

Certifying Algorithms for Automated Reasoning

Nikolaj S. Bjørner^{*1}, Marijn J. H. Heule^{*2}, Daniela Kaufmann^{*3},
Jakob Nordström^{*4}, and Wietze Koops^{†5}

1 Microsoft - Redmond, US. nbjorner@microsoft.com

2 Carnegie Mellon University - Pittsburgh, US. marijn@cmu.edu

3 TU Wien, AT. dk@danielakaufmann.at

4 University of Copenhagen, DK & Lund University, SE. jn@di.ku.dk

5 Lund University, SE & University of Copenhagen, DK. wietze.koops@cs.lth.se

Abstract

Modern automated reasoning has transformed large parts of industry and has also found numerous scientific applications. But many reasoning problems are computationally very challenging, or sometimes even undecidable. Because of this, the reasoning algorithms used are often very complex, and even the best current algorithms at times produce wrong results. As these tools are increasingly being used autonomously, sometimes even in life-critical applications, it is urgent to ensure that what they compute is valid. Software testing, while immensely useful, cannot guarantee correctness, and state-of-the-art algorithms are far beyond what techniques for producing formally verified software can handle.

The focus of this Dagstuhl Seminar was the approach of addressing such issues by designing certifying algorithms using so-called proof logging, meaning that algorithms output not only a result but also a machine-verifiable proof of correctness. This proof can then be fed to a dedicated proof checker for verification. Crucially, such proofs should require low overhead to generate and be easy to check, but still supply 100% correctness guarantees. Besides ensuring correctness of outputs for complex algorithms, proof logging can also provide new tools for algorithm development and analysis, software debugging, and even research into explainability in the context of AI.

Seminar June 1–6, 2025 – <http://www.dagstuhl.de/25231>

2012 ACM Subject Classification Mathematics of computing → Solvers; Theory of computation → Proof theory; Theory of computation → Automated reasoning; Software and its engineering → Software verification

Keywords and phrases ATP, Computer Algebra, DRAT, DRUP, MIP, Propagation Redundancy, QBF, SAT, SMT

Digital Object Identifier 10.4230/DagRep.15.6.1

* Editor / Organizer

† Editorial Assistant / Collector



Except where otherwise noted, content of this report is licensed under a Creative Commons BY 4.0 International license

Certifying Algorithms for Automated Reasoning, *Dagstuhl Reports*, Vol. 15, Issue 6, pp. 1–31

Editors: Nikolaj S. Bjørner, Marijn J. H. Heule, Daniela Kaufmann, Jakob Nordström, and Wietze Koops



DAGSTUHL
REPORTS

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


1 Executive Summary

Nikolaj S. Bjørner (Microsoft - Redmond, US)

Marijn J. H. Heule (Carnegie Mellon University - Pittsburgh, US)

Daniela Kaufmann (TU Wien, AT)

Jakob Nordström (University of Copenhagen, DK & Lund University, SE)

License  Creative Commons BY 4.0 International license

© Nikolaj S. Bjørner, Marijn J. H. Heule, Daniela Kaufmann, and Jakob Nordström

Automated reasoning has been widely adopted over the last decades for developing formally verified software and also in the context of combinatorial optimization. The foundation is built on automated deduction algorithms that are used for determining the satisfiability of propositional logic, first-order logic, and arithmetic formulas.

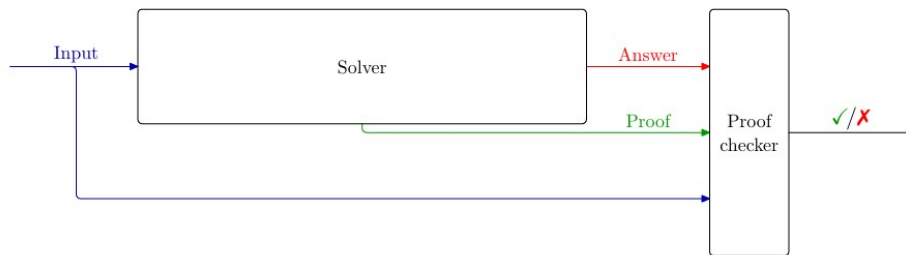
Algorithms for determining the validity of Boolean satisfiability (SAT), mixed integer programming (MIP), satisfiability modulo theories (SMT), and first-order automated theorem proving (ATP) formulas are integral components in verification tools. The question of how to certify correctness of the conclusions reached by such reasoning algorithms has received long-running attention, since in industrial formal verification they form the trusted base for correctness of safety-critical applications such as control systems for trains and airplanes.

Another application of automated reasoning is in combinatorial optimization, which studies problems where solutions are constructed by combining objects, but where the supply of objects is limited and there are constraints how they can be utilized together. Combinatorial optimization problems are encountered in a multitude of practical scenarios including logistics, scheduling, and disaster management. Here reasoning algorithms are employed not only to establish when a feasible solution is separated from non-feasible bounds, but they are also deeply integrated within combinatorial optimization solvers that can provide guaranteed optimal answers. The computational complexity of solving combinatorial optimization problems, which are typically harder than *NP*-complete, as well as the complexity of implementing sophisticated combinatorial solvers in practice, provide major challenges.

Algorithms used in symbolic solvers are often stunningly powerful in practice, and are today used routinely to solve large-scale real-world problems in a wide range of application areas. But the “dirty little secret” is that the solvers are sometimes wrong. It is well documented that even the best constraint programming (CP) and mixed integer programming (MIP) solvers sometimes return “solutions” that do not satisfy the constraints, or erroneously claim optimality, and that verification tools can erroneously claim a set of constraints is infeasible when, on the contrary, it has a solution. Also, in more complex scenarios, where solvers are used to solve subproblems, even seemingly innocuous off-by-one mistakes can snowball into huge overall errors.

Dealing with errors in software is, of course, not a new problem. The traditional method to discover and eliminate bugs is *software testing*. However, while substantial progress has been made recently on powerful so-called *fuzzing-based tools* applied to symbolic solvers, they cannot offer any guarantees that results produced by a solver are correct. It is an inherent limitation that testing can only reveal the presence of bugs, but never prove their absence.

Another very appealing approach is *formal verification*. This means that one writes down a formal, mathematical, specification of how the solver should work, and then provides a proof that the solver adheres to this specification. The main obstacle for this method is that the advanced techniques used in state-of-the-art solvers go far beyond what formal verification can currently handle. And even a fully verified solver cannot deal with the problem of incorrect results arising from hardware failures, which are unavoidable in large-scale computations.



■ **Figure 1** Schematic workflow for solver with proof logging.

Thus, the state of the art when it comes to verifiably correct automated reasoning is that this is a well-recognized problem that has remained without convincing solutions.

The Main Focus of This Seminar

This seminar focused on what currently appears to be the most promising way to eliminate errors in automated reasoning algorithms, namely *proof logging*. This means turning solvers into *certifying algorithms* in the sense of [1, 12] by having them output not only an answer but also a simple, machine-verifiable *proof* that this answer is correct. With such a solver, the workflow becomes as follows (see also Figure 1):

1. Run the solver on a problem to obtain an answer together with a proof.
2. Feed the problem, the answer, and the proof to a special computer program, called a *proof checker*.
3. Accept the answer if the proof checker says that the proof is valid.

For this to be feasible, the proof format needs to be sufficiently powerful, so that the solver can generate concise proofs even for sophisticated reasoning without incurring any serious overhead in running time. But the proof format should also be very simple, so that checking correctness becomes almost trivial – the point is that the proof checker, in contrast to the solver, should be so easy to code up that we can be confident that it is correct. Clearly, there is a conflict between expressivity and simplicity here. Perhaps asking for both at the same time is a little bit too good to be true? This tension between succinct proofs and easy verification goes to the heart of proof logging, and discussions of different ways of managing this trade-off was one of the key topics of the seminar.

One example of an approach that has so far been found to be unsatisfactory are methods in constraint programming using *explanations* [13, 15, 5], which essentially boils down to writing out reasons for solver conclusions trusting that these reasons are correct. The problem is that this means that proofs are so expressive that they cannot be efficiently verified by simple proof checking algorithms.

A much more successful approach is the *DRAT* proof system [9, 8, 16], which has become standard in SAT solving. This proof system is simple enough that the proof checker can even be implemented as a formally verified piece of software [4, 11], meaning that the full power of formal methods can be harnessed to guarantee correctness of the result produced by the solving algorithm. The crucial change of perspective making this possible is that the guarantee is not that the *algorithm* is correct, but that the *answer* found by the algorithm is.

And, in some sense, this even goes further than formally verified software in two important ways. Firstly, formally verified proof checking makes it possible to detect errors even if they are not due to faults in the solver but are caused by a buggy compiler, faulty hardware, or even cosmic rays during solver execution. Secondly, formally verified proof checking can allow us to fully trust the results even from buggy solvers. If the proof generated by a concrete algorithm execution checks out, then we can be fully confident that this particular computation reached the correct result, regardless of what bugs might be triggered for other inputs. It seems fair to argue that for reasons like this, SAT proof logging is perhaps the most successful showcase of certifying algorithms for computationally challenging problems to date, and for this reason it was natural to survey this area and discuss how similar advances could be made in other areas of automated reasoning.

However, when one tries to extend proof logging to stronger optimization paradigms such as *maximum satisfiability (MaxSAT)*, *pseudo-Boolean optimization*, *CP*, and *MIP*, or other areas of automated reasoning such as *automated planning* and *SMT solving*, the conflict between expressivity and simplicity reasserts itself. The clean and efficient reasoning in terms of disjunctive clauses used in *DRAT* seems poorly suited to capture reasoning about more complex objects like constraint programming propagators. Some of the other reasoning performed in more advanced solvers also seems hard to express in terms of the disjunctive clauses used in *DRAT*, and this limitation also makes it hard to argue about, e.g., values of objective functions in optimization problems. At a high-level, the reason that this is a highly nontrivial challenge is that the stronger the solving techniques used, the harder it becomes to design simple proof logging methods that can efficiently certify the correctness of these techniques. At the same time, the fact that CP, MIP, and SMT solvers are so fiendishly complex only makes the need for proof logging methods in these settings even more urgent.

In SMT solving, the most popular approach to date seems to be to design very expressive proof systems that can capture all the different theories considered and their combinations. One downside with this is that the proof systems become extremely complex, meaning that not only does the proof checking algorithm have to be very involved, but it is even a highly nontrivial task to even decide whether the proof system itself is consistent. Another direction, which has recently been pursued in the context of CP proof logging, is to compile all information about the input problem down to a simpler format in a trusted (or formally verified) way, and then mirror the reasoning performed in the solver by a proof in this simpler format. During the seminar, different subcommunities in automated reasoning were able to exchange experiences and best practices for these and other proof logging approaches for automated reasoning paradigms beyond SAT.

Further Topics Discussed During the Seminar

While the initial motivation for proof logging techniques is that they provide a way of ensuring the validity of outputs from complex algorithms, discussions at the seminar ranged over a number of topics that went beyond just providing formal **certificates of correctness** for answers produced by automated reasoning algorithms. Several participants of the seminar highlighted that proof logging can also be used as a tool for **debugging** during software development. Bugs that only very rarely affect the final result can be next to impossible to discover, but with proof logging switched on, it is easy to detect that the algorithm is performing unsound reasoning even when the output happens to be correct (as shown in, e.g., [6, 7, 10, 3]). It is also worth noting that it simplifies test case generation during debugging. There is no need to know what the correct output is, and instead testing can be done by checking the proof log.

Designing proof logging for a concrete algorithm typically involves describing in a formal proof system how the algorithm works, so that different reasoning steps can be written down as rule applications in the proof system. This type of analysis can also be quite helpful for **algorithm design** in that it can identify limitations in the current implementation of an algorithm and uncover potential for further improvements (if the proof system suggest that more powerful reasoning steps could be applied than what the algorithm actually does). Furthermore, since proof logging allows us to “peek inside” the algorithm, as it were, to get detailed information about what reasoning steps were performed, this provides a new tool for in-depth, scientifically rigorous, **performance analysis**.

Going further, it can be noted that proof logging by its very nature enables **auditability**, since once an algorithm execution has finished we can save the problem, answer, and proof for posterity so that it can be verified at any time by a third party, even if this third party has no access to the original algorithm used to solve the problem. Also, the fact that a proof for an optimization problem shows in a formal, mathematical, sense why a solution is correct, and/or why no better solution is possible, means that proof logging can serve as a stepping stone towards **explainability**, which is a topic of growing importance in the context of artificial intelligence (AI). These and other aspects were discussed in presentation by participants and also during a dedicated panel discussion towards the end of the seminar week.

The topics outlined above are well-represented in the research pursued by the researchers who participated in the seminar. In the SAT community, Armin Biere, Katalin Fazekas, Mathias Fleury, Marijn Heule, Adrián Rebola-Pardo and others have developed proof systems based on *DRAT* and extensions. Efficient and formally verified proof checkers for such proof systems have been built by Magnus Myreen and Yong Kiam Tan. Proof logging techniques for combinatorial optimization paradigms beyond SAT have been successfully pursued by Jeremias Berg, Bart Bogaerts, Wietze Koops, Ciaran McCreesh, Matthew McIlree, Jakob Nordström, Andy Oertel, and their collaborators. For mixed integer programming, Ambros Gleixner has done important exploratory work on proof logging for perhaps the most well-known open-source MIP solver *SCIP* together with collaborators. Certification of SMT solvers has received long-running attention, including work on *Z3* by Nikolaj Bjørner and *cvc5* by Haniel Barbosa, Bruno Dutertre, Hanna Lachnitt, and Andrew Reynolds together with collaborators. Many of these researchers gave presentations of their work where they also identified important future challenges.

In the context of automated theorem proving (ATP) for first-order logic, there are natural connections between formats for certifying deductions of ATP systems on the one hand and proof logging and model verification on the other hand, but also formidable technical (and organizational) obstacles to wider adoption of such techniques. During the seminar, Geoff Sutcliffe and Michael Rawson presented new results on ATP proof logging and checking. The participants also provided coverage of research on algebraic algorithms (Daniela Kaufmann), quantified Boolean formula solving (Martina Seidl), automated planning (Malte Helmert and Tanja Schindler), hardware verification (Dirk Beyer and Randal Bryant), hybrid systems (Erika Ábrahám), and several other topics related to automated reasoning.

Outcomes

The seminar aimed to advance the state of the art in the integration of proof logging with symbolic solvers, and to establish deeper contacts between different research communities working on certifying algorithms where interaction has previously been quite limited or non-

existent. The intention was to achieve this broad goal by assembling stakeholders in Boolean satisfiability (SAT) solving, constraint programming (CP), Mixed integer programming (MIP), satisfiability modulo theories (SMT) solving, automated theorem proving (ATP), and other closely related communities, including leading researchers in the areas of solver development, deployment of solver tools in applications, and design of proof logging techniques. Concretely, the seminar aimed to:

1. Connect automated-reasoning experts from the different domains around proof logging techniques.
2. Infuse the communities with new insights into the practical integration of proof logging and methods to develop formally verified proof checkers.
3. Facilitate technology transfer between different research areas in automated reasoning, in particular, concerning techniques for certifying correctness.

Going by the evaluations, the seminar was very successful in reaching these goals. It is our hope that this seminar will turn out to be only the first in a series of seminars dedicated to the important topic of certifying algorithms for automated reasoning. In the longer perspective, our vision is that such a series of seminars would contribute to a fundamental shift in how the computer science community thinks about algorithms, so that in the future algorithms will be expected to not just produce output but to prove that this output is in fact correct.

Seminar Structure

The scientific program of the seminar consisted of 30 presentations. Among these there were eleven 50-minute surveys of different core topics of the seminar. These talks occupied most of the morning schedule Monday-Wednesday, and were intended to make sure that the diverse audience would have a bit of a common background for the more technical talks reporting on recent progress and/or ongoing research. The list of survey talks and speakers were as follows:

- Certified SAT solving (Katalin Fazekas)
- Certified subgraph solving (Ciaran McCreesh)
- Certified constraint programming (Matthew McIlree)
- Proof logging for algebraic algorithms (Daniela Kaufmann)
- Proof logging for MIP (Ambros Gleixner)
- Certified automated planning (Malte Helmert)
- Certified SMT solving (Haniel Barbosa)
- Certified model counting and knowledge compilation (Randal Bryant)
- Certified QBF solving (Martina Seidl)
- Certified first-order theorem proving (Michael Rawson)
- Formally verified proof checking (Magnus O. Myreen & Yong Kiam Tan)

The rest of the talks were 25-minute presentations on recent research of the participants. The time after lunch each day was left for self-organized collaborations and discussions, and there was no schedule on Wednesday afternoon.

Based on polling of participants during the seminar, it was decided to have a panel discussion on Thursday afternoon. The poll also asked whether an open-problem session should be organized, but the support for this idea was weaker, and several participants emphasized that the seminar program should not be too dense and that the evenings should be left free of any program. Therefore, the organizers decided not to have an open-problem session.

References

- 1 Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, Christine Rizkallah, and Pascal Schweitzer. An introduction to certifying algorithms. *it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 53(6):287–293, December 2011.
- 2 Haniel Barbosa, Clark W. Barrett, Byron Cook, Bruno Dutertre, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Cesare Tinelli, and Yoni Zohar. Generating and exploiting automated reasoning proof certificates. *Commun. ACM*, 66(10):86–95, 2023.
- 3 Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, July 2023.
- 4 Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In *Proceedings of the 26th International Conference on Automated Deduction (CADE-26)*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, August 2017.
- 5 Nicholas Downing, Thibaut Feydy, and Peter J. Stuckey. Explaining alldifferent. In *Proceedings of the 35th Australasian Computer Science Conference (ACSC '12)*, pages 115–124, January 2012.
- 6 Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. In *Proceedings of the 22nd International Conference on Integer Programming and Combinatorial Optimization (IPCO '21)*, volume 12707 of *Lecture Notes in Computer Science*, pages 163–177. Springer, May 2021.
- 7 Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.
- 8 Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13)*, pages 181–188, October 2013.
- 9 Marijn J. H. Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In *Proceedings of the 24th International Conference on Automated Deduction (CADE-24)*, volume 7898 of *Lecture Notes in Computer Science*, pages 345–359. Springer, June 2013.
- 10 Sonja Kraiczy and Ciaran McCreesh. Solving graph homomorphism and subgraph isomorphism problems faster through clique neighbourhood constraints. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI '21)*, pages 1396–1402, August 2021.
- 11 Peter Lammich. Efficient verified (UN)SAT certificate checking. *Journal of Automated Reasoning*, 64(3):513–532, March 2020. Extended version of paper in *CADE 2017*.
- 12 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, May 2011.
- 13 Olga Ohrimenko, Peter J. Stuckey, and Michael Codish. Propagation via lazy clause generation. *Constraints*, 14(3):357–391, January 2009.
- 14 Alexander Steen, Geoff Sutcliffe, Pascal Fontaine, and Jack McKeown. Representation, verification, and visualization of Tarskian interpretations for typed first-order logic. In *LPAR 2023: Proceedings of 24th International Conference on Logic for Programming*,

Artificial Intelligence and Reasoning, volume 94 of *EPiC Series in Computing*, pages 369–385. EasyChair, June 2023.

- 15 Michael Veksler and Ofer Strichman. A proof-producing CSP solver. In *Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI '10)*, pages 204–209, July 2010.
- 16 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, July 2014.

2 Table of Contents

Executive Summary

Nikolaj S. Bjørner, Marijn J. H. Heule, Daniela Kaufmann, and Jakob Nordström 2

Overview of Talks

The certification problem for real algebra <i>Erika Ábrahám</i>	11
Symbolic Conflict Analysis in Pseudo-Boolean Solving <i>Albert Oliveras</i>	11
Faster Certified Symmetry Breaking in SAT <i>Markus Anders</i>	12
First Results on How to Certify Subsumptions Computed by the EL Reasoner Elk Using the Logical Framework with Side Conditions <i>Franz Baader</i>	12
SMT proof production, checking and reconstruction <i>Haniel Barbosa</i>	13
Certifying Software Verification <i>Dirk Beyer</i>	13
Certifying Hardware Model Checking <i>Armin Biere</i>	14
Certifying Pareto Optimality in Multi-Objective Maximum Satisfiability <i>Bart Bogaerts</i>	14
Checkable Proofs for Model Counting and Knowledge Compilation <i>Randal E. Bryant</i>	15
Certified SAT solving <i>Katalin Fazekas</i>	16
Consuming CaDiCaL Proofs <i>Mathias Fleury</i>	16
Proof logging and proof production for Mixed-Integer Programming <i>Ambros Gleixner</i>	16
Speculative SAT modulo SAT <i>Arie Gurfinkel</i>	17
Certified Automated Planning <i>Malte Helmert</i>	18
Graph Symmetries, Patterns, and Encodings <i>Mikoláš Janota</i>	18
Certifying Ideal Membership Tests <i>Daniela Kaufmann</i>	19
Practically Feasible Proof Logging for Pseudo-Boolean Optimization <i>Wietze Koops</i>	19
Proof Logging for Subgraph-Finding Algorithms <i>Ciaran McCreesh</i>	19

Certified Constraint Programming <i>Matthew McIlree</i>	20
Certifying Presolving/Preprocessing for 0-1 Integer Linear Programming and Max-SAT <i>Andy Oertel</i>	21
Certified First-Order Theorem Proving: confessions, excuses and a few ways out. <i>Michael Rawson</i>	21
Trimming SMT Proofs <i>Joseph Reeves</i>	22
Engineering Complete SMT Proofs in <i>cvc5</i> with Ethos/Eunoia <i>Andrew Reynolds</i>	22
Pseudo-Boolean Proof Logging for Optimal Planning <i>Tanja Schindler</i>	23
Certified QBF Solving <i>Martina Seidl</i>	23
Certifying Algorithms in Railway Verification <i>Monika Seisenberger</i>	23
Proof Verification with GDV and LambdaPi - It's a Matter of Trust <i>Geoff Sutcliffe</i>	24
Certifying Dynamic Symmetry Breaking for Graph Search in SAT and QBF <i>Stefan Szeider</i>	25
The Past, Present, and Future of Verified Proof Checkers <i>Yong Kiam Tan and Magnus Myreen</i>	25
Certification in SCL <i>Christoph Weidenbach</i>	26
Panel Discussion: The Future of Certifying Algorithms	
Opening Statements	26
Discussion	27
Closing Statements	28
Evaluation of the Seminar by Participants	29
Participants	31

3 Overview of Talks

3.1 The certification problem for real algebra

Erika Ábrahám (RWTH Aachen University, DE)

License © Creative Commons BY 4.0 International license

© Erika Ábrahám

Joint work of Erika Ábrahám, Jasper Nalbach, Valentin Promies

SMT solvers' traditional functionality is to check the satisfiability of quantifier-free formulas of first-order logic over different theories.

With their increasing efficiency and usage, this original functionality is being extended in different directions. One of them is the ability to provide some kind of assurance for the correctness of the computations, most prominently in the form of certificates.

Whereas some SMT solvers can already provide certificates for a wide range of theories, the theory of (quantifier-free non-linear) real algebra poses a hard challenge, and a solution seems to be yet completely out of reach.

In this talk, we discussed why this problem is especially hard, and which directions could be considered to make some progress.

3.2 Symbolic Conflict Analysis in Pseudo-Boolean Solving

Albert Oliveras (UPC Barcelona Tech, ES)

License © Creative Commons BY 4.0 International license

© Albert Oliveras

Joint work of Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, Rui Zhao

Main reference Robert Nieuwenhuis, Albert Oliveras, Enric Rodríguez-Carbonell, Rui Zhao: "Symbolic Conflict Analysis in Pseudo-Boolean Optimization", in Proc. of the 28th International Conference on Theory and Applications of Satisfiability Testing, SAT 2025, August 12-15, 2025, Glasgow, Scotland, LIPIcs, Vol. 341, pp. 23:1–23:18, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

URL <https://doi.org/10.4230/LIPICS.SAT.2025.23>

In the last two decades, a lot of effort has been devoted to the development of satisfiability-checking tools for a variety of SAT-related problems. However, most of these tools lack optimization capabilities. That is, instead of finding any solution, one is sometimes interested in a solution that is best according to some criterion.

Pseudo-Boolean solvers can be used to deal with optimization by successively solving a series of problems that contain an additional pseudo-Boolean constraint expressing that a better solution is required. A key point for the success of this simple approach is that lemmas that are learned for one problem can be reused for subsequent ones.

In this talk we go one step further and show how, by using a simple symbolic conflict analysis procedure, not only can lemmas be reused between problems but also strengthened, thus further pruning the search space traversal. In addition, we show how this technique automatically allows one to infer upper bounds in maximization problems, thus giving an estimation of how far the solver is from finding an optimal solution. Experimental results with our PB solver reveal that (i) this technique is indeed effective in practice, providing important speedups in problems where several solutions are found and (ii) on problems with very few solutions, where the impact of our technique is limited, its overhead is negligible.

3.3 Faster Certified Symmetry Breaking in SAT

Markus Anders (RPTU Kaiserslautern-Landau, DE)

License  Creative Commons BY 4.0 International license
© Markus Anders

Joint work of Markus Anders, Bart Bogaerts, Benjamin Bogø, Arthur Gontier, Wietze Koops, Ciaran McCreesh, Magnus Myreen, Jakob Nordström, Andy Oertel, Adrián Rebola-Pardo, Yong Kiam Tan


Symmetry breaking is a standard technique in many areas of automated reasoning. Recently, the possibility for proof logging symmetry breaking techniques in SAT solvers has become available by means of the dominance rule and VeriPB proof system [2]. It turns out however, that the proposed logging and checking techniques pose a severe bottleneck for efficient, modern symmetry handling algorithms [1]. In this talk, I gave a brief overview of symmetry handling algorithms and related proof logging techniques. Then, I discussed recent developments to improve logging and checking performance through the introduction of auxiliary variables. Lastly, I mentioned some of the remaining challenges.

References

- 1 Markus Anders, Sophie Brenner, and Gaurav Rattan. *Satsuma: Structure-Based Symmetry Breaking in SAT*. In Proc. of the 27th International Conference on Theory and Applications of Satisfiability Testing, SAT 2024, August 21-24, 2024, Pune, India, LIPIcs, Vol. 305, pp. 4:1–4:23, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.
<https://doi.org/10.4230/LIPICS.SAT.2024.4>
- 2 Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, Jakob Nordström. *Certified Dominance and Symmetry Breaking for Combinatorial Optimisation*. J. Artif. Intell. Res., Vol. 77, 2023.
<https://doi.org/10.1613/JAIR.1.14296>

3.4 First Results on How to Certify Subsumptions Computed by the EL Reasoner Elk Using the Logical Framework with Side Conditions

Franz Baader (TU Dresden, DE)

License  Creative Commons BY 4.0 International license
© Franz Baader

Joint work of Franz Baader, Patrick Koopmann, Cesare Tinelli
Main reference Franz Baader, Patrick Koopmann, Cesare Tinelli: “First Results on How to Certify Subsumptions Computed by the EL Reasoner ELK Using the Logical Framework with Side Conditions”, in Proc. of the 33rd International Workshop on Description Logics (DL 2020) co-located with the 17th International Conference on Principles of Knowledge Representation and Reasoning (KR 2020), Online Event [Rhodes, Greece], September 12th to 14th, 2020, CEUR Workshop Proceedings, Vol. 2663, CEUR-WS.org, 2020.

URL <https://ceur-ws.org/Vol-2663/paper-5.pdf>

The generation of proof certificates and the use of proof checkers is nowadays standard in first-order automated theorem proving and related areas. They have, to the best of our knowledge, not yet been employed in Description Logics, where the focus was on detecting and repairing errors in the ontology, rather than on catching erroneous consequences created by an incorrect reasoner. This paper reports on first steps towards remedying this deficit for subsumptions computed by the DL reasoner Elk. We use an existing tool for generating proofs of consequences from Elk, and transform these proofs into a format that is accepted as certificates by our proof checker. The checker is obtained as an instance of a generic certification tool based on the Logical Framework with Side Conditions (LFSC), by formalizing the inference rules of Elk in LFSC. We report on the results of applying this approach to the classification of a large number of real-world OWL 2 EL ontologies.

3.5 SMT proof production, checking and reconstruction

Haniel Barbosa (*Federal University of Minas Gerais-Belo Horizonte, BR*)

License © Creative Commons BY 4.0 International license
© Haniel Barbosa

Main reference Haniel Barbosa, Clark W. Barrett, Byron Cook, Bruno Dutertre, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Cesare Tinelli, Yoni Zohar: “Generating and Exploiting Automated Reasoning Proof Certificates”, *Commun. ACM*, Vol. 66(10), pp. 86–95, 2023.

URL <https://doi.org/10.1145/3587692>

Main reference Haniel Barbosa, Andrew Reynolds, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Arjun Viswanathan, Scott Viteri, Yoni Zohar, Cesare Tinelli, Clark Barrett: “Flexible proof production in an industrial-strength SMT solver,” in *Proc. of the 11th International Joint Conference on Automated Reasoning, IJCAR 2022, August 8-10, 2022, Haifa, Israel, LNCS*, Vol. 13385, pp. 15–35, Springer, 2022.

URL https://doi.org/10.1007/978-3-031-10769-6_3

Main reference Abdalrhman Mohamed, Tomaz Mascarenhas, Harun Khan, Haniel Barbosa, Andrew Reynolds, Yicheng Qian, Cesare Tinelli, Clark Barrett: “Lean-SMT: An SMT tactic for discharging proof goals in Lean”. *CoRR*, Vol. abs/2505.15796, 2025.

URL <https://doi.org/10.48550/ARXIV.2505.15796>

SMT solvers can be hard to trust, since it generally means assuming their large and complex codebases do not contain bugs leading to wrong results. Machine-checkable certificates, via proofs of the logical reasoning the solver has performed, address this issue by decoupling confidence in the results from the solver’s implementation. In this talk we will describe the extensive proof infrastructure of the state-of-the-art SMT solver *cvc5*, which has enabled the production of proofs in a number of complex domains. We will also show how these proofs are checked or reconstructed in different formats by different systems, from ad-hoc high-performance proof checkers to proof assistants such as Lean.

3.6 Certifying Software Verification

Dirk Beyer (*LMU München, DE*)

License © Creative Commons BY 4.0 International license
© Dirk Beyer

Joint work of Paulína Ayaziová, Dirk Beyer, Marian Lingsch-Rosenfeld, Martin Spiessl, Jan Strejček

Main reference Paulína Ayaziová, Dirk Beyer, Marian Lingsch-Rosenfeld, Martin Spiessl, Jan Strejček: “Software Verification Witnesses 2.0”, in *Proc. of the Model Checking Software - 30th International Symposium, SPIN 2024, Luxembourg City, Luxembourg, April 8-9, 2024, Proceedings, Lecture Notes in Computer Science*, Vol. 14624, pp. 184–203, Springer, 2024.

URL https://doi.org/10.1007/978-3-031-66149-5_11

Over the last years, certifying software verification has become an established practice in the area of automatic software verification: An independent validator re-establishes verification results of a software verifier using verification certificates (also called witnesses), which are stored in a standardized exchange format. In addition to validation, such exchangeable information about proofs and alarms found by a verifier can be shared across verification tools, and users can apply independent third-party tools to visualize and explore certificates to help them comprehend the causes of bugs or the reasons why a given program is correct. To achieve the goal of making verification results more accessible to engineers, it is necessary to consider certificates as first-class exchangeable objects, stored independently from the source code and checked independently from the verifier that produced them, respecting the important principle of separation of concerns. We present the conceptual principles of software-verification certificates and illustrate the contents of such certificates.

Material:

- Software Verification Witnesses 2.0 https://doi.org/10.1007/978-3-031-66149-5_11
- Verification Witnesses <https://doi.org/10.1145/3477579>

References

- 1 Paulína Ayaziová, Dirk Beyer, Marian Lingsch-Rosenfeld, Martin Spiessl, Jan Strejček. *Software verification witnesses 2.0*. In Proc. of the 30th International Symposium on Model Checking Software, SPIN 2024, April 8-9, 2024, Luxembourg City, Luxembourg, LNCS, Vol. 14624, pp. 184–203, Springer, 2024. https://doi.org/10.1007/978-3-031-66149-5_11
- 2 Dirk Beyer, Matthias Dangl, Daniel Dietsch, Matthias Heizmann, Thomas Lemberger, Michael Tautschnig. *Verification witnesses*. ACM Trans. Softw. Eng. Methodol, Vol. 31(4), pp. 57:1–57:69, 2022. <https://doi.org/10.1145/3477579>

3.7 Certifying Hardware Model Checking

Armin Biere (Universität Freiburg, DE)

License  Creative Commons BY 4.0 International license
 Armin Biere

Joint work of Nils Froyleyks, Emily Yu, Mathias Preiner, Armin Biere, Keijo Heljanko
Main reference Nils Froyleyks, Emily Yu, Mathias Preiner, Armin Biere, Keijo Heljanko: “Introducing Certificates to the Hardware Model Checking Competition”, in Proc. of the Computer Aided Verification: 37th International Conference, CAV 2025, Zagreb, Croatia, July 23-25, 2025, Proceedings, Part I, p. 281–295, Springer-Verlag, 2025.
URL https://doi.org/10.1007/978-3-031-98668-0_14

Design faults in hardware design are costly. Thus hardware model checking has routinely been applied during the chip design process for decades. However, both academic and industrial model checkers are complex software tools and arguably hard to get correct. To increase trust in model checkers we therefore propose a model checking certification flow. The model checker produces a witness circuit which simulates the original model and for safety properties has an inductive property implying the original property. Checking simulation and inductiveness can be done by SAT solving. We have applied this idea to different model checking techniques, particularly preprocessing techniques. The single safety property track of the hardware model checking competition in 2024 required all participants to produce such certificates. The competition showed that certification is possible and cheap, i.e., both with respect to certificate production and checking. Furthermore the winner of the competition surpasses the previous state-of-the-art, while producing machine checked witnesses. This is joint work with Emily Yu, Nils Froyleyks, Mathias Preiner and Keijo Heljanko.

3.8 Certifying Pareto Optimality in Multi-Objective Maximum Satisfiability

Bart Bogaerts (KU Leuven, BE)

License  Creative Commons BY 4.0 International license
 Bart Bogaerts

Joint work of Christoph Jabs, Bart Bogaerts, Jeremias Berg, Matti Järvisalo
Main reference Christoph Jabs, Jeremias Berg, Bart Bogaerts, Matti Järvisalo: “Certifying Pareto-Optimality in Multi Objective Maximum Satisfiability”, in Proc. of the Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON,

Canada, May 3-8, 2025, Proceedings, Part II, Lecture Notes in Computer Science, Vol. 15697, pp. 108–129, Springer, 2025.

URL https://doi.org/10.1007/978-3-031-90653-4_6

Due to the wide employment of automated reasoning in the analysis and construction of correct systems, the results reported by automated reasoning engines must be trustworthy. For Boolean satisfiability (SAT) solvers – and more recently SAT-based maximum satisfiability (MaxSAT) solvers – trustworthiness is obtained by integrating proof logging into solvers, making solvers capable of emitting machine-verifiable proofs to certify correctness of the reasoning steps performed. In this work, we enable for the first time proof logging based on the VeriPB proof format for multi-objective MaxSAT (MO-MaxSAT) optimization techniques. Although VeriPB does not offer direct support for multiobjective problems, we detail how preorders in VeriPB can be used to provide certificates for MO-MaxSAT algorithms computing a representative solution for each element in the non-dominated set of the search space under Pareto-optimality, without extending the VeriPB format or the proof checker. By implementing VeriPB proof logging into a state-of-the-art multi-objective MaxSAT solver, we show empirically that proof logging can be made scalable for MO-MaxSAT with reasonable overhead.

3.9 Checkable Proofs for Model Counting and Knowledge Compilation

Randal E. Bryant (Carnegie Mellon University - Pittsburgh, US)

License © Creative Commons BY 4.0 International license
© Randal E. Bryant

Joint work of Wojciech Nawrocki, Jeremy Avigad, Randal E. Bryant, Yong Kiam Tan, Marijn J. H. Heule
Main reference Randal E. Bryant, Yong Kiam Tan, Marijn J. H. Heule: “Certifying Projected Knowledge Compilation”, in Proc. of the 28th International Conference on Theory and Applications of Satisfiability Testing (SAT 2025), Leibniz International Proceedings in Informatics (LIPIcs), Vol. 341, pp. 8:1–8:22, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025.

URL <https://doi.org/10.4230/LIPIcs.SAT.2025.8>

Knowledge compilers convert Boolean formulas, given in conjunctive normal form (CNF), into representations that enable efficient evaluation of unweighted and weighted model counts, as well as a variety of other useful properties. Certifying the correctness of a knowledge compiler’s output, requires proving that 1) the generated formula is logically equivalent to the input formula, and 2) the generated formula satisfies the structural properties that enable efficient model counting.

Our Certified Partitioned-Operation Graph (CPOG) proof framework provides a way to encode the output of a knowledge compiler as well as a set of steps providing a checkable proof of correctness. Most recently, we have extended this framework to Skolem CPOG (SCPOG) supporting projected knowledge compilation, where a subset of the variables is abstracted away via existential quantification. Doing so requires a method to encode Skolem assignments, describing instantiations of the quantified variables.

We have developed formally verified checkers for both CPOG and SCPOG, one in Lean4 and the other in CakeML/HOL. In doing so, we formally verified the soundness of the frameworks.

3.10 Certified SAT solving

Katalin Fazekas (TU Wien, AT)

License  Creative Commons BY 4.0 International license
© Katalin Fazekas

Joint work of Katalin Fazekas, Florian Pollitt, Mathias Fleury, Armin Biere


Main reference Katalin Fazekas, Florian Pollitt, Mathias Fleury, Armin Biere: “Certifying Incremental SAT Solving”, in Proc. of the LPAR 2024: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning, Port Louis, Mauritius, May 26-31, 2024, EPIc Series in Computing, Vol. 100, pp. 321–340, EasyChair, 2024.

URL <https://doi.org/10.29007/PDCC>

This invited survey talk explores certified SAT solving, which is crucial for establishing trust in the reasoning steps and results of Boolean Satisfiability (SAT) solvers. We will cover the related fundamental concepts of SAT solving and discuss how proof-producing solvers and external checkers enable certification. A key focus will be on certifying incremental SAT solving, an essential technique that allows solvers to efficiently tackle sequences of related problems while maintaining correctness guarantees.

3.11 Consuming CaDiCaL Proofs

Mathias Fleury (Universität Freiburg, DE)

License  Creative Commons BY 4.0 International license
© Mathias Fleury

Joint work of Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froylyks, Florian Pollitt

Main reference Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froylyks, Florian Pollitt: “CaDiCaL 2.0”, in Proc. of the Computer Aided Verification - 36th International Conference, CAV 2024, Montreal, QC, Canada, July 24-27, 2024, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 14681, pp. 133–152, Springer, 2024.

URL https://doi.org/10.1007/978-3-031-65627-9_7

CaDiCaL offers a proof tracer interface that makes it possible to get information on the derivation worked. This interface hides only some of the internal information. In this talk, I describe the notification model and what information is produced.

3.12 Proof logging and proof production for Mixed-Integer Programming

Ambros Gleixner (HTW - Berlin, DE)

License  Creative Commons BY 4.0 International license
© Ambros Gleixner

Joint work of Ambros Gleixner, Leon Eifler, Alexander Hoen

Standard solvers for mixed-integer linear programming define feasibility and optimality of solutions within numerical tolerances and the correctness of their results, even within these tolerances, is subject to roundoff errors stemming from the unsafe use of floating-point arithmetic. By contrast, starting with version 10, the open-source MIP solver SCIP ships a numerically exact solving mode without tolerances and can produce an independently verifiable proof log for most of the exact solving techniques. Besides giving an overview on these recent advances and remaining limitations in software for verified MIP solving, we try to gauge to what extent floating-point MIP solvers can be used directly to produce verifiably correct proof logs. Our computational study with a pure LP-based branch-and-bound version

of SCIP confirms the expectation that in the overwhelming majority of cases, all critical decisions during the solving process are correct. When errors do occur on numerically challenging instances, they typically affect only a small, typically single-digit, amount of leaf nodes that would require further processing.

References

- 1 Leon Eifler and Ambros Gleixner. *Safe and verified Gomory mixed-integer cuts in a rational mixed-integer program framework*. SIAM Journal on Optimization, Vol. 34(1), pp. 742–763, 2024. <https://doi.org/10.1137/23M156046X>
- 2 Alexander Hoen and Ambros Gleixner. *Analyzing the numerical correctness of branch-and-bound decisions for mixed-integer programming*. In Proc. of the 22nd International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR 2025, November 10–13, 2025, Melbourne, Victoria, Australia, LNCS, Vol. 15763, pp. 35–50, Springer, 2025. https://doi.org/10.1007/978-3-031-95976-9_3

3.13 Speculative SAT modulo SAT

Arie Gurfinkel (University of Waterloo, CA)

License  Creative Commons BY 4.0 International license
© Arie Gurfinkel

Joint work of Arie Gurfinkel, Hari Govind VK, Isabel Garcia-Contreras, Sharon Shoham

Main reference Hari Govind V. K., Isabel Garcia-Contreras, Sharon Shoham, Arie Gurfinkel: “Speculative SAT Modulo SAT”, in Proc. of the Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 14570, pp. 43–60, Springer, 2024.

URL https://doi.org/10.1007/978-3-031-57246-3_4

State-of-the-art model-checking algorithms like IC3/PDR are based on unidirectional modular SAT solving for finding and/or blocking counterexamples. Modular SAT solvers divide a SAT-query into multiple sub-queries, each solved by a separate SAT solver (called a module), and propagate information (lemmas, proof obligations, blocked clauses, etc.) between modules. While modular solving is key to IC3/PDR, it is obviously not as effective as monolithic solving, especially when individual sub-queries are harder to solve than the combined query. This is partially addressed in SAT modulo SAT (SMS) by propagating unit literals back and forth between the modules and using information from one module to simplify the sub-query in another module as soon as possible (i.e., before the satisfiability of any sub-query is established). However, bi-directionality of SMS is limited because of the strict order between decisions and propagation – only one module is allowed to make decisions, until its sub-query is SAT. In this talk, I will describe our generalization of SMS, called SPEC SMS, that speculates decisions between modules. This makes it bi-directional – decisions are made in multiple modules, and learned clauses are exchanged in both directions. We further extend DRUP proofs and interpolation, these are useful in model checking, to SPEC SMS. We have implemented SPEC SMS in Z3 and show that it performs exponentially better on a series of benchmarks that are provably hard for SMS.

3.14 Certified Automated Planning


Malte Helmert (*Universität Basel, CH*)

License  Creative Commons BY 4.0 International license
© Malte Helmert

In my talk, I introduced the classical planning problem and explained its relevance to the seminar by contrasting it with SAT. For those that haven't seen planning before, the aim was to provide some basic understanding of the problem and why it is of interest. For those familiar with planning, I attempted to give additional perspectives on the problem and its complexity. I also gave the seminar participants a brief update on research in the planning community that tackles the main motivating questions of the seminar, in particular discussing results and open challenges for certifying planning algorithms.

3.15 Graph Symmetries, Patterns, and Encodings

Mikoláš Janota (*Czech Technical University - Prague, CZ*)

License  Creative Commons BY 4.0 International license
© Mikoláš Janota

Joint work of Mikoláš Janota, Michael Codish

Main reference Michael Codish, Mikoláš Janota: “Breaking Symmetries with Involutions”, in Proc. of the 31st International Conference on Principles and Practice of Constraint Programming, CP 2025, August 10-15, 2025, Glasgow, Scotland, LIPIcs, Vol. 340, pp. 8:1–8:17, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

URL <https://doi.org/10.4230/LIPICS.CP.2025.8>

When searching for graphs specifying certain conditions, the LexLeader approach is used to avoid symmetric graphs. A graph G is a LexLeader if its adjacency matrix is lexicographically smallest among all adjacent matrices representing graphs isomorphic to G . Symmetry breaking then amounts to adding constraints that eliminate non-LexLeader graphs from the search. We give a fresh perspective on symmetry breaking as a set cover problem and quantifier elimination problem [1]. A permutation π covers a graph G if G 's adjacency matrix becomes smaller under π . We further define the notion of a *pattern* [2], which describe a set of graphs that become smaller because of some permutation π at position i . To encode patterns as CNF, extra variables are needed. These are Tseitin variables that describe an equality between two edge variables. Notably, these variables are reused across multiple patterns. Like so, a set of patterns enables us elegantly expressing a set of non-lexleaders and therefore can be interpreted as a certificate for a symmetry breaking constraint.

References

- 1 Michael Codish and Mikoláš Janota. *Breaking symmetries from a set-covering perspective*. In Proc. of the 22nd International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, CPAIOR 2025, November 10–13, 2025, Melbourne, Victoria, Australia, LNCS, Vol. 15762, pp. 169–187, Springer, 2025.
- 2 Michael Codish and Mikoláš Janota. *Breaking symmetries with involutions*. In Proc. of the 31st International Conference on Principles and Practice of Constraint Programming, CP 2025, August 12-15, 2025, Glasgow, Scotland, LIPIcs, Vol. 340, pp. 8:1–8:17, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

3.16 Certifying Ideal Membership Tests

Daniela Kaufmann (TU Wien, AT)

License  Creative Commons BY 4.0 International license
© Daniela Kaufmann

Deciding ideal membership is a central problem in computer algebra, with wide-ranging applications in geometry, verification, and symbolic computation. While Gröbner bases provide a complete method for deciding ideal membership, their outputs are often complex, making certification difficult.

I present a framework for certifying ideal membership tests using a practical algebraic calculus (PAC) that allows tracking polynomial manipulations. The calculus supports different levels of granularity, allowing proofs to be either concise or detailed; depending on whether the emphasis is on debugging or efficient proof checking.

3.17 Practically Feasible Proof Logging for Pseudo-Boolean Optimization

Wietze Koops (Lund University, SE & University of Copenhagen, DK)

License  Creative Commons BY 4.0 International license
© Wietze Koops

Joint work of Wietze Koops, Daniel Le Berre, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Yong Kiam Tan, Marc Vinyals

Main reference Wietze Koops, Daniel Le Berre, Magnus O. Myreen, Jakob Nordström, Andy Oertel, Yong Kiam Tan, Marc Vinyals: “Practically Feasible Proof Logging for Pseudo-Boolean Optimization”, in Proc. of the 31st International Conference on Principles and Practice of Constraint Programming, CP 2025, August 10-15, 2025, Glasgow, Scotland, LIPICs, Vol. 340, pp. 21:1–21:27, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.


URL <https://doi.org/10.4230/LIPICs.CP.2025.21>

Certifying solvers have long been standard in Boolean satisfiability (SAT), allowing for proof logging and checking with limited overhead. However, developing similar tools for combinatorial optimization has remained a challenge. A recent promising approach covering a wide range of paradigms is pseudo-Boolean proof logging, but this has mostly consisted of proof-of-concept works far from delivering the performance required for real-world deployment.

In this work, we present an efficient toolchain based on VeriPB and CakePB for formally verified pseudo-Boolean optimization, and implement proof logging for the full range of techniques in the state-of-the-art solvers RoundingSat and Sat4j. Our experimental evaluation shows that proof logging and checking performance in this much more expressive paradigm is now quite close to the level of SAT solving, and hence clearly practically feasible.

3.18 Proof Logging for Subgraph-Finding Algorithms

Ciaran McCreesh (University of Glasgow, GB)

License  Creative Commons BY 4.0 International license
© Ciaran McCreesh

Many interesting problems involve finding a little graph inside a bigger graph: for example, maximum clique asks for the largest set of vertices where everything is adjacent, whilst subgraph isomorphism asks whether a specific pattern occurs inside a given target graph.

Although these problems are computationally hard in theory, in practice solvers can often handle these problems extremely quickly, even on graphs with thousands of vertices. However, these solvers are not always perfect, and sometimes contain bugs that lead to wrong answers being produced. I'll explain how, using the VeriPB proof system, we can augment these solvers to produce correctness certificates, allowing us to be confident they have definitely given the right answers. To do this, we'll need to be able to justify a wide range of algorithmic inference steps, including colour bounds, all-different filtering, and degree reasoning; perhaps surprisingly, VeriPB is able to do all of these efficiently, despite not having any notion of what a graph is.

3.19 Certified Constraint Programming

Matthew McIlree (University of Glasgow, GB)

License © Creative Commons BY 4.0 International license
© Matthew McIlree

Joint work of Matthew McIlree, Ciaran McCreesh, Stephan Gocht, Jakob Nordström, Jan Elffers

Main reference Stephan Gocht, Ciaran McCreesh, Jakob Nordström: “An Auditable Constraint Programming Solver”, in Proc. of the 28th International Conference on Principles and Practice of Constraint Programming, CP 2022, July 31 to August 8, 2022, Haifa, Israel, LIPIcs, Vol. 235, pp. 25:1–25:18, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

URL <https://doi.org/10.4230/LIPICS.CP.2022.25>

Constraint programming (CP) is a powerful paradigm for expressing and solving satisfaction and optimisation problems involving finite domain variables and high-level constraints. But the implementation and engineering of CP algorithms can be extremely complex, error-prone, and difficult to test. We are much more likely to trust the output of a solver if it can provide some kind of certificate of correctness via proof logging.

In this talk, I will discuss the current state of research into adding proof logging to CP solvers. I'll cover how we can prove unsatisfiability and optimality; what makes this different from established proof logging technology for SAT solvers; and the efforts towards devising efficient justification procedures for the huge variety of propagation algorithms available in the modern CP repertoire.

3.20 Certifying Presolving/Preprocessing for 0-1 Integer Linear Programming and MaxSAT

Andy Oertel (Lund University, SE)

License © Creative Commons BY 4.0 International license
© Andy Oertel

Joint work of Jeremias Berg, Ambros Gleixner, Alexander Hoen, Hannes Ihalainen, Matti Järvisalo, Magnus Myreen, Jakob Nordström, Andy Oertel, Yong Kiam Tan

Main reference Alexander Hoen, Andy Oertel, Ambros M. Gleixner, Jakob Nordström: “Certifying MIP-Based Presolve Reductions for 0-1 Integer Linear Programs”, in Proc. of the Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 14742, pp. 310–328, Springer, 2024.

URL https://doi.org/10.1007/978-3-031-60597-0_20

Main reference Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, Jakob Nordström: “Certified MaxSAT Preprocessing”, in Proc. of the Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I, Lecture Notes in Computer Science, Vol. 14739, pp. 396–418, Springer, 2024.

URL https://doi.org/10.1007/978-3-031-63498-7_24

It is well known that reformulating the original problem can be crucial for the performance of mixed-integer programming (MIP) and maximum satisfiability (MaxSAT) solvers. While the idea is the same in both the MIP and MaxSAT community, the presolving reductions in MIP and preprocessing in MaxSAT apply slightly different techniques to reformulate the problem. To ensure the correctness of the reformulations, all transformations must preserve the feasibility and optimal value of the problem, but there is currently no established methodology to express and verify the equivalence of two optimization problems.

In this talk, it is presented how pseudo-Boolean proof logging can be used to certify the correctness of a wide range of modern MIP presolving and MaxSAT preprocessing techniques. By combining and extending the VeriPB and CakePB tools, we obtain a formally verified end-to-end proof checking tool chain to verify the correctness of reformulations of pseudo-Boolean problems.

This talk is based on the following two papers. The first paper was published at CPAIOR 2024 together with Alexander Hoen, Ambros Gleixner, and Jakob Nordström. The second paper was published at IJCAR 2024 together with Hannes Ihalainen, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström.

3.21 Certified First-Order Theorem Proving: confessions, excuses and a few ways out.

Michael Rawson (University of Southampton, GB)

License © Creative Commons BY 4.0 International license
© Michael Rawson

Joint work of Michael Rawson, Anja Petković Komel, Martin Suda

Automated Theorem Provers (ATPs) have been around a long time, but their proof certification ecosystem is nowhere near as well-developed as, say, SAT solvers. There are several reasons for this: a plurality of proof calculi, equisatisfiable inferences, and theories, to name a few. I will outline ATP systems and their proof certification, explain some difficult areas, present some recent developments, and offer a few paths to salvation.

3.22 Trimming SMT Proofs

Joseph Reeves (Carnegie Mellon University - Pittsburgh, US)

License  Creative Commons BY 4.0 International license
 © Joseph Reeves

Joint work of Joseph Reeves, Haniel Barbosa, Marijn J. H. Heule, Andrew Reynolds

Automated reasoning tools require high trust, prompting modern solvers to produce proof certificates for verification. In propositional satisfiability (SAT), proof generation and checking are relatively inexpensive, but in satisfiability modulo theories (SMT), justifying theory lemmas can introduce significant overhead. Recent approaches mitigate this issue by having the SMT solver produce a proof skeleton containing only the propositional reasoning in the DRAT format and unjustified theory lemmas, whose justifications are deferred to the checking phase. Preprocessing, a key element of SMT solving that can be challenging to justify a posteriori, is not justified nor checked. We extend these approaches by including proofs for preprocessing; by reducing the checker workload via iteratively eliminating theory lemmas from proof skeletons through SAT solving and proof trimming; and by proposing two justification methods for theory lemmas: one batches justifications for parallelization, while the other not only checks the theory lemma justifications but also integrates them into a fully detailed proof that could be checked with standard approaches. Experimental results on SMT-LIB benchmarks show the benefits of our approach in reducing solving time when producing proof skeletons that can be effectively checked externally. In particular, the extended trimming techniques can significantly reduce the number of theory lemmas to be checked beyond standard trimming, thereby improving sequential and parallel checking times.

3.23 Engineering Complete SMT Proofs in *cvc5* with Ethos/Eunoia

Andrew Reynolds (University of Iowa - Iowa City, US)

License  Creative Commons BY 4.0 International license
 © Andrew Reynolds

Over the past 5 years, the SMT solver *cvc5* has been instrumented to produce proofs for a majority of its theories. This talk reports on a new milestone for this work, namely that all mainstream features of *cvc5* are 100% proof producing and checkable in an external proof checker (Ethos). In detail, Ethos is a high performance proof checker written in around 10k lines of C++. Its native input language is Eunoia, a logical framework for defining proof systems that is heavily inspired by the forthcoming SMT-LIB version 3.0 language. To engineer complete proofs for *cvc5*, I will discuss the introduction of a “safe mode” of *cvc5*, which defines a subset of the features of *cvc5* that are free of known bugs and have complete proof support. The internal proof calculus of *cvc5*, now known as the Cooperating Proof Calculus (CPC), has been formalized in around 6500 lines of Eunoia definitions. Notably, this formalization now covers all mainstream theories of *cvc5*, including those currently used by industrial users of *cvc5*.

3.24 Pseudo-Boolean Proof Logging for Optimal Planning

Tanja Schindler (Universität Basel, CH)

License © Creative Commons BY 4.0 International license
© Tanja Schindler

Joint work of Simon Dold, Malte Helmert, Jakob Nordström, Gabriele Röger, Tanja Schindler
Main reference Simon Dold, Malte Helmert, Jakob Nordström, Gabriele Röger, Tanja Schindler: “Pseudo-Boolean Proof Logging for Optimal Classical Planning”, in Proc. of the 35th International Conference on Automated Planning and Scheduling, ICAPS 2025, November 9-14, 2025, Melbourne, Victoria, Australia, Proc. ICAPS, Vol. 35(1), pp. 54–63, AAAI Press, 2025.

URL <https://doi.org/10.1609/ICAPS.V35I1.36101>

Optimal classical planning is the problem to find an action sequence with minimal cost from a given initial state to a goal state. Checking that a given action sequence is a plan can easily be done by applying the actions one after another, but if a planning system claims that a plan it has found is optimal or that there is no plan, a different kind of proof is needed. In my talk, I present our recent efforts to provide such a proof in the form of lower-bound certificates based on pseudo-Boolean constraints. These certificates can then be checked by VeriPB. I demonstrate how planning systems based on heuristic search can be modified to produce such certificates, and discuss the current status of the approach.

3.25 Certified QBF Solving

Martina Seidl (Johannes Kepler Universität Linz, AT)

License © Creative Commons BY 4.0 International license
© Martina Seidl

Over the last years, much progress has been made in theory and practice of solving quantified Boolean formulas (QBFs). In principle, it is also well understood how to certify solving results found on solvers based on different solving paradigms like QCDCL as well as abstraction- and expansion-based solving. QBF certification is strongly inspired by approaches successfully used in SAT. Nevertheless, state-of-the-art solvers support certification to a limited extent only.

In this talk, an overview is given on the state of the art of certified QBF solving and the challenges that need to be addressed to obtain a fully operational certification workflow.

3.26 Certifying Algorithms in Railway Verification

Monika Seisenberger (Swansea University, GB)

License © Creative Commons BY 4.0 International license
© Monika Seisenberger

Joint work of Harry Bryant, Alec Critten, Andrew Lawrence, Monika Seisenberger, Anton Setzer

We report on one old and some recent advances we made in the context of applying SAT/SMT solving in the area of Railway Verification.

The first concerns a provably correct DPLL/Resolution solver [1] that was extracted from a formal proof (in the theorem prover Minlog) that for every clause set there is either a satisfying assignment of variables or a proof of unsatisfiability. The extracted program from this (Minlog) proof yields a program (in Haskell) that either computes a model or a (DPLL system) proof of unsatisfiability. The extracted solver was applied to a number of Railway problems.

Our new work concerns a certified RUP checker that has been extracted from a formal proof in the Theorem Provers Rocq and Agda [2]. We formalised the RUPchecker in Rocq, provided a soundness proof and extracted the checker from it. The procedure also allows to produce the corresponding Unitresolution proof, but to do so is not required for the correctness of the checked result.

This is joint work with Harry Bryant, Alec Critten, Andrew Lawrence (Siemens Mobility), and Anton Setzer.

References

- 1 Ulrich Berger, Andrew Lawrence, Fredrik Nordvall Forsberg, and Monika Seisenberger. *Extracting verified decision procedures: DPLL and Resolution*. Logical Methods in Computer Science, Vol. 11(1), 2015. [https://doi.org/10.2168/LMCS-11\(1:6\)2015](https://doi.org/10.2168/LMCS-11(1:6)2015)
- 2 Harry Bryant, Andrew Lawrence, Monika Seisenberger, Anton Setzer. *Verifying Z3 RUP proofs with the interactive theorem provers Coq/Rocq and Agda*. In Proc. of the 31st International Conference on Types for Proofs and Programs, TYPES 2025, June 9-13, 2025, Glasgow, Scotland, Abstracts, 2025.

3.27 Proof Verification with GDV and LambdaPi - It's a Matter of Trust

Geoff Sutcliffe (*University of Miami, US*)

License © Creative Commons BY 4.0 International license
© Geoff Sutcliffe

Joint work of Geoff Sutcliffe, Frédéric Blanqui, Guillaume Burel
Main reference Geoff Sutcliffe, Frédéric Blanqui, Guillaume Burel: “Proof Verification with GDV and LambdaPi - It's a Matter of Trust”, in Proc. of the 38th International Florida Artificial Intelligence Research Society Conference, FLAIRS 2025, Daytona Beach, FL, USA, May 20-23, 2025, Florida Online Journals, 2025.

URL <https://doi.org/10.32473/FLAIRS.38.1.138642>

Automated Theorem Proving (ATP) is concerned with the development and use of software that automates sound reasoning. An ATP system can be required to output a proof that serves as a certificate for the system's claim. To ensure that a proof is correct, verification can be required. If the verifier outputs evidence in a form that can be independently checked, that evidence serves as a certificate for the verifier's claim. The sequence of finding a proof, verifying the proof, and certifying the verification, builds an increasing level of trust in the system. This talk traces one such path for TPTP format proofs generated by ATP systems, via the GDV derivation verifier, and ending at the LambdaPi checker.

References

- 1 Geoff Sutcliffe, Frédéric Blanqui, Guillaume Burel. *Proof Verification with GDV and LambdaPi - It's a Matter of Trust*. In Proc. of the 38th International FLAIRS Conference, FLAIRS-38, May 20-23, 2025, Daytona Beach, Florida, USA, Proc. FLAIRS, Vol. 38, Florida Online Journals, 2025.

3.28 Certifying Dynamic Symmetry Breaking for Graph Search in SAT and QBF

Stefan Szeider (TU Wien, AT)

License © Creative Commons BY 4.0 International license
© Stefan Szeider

Joint work of Mikoláš Janota, Markus Kirchweger, Tomás Peitl, Stefan Szeider

Main reference Mikoláš Janota, Markus Kirchweger, Tomás Peitl, Stefan Szeider: “Breaking Symmetries in Quantified Graph Search: A Comparative Study”, in Proc. of the AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA, pp. 11246–11254, AAAI Press, 2025.

URL <https://doi.org/10.1609/AAAI.V39I11.33223>

SAT Modulo Symmetries (SMS) performs dynamic symmetry breaking for graph generation by detecting and excluding non-canonical graphs during CDCL search. In this talk, we will focus on the certification mechanisms that ensure the correctness and completeness of this approach across different settings.

We will consider three types of certificates: (1) nc-certificates (non-canonicity certificates), which are permutations proving that a partially defined graph cannot be extended to a lexicographically minimal solution; (2) DRAT proofs where the symmetry-breaking clauses learned via the minimality check are added as axioms to the proof, with these axioms themselves certified by their corresponding nc-certificates; and (3) uniform proofs for the QBF setting, where SMS handles quantified graph search problems with formal verification through LDQ-resolution.

These certification mechanisms provide mathematical guarantees for the correctness of SMS and enable independent verification of results in challenging combinatorial problems, from confirming the Murty-Simon conjecture to computing Ramsey graphs with formal proofs.

3.29 The Past, Present, and Future of Verified Proof Checkers

*Yong Kiam Tan (Institute for Infocomm Research (I2R), A*STAR, Singapore, SG & Nanyang Technological University, Singapore, SG) and Magnus Myreen (Chalmers University of Technology - Göteborg, SE)*

License © Creative Commons BY 4.0 International license
© Yong Kiam Tan and Magnus Myreen

URL <https://cakeml.org/checkers.html>


This survey talk will be split into two parts.

First, we will survey various automated reasoning theories/domains where verified proof checkers have been built. We will also present some of our ongoing work, and we will argue that verification can help enable the design of more complex proof systems/checkers while preserving trust in the overall certification process.

Then, we will take a deeper dive into verification infrastructure available in various theorem provers. Special focus will be given to *HOL4* and the *CakeML* project – we have used *CakeML* to build several end-to-end verified proof checkers with machine-code level correctness guarantees. We will discuss synergies between what *CakeML* can bring to proof checking and what proof checking can bring to *CakeML*.

3.30 Certification in SCL

Christoph Weidenbach (MPI für Informatik - Saarbrücken, DE)

License  Creative Commons BY 4.0 International license
© Christoph Weidenbach

SCL is a new paradigm for automatic reasoning in various logics. I discuss certification of proofs in SCL.

4 Panel Discussion: The Future of Certifying Algorithms

On Thursday afternoon, a 1-hour panel discussion with the title *The Future of Certifying Algorithms* was organized. The panelists were Marijn Heule, Haniel Barbosa, Ciaran McCreesh, and Ambros Gleixner, and the session was moderated by Jakob Nordström.

The panel started with an opening statement by each panelist, where they could choose to either address a (suitably provocative) question provided by the panel moderator, or to talk about some other topic of their own choice. After that, the audience was invited to ask questions. Below follow summaries of the opening statements and some of the ensuing discussion.

4.1 Opening Statements

Marijn Heule. *Question:* “Is the point of proof logging to create terabyte-size or petabyte-size proofs that nobody can understand? Is this the best way in which combinatorial solving can help mathematics, or can we go beyond this?”

Response: Proof logging for SAT has been successful despite the size of the proofs. Finding a proof is hard, but making the proofs more compact using trimming and checking such smaller proofs is easier. So trimming is key to the success of SAT proof logging. In this seminar there have not been enough talks about trimming. Proof trimming techniques should be higher on the agenda for proof logging efforts that go beyond SAT.

Another question is when we can produce small certificates. For example, for some maximum clique-finding problems, one can certify optimality just by giving a coloring. So there should be more work on finding small proofs.

Haniel Barbosa. *Question:* “Why would SMT proof logging have to be so slow? SAT solvers have blisteringly fast proof logging despite learning upwards of 50,000 lemmas per second – are we saying that SMT solvers are doing substantially more reasoning per time unit?”

Response: The main issue is that SMT solvers are doing 30 procedures that are each as hard as SAT solving. So far, the SMT community has focused on producing proofs that can be checked, and ensuring that each component of the solver can output justifications. The proof checking overhead is higher than it should be, but there is just too much to do with all the different components of the SMT solver.

Ciaran McCreesh. *Question:* “Designing proof logging for more and more applications is all well and fine, but what is the point if the *VeriPB* proof system is so complicated that nobody can use it? Should proof logging beyond SAT require a PhD in computational complexity theory?”

Response: It should not be socially acceptable to claim to have a faster solver without implementing proof logging. To do proof logging, people do need to have some understanding, but usually it is not necessary to understand all of *VeriPB*. But it does not have to be hard: proof logging for clique (which does not use the so-called redundance-based strengthening rule) was done by a master student. It is not that much extra that we are asking.

Ambros Gleixner. *Question:* “What could possibly make the operations research (OR) community care about certifying solvers? And even if they cared, would closed-source commercial solvers ever release their proof logs, or would they worry more about leakage of commercial solver secrets?”

Response: Customers are just interested in finding the best primal solution as fast as possible, and this is what drives commercial MIP solvers. The customer is happy with a good feasible solution, while optimality is mainly to give a criterion when the solver can stop looking for better solutions. It is not in the economic interest of commercial parties to implement proof logging. Customers are currently unaware of any problems. The question of whether we can do proof logging without revealing every secret is a good academic research question.

Operations research is not just about commercial MIP solvers, but also about solving practically relevant problems. To convince the OR community, we need to show examples where people are interested in certified correctness, e.g., kidney exchange problems, or combinatorial auctions for which there are legal fairness requirements. There is hope, but we can only do convince people if we can show that we can do proof logging, and we are not there yet. But we should identify practical problems and show that we can solve them with proof logging.

4.2 Discussion

The ensuing discussion touched upon a range of topics as outlined below.

SAT competition track for producing small proofs. Dirk Beyer proposed to have a SAT competition track where the goal is to produce small proofs, rather than necessarily solve the most instances. Marijn Heule agreed that this was an excellent idea, although it might be tricky to get the incentives right. A question is whether the evaluation metric should be the number of proof steps rather than the checking time.

Using proof logging as debugging tool. To facilitate the use of proof logging and checking as a debugging tool, it would be desirable to have an interactive proof checker, so that the proof could be fed to the checker line by line and it would be possible to stop the checker and study what the internal state is. Also, to facilitate such usage of proof logging, the proof format should be easier to read by humans, for instance, by allowing general variable names (instead of just numbers as in *DRAT*).

To what extent is certification required? There was a discussion about the papers by Mehlhorn et al. [1, 2] on certifying algorithms. A provocative question: Should we teach in our undergraduate algorithm courses that every algorithm should be certifying? The general consensus was that this would be too extreme. While there are some nice examples of certifying algorithms, these are in the minority, and it is not really doable to have certification for every single algorithm. The proof logging may not even be related to how we justify the algorithm theoretically. One should not get away with not thinking about certification at all, but banning algorithms that are not certifying would be too radical.

Manifesto on proof logging. Next, there was a discussion that it would be nice to have a manifesto on proof logging. Such a document could explain in detail what is meant by a certifying algorithms, and what different flavours of proof logging and proof checking are employed in different context. It could also establish common terminology for different concepts that arise in the context of certifying algorithms. Finally, such a manifesto could be a valuable reference, that could be cited to substantiate that research on certifying algorithms is an important endeavour.

Outcomes of the seminar

Towards the end of the panel discussion, there was a conversation about the to what extent the seminar week had been successful. It was noted that the seminar had gathered researchers from many different communities working on certifying algorithms, and that it was interesting and useful to compare and contrast different proof logging approaches. Meeting and talking, and in this way creating a common understanding in between different communities, had been valuable, and developing a common language makes it possible to communicate, and collaborate, more productively going forward. Some seminar participants pointed out, in particular, that it had been very interesting to learn about techniques for building formally verified proof checkers as a way to provide combinatorial solving with end-to-end verification.

References

- 1 Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, Christine Rizkallah and Pascal Schweitzer. *An Introduction to Certifying Algorithms*. Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik, Vol. 53(6), pp. 287–293, 2011.
- 2 Ross M. McConnell, Kurt Mehlhorn, Stefan Näher and Pascal Schweitzer. *Certifying Algorithms*. Computer Science Review, Vol. 5(2), pp. 119–161, 2011.

4.3 Closing Statements

Finally, each panelist was given the opportunity to give a short closing statement.

Ciaran McCreesh. Certification can do really something good and useful to make world a better place. Algorithms have a bad name currently. Instead, “algorithm” should become a word that people trust, like they trust, for example, bridges and elevators not to collapse.

Marijn Heule. The tools that we are developing should be used more widely. For instance, there should be potential for proof logging in the context of large language models (LLMs). Also, it would be good to develop tools to find small proofs and small counterexamples.

Ambros Gleixner. For some applications, there will be a demand for certifying algorithms, but not for every application. The first order of business, though is to develop the tools that will make proof logging possible.

Haniel Barbosa. Developers of other combinatorial solvers will not want to go through the immense pain that it has been to make *cvc5* proof producing. Proofs should have fewer details, to make it easier to produce the proof and reduce the overhead.

5 Evaluation of the Seminar by Participants

In addition to the traditional Dagstuhl evaluation after the seminar, the organizing committee also arranged for a separate evaluation which specific questions about different aspects of the seminar. Below follows a (brief and selective) summary of the answers collected in both evaluations.

In the Dagstuhl survey, the scientific quality of the seminar was ranked very highly, and the seminar also scored highly on the questions whether it inspired new ideas, led to insights from neighbouring fields or communities, and inspired new research.

In the organizers survey (which was filled in by 21 participants – a bit less than the 28 persons filling in the official Dagstuhl survey), the decision *not* to have an open problem session was mostly assessed as good or very good. A majority of respondents to the organizer poll agreed that the discussion panel that was organized on Thursday afternoon was very good. Regarding the amount of scheduled activities in the seminar program overall, there was an even split between votes for “about right” and “a bit too much.” A large majority found the balance between longer survey talks and shorter contributed talks in the program to be good, but a noticeable minority found the number of survey talks to be a bit too high.

Since the seminar tried to cover a fairly large number of different research areas related to automated reasoning and combinatorial solving, the organizer poll asked the participant whether there was a good balance between depth and breadth. Several participants commented that the coverage of many different areas was a good aspect of the seminar, and there were even suggestions for other areas to cover, such as knowledge representation and reasoning. Having more participants from industry (or more applied research) would also have been good, according to several respondents.

Among good aspects to keep for future editions in this seminar series the responses listed:

- The good balance of participants, including the mix of senior and junior researchers and coverage of different research areas.
- The balance between talks and informal discussions.
- The survey talks (where it would be good to think about how to make sure that there would be sufficient time for questions and discussions).
- The panel discussion.

Some aspects that could be improved were as follows:

- Maybe a “reading list” or similar could be provided to help participants prepare for the seminar (and to avoid survey talks having to start with the basics). Also, maybe some of the more “basic” survey talks could be scheduled in parallel?
- It would be good to get people to talk more about the problems they are encountering right now to encourage more future-looking discussions. For presentations of successfully concluded projects, there could be more of an emphasis on highlighting tools and techniques that could be useful also for other applications.
- It would be good to consider having an open problem session early on during the week, with an update on the last day on any progress made during the week.
- For future seminars on certifying algorithms, it would be good to have fewer talks. It could also make sense to have working groups on different topics scheduled on short notice during the seminar week.
- More in-depth technical discussions of different proof formats, what they can do or not do, and what the pros and cons are of different approaches.
- Some hands-on demos of different proof logging tools would have been good.

All in all, it seems fair to say that the feedback from the participants was overwhelmingly positive. When asked if they would come to a similar seminar again in Europe, 20 out of 21 respondents in the organizer poll replied that this is very likely. If such a seminar were instead to be held in North America, a clear majority would still want to come, but the enthusiasm in the responses went down slightly. For a seminar in East Asia or India, the responses were even more mixed, with positive and negative votes perfectly balanced.

Participants

- Erika Ábrahám
RWTH Aachen University, DE
- Markus Anders
RPTU Kaiserslautern-
Landau, DE
- Franz Baader
TU Dresden, DE
- Haniel Barbosa
Federal University of Minas
Gerais-Belo Horizonte, BR
- Jeremias Berg
University of Helsinki, FI
- Dirk Beyer
LMU München, DE
- Armin Biere
Universität Freiburg, DE
- Nikolaj S. Bjørner
Microsoft – Redmond, US
- Bart Bogaerts
KU Leuven, BE
- Benjamin Bogø
University of Copenhagen, DK
- Randal E. Bryant
Carnegie Mellon University –
Pittsburgh, US
- Sam Buss
University of California –
San Diego, US
- Simon Dold
Universität Basel, CH
- Bruno Dutertre
Amazon Web Services –
Santa Clara, US
- Katalin Fazekas
TU Wien, AT
- Mathias Fleury
Universität Freiburg, DE
- Ambros Gleixner
HTW – Berlin, DE
- Arie Gurfinkel
University of Waterloo, CA
- Malte Helmert
Universität Basel, CH
- Marijn J. H. Heule
Carnegie Mellon University –
Pittsburgh, US
- Matti Järvisalo
University of Helsinki, FI
- Mikoláš Janota
Czech Technical University –
Prague, CZ
- Daniela Kaufmann
TU Wien, AT
- Wietze Koops
Lund University, SE & University
of Copenhagen, DK
- Konstantin Korovin
University of Manchester, GB
- Hanna Lachnitt
Stanford University, US
- Ciaran McCreesh
University of Glasgow, GB
- Matthew Mclree
University of Glasgow, GB
- Magnus Myreen
Chalmers University of
Technology – Göteborg, SE
- Jakob Nordström
University of Copenhagen, DK &
Lund University, SE
- Andy Oertel
Lund University, SE
- Albert Oliveras
UPC Barcelona Tech, ES
- Michael Rawson
University of Southampton, GB
- Joseph Reeves
Carnegie Mellon University –
Pittsburgh, US
- Andrew Reynolds
University of Iowa –
Iowa City, US
- Tanja Schindler
Universität Basel, CH
- Martina Seidl
Johannes Kepler Universität
Linz, AT
- Monika Seisenberger
Swansea University, GB
- Mate Soos
Ethereum – Berlin, DE
- Geoff Sutcliffe
University of Miami, US
- Stefan Szeider
TU Wien, AT
- Yong Kiam Tan
Institute for Infocomm Research
(I2R), A*STAR, Singapore, SG
& Nanyang Technological
University, Singapore, SG
- Dieter Vandesande
VU – Brussels, BE
- Christoph Weidenbach
MPI für Informatik –
Saarbrücken, DE

