Checkable Proofs for First-Order Theorem Proving

Giles Reger\textsuperscript{1} and Martin Suda\textsuperscript{2}

\textsuperscript{1} University of Manchester, Manchester, UK
\textsuperscript{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.

\textsuperscript{*}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 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]. 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 guarantee a trusted system will be able to reprove every sound inference within the given resource limits,

---

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.
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 that 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 DRAT [15, 16]. While it may be too optimistic to assume that an analogous sufficiently versatile small

³In 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.

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

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

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]. 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].
References


Checkable Proofs for First-Order Theorem Proving


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