Ground Truth: Checking VAMPIRE Proofs via Satisfiability Modulo Theories

Michael Rawson[3] (✉), Andrei Voronkov[1,2], Johannes Schoisswohl[4], and Anja Petković Komel[4]

[1] EasyChair
[2] University of Manchester, United Kingdom
[3] University of Southampton, United Kingdom, michael@rawsons.uk
[4] TU Wien, Austria

Abstract. The VAMPIRE automated theorem prover is extended to output proofs in such a way that each inference is represented by a quantifier-free SMT instance. If every instance is unsatisfiable, the proof can be considered verified by an external SMT solver. This pragmatic form of proof checking places only a very light burden on the SMT solver, and can easily handle inferences that other systems may find difficult, such as theory inferences or extensive ground reasoning. The method is considerably easier to implement than proof formats based on small kernels and covers a greater variety of modern-day inferences.

Keywords: automated theorem proving · SMT · verified proofs

VAMPIRE [27] is an advanced automated theorem prover (ATP) for first-order logic, extended to reason in a polymorphic logic with *theories* such as real arithmetic and datatypes. For various reasons, not least VAMPIRE's complexity, users are sometimes unwilling to take VAMPIRE's proofs on trust, and so they *check* them to increase their confidence in the proof. User attitudes and requirements vary, and so too their methods of checking proofs. In particular, some users require that the proof be reconstructed in another format, such as the internal language and logic of an interactive theorem prover.

One way of checking proofs [42] is to use an external ATP[5] to replay proofs inference-by-inference, invoking the external ATP to re-prove each step in the proof. The external ATP could be an "old faithful" theorem prover like Otter [29], assumed free of bugs due to long service, a system so small that it is hard to doubt its correctness, like leanCoP [34], or an ATP that itself produces checkable proofs, like Metis [22]. The underlying assumption here is that each step in the proof is small enough that it can be easily re-proved by the external ATP, but in practice this is not always the case. This is exacerbated with modern inference systems that may involve many premises and implicit deductions [45, 46] or

[5] Or even the ATP itself: VAMPIRE developers sometimes use VAMPIRE to check VAMPIRE proof steps. The resulting ouroboros nevertheless catches bugs.

reasoning modulo theory [25, 37]. For reconstructing proofs the inference-by-inference method works but is labour-intensive in practice [18, 21].

Instead, a degenerate form of this approach is popular for proof automation in interactive theorem proving. Here, there are usually many more axioms than necessary to prove a given conjecture. A strong ATP like Vampire is used to find a proof in the presence of superfluous axioms. The proof is then discarded, recording only which axioms were necessary to find the proof. A proof is then found *again* using only these axioms and the interactive theorem prover's internal proof search. Isabelle's *Sledgehammer* [30] uses this to great effect. Here it is even more likely that the reconstructing ATP is unable to find a proof.

Finally, it is possible to modify ATPs to output proofs directly in the format desired for checking. This could be an interactive theorem prover's logic, an intermediate language designed for inter-operation [4], or a language specifically designed only for proof checking [6]. In principle this approach means that proof checking is straightforward and guaranteed to work. Unfortunately, doing this in practice is also labour-intensive. Typically, formats designed for checking proofs require spelling out simple inferences in great detail, do not accept ATP shorthands (e.g. treating the equality predicate as symmetric), and cannot reason modulo theory. This last limitation is particularly vexing as Vampire grows more capability for reasoning with theories, while small proof checking systems are unlikely to offer built-in theory reasoning. In some cases Vampire relies on the Z3 SMT solver [32] to perform inferences such as theory instantiation [38], so Vampire does not even "know" what theory reasoning was done.

We propose a new technique for proof checking (Section 1) that:

- is simple to implement
- is simple to check, using only an incremental SMT solver
- is fast, not requiring excess time in practice
- is relatively human-readable
- provides counter-examples for debugging proof steps

We also discuss its implementation in Vampire (Section 2), carry out experiments on a large subset of TPTP [43] and several SMT-LIB [36] logics (Section 3) and propose a way to completely reconstruct Vampire proofs in other systems using the technique (Section 5.1).

Preliminaries. We assume familiarity with automated and interactive theorem proving, the resolution calculus [5], and satisfiability modulo theories (SMT) [10]. Some exposure to the SMT-LIB input language may help [9]. Notation is standard, but we occasionally use a bar over variables $\bar{x}$ or terms $\bar{t}$ to indicate possibly-many variables or terms in that context.

1 Universal Inferences as Ground SMT Instances

Saturation theorem proving [5] is the gold standard for reasoning in first-order logic. Proofs are performed by contradiction. First, the input set of formulas is preprocessed and brought into clausal normal form, then inference rules are applied deriving new clauses until at some point the empty clause, witnessing contradiction, is derived. Completeness of state-of-the-art saturation theorem provers builds on *Herbrand's Theorem*, which might be stated as "a set of clauses is unsatisfiable iff there are ground instances which are ground unsatisfiable". This means that in order to find a proof we can first compute all ground instances of a clause, and then use a complete ground inference system on them.

Example 1. Consider the clauses $\forall x, y. P(y) \vee Q(x,a)$, $\forall z. \neg Q(b,z)$, and $\forall w. \neg P(w)$. We first compute the grounding

$$\begin{array}{cc} P(a) \vee Q(a,a) & \neg Q(b,a) \\ P(b) \vee Q(a,a) & \neg Q(b,b) \\ P(a) \vee Q(b,a) & \neg P(a) \\ P(b) \vee Q(b,a) & \neg P(b) \end{array}$$

then apply ground *binary resolution* twice to derive a contradiction:

$$\dfrac{\dfrac{P(b) \vee Q(b,a) \qquad \neg Q(b,a)}{P(b)} \qquad \neg P(b)}{\square}$$

The grounding of a set of clauses is typically very large, and becomes infinite in the presence of function symbols. This makes directly enumerating ground instances and searching for applicable inferences in the ground set impractical. Modern first-order theorem provers instead bypass grounding by working with universally-quantified clauses and using *unification* [5] to compute the most general instance of two clauses that makes an inference rule applicable.

Example 2. Consider the clauses from example 1. We apply non-ground binary resolution twice to derive a contradiction, without first computing a grounding.

$$\dfrac{\dfrac{\forall x, y. P(y) \vee Q(x,a) \qquad \forall z. \neg Q(b,z)}{\forall y. P(y)} \qquad \forall w. \neg P(w)}{\square}$$

The idea of using unification instead of explicit grounding is called *lifting* [5] and can be seen as compressing a (potentially infinite) set of inferences into a single inference. Lifting is a core principle of modern first-order theorem provers that sets them apart from Nelson-Oppen style SMT solvers.

1.1 Proposed Technique

Proof-checking such *lifted* inferences between quantified formulae poses a challenge for SMT solvers and automated theorem provers alike, especially when mixing theory reasoning with uninterpreted functions. This is where our method ties in. Suppose we wish to check that a particular inference occurring in a proof is sound. This means that we will need to show the conclusion $\forall \bar{x}.\ C[\bar{x}]$ follows from the premises $\forall \bar{y}.P_i[\bar{y}]$. We can show this by refutation. For that, we first negate and Skolemise the conclusion to obtain $\neg C[\bar{\sigma}]$. If there are instances of the premises $P_i[\bar{t_i}]$ that, taken together, refute $\neg C[\bar{\sigma}]$, we can conclude that the inference is valid. As our calculus uses lifting by unification such terms $\bar{t_i}$ can be directly found by using the unifying substitution θ computed in the rule application: first we instantiate the premises with θ, and then we replace all variables $\bar{x}$ that occured in the conclusion $C[\bar{x}]$ with the respective Skolem constants $\bar{\sigma}$.

Example 3. Take the first inference from example 2. The rule uses binary resolution unifying $Q(x, a)$ and $Q(b, z)$ computing the unifier $\theta = \{x \mapsto b, z \mapsto a\}$. In order to prove that the rule is sound we can instantiate the premises with the unifier, and instantiate the remaining free variable y with the Skolem constant σ in order to get the instantiated premises $P(\sigma) \vee Q(b, a)$ and $\neg Q(b, a)$. Negating the conjecture produces $\neg P(\sigma)$. This set of clauses is unsatisfiable iff the inference is sound.

As the resulting set of clauses is ground, it is most easily refuted by an SMT solver implementing a ground decision procedure.

Inhabited Sorts An edge case occurs when two variables are being unified. In this case the unifier will contain variables that are not present in the conclusion, thus $P_i[\bar{t_i}]\theta$ cannot be grounded by using the conclusion's Skolem constants. In such a case these variables can be instantiated with any ground term. If there are no constant symbols in the signature we can introduce a new Skolem symbol as sorts in first-order logic are assumed to be inhabited.

Example 4. Consider the last inference from example 2. Binary resolution is used to unify $P(y)$ and $P(w)$ computing the unifier $\theta = \{w \mapsto y\}$. If we apply θ to the premises we obtain the two non-ground clauses $P(y)$, $\neg P(y)$. In order to ground them we can choose any arbitrary ground term for y, say b, to obtain the ground premises $P(b)$, $\neg P(b)$.

1.2 Abilities and Limitations

As explained at the beginning of this section, most of Vampire's inferences *lift* ground inferences, which means that they can be covered by our method. This includes the core superposition calculus, simplification rules such as subsumption resolution, and most rules of the ALASCA calculus [25] for arithmetic reasoning. Further we support simple preprocessing rules like definition unfolding, AVATAR [45] inferences except definition introduction (including the dreaded `avatar sat refutation`), and axiomiatised theories; these rules do not instantiate variables in premises and can be implemented systematically. Our implementation in Vampire uses the same code for 58 such rules.

The key benefit of our approach is that the queries posed to the SMT solver are ground and relatively small compared to the size of the whole proof, which means that they are easily checked by state-of-the-art decision procedures. In contrast to checking every proof step without transforming it into the relevant ground instances, with our method the SMT solver can not only check the correctness of a rule application, but also give a counter-example if some proof-step is not correct. This can give insight into why a given inference is not sound.

However, there are some limitations to our new approach. Notably most preprocessing steps cannot be handled, as they are not sound inferences but transform the entire problem into an equisatisfiable form [20], relying on global properties of the whole problem (like freshness of introduced symbols) instead of performing a valid clause-to-clause inference. Proof steps in this category are usually found in the conversion to normal form, such as Skolemisation or definition introduction. We note that these steps also cause other proof checking systems trouble [19, 42]. This does also occasionally occur *during* proof search, such as when introducing definitions for AVATAR or clausifying higher-order terms on the fly [33].

Further unsupported inferences are those that cannot be grounded. Quantifier elimination rules used by the ALASCA [25,40] calculus, for example deriving $a \leq b$ from $\forall x : \mathbb{Q}.\ a \leq x \vee x \leq b$, where an instance for x cannot be found by unification. Still further limitations occurs with VAMPIRE's support for induction [26]. SMT solvers can check theory tautologies such as axiomatisations of arithmetic, but do not by design detect that a particular induction scheme is valid. Nascent support for induction [39] may later prove useful here.

2 Implementation

To demonstrate the method, we have a proof-of-concept implementation in VAMPIRE[6]. When activated with the flag `-p smtcheck`, VAMPIRE does not produce its usual proof format but instead an SMT-LIB script [9] suitable for processing by an incremental SMT solver such as Z3 [32] or `cvc5` [8]. If the SMT solver produces only `unsat` outputs, the proof can be considered verified by the SMT solver. Messages beginning "sorry" indicate that a step cannot yet be checked and should be further inspected or processed. Errors or `sat` responses (which cause an error) indicate a bug in VAMPIRE or the SMT solver.

2.1 Script Format

The script first declares symbols from the input problem (including sorts, if necessary) and provides a constant for each sort (Section 1.1). Then, for each inference in VAMPIRE's proof, the script:

1. Uses an SMT-LIB `(push)` command to isolate the context for this inference.
2. Declares new Skolem constants from the negated conjecture.
3. Asserts the instances of the premises required.
4. Asserts the negated conjecture.
5. Checks satisfiability, `(check-sat)`. The construct `(set-info :status unsat)` is used to produce an error and halt if any steps fail checking.

[6] https://github.com/vprover/vampire

6. Clears the assertions and declared Skolems with `(pop)`.

For inferences that are not yet or cannot be implemented, the `(echo ...)` command is used to inform the user that the inference must be taken on trust. No satisfiability check occurs in this case. Figure 1 shows a single inference.

```
; signature declared once at the start of the script
(declare-sort iota 0) ; the built-in default sort
(declare-fun |_identity| () iota) ; user-provided symbols
(declare-fun |_multiply| (iota iota) iota)

(push) ; inference starts here
(declare-const v0 iota) ; Skolem constant
(declare-const v1 iota) ; Skolem constant
(assert (or (=
  (|_multiply| (|_multiply| v0 v0) v1)
  (|_multiply| v0 (|_multiply| v0 v1))))) ; premise
(assert (or (= |_identity| (|_multiply| v0 v0)))) ; premise
(assert (not (=
  (|_multiply| v0 (|_multiply| v0 v1))
  (|_multiply| |_identity| v1)))) ; conclusion
(set-info :status unsat)
(check-sat)
(pop) ; inference finishes here
```

Fig. 1. A self-contained extract from a group theory proof, deriving $x\cdot(x\cdot y) = e\cdot y$ from $x\cdot(y\cdot z) = (x\cdot y)\cdot z$ and $x\cdot x = e$. Note the instantiation of both axioms with appropriate Skolem constants from the conclusion. User provided symbols are `|_quoted|`.

The user must presently trust that Vampire soundly instantiates and correctly transcribes clauses present in the script. This situation could be improved with greater support for definitions in SMT-LIB: `(define-fun ...)` would allow us to define a clause in terms of its variables and later instantiate it soundly by using the definition. However, our understanding is that this construct may be treated as a universally-quantified formula rather than a macro by the SMT solver, so we were unwilling to use it for this reason. A version of `(define-fun ...)` that *must* be treated as a macro would be very useful for us. An anonymous reviewer kindly suggested that using SMT solvers' support for annotations/*triggers* [31] would be another way to guide quantifier instantiation. We carried out a short experiment and can confirm that trigger-based instantiation works for this use-case. Like `define-fun`, however, triggers also rely on a well-behaved quantifier instantiation routine.

Vampire supports both SMT-LIB and TPTP input problems. As some of the features of TPTP are not supported in SMT-LIB, proofs of these TPTP problems cannot be checked in a straightforward manner. Proofs which make use of polymorphism [11] cannot be checked, although the recent SMT-LIB 2.7 standard and the upcoming SMT-LIB 3 should make this possible. If this were not the case, *monomorphisation* would be an option, creating one function for each instantiation of a polymorphic function. Arithmetic functions in TPTP [44] that are not supported by SMT-LIB (e.g. the floor function $\lfloor\cdot\rfloor : \mathbb{R} \to \mathbb{R}$) also cause trouble: some of these we "polyfill" by injecting a suitable translation, others simply cannot be proof checked currently.

2.2 Computing Instances

When printing a proof, we now need to compute the instances of the premises required for proof checking. It may at first appear that a substitution must be kept for each inference during proof search, but in fact this is overkill. Instead, we record only enough information to recover the substitution we need. Taking binary resolution as a representative example, we record which literals from the parent clauses were resolved upon. At proof-printing time, we then re-unify the literals to produce the required premise substitution. The overhead of this implementation technique is non-zero, but small. Vampire loses only 5 proofs out of 7,839 in a single-strategy configuration over TPTP when the `--proof_extra full` flag is set.

3 Experiments

The proof-of-concept implementation covers all Vampire's core search inferences, AVATAR, theory axioms, and the more common ALASCA inferences. Other inferences are skipped, reporting a warning but allowing proof checking to continue. To test this implementation, we ran Vampire:

1. In its default mode for 10 seconds over TPTP 9.0.0 problem set, covering the CNF, FOF, and TF0 languages with arithmetic but excluding known-satisfiable instances. `ARI502_1` was excluded as it uses TPTP rational arithmetic features that cannot be easily translated to SMT-LIB.
2. Using a fixed ALASCA configuration[7] for 10 seconds over the non-incremental SMT-LIB 2024 [36] problem set restricted to the `LRA`, `NRA`, and `UFLRA` logics.

In total this generated 13,159 SMT-LIB proof scripts. Generally these are short, but there are some outliers. The largest script came from the TPTP problem `SWV545-1.010`, weighing in at 331MB. The proof with the greatest number of checked inferences is TPTP's `NUM378+1.010.015` with 378,298.

We ran the Z3 [32] and `cvc5` [8] SMT solvers on the scripts. No unsoundness was found in Vampire's proofs, although a known ALASCA bug was rediscovered. After pulling the fix from upstream, all proofs checked successfully. Z3 is more permissive with identifiers than `cvc5`, which rejected a handful of TPTP proof scripts, but suitably escaping identifiers fixed the problem. Z3 could check

[7] `-alasca on -to qkbo -uwa alasca_main`

all instances, but `cvc5` times out on the final AVATAR refutation (an unsatisfiable SAT instance) for the TPTP problems `ALG034+1`, `ALG035+1`, and `ALG101+1`. Switching `cvc5`'s SAT solver [17] resolved this.

3.1 Quantified Proof Scripts

We have gone to some effort in order to produce *ground* SMT proof scripts. However, SMT solvers have good support for handling quantifiers [7], so this may not be needed in practice. In order to check this, we produced quantified scripts that simply include the universally-quantified premises and the negated conjecture for each step. We found that although the SMT solvers dispatched many proof steps quickly, unsurprisingly there are some VAMPIRE steps for which both Z3 and `cvc5` run out of time. Out of 4,194 quantified scripts produced from the SMT-LIB set of problems above, 85 exceeded a 10-second timeout when checked with Z3, and a further 19 required in excess of 1 second to check. For comparison, all ground scripts were checked in less than 0.2 seconds. After these extreme cases, the difference between the two falls rapidly, and the majority can be checked in less than 0.05 seconds.

4 Comparison with Other Methods

There are several approaches to proof-checking the output of ATPs. Verifying all proof steps with a small kernel of an interactive theorem prover (ITP), especially when an ITP has been around long enough to be tried and tested (like Coq [16], Agda [1] or Isabelle [23]), is currently deemed most trusted. Ideally the amount of proof search during the verification phase is reduced to a minimum, and outputting proofs directly in an ITP-checkable format can be done for the core calculus [12, 14, 15, 24, 35]. Some features of VAMPIRE that our approach struggles with are directly expressible in this manner, like polymorphism, induction and introducing definitions. Additionally, exporting proofs directly to ITPs contributes to interoperability [4]. However, in the presence of theories extending the core calculus, this becomes less feasible in practice. ITPs have some limited support to reason with theories — the tactic *omega* in Coq is the canonical example — but they tend to be inefficient compared to SMT solvers. It would also require carrying around superfluous (to VAMPIRE) information and produce proofs of unreasonable size in some cases.

A similar approach to ours is treating every (quantified, not ground) inference in a proof as a separate conjecture and re-deriving each individual step from the premises it refers to, using an external ATP. As we are dealing with non-ground problems, no decision procedures exist, which means ATPs and SMT solvers alike may time out on at least some valid proof steps, which we confirm occurs in practice in Section 3.1. In contrast, with our approach we are guaranteed that SMT solvers can prove sound proof steps, and can even obtain witnesses for unsoundness (i.e. counter-models) if an unsound proof step is detected. An example for this is the tool *ekstrakto* [19], which re-derives required steps from a

TSTP proof using external ATPs that produces proof in the $\lambda\Pi$-modulo calculus [12, 14, 15, 24]. An advantage of such an approach is that it can in principle work on most first-order provers without modification, but in practice success is limited, as it fails to handle advanced proof search features like AVATAR inferences or theories.

Proofs can also be checked by a separate verifier designed for a specific proof format. *Theory-Extensible Sequent Calculus (TESC)* [6] is a proof format for first-order ATPs. The calculus subsumes a large fraction of first-order formulas handled by VAMPIRE, including reasoning with AVATAR. Not all problems solved by VAMPIRE can be translated from the TSTP trace to TESC automatically, and the support for reasoning with theories is limited. There are three verifiers for TESC currently available, one of which is formalized in ITP Agda [1], enhancing the trustworthiness of the verifier. One can also verify the proof steps semantically as in GDV [42]: here not only are proof steps checked as in Section 3.1 with a trusted ATP, but additionally some of the proof structure is checked, for example that the parents of inferences exist and that a refutation ends in falsum.

Similar techniques are applied to SMT solvers. Proofs emitted in the recently proposed format Alethe [41] (currently cvc5 [8] and veriT [13] support this format), can be verified either by a separate verifier Carcara [2] (not formally checked); imported to Isabelle via the smt tactic [28], checking each step of the proof with Isabelle's internal kernel; or imported to Coq using the SMTCoq plugin [3].

5 Conclusion

We introduced a method for efficiently checking proofs produced from state-of-the-art saturation theorem provers. Any valid inference that lifts a ground inference is supported in principle. Checking each individual quantified proof step is reduced to a ground satisfiability check which can be performed efficiently by the decision procedures present in SMT solvers. We therefore support, out of the box, proofs that involve theory reasoning (like arithmetic), which pose a severe challenge to other ATP proof checking systems. In practice, even very lengthy, difficult VAMPIRE proofs are efficiently verified by third-party SMT solvers.

5.1 Fully Reconstructed Proofs

Users who wish to fully reconstruct proofs in their system of choice may object that we have not addressed this, only moved trust from VAMPIRE to the SMT solver. However, SMT proof technology is improving rapidly with several recent proposals, and it seems likely that in the near future this will be possible. Using our method, a proof could be fully reconstructed by gluing together the SMT unsatisfiability proof for each VAMPIRE inference together into one long continuous proof. The host system would still need to check that premise instances are instances of known clauses (Section 2.1), that the input has been reproduced faithfully, and deal with inferences that are not yet possible (Section 1.2), but the hard part of reconstructing VAMPIRE proofs is done.

Acknowledgments. We acknowledge funding from the ERC Consolidator Grant ARTIST 101002685, the COST Action CA20111 EuroProofNet, the TU Wien SecInt Doctoral College, the FWF SFB project SpyCoDe F8504, the WWTF ICT22-007 grant ForSmart, and the Amazon Research Award 2023 QuAT. We thank the anonymous reviewers for their suggestions.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. The Agda proof assistant. https://wiki.portal.chalmers.se/agda/ (2021)
2. Andreotti, B., Lachnitt, H., Barbosa, H.: Carcara: An efficient proof checker and elaborator for SMT proofs in the alethe format. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13993, pp. 367–386. Springer (2023). https://doi.org/10.1007/978-3-031-30823-9_19, https://doi.org/10.1007/978-3-031-30823-9_19
3. Armand, M., Faure, G., Grégoire, B., Keller, C., Théry, L., Werner, B.: A modular integration of sat/smt solvers to coq through proof witnesses. In: Jouannaud, J.P., Shao, Z. (eds.) Certified Programs and Proofs. pp. 135–150. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
4. Assaf, A., Burel, G., Cauderlier, R., Delahaye, D., Dowek, G., Dubois, C., Gilbert, F., Halmagrand, P., Hermant, O., Saillard, R.: Dedukti: a logical framework based on the $\lambda\Pi$-calculus modulo theory. CoRR **abs/2311.07185** (2023). https://doi.org/10.48550/ARXIV.2311.07185, https://doi.org/10.48550/arXiv.2311.07185
5. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, J.A., Voronkov, A. (eds.) Handbook of Automated Reasoning (in 2 volumes), pp. 19–99. Elsevier and MIT Press (2001). https://doi.org/10.1016/B978-044450813-3/50004-7, https://doi.org/10.1016/b978-044450813-3/50004-7
6. Baek, S.: A formally verified checker for first-order proofs. In: Cohen, L., Kaliszyk, C. (eds.) 12th International Conference on Interactive Theorem Proving, ITP 2021, June 29 to July 1, 2021, Rome, Italy (Virtual Conference). LIPIcs, vol. 193, pp. 6:1–6:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021). https://doi.org/10.4230/LIPICS.ITP.2021.6, https://doi.org/10.4230/LIPIcs.ITP.2021.6
7. Barbosa, H.: New techniques for instantiation and proof production in SMT solving. Ph.D. thesis, Université de Lorraine (2017)
8. Barbosa, H., Barrett, C.W., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.: cvc5: A versatile and industrial-strength SMT solver. In: Fisman, D., Rosu, G. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13243, pp. 415–442. Springer (2022). https://doi.org/10.1007/978-3-030-99524-9_24, https://doi.org/10.1007/978-3-030-99524-9_24

9. Barrett, C., Stump, A., Tinelli, C., et al.: The SMT-LIB standard: Version 2.0. In: Proceedings of the 8th international workshop on Satisfiability Modulo Theories (Edinburgh, UK). vol. 13, p. 14 (2010)
10. Barrett, C.W., Tinelli, C.: Satisfiability modulo theories. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 305–343. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_11, https://doi.org/10.1007/978-3-319-10575-8_11
11. Blanchette, J.C., Paskevich, A.: TFF1: the TPTP typed first-order form with rank-1 polymorphism. In: Bonacina, M.P. (ed.) Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7898, pp. 414–420. Springer (2013). https://doi.org/10.1007/978-3-642-38574-2_29, https://doi.org/10.1007/978-3-642-38574-2_29
12. Bonichon, R., Delahaye, D., Doligez, D.: Zenon: An extensible automated theorem prover producing checkable proofs. In: Dershowitz, N., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning. pp. 151–165. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
13. Bouton, T., Oliveira, D.C.B.D., Déharbe, D., Fontaine, P.: verit: An open, trustable and efficient smt-solver. In: Schmidt, R.A. (ed.) Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5663, pp. 151–156. Springer (2009). https://doi.org/10.1007/978-3-642-02959-2_12, https://doi.org/10.1007/978-3-642-02959-2_12
14. Burel, G., Bury, G., Cauderlier, R., Delahaye, D., Halmagrand, P., Hermant, O.: First-order automated reasoning with theories: When deduction modulo theory meets practice. Journal of Automated Reasoning **64**(6), 1001–1050 (2020). https://doi.org/10.1007/s10817-019-09533-z
15. Bury, G.: Integrating rewriting, tableau and superposition into SMT. Theses, Université Sorbonne Paris Cité (Jan 2019), https://theses.hal.science/tel-02612985
16. The Coq proof assistant, version 2021.02.2. https://coq.inria.fr/ (2021)
17. Fazekas, K., Niemetz, A., Preiner, M., Kirchweger, M., Szeider, S., Biere, A.: IPASIR-UP: user propagators for CDCL. In: Mahajan, M., Slivovsky, F. (eds.) 26th International Conference on Theory and Applications of Satisfiability Testing, SAT 2023, July 4-8, 2023, Alghero, Italy. LIPIcs, vol. 271, pp. 8:1–8:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.SAT.2023.8, https://doi.org/10.4230/LIPIcs.SAT.2023.8
18. Fleury, M., Blanchette, J.: Translation of proofs provided by external provers. Tech. rep., Tech. rep. Techniche Universität München (2014)
19. Haddad, M.Y.E.: Integrating Automated Theorem Provers in Proof Assistants. (Utiliser des démonstrateurs automatiques dans un assistant à la preuve). Ph.D. thesis, University of Paris-Saclay, France (2021), https://tel.archives-ouvertes.fr/tel-03387912
20. Heule, M., Kiesl, B.: The potential of interference-based proof systems. In: Reger, G., Traytel, D. (eds.) ARCADE 2017, 1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements, Gothenburg, Sweden, 6th August 2017. EPiC Series in Computing, vol. 51, pp. 51–54. EasyChair (2017). https://doi.org/10.29007/VR7N, https://doi.org/10.29007/vr7n
21. Hurd, J.: Integrating gandalf and HOL. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin-Mohring, C., Théry, L. (eds.) Theorem Proving in Higher Order

Logics, 12th International Conference, TPHOLs'99, Nice, France, September, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1690, pp. 311–322. Springer (1999). https://doi.org/10.1007/3-540-48256-3_21, https://doi.org/10.1007/3-540-48256-3_21

22. Hurd, J.: First-order proof tactics in higher-order logic theorem provers. Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports pp. 56–68 (2003)
23. Isabelle. https://isabelle.in.tum.de/ (2016)
24. Korovin, K.: iprover – an instantiation-based theorem prover for first-order logic (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) Automated Reasoning. pp. 292–298. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
25. Korovin, K., Kovács, L., Reger, G., Schoisswohl, J., Voronkov, A.: ALASCA: reasoning in quantified linear arithmetic. In: Sankaranarayanan, S., Sharygina, N. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 29th International Conference, TACAS 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Paris, France, April 22-27, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 13993, pp. 647–665. Springer (2023). https://doi.org/10.1007/978-3-031-30823-9_33, https://doi.org/10.1007/978-3-031-30823-9_33
26. Kovács, L., Hozzová, P., Hajdú, M., Voronkov, A.: Induction in saturation. In: Benzmüller, C., Heule, M.J.H., Schmidt, R.A. (eds.) Automated Reasoning - 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3-6, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14739, pp. 21–29. Springer (2024). https://doi.org/10.1007/978-3-031-63498-7_2, https://doi.org/10.1007/978-3-031-63498-7_2
27. Kovács, L., Voronkov, A.: First-order theorem proving and VAMPIRE. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8044, pp. 1–35. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_1, https://doi.org/10.1007/978-3-642-39799-8_1
28. Lachnitt, H., Fleury, M., Aniva, L., Reynolds, A., Barbosa, H., Nötzli, A., Barrett, C., Tinelli, C.: Isarare: Automatic verification of smt rewrites in isabelle/hol. In: Finkbeiner, B., Kovács, L. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 311–330. Springer Nature Switzerland, Cham (2024)
29. McCune, W.: OTTER 3.3 reference manual. CoRR **cs.SC/0310056** (2003), http://arxiv.org/abs/cs/0310056
30. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reason. **40**(1), 35–60 (2008). https://doi.org/10.1007/S10817-007-9085-Y, https://doi.org/10.1007/s10817-007-9085-y
31. Moskal, M.: Programming with triggers. In: Proceedings of the 7th International Workshop on Satisfiability Modulo Theories. pp. 20–29 (2009)
32. de Moura, L.M., Bjørner, N.S.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings. Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008). https://doi.org/10.1007/978-3-540-78800-3_24, https://doi.org/10.1007/978-3-540-78800-3_24

33. Nummelin, V., Bentkamp, A., Tourret, S., Vukmirovic, P.: Superposition with first-class booleans and inprocessing clausification. In: Platzer, A., Sutcliffe, G. (eds.) Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12699, pp. 378–395. Springer (2021). https://doi.org/10.1007/978-3-030-79876-5_22, https://doi.org/10.1007/978-3-030-79876-5_22

34. Otten, J., Bibel, W.: leancop: lean connection-based theorem proving. J. Symb. Comput. **36**(1-2), 139–161 (2003). https://doi.org/10.1016/S0747-7171(03)00037-3, https://doi.org/10.1016/S0747-7171(03)00037-3

35. Petković Komel, A., Rawson, M., Suda, M.: Case study: Verified VAMPIRE proofs in the $\lambda\Pi$-calculus modulo. CoRR **abs/2503.15541** (2025). https://doi.org/10.48550/ARXIV.2503.15541, https://doi.org/10.48550/arXiv.2503.15541

36. Preiner, M., Schurr, H., Barrett, C.W., Fontaine, P., Niemetz, A., Tinelli, C.: SMT-LIB release 2024 (non-incremental benchmarks) (version 2024.04.23). https://doi.org/10.5281/zenodo.11061097 (Apr 2024). https://doi.org/10.5281/ZENODO.11061097, https://doi.org/10.5281/zenodo.11061097, accessed on YYYY-MM-DD.

37. Reger, G., Bjørner, N.S., Suda, M., Voronkov, A.: AVATAR modulo theories. In: Benzmüller, C., Sutcliffe, G., Rojas, R. (eds.) GCAI 2016. 2nd Global Conference on Artificial Intelligence, September 19 - October 2, 2016, Berlin, Germany. EPiC Series in Computing, vol. 41, pp. 39–52. EasyChair (2016). https://doi.org/10.29007/K6TP, https://doi.org/10.29007/k6tp

38. Reger, G., Suda, M., Voronkov, A.: Unification with abstraction and theory instantiation in saturation-based reasoning. In: Beyer, D., Huisman, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10805, pp. 3–22. Springer (2018). https://doi.org/10.1007/978-3-319-89960-2_1, https://doi.org/10.1007/978-3-319-89960-2_1

39. Reynolds, A., Kuncak, V.: Induction for SMT solvers. In: D'Souza, D., Lal, A., Larsen, K.G. (eds.) Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings. Lecture Notes in Computer Science, vol. 8931, pp. 80–98. Springer (2015). https://doi.org/10.1007/978-3-662-46081-8_5, https://doi.org/10.1007/978-3-662-46081-8_5

40. Schoisswohl, J., Kovács, L., Korovin, K.: VIRAS: conflict-driven quantifier elimination for integer-real arithmetic. In: Bjørner, N.S., Heule, M., Voronkov, A. (eds.) LPAR 2024: Proceedings of 25th Conference on Logic for Programming, Artificial Intelligence and Reasoning, Port Louis, Mauritius, May 26-31, 2024. EPiC Series in Computing, vol. 100, pp. 147–164. EasyChair (2024). https://doi.org/10.29007/KG4V, https://doi.org/10.29007/kg4v

41. Schurr, H., Fleury, M., Barbosa, H., Fontaine, P.: Alethe: Towards a generic SMT proof format (extended abstract). In: Keller, C., Fleury, M. (eds.) Proceedings Seventh Workshop on Proof eXchange for Theorem Proving, PxTP 2021, Pittsburg, PA, USA, July 11, 2021. EPTCS, vol. 336, pp. 49–54 (2021). https://doi.org/10.4204/EPTCS.336.6, https://doi.org/10.4204/EPTCS.336.6

42. Sutcliffe, G.: Semantic derivation verification: Techniques and implementation. Int. J. Artif. Intell. Tools **15**(6), 1053–1070 (2006). https://doi.org/10.1142/S0218213006003119, https://doi.org/10.1142/S0218213006003119

43. Sutcliffe, G.: The TPTP problem library and associated infrastructure - from CNF to th0, TPTP v6.4.0. J. Autom. Reason. **59**(4), 483–502 (2017). https://doi.org/10.1007/S10817-017-9407-7, https://doi.org/10.1007/s10817-017-9407-7
44. Sutcliffe, G., Schulz, S., Claessen, K., Baumgartner, P.: The TPTP typed first-order form with arithmetic. In: Bjørner, N.S., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning - 18th International Conference, LPAR-18, Mérida, Venezuela, March 11-15, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7180, pp. 406–419. Springer (2012). https://doi.org/10.1007/978-3-642-28717-6_32, https://doi.org/10.1007/978-3-642-28717-6_32
45. Voronkov, A.: AVATAR: the architecture for first-order theorem provers. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8559, pp. 696–710. Springer (2014). https://doi.org/10.1007/978-3-319-08867-9_46, https://doi.org/10.1007/978-3-319-08867-9_46
46. Xu, Y., Liu, J., Chen, S., Zhong, X., He, X.: Contradiction separation based dynamic multi-clause synergized automated deduction. Inf. Sci. **462**, 93–113 (2018). https://doi.org/10.1016/J.INS.2018.04.086, https://doi.org/10.1016/j.ins.2018.04.086