ECAI 2024
U. Endriss et al. (Eds.)

doi:10.3233/FAIA241009

Machine Learning for Quantifier Selection in cvc5

Jan Jakubův[a,b,*], **Mikoláš Janota**[a], **Jelle Piepenbrock**[a,c] **and Josef Urban**[a]

[a]Czech Technical University in Prague, Prague, Czech Republic
[b]University of Innsbruck, Innsbruck, Austria
[c]Radboud University, Nijmegen, Netherlands

Abstract. In this work we considerably improve the state-of-the-art SMT solving on first-order quantified problems by efficient machine learning guidance of quantifier selection. Quantifiers represent a significant challenge for SMT and are technically a source of undecidability. In our approach, we train an efficient machine learning model that informs the solver which quantifiers should be instantiated and which not. Each quantifier may be instantiated multiple times and the set of the active quantifiers changes as the solving progresses. Therefore, we invoke the ML predictor many times, during the whole run of the solver. To make this efficient, we use fast ML models based on gradient boosted decision trees. We integrate our approach into the state-of-the-art cvc5 SMT solver and show a considerable increase of the system's holdout-set performance after training it on a large set of first-order problems collected from the Mizar Mathematical Library.

1 Introduction

The use of machine learning methods in various fields of automated reasoning is an emerging research topic with successful applications [8, 21, 39]. Machine learning methods were previously integrated into various provers, solvers, and related systems [25, 18]. Here, we focus on the task of selecting quantifiers within the state-of-the-art *Satisfiability Modulo Theory* (SMT) solver, cvc5, where various instantiation methods are applied to quantified formulas to refine problem representation. While cvc5's instantiation methods generate instantiations for all applicable quantified formulas, we aim to filter out the formulas considered by the instantiation module using a machine-learned predictor trained on previously proved similar problems. We refer to this process as *quantifier selection* for simplicity.

In contrast to previous research efforts that concentrated on fine-grained control of the solver at the *term level* [26, 28] , we simplify the problem by shifting our focus to controlling the solver at the *quantifier level*. This simplification allows us to employ our method basically with any quantifier instantiation method supported by cvc5. Our methods also improve upon a related method of *offline premise selection* [1, 18], where initial quantified assumptions are filtered *only once* before launching the solver. Since this filtering can result in an irrecoverable mistake (a *deletion abstraction* [30] resulting in a too weak theory) when a necessary assumption is deleted, our method ensures that every quantified formula will be considered for instantiations with a non-zero probability. This yields a more complex, probabilistically guided framework implementing deletion and instantiation abstractions in the framework proposed in [30]. As we employ an efficient version of *gradient boosting decision trees* with simple, but easily computable *bag of words features*, our implementation produces only a minimal overhead over the standard cvc5 run.

* Corresponding Author. Email: jakubuv@gmail.com

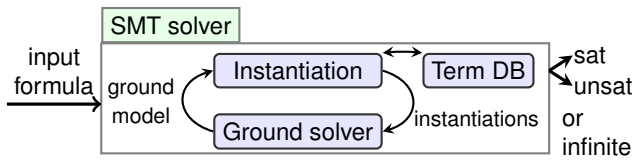

Figure 1. Schema of quantifier instantiation in SMT, adapted from [27].

The simplicity of our approach allows us to apply it extensively and train on a large number of problems from the *Mizar Mathematical Library* (MML). The method is also remarkable for its cross-strategy *model transfer*, where a model trained on samples from one strategy improves the performance of virtually any other strategy. This allows us to improve the state-of-the-art performance of cvc5 on MML by more than 20% in both the *single strategy* and *portfolio* scenarios. Our best machine-learned strategy, alone, outperforms the state-of-the-art cvc5's portfolio from the latest CASC competition by more than 10%.

2 Background: SMT solving

To solve formulas *without* quantifiers (aka *ground* formulas), SMT solvers combine SAT solving with theory solving [5]. The SAT solver handles the boolean structure of the formula and theory solvers reason about concrete theories, such as theory or reals, integers, etc. Problems in FOL do not contain such theories explicitly and therefore, to reason about FOL, an SMT solver needs to only support *uninterpreted functions* and *equality* (combination known as `EUF`). For reading this paper, it is not essential to understand how exactly the SMT solvers solve ground formulas; we refer the interested reader to relevant literature [14, 3, 5, 12].

Formulas with *with* quantifiers represent a significant challenge for SMT. In general, SMT solvers use *instantiations*—unless they deal with decidable quantified theories [13, 7, 37]. Instantiations are done with the goal of achieving a contradiction. This style of reasoning can be seen as a direct application of the Herbrand's theorem. For example, for $(\forall x : \mathbb{R}.\, x > 0)$ instantiating x with the value 0 yields $0 > 0$, which immediately gives a contradiction (in the theory of reals).

Input formulas do not need to be in prenex form, which effectively means that the solver may activate only some of the quantified subformulas. For this purpose, a subformula $(\forall x_1 \ldots x_n\, \phi)$ is seen as a generator of lemmas of the form $(\forall x_1 \ldots x_n\, \phi) \rightarrow \phi[x_1/t_1, \ldots, x_n/t_n]$, with t_i ground terms. For

example, $\forall x\, R(f(x), c)$ may be instantiated as $(\forall x\, R(f(x), c)) \rightarrow R(f(c), c)$. Existential quantifiers are removed by skolemization.

The solving process alternates between a *ground solver* and an *instantiation module* (Figure 1), where the ground solver perceives quantifiers as opaque propositions. After identifying a model for the ground part, control shifts to the *instantiation module*. This module generates new instances of the quantified sub-formulas that are currently meant to hold. A new instance is added to the ground part of the formula, thus making it stronger. The process stops if the ground part becomes unsatisfiable, if ever (model-based quantifier instantiation can also lead to satisfiable answers [16]).

The cvc5 solver implements several instantiations methods. Some of those can be seen as syntactic-driven approaches, *e-matching* [14] or syntax-guided instantiation [31]. Other methods are semantic-driven such as *model-based* [16, 35] or *conflict-based* [36]. A straightforward, but complete for FOL is *enumerative instantiation* [27, 38] which exhaustively generates all possible instantiations. Both e-matching and enumerative instantiation require the solver to maintain a database of ground terms. This database grows as new instantiations are performed. In the case of enumerative instantiation, the terms are selected systematically going from the oldest to the newest. E-matching tries to instantiate in a way to match an existing term. We dedicate Sections 2.1 and 2.2 to describing these two prominent methods because they are the most effective for quantifier instantiation, and we anticipate that they will derive the greatest benefit from machine learning.

All these techniques target the quantifiers individually, i.e., the instantiation of one quantifier does not directly influence the instantiation of another one. The default implementation of cvc5 is to iterate over all currently-active quantifiers and instantiate them, one by one. If such instantiation step is superfluous, the generated lemma not only burdens the underlying SAT solver, but it also pollutes the considered set of terms with all the lemma's subterms.

2.1 Enumerative instantiation

This *enumerative instantiation* mode enumerates all possible tuples of terms from the term database that can be used to instantiate a quantified expression [38, 27]. Given an ordering of the suitable ground terms for each quantifier, the strategy enumerates tuples by starting at the tuple that contains the first term in each quantifier-term ordering, and moves further into the orderings by incrementing the indices that retrieve further terms in the quantifier-term index.

The particular criterion for term ordering that is used in enumerative instantiation is the *age* of the ground terms, with terms with higher age being preferred. That is, ground terms that were in the original problem have the highest priority for being tried. As an example of the logic of the procedure, imagine that there is a ground part $\{p(c)\}$ and a quantified expression $\forall x.\, q(f(x))$. In this case, the ground term c is available at the start of the procedure and therefore will be tried first. This creates a ground lemma that is a consequence of the quantified expression, $q(f(c))$. The solver would now recognize that $f(c)$ is also an available ground term, insert it into the term database and it would be used in the next round of instantiations. With nested terms, many new ground terms may be created by a single instantiation step.

2.2 E-matching

The *e-matching* instantiation procedure searches for instantiations that match some already available ground term [14], taking term equality into account. Of course, at any point and especially late in the procedure when many terms have been generated, there can be many possible ways to create such matching terms. The e-matching instantiation process is therefore focused using *triggers*, which are user-supplied or heuristically generated patterns.

We show the following example for a more concrete perspective. In the example, we assume that a theory module that can evaluate integer arithmetic statements is available. Given the following ground facts $\{p(a), a = f(24)\}$ and the quantified expression $\forall x.\, \neg p(f(x)) \vee x < 0$. A trigger in the form $p(f(x))$ leads to x being instantiated with the ground term 24 as there is an existing ground term $p(f(24))$, when taking equality into account. Instantiating with 24 generates the consequent lemma $\neg p(f(24)) \vee 24 < 0$. This lemma contradicts the other ground facts (as the theory of integer arithmetic knows that in fact $24 > 0$), and therefore the solver stops and reports that a contradiction (unsat) was found. An automated trigger generation method is implemented in the e-matching module of cvc5, and its behavior can be influenced through various options provided by the user.

3 Machine-learned quantifier selection

In this section, we detail our quantifier selection method and its implementation in the cvc5 solver, covering the process from extraction of training examples to model training, and integration.

3.1 Instantiation modules

The instantiation methods supported by cvc5 are implemented through various *instantiation modules* that share a common interface. An instantiation module is invoked via its *check* method to refine information about quantified formulas by introducing appropriate formula instances. Each module is provided with information about currently asserted quantified formulas, and it selects formulas and generates their instances in accordance with the implemented instantiation method. Subsequently, control is returned to the ground solver.

Effective quantifier and instance selection can significantly enhance performance, and different instantiation methods offer grounds for various possible applications of machine learning methods at a method-specific level [26, 28]. However, we propose a generic method for *quantifier selection* by limiting the formulas visible to the modules. As all instantiation modules iterate over available quantified formulas and process them one by one, we can seamlessly integrate a quantifier selector into any module and simply skip the processing of undesirable quantifiers. To predict the quality of quantifiers, we utilize an efficient implementation of *decision tree ensembles* (LightGBM [29]) that enables easy and fast integration with cvc5. Decision tree models can be trained to classify quantified formulas as *positive* or *negative* based on provided training examples. The trained model can be employed within an instantiation module to skip the processing of negative quantifiers.

3.2 Feature vectors and training examples

To use decision trees for classifying quantified formulas in cvc5, we need to represent SMT formulas by numeric feature vectors. SMT formulas within cvc5 are represented using directed acyclic graphs with shared node representation. The nodes of the graph are labeled with symbols representing logical connectives, quantifiers, variables, and interpreted/uninterpreted theory symbols. Each symbol falls into

one of finitely many *kinds*, designating theory-specific (builtin) symbols. Different kinds exist for logical connectives ($\neg$, $\wedge$, $\vee$, etc.), quantifiers ($\forall$, $\exists$), interpreted symbols ($+$, $<$, 0, etc.), and variables. There is only one kind for all uninterpreted functions, including uninterpreted constants. We utilize cvc5's kinds as *bag of words* features, representing the quantified formula q with the counts of symbols of each kind k. In particular, we establish an enumeration of the kinds, and the quantifier q is represented by the vector φ_q, where $\varphi_q[i]$ denotes the number of symbols of kind i in q. More than 300 different kinds are defined in cvc5.

The usefulness of a quantified formula might depend on the context of the problem being solved. Hence it is essential to allow our models to produce context-dependent predictions. We achieve this by embedding the problem features within the feature vector. First, we represent the problem being solved (P) by the vector φ_P, obtained as the sum $\sum_{f \in P} \varphi_f$ where φ_f is the feature vector of each formula f asserted in problem P. Second, the context vector φ_P and the quantifier vector φ_q are concatenated into the double-length vector (φ_P, φ_q) representing the quantified formula q when solving problem P. This enables problem-specific predictions. To generate training examples, we utilize cvc5's option to dump instantiations (`dump-instantiations`) employed in solving a problem. When this option is combined with the `produce-proofs` option, only instantiations necessary for the proof are printed. We use this differentiation to classify formulas as positive or negative as follows. Following a successful (unsat) run of cvc5 on problem P, we gather all the quantified formulas q processed by instantiation modules during the run. To construct training examples, we label a feature vector (φ_P, φ_q) as *positive* if q generated an instantiation required by the proof of P, otherwise we label it as *negative*. Training examples can thus be extracted from both ML-guided and unguided runs.

3.3 Model training

To construct a training dataset, we collect labeled training vectors from a large set of successful runs of cvc5. Note that no training data is extracted from unsuccessful runs, as there is no reference point to label processed formulas as positive or negative. It is essential to collect a substantially large amount of training data to enhance the generalization capabilities of the model. The gradient boosting decision tree models can be easily constructed using the LightGBM library [29], which is known to handle large training data well. A model consists of a sequence of decision trees, where each tree is trained to correct any imprecision introduced by the previous trees in the sequence. The number of different trees in the model is one of the many LightGBM *hyperparameters* that influence the process of model training and also the performance of the resulting model.

In order to prevent overfitting of the model to the training data, we split the data into training and development sets. This split is done at the problem level, ensuring that all training vectors from a single problem contribute to the same set. We use binary classification as the model objective and we train several models on the training set with various values of selected hyperparameters. Out of these models, we select the model with the best performance on the development set.

The performance of the model on the development set is measured in terms of prediction accuracy. That is, all vectors from the development set are evaluated using the trained model, and the predicted labels are compared with the expected labels from the development set. While processing a redundant quantified formula cannot completely prevent an SMT solver from finding a solution, omitting to process an important formula can ultimately close the door to success. Hence, it is essential for models for quantifier selection in SMT to favor recognition of positive examples, that is, to minimize *false negative* errors. We achieve this by computing separately the accuracies on positive and negative development examples, denoted pos and neg respectively, and selecting as the model with the best value of $2 \cdot pos + neg$ as the final model. In this way, we favor models with better performance on positive training examples.

3.4 Model integration

Since LightGBM provides a C++ interface for model predictions, it can be easily integrated into the cvc5 codebase, which is also written in C++. The interface offers methods to load a model and compute the prediction of a quantified formula represented by a feature vector. Since we employ binary classification for the model objective, the prediction function returns a floating-point value between 0 and 1 which is typically compared with a fixed threshold, like 0.5, to distinguish positive and negative clauses.

However, we do not directly utilize the predicted values in this manner. Instead, each time we predict a quantified formula, we generate a random number between 0 and 1 to serve as the threshold for the comparison. This approach allows us to process formulas scored below 0.5, ensuring that every formula will eventually be processed with some non-zero probability. In our experiments, the random threshold slightly outperformed any fixed threshold, and it further guards against potential irreparable errors resulting from false negative predictions. Moreover, this approach sets our method apart from a related technique known as *offline premise selection* [1, 18], where formulas are filtered in advance and never passed to the solver.

The two most prominent modules for quantifier selection in cvc5 are *enumerative instantiation* and *e-matching*. We implement our machine learning guidance for them, as well as for the modules relying on *conflict-based quantifier instantiation* and *finite model finding*.

4 Experiments on large Mizar dataset

The Mizar Mathematical Library (MML) [2] stands as one of the earliest extensive repositories of formal mathematics, encompassing a broad spectrum of lemmas and theorems across various mathematical domains. We utilize translations of MML problems into first-order logic, a process facilitated by the MPTP system [41, 42]. The MPTP benchmark has emerged as a valuable resource for machine learning research, offering a diverse collection of related problems [33, 23, 39, 34, 19, 10]. In our work, we focus on the easier *bushy* variants of MPTP problems, wherein premises are partially filtered externally beforehand.

We use a train/development/holdout split from other experiments in literature [25, 17] to allow a competitive evaluation. This splits the whole set of Mizar problems into the *training*, *development*, and *holdout* subsets, using a 90 : 5 : 5 ratio, yielding 52,125 problems in the training set, 2,896 in devel, and **2,896 problems in the holdout set**. We use the *training* set to collect training examples for machine learning, and we use the *development* set to select the best model out of several candidates trained with different hyperparameters. The best portfolio constructed on the development is henceforth evaluated on the *holdout* set and compared with the baseline portfolio. All strategies in this section are evaluated with the **time limit of 60 seconds** per strategy and problem.[1] Note that this includes the time used

[1] All the experiments were performed on machines with two AMD EPYC 7513 32-Core processors @ 3680 MHz and with 514 GB RAM.

by the trained ML predictor, that is, the times used for the guided and unguided runs are fully comparable.

4.1 Baseline strategies and portfolio

We employ a state-of-the-art portfolio of cvc5 strategies as a baseline to evaluate our machine learning approach. We incorporate all 16 strategies available in the CASC competition portfolio of cvc5.[2] A portfolio consists of strategies that are typically executed sequentially, each for a fraction of the overall time limit. The first successful strategy returns the solution and terminates the portfolio execution. The CASC competition focuses on problems in automated theorem proving, and cvc5 has recently demonstrated outstanding performance on problems from theorem proving systems [18]. We utilize the FOF (first-order formulas) portfolio, specifically designed for problems in the theory of uninterpreted functions (UF), which aligns well with our intended application on Mizar MML problems expressed in the same logic.

In addition to the 16 CASC strategies, we also incorporate 3 publicly available strategies[3] developed recently for the Mizar MML problems using the Grackle strategy invention system [20]. The Grackle strategies in that work were tailored for slightly different versions of Mizar problems, employing a more precise method of premise selection. However, they also perform well on the *bushy* version of Mizar problems used in our work here.

The selected CASC strategies casc_n are detailed in Figure 2, while the three Grackle strategies grk_n are outlined in Figure 3. In both cases, strategies are described in terms of their cvc5 command line options, either as a boolean `flag` or as an `option=value` pair. Common options appearing in more than one strategy are typeset in **`bold`**. Options adjusting the trigger behavior of the e-matching module are highlighted in ***`bold-italics`***. Note that almost all the strategies activate enumerative instantiation using `full-saturate-quant`, while keeping e-matching and conflict-based quantifier instantiation (`cbqi`) turned on by default. The option `cbqi-vo-exp`, which activates an experimental mode for variable ordering, often significantly boosts strategy performance. Strategies casc_4, casc_7, casc_{10}, casc_{13}, and casc_{14} are selected as the strongest CASC strategies, with casc_{13} being the best among them. The rationale behind selecting casc_6 and casc_8 is revealed later in Section 4.3. Among the three Grackle strategies, grk_1 performs the best. It is worth noting that grk_1 and grk_2 are proper extensions of casc_{14}, while grk_3 is a proper extension of casc_7. The core of a successful cvc5 strategy for Mizar problems appears to be enumerative instantiation with appropriate adjustments of triggers for e-matching.

We evaluate the baseline strategies directly on the *holdout* set to establish the best possible state-of-the-art portfolio on our Mizar benchmark. The most effective of the CASC strategies is casc_{13}, solving 1,406, while the best Grackle strategy, grk_1, solves 1,443. The three most complementary CASC strategies (casc_{13}, casc_{14}, and casc_7) collectively solve 1,516, whereas the three Grackle strategies solve 1,552. The best portfolio of 6 strategies, whether CASC or Grackle, **solves 1,614 holdout problems (55.8 %)**. Interestingly, this portfolio includes all three Grackle strategies and CASC strategies casc_{13}, casc_4, and casc_{10}. Notably, casc_7 is replaced by casc_4 compared to the top three CASC strategies. This is expected as casc_7 is a proper extension of grk_3, thus being surpassed by grk_3, allowing casc_4 to enter the best portfolio. It is worth noting that casc_4 is the only strategy not relying on enumerative instantiations or e-matching, instead employing finite model finding [35]. While finite model finding is not anticipated to perform optimally in the case of unsatisfiable problems, it can assist in solving specific instances.

[2] https://github.com/cvc5/cvc5/blob/cvc5-1.1.1/contrib/competitions/casc/run-script-cascj11-fof

[3] https://github.com/ai4reason/cvc5_grackle_mizar

Table 1. Baseline portfolio performance without ML.

strategy	*model*	*solves*	*+new*	*=total*	*adds*
grk_1	−	1,443	+1,443	= 1,443	−
casc_{13}	−	1,406	+75	= 1,518	+5.20 %
grk_3	−	1,365	+45	= 1,563	+2.96 %
grk_2	−	1,408	+25	= 1,588	+1.60 %
casc_4	−	740	+14	= 1,602	+0.88 %
casc_{10}	−	1,268	+12	= **1,614**	+0.75 %

A common method for assessing the performance of a strategy portfolio is through the *greedy cover* sequence. This sequence is generated by initially choosing the most powerful individual strategy and marking the problems it solves. Subsequently, the next best strategy, based on the remaining unsolved problems, is selected, and this process continues iteratively until it is no longer feasible nor necessary. Through this approach, the greedy cover identifies strategies that complement each other, thereby optimizing the total number of solved problems. This method serves as an approximation of the NP-complete problem of *set cover* [15], which could be utilized instead but typically does not yield significantly improved results [11, 6].

The greedy cover sequence of the best portfolio of size 6, considered henceforth as the *baseline portfolio*, is presented in Table 1. The column *strategy* lists the strategy names in the order determined by the greedy cover. The column *model* is reserved for describing an ML model in later sections (here unused, denoted by "−"). The column *solves* indicates the number of holdout problems solved by each strategy individually, while the column *+new* shows how many new problems are contributed by each strategy to the portfolio. The column *total* denotes the total number of problems solved by the portfolio up to that point, where *total* equals *new* plus the *total* from the previous line. Finally, the column *adds* presents the value of *new* as a percentage of the current portfolio performance (for instance, casc_{13} adds 75 problems to the currently solved 1,443, representing 5.20%). The overall performance of the portfolio is reflected in the bottom value of the *=total* column.

4.2 Training and evaluation

We train two different models, denoted as single and three, using distinct sets of training examples generated by evaluating baseline strategies, as described in Section 3.2. We follow two approaches: (1) utilizing only training data generated by a single strategy, or (2) combining training examples from various strategies.

Model single: Trained on data generated by the *strongest* individual strategy grk_1. The model is trained on 25,415 solved training problems, yielding 2,281,923 training vectors, with 15.7 % of them being positive. The size of the text training file is 255 MB, and the model size is 16 MB.

Model three: Trained on data generated by all three Grackle strategies grk_1, grk_2, and grk_3. The model is trained on 27,032 solved training problems, with up to three solutions per problem, resulting in 6,318,660 training vectors, of which 16.2 % are positive. The size of the text training file is 707 MB, and the model size is 22 MB.

The strategy grk_1 was chosen for the model single due to its status

strategy	*cvc5 command line options*
casc$_4$	`finite-model-find uf-ss=no-minimal`
casc$_6$	**`full-saturate-quant`** ***`trigger-sel=max`***
casc$_7$	**`full-saturate-quant`** ***`multi-trigger-priority`*** ***`multi-trigger-when-single`***
casc$_8$	**`full-saturate-quant`** ***`multi-trigger-cache`***
casc$_{10}$	**`full-saturate-quant`** `enum-inst-interleave decision=internal`
casc$_{13}$	**`full-saturate-quant`** `pre-skolem-quant=on`
casc$_{14}$	**`full-saturate-quant`** **`cbqi-vo-exp`**

Figure 2. Selected cvc5's strategies from the CASC portfolio.

strategy	*cvc5 command line options*
grk$_1$	**`full-saturate-quant`** **`cbqi-vo-exp`** ***`relational-triggers`*** `cond-var-split-quant=agg`
grk$_2$	**`full-saturate-quant`** **`cbqi-vo-exp`** ***`relevant-triggers`*** ***`multi-trigger-priority`*** `ieval=off no-static-learning miniscope-quant=off`
grk$_3$	**`full-saturate-quant`** ***`multi-trigger-priority`*** ***`multi-trigger-when-single`*** `term-db-mode=relevant`

Figure 3. Grackle strategies optimized for minimized MML problems.

Table 2. The best development portfolio evaluated on holdout.

strategy	*model*	*solves*	*+new*	*=total*	*adds*
grk$_1$	single	1,781	+1,781	= 1,781	−
grk$_2$	three	1,715	+86	= 1,867	+4.83 %
grk$_3$	single	1,688	+47	= 1,914	+2.52 %
grk$_1$	three	1,763	+24	= 1,938	+1.25 %
grk$_3$	three	1,674	+14	= 1,952	+0.72 %
grk$_2$	single	1,715	+8	= **1,960**	+0.41 %

Table 3. Best possible ML portfolio on holdout.

strategy	*model*	*solves*	*+new*	*=total*	*adds*
grk$_1$	single	1,781	+1,781	= 1,781	−
grk$_2$	three	1,715	+86	= 1,867	+4.83 %
grk$_3$	single	1,688	+47	= 1,914	+2.52 %
grk$_1$	three	1,763	+24	= 1,938	+1.25 %
casc$_{10}$	single	1,601	+15	= 1,953	+0.77 %
grk$_3$	three	1,674	+12	= **1,965**	+0.61 %

as the strongest individual strategy, while the three Grackle strategies were selected based on their complementarity. This configuration enables us to examine whether a model trained on data from one strategy can effectively generalize to a different strategy. Additionally, we aim to investigate the potential success of combining training data from multiple strategies.

To generate training data, we assess three Grackle strategies on both the training and development sets. Training examples extracted from the training set are utilized for model training, while examples from the development set aid in hyperparameter tuning, as outlined in Section 3.3. This process results in the creation of two models: single and three. Subsequently, we evaluate both models using all baseline strategies on the development set. We determine the best portfolio on the development set and evaluate its performance on the holdout set.

The resulting ML portfolio, evaluated on the holdout set, is shown in Table 2. Notably, the combination of the three Grackle strategies with both models dominates the portfolio, surpassing the CASC strategies. Particularly, the best strategy grk$_1$ benefits significantly from model single, improving from 1,443 to 1,781, representing an increase of more than 23 %. This top-performing ML strategy alone now outperforms the initial portfolio's performance (1,614) by over 10 %. **Overall, the ML portfolio demonstrates remarkable improvement over the baseline portfolio, increasing from 1,614 to 1,960 (67.7 % of holdout problems), marking a significant enhancement of 21.4 %.** Notably, our method improves the state-of-the-art cvc5 portfolio of strategies. In particular, since the term-based selection methods from the literature [26, 28] have not been integrated into e-matching or compared with any of the stronger cvc5 strategies, we are not aware of any method-specific approach from the literature that reports a similar achievement for SMT solvers.

To assess the potential for overfitting in our models, we further evaluate all baseline strategies with all models on the holdout set. This allows us to determine the best possible portfolio and compare its performance with our portfolio constructed on the development set. The best possible portfolio of 6 strategies on the holdout set is outlined in Table 3. It's evident that the portfolio constructed on the development set closely approximates the best possible result. The only strategy, casc$_{10}$, paired with the model single, was able to marginally improve the portfolio by solving 5 additional problems.

The impact of ML models on the performance of the best strategy grk$_1$ is further elucidated in Figure 4 through scatter plots. Each plot compares the runtimes of two strategies, s_1 and s_2, by plotting a dot for each problem p at coordinates (r_1, r_2), representing the runtimes of s_1 and s_2 on p. Points clustered around the diagonal line indicate similar results, while points below the diagonal (depicted in blue) represent an improvement of s_2 over s_1. In our context, the first strategy s_1 is consistently grk$_1$ (without ML), which is compared with different strategies in various plots.

In Figure 4, the first two plots compare grk$_1$ with (1) another run of itself and (2) a run of grk$_1$ with a different random seed. The first plot illustrates the deterministic nature of the solver, showing similar performance when launched again on the same problem. However, certain aspects of cvc5 involve randomization, and running with a different initial random seed can yield different results, as demonstrated by the second plot. We observe a typical addition of around 2 % between two runs with different random seeds.

Subsequently, the last two plots compare grk$_1$ without ML with grk$_1$ utilizing the single model on the development set (third plot from the left) and the holdout set (right). Remarkably, there is no substantial difference in performance between the development and holdout sets. Furthermore, it is evident that the impact of ML is much more significant than that of a different random seed. Notably, there

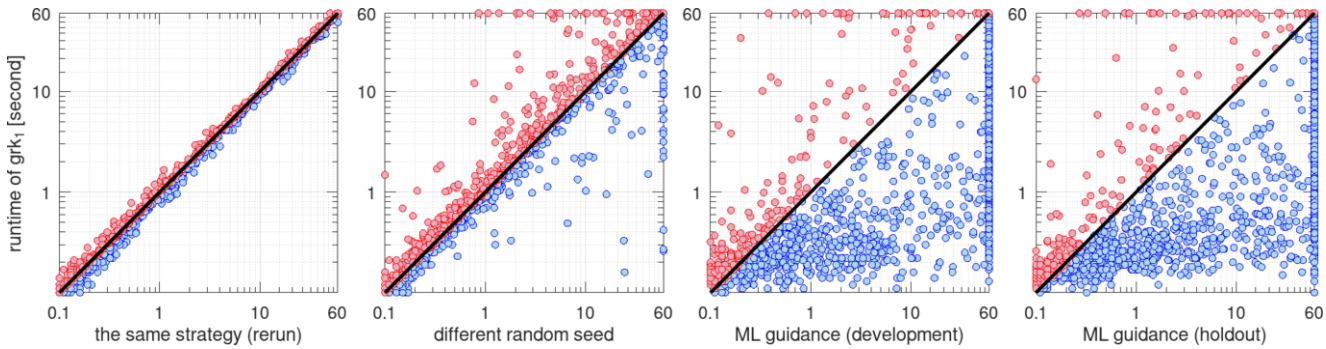

Figure 4. Impact of machine guidance on runtime.

Table 4. Impact of ML models on selected strategies.

	w/o ML	*ML model* single		*ML model* three	
strategy	*solves*	*solves*	*gain*	*solves*	*gain*
grk$_1$	**1,443**	**1,781**	+23.42%	**1,763**	+22.18 %
grk$_2$	1,408	1,715	+21.80 %	1,715	+21.80 %
grk$_3$	1,365	1,688	**+23.66%**	1,674	**+22.64%**
casc$_{13}$	**1,406**	1,426	+1.42 %	1,407	+0.07 %
casc$_{14}$	1,389	**1,696**	+22.10 %	**1,693**	+21.89 %
casc$_7$	1,354	1,650	+21.86 %	1,631	+20.46 %
casc$_{10}$	1,268	1,601	+26.26 %	1,599	+26.10 %
casc$_6$	1,147	1,553	+35.40 %	1,562	**+36.18%**
casc$_8$	1,105	1,498	**+35.57%**	1,466	+32.67 %
casc$_4$	740	805	+8.78 %	788	+6.49 %

is a large number of points below the diagonal and a notable cluster of points at the right axis, showing problems solved due to ML.

We have also conducted experiments with random quantifier selection, where the quantified formulae to be instantiated are chosen randomly. The results show a similar behavior to runs with a different random seed, as described in Figure 4 (second plot). In summary, random quantifier selection can solve some new problems, but it also loses a significant number of solutions. Overall, random quantifier selection falls short of achieving a 20% improvement over the baseline (select-all) strategy.

4.3 Cross-strategy model transfer

In this section, we investigate the transferability of knowledge learned from one strategy to different strategies. Specifically, we examine how well a model trained on data from one strategy performs when applied to different strategies. For instance, since the model single is trained on data from grk$_1$, we aim to compare its performance with grk$_1$ and with other strategies. Similarly, we seek to evaluate how the model three performs when applied to the source Grackle strategies compared to other CASC strategies.

Table 4 provides a summary of the impact of ML models single and three on selected strategies. The first numeric column (*w/o ML solves*) displays the performance of each strategy without any ML model. The two middle columns (*ML model* single) illustrate the impact of the model single on the performance of each strategy individually. The column *solves* indicates the number of problems solved by the strategy guided by the model, while the column *gain* calculates the improvement over the strategy without ML. The highest values in each column are highlighted separately for the Grackle and CASC strategies. The last two columns show the impact of the model three.

Initially, we observe that all Grackle strategies derive equal benefits from both models. While grk$_1$ experiences a slightly greater improvement from its own data (single), other Grackle strategies also benefit significantly from the data provided by single. There is no discernible distinction in their performance, as both models enhance the performance of all Grackle strategies by more than 20 %. However, there is evidently a considerable difference in the problems solved by different models, as evidenced by the inclusion of Grackle strategies in the best ML portfolio (Table 2).

Interestingly, many other CASC strategies also experience significant benefits from training data generated by Grackle strategies. However, as an exception, only the best CASC strategy, casc$_{13}$, failed to benefit from the training data and achieved only negligible improvement. Upon inspecting Figure 2, we observe that casc$_{13}$ is the only strategy that utilizes `pre-skolem-quantifier=on`. Since all other strategies without this option were able to improve, it's possible that eager skolemization influences feature vectors and renders the training data incompatible. The second worst result is observed with casc$_4$, which experiences only an 8.78 % improvement. This is the only strategy that employs finite model finding instead of enumerative instantiation and e-matching. It is noteworthy that even a fundamentally different instantiation approach can, at least partially, benefit from knowledge learned from related methods, which is not always the case [18].

The most improved CASC strategies are casc$_6$ and casc$_8$, both of which see improvements of more than 35 %. Notably, both casc$_6$ and casc$_8$ utilize options to adjust triggers for e-matching, similar to Grackle strategies. From this, we conclude that our models generally transfer well to different strategies. Transfer tends to be more effective between strategies based on similar methods, particularly among strategies that adjust trigger behavior. Finally, no significant improvement is gained when merging data from various source strategies.

4.4 Analysis of models and training data

We proceed with an analysis of the trained models by examining features present in the training data and their importance in the model. LightGBM models report for each feature its *importance*, indicating how much each feature contributes to the predictions made by the model. This aids in understanding which features are most relevant for the model's performance. We collect the training data produced by grk$_1$ on the development set. Remarkably, only 24 features are utilized in the data, but they exhibit a relatively high number of different values, especially in the case of context features. Many features are present in all training examples. Most of the features describe the count of logical connectives in a formula (and, or, imply, not, forall,

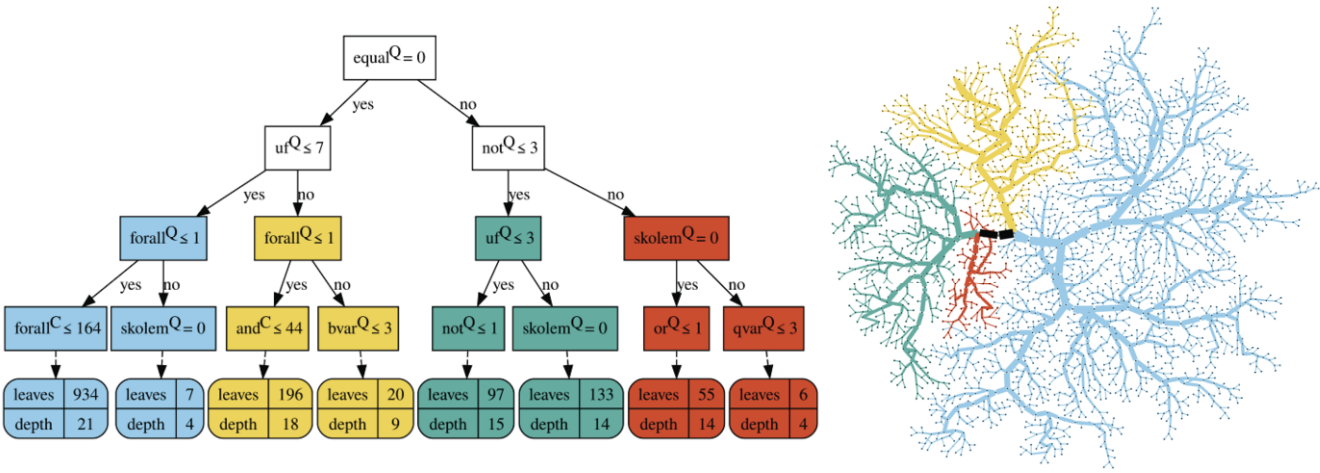

Figure 5. Decision tree visualization.

equal). Additionally, the feature uf accumulates the count of first-order (uninterpreted function) symbols in a formula. Note that we do not distinguish between different (uninterpreted) symbols names, and hence our methods are *symbol-independent* [24]. Other features count special symbols, like variables (var), bound variables (bvar), quantified variables (qvar), or skolem symbols (skolem). The key quantifier feature (marked Q) is symbol count (uf^{Q}), while the count of equalities ($\mathsf{equal}^{\mathsf{C}}$) dominates among context features (C).

Figure 5 presents a visualization of the first decision tree in the model single. The first of the 100 trees in the model is expected to contribute significantly to the final prediction. This decision tree is a binary tree with inner nodes labeled by conditions on feature values, while the leaves are labeled by scores. Edges labeled by true or false navigate each feature vector to a unique leaf to compute the score. Scores from all trees are combined into the final prediction. The root of the first tree in single is located at the left part of Figure 5. The root condition checks the count of equalities in the quantified formula (feature $\mathsf{equal}^{\mathsf{Q}}$). If the value is 0, the "yes" edge is followed, and the next condition $\mathsf{uf}^{\mathsf{Q}} \leq 7$ is checked. The first four levels of the tree are displayed, and the bottom layer indicates the number of leaves in and the depth of each subtree under that node.

To provide an overview of the complexity of the tree, the full tree is visualized on the right part of Figure 5. Each edge has a width proportional to the size of the subtree under this edge. The first two splits naturally divide the tree into 4 subtrees, and these subtrees are colored by corresponding colors in both trees. Therefore, it is apparent that more than half of the tree (the blue part) is concerned with formulas without equality ($\mathsf{equal}^{\mathsf{Q}} = 0$) and few symbols ($\mathsf{uf}^{\mathsf{Q}} \leq 7$). Note that context features (like $\mathsf{and}^{\mathsf{C}}$) are used only from the fourth level of the tree. This tree visualization illustrates that even with a dozen relatively simple features, decision trees might be quite complex, with more than 1,500 leaves.

5 Conclusions and future work

This paper develops a novel approach that trains a machine learning model to decide which quantifiers should be instantiated inside an SMT solver. The approach is conceptually simple: in each instantiation round, each quantifier is considered to be selected for instantiation or not; the exact way *how* the quantifier is instantiated, is left to the SMT solver. This enables us to employ our approach for various instantiation techniques [14, 35, 27]. This contrasts with existing research that always instantiates *all* active quantifiers and tries to modify *one specific instantiation strategy* with an ML model [28, 32] — and it is limited to that particular instantiation strategy. Also, in contrast to the existing work, we showed that we can boost the solver's performance if we take a portfolio of its configurations. This is a tall order. Indeed, it is highly challenging to improve one particular configuration of the solver in a way that is not covered by another configuration of the solver. Thanks to the versatility of our approach, we could integrate it with various instantiation strategies, thereby enhancing multiple solver configurations and achieving a notable **21.4% improvement over the baseline state-of-the-art portfolio**. The key aspects contributing to this success include our ability to implement our selection method with **different instantiation modules** and effectively **transfer learned knowledge** across strategies. This enables us to substantially enhance each individual strategy within the portfolio, consequently boosting the overall portfolio performance. Our best strategy solves 1,781 Mizar holdout problems in 60 s while it solves 1,743 in 30 s. This number can be informatively compared with experiments on the same dataset from the literature [17], where the state-of-the-art ATP prover E with advanced machine learning methods, solves only 1,632 of the holdout problems in 30 s.

Our approach differs from existing research that aimed to learn which quantifiers to select on a single problem instance [26] — using the multi-armed bandit paradigm (MAB). While previous attempts failed to achieve experimental improvements, combining MAB with offline training presents an intriguing avenue for future exploration.

Another research direction is to apply and assess our methodology on various benchmarks. We conducted evaluations on Mizar MML problems due to their similarity and relevance, enhancing the potential for machine learning methods to recognize useful quantified formulas. We anticipate similar outcomes on related problems, such as those from Isabelle/Sledgehammer [9], since learning-based methods typically transfer between various ITP corpora [18, 25].

More diverse benchmarks like TPTP [40] or SMT-LIB [4] present significantly greater challenges. These benchmarks contain problems from various sources, often with few related instances, limiting learning opportunities. Furthermore, the transfer of learned knowledge across different SMT logics complicates the situation with SMT-LIB. It is remarkable how much can be gained using the minimal bag-of-words features employed here, yet we anticipate enhanced results with more sophisticated formula features [22].

Acknowledgements

Supported by the Czech MEYS under the ERC CZ project no. LL1902 *POSTMAN*, Amazon Research Awards, EU ICT-48 2020 project no. 952215 *TAILOR*, ERC PoC grant no. 101156734 *FormalWeb3*, by the EU under the project *ROBOPROX* (reg. no. CZ.02.01.01/00/22_008/0004590), by the Czech Science Foundation project no. 24-12759S, by the *RICAIP* project that has received funding from the EU's Horizon 2020 research and innovation programme under grant agreement No 857306, and by by the HIM trimester "*Prospects of formal Mathematics*" under Germany's Excellence Strategy – EXC-2047/1 – 390685813.

References

[1] J. Alama, T. Heskes, D. Kühlwein, E. Tsivtsivadze, and J. Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014. ISSN 0168-7433. doi: 10.1007/s10817-013-9286-5.

[2] G. Bancerek, C. Bylinski, A. Grabowski, A. Kornilowicz, R. Matuszewski, A. Naumowicz, and K. Pak. The role of the Mizar Mathematical Library for interactive proof development in Mizar. *J. Autom. Reason.*, 61(1-4):9–32, 2018.

[3] 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, and Y. Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS (1)*, volume 13243. Springer, 2022.

[4] C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[5] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Satisfiability modulo theories. In A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009. ISBN 978-1-58603-929-5.

[6] F. Bártek, K. Chvalovský, and M. Suda. Regularization in spider-style strategy discovery and schedule construction. In *IJCAR (1)*, volume 14739 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2024.

[7] N. S. Bjørner and M. Janota. Playing with quantified satisfaction. In *LPAR*, pages 15–27. EasyChair, 2015.

[8] L. Blaauwbroek, D. M. Cerna, T. Gauthier, J. Jakubuv, C. Kaliszyk, M. Suda, and J. Urban. Learning guided automated reasoning: A brief survey. In *Logics and Type Systems in Theory and Practice*, volume 14560 of *Lecture Notes in Computer Science*, pages 54–83. Springer, 2024.

[9] S. Böhme and T. Nipkow. Sledgehammer: Judgement Day. In J. Giesl and R. Hähnle, editors, *IJCAR*, volume 6173 of *LNCS*, pages 107–121. Springer, 2010. ISBN 978-3-642-14202-4.

[10] K. Chvalovský, K. Korovin, J. Piepenbrock, and J. Urban. Guiding an instantiation prover with graph neural networks. In *LPAR*, volume 94 of *EPiC Series in Computing*, pages 112–123. EasyChair, 2023.

[11] V. Chvatal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979. ISSN 0364765X, 15265471. URL http://www.jstor.org/stable/3689577.

[12] E. M. Clarke, O. Grumberg, D. Kroening, D. A. Peled, and H. Veith. *Model checking, 2nd Edition*. MIT Press, 2018. ISBN 978-0-262-03883-6.

[13] J. H. Davenport, Y. Siret, and E. Tournier. *Computer algebra - systems and algorithms for algebraic computation (2. ed.)*. Academic Press, 1993. ISBN 978-0-12-204232-4.

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

[15] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. ISBN 0-7167-1044-7.

[16] Y. Ge and L. M. de Moura. Complete instantiation for quantified formulas in satisfiability modulo theories. In *Computer Aided Verification, 21st International Conference, CAV*, pages 306–320, 2009.

[17] Z. A. Goertzel, K. Chvalovský, J. Jakubův, M. Olsák, and J. Urban. Fast and slow Enigmas and parental guidance. In *FroCoS*, volume 12941 of *Lecture Notes in Computer Science*, pages 173–191. Springer, 2021.

[18] Z. A. Goertzel, J. Jakubův, C. Kaliszyk, M. Olsák, J. Piepenbrock, and J. Urban. The Isabelle ENIGMA. In *ITP*, volume 237 of *LIPIcs*, pages 16:1–16:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[19] E. K. Holden and K. Korovin. Graph sequence learning for premise selection. *CoRR*, abs/2303.15642, 2023.

[20] J. Hůla, J. Jakubův, M. Janota, and L. Kubej. Targeted configuration of an SMT solver. In *CICM*, volume 13467 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2022.

[21] J. Jakubův and J. Urban. ENIGMA: efficient learning-based inference guiding machine. In *CICM*, volume 10383 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 2017.

[22] J. Jakubův and J. Urban. Enhancing ENIGMA given clause guidance. In *CICM*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018.

[23] J. Jakubův and J. Urban. Hammering Mizar by learning clause guidance. In *ITP*, volume 141 of *LIPIcs*, pages 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[24] J. Jakubův, K. Chvalovský, M. Olsák, B. Piotrowski, M. Suda, and J. Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*, pages 448–463. Springer, 2020.

[25] J. Jakubův, K. Chvalovský, Z. A. Goertzel, C. Kaliszyk, M. Olsák, B. Piotrowski, S. Schulz, M. Suda, and J. Urban. MizAR 60 for Mizar 50. In *ITP*, volume 268 of *LIPIcs*, pages 19:1–19:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

[26] J. Jakubův, M. Janota, B. Piotrowski, J. Piepenbrock, and A. Reynolds. Selecting quantifiers for instantiation in SMT. In *SMT*, volume 3429 of *CEUR Workshop Proceedings*, pages 71–77. CEUR-WS.org, 2023.

[27] M. Janota, H. Barbosa, P. Fontaine, and A. Reynolds. Fair and adventurous enumeration of quantifier instantiations. In *Formal Methods in Computer-Aided Design*, 2021.

[28] M. Janota, J. Piepenbrock, and B. Piotrowski. Towards learning quantifier instantiation in SMT. In *SAT*, volume 236 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

[29] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, pages 3146–3154, 2017.

[30] J. C. López-Hernández and K. Korovin. An abstraction-refinement framework for reasoning with large theories. In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 663–679. Springer, 2018.

[31] A. Niemetz, M. Preiner, A. Reynolds, C. W. Barrett, and C. Tinelli. Syntax-guided quantifier instantiation. In *TACAS (2)*, volume 12652 of *Lecture Notes in Computer Science*, pages 145–163. Springer, 2021.

[32] D. E. Ouraou, P. Fontaine, and C. Kaliszyk. Machine learning for instance selection in SMT solving, 2019.

[33] M. Rawson and G. Reger. A neurally-guided, parallel theorem prover. In *FroCos*, volume 11715 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2019.

[34] M. Rawson and G. Reger. lazyCoP: Lazy paramodulation meets neurally guided search. In *TABLEAUX*, volume 12842 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2021.

[35] A. Reynolds, C. Tinelli, A. Goel, S. Krstic, M. Deters, and C. W. Barrett. Quantifier instantiation techniques for finite model finding in SMT. In *CADE*, volume 7898 of *Lecture Notes in Computer Science*, pages 377–391. Springer, 2013.

[36] A. Reynolds, C. Tinelli, and L. M. de Moura. Finding conflicting instances of quantified formulas in SMT. In *Formal Methods in Computer-Aided Design, FMCAD*, pages 195–202. IEEE, 2014.

[37] A. Reynolds, T. King, and V. Kuncak. Solving quantified linear arithmetic by counterexample-guided instantiation. *Formal Methods Syst. Des.*, 51(3):500–532, 2017.

[38] A. Reynolds, H. Barbosa, and P. Fontaine. Revisiting enumerative instantiation. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 10806, pages 112–131, 2018.

[39] M. Suda. Vampire with a brain is a good ITP hammer. In *FroCoS*, volume 12941 of *Lecture Notes in Computer Science*, pages 192–209. Springer, 2021.

[40] G. Sutcliffe, C. B. Suttner, and T. Yemenis. The TPTP problem library. In *CADE*, volume 814 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 1994.

[41] J. Urban. Translating Mizar for First Order Theorem Provers. In A. Asperti, B. Buchberger, and J. Davenport, editors, *Proceedings of the 2nd International Conference on Mathematical Knowledge Management*, number 2594 in LNCS, pages 203–215. Springer, 2003.

[42] J. Urban. MPTP – Motivation, Implementation, First Experiments. *J. Autom. Reasoning*, 33(3-4):319–339, 2004. doi: 10.1007/s10817-004-6245-1.