Marijn Heule November 2024

Research Statement

Automated reasoning has become a crucial technology in industry and academia for applications ranging from proving correctness of hardware and software to solving long-standing open problems in mathematics. This tremendous progress in performance has been labeled the "automated-reasoning revolution" [Var16]. Early in my career, I have made various contributions to that progress and implemented high-performance, award-winning solvers. Later, I focused on broader research themes that I consider at least as important as mere performance: *trust* in the correctness of solver results; widening the range of *applications* that are suitable for automated reasoning; and expanding the number of *users* that can effectively use these tools.

Trustworthiness is a key element in the value proposition of automated-reasoning tools. No matter how strong their performance, if their results cannot be trusted, they are only of limited use. The key to establishing trustworthiness is *proof*: I invented a proof system based on a single proof rule that can express all state-of-the-art propositional reasoning techniques and I demonstrated that this proof system allows reasoning tools to produce proofs while adding only little overhead to memory and runtime [9]. Members of the research community embraced this new proof system, incorporating it into their tools. The results of top-tier tools, represented in this proof system, are now routinely validated in competitions, papers, and industry, thereby boosting trust. I once received the dubious honor of producing "the largest math proof ever" in *Nature News* [Lam16]. However, even though such large proofs cannot be comprehended by humans, their validity can be checked efficiently with formally-verified proof checkers, which I co-developed [5].

To broaden the range of applications that are suitable for automated reasoning, several limitations of the technology need to be lifted. One of these limitations was the lack of scalability on multicore machines. I developed a solving paradigm that facilitates linear-time speedups even when using thousands of cores. This enabled the solution of long-standing open problems—including the fifth *Schur Number* [6] and *Keller's conjecture* [2]—using automated reasoning tools. Both problems had remained open for a century.

Finally, I want to empower people around the world by unlocking the potential of automated reasoning tools. I have collaborated with researchers from software engineering and bioinformatics to produce logic-based tools that advanced the state of the art in their respective fields [17, 3]. In both cases, the researchers were intimately familiar with the problems they had been trying to solve, but they didn't know how to represent these problems effectively in logic, to leverage the power of automated reasoning. Constructing effective problem representations is often more of an art than a science. I have thus developed powerful *automated reencoding* techniques [19], allowing users to focus on the domain problems themselves instead of their effective representations.

To achieve the high-level goals stated above, I have worked with my group on the following research topics: (1) boosting the trustworthiness of automated-reasoning tools via independently checkable certificates (Section 1); (2) understanding automated reasoning and its results (Section 2); (3) mechanizing abstract reasoning (Section 3); (4) developing representations that enable efficient reasoning (Section 4); and (5) reasoning in the cloud (Section 5).

1 Trusted Computing

Fully automated reasoning tools, which are frequently used in industry to determine the presence or absence of bugs in hardware or software, have become significantly more powerful in the last two decades. They are now able to solve long-standing open problems, but they are also highly complex, which raises the question of whether we can trust their results. This question is particularly important when automated-reasoning tools are used to determine the correctness of safety-critical systems. In such a context, correctness means that there exists no input that violates the safety requirements, which requires the exploration of the entire space of inputs. Unfortunately, verifying the correctness of automated reasoning tools often demands an infeasible amount of human effort, and it usually results in unacceptable loss of performance. There is, however, a way to work around this problem—instead of verifying the correctness of the reasoning tools themselves, we can make them produce proofs that verify their computations. This frees developers from the burden of proving their reasoning tools correct, and it allows users to check the correctness of results independently.

I played a pioneering role in the line of research to show the correctness of results via proofs. Proof production and validation have been studied for problems in propositional logic since the early 2000s, but the suggested solutions failed to become mainstream. The main hurdle was that proofs were either too big or that checking them was too costly. Together with my co-authors, I developed a proof system that bridges the gap between compact representation and efficient validation of proofs [8]. This proof system is now supported by numerous state-of-the-art tools and it has been used to provide proofs for several open mathematical problems, including the Erdős Discrepancy Problem [KL15], the Boolean Pythagorean Triples Problem [13], and Keller's Conjecture [2]. Moreover, the correctness of these proofs can be checked with tools that have been formally verified [5, Lam17]. Our proof system and the corresponding proof checkers have led to a decrease of bugs in automated-reasoning tools and to an increase of trust in their output.

One of the difficulties regarding proof production is to express—within a given proof system—the high-level reasoning techniques performed by some solvers. Even if, in theory, a proof system supports certain reasoning techniques, it can still be the case that expressing the techniques in practice is far from easy or cannot be done compactly. In a recent paper with Randy Bryant, we show an effective approach—based on binary decision diagrams—to produce proofs for many high-level reasoning techniques [4].

In the coming years, automated reasoning is poised to solve hard problems that have been open for many decades. Mathematical challenges that may be feasible for automated techniques include the *Collatz conjecture* and the *Chromatic Number of the Plane*. If automated reasoning helps us solve these problems, the solutions might reveal crucial insights that would otherwise be overlooked. However, even if this was not the case, we can still be confident that the solutions we obtain are correct because we can validate them with trustworthy systems.

Future Work I envision that the successes in establishing trust in the results of propositional reasoning can be exported to related reasoning paradigms. Some groups are exploring various options in this direction, in particular model counting (#SAT) [FHR22] and optimization [BGMN22]. Although these works represent an important step in the right direction, the checkers for these certificates are still inefficient and overly complicated. Efficient and formally-verified checkers will be crucial to have full confidence in the correctness of results in these fields. Together with colleagues at CMU, I have been working towards that goal for #SAT. One of my new Ph.D. students is expected to work on a formally-verified approach for optimization.

Also, I plan to lift our proof system to richer logics such as first-order logic and popular SMT theories, since bugs in the corresponding solvers are known to be widespread [BB09]. I have already started to lift our proof system to quantified Boolean formulas (QBF), where it can be used to certify the correctness of virtually all common preprocessing techniques. Also, insights from the QBF proof system have contributed to the development of new QBF reasoning techniques [15]. I expect that similar proof systems for SMT and first-order logic will also enable improved solving methods.

2 Reasoning and Understanding

Human proofs and computer-generated proofs differ in many respects. An argument often heard is that human proofs provide us with understanding, whereas computer-generated proofs merely show the correctness of a statement. While there is some truth to this, it is also true that modern proof-checking tools can and actually do provide us with many important insights into the nature of a problem. For instance, many proof checkers can tell us which parts of a problem specification were involved in a particular proof [9]. This can, for example, play a role when a proof of a software-verification problem shows that a certain line of code is not reachable, because the checker can help us extract the program parts that prevent the line from being reachable. Other important information such as so-called *Craig interpolants* can be obtained from automatically-generated proofs [McM03]. In a recent paper, we also showed how to extract a "skeleton" (a subset of the most important reasoning steps, akin to a proof sketch) of a proof [20]. This can help us understand the involved reasoning. Also, as we could demonstrate, a skeleton is many orders of magnitude smaller than the actual proof, making it feasible to store the skeleton and later reuse it to solve the original problem or a slight variant of it more efficiently.

Future Work Two new research directions are emerging: minimizing proofs and extracting understandable arguments from proofs. Automatically generated proofs of hard problems are typically huge and impossible to understand in practice (an example is the two-petabytes proof of Schur Number Five [6]). This is caused by weak proof systems used for reasoning. It is well known that more advanced reasoning mechanisms—which can be expressed in stronger proof systems—allow for exponentially smaller proofs. One approach for proof minimization is to "delta debug" proofs, that is, to remove as many proof parts as possible by using more advanced reasoning mechanisms. Initial experiments show that this method can shrink proofs significantly. Another approach is to solve a problem multiple times, each time jump-starting the solver with valuable information extracted from an earlier proof. Shortening an existing proof is much easier than finding a proof. Also, we can distill vital decisions from a proof and use these decisions to guide the search in a subsequent run. Machine learning techniques could be very helpful here. Ultimately, I would like to transform proofs into humanly understandable arguments. Such arguments could provide us with insights into how problems were solved, which in turn could allow mathematicians to construct smaller proofs. Moreover, it could make an important contribution to the rising field of Explainable Artificial Intelligence.

3 Mechanizing Abstract Reasoning

The success of automated reasoning presents us with an interesting peculiarity: while modern solving tools can routinely handle gigantic real-world problems, they can fail miserably on supposedly easy problems. Such poor performance is frequently caused by the weakness of their underlying proof systems—which only allow very specific kinds of low-level reasoning—while humans can often solve these problems easily by reasoning on a more abstract level. Although there exist strong proof systems that allow more abstract kinds of reasoning, their success is limited in practice as they do not seem to lend themselves easily to automation.

To deal with this issue, I have co-developed a new proof system that not only generalizes strong existing proof systems, but that is also well-suited for mechanization. This line of work was recognized with best paper awards at CADE'17 [11], HVC'17 [12], and IJCAR'18 [18]. Our new proof system turned out to be surprisingly strong, even without the introduction of new abstractions or definitions, which is a key feature of short arguments presented in the proof-complexity literature. There exist short arguments in this proof system for many well-known problems that are hard for existing automated-reasoning approaches. These problems include the so-called pigeonhole formulas, Tseitin formulas, and mutilated chessboard problems. Exploiting our new proof system's potential for automation, we later implemented a new decision procedure that finds these short arguments automatically [12].

Automated approaches can find arguments that humans never thought of, simply by reasoning on a different problem representation. We observed this many times while analyzing automatically generated solutions. Therefore, the key to solving problems that require some form of abstract reasoning is not the ability to simulate human thinking, but to equip the machine with appropriate reasoning capabilities to find its own short arguments.

Future Work Over half a century, the research regarding proof systems focused on the *existence* of short(er) arguments in one proof system compared to another proof system, without addressing the issue of how to actually *compute* these arguments. To solve hard problems, I expect that we need to shift the focus to the computability of short arguments and develop stronger and stronger proof systems together with new decision procedures and heuristics. Our new proof system is promising, although I consider it just a first step in a very exciting new research direction. The main challenge is to develop an effective decision procedure to find proofs in the new proof system for a broad range of problems.

4 Reasoning-Enabling Representations

A common approach in automated reasoning is to translate a given problem statement into propositional logic and then solve the resulting formula with a dedicated solver. As the quality of the translation has a big impact on solver performance, it is no coincidence that solvers are highly successful in the field of hardware verification: digital electronic circuits have a direct translation to propositional logic which is often adequate for solving. However, the same is not true for many other applications. For example, a former colleague tried to use off-the-shelf tools to solve problems from software-model synthesis, but his approach suffered from bad computational performance. I helped him develop a translation that was much more compact than the (published) translations he initially applied. Our resulting tool reduced the solving times for central benchmarks of the field to mere seconds. Moreover, it won the StaMinA competition 2010 where it beat existing, problem-specific approaches by a wide margin [17]. More recently, I designed effective representations of well-known problems, including Hamiltonian cycles [7] and graph coloring [10].

I have also studied how to automatically improve translations so that non-experts can achieve strong solver performance on their applications. This line of work resulted in a technique that improves the performance of reasoning engines on extremely hard bioinformatics benchmarks—used at competitive events—by two orders of magnitude [19].

Many important questions are related to proving properties of software. In contrast to hardware-related questions, software-related questions are difficult to express. Fully translating them into propositional logic is therefore often impossible, and if it is possible, it can lead to formulas that are too large for any solver. A possible approach to solve these problems is to use solvers for more specific logics such as SMT (Satisfiability Modulo Theories). However, many software-related questions are even too hard for SMT solvers or any other existing approach. One example is the question of whether or not a given function terminates. I envision that many of these questions, which are undecidable in general, can be answered using the right finitization of the search space. A promising finitization approach is the matrix interpretation method, which was able to solve various open termination problems [EWZ06]. We explored how the famous Collatz conjecture—also known as the $3n+1$ problem—could be solved by the matrix interpretation method. The Collatz conjecture, which has been open for many decades, asks if a simple recursive function terminates. Our finitization approach [21] can solve some variants of the Collatz conjecture, including Farkas' variant (possibly the hardest variant for which a proof is known), although the full conjecture is expected to be significantly harder.

Future Work The SAT community has focused almost exclusively on propositional formulas in conjunctive normal form (CNF). That focus is arguably one of the reasons for the progress, as all researchers use the same input format and a clear comparison can be made between tools. However, the challenges to construct a high-quality representation are also a result of the focus on CNF. I expect that a somewhat higher-level representation would further boost automated reasoning. An interesting generalization of CNF is a conjunction of at-least-k constraints, which express that at least k of a given set of propositions must be true. Practically all data-structure optimizations inside solvers can be efficiently lifted from clauses to at-least-k constraints, so the generalization comes without a general performance penalty while enabling much faster reasoning on many problems.

5 Reasoning in the Cloud

Supercomputers offer enormous potential to solve hard problems, but to capitalize on this potential, we need to find ways to distribute computation evenly over many different cores, which is far from trivial. I co-developed a new parallel solving paradigm, called Cube-and-Conquer [14] (best paper award HVC'11), that has been successful in realizing this goal for long-standing open problems such as the Boolean Pythagorean Triples problem [13] (best paper award SAT'16), the computation of the fifth Schur Number [6], and the resolution of Keller's conjecture [2] (best paper award IJCAR'20). The first two problems are part of Ramsey theory, a branch of mathematics that studies the emergence of patterns within structures. For many problems in Ramsey theory, there may not exist reasonably short proofs that could be found by a single computer, and the same likely holds for other problems—such as safety-related questions—too. Automated reasoning in the cloud might therefore be a viable option to solve these problems.

Reasoning in the cloud introduces various interesting new challenges. For example, in a cloud environment where software must be resilient against hardware failures, the runtime of jobs is often restricted, e.g., on AWS Lambda. If a formula cannot be solved within the runtime limit, one would like to store the current progress and continue the computation elsewhere. We developed an efficient solution for this issue, which even allows migrating the state of one automated reasoning tool to another [1].

I envision a future in which automated reasoning will be a cloud service that embodies the human-computer partnership to solve both easy and hard problems. This service will not only be able to answer many important questions in industry and academia; it will also allow external validation of enormous proofs and provide explanations that help us understand its results.

Future Work Cube-and-Conquer can be a promising technology for future parallel reasoning. However, to achieve peak performance, the method still requires expert knowledge to configure certain parameters. To remove this obstacle, I will focus on adaptive algorithms ("optimize parameters during search") and automated tuning [HHLBS09] ("optimize parameters before search"). My solvers use several adaptive algorithms [16, 14] which Donald Knuth called "elegant" [Knu15]. These algorithms were the most essential contributions that allowed my solvers to win multiple awards at the international solver competitions. I plan to develop adaptive algorithms for all major parameters in Cube-and-Conquer. Additionally, I want to enhance the paradigm such that it can efficiently count or enumerate all solutions of a given problem, thereby making it applicable for various interesting problems in cryptanalysis and system design.

To encourage users to devote substantial resources to solve a given problem, a runtime estimation is required: after spending CPU hours or years on a problem, it is disappointing to get "time out" as an answer. A reliable estimation is therefore crucial. For both the Boolean Pythagorean Triples problem and the fifth Schur Number, I was able to make accurate estimations based on sampling subproblems. Combining sampling with machine learning using online features (observations while the solver is running) is probably key to effective runtime estimation of hard problems.

Referenced Own Publications

[1] Armin Biere, Md. Solimul Chowdhury, Marijn J. H. Heule, Benjamin Kiesl, and Michael W. Whalen. Migrating solver state. In *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022*, volume 236 of *LIPIcs*, pages 27:1–27:24. Schloss Dagstuhl, 2022.

[2] Joshua Brakensiek, Marijn J. H. Heule, John Mackey, and David E. Narváez. The resolution of Keller's conjecture. *J. Autom. Reason.*, 66(3):277–300, 2022.

[3] Keenan Breik, Chris Thachuk, Marijn J. H. Heule, and David Soloveichik. Computing properties of stable configurations of thermodynamic binding networks. *Theoretical Computer Science*, 785:17–29, 2019.

[4] Randal E. Bryant, Armin Biere, and Marijn J. H. Heule. Clausal proofs for pseudo-boolean reasoning. In *TACAS'22*, volume 13243 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2022.

[5] Luís Cruz-Filipe, Marijn J. H. Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In *Automated Deduction – CADE-26*, volume 10395 of *Lecture Notes in Computer Science*, pages 220–236. Springer, 2017.

[6] Marijn J. H. Heule. Schur Number Five. In *Proc. of the 32nd AAAI Conference*. AAAI Press, 2018.

[7] Marijn J. H. Heule. Chinese remainder encoding for Hamiltonian cycles. In *Theory and Applications of Satisfiability Testing - SAT 2021 - 24th International Conference, Barcelona, Spain, July 5-9, 2021, Proceedings*, volume 12831 of *Lecture Notes in Computer Science*, pages 216–224. Springer, 2021.

[8] Marijn J. H. Heule, Warren A. Hunt, and Nathan Wetzler. Bridging the gap between easy generation and efficient verification of unsatisfiability proofs. *Softw. Test. Verif. Reliab.*, 24(8):593–607, September 2014.

[9] Marijn J. H. Heule, Warren A. Hunt, Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *FMCAD*, pages 181–188. IEEE, 2013.

[10] Marijn J. H. Heule, Anthony Karahalios, and Willem-Jan van Hoeve. From cliques to colorings and back again. In *28th International Conference on Principles and Practice of Constraint Programming, CP 2022*, volume 235 of *LIPIcs*, pages 26:1–26:10. Schloss Dagstuhl, 2022.

[11] Marijn J. H. Heule, Benjamin Kiesl, and Armin Biere. Short proofs without new variables. In *CADE-26*, volume 10395 of *LNCS*, pages 130–147. Springer, 2017.

[12] Marijn J. H. Heule, Benjamin Kiesl, Martina Seidl, and Armin Biere. PRuning through satisfaction. In *HVC'17*, volume 10629 of *LNCS*, pages 179–194. Springer, 2017.

[13] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek. Solving and verifying the Boolean Pythagorean Triples Problem via cube-and-conquer. In *SAT'16*, volume 9710 of *LNCS*, pages 228–245. Springer, 2016.

[14] Marijn J. H. Heule, Oliver Kullmann, Siert Wieringa, and Armin Biere. Cube and conquer: Guiding CDCL SAT solvers by lookaheads. In *HVC'11*, volume 7261 of *LNCS*, pages 50–65, 2011.

[15] Marijn J. H. Heule, Martina Seidl, and Armin Biere. Blocked literals are universal. In *NFM'15*, volume 9058 of *LNCS*, pages 436–442. Springer, 2015.

[16] Marijn J. H. Heule and Hans van Maaren. Effective incorporation of double look-ahead procedures. In *SAT'07*, volume 4501 of *LNCS*, pages 258–271. Springer, 2007.

[17] Marijn J. H. Heule and Sicco Verwer. Exact DFA identification using SAT solvers. In *Grammatical Inference: Theoretical Results and Applications*, pages 66–79. Springer, 2010.

[18] Benjamin Kiesl, Adrián Rebola-Pardo, and Marijn J. H. Heule. Extended resolution simulates DRAT. In *Automated Reasoning*, pages 516–531. Springer, 2018.

[19] Norbert Manthey, Marijn J. H. Heule, and Armin Biere. Automated reencoding of boolean formulas. In *HVC'12*, volume 7857 of *LNCS*, pages 102–117. Springer, 2013.

[20] Joseph E. Reeves, Benjamin Kiesl-Reiter, and Marijn J. H. Heule. Propositional proof skeletons. In *Tools and Algorithms for the Construction and Analysis of Systems - TACAS 2023*. Springer, 2023.

[21] Emre Yolcu, Scott Aaronson, and Marijn J. H. Heule. An automated approach to the collatz conjecture. In *Automated Deduction – CADE 28*, pages 468–484, Cham, 2021. Springer.

Other References

[BB09] Robert Brummayer and Armin Biere. Fuzzing and delta-debugging SMT solvers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories*, SMT '09, pages 1–5. ACM, 2009.

[BGMN22] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *AAAI-22*, pages 3698–3707. AAAI Press, 2022.

[EWZ06] Jörg Endrullis, Johannes Waldmann, and Hans Zantema. Matrix interpretations for proving termination of term rewriting. In *Automated Reasoning*, pages 574–588. Springer, 2006.

[FHR22] Johannes K. Fichte, Markus Hecher, and Valentin Roland. Proofs for Propositional Model Counting. In *SAT 2022*, volume 236 of *LIPIcs*, pages 30:1–30:24. Schloss Dagstuhl, 2022.

[HHLBS09] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: An automatic algorithm configuration framework. *J. Artif. Int. Res.*, 36(1):267–306, 2009.

[KL15] Boris Konev and Alexei Lisitsa. Computer-aided proof of Erdős discrepancy properties. *Artificial Intelligence*, 224(C):103–118, July 2015.

[Knu15] Donald E. Knuth. *The Art of Computer Programming*, volume 4 fascicle 6, Satisfiability. Addison-Wesley, 2015.

[Lam16] Evelyn Lamb. Maths proof smashes size record: Supercomputer produces a 200-terabyte proof – but is it really mathematics? *Nature*, 534:17–18, June 2016.

[Lam17] Peter Lammich. Efficient verified (UN)SAT certificate checking. In *Automated Deduction – CADE-26*, volume 10395 of *Lecture Notes in Computer Science*, pages 237–254. Springer, 2017.

[McM03] Kenneth L. McMillan. Interpolation and SAT-based model checking. In *Computer Aided Verification*, pages 1–13. Springer, 2003.

[Var16] Moshe Y. Vardi. The automated-reasoning revolution: from theory to practice and back, Spring 2016. Distinguished Lecture at NSF CISE.