Research Statement

Marijn J.H. Heule

What makes some scientific problems hard and others easy? There are many answers to this question, such as the level of abstraction and the amount of computation required to obtain a solution. One thing is for certain: there is no shortage of problems that are hard *by any metric*, and these problems can bring even the greatest minds to their knees. Luckily, mankind is not alone in its quest for knowledge; it created itself a powerful companion—the computer. While humans are great at many aspects of problem solving, we are no match for computers in a range of ways, including precise logical reasoning. Such reasoning is crucial in industry and academia, for instance to ensure that computer systems work exactly as intended, in fields ranging from health care to finance to aviation. Together, man and machine can form a partnership with skills that go far beyond what either of them is capable of individually. I believe in a future where they work closely together to tackle the hard problems we so desperately crave to solve. In fact, I am already contributing to that future.

For example, I have co-developed an automated-reasoning approach that was able to solve the *Pythagorean Triples Problem* [8], a long-standing mathematical problem whose solution received substantial media coverage: *Nature News* [17] and dozens of other media outlets around the world reported on what they called the "the largest math proof ever"—a 200-terabyte computer-generated proof. I followed up on this result by computing the fifth *Schur Number* [4] (a combinatorial challenge that remained open for a century) and the smallest known unit-distance graph with chromatic number five. The latter result was published in *Quanta Magazine* [18], reprinted by *Wired*, and reported on in international newspapers. I have also collaborated with people from software engineering and bioinformatics to produce logic-based tools that advanced the state of the art in their respective fields.

But these examples just show the potential. My goal is to solve many hard open problems arising from scientific and industrial challenges, and to help others to do so. To realize this goal, I plan to focus on the following research topics: (1) boosting the trustworthiness of automated-reasoning tools via independently checkable certificates; (2) understanding automated reasoning and its results; (3) mechanizing abstract reasoning; (4) developing representations that enable efficient reasoning; and (5) reasoning in the cloud.

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 these tools are often 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. Here, correctness means that there exists no input that violates the safety requirements. So we need to be sure that the entire space of inputs has been explored.

To deal with this issue, automated-reasoning tools are often required to provide an easily checkable output that certifies the correctness of their answers—a so-called *proof*. 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 [5]. This proof system is now supported by a large number of state-of-the-art tools and it has been used to provide proofs for several open mathematical problems, including the Erdős Discrepancy Problem [16] and the Boolean Pythagorean Triples Problem [8]. Moreover, formally-verified tools can check the correctness of these proofs [2, 19]. Our proof system and the corresponding proof checkers have led to a decrease of observed bugs in automated-reasoning tools and to an increase of trust in their output. In the future, 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 [1]. 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. One of my NSF grants focuses on finalizing this QBF proof system and providing a formally-verified proof checker. Also, insights from the QBF proof system have contributed to the development of new QBF reasoning techniques [10]. I expect that similar proof systems for SMT and first-order logic will also enable improved solving methods.

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 are, for instance, *Ramsey Number Five*, the *Collatz conjecture*, and the *Chromatic Number of the Plane*. The solutions of these problems may reveal crucial insights that might otherwise be overlooked. However, even if this is not the case, we can be confident that they are correct because we can validate them with trustworthy systems.

Reasoning and Understanding

Human proofs and computer-generated proofs differ in many aspects. 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 argument, it is also true that modern proof-checking tools can actually 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. 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. Also, other important information such as so-called *Craig interpolants* can be obtained from automatically-generated proofs [21]. I envision that this is just the tip of the iceberg when it comes to mining proofs.

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 2-petabytes proof of Schur Number Five [4]). This is caused by the 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 proofs. 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 would be an important contribution to the rising field of Explainable Artificial Intelligence.

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 often fail miserably on supposedly easy problems. Their 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 to automation.

To deal with this issue, I have been co-developing 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 [6], HVC'17 [7], and IJCAR'18 [14]. 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 pigeon hole 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 [7].

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. So 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.

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 *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.

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 the 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 [12]. I have also been studying how to automatically fix 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 in competitive events—by two orders of magnitude [20].

Many important questions are related to proving properties of software. Unfortunately, software-related questions are more complicated than hardware-related questions. 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 [3]. In a recently awarded NSF grant, we explore 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. Preliminary results of our work show that our finitization approach can solve some weaker variants of the Collatz conjecture (which were too hard for alternative methods), although the full conjecture is expected to be significantly harder.

Reasoning in the Cloud

Supercomputers offer enormous potential to solve hard reasoning problems. However, 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 [9] (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 [8] (best paper award SAT'16) and the computation of the fifth Schur Number [4]. Both 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.

The Cube-and-Conquer paradigm realized linear speedups on both the Pythagorean Triples problem and the Schur Number five problem, even when using thousands of cores to solve them. In other words, it was able to distribute the computations in such a way that the speedup depended linearly on the number of cores employed, thus making efficient use of parallelism. In a nutshell, Cube-and-Conquer partitions a problem into a large number (sometimes even billions) of subproblems using heuristics that reason about the entire search space. It then handles these subproblems with dedicated techniques that can solve them quickly. 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 [13] ("optimize parameters before search"). My solvers use several adaptive algorithms [11, 9] which Donald Knuth called "elegant" in his latest book [15]. These algorithms were the most essential contributions that allowed my solvers to win many 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.

I envision a future in which automated reasoning will be a cloud service that embodies the human-computer partnership to solve 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.

References

[1] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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, September 2009.

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

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

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

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

[18] Evelyn Lamb. Decades-old graph problem yields to amateur mathematician. *Quanta Magazine*, April 17, 2018.

[19] 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.

[20] 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.

[21] K. L. McMillan. Interpolation and SAT-based model checking. In Warren A. Hunt and Fabio Somenzi, editors, *Computer Aided Verification*, pages 1–13, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.