EPiC Series in Computing
Volume 51, 2017, Pages 55–63
ARCADE 2017. 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements

Checkable Proofs for First-Order Theorem Proving*

Giles Reger[1] and Martin Suda[2]

[1] University of Manchester, Manchester, UK
[2] TU Wien, Vienna, Austria

Abstract

Inspired by the success of the DRAT proof format for certification of boolean satisfiability (SAT), we argue that a similar goal of having unified automatically checkable proofs should be sought by the developers of automatic first-order theorem provers (ATPs). This would not only help to further increase assurance about the correctness of prover results, but would also be indispensable for tools which rely on ATPs, such as "hammers" employed within interactive theorem provers. The current situation, represented by the TSTP format, is unsatisfactory, because this format does not have a standardised semantics and thus cannot be checked automatically. Providing such semantics, however, is a challenging endeavour. One would ideally like to have a proof format which covers only-satisfiability-preserving operations such as Skolemisation and is versatile enough to encompass various proving methods (i.e. not just superposition) or is perhaps even open-ended towards yet to be conceived methods or at least easily extendable in principle. Going beyond pure first-order logic to theory reasoning in the style of SMT, or beyond proofs to certification of satisfiability are further interesting challenges. Although several projects have already provided partial solutions in this direction, we would like to use the opportunity of ARCADE to further promote the idea and gather critical mass needed for its satisfactory realisation.

1 The challenge

We would like to propose to the first-order ATP community the challenge of designing, implementing and bringing into practice a unified mechanically checkable proof format along with an efficient proof checker. The format should support the whole reasoning pipeline including formula preprocessing, be sufficiently general to cover all the solving techniques currently employed by ATPs, and be open to future extensions for proof recording of techniques yet to be developed. In this paper, we summarise the current situation regarding proof output of ATPs, explain why we think striving for a mechanically checkable proof format is a worthy effort, list the main properties we believe an ideal format should satisfy, attempt to give an overview of work already done in the first-order ATP community and related areas, and, finally, suggest possible avenues and the next steps to be taken for meeting the challenge.

At this point we add the disclaimer that other people have already examined this challenge in various ways. We attempt to present this previous work and do not claim that what we are suggesting is novel, but instead we are calling for further work in this area. Our main aim at ARCADE is to solicit opinions from experts on why the proposed idea has not yet made its way to practice and on how exactly should the community proceed to achieve the envisioned goal.

*Martin Suda was supported by ERC Starting Grant 2014 SYMCAR 639270 and the Austrian research projects FWF S11403-N23 and S11409-N23.

2 Why (mechanically checkable) proofs?

There are various reasons why we might want a reasoning system to provide a proof as a justification for its answer. First of all, a proof may serve as *an explanation* for the user, showing why exactly the established result holds. This is, roughly speaking, why mathematicians are interested in proofs. Although improving human understanding of machine generated proofs is an interesting research topic, it is separate from our challenge and we do not pursue it here.

As a second perspective, we may view a proof as *a certificate* supporting the claim of the reasoner, thus increasing our confidence that the reported result actually holds. In fact, a *mechanically checkable* proof completely lifts the trust burden from the reasoner—which may be e.g. too complex to be fully trusted—because it provides an independent means for verifying the result. As a side note in this context, we point out that an automatic proof checker can serve as an indispensable tool for debugging the reasoner during its development [43].

Finally, a proof can be understood as the *primary output* of the reasoner to be further processed by other tools. A proof may be, e.g., 1) communicated to another tool within a combined reasoning system such as a static program analyser, 2) used for extracting interpolants [29, 35], or 3) visualised [54]. An important category of tools which rely on ATPs and on proofs generated by them are the so called *hammers* [40, 30, 1]. A hammer is a tactic in an interactive theorem prover which harnesses the power of ATPs to improve the user experience by trying to discharge selected proof obligations. These tools typically employ *proof reconstruction* techniques to lift a proof found by an ATP to their internal format.

We can see that mechanically checkable proofs are clearly desirable, especially in the context where the correctness of the reasoner's result is of high importance but its code cannot be fully trusted. It seems intuitively clear that a necessary step on the way to full automation relies on providing not only syntax but also a formal semantics for the proposed proof format. Existence of unified syntax and semantics should also make the development of hammers much easier. Not only because there would be no need to interface each individual employed ATP separately, but also because it should allow for a provably unfailing translation from the ATP proof to the internal format of the interactive theorem prover. Currently, proof reconstruction in a hammer may fail for several reasons [8].

3 Current situation

Most current first-order ATPs, such as E [45], iProver [34], or Vampire [36], output proofs in the TSTP (Thousands of Solutions from Theorem Provers) format [51] from the TPTP initiative [50].[1] This format has a standard syntax and fixed conventions [53]. It is naturally centred around the notion of an inference that connects a list of premise formulas to a conclusion formula and records an identifier of the used *inference rule*. It is expected that a typical inference is *sound*, i.e. that the premises imply the conclusion, although a mechanism exists for marking certain inferences (such as Skolemisation steps) as satisfiability preserving only. In general, however, it is left unspecified what exact formula manipulation each individual inference represents and the inference rule identifiers are used simply as tags in a system dependent manner.[2]

Clearly TSTP can be used to capture a lot of the needed information about proofs. However, since TSTP has no official semantics it precludes reliable automatic checking. Still, all the inferences which are marked as logical entailments can be independently verified by a trusted system, as done for example by the GDV verifier [49]. Vampire provides a similar functionality [41]. However, we remark there is no

[1] One exception is SPASS [57], which generates proofs in its custom DFG (Deutsche Forschungsgemeinschaft) format.
[2] Although see [9] on extension of the TSTP framework to support semantic specification of inference rules used in proofs.

guarantee a trusted system will be able to reprove every sound inference within the given resource limits, because even for simple steps like resolution the resulting proof obligation may be far from trivial.[3]

4 An ideal proof format

We argued that it is desirable to have a mechanically checkable proof format for first-order ATPs. Here we explicitly state additional properties we believe such a format should have.

Generality The format should be sufficiently general to cover all the solving techniques currently employed by ATPs. It should be able to accommodate not only the mainstream calculi such as resolution and superposition [2, 37], but also less traditional ones (e.g. InstGen [33], Model Evolution [6], or Geometric Resolution [20]).[4] Ideally, however, we envision a format open to future extensions for recording of techniques yet to be developed.

Preprocessing and "unsound" steps Although the traditional view of a proof is that of a sequence of sound inferences, i.e. ones in which the conclusion logically follows from the premises, the proof format we envision should cater for steps which violate this condition. This is, in particular, important for capturing preprocessing and normal form transformation rules such as formula naming and Skolemisation [39, 42] which introduce new symbols and are logically unsound.[5] Another useful technique that would fit into this category would be symmetry breaking [22].

Efficiency We believe that a good proof format should be checkable efficiently, i.e. ideally in low order polynomial time. Note that is in general incompatible with leaving up to the checker to perform any kind of search and might require recording more information than provers traditionally do, such as the applied unifier or the term position where demodulation is applied [41].

Easy implementation and low overhead Slightly incompatible with the previous item is the requirement that proof output generation should be easy to implement and should not slow down the actual execution of the prover more than by a small margin.

General adoption Finally, an ideal proof format is generally accepted by the community and supported by all the major tools. This signals to the developers of new systems or auxiliary tools that their investment into adopting the format will have the highest possible impact.

Although the last requirement is "societal" rather than technical, we admit it may play the role of the major obstacle to our challenge.

5 Some previous work and related approaches

One piece of inspiration for this proposal is the existence of the DRAT proof format for certifying propositional satisfiability (SAT) solvers [26, 27]. It is a proof format that—in the context of SAT—appears to satisfy all the properties listed in the previous section. DRAT is general enough to capture most if not all the presently known SAT techniques, goes beyond sound inferences to accommodate recording of preprocessing and in-processing steps [28], and relies on a small fixed set of rules which makes it easy to both emit and check [58]. Recent work has focused on developing a verified checker for

[3] In our own experience checking Vampire proofs in this way, a small percentage of proof steps were only reproducible by earlier versions of Vampire, and a (very) few (manually verified) steps were not reproducible by any other solver.

[4] Another well known challenge is that of capturing clause splitting rules [56, 55].

[5] A correct application of such a rule relies on checking a global "freshness" condition for the new symbol.

DRAT [15, 16]. While it may be too optimistic to assume that an analogous sufficiently versatile small fixed set of rules can be found for first-order logic, the idea should nevertheless be seriously considered as topic for research. Some groundwork has already been laid by Kiesl and Suda [32].

An alternative approach to achieving generality of a proof format is by framing it as a translation into a sufficiently strong target logic, ideally already equipped with a trusted checker. For instance, the certifier for automatically generated termination proofs CeTA [52] ultimately relies on formalisation in Isabelle/HOL [38]. It is backed up by the library IsaFoR (Isabelle Formalization of Rewriting) which can be gradually extended as new termination techniques are invented. Moreover, thanks to Isabelle's support for code-generation [24], CeTA arises as a standalone executable providing arguably higher performance of certification than could be achieved by targeting Isabelle itself and checking the proofs through its LCF-style trusted kernel [23].

An independently developed universal proof format Dedukti is based on $\lambda\Pi$-*calculus modulo* [14, 10]. Dedukti comes equipped with a proof checker and translators from several existing proof formats [21] and has been used to encode resolution and superposition proofs [13], among others. It seems that the Dedukti proof format deserves more attention from the ATP community than it has received so far.

A similarly ambitious initiative based on the same foundations in the context of SMT is the LFSC proof format proposed by Stump et al. [47, 46, 48]. Here, LF stands for the Edinburgh Logical Framework [25] (also based on $\lambda\Pi$-calculus as its meta-language) and SC stands for Side Conditions, which are pieces of trusted code in a small custom programming language used to express more complicated rules. LFSC is the proof language currently employed by the SMT solver CVC4 [4].[6]

There have been other efforts by the SMT community to devising a common proof format. One example is the proposal by Besson et al. [7] inspired in syntax by the SMT-LIB 2.0 standard and currently adopted by veriT [12].[7] Another format is used in Z3 [17, 18]. These differ by the level of detail provided, which correlates with the ease of generation and the efficiency of subsequent proof checking. We refer the reader to a recent survey of the landscape of proof formats in SMT [5].

6 Wrapping up

While the above list of references is by no means complete, it shows that a lot has already been done in the direction of universal automatically checkable proof formats. We believe that the developers of first-order ATPs should closely follow especially the developments in the SMT community for the following reasons: 1) many-sorted first-order logic, the input logic of state-of-the-art ATPs, is subsumed by the SMT input format, namely by the (quantified) theory of uninterpreted functions; 2) at the same time, there is already a growing interest in extending the capabilities of ATPs to supporting other theories such as forms of arithmetic; 3) application areas such as program analysis and verification would clearly benefit from a common format.

Before attempting a new solution, sufficient time should be spent on analysing the existing approaches, their advantages and disadvantages, and especially the possible reasons why none of them has yet become a standard in its target domain. The above mentioned survey [5] lists the following as possible reasons why none of the proposed formats have caught on as a general proof language: low priority of the proof output effort amongst other development tasks, differences of opinion on what features should be included in the standard, and the overhead connected with switching from the currently adopted approach to a different one. It is also universally acknowledged that an effective incentive for the developers is requiring proof output during systems' competitions. We hope that by formulating this challenge as a topic for discussion at ARCADE, we are further contributing to its successful realisation.

[6] Recent work on proof output in CVC4 focuses on efficient *lazy* proof generation [31].

[7] While the original proposal openly avoided dealing with preprocessing to keep the exposition simple, in a more recent work Barbosa et al. [3] focus exactly on this topic. They remark that work has already been done in this direction by De Nivelle [19].

7 Post Workshop Reflections

We have chosen to include a brief section summarising discussions at the ARCADE workshop, rather than attempting to inline the results of these discussions.

Are we ready? It was pointed out that we are perhaps not ready for a general proof format and that trying to create one before it is clear what it should be might restrict future work. This seems especially true if we are trying to accommodate both superposition-based ATPs and CDCL(T)-style SMT solvers. First-order theorem proving was contrasted with SAT solving and it was highlighted that there is no *prime calculus* in the first-order case, i.e., we are not discussing variants of, e.g., resolution but fundamentally different calculi.

What do we want? Our starting point was that we want checkable proofs where the reason for correctness was easy to understand. There were two counterpoints to this. Firstly, one person argued that they just want to know that the proof is correct, not why. This seems a reasonable lesser goal for many use-cases. However, we feel that the why is still important in some cases. Secondly, and more significantly, within the interactive theorem proving setting (e.g. the application of so-called Hammers) it was suggested that

1. a proof should be at the same level of *abstraction* as the input; and
2. a proof should be *short*

both of which are (to different degrees) at odds with our goal of checkable proofs. The shortness requirement comes from the fact that these systems view proofs as proof sketches; they do not want to trust the proofs but want to use them to attempt to reconstruct proofs in their own systems and therefore they need proofs to be short. However, one approach to producing a checkable proof format would be to define a simple calculus where inferences in a particular system are translated into multiple steps in this simple calculus i.e. proofs would become longer. In general, including enough information to support checkability will necessarily lengthen a proof.

Competition support? It was generally agreed that if a general proof format is to be developed then it would need to be supported by competitions such as CASC and SMT-COMP. It should be noted that these two competitions are at different places in their requirement for proofs – (semi-formal) proofs are required in CASC but not in SMT-COMP, reflecting the different statuses in the related two fields.

References

[1] Jesse Alama. Escape to mizar from atps. In Pascal Fontaine, Renate A. Schmidt, and Stephan Schulz, editors, *Third Workshop on Practical Aspects of Automated Reasoning, PAAR-2012, Manchester, UK, June 30 - July 1, 2012*, volume 21 of *EPiC Series in Computing*, pages 3–11. EasyChair, 2012.

[2] L. Bachmair and H. Ganzinger. Resolution theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 2, pages 19–99. Elsevier Science, 2001.

[3] Haniel Barbosa, Jasmin Christian Blanchette, and Pascal Fontaine. An efficient proof-producing framework for formula processing. In *CADE 2017*, 2017. To appear.

[4] C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification*, number 6806 in Lecture Notes in Computer Science, pages 171–177. Springer-Verlag, 2011.

[5] C. Barrett, L. de Moura, and P. Fontaine. Proofs in satisfiability modulo theories, 2015. All about Proofs, Proofs for All.

[6] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In Franz Baader, editor, *Automated Deduction - CADE-19, 19th International Conference on Automated Deduction Miami Beach, FL, USA, July 28 - August 2, 2003, Proceedings*, volume 2741 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2003.

[7] Frédéric Besson, Pascal Fontaine, and Laurent Théry. A Flexible Proof Format for SMT: a Proposal. In Pascal Fontaine and Aaron Stump, editors, *First International Workshop on Proof eXchange for Theorem Proving - PxTP 2011*, Wroclaw, Poland, August 2011.

[8] Jasmin Christian Blanchette, Sascha Böhme, Mathias Fleury, Steffen Juilf Smolka, and Albert Steckermeier. Semi-intelligible Isar proofs from machine-generated proofs. *J. Autom. Reasoning*, 56(2):155–200, 2016.

[9] Roberto Blanco, Tomer Libal, and Dale Miller. Defining the meaning of TPTP formatted proofs. In Boris Konev, Stephan Schulz, and Laurent Simon, editors, *IWIL@LPAR 2015, 11th International Workshop on the Implementation of Logics, Suva, Fiji, November 23, 2015*, volume 40 of *EPiC Series in Computing*, pages 78–90. EasyChair, 2015.

[10] Mathieu Boespflug, Quentin Carbonneaux, and Olivier Hermant. The lambda-pi-calculus modulo as a universal proof language. In David Pichardie and Tjark Weber, editors, *Second International Workshop on Proof Exchange for Theorem Proving*, 2012.

[11] Maria Paola Bonacina, editor. *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, volume 7898 of *Lecture Notes in Computer Science*. Springer, 2013.

[12] Thomas Bouton, Diego Caminha Barbosa De Oliveira, David Déharbe, and Pascal Fontaine. veriT: An open, trustable and efficient smt-solver. In Schmidt [44], pages 151–156.

[13] Guillaume Burel. A shallow embedding of resolution and superposition proofs into the $\lambda\Pi$-calculus modulo. In Jasmin Christian Blanchette and Josef Urban, editors, *Third International Workshop on Proof Exchange for Theorem Proving, PxTP 2013, Lake Placid, NY, USA, June 9-10, 2013*, volume 14 of *EPiC Series in Computing*, pages 43–57. EasyChair, 2013.

[14] Denis Cousineau and Gilles Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In Simona Ronchi Della Rocca, editor, *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4583 of *Lecture Notes in Computer Science*, pages 102–117. Springer, 2007.

[15] Luís Cruz-Filipe, Marijn Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. *CoRR*, abs/1612.02353, 2016.

[16] Luís Cruz-Filipe, Marijn Heule, Warren A. Hunt Jr., Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified RAT verification. In *The 26th International Conference on Automated Deduction, Gothenburg 6–11 August*, 2017. To appear.

[17] Leonardo Mendonça de Moura and Nikolaj Bjørner. Proofs and refutations, and Z3. In Piotr Rudnicki, Geoff Sutcliffe, Boris Konev, Renate A. Schmidt, and Stephan Schulz, editors, *Proceedings of the LPAR 2008 Workshops, Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics, Doha, Qatar, November 22, 2008*, volume 418 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[18] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proc. of TACAS*, volume 4963 of *LNCS*, pages 337–340, 2008.

[19] Hans de Nivelle. Extraction of proofs from the clausal normal form transformation. In Julian C. Bradfield, editor, *Computer Science Logic, 16th International Workshop, CSL 2002, 11th Annual Conference of the EACSL, Edinburgh, Scotland, UK, September 22-25, 2002, Proceedings*, volume 2471 of *Lecture Notes in Computer Science*, pages 584–598. Springer, 2002.

[20] Hans de Nivelle and Jia Meng. Geometric resolution: A proof procedure based on finite model search. In Ulrich Furbach and Natarajan Shankar, editors, *Automated Reasoning, Third International Joint Conference, IJCAR 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, volume 4130 of *Lecture Notes in Computer*

Science, pages 303–317. Springer, 2006.

[21] Dedukti. http://dedukti.gforge.inria.fr/. Accessed: 2017-06-09.

[22] David Déharbe, Pascal Fontaine, Stephan Merz, and Bruno Woltzenlogel Paleo. Exploiting symmetry in SMT problems. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 222–236. Springer, 2011.

[23] Mike Gordon. From LCF to HOL: a short history. In Gordon D. Plotkin, Colin Stirling, and Mads Tofte, editors, *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 169–186. The MIT Press, 2000.

[24] Florian Haftmann and Tobias Nipkow. Code generation via higher-order rewrite systems. In Matthias Blume, Naoki Kobayashi, and Germán Vidal, editors, *Functional and Logic Programming, 10th International Symposium, FLOPS 2010, Sendai, Japan, April 19-21, 2010. Proceedings*, volume 6009 of *Lecture Notes in Computer Science*, pages 103–117. Springer, 2010.

[25] Robert Harper, Furio Honsell, and Gordon D. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, 1993.

[26] Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Trimming while checking clausal proofs. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 181–188. IEEE, 2013.

[27] Marijn Heule, Warren A. Hunt Jr., and Nathan Wetzler. Verifying refutations with extended resolution. In Bonacina [11], pages 345–359.

[28] Matti Järvisalo, Marijn Heule, and Armin Biere. Inprocessing rules. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *Automated Reasoning - 6th International Joint Conference, IJCAR 2012, Manchester, UK, June 26-29, 2012. Proceedings*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2012.

[29] Ranjit Jhala and Kenneth L. McMillan. A practical and complete approach to predicate refinement. In *"Tools and Algorithms for the Construction and Analysis of Systems (TACAS)"*, volume 3920 of *LNCS*, pages 459–473. Springer, 2006.

[30] Cezary Kaliszyk and Josef Urban. Proch: Proof reconstruction for HOL light. In Bonacina [11], pages 267–274.

[31] Guy Katz, Clark W. Barrett, Cesare Tinelli, Andrew Reynolds, and Liana Hadarean. Lazy proofs for DPLL(T)-based SMT solvers. In Ruzica Piskac and Muralidhar Talupur, editors, *2016 Formal Methods in Computer-Aided Design, FMCAD 2016, Mountain View, CA, USA, October 3-6, 2016*, pages 93–100. IEEE, 2016.

[32] Benjamin Kiesl and Martin Suda. A unifying principle for clause elimination in first-order logic. In *The 26th International Conference on Automated Deduction, Gothenburg 6–11 August*, 2017. To appear.

[33] K. Korovin. Instantiation-based automated reasoning: From theory to practice. In R. A. Schmidt, editor, *22nd International Conference on Automated Deduction CADE-22*, volume 5663 of *Lecture Notes in Computer Science*, pages 163–166. Springer, 2009.

[34] Konstantin Korovin. iProver An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer Berlin Heidelberg, 2008.

[35] Laura Kovács and Andrei Voronkov. Interpolation and symbol elimination. In Schmidt [44], pages 199–213.

[36] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35, 2013.

[37] R. Nieuwenhuis and A. Rubio. Paramodulation-based theorem proving. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume I, chapter 7, pages 371–443. Elsevier Science, 2001.

[38] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[39] Andreas Nonnengart and Christoph Weidenbach. Computing small clause normal forms. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 335–367. Elsevier

and MIT Press, 2001.

[40] 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, *The 8th International Workshop on the Implementation of Logics, IWIL 2010, Yogyakarta, Indonesia, October 9, 2011*, volume 2 of *EPiC Series in Computing*, pages 1–11. EasyChair, 2010.

[41] Giles Reger. Better proof output for Vampire. In Laura Kovács and Andrei Voronkov, editors, *Vampire@IJCAR 2016. Proceedings of the 3rd Vampire Workshop, Coimbra, Portugal, July 2, 2016.*, volume 44 of *EPiC Series in Computing*, pages 46–60. EasyChair, 2016.

[42] Giles Reger, Martin Suda, and Andrei Voronkov. New techniques in clausal form generation. In Christoph Benzmüller, Geoff Sutcliffe, and Raul Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41 of *EPiC Series in Computing*, pages 11–23. EasyChair, 2016.

[43] Giles Reger, Martin Suda, and Andrei Voronkov. Testing a saturation-based theorem prover: Experiences and challenges. In *11th International Conference on Tests and Proofs, 19-20 July 2017, Marburg, Germany*, 2017.

[44] Renate A. Schmidt, editor. *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*. Springer, 2009.

[45] Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*. Springer, 2013.

[46] Aaron Stump. Proof checking technology for satisfiability modulo theories. *Electr. Notes Theor. Comput. Sci.*, 228:121–133, 2009.

[47] Aaron Stump and Duckki Oe. Towards an SMT proof format. In *Proceedings of the Joint Workshops of the 6th International Workshop on Satisfiability Modulo Theories and 1st International Workshop on Bit-Precise Reasoning - SMT 08/BPR 08*, page 27. ACM Press, 2008.

[48] Aaron Stump, Duckki Oe, Andrew Reynolds, Liana Hadarean, and Cesare Tinelli. SMT proof checking using a logical framework. *Formal Methods in System Design*, 42(1):91–118, 2013.

[49] G. Sutcliffe. Semantic Derivation Verification. *International Journal on Artificial Intelligence Tools*, 15(6):1053–1070, 2006.

[50] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *J. Autom. Reasoning*, 43(4):337–362, 2009.

[51] Geoff Sutcliffe, Jürgen Zimmer, and Stephan Schulz. TSTP data-exchange formats for automated theorem proving tools. In W. Zhang and V. Sorge, editors, *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, volume 112 of *Frontiers in Artificial Intelligence and Applications*, page 201215. IOS Press, 2004.

[52] René Thiemann and Christian Sternagel. Certification of termination proofs using CeTA. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009.

[53] TPTP format for derivations. `http://www.cs.miami.edu/~tptp/TPTP/QuickGuide/Derivations.html`. Accessed: 2017-06-09.

[54] Steven Trac, Yury Puzis, and Geoff Sutcliffe. An interactive derivation viewer. *Electr. Notes Theor. Comput. Sci.*, 174(2):109–123, 2007.

[55] Andrei Voronkov. AVATAR: The architecture for first-order theorem provers. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 696–710. Springer International Publishing, 2014.

[56] C. Weidenbach. Combining superposition, sorts and splitting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume II, chapter 27, pages 1965–2013. Elsevier Science, 2001.

[57] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS version 3.5. In Schmidt [44], pages 140–145.

[58] Nathan Wetzler, Marijn Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In Carsten Sinz and Uwe Egly, editors, *Theory and Applications of Satisfiability Testing - SAT 2014 - 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 14-17, 2014. Proceedings*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, 2014.