

Challenges in SMT Proof Production and Checking for Arithmetic Reasoning

Haniel Barbosa¹

¹Universidade Federal de Minas Gerais, Belo Horizonte, Brazil

Abstract

Satisfiability Modulo Theories (SMT) solvers are widely used as backbones of formal methods tools in a variety of applications, often safety-critical ones. These tools rely on the solver's correctness to guarantee the validity of their results. Producing and checking proofs is the *de facto* standard to ensure the correctness of SMT solvers independently from implementations, which are often prohibitively difficult to verify. Arithmetic reasoning is one of the foundations of SMT reasoning, and therefore it is essential to support proof production and checking for it. In this extended abstract, to accompany a keynote talk at the 2023 SC-Square Workshop, we survey recent work (both the author's and from the literature) and discuss challenges on the production and checking of proofs from SMT solvers for arithmetic reasoning.

Keywords

Satisfiability Modulo Theories, Symbolic Computation, Proof Production, Proof Checking

1. Introduction

State-of-the-art SMT solvers, in order to optimize performance, generally have large and complex codebases written in system languages. This makes it hard to guarantee that the solver results are not compromised by implementation issues, which can happen despite the best efforts of developers. To increase trust, a possible solution is to formally verify or to qualify the solvers. However, these approaches are costly and once accomplished tend to “freeze” the systems, since changes require a new verification or qualification process. Moreover, the sheer complexity of the task frequently leads to compromises, with systems being less performant than the state of the art, thus hindering their usability.

An alternative is to make confidence in the results independent from the implementation via machine-checkable certificates of the correctness of these results. For SMT solvers, certificates are *models*, for satisfiable results, and *proofs* for unsatisfiable results. While satisfiable results generally denote a desired condition is not valid, unsatisfiable results ensure it is, which significantly increases the importance of proofs. But while model production is well established in state-of-the-art SMT solving (although in the presence of, for example, transcendental functions, there are challenges [1]), producing proofs is not.

8th International Workshop on Satisfiability Checking and Symbolic Computation, July 28, 2023, Tromsø, Norway, Collocated with ISSAC 2023


✉ hbarbosa@dcc.ufmg.br (H. Barbosa)

🌐 <http://hanielbarbosa.com/> (H. Barbosa)

🆔 0000-0003-0188-2300 (H. Barbosa)



© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

The main challenge for *producing* proofs is to justify the combination of heterogeneous, theory-specific algorithms used by solvers to derive unsatisfiability, while keeping the solver performant and providing enough details to allow *scalable* proof checking, i.e., checking that is fundamentally simpler than solving. We have tackled this issue in previous work [2, 3] and nowadays the cvc5 SMT solver [4] is a state-of-the-art solver that can effectively produce proofs for significant parts of their reasoning, without sacrificing performance too much.

Once proofs are produced, the challenge is to *check* they are correct. Proof checkers are given the proofs produced by the solvers and validate whether the steps in the proof are correct with relation to the proof calculus of which the proof is an instance. To maximize trustworthiness, the proof checker should be small and simple (ideally, even formally verified). Alternatively, the proof can be embedded in a highly trusted system such as a skeptical interactive theorem prover. The SMT community is increasingly embracing this approach, with proof production becoming a major focus in recent years [3, 5].

When considering interactive theorem provers or formally verified checkers as the targets of SMT proofs, the difficulty is to embed in these systems the formal calculus representing the proofs, which requires proving this calculus correct with relation to the logic of these systems. This task is already complex for the overall reasoning performed by SMT solvers [6, 7], but even more so for the formal calculi used when solving for some theories, such as strings [8] and, specially, non-linear arithmetic [9].

In this paper we discuss the state of the art and current challenges in proof production and proof checking for SMT solvers, in particular for their arithmetic reasoning.

2. Proof Production in SMT Solvers

Recently [3] we introduced a flexible proof-production architecture for SMT solvers, which is shown in Figure 1 (from [3]). We summarize this architecture below, but full details can be seen in [3].

The proof architecture is intertwined with the CDCL(\mathcal{T}) architecture [10], the most common architecture of SMT solvers. This architecture for producing proofs emphasizes modularity, given the highly modular design of SMT solvers. Proofs are produced and stored by each solving component, which also guarantee they follow the structure for proofs of that component, as described below. Once the proofs need to be combined, a post-processor does so guaranteeing that they are compatible.

The modules in Figure 1 are used to solve an input formula φ in the following manner: the *pre-processor* receives φ and simplifies it in a variety of ways into formulas ϕ_1, \dots, ϕ_n . For each ϕ_i , the pre-processor stores a proof $P : \varphi \rightarrow \phi_i$ justifying its derivation from φ . The *propositional engine* receives ϕ_1, \dots, ϕ_n and *clausifies* them into $C_1 \wedge \dots \wedge C_l$. A proof $P : \psi \rightarrow C_i$ is stored for each clause C_i . Note that several clauses may derive from each formula. Corresponding propositional clauses C_1^p, \dots, C_l^p , where first-order atoms are abstracted as Boolean variables, are sent to the SAT solver, which checks whether their conjunction is satisfiable. The propositional engine and the *theory engine* work in a loop where the SAT solver asserts literals, which witness the satisfiability of C_1^p, \dots, C_l^p , and the theory engine checks their satisfiability modulo a *combination of theories* T . If the literals are T -unsatisfiable, a lemma

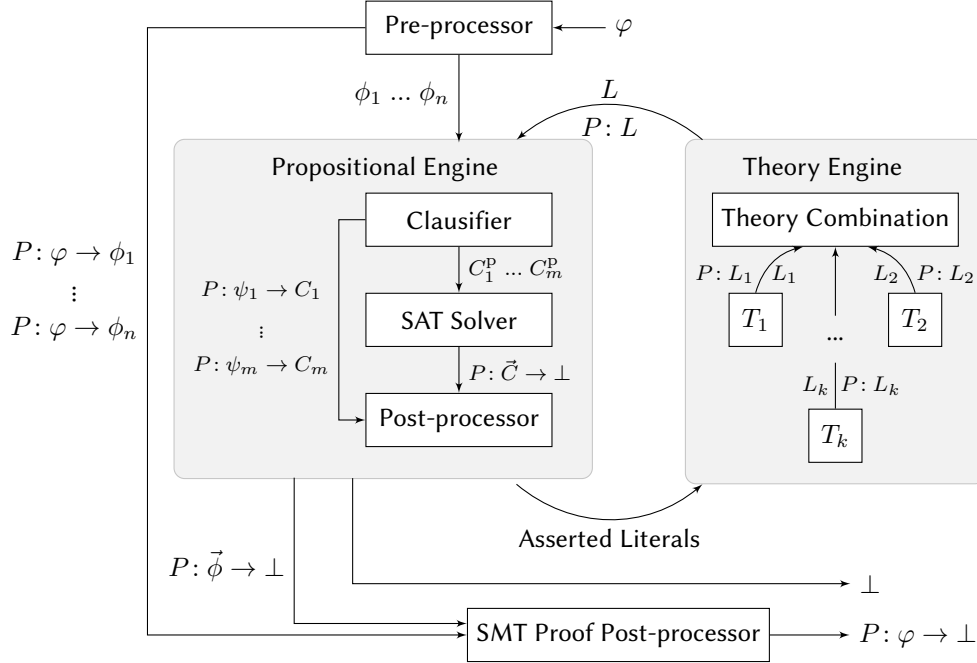


Figure 1: Flexible proof-production architecture for CDCL(\mathcal{T})-based SMT solvers. In the above, $\psi_i \in \{\vec{\phi}, \vec{L}\}$ for each i , with ψ_i not necessarily distinct from ψ_{i+1} .

L is sent to the propositional engine, together with a proof $P : L$ of its T -validity. This lemma will force the SAT solver to search for a different way to satisfy the classified formulas. If this is not possible, then all the clauses C_1, \dots, C_m generated until then are jointly unsatisfiable, and the SAT solver yields a proof $P : C_1 \wedge \dots \wedge C_m \rightarrow \perp$. Note that the proof is in terms of the first-order clauses, as are the derivation rules that conclude \perp from them. The propositional abstraction does not need to be represented in the proof.

The post-processor of the propositional engine connects the SAT assumptions with the classifier proofs, building a proof $P : \phi_1 \wedge \dots \wedge \phi_n \rightarrow \perp$. Since theory lemmas are T -valid, the resulting proof only has preprocessed formulas as assumptions. The final proof is built by the SMT solver's post-processor combining this proof with the preprocessing proofs $P : \varphi \rightarrow \phi_i$. The resulting proof $P : \varphi \rightarrow \perp$ justifies the T -unsatisfiability of the input formula.

The arithmetic reasoning performed by the solver happens in the pre-processor, where *rewrite rules* are used to simplify arithmetic terms occurring in the input, and in the theory solvers, where the main arithmetic reasoning takes place. We will focus on the current state and challenges related to arithmetic reasoning in the theory solvers, where we discuss separately about linear (Section 2.1) and non-linear reasoning (Section 2.2), given the significant differences between producing and checking proofs for them. The arithmetic pre-processing reasoning is comparatively simple and less prominent, and the challenges are mostly related to the handling of numerous rewrite rules [11], although some particular pre-processing techniques, such as aggressive ITE elimination, can be quite challenging.

We also note that the majority of the arithmetic reasoning in SMT solvers takes place on

quantifier-free formulas, since solvers employ dedicated procedures for logic fragments without quantifiers that are supplemented by instantiation techniques [12] when quantifiers are present. Producing and checking proofs for quantifier instantiation is generally well understood and does not pose issues.

2.1. Linear Arithmetic Proofs

The workhorse of linear arithmetic solving in SMT solvers is an adapted version of the *simplex* algorithm [13], which is tailored for *incremental* reasoning by enabling fast backtracking as well as efficient *theory propagation*¹. We first discuss reasoning only for reals and afterwards the challenges involving integers.

The simplex algorithm will determine the satisfiability of a conjunction of theory atoms $\bigwedge_i \sum_j c_{i,j} X_j \bowtie_j c_{i,0}$ for real constants $c_{i,j}$, real variables X_j , and relations $\bowtie_i \in \{\leq, <\}$. The Farkas lemma shows that if this conjunction is unsatisfiable, then there exist *Farkas coefficients* $s_i > 0$ such that $\sum_i s_i (\sum_j c_{i,j} X_j) = 0$ and $\sum_i s_i c_{i,0} \bowtie 0$, where \bowtie is \geq when any \bowtie_j are strict or $>$ otherwise. Moreover, the simplex algorithm can be instrumented to compute these coefficients for unsatisfiable queries, with minimal overhead. Thus, solvers can compute Farkas coefficients during solving and provide them when proofs for the unsatisfiability of a given conjunction of inequalities is requested, which is the case to state the validity of the theory lemma produced by the solver, which corresponds to the negation of this conjunction of literals. A proof checker then can use the coefficients to reduce the respective linear combination to false and thus prove the validity of its negation, the theory lemma. In the Alethe proof format for SMT solvers [14], the proof rule for this justification has its semantics defined via the algorithm to reduce the linear combination to false [15, Sec 5.4, Rule 9].

In *cvc5*, we have instrumented the solver to provide real linear arithmetic proofs in a more fine-grained manner. This is motivated to facilitate the algorithm that must be applied during proof checking to justify the reduction of the linear combination to false. We have instrumented *cvc5* to leverage the Farkas lemma by allowing bounds $\sum_j c_{i,j} X_j \bowtie_i c_{i,0}$ to be *scaled* by constants and also *summed* (the comparator for the sum is strict if and only any summand’s comparator is). The Farkas coefficients give a linear combination of bounds that can be shown to be equivalent to false via simple theory rewriting rather than a more complex algorithm, as above, which suffices to show the unsatisfiability of the conjunction. For example, $x < -1$ and $-x \leq 1$ sum to $x - x < 1 - 1$, which rewrites to false.

Some SMT solvers, such as Z3 [16], do not provide the coefficients needed to efficiently check the validity of theory lemmas produced by the linear real arithmetic solver. This puts a considerable burden in proof checkers for Z3 proofs involving this theory. For example, Schurr et al. [17] have shown that when checking SMT proofs for linear real arithmetic, without the coefficients, in the interactive theorem prover Isabelle/HOL (which uses the *linarith* tactic to do so, via the Sledgehammer tool [7]), most of the time is spent searching for the coefficients, which can lead to proof checking failures due to incompleteness of the procedure as well as to

¹Theory propagation, i.e., for a theory solver to eagerly communicate to the SAT solver that a given theory literal is valid in the current context of asserted literals, is not shown in Figure 1. This is because theory propagation is handled in the same manner as theory lemmas for producing proofs: when a justification for the literal being propagated is needed, the respective theory solver produces a theory lemma for it, together with a proof.

running out of resources.

2.1.1. Challenges with Integer Reasoning

Integer reasoning in SMT solvers also generally leverages the simplex method, but requires branching and bounding on integer variables before it can be applied with the assumption that the integer variables are real ones. For example [15, Sec. 4.3], if x is an integer, to reason with the inequality $2x < 3$ we can infer $x \leq 1$. This can also be justified with Farkas reasoning, showing the validity of the clause $\neg(x \leq 1) \vee 2x < 3$ with coefficients 1 and $1/2$. Bound tightening requires further proof support. It suffices to add proof rules for tightening strict and loose bounds. From a strict bound on integer term i : i.e., $i < c$, one can deduce that i is at most the greatest integer less than c . From $i \leq c$, one deduces $i \leq \lfloor c \rfloor$.

Instrumenting solvers to produce the justification for the integer reasoning they provide can still be challenging, however. For example, the veriT solver often produces no justifications for the integer reasoning it applies, and cvc5 also fails to do so in some cases [18]. Similarly to the challenge described for Z3 above, it falls to proof checkers to handle these coarse-grained proof steps. For example, in SMTCoq [6], a tool for reconstructing SMT proofs within the interactive theorem prover Coq, arithmetic reasoning is handled via the micromomega tactic², which provides a procedure complete for linear integer and real arithmetic but incomplete for non-linear arithmetic. Therefore it is a full-fledged procedure that has higher complexity than what would be needed for checking fine-grained proof steps. In Isabelle/HOL, the linarith tactic, for automating arithmetic reasoning when search is needed, is more limited and can only find integer coefficients and always fails if strengthening is required.

In CARCARA [18], a proof checker for the Alethe format, we have mitigated these issues by *elaborating* coarse-grained proof steps into fine-grained ones. We use cvc5 to produce fine-grained proofs, with simple steps for bound tightening and Farkas with coefficients, for coarse-grained linear arithmetic steps. A case study is made with coarse steps produced by the veriT solver for integer reasoning, and the overwhelming majority of these steps can be successfully converted to fine-grained proofs via cvc5. This approach can be specially impactful when considering the integration with interactive theorem provers of solvers that produce coarse-grained linear arithmetic proofs, such as veriT for integer reasoning and Z3 in general.

2.2. Non-Linear Arithmetic Proofs

SMT solvers adopt different solutions to solve non-linear arithmetic problems. For example, the MathSAT solver uses *incremental linearization* [1], in a abstraction-refinement loop with the linear solver, to handle non-linear arithmetic. In Z3 and Yices [19], a complete procedure based on cylindrical algebraic decomposition (CAD) is employed. In cvc5, we use incremental linearization and cylindrical algebraic coverings [20], a variation of CAD better suited for SMT solving. They are combined so that the incomplete procedure can be supplemented by the complete one [21].

²<https://coq.inria.fr/refman/addendum/micromomega.html>

2.2.1. Proofs for Incremental Linearization

To produce proofs for the incremental linearization procedure it is necessary to capture with proof rules how lemmas are generated to refine wrong models produced by the linear solver. For example [21, Sec. 2], $x \cdot y > 0 \wedge x > 1 \wedge y < 0$ is handled by the linear solver with $x \cdot y$ being abstracted as a variable. A possible model found by the linear solver has $x \mapsto 2, y \mapsto -1$, and $x \cdot y \mapsto 1$. The refinement loop then generates the lemma $x > 0 \wedge y < 0 \rightarrow x \cdot y < 0$ to rule out this wrong solution. This lemma is an instance of this proof rule, which must be instantiated in a proof:

$$\frac{- \mid f_1 \dots f_k, m}{(f_1 \wedge \dots \wedge f_k) \rightarrow m \diamond 0}$$

where the “ $- \mid f_1 \dots f_k, m$ ” notation means that this rule takes no premises and has $f_1 \dots f_k$ and m as arguments, where the former are variables compared to zero (less, greater or not equal), and m is a monomial from these variables, with \diamond being the comparison (less or equal) that results from the signs of the variables. Moreover, all variables with even exponent in m should be given as not equal to zero while all variables with odd exponent in m should be given as less or greater than zero. Another challenge regarding incremental linearization is that these proof rules must be reflected in the proof checkers. For example, in our work-in-progress integration between `cvc5` and the Lean interactive theorem prover [22], proving the correctness of this rule and defining a reconstruction procedure for it requires 250 lines of Lean code, while borrowing heavily from Lean’s mathematical library, `Mathlib`.

Since the incremental linearization technique uses multiple lemma schemas in the refinement loop, all of them must be instrumented to produce proofs to capture the reasoning of this technique. While this is relatively simple for some lemmas, which can be proven via propositional and basic arithmetic rules rather than dedicated proof rules, some, such as the one above and the tangent plane lemma³, need involved proof rules. It is still work in progress in `cvc5` to produce proofs for all the lemma schemas used in the incremental linearization procedure, let alone to reflect them in proof checkers.

2.2.2. Proofs for Cylindrical Algebraic Coverings

Proofs for the infeasible subsets generated by cylindrical algebraic coverings are much more complex, even though they are more accessible than for regular CAD-based theory solvers [23]. To prove a conflict in regular CAD it is necessary to show that each considered candidate solution fails, and that the list of candidate solutions is indeed exhaustive to cover the whole real space. This argument is all but trivial to check. Proofs for cylindrical algebraic coverings on the other hand can be built in a constructive manner via rules that successively exclude parts of the search space, as well as compose these parts, providing a stratified argument as a tree-like proof.

In `cvc5` we have an initial proof calculus based on the above idea. The calculus consists of two rules for excluding an interval in some dimension: one based on an assertion and a

³See the `ARITH_MULT_TANGENT` rule in https://cvc5.github.io/docs/latest/proofs/proof_rules.html

partial assignment, and the other one based on a full covering of the next dimension. They are combined and essentially follow the computation of the solver (pruned from unneeded branches). Constructing the actual proof rule applications is complicated by the fact that the premise of a subtree, i.e., the description of the interval that is excluded, is only known when the subtree is closed. We thus generate proofs lazily to manage the construction of a tree-shaped proof and the issue of lazy premises.

The cylindrical algebraic covering proofs are still coarse-grained proofs, with their proof checking not being inherently simpler than solving. Therefore, currently only marginal gains can be expected from the `cvc5` proofs versus not providing any details at all. In particular, the second proof rule that lifts a covering to an interval in a lower dimension rests on a significant portion of CAD-related theory. The proper formalization of CAD theory and its foundational algorithms in interactive theorem provers is a significant challenge in this respect and subject of past and future research [24, 25].

As in the case of linear arithmetic proofs where solvers give no details, for non-linear arithmetic one can also rely on powerful tactics in interactive theorem provers to attempt to check these steps, until more fine-grained steps are not produced by the solvers. Recent work by Kosaian et al. [9] has formalized in Isabelle/HOL a complete procedure for non-linear real arithmetic⁴ that could help such an integration.

3. Summary

Much work has been done for instrumenting SMT solvers to produce proofs for arithmetic reasoning as well as to check these proofs in ad-hoc tools, verified checkers or in interactive theorem provers. However, there are still multiple challenges to be addressed, as shown in the previous sections, specially as SMT solvers are extended to employ more advanced reasoning techniques from the computer algebra community. Combining efforts from the satisfiability checking and the symbolic computation communities will be fundamental to better address these challenges and allow for more trustworthy SMT solvers.

References

- [1] A. Cimatti, A. Griggio, A. Irfan, M. Roveri, R. Sebastiani, Satisfiability Modulo Transcendental Functions via Incremental Linearization, in: L. de Moura (Ed.), Proc. Conference on Automated Deduction (CADE), volume 10395 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 95–113. doi:10.1007/978-3-319-63046-5.
- [2] H. Barbosa, J. C. Blanchette, M. Fleury, P. Fontaine, Scalable Fine-Grained Proofs for Formula Processing, *Journal of Automated Reasoning* 64 (2020) 485–510. doi:10.1007/s10817-018-09502-y.
- [3] H. Barbosa, A. Reynolds, G. Kremer, H. Lachnitt, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Viswanathan, S. Viteri, Y. Zohar, C. Tinelli, C. W. Barrett, Flexible Proof

⁴It is the first complete multivariate quantifier elimination algorithm formalized in Isabelle/HOL, whereas univariate or incomplete multivariate ones, which are considerably simpler, already existed.

- Production in an Industrial-Strength SMT Solver, in: J. Blanchette, L. Kovács, D. Patinson (Eds.), International Joint Conference on Automated Reasoning (IJCAR), volume 13385 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 15–35. doi:10.1007/978-3-031-10769-6_3.
- [4] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, Y. Zohar, cvc5: A Versatile and Industrial-Strength SMT Solver, in: D. Fisman, G. Rosu (Eds.), TACAS, Part I, volume 13243 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 415–442. doi:10.1007/978-3-030-99524-9_24.
- [5] J. Hoenicke, T. Schindler, A Simple Proof Format for SMT, in: D. Déharbe, A. E. J. Hyvärinen (Eds.), International Workshop on Satisfiability Modulo Theories (SMT), volume 3185 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 54–70. URL: <http://ceur-ws.org/Vol-3185/paper9527.pdf>.
- [6] B. Ekici, A. Mebsout, C. Tinelli, C. Keller, G. Katz, A. Reynolds, C. W. Barrett, SMTCoq: A Plug-In for Integrating SMT Solvers into Coq, in: R. Majumdar, V. Kuncak (Eds.), Computer Aided Verification (CAV), volume 10427 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 126–133. doi:10.1007/978-3-319-63390-9_7.
- [7] J. C. Blanchette, S. Böhme, L. C. Paulson, Extending Sledgehammer with SMT Solvers, *Journal of Automated Reasoning* 51 (2013) 109–128. doi:10.1007/s10817-013-9278-5.
- [8] S. Kan, A. W. Lin, P. Rümmer, M. Schrader, CertiStr: a certified string solver, in: A. Popescu, S. Zdancewic (Eds.), Certified Programs and Proofs (CPP), ACM, 2022, pp. 210–224. doi:10.1145/3497775.3503691.
- [9] K. Kosaian, Y. K. Tan, A. Platzer, A First Complete Algorithm for Real Quantifier Elimination in Isabelle/HOL, in: R. Krebbers, D. Traytel, B. Pientka, S. Zdancewic (Eds.), Certified Programs and Proofs (CPP), ACM, 2023, pp. 211–224. doi:10.1145/3573105.3575672.
- [10] R. Nieuwenhuis, A. Oliveras, C. Tinelli, Solving SAT and SAT Modulo Theories: From an Abstract Davis–Putnam–Logemann–Loveland Procedure to DPLL(T), *J. ACM* 53 (2006) 937–977. doi:10.1145/1217856.1217859.
- [11] A. Nötzli, H. Barbosa, A. Niemetz, M. Preiner, A. Reynolds, C. W. Barrett, C. Tinelli, Reconstructing Fine-Grained Proofs of Rewrites Using a Domain-Specific Language, in: A. Griggio, N. Rungta (Eds.), Formal Methods In Computer-Aided Design (FMCAD), IEEE, 2022, pp. 65–74. doi:10.34727/2022/isbn.978-3-85448-053-2_12.
- [12] A. Reynolds, H. Barbosa, P. Fontaine, Revisiting Enumerative Instantiation, in: D. Beyer, M. Huisman (Eds.), TACAS, Part II, volume 10806 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 112–131. doi:10.1007/978-3-319-89963-3_7.
- [13] B. Dutertre, L. de Moura, A Fast Linear-Arithmetic Solver for DPLL(T), in: T. Ball, R. B. Jones (Eds.), Computer Aided Verification (CAV), volume 4144 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2006, pp. 81–94. doi:10.1007/11817963_11.
- [14] H. Schurr, M. Fleury, H. Barbosa, P. Fontaine, Alethe: Towards a Generic SMT Proof Format (extended abstract), CoRR abs/2107.02354 (2021). URL: <https://arxiv.org/abs/2107.02354>. arXiv:2107.02354, in Workshop on Proof eXchange for Theorem Proving (PxTP).
- [15] The Alethe Proof Format: A Speculative Specification and Reference, <https://verit.loria.fr/documentation/alethe-spec.pdf>, Oct 2022.
- [16] L. M. de Moura, N. Bjørner, Z3: An Efficient SMT Solver, in: C. R. Ramakrishnan, J. Rehof

- (Eds.), TACAS, volume 4963 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 337–340. doi:10.1007/978-3-540-78800-3_24.
- [17] H. Schurr, M. Fleury, M. Desharnais, Reliable Reconstruction of Fine-grained Proofs in a Proof Assistant, in: A. Platzer, G. Sutcliffe (Eds.), Proc. Conference on Automated Deduction (CADE), volume 12699 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 450–467. doi:10.1007/978-3-030-79876-5_26.
- [18] B. Andreotti, H. Lachnitt, H. Barbosa, Carcara: An Efficient Proof Checker and Elaborator for SMT Proofs in the Alethe Format, in: S. Sankaranarayanan, N. Sharygina (Eds.), TACAS, Part I, volume 13993 of *Lecture Notes in Computer Science*, Springer, 2023, pp. 367–386. doi:10.1007/978-3-031-30823-9_19.
- [19] D. Jovanović, L. de Moura, Solving Non-linear Arithmetic, in: B. Gramlich, D. Miller, U. Sattler (Eds.), International Joint Conference on Automated Reasoning (IJCAR), volume 7364 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2012, pp. 339–354. URL: http://dx.doi.org/10.1007/978-3-642-31365-3_27. doi:10.1007/978-3-642-31365-3_27.
- [20] E. Ábrahám, J. H. Davenport, M. England, G. Kremer, Deciding the consistency of non-linear real arithmetic constraints with a conflict driven search using cylindrical algebraic coverings, *J. Log. Algebraic Methods Program.* 119 (2021) 100633. doi:10.1016/j.jlamp.2020.100633.
- [21] G. Kremer, A. Reynolds, C. W. Barrett, C. Tinelli, Cooperating Techniques for Solving Nonlinear Real Arithmetic in the cvc5 SMT Solver (System Description), in: J. Blanchette, L. Kovács, D. Pattinson (Eds.), International Joint Conference on Automated Reasoning (IJCAR), volume 13385 of *Lecture Notes in Computer Science*, Springer, 2022, pp. 95–105. doi:10.1007/978-3-031-10769-6_7.
- [22] L. de Moura, S. Ullrich, The Lean 4 Theorem Prover and Programming Language, in: A. Platzer, G. Sutcliffe (Eds.), Proc. Conference on Automated Deduction (CADE), volume 12699 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 625–635. doi:10.1007/978-3-030-79876-5_37.
- [23] E. Ábrahám, J. H. Davenport, M. England, G. Kremer, Proving UNSAT in SMT: the case of quantifier free non-linear real arithmetic, in: M. Suda, S. Winkler (Eds.), Proceedings of ARCADE 2021, 2021, pp. 1–5. URL: <https://arxiv.org/abs/2108.05320>.
- [24] A. Mahboubi, Implementing the cylindrical algebraic decomposition within the Coq system, *Mathematical Structures in Computer Science* 17 (2007) 99–127. doi:10.1017/S096012950600586X.
- [25] S. J. C. Joosten, R. Thiemann, A. Yamada, A Verified Implementation of Algebraic Numbers in Isabelle/HOL, *Journal of Automated Reasoning* 64 (2020) 363–389. doi:10.1007/s10817-018-09504-w.