EPiC Series in Computing
Volume 51, 2017, Pages 11–15
ARCADE 2017. 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements

Making Automatic Theorem Provers more Versatile

Simon Cruanes

University of Lorraine, CNRS, Inria, LORIA, 54000 Nancy, France

Abstract

We argue that automatic theorem provers should become more versatile and should be able to tackle problems expressed in richer input formats. Salient research directions include (i) developing tight combinations of SMT solvers and first-order provers; (ii) adding better handling of theories in first-order provers; (iii) adding support for inductive proving; (iv) adding support for user-defined theories and functions; and (v) bringing to the provers some basic abilities to deal with logics beyond first-order, such as higher-order logic.

Much effort has been spent on developing Automatic Theorem Provers for first-order logic, most notably with superposition [3, 22, 24], and Satisfiability Modulo Theories (SMT) [4, 5, 16]. Other domains of research have made extensive use of such automatic provers. Domains of application include powering software verification tools such as F* [25] and Why3 [7]; discharging proofs in proof assistants such as Isabelle/HOL with Sledgehammer [20] and TLAPS [13]; synthesizing functions from specifications [1]; etc. (see Voronkov's study [26] for more details).

However, in many cases, the current technologies are limiting factors in their applications, and need to be complemented by user intervention. In Why3, some proofs must be discharged manually in Coq [14]; in ACL2, the user is expected to provide some inductive lemmas; in Isabelle/HOL, Sledgehammer does not succeed in the majority of cases. Even SMT solvers, very successful on ground problems, are often called on quantified formulas — where they are incomplete — to express frame conditions or axiomatize theories they do not handle natively. Automatic inductive provers do exist, but they are usually not good on general first-order logic (a notable exception is CVC4 [21]).

Even then, some use cases involve formulas that lie completely outside the scope of all existing systems. Sledgehammer needs to tackle problems containing higher-order formulas (even constructs as basic as sets are represented by predicates $\alpha \to \mathsf{bool}$), (co)datatypes (and therefore (co)induction), polymorphic types, choice operators, etc. Because even state of the art automatic provers are unable to process such rich problems, Sledgehammer relies on costly encodings which tend to add a lot of overhead, making problems much harder. An equivalent tool for Coq would, in addition, need to deal with dependent types, but many first-order provers still do not support even simple types. In software verification, tools often need to perform induction themselves, because the SMT solvers they call are unable to do so; and theorem proving in higher-order logic is still very difficult to automate.

Therefore, we reckon that the limited scope of individual automatic proof systems is a critical limitation to their applicability. However, the decades of theoretical research and practical implementation, both in SMT and first-order theorem proving, should not be lost. Extending the scope of existing provers and techniques should therefore be an important direction of

research: it is more useful to have provers that are good on a wide range of problems, than provers which are excellent for a narrow set of constructs that are not sufficiently expressive for applications. We list a few salient scope extensions that are already required in many use cases.

Handling Theories by Combining SMT and Superposition A part of the success of SMT solvers in software and hardware verification is their reliable support for multiple theories, such as arithmetic (for integers, rationals, reals), arrays, bitvectors, etc. First-order provers have been historically more limited in this area. It is only in the last few years, with refinements of hierarchic superposition [6], AVATAR [27], or Tableaux with constraints [23] that first-order that some provers with support for arithmetic have appeared. In some cases, Superposition provers can be used as SMT solvers [2]. A challenging, but rewarding, direction of research should be to combine more tightly superposition-based provers — which have been dominating FO logic — and SMT solvers. Such an endeavour should spark cooperation between the first-order and SMT communities. Indeed, the former excel in quantified formulas with equality and function symbols, whereas the latter are good on ground problems with theories but use heuristic techniques for quantified formulas. Some already existing techniques for solving this problem are AVATAR and hierarchic superposition, both implemented in theorem provers; additional theoretical work include DPLL(Γ+T) [8] and speculative inferences [9], but it remains to be seen whether practical implementations can be devised.

User-defined Theories with Deduction Modulo If a prover does not support a particular theory, the user's only recourse if generally to add (quantified) axioms. Each axiom can have a significant impact on the search space, the number of formulas, and therefore on the time needed to discharge a goal. In superposition, the axioms might interact with one another, creating many clauses that do not contribute to refuting the goal; in SMT solvers with trigger-based instantiation, each axiom can yield many ground formulas that will slow the solver down. Some axioms, such as associativity and commutativity of a symbol, are known to be extremely harmful to provers that lack specific mechanisms to handle them.

A solution to that issue might be Deduction Modulo [17], in which theories are specified by a set of rewrite rules. The rewrite rules act on terms, but also on (atomic) formulas. For example, the theory of typed sets can be encoded into rewrite rules such as $x \in (A \cup B) \rightsquigarrow (x \in A \vee x \in B)$; this helps solving proofs obligations stemming from the B method [12].

A related issue is that automated provers usually lack the notion of (recursive) defined function that is pervasively used in proof assistants or inductive theorem provers such as ACL2 [19]. Efficient processing of such functions is still beyond the reach of provers; deduction modulo and rewriting might be a solution.

Induction Users of Sledgehammer might be surprised to realize that Sledgehammer cannot show $\forall x\, y : \mathsf{nat}.\ x + y = y + x$. Proof assistants such as Coq and Isabelle/HOL make heavy use of datatypes and induction in their standard library as well as in many developments. In hardware verification, because state spaces are huge, proving a property might involve finding an inductive strengthening of the property, rather than exploring every state [10]. In software verification, loop invariants or function invariants abound.

Still, to the best of our knowledge, the intersection of inductive provers and generalist automatic provers contains only CVC4 [21]. The vast majority of automatic provers are unable to perform even the simplest inductive proof. For simple induction, it suffices to generate each case in some other tool and call the automatic prover on each case. However, it is often

necessary to analyze why subgoals failed, to strengthen the property, and to produce and prove intermediate lemmas [11, 19]. This can only be done from within the prover. Successful automatic induction requires tight cooperation between the procedure that proves each case (e.g., to show $p(n) \Rightarrow p(n+1)$) and the procedure responsible for applying the induction schema in nested induction or to show intermediate lemmas. A classic example of such cooperation is the Boyer-Moore waterfall [19].

Beyond First-order Logic First-order logic occupies a sweet spot between expressiveness and computational properties — its semi-decidability, and the ability to rely on unification to guide proofs. Alas, sometimes its expressiveness is not quite sufficient for the task at hand. Properties involving sets, comprehensions, sums and integrals, as often expressed in proof assistants, are difficult to express with first-order constructs. Sometimes, a single higher-order formula is enough to prevent Sledgehammer from finding a proof.

Mixing terms and formulas is already possible in SMT solvers and in Vampire [18]. However, higher-order provers have difficulties coping with the search space and perform poorly on first-order logic, suggesting that there is room for improvement here. The Matryoshka project is an ongoing effort to remediate to this issue, but it seems that going towards richer logics will likely continue to be a challenge.

The Current Situation Some proof systems have been making progress in the directions listed above, as hinted by the following (non-exhaustive) list:

CVC4 [4] is a comprehensive SMT solver, featuring many theories, including strings, bitvectors, (co)datatypes, and floating point numbers; moreover, it can perform inductive reasoning on datatypes and generate inductive lemmas. It also performs quite well on first-order logic and has been entering the arithmetic track of CASC; CVC4 is possibly the most versatile automatic prover to date.

Vampire [22] has started participating in the SMT competition with good results, in addition to its excellent results in most categories of CASC. It now supports arithmetic via an extension of AVATAR, and boolean formulas occurring in terms.

Zipperposition [15] is our superposition prover, extended with integer linear arithmetic, deduction modulo, and inductive reasoning; we are currently implementing support for higher-order logic.

References

[1] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, pages 1–8. IEEE, 2013.

[2] Alessandro Armando, Maria Paola Bonacina, Silvio Ranise, and Stephan Schulz. New results on rewrite-based satisfiability procedures. *ACM Transactions on Computational Logic (TOCL)*, 2009.

[3] Leo Bachmair and Harald Ganzinger. On Restrictions of Ordered Paramodulation with Simplification. In Mark E. Stickel, editor, *10th International Conference on Automated Deduction*, volume 449 of *Lecture Notes in Computer Science*, pages 427–441. Springer, 1990.

[4] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *CAV 2011*, volume 6806 of *LNCS*, pages 171–177. Springer, 2011.

[5] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in SAT modulo theories. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 512–526. Springer, 2006.

[6] Peter Baumgartner and Uwe Waldmann. Hierarchic superposition with weak abstraction. In *Automated Deduction–CADE-24*. Springer, 2013.

[7] François Bobot, Jean-Christophe Filliâtre, Claude Marché, and Andrei Paskevich. Why3: Shepherd Your Herd of Provers. In *Boogie 2011: First International Workshop on Intermediate Verification Languages*, pages 53–64, Wrocław, Poland, August 2011.

[8] Maria Paola Bonacina, Christopher A. Lynch, and Leonardo de Moura. On deciding satisfiability by DPLL(Γ+T) and unsound theorem proving. *Proceedings of the Twenty-second International Conference on Automated Deduction (CADE)*, pages 35–50, 2009.

[9] Maria Paola Bonacina, Christopher A. Lynch, and Leonardo de Moura. On deciding satisfiability by theorem proving with speculative inferences. 2011.

[10] Aaron R Bradley. SAT-based model checking without unrolling. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 70–87. Springer, 2011.

[11] Alan Bundy, Andrew Stevens, Frank van Harmelen, Andrew Ireland, and Alan Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62(2):185 – 253, 1993.

[12] Guillaume Bury, David Delahaye, Damien Doligez, Pierre Halmagrand, and Olivier Hermant. Automated Deduction in the B Set Theory using Typed Proof Search and Deduction Modulo. In *LPAR (short papers)*, pages 42–58, 2015.

[13] Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. Verifying safety properties with the TLA+ proof system. In *International Joint Conference on Automated Reasoning*, pages 142–148. Springer, 2010.

[14] The Coq Development Team. The Coq Proof Assistant. http://coq.inria.fr/.

[15] Simon Cruanes. *Extending Superposition with Integer Arithmetic, Structural Induction, and Beyond*. PhD thesis, École polytechnique, September 2015.

[16] Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, chapter 24, pages 337–340. Springer, Berlin, Heidelberg, 2008.

[17] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem Proving Modulo. *Journal of Automated Reasoning*, 2003.

[18] Evgenii Kotelnikov, Laura Kovács, Giles Reger, and Andrei Voronkov. The Vampire and the FOOL. In *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 37–48. ACM, 2016.

[19] J Strother Moore and Claus-Peter Wirth. Automation of Mathematical Induction as part of the History of Logic. *arXiv preprint arXiv:1309.6226*, 2013.

[20] Lawrence C. Paulson and Jasmin Christian Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In Geoff Sutcliffe, Stephan Schulz, and Eugenia Ternovska, editors, *IWIL 2010*. EasyChair, 2012.

[21] Andrew Reynolds and Viktor Kuncak. Induction for SMT solvers. In *Verification, Model Checking, and Abstract Interpretation (VMCAI)*, 2015.

[22] Alexandre Riazanov and Andrei Voronkov. Vampire 1.1 (system description). In *Proceedings of the First International Joint Conference on Automated Reasoning*, IJCAR '01, pages 376–380, London, UK, UK, 2001. Springer-Verlag.

[23] Philipp Rümmer. A Constraint Sequent Calculus for First-Order Logic with Linear Integer Arithmetic. In Iliano Cervesato, Helmut Veith, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, volume 5330 of *Lecture Notes in Computer Science*, pages 274–289. Springer Berlin Heidelberg, 2008.

[24] Stephan Schulz. E - a brainiac theorem prover. *AI Commun.*, 15, 2002.

[25] Nikhil Swamy, Juan Chen, Cédric Fournet, Pierre-Yves Strub, Karthikeyan Bhargavan, and Jean Yang. Secure distributed programming with value-dependent types. In Manuel M. T. Chakravarty, Zhenjiang Hu, and Olivier Danvy, editors, *Proceeding of the 16th ACM SIGPLAN international conference on Functional Programming*, pages 266–278. ACM, 2011.

[26] Andrei Voronkov. Automated reasoning: Past story and new trends. In *IJCAI*, pages 1607–1612, 2003.

[27] Andrei Voronkov. AVATAR: the architecture for first-order theorem provers. In *CAV 2014*, pages 696–710, 2014.