From MBQI to Enumerative Instantiation and Back

Marek Dančo, Petra Hozzová and Mikoláš Janota

Czech Technical University in Prague, Prague, Czech Republic

Abstract

This work investigates the relation between model-based quantifier instantiation (MBQI) and enumerative instantiation (EI) in Satisfiability Modulo Theories (SMT). MBQI operates at the semantic level and guarantees to find a counterexample to a given a model. However, it may lead to weak instantiations. In contrast, EI strives for completeness by systematically enumerating terms at the syntactic level. However, such terms may not be counter-examples. Here we investigate the relation between the two techniques and report on our initial experiments of the proposed algorithm that combines the two.

Keywords

SMT, quantifiers, MBQI, enumerative instantiation

1. Introduction

In this paper we report on work in progress in the area of quantifier instantiation for the SMT theories of (non-)linear integer and real arithmetic combined with uninterpreted functions.[1] While checking the satisfiability of formulas in these theories with quantifiers ranges from computationally expensive to undecidable, there are many incomplete methods for quantifier instantiation that work reasonably well in practice. It has been demonstrated repeatedly that various quantifier instantiation techniques tend to be very complementary in performance [1, 2]. *E-matching* [3] is very powerful and often successful in verification benchmarks but is inherently incomplete and suffers from self capturing loops. In contrast, *enumerative instantiation* [2, 4] and *model-based quantifier instantiation* (MBQI) [5], have completeness guarantees in certain scenarios and have a semantic grounding.

In this paper we combine MBQI supported by relevant domains with enumerative instantiation. We implemented our algorithm and compared it to cvc5 [6] and Z3 [7].

2. Preliminaries

We assume a basic understanding of SMT [8]. In certain special cases, such as (linear) arithmetic, there exist decision procedures for theories with quantifiers [9, 10, 11, 12, 13]. In the general case, however, quantifiers lead to undecidability and SMT solvers rely on quantifier instantiation techniques.

We give a brief overview of MBQI. For simplicity of presentation, assume the input formula is given in the form

$$G \wedge \bigwedge_j \forall \bar{x}^j . \varphi_j \tag{1}$$

where G is ground, and the only free variables in each φ_j are $\bar{x}^j$. Let $\mathcal{M}$ be a model of G that can be expressed in the underlying theory, and let $\varphi_j^{\mathcal{M}}$ be the evaluation of φ_j in $\mathcal{M}$, meaning that each non-variable symbol in φ_j is replaced by its interpretation from $\mathcal{M}$. We then check the satisfiability of

$$\neg \varphi_j^{\mathcal{M}},$$

SMT 2025: 23rd International Workshop on Satisfiability Modulo Theories, August 10–11, 2025, Glasgow, UK

https://people.ciirc.cvut.cz/~janotmik (M. Janota)

0009-0008-3031-113X (M. Dančo); 0000-0003-0845-5811 (P. Hozzová); 0000-0003-3487-784X (M. Janota)

[1] We do not consider uninterpreted sorts.

which is quantifier-free and contains only the free variables $\bar{x}^j$ and interpreted functions and predicates. (We note this makes the satisfiablity check easier than checking (1), although for some theories it is still undecidable).

If the formulas $\neg\varphi_j^{\mathcal{M}}$ for all j are *unsatisfiable*, then the original formula (1) is true in $\mathcal{M}$ and hence it is satisfiable. Further, we know that $\mathcal{M}$ is a model of (1) and the algorithm terminates. If some $\neg\varphi_j^{\mathcal{M}}$ is *satisfiable*, then the values $\bar{v}$ of $\bar{x}^j$ give a *counterexample* to the model $\mathcal{M}$. We select ground terms $\bar{t}$ that have the values $\bar{v}$ in $\mathcal{M}$, instantiate φ_j with $\bar{t}$, and repeat the process with this instantiation, i.e., repeat with the following formula:

$$\varphi_j[\bar{x}^j \mapsto \bar{t}] \wedge G \wedge \bigwedge_j \forall \bar{x}^j . \varphi_j$$

3. Algorithm

The key observation driving our work is that in arithmetic theories, MBQI has an infinite set of theory constants, i.e. terms that denote value in a theory, that can possibly serve as a counter-example to the current model. Instantiating by such arbitrary constants introduces new terms at the ground level. On the other hand, reducing the number of new terms can have a significant positive impact on the performance [4]. To this end, we modify MBQI to search for counterexamples by enumerating ground terms already present in the formula. We construct a *relevant domain* to guide the instantiation process. If none of the terms in this domain serve as a counterexample, the algorithm proceeds to search for a theory constant.

Although the original MBQI paper [5] introduced an enumerative approach, MBQI implementations in practice resort to instantiation by arbitrary theory constants; disregarding the theory constants and uninterpreted function symbols occurring in the problem. This approach is suboptimal. For example, consider the following unsatisfiable problem from the SMT-LIB non-incremental UFLIA benchmark set,

$$\forall x.\ x > 0 \longrightarrow f(x) = -1000 * f(x-1),\ f(0) = 1,\ f(20) < 0$$

To reach a contradiction, the solver must consider the sequence of numbers from 0 to 20. However, both cvc5 and Z3, when running MBQI without E-matching, fail to conclude unsatisfiability.

Algorithm 1 shows the general outline, following the recipe for MBQI (Section 2). As customary, the algorithm assumes that the formula is split into a set of quantified subformulas ($\forall \bar{x}^j . \varphi_j$) and a ground part G, where φ_j are quantifier-free and contain only free variables in $\bar{x}^j$. Such form can be obtained by clausification [14], even though the formulas φ_j do not necessarily have to be clauses.

We note that we look for a new model of the ground part only after producing a counterexample for all quantified formulas for which there was one. The instantiation step and the calculation of relevant domains are explained further in Subsections 3.1 and 3.2, respectively.

We assume that constructing the formula $\varphi_j^{\mathcal{M}}$ is straightforward (line 9) as we work only with arithmetic sorts and thus each element from the model has a uniquely corresponding interpreted constant or function. This is strictly speaking not true because of the real domain, which is uncountable. However, in practice, solvers only produce "well-behaved models".

3.1. Finding a Counterexample

This section focuses on how a counterexample is chosen in Algorithm 1 (step 11). Consider a quantified formula $\forall x^j . \varphi_j[\bar{x}^j]$ and a model $\mathcal{M}$. The objective is to construct a counterexample from the relevant terms. There may be still too many of them, so we restrict those heuristically as follows.

For each variable x in $\bar{x}^j$, first determine a total preference ordering on the terms in its relevant domain $\mathcal{U}(T_{x,j})$ based on the current set of formulas. The ordering is a lexicographic ordering based on how frequently does a term appears in the formula, its depth, and in which iteration the term was added to the formula. We make the ordering total by appending a comparison based on the string

Algorithm 1: MBQI with Relevant Domain

Input: A formula $G \wedge \bigwedge_j \forall \bar{x}^j.\, \varphi_j$, where G is ground and each φ_j has only the free variables $\bar{x}^j$
Output: Satisfiability of the input formula

```
while true do
    foreach j and each variable x in x̄^j do
        Construct the relevant domain U(T_{x,j})                    // Section 3.2
    Find a model M of G
    if no such model exists then
        return unsatisfiable
    is-sat ← true
    foreach φ_j[x̄^j] do
        Fix all non-variable symbols according to M, obtaining φ_j^M[x̄^j]
        if ¬φ_j^M[x̄^j] is satisfiable then
            Using U(T_{x,j}), find ground terms t̄ s.t. ¬φ_j^M[t̄] is true   // Section 3.1
            G ← G ∧ φ_j[t̄]
            is-sat ← false
    if is-sat then
        return satisfiable
```

representation of the term. Let $\mathcal{P}(T_{x,j})$ denote the first third of the terms in $\mathcal{U}(T_{x,j})$ with respect to this ordering. We call these the *preferred terms*.

To look for a counterexample using the preferred terms, consider again the formula $\varphi_j^{\mathcal{M}}[\bar{x}^j]$, where all non-variable uninterpreted symbols in φ_j are interpreted by their interpretation in the model $\mathcal{M}$. Assume $\neg\varphi_j^{\mathcal{M}}[\bar{x}^j]$ is satisfiable. Then, construct the following formula.

$$\neg\varphi_j^{\mathcal{M}}[\bar{x}^j] \wedge \bigwedge_{x \in \bar{x}} \bigvee_{t \in \mathcal{P}(T_{x,j})} x = t^{\mathcal{M}} \tag{2}$$

As in MBQI, the formula looks for a counterexample to the model $\mathcal{M}$, but additionally, it restricts that the counterexample is equal to a vector of preferred terms (all this modulo what holds in the model $\mathcal{M}$). Such counterexample might not exist. Therefore, we proceed by gradually loosening the domains of variables from $\bar{x}^j$.

If the formula (2) is satisfiable, extract the equalities $x = t^{\mathcal{M}}$ that are satisfied in it and add the corresponding instantiation $\varphi_j[\bar{x} \mapsto \bar{s}]$ to the ground part, where each s is the least term such that $s^{\mathcal{M}} = t^{\mathcal{M}}$. In case it is unsatisfiable, one of the new clauses must be unsatisfiable. We look for that clause and extend the set of available terms to all terms in the relevant domain of the corresponding variable. We do this by taking an arbitrary such clause from an unsatisfiable core of (2). This clause corresponds to some variable x. Next, we permit to range over the whole set of relevant domain, not just the preferred terms. Let $\mathcal{C} = \{x\}$ and $\mathcal{NC} = \bar{x}^j \setminus \mathcal{C}$. Next, construct the following formula.

$$\neg\varphi_j^{\mathcal{M}}[\bar{x}^j] \wedge \bigwedge_{x \in \mathcal{NC}} \bigvee_{t \in \mathcal{P}(T_{x,j})} x = t^{\mathcal{M}} \wedge \bigwedge_{x \in \mathcal{C}} \bigvee_{t \in \mathcal{U}(T_{x,j})} x = t^{\mathcal{M}} \tag{3}$$

If formula (3) is satisfiable, we have a counterexample, as before. Otherwise, if (3) is unsatisfiable, continue by loosening the set of possible terms to the whole relevant domain for variables in the core, one by one. Effectively, moving variables from $\mathcal{NC}$ to $\mathcal{C}$ in (3). It this process does not yield a counterexample, i.e. if the relevant domain is too restrictive, we gradually drop the clauses altogether, effectively allowing the underlying SMT solver to pick a theory constant (numeral). This is done analogously as before by taking the core of (3) and removing variables from $\mathcal{C}$, for some x whose clause appears in the core.

The approach overall can be seen a mix of enumerative instantiation [2] and MBQI [5]. In the first phase, it looks for terms that are present in the current formula. If that fails it goes back to MBQI terms. This happens per variable; therefore, the approach is not a simple union of the two.

3.2. Relevant Domain

In arithmetic benchmarks, integers/reals are often used to model the set of some more specific objects, e.g. addresses on the heap. The objective of relevant domain is to identify such scenarios. For example, if we know that the function $f : \mathbb{Z} \to \mathbb{Z}$ operates on addresses, we want to instantiate $f(x)$ only with terms that also represent addresses, even though x ranges over all integers.

Therefore, for each quantified variable, we construct a set of candidate terms that are relevant for its instantiation. We call such a set the *relevant domain*. To compute these sets, we use the relevant domain approach based on [4, 5, 15]. In our approach, however, the inferred sets only serve as "known relevant options" for ground terms. Our approach is not complete — it is not always sufficient to consider only the terms in the relevant domain. In that case we disregard the relevant domain, allowing for instantiation with an arbitrary value.

We start by creating and initializing the following domains:

- $T_{x,j} = \emptyset$ for each formula φ_j and each x in $\bar{x}^j$, called the *relevant domain* for x,
- $T_f = \emptyset$ for each uninterpreted function or predicate symbol f,
- $T_{f,i} = \emptyset$ for each uninterpreted symbol f with arity $n \geq 1$ and each $1 \leq i \leq n$,
- $T_u = \{u\}$ for each ground term u occurring anywhere in the input formula.

Then we follow the rules shown below to merge these sets. We represent the merged set by a union containing the sets it was formed from, and we abuse to notation to refer to the union by any of its constituting sets. For example, the result of merging T_f and $T_{x,1}$ is $\{T_f, T_{x,1}\}$, and the result of a subsequent merge of T_f and T_a is $\{T_f, T_{x,1}, T_a\}$. To denote that T_1 and T_2 should be merged, we write $\texttt{merge}(T_1, T_2)$.

Although the sets $T_{x,j}$ are initially empty, after applying the merging rules exhaustively, they may become members of a union containing some sets of ground terms T_t. Let us define $\mathcal{U}(T_{x,j})$ as $\{t \mid t \text{ is a ground term} \wedge \exists U.\, T_t \in U \wedge T_{x,j} \in U\}$ – i.e., the set of ground terms t whose corresponding set T_t belongs to the same union as $T_{x,j}$. We consider all the terms of $\mathcal{U}(T_{x,j})$ to be relevant for instantiating the quantified variable x in the subformula φ_j. If $\mathcal{U}(T_{x,j})$ is empty, we add a default theory constant to it; in our case of arithmetic, this default is 0.

We now explain the merging rules. First, we define the function *tls* that assigns to each term[2] its *top level symbol*:

$$\mathit{tls}(t) = \begin{cases} u & \text{if } t \text{ is a ground term } u \\ x^j & \text{if } t \text{ is a quantified variable } x \text{ in subformula } j \\ g & \text{if } t \text{ is } g(t_1, \ldots, t_n), \text{ where } g \text{ is uninterpreted and } t \text{ is not ground} \\ \mathit{tls}(t_1) & \text{if } t \text{ is } t_1 * \cdots * t_n \text{ and } t \text{ is not ground} \\ \mathit{tls}(t_1) & \text{if } t \text{ is } t_1 + \cdots + t_n \text{ and } t \text{ is not ground} \\ \mathit{tls}(t_1) & \text{if } t \text{ is } t_1 - t_2 \text{ and } t \text{ is not ground} \end{cases}$$

Next, we define the function T^* that maps terms to their respective sets based on their top level symbols.

$$T^*(t) = \begin{cases} T_u & \text{if } \mathit{tls}(t) \text{ is a ground term } u \\ T_{x,j} & \text{if } \mathit{tls}(t) \text{ is a quantified variable } x^j \text{ in subformula } j \\ T_f & \text{if } \mathit{tls}(t) \text{ is an uninterpreted function (predicate) symbol } f \text{ and } t \text{ is not ground} \end{cases}$$

[2] In practice, SMT-LIB benchmarks typically do not include division operations, so we disregard such terms.

The rules merge the introduced sets based on where a term appears within the formula. Specifically, we distinguish between terms that occur as arguments of uninterpreted function symbols, arithmetic operations, or relational operators such as $\{=, \leq, <\}$. We apply these rules based on term occurrences in the whole formula $G \wedge \bigwedge_j \forall \bar{x}^j.\, \varphi_j$.

Term occurring in the formula	Operation(s)
$f(t_1, \ldots, t_n)$	$\texttt{merge}(T_{f,i}, T^*(t_i))$ for $1 \leq i \leq n$
$t_1 + \cdots + t_n$	$\texttt{merge}(T^*(t_1), T^*(t_i))$ for $2 \leq i \leq n$
$t_1 - t_2$	$\texttt{merge}(T^*(t_1), T^*(t_2))$
$t_1 * \cdots * t_n$	$\texttt{merge}(T^*(t_1), T^*(t_i))$ for $2 \leq i \leq n$
$t_1 = t_2$	$\texttt{merge}(T^*(t_1), T^*(t_2))$
$t_1 \leq t_2$	$\texttt{merge}(T^*(t_1), T^*(t_2))$
$t_1 < t_2$	$\texttt{merge}(T^*(t_1), T^*(t_2))$

As an example, consider the following SMT formula.

$$\underbrace{(f(3,a) \geq 4 + g(b))}_{G} \wedge \underbrace{(\forall x, y.\, f(x,y) < x + g(y))}_{\forall x,y.\varphi_1} \wedge \underbrace{(\forall x.\, g(x) = g(x+2))}_{\forall x.\varphi_2}$$

Because x and $g(y)$ are summed in φ_1, we apply the operation $\texttt{merge}(T_{x,1}, T_g)$. Similarly, because x is also the first argument of f, we apply $\texttt{merge}(T_{f,1}, T_{x,1})$. Using these two rules we get the union $\{T_{x,1}, T_{f,1}, T_g\}$. After performing all the possible merges, we get two disjoint unions of sets.

$$\{T_{x,1}, T_{f,1}, T_g, T_f, T_4, T_3, T_{(4+g(b))}\}$$
$$\{T_{y,1}, T_{f,2}, T_{g,1}, T_a, T_b, T_{x,2}, T_2\}$$

Since the first set contains both $T_{x,1}$ as well as T_4, T_3, and $T_{(4+g(b))}$, we get that $\mathcal{U}(T_{x,1}) = \{3,\, 4,\, 4 + g(b)\}$, and similarly $\mathcal{U}(T_{y,1}) = \mathcal{U}(T_{x,2}) = \{2,\, a,\, b\}$.

Note that we repeat the merging after each instantiation step (line 12 of Algorithm 1). This is because an instantiation may give us more information about the relevant domains. For example, instantiating $x = 2$ in the last subformula yields a ground formula $g(2) = g(4)$. After adding it to the ground part, we apply $\texttt{merge}(T_{g,1}, T_4)$ and merge the two unions of sets into one.

4. Preliminary Experiments

We initially implemented the algorithm described in Section 3 as a standalone Python module using PySMT [16], with Z3 [7] as its backend SMT solver for quantifier-free queries.

Benchmarks. For preliminary evaluation, we used a subset of benchmarks from [17] based on problems from the International Mathematical Olympiad, encoded in SMT-LIB [18] using UFNRA logic. We chose 22 problems from this set for which we manually confirmed they are unsatisfiable.

Experimental setup. We compared the performance of our implementation with the SMT solvers cvc5 [6], Z3 [7], and Interpol [19], and with the first-order theorem prover Vampire [20]. For cvc5, we used 5 instances with the following configurations, later denoted as cvc5-{1,2,3,4,5}:

1. default mode
2. `--enum-inst`
3. `--mbqi`
4. `--no-e-matching --enum-inst`
5. `--simplification=none --enum-inst`

We ran all the other solvers in default mode. The experiments were performed on machines with two AMD EPYC 7513 32-Core processors and with 514 GiB RAM. We used the time limit of 600 seconds per problem for each solver.

Table 1
Numbers of solved problems out of total 22.

This work	cvc5-1	cvc5-2	cvc5-3	cvc5-4	cvc5-5	Interpol	Vampire	Z3
9	5	10	6	13	10	5	15	10

Results. We display the numbers of solved problems in Table 1. Overall, Vampire solves most problems. Our prototype is more-or-less competitive with the SMT solvers, solving more problems than cvc5-1, cvc5-3, and Interpol, one less problem than cvc5-2, cvc5-5, and Z3, and four problems less than cvc5-4. Interestingly, our prototype also solves one problem which none of the other SMT solvers solve – the only other tool to solve it is Vampire. We therefore conclude that our preliminary results show promise and we plan to further develop our approach.

5. Conclusion and Future Work

In this paper we gave preliminary results on MBQI combined with a custom relevant domain approach. The approach resembles enumerative instantiation [4, 2] and MBQI [5], as it focuses on the ground terms currently present in the formula, and only abandons those if no counterexample (meaning a good instantiation) exists. The approach can be naturally generalized to other non-arithmetic sorts.

The current prototype shows promise since it allows solving small but difficult problems. We plan to investigate improvements to the implementation that would enable solving also larger problem instances and integrating it into an existing solver.

Acknowledgements

This work is supported by the Czech MEYS under the ERC CZ project no. LL1902 *POSTMAN* and by the European Union under the project *ROBOPROX* (reg. no. CZ.02.01.01/00/22_008/0004590) and by the *RICAIP* project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857306.

Declaration on Generative AI

The authors used ChatGPT and Grammarly for grammar and spell-checking. After using these tools, the authors reviewed and edited content as needed and take full responsibility for the publication's content.

References

[1] L. Kondylidou, A. Reynolds, J. Blanchette, Augmenting model-based instantiation with fast enumeration, in: A. Gurfinkel, M. Heule (Eds.), Tools and Algorithms for the Construction and Analysis of Systems - 31st International Conference, TACAS 2025, Held as Part of the International Joint Conferences on Theory and Practice of Software, ETAPS 2025, Hamilton, ON, Canada, May 3-8, 2025, Proceedings, Part I, volume 15696 of *Lecture Notes in Computer Science*, Springer, 2025, pp. 85–103. doi:10.1007/978-3-031-90643-5_5.

[2] M. Janota, H. Barbosa, P. Fontaine, A. Reynolds, Fair and adventurous enumeration of quantifier instantiations, in: Formal Methods in Computer-Aided Design, IEEE, 2021, pp. 256–260. doi:10.34727/2021/ISBN.978-3-85448-046-4_35.

[3] D. Detlefs, G. Nelson, J. B. Saxe, Simplify: A theorem prover for program checking, J. ACM 52 (2005) 365–473. doi:10.1145/1066100.1066102.

[4] A. Reynolds, H. Barbosa, P. Fontaine, Revisiting enumerative instantiation, in: Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 10806, Springer, 2018, pp. 112–131. doi:10.1007/978-3-319-89963-3_7.

[5] Y. Ge, L. M. de Moura, Complete instantiation for quantified formulas in satisfiabiliby modulo theories, in: Computer Aided Verification CAV, 2009, pp. 306–320. doi:10.1007/978-3-642-02658-4_25.

[6] 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: Tools and Algorithms for the Construction and Analysis of Systems, TACAS, volume 13243 of *LNCS*, Springer, 2022, pp. 415–442. doi:10.1007/978-3-030-99524-9_24.

[7] L. M. de Moura, N. Bjørner, Z3: an efficient SMT solver, in: Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, 2008, pp. 337–340. doi:10.1007/978-3-540-78800-3_24.

[8] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, Satisfiability modulo theories, in: A. Biere, M. Heule, H. van Maaren, T. Walsh (Eds.), Handbook of Satisfiability - Second Edition, volume 336 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2021, pp. 1267–1329. URL: https://doi.org/10.3233/FAIA201017. doi:10.3233/FAIA201017.

[9] V. Weispfenning, The complexity of linear problems in fields, Journal of Symbolic Computation 5 (1988) 3–27. doi:10.1016/S0747-7171(88)80003-8.

[10] A. Tarski, A Decision Method for Elementary Algebra and Geometry, Univ. of California Press, Berkeley, 1951. doi:https://doi.org/10.2307/jj.8501420, also in [?].

[11] J. H. Davenport, J. Heintz, Real quantifier elimination is doubly exponential, Journal of Symbolic Computation 5 (1988) 29–35. doi:10.1016/S0747-7171(88)80004-X.

[12] N. Bjørner, M. Janota, Playing with quantified satisfaction, in: International Conferences on Logic for Programming, Artificial Intelligence and Reasoning (LPAR), 2015, pp. 15–27.

[13] M. P. Bonacina, S. Graham-Lengrand, C. Vauthier, QSMA: A new algorithm for quantified satisfiability modulo theory and assignment, in: Automated Deduction – CADE 29, Springer Nature Switzerland, 2023, p. 78–95. doi:10.1007/978-3-031-38499-8_5.

[14] J. A. Robinson, A. Voronkov (Eds.), Handbook of Automated Reasoning (in 2 volumes), Elsevier and MIT Press, 2001. URL: https://www.sciencedirect.com/book/9780444508133/handbook-of-automated-reasoning.

[15] K. Korovin, Non-cyclic sorts for first-order satisfiability, in: P. Fontaine, C. Ringeissen, R. A. Schmidt (Eds.), Frontiers of Combining Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 214–228.

[16] M. Gario, A. Micheli, PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms, in: SMT Workshop 2015, 2015.

[17] C. E. Brown, K. Chvalovský, M. Janota, M. Olšák, S. Ratschan, SMT and functional equation solving over the reals: Challenges from the IMO, 2025. URL: https://arxiv.org/abs/2504.15645. arXiv:2504.15645.

[18] C. Barrett, P. Fontaine, C. Tinelli, The SMT-LIB Standard: Version 2.6, Technical Report, Department of Computer Science, The University of Iowa, 2017. Available at www.SMT-LIB.org.

[19] J. Christ, J. Hoenicke, A. Nutz, SMTInterpol: An interpolating SMT solver, in: A. F. Donaldson, D. Parker (Eds.), Model Checking Software - 19th International Workshop, SPIN 2012, Oxford, UK, July 23-24, 2012. Proceedings, volume 7385 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 248–254. doi:10.1007/978-3-642-31759-0_19.

[20] L. Kovács, A. Voronkov, First-order theorem proving and Vampire, in: N. Sharygina, H. Veith (Eds.), Computer Aided Verification - 25th International Conference, CAV, volume 8044 of *LNCS*, Springer, 2013, pp. 1–35. doi:10.1007/978-3-642-39799-8_1.