The TESC Proof Format for First-Order ATPs (Extended Abstract)

Seulkee Baek

Department of Philosophy, Carnegie Mellon University

27th June, 2020

Abstract

This paper presents TESC, a low-level proof format for first-order ATPs, and its associated toolchain. Every part of a TESC proof is machine-checkable, including Skolemization. TESC can also be flexibly extended to accommodate a variety of first-order theories. The TESC toolchain compiles ATP-generated TSTP solutions to TESC proofs and verifies them against TPTP problems. The current version of TESC toolchain can compile solutions in first-order logic with equality generated by Vampire and E. In a test using randomly selected TPTP problems, it successfully compiled 95% and 93% of solutions generated by Vampire and E, respectively.

1 Background

The lack of a standard machine-checkable proof format is a long-standing issue for first-order automated theorem provers (ATPs). Although many ATP systems support output in the TSTP format [11], TSTP solutions cannot (in general) be mechanically checked as the format lacks a semantic specification. The absence of a standard format hampers ATP technologies in a variety of ways, including compromised confidence in answers given by ATPs, costly repetition of proof searches to verify previous results, and difficulty of interface with external tools. Considering the candidate formats that have been proposed [9, 4] and the feedback from the community [7], two desiderata for a practical format seem to stand out in particular:

- The format should be readily usable with existing ATPs. If significant modification is required to emit a proof in the format, the change is not only costly for developers to implement, but also likely to incur prohibitive overheads in performance. A good format will work with minimally modified solvers, preferably with off-the shelf versions.

- The format should allow every step of a proof to be explicitly recorded and mechanically checked. ATPs often introduce new formulas which, although not logically entailed by existing premises, preserve the satisfiability of their respective contexts (for convenience, we call such steps "satisfiability-preserving" for the remainder of the paper). Some important examples of such steps include Skolemization and predicate definition. Failing to support these steps substantially undermines the motivation for using a low-level proof format.

The two requirements may seem mutually exclusive, but the success of the DRAT [12] and LRAT [5] formats for SAT solvers proves otherwise. The key lesson from the SAT solving community is that we can pair a minimal proof format easily emitted by existing solvers (DRAT) with a detailed and archivable format (LRAT) by using a postprocessor (DRAT-trim) which compiles the former to the latter. This paper takes a similar approach for first-order ATPs.

The following sections are organized as follows: Section 2 discusses how this project relates to existing work. Section 3 introduces the new TESC format and describes its syntax and semantics. Section 4 gives an overview of the TESC toolchain. Section 5 presents the results of tests using problems from the TPTP library. Section 6 summarizes the implications and limitations of what was accomplished.

2 Related Works

The TESC format and toolchain builds on top of the existing TPTP [10] infrastructure, especially the TSTP [11] format supported by various ATPs.

As mentioned above, the DRAT [12] and LRAT [5] formats have largely solved for SAT solvers the problem which the TESC format and toolchain seeks to address for first-order ATPs, and were an important inspiration for this work.

In terms of practical objectives, the TESC toolchain is most closely related to GDV [9], as it also aims to provide a general verification method for TSTP

solutions. The main difference is that GDV checks TSTP solutions directly, while the TESC toolchain produces a separate, low-level proof object.

Foundational Proof Certificates (FPC) [4] is a flexible format for machine-checkable proofs that can accommodate various logical systems. The scope of the project is much more broad than just first-order ATPs, but it has been used to verify TSTP proofs produced by the E theorem prover [3].

SMT solvers have made more headway into checkable proofs than ATPs, most notably with the Logical Framework with Side Conditions (LFSC) [2] and the veriT proof format [1]. The latter in particular could also potentially serve as a proof format for ATPs, given its support for Skolemization.

A relatively recent report by Reger and Suda [7] gives a concise summary of the current situation regarding machine-checkable FOL proof formats, and also salient arguments on why such a format would be desirable.

3 The TESC Proof Format

Theory Extensible Sequent Calculus (TESC [1]) is a low-level, machine-checkable proof format for first-order ATPs. As the name suggests, the format is based on sequent calculus [6] and allows extensions for accommodating a variety of first-order theories.

Sequent calculus may seem an odd choice of a compilation target for ATP-produced solutions, since most major ATPs today are based on some variant of the superposition calculus. There are three main reasons behind this choice. First, the underlying calculus must be capable of expressing steps that involve general FOL formulas, since input problems for first-order ATPs are not always given in clausal form. Second, sequent calculus allows convenient management of contexts because every subgoal is associated with a specific context, which is especially helpful for handling satisfiability-preserving steps whose correctness can only be established against the entire context. Third, it is easy to implement and modify proof search in sequent calculus, which considerably simplifies proof compiler design.

Table 1 shows the axiom and inference rules of the TESC proof format. Some of its notable features are:

- Positive formulas correspond to left formulas of two-sided sequent calculus, and negative formulas to right.

[1] `https://github.com/skbaek/tesc`

Rule	Conditions
$\frac{\Gamma, X}{\Gamma} A(d)$	$\Phi \in \Gamma,\ A(d, \Phi, X)$
$\frac{\Gamma, X \qquad \Gamma, \Psi}{\Gamma} B$	$\Phi \in \Gamma,\ B(\Phi, X, \Psi)$
$\frac{\Gamma, X}{\Gamma} C(t)$	$\Phi \in \Gamma,\ fv(t) = 0,\ fp(t) \leq fp(\Gamma),\ C(t, \Phi, X)$
$\frac{\Gamma, X}{\Gamma} D$	$\Phi \in \Gamma,\ D(fp(\Gamma), \Phi, X)$
$\frac{\Gamma, -\phi \qquad \Gamma, +\phi}{\Gamma} F$	$fv(\phi) = 0,\ fp(\phi) \leq fp(\Gamma)$
$\frac{\Gamma, X}{\Gamma} S$	$\Phi \in \Gamma,\ S(\Phi, X)$
$\frac{\Gamma, \Phi}{\Gamma} T$	$T(\Gamma, \Phi)$
$\frac{\Gamma}{\Gamma, \Phi} W$	None
$\frac{}{\Gamma, +\phi, -\phi} X$	None

Table 1: TESC axiom and inference rules.

- Instead of left and right rules for each connective, there are A, B, C, D, S rules for decomposition of signed formulas, similar to the $\alpha, \beta, \gamma, \delta$ rules of Smullyan's analytic tableaux [8]. These rules require decomposition relations to hold between signed formulas, which are given in Figure 1.

- Variables have the form $\upsilon(k)$, where k is its De Brujin index. Quantifiers are written without variables.

- $\phi[k \mapsto t]$ is the result of replacing all variables in ϕ bound to the kth quantifier with term t. E.g., $p(\upsilon(0), \upsilon(1))[1 \mapsto c] = p(\upsilon(0), c)$.

- D rules use parameters of the form $\pi(k)$ that are syntactically distinct from variables.

- $fv(x)$ is the smallest De Brujin index not occurring in x, taking quantifiers into account. E.g., $fv(p(\upsilon(0), \upsilon(1))) = 2$ and $fv(\forall p(\upsilon(0), \upsilon(1))) = 1$. Similarly, $fp(x)$ is the smallest parameter index not occurring in x.

$$
\begin{array}{ll}
 & B(+\phi \vee \psi, +\phi, +\psi) \\
A(l, -\phi \vee \psi, -\phi) & B(-\phi \wedge \psi, -\phi, -\psi) \\
A(r, -\phi \vee \psi, -\psi) & B(+\phi \rightarrow \psi, -\phi, +\psi) \\
A(l, +\phi \wedge \psi, +\phi) & B(-\phi \leftrightarrow \psi, -\phi \rightarrow \psi, -\psi \rightarrow \phi) \\
A(r, +\phi \wedge \psi, +\psi) & C(t, +\forall \phi, +\phi[0 \mapsto t]) \\
A(l, -\phi \rightarrow \psi, +\phi) & C(t, -\exists \phi, -\phi[0 \mapsto t]) \\
A(r, -\phi \rightarrow \psi, -\psi) & D(k, +\exists \phi, +\phi[0 \mapsto \pi(k)]) \\
A(l, +\phi \leftrightarrow \psi, +\phi \rightarrow \psi) & D(k, -\forall \phi, -\phi[0 \mapsto \pi(k)]) \\
A(r, +\phi \leftrightarrow \psi, +\psi \rightarrow \phi) & S(+\neg \phi, -\phi) \\
 & S(-\neg \phi, +\phi)
\end{array}
$$

Figure 1: Decomposition relation between signed formulas.

- The T rule introduces new hypotheses that preserve satisfiability under the given background theory.

There are two main uses of the T rule. First, it is used to introduce formulas that were originally added by satisfiability-preserving steps (e.g. Skolemization) in the input TSTP solution. Second, it is used to introduce axioms of the background theory. For instance, it may be used to add $+(0 < 1)$ when working in the theory of integer arithmetic. This is why the condition $T(\Gamma, \Phi)$ is left unspecified in Figure 1; its details depend on the target theory, and you can obtain specific 'implementations' of TESC by determining what counts as valid applications of the T rule. The current implementation of TESC targets first-order logic with equality, and allows addition of (1) equality axioms, (2) choice axioms, and (3) fresh predicate definitions.

3.1 Skolemization

In TESC proofs, Skolemization is handled by introducing and applying choice axioms of the form

$$\forall \ldots \forall (\exists \phi \rightarrow \phi[0 \mapsto f(v(0), \ldots, v(k)), 1 \mapsto v(0), \ldots, k+1 \mapsto v(k)])$$

where k is the number of outermost universal quantifiers and f is a fresh Skolem function symbol. The condition $T(\Gamma, +\phi)$ for T rule holds if ϕ has

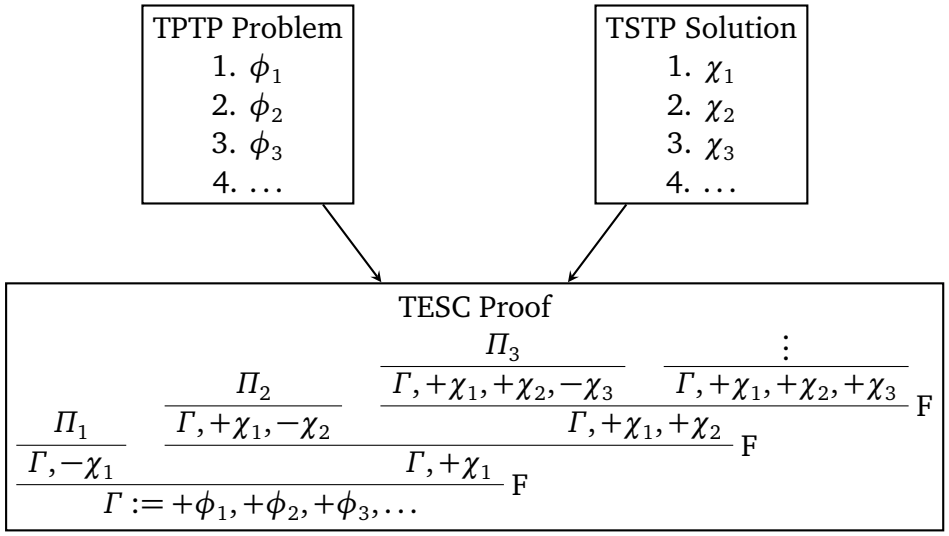

Figure 2: Proof compilation overview.

the above form and f does not occur in Γ. There is no other Skolemization-related axiom or inference rule, so all Skolemization steps have to be 'spelled out' in terms of choice axioms.

4 The TESC Toolchain

The TPTP-TSTP Compiler (TTC) accepts a TPTP problem and its TSTP solution as input and constructs a corresponding TESC proof. The current version of TTC can compile TSTP solutions in first-order logic with equality generated by Vampire and E.

Figure 2 shows a high-level overview of the proof compilation process. Γ is the initial sequent, obtained by adding a positive sign to each formula that occurs in the input TPTP problem. The formulas $\chi_1, \chi_2, \chi_3, \ldots$ are processed in the order they appear in the input TSTP solution: for each χ_k, The proof tree is split into a side branch (left) with $-\chi_k$ in the context, and a main branch (right) with $+\chi_k$. Intuitively, the sub branch is a subgoal in which we must prove χ_k using all previously existing hypotheses, and the main branch is a new main goal in which χ_k has become available by discharging the subgoal. TTC uses the information associated with the formula χ_k (obtained from the clause

containing χ_k in the TSTP solution) to construct its corresponding subproof Π_k, and proceeds to the next formula χ_{k+1}. This procedure is repeated until $+\bot$ is added to the main branch by processing the last clause of the TSTP solution.

Clearly, the most interesting part of proof compilation is the construction of subproofs $\Pi_1, \ldots, \Pi_k$, but the current construction methods are too varied and ad hoc to permit concise presentation. The follow-up full-length paper will discuss a number of most frequently used proof construction strategies, along with some simple examples.

In addition to TTC, the TESC toolchain also includes the TPTP-TESC Verifier (TTV) which accepts a TPTP problem and a TESC proof as input and verifies that the latter is a correct proof of the former.

5 Test Results

The problems used to test TTC and TTV were selected as follows: first, all TPTP problems with theorem/unsatisfiable status whose names are marked with '+' (FOF) or '—' (CNF) were exported using the TPTP2X tool. The export step failed for some problems due to memory constraints, yielding a total of 13137 TPTP files (this step was necessary because TTC and TTV only accept standalone TPTP files as input). Then the exported problems were randomly selected and solved with Vampire until 200 TSTP solutions were obtained. The same process was repeated for E, producing a different set of 200 solutions.[2] The test was performed using Vampire 4.4.0 and E 2.4 Sandakphu on a computer with Intel Core i7-7500U CPU (2.70GHz) and 8GB of RAM. The ATPs were run with default settings except for enabling TSTP solution production and setting the time limit to 60s.

Table 2 shows the test results using the randomly selected problems. Solution time is the time it takes for ATPs to produce TSTP solutions, compilation time is the time spent by TTC to compile them to TESC proofs, and verification time is the time TTV takes to verify TESC proofs. Values that pertain to all 200 problems are labeled as 'total', and values for problems whose solutions were successfully compiled are labeled as 'compilable'. For instance, the 'total' compilation time is longer than 'compilable' compilation time because the former includes the time TTC spent on solutions it ultimately failed to compile. Verification succeeded for all proofs, so there is no need to distinguish between

[2]The TSTP solutions used can be found in the `tests` directory of the GitHub repository.

ATP	Vampire	E
TPTP problems (total)	200	200
TPTP problems (compilable)	190	186
Solution time (total, seconds)	956.77 [3]	406.19
Solution time (compilable, seconds)	915.28	326.68
Compilation time (total, seconds)	805.84	4111.44
Compilation time (compilable, seconds)	611.50	1600.91
Verification time	59.21	29.06
TSTP solutions size (total, bytes)	5012	31140
TSTP Solutions size (compilable, bytes)	4668	29096
TESC Proofs size	63224	16892

Table 2: Test results for TTC & TTV.

total vs. verifiable proofs.

The compilation success rates are over 90% for both Vampire (95%) and E (93%). As expected, verifying TESC proofs is significantly faster than solving TPTP problems with ATPs: the results show speedup by factor of approximately 15.46 for Vampire, and 11.24 for E. The main drawbacks of TESC seem to be potentially long compilation times (E) and file size blowups (Vampire).

6 Conclusion

In this paper, we have shown that the TESC format and toolchain can be used in conjunction with first-order ATPs to produce machine-checkable proofs at acceptable costs. In particular, we could use two state-of-the-art ATP systems, Vampire and E, without code modifications or special settings that hamper their performance. The results seem to warrant further investigation involving other systems and theories to see whether TESC is viable as a standard proof format for first-order ATPs.

Acknowledgements

This work has been partially supported by AFOSR grant FA9550-18-1-0120.

[3] There were three problems that Vampire successfully solved under 60s initially, but timed out afterwards for unknown reasons. For these problems, we assume solution time = 60s.

References

[1] Haniel Barbosa, Jasmin Christian Blanchette, and Pascal Fontaine. Scalable fine-grained proofs for formula processing. In *International Conference on Automated Deduction*, pages 398–412. Springer, 2017.

[2] Frédéric Besson, Pascal Fontaine, and Laurent Théry. A flexible proof format for SMT: A proposal. 2011.

[3] Zakaria Chihani, Tomer Libal, and Giselle Reis. The proof certifier checkers. In *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 201–210. Springer, 2015.

[4] Zakaria Chihani, Dale Miller, and Fabien Renaud. Foundational proof certificates in first-order logic. In *International Conference on Automated Deduction*, pages 162–177. Springer, 2013.

[5] Luís Cruz-Filipe, Marijn JH Heule, Warren A Hunt, Matt Kaufmann, and Peter Schneider-Kamp. Efficient certified rat verification. In *International Conference on Automated Deduction*, pages 220–236. Springer, 2017.

[6] Gerhard Gentzen. Untersuchungen über das logische schließen. i. *Mathematische zeitschrift*, 39(1):176–210, 1935.

[7] Giles Reger and Martin Suda. Checkable proofs for first-order theorem proving. In *ARCADE@ CADE*, pages 55–63, 2017.

[8] Raymond M Smullyan. *First-order logic*. Courier Corporation, 1995.

[9] Geoff Sutcliffe. Semantic derivation verification: Techniques and implementation. *International Journal on Artificial Intelligence Tools*, 15(06):1053–1070, 2006.

[10] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 43(4):337, 2009.

[11] Geoff Sutcliffe, Jürgen Zimmer, and Stephan Schulz. TSTP data-exchange formats for automated theorem proving tools. *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, 112:201, 2004.

[12] Nathan Wetzler, Marijn JH Heule, and Warren A Hunt. Drat-trim: Efficient checking and trimming using expressive clausal proofs. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 422–429. Springer, 2014.