpolants for different first-order theories, and highlight specific techniques implemented in our interpolating SMT solver SMTInterpol that address a couple of different difficulties in proof-based interpolation.

References

1 Jochen Hoenicke, Tanja Schindler: Efficient Interpolation for the Theory of Arrays. IJCAR 2018
2 Jochen Hoenicke, Tanja Schindler: Solving and Interpolating Constant Arrays Based on Weak Equivalences. VMCAI 2019

4.18 Machine Learning Clause DB Managment

Mate Soos (Hobbyist – Berlin, DE)

Joint work of Mate Soos, Raghav Kulkarni, Kuldeep Meel

In this talk, we present a machine-learning based system for learnt clause database management. The system has three main parts. The data gathering, the data crunching and machine learning model generation, and validation. The data we have collected is many GBs of relevant, never-before seen data about SAT solver behavior. We then crunch this data through sampling and data modeling to create a machine learning model that can be executed inside the SAT solver. Finally, we run the SAT solver with the learnt model inside. The results validate the approach but more interestingly, the side-effect of having so much valuable data is something that we didn't anticipate and may well be as important as the final results themselves.

4.19 Unison 101

Christian Schulte (KTH Royal Institute of Technology – Stockholm, SE)

Joint work of Christian Schulte, Mats Carlsson, Roberto Castañeda Lozano, Gabriel Hjort Blindell
URL https://unison-code.github.io/

This talk shows how Unison improves code generation in compilers by using constraint programming (CP) as a method for solving combinatorial optimization problems. It presents

how register allocation (assigning program variables to processor registers) and instruction scheduling (reordering processor instructions to increase throughput) can be modeled and solved using CP. Unison is significant as its addresses the same aspects as traditional code generation algorithms, yet is based on simple models and can robustly generate better code.

Unison is a collaboration between SICS, KTH, and Ericsson.

References

1 Roberto Castañeda Lozano, Mats Carlsson, Gabriel Hjort Blindell, Christian Schulte: Register allocation and instruction scheduling in Unison. CC 2016. Springer, 2016
2 Gabriel Hjort Blindell, Roberto Castañeda Lozano, Mats Carlsson, Christian Schulte: Modeling Universal Instruction Selection. CP 2015. Springer 2015
3 Roberto Castañeda Lozano, Mats Carlsson, Gabriel Hjort Blindell, Christian Schulte: Combinatorial Register Allocation and Instruction Scheduling. CoRR abs/1804.02452 (2018)

4.20 Solver Independent Rotating Workforce Scheduling

Peter J. Stuckey (The University of Melbourne, AU)

Joint work of Peter J. Stuckey, Andreas Schutt, Nysret Musliu
Main reference Nysret Musliu, Andreas Schutt, Peter J. Stuckey: "Solver Independent Rotating Workforce Scheduling", in Proc. of the Integration of Constraint Programming, Artificial Intelligence, and Operations Research – 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26-29, 2018, Proceedings, Lecture Notes in Computer Science, Vol. 10848, pp. 429–445, Springer, 2018.
URL https://doi.org/10.1007/978-3-319-93031-2_31

We give two solver independent models for the rotating work-force scheduling and compare them using different solving technology, both constraint programming and mixed integer programming. We show that the best of these models outperforms the state-of-the-art complete approaches for the rotating workforce scheduling problem, and that solver independent modelling allows us to use different solvers to achieve different aims: e.g. speed to solution, robustness of solving (particular for unsatisfiable problems) and how quickly we can generate good solutions (for optimization versions of the problem).

The lessons learned from this problem are interesting. An expert modeller constructed a model targeting CP solvers and another model targeting MIP solvers, but in practice the best model for CP solvers was the MIP one, and the best solver for MIP solvers was the CP one. This shows the importance of solver independent modelling where we dont commit to the solving technology we use during the modelling process.

4.21 Counterexample-Guided Quantifier Instantiation in Logical Theories

Cesare Tinelli (University of Iowa – Iowa City, US)

Joint work of Cesare Tinelli, Andrew Reynolds, Haniel Barbosa, Clark Barrett, Pascal Fontaine, Amit Goel, Dejan Jovanovic, Sava Krstič, Leonardo de Moura, Aina Niemetz, Andres Noetzli, Mathias Preiner

Main reference Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark Barrett, Cesare Tinelli: "Solving Quantified Bit-Vectors Using Invertibility Conditions", in Proc. of the Computer Aided Verification – 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II, Lecture Notes in Computer Science, Vol. 10982, pp. 236–255, Springer, 2018.

URL http://dx.doi.org/10.1007/978-3-319-96142-2_16

This talk provides an overview of a general approach to reason with quantified formulas in SMT. The approach maintains a set S of ground formulas incrementally expanded with selected instances of quantified input formulas, with the selection based on counter-models of S. In particular, in first-order theories that admit quantifier elimination and have a decidable universal fragment, this approach leads to practically efficient decision procedures for the full theory.

4.22 Creating a Program Schedule in EasyChair

Andrei Voronkov (University of Manchester, GB)

After giving a short overview of EasyChair and constraint satisfaction problems in it, the talk focuses on the problem of creating automatically a high-quality program schedule.

The talk formulates the problem, explains why it is hard and proposes it as a challenge to the SAT/CSP community.

5 Discussions

5.1 Food for Thought on CP and SAT/SMT

Pierre Flener (Uppsala University, SE)

I discuss existing bridges between the CP and SAT/SMT solving technologies, awareness issues, as well as differences and cross-fertilisation opportunities.

5.2 Numerical Challenges

Yannick Moy (AdaCore – Paris, FR)

SPARK is a subset of the Ada programming language targeted at formal verification. It comes with a formal verification tool called GNATprove based on the Why3 platform and relies on SMT solvers (Alt-Ergo, CVC4 and Z3) as the main engines of proof. Customers and users of SPARK face three main challenges regarding proof of numerical properties: non-linear arithmetic (in integers or bitvectors), floating-point arithmetic, combining theories. Many properties involving one of these are not provable by SMT solvers, and currently require proof in Coq of corresponding lemmas. For each of these, the challenge is both to prove true properties, and to generate counterexamples for false properties.

5.3 A SAAS Solver

Robert Nieuwenhuis (UPC – Barcelona, ES)

Shouldn't we use hundreds of machines (not cores), available at 0.01€ per machine · h, to offer a SAAS solver? Indeed, CDCL (underlying SAT, SMT, Pseudo-Boolean solving, lazy-clause-generation-based CP, etc.) is terribly sequential (perhaps even more than you think). So, why are we stubbornly trying to parallelize it with portfolios (sharing what?)? Alternative ideas could be to exploit community structure (perhaps computed by some short CDCL runs?) and parallel inprocessing. In this setting, are we married with clauses? Couldn't we handle any (more or less efficiently unit-propagatable?) representation like PB-constraints or even BDDs?

5.4 Inprocessing in SAT?

Mate Soos

This talk discusses how inprocessing has been used in modern SAT solvers. There have been a lot of papers about inprocessing, but they are rarely used. Why?

6 Programme

The seminar was mainly organized around short talks (15 min) in order to give the opportunity to each participant to present his work and to share his thoughts on the topic. The short duration was intended to keep a fast pace and good interactions. Besides, the first day featured several longer tutorials (SAT – 1h, CP – 1h, SMT – 30 min, learning in CP – 30 min) in order to bring the audience a minimal common background and to efficiently bootstrap

interactions. Finally, four discussion sessions (30 min) were devoted to open discussions on hot topics, the speakers briefly introducing the challenge and their views and then the organizers leading the discussion. This schedule is only indicative: many sessions took longer because of intense and fruitful discussions.

Programme for Monday 4th of February

- Organizers – *Overview to seminar, introduction of participants, proposals*
- Armin Biere – *(tutorial) The State-of-the-Art in SAT Solving: Search, Simplify, Prove!*
- Nikolaj Bjørner – *(tutorial) Overview of SMT Solving*
- Pierre Flener – *(tutorial) The State of the Art in CP Solving: Infer, Relax, Search!*
- Peter Stuckey – *(tutorial) Learning in Constraint Programming*
- Maria Christakis – *Greybox Fuzzing with Cost-Directed Input Prediction*
- Frank Busse – *Replayable Symbolic Execution*
- Timotej Kapus – *Symbolic Pointers*
- Yannick Moy – *(discussions) Numerical challenges*
- Robert Nieuwenhuis – *Experiences with a little combinatorial-optimization startup*

Programme for Tuesday 5th of February

- Laurent Michel – *Floating-point Program Verification with CP Technology*
- François Bobot – *Real Behavior of Floating Point numbers*
- Mate Soos – *(discussion) Inprocessing in SAT – what happened?*
- Peter Stuckey – *Solver Independent Modelling for Rotating Workforce Scheduling*
- Christian Schulte – *Unison 101: Generating Code with Constraint Programming*
- Cristian Cadar – *JFS: Constraint solving via fuzzing*
- Mauro Pezzè – *Reusing Solutions Modulo Theories*
- Robert Nieuwenhuis – *(discussion) A SAAS Solver*
- Jakob Nordström – *On Division Versus Saturation in Cutting Planes*
- Laura Kovács – *Symbol Elimination and Vampire*
- Armin Biere – *Redundancy in Clausal Proofs and Satisfaction Driven Clause Learning*
- Pierre Flener – *(discussion) Food for Thought on CP and SAT/SMT*
- Benjamin Farinier – *SMT solving for security*
- Cesare Tinelli – *Quantifier Instantiation Techniques in SMT*

Programme for Wednesday 6th of February

- Mate Soos – *Supervised Machine Learning for Clause Deletion Strategies*
- Marie Pelleau – *Continuous constraint solving*
- Tanja Schindler – *Interpolation in SMT*
- Andrei Voronkov – *Creating a program schedule in EasyChair*
- Sean Heelan – *SMT-Based Exploit Generation: Past, Present and Future*
- Nikolaj Bjørner – *Bigly solving with Z3*
- Bruno Dutertre – *Better Bitblasting in Yices*

Participants

- Sébastien Bardin
CEA LIST, FR
- Armin Biere
Johannes Kepler Universität
Linz, AT
- Nikolaj S. Bjorner
Microsoft Research –
Redmond, US
- François Bobot
CEA LIST – Nano-INNOV, FR
- Frank Busse
Imperial College London, GB
- Cristian Cadar
Imperial College London, GB
- Maria Christakis
MPI-SWS – Kaiserslautern, DE
- Bruno Dutertre
SRI – Menlo Park, US
- Benjamin Farinier
CEA LIST – Nano-INNOV, FR
- Pierre Flener
Uppsala University, SE
- Sean Heelan
University of Oxford, GB
- Matti Järvisalo
University of Helsinki, FI
- Timotej Kapus
Imperial College London, GB
- Laura Kovács
TU Wien, AT
- Laurent Michel
University of Connecticut –
Storrs, US
- Yannick Moy
AdaCore – Paris, FR
- Robert Nieuwenhuis
UPC – Barcelona, ES
- Jakob Nordström
KTH Royal Institute of
Technology – Stockholm, SE
- Marie Pelleau
Laboratoire I3S – Sophia
Antipolis, FR
- Mauro Pezzè
University of Lugano, CH
- Tanja Schindler
Universität Freiburg, DE
- Christian Schulte
KTH Royal Institute of
Technology – Stockholm, SE
- Laurent Simon
University of Bordeaux, FR
- Mate Soos
Hobbyist – Berlin, DE
- Peter J. Stuckey
The University of Melbourne, AU
- Cesare Tinelli
University of Iowa –
Iowa City, US
- Andrei Voronkov
University of Manchester, GB