Solving Hard Mizar Problems with Instantiation and Strategy Invention

Jan Jakubův[1,2], Mikoláš Janota[1], and Josef Urban[1]

[1] Czech Technical University in Prague, Czechia
[2] University of Innsbruck, Austria

Abstract. In this work, we prove over 3000 previously ATP-unproved Mizar/MPTP problems by using several ATP and AI methods, raising the number of ATP-solved Mizar problems from 75% to above 80%. First, we start to experiment with the cvc5 SMT solver which uses several instantiation-based heuristics that differ from the superposition-based systems, that were previously applied to Mizar, and add many new solutions. Then we use automated strategy invention to develop cvc5 strategies that largely improve cvc5's performance on the hard problems. In particular, the best invented strategy solves over 14% more problems than the best previously available cvc5 strategy. We also show that different clausification methods have a high impact on such instantiation-based methods, again producing many new solutions. In total, the methods solve 3,021 (21.3%) of the 14,163 previously unsolved hard Mizar problems. This is a new milestone over the Mizar large-theory benchmark and a large strengthening of the hammer methods for Mizar.

1 Introduction: Mizar, ATPs, Hammers

The Mizar Mathematical Library (MML) [1] is one of the earliest large libraries of formal mathematics, containing a wide selection of lemmas and theorems from various areas of mathematics. The MML and the Mizar system [35,2,19] has been used as a source of automated theorem proving (ATP) [46] problems for over 25 years, starting with the export of several Mizar articles done by the ILF system [11,10]. Since 2003, the MPTP system [52,53] has been used to export the MML in the DFG [20] and later TPTP [51] formats. In the earliest (2003) ATP experiments over the whole library, state-of-the-art ATPs could prove about 40% of these problems when their premises were limited to those used in the human-written Mizar proofs (the so called *bushy*[3], i.e., easier, mode).

Since 2013, a fixed version of the MML (1147) and MPTP consisting of 57880 problems has been used as a large benchmark for ATPs and related hammer [7] (large-theory) methods over Mizar [41,28,50,42,21,9]. When using many ATP and premise-selection methods, 56.2% of the problems could be proved in [30]. This was recently raised to 75.5% [24], mainly by using the learning-guided E [48] (ENIGMA [27,17]) and Vampire [33] (Deepire [49]) systems.

[3] https://tptp.org/MPTPChallenge

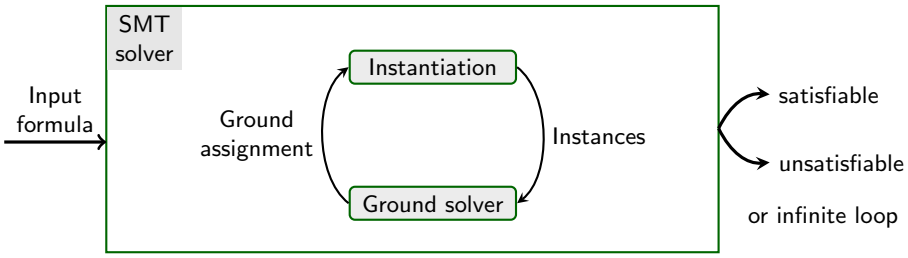

Fig. 1. Schematic of an SMT solver with quantifier instantiation.

Both E and Vampire are mainly saturation-style superposition systems. In the recent years, instantiation-based systems and satisfiability modulo theories (SMT) solvers such as cvc5 [3], iProver [32] and Z3 [12] are however becoming competitive even for problems that do not contain explicit theories in the SMT sense [6,13,18]. The problems that they solve are often complementary to those solved by the superposition-based systems.

1.1 Contributions

In this work, we use instantiation-based methods (Section 2) to solve automatically as many hard Mizar problems as possible. Our main result is that the set of ATP-provable MPTP problems has been increased by over 3,000, from 75.5% to 80.7%. All these problems are proved by the cvc5 system which we improve in several ways. First, we use the Grackle system [22] (Section 3) to automatically invent stronger strategies for MPTP (Section 4.3). Our best strategy outperforms the previously best cvc5 strategy by 14% and our best 7-strategy portfolio solves 8.8% more problems than the corresponding CASC portfolio (Section 4.4). We also combine strategy development with alternative clausification methods. This turns out to have a surprisingly high impact on the instantiation-based system, contributing many new solutions (Section 4.5). Finally, we obtain further solutions by modifying the problems with premise selection (Section 4.6). Ultimately, these methods double the number of the previously ATP-unproved Mizar problems solved by cvc5 from 1,534 to 3,021. In this context, we consider a problem proved if there is at least one system that can solve it. We show that the methods extend to previously unseen Mizar problems coming from newly added articled in a new version of MML (Section 4.7). We analyze the invented strategies (Section 5) and discuss several hard Mizar problems proved by them (Section 6).

2 Instantiation-Based Methods

In contrast to saturation-style superposition systems, SMT solvers, namely cvc5, tackle quantifiers by *instantiations*, which can be seen as a direct application of

the Herbrand's theorem. A subformula $(\forall x_1 \dots x_n\, \phi)$ produces lemmas of the form $(\forall x_1 \dots x_n\, \phi) \rightarrow \phi[x_1/t_1, \dots, x_n/t_n]$, with ϕ quantifier-free and 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.

This approach consists of a loop alternating between a *ground solver* and an *instantiation module* (Figure 1), where the ground solver perceives quantified formulas 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, strengthening the grounded part of the formula. 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 instantiation methods. For decidable fragments, dedicated approaches exist, such as bit-vectors or linear arithmetic [44,15,38,5]. Some of those can be seen as syntactic-driven approaches, *e-matching* [14,36] or syntax-guided instantiation [39]. Other methods are semantic-driven such as *model-based* [16] or *conflict-based* [45]. A straightforward, but complete, approach for FOL is *enumerative instantiation* [29,43], which exhaustively goes through all ground terms in the ground part. Instantiation itself can also be guided by ML methods [47].

3 Grackle: Targeted Strategy Invention for cvc5

Grackle [22] is a system for the automated invention of a portfolio of solver strategies targeted to selected benchmark problems. A user provides a set of benchmark problems and Grackle can automatically discover a set of diverse solver strategies that maximize the number of solved benchmark problems. Grackle supports the invention of good-performing strategies for several solvers, including ATP solvers E [48], Vampire [33], Lash [8], and an SMT solver Bitwuzla [37]. Support for additional solvers can be easily added by providing a parametrization of the solver strategy space, and by implementing a simple wrapper to launch the solver. In this paper, we extend Grackle to support an SMT solver cvc5 [3], and we evaluate its capabilities on a first-order translation of Mizar problems.

Grackle is a successor of BliStr [54], with which Grackle shares the core of the strategy invention algorithm. Grackle, however, generalizes the algorithm for an arbitrary solver. BliStr/Grackle starts with user-provided solver strategies and interleaves a *strategy evaluation* with a *strategy invention* phase. During the strategy evaluation phase, all available strategies are evaluated on all benchmark problems, typically with some higher resource limit T. This evaluation partitions the benchmark problems by individual strategy performance, giving us, for each strategy S, the set of problems P_S where S performs best. The best strategy S is then *specialized* on problems P_S in the follow-up strategy invention phase in order to search for a strategy S' with an increased performance on P_S. This is achieved by launching an external parameter tuning software, like ParamILS [23] or SMAC3 [34], on problems P_S with the strategy S as the initial starting point.

Moreover, a lower resource limit t than in the evaluation phase (T) is imposed on the solver during the tuning in order to guide the tuner towards an improved performance on P_S. The core idea, verified in previous research [54,26,25,22], is that improved performance on P_S will bring about an improvement on other not-yet-solved problems as well. A new evaluation phase then proceeds with the extended portfolio. Grackle has been extensively described [22] and we refer the reader therein for a detailed exposition.

To use cvc5 with Grackle requires providing a parametrization of the cvc5 strategy space. A strategy for cvc5 is specified as command line options and their values. While cvc5 supports more than 400 different options, we select all options with non-numeric values relevant to problems in the theory of uninterpreted functions (UF) with quantifiers. This choice is guided by our indented application on the Mizar benchmark problems which are expressed in the UF theory with a large number of quantified formulae. The cvc5 solver divides its options between *regular* and *expert*. Hence we construct two parametrizations of cvc5 strategy space, one smaller with the regular options only, and the second one with both regular and expert options. The regular parametrization has 98 parameters and the strategy space covers about 10^{35} different strategies, while the full parametrization has 168 parameters and the space size is about 10^{58}. As an exception, one of the expert options, namely `--cbqi-vo-exp`, was used also in the regular strategy space, to accommodate all the options from the CASC strategies in both spaces. We automatically extract all the options and their values from cvc5's source files decision_options.toml, prop_options.toml, quantifiers_options.toml, smt_options.toml, and uf_options.toml.[4]

Grackle additionally allows to express dependencies among options and thus to describe options that are effective only under specific settings of another option. We automatically construct some dependencies from common prefixes of option names, for example, the option `--cbqi-mode` is applicable only when the option `--cbqi` is turned on. While many of the dependencies might be left unspecified, and while many of the options might be unrelated to our benchmark problems, we leave this problem to Grackle and to the underlying parameter tuner to deal with. In this way, we also test Grackle's abilities to deal with redundancies in the strategy space. The cvc5 strategy space for Grackle can be found in the Grackle repository.[5]

4 Experiments

4.1 Dataset

Our goal is to prove as many of the remaining ATP-unproved MPTP problems as possible. Of the 57,880 problems, 43,717 have been proved[6] in total in the previous experiments [24,30], thus, 14,163 problems remain to be proved. Our strategy

[4] https://github.com/cvc5/cvc5/tree/cvc5-1.1.1/src/options
[5] https://github.com/ai4reason/grackle/tree/v0.2/grackle/trainer/cvc5
[6] https://github.com/ai4reason/ATP_Proofs

invention methods work by gradually developing strategies that are faster and faster on solvable problems. That is why we extend the set of the 14,163 ATP-unproved problems by another 4,283 hard problems that were proved only in the latest stages of the previous ATP experiments. We will use their versions with heuristically minimized premises (using *subproblem based minimization* [24]) to increase the chances of the ATP systems. We also remove from this set 1,585 problems for which the minimization was not done yet.[7] This results in a set of 16,861 hard problems on which we develop our strategies. These problems are by default in the FOF format. We denote them min_{fof} below. Later on (Section 4.5), we apply different clausifications to them. In Section 4.6 we additionally experiment with different premise selections for them.

4.2 Overview of the Experiments

Most of the experiments in this paper are conducted on the set of 16,861 hard Mizar problems described in Section 4.1. Section 4.3 focuses solely on describing three Grackle runs performed to develop a robust portfolio of cvc5 strategies specialized for Mizar problems. All strategies are evaluated with a time limit of 30 seconds. Since increasing the time limit can still yield significant improvements, selected strategies are evaluated in Section 4.4 with a higher time limit, namely 600 seconds. Sections 4.3 and 4.4 use the same version of problems (min_{fof}) and differ only in the time limit. In Section 4.5, we explore different clausification methods, and in Section 4.6, we investigate various premise selection methods. This implies that Sections 4.5 and 4.6 use syntactically different but semantically equivalent versions of Mizar problems. On the other hand, Section 4.7 attempts to assess the overfitting of Grackle-invented strategies on a new version of MML, thus the numbers reported therein are not directly comparable with those in previous sections since the problem sets differ.

4.3 Experiments with Grackle Strategy Invention

We evaluate[8] the Grackle's ability to invent good-performing strategies for cvc5 on the min_{fof} benchmark. As a baseline, we consider all 16 strategies used in the cvc5's CASC competition script (see Table 5). We evaluate these 16 strategies with a 30-second time limit per strategy and problem. The best strategy solves 2,508 problems, while all the strategies together solve 3,460 problems. The two most complementary strategies are used as the initial portfolio for the first Grackle run.

We perform consequently three Grackle runs, each with an overall timeout of 7 days. Grackle terminates when all strategies have been already specialized, or when the time is exhausted. All three runs were terminated by timeout.

[7] These are the non-theorem Mizar toplevel lemmas, for which the subproblem look-up (and thus also minimization) is more challenging.

[8] On two AMD EPYC 7513 32-Core processors @ 3680 MHz and with 514 GB RAM.

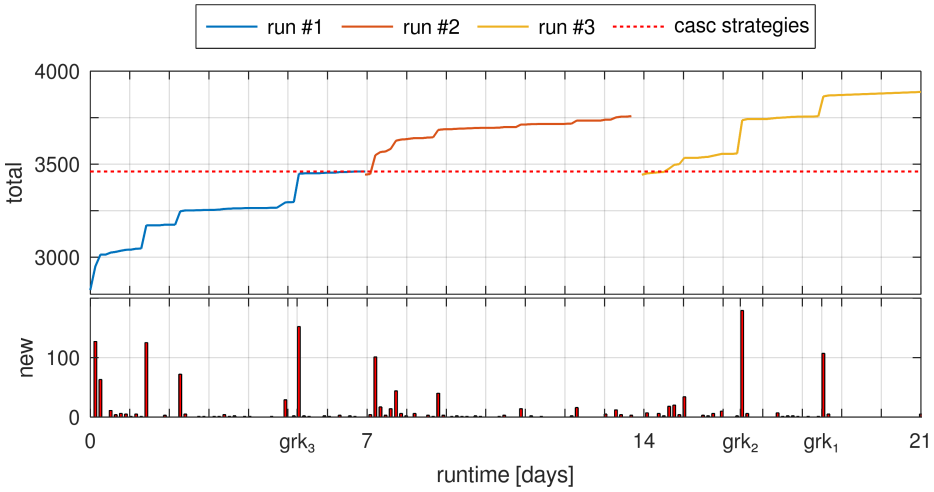

Fig. 2. Time progress of solved problems in Grackle runs.

(run #1) The first Grackle run starts with the two most complementary CASC strategies and uses the *regular* strategy space (see Section 3). The first run terminated after 7 days with 50 new strategies solving together 3,459 problems with a 30-second time limit per strategy and problem. Out of these problems, 345 are not solved by any of the 16 baseline CASC strategies.

(run #2) The second run uses the same regular strategy space as run #1 but it starts from the best 6 strategies found in the first Grackle run. These initial strategies solve 3,425 problems and Grackle invented 45 new strategies solving together 3,696 problems with 485 unsolved by baseline strategies.

(run #3) The third run uses the same setup as run #2 but it uses the full strategy space instead of the regular one. Grackle invented 48 new strategies solving together 3,856 problems with 629 unsolved by baseline strategies.

The time progress of the Grackle runs is visualized in Figure 2. The lower part (*new*) shows the number of new problems solved by the strategy invented at that time, while the upper part (*total*) shows the total number of problems solved. The red dotted line marks the performance of the 16 baseline CASC strategies. The x-axis additionally shows the time of invention of the best three strategies (grk_i).

The figure shows that the expert options added in run #3 helped to develop stronger strategies and to improve the results. Strong strategies are sometimes invented after several days of stagnation. Grackle invented 143 strategies, which together solve 4,113 problems. The best 16 Grackle strategies solve 4,039, which increases the number of 3,460 problems solved by the 16 baseline strategies by 16.7%. The best single strategy solves 2,796, which improves on the best baseline strategy (which solves 2,508) by 11.5%.

run	*solved*				*single*	*strategies*			*specializations*	
	initial	*final*	*new*	*casc+*	*best*	*initial*	*new*	*needed*	*total*	*failed*
#1	2823	3459	+636	+345	2696	2	50	28	56	6
#2	3425	3696	+271	+482	2696	6	45	27	56	11
#3	3425	3856	+431	+629	2796	6	48	29	58	10

Table 1. Grackle strategy invention for cvc5 on Mizar problems (Section 4.3).

Additional data from the experiments are depicted in Table 1. Columns *solved* describe the initial and the final count of solved problems together with the improvement over the initial strategies (*new*), and over the 16 baseline CASC strategies (*casc+*). The column *single best* states the number of problems solved by the best invented strategy. Columns *strategies* describe the *initial* count of strategies, the count of *new* strategies, and how many strategies are needed to cover the *final* number of solved problems (the length of the greedy cover sequence). Columns *specializations* provide the number of attempted specializations (*total*) and *failed* specializations where the output strategy was already known.

4.4 Experiments with Higher Time Limits

The above (Section 4.3) strategies were evaluated in a 30 s time limit. To solve more Mizar problems we proceed by evaluating the best strategies with a higher time limit of 600 s. Evaluation of a single strategy with this time limit takes about 20 hours. Hence, we evaluate strategies selectively as follows. We start with the strategies evaluated in 30 s and construct their greedy cover. The best 30 s strategy is then evaluated in 600 s. The newly evaluated strategy is then added to the greedy cover and the process is iterated until new problems are being proved. The best Grackle strategy grk_1 solves 3,496 problems, while the best CASC strategy solves 3,059. This is a 14.3% improvement. We ended up with 7 Grackle strategies and with 7 CASC strategies evaluated in 600s solving together 4,653 problems (4,398 by Grackles and 4,043 by CASCs). Together with the strategies evaluated in the lower time limit we solved altogether 5,035 of the benchmark problems at this point.

This experiment with higher time limits also shows an interesting difference in behavior between saturation-based ATP provers and instantiation-based SMT solvers. While SMT solvers seem to benefit significantly from higher time limits, ATP solvers typically benefit much less from it. For example, in the comparable single-strategy setting, cvc5 solves almost 50% more problems when the time limit is increased from 60 to 600 seconds (see columns *alone* of grk_1 in version bushy in Table 2 and Table 4). However, E Prover in the *auto* mode solves only 11% more problems on the same benchmark problems with the same increase of the time limit. A similar relative performance gap is observed across different benchmarks.

4.5 Experiments with Clausification Methods

The Mizar problems are given as TPTP [51] problems in first-order logic (FOF). For cvc5 we translate them to the SMT2 language [4] in the theory of uninterpreted functions (UF). By default, cvc5 converts to clausal normal form (CNF) internally but since instantiation-based heuristics seem sensitive to problem reformulation, we also experiment with external clausification. This gives us syntactically different variants of the problems and we can test whether cvc5 benefits from such alternative ways of clausification.

We use E as the external clausifier and we construct two more problem variants min_{cnf1} and min_{cnf2}. The min_{cnf1} version is produced by using E's default clausification parameters, while for min_{cnf2} we use a much more aggressive introduction of definitions for frequent subformulas. In particular, this is achieved by the E Prover's option `--definitional-cnf` with values 24 (default) and 4, respectively. The value indicates how many times a subformula needs to appear for a new definition to be introduced via a new constant. The min_{cnf2} methods compared to min_{cnf1} almost halve the average number of literals in the problems (368.3 vs 668.2) and the average number of symbols drops to 60% (2,512.7 vs 4,124.1).

The best Grackle strategy then solves 3,231 problems in 600 s compared to 3,125 by the best CASC strategy. Both these results use the min_{cnf1} clausification. While the individual performance of the strategies on the externally clausified problems is lower than on the FOF variants, they are indeed highly complementary. Eight Grackle strategies and six CASC strategies evaluated in 600 s increase the number of the solved hard problems from 5,035 to 5,404.

4.6 Experiments on Premise Selection Slices

Based on the success with such problem reformulation, we perform additional experiments, this time with different premise selection methods developed in our prior work [24]. Namely, we evaluate Grackle and baseline strategies on the bushy (i.e., not subproblem-minimized) variants of the problems, on the strongest GNN (graph neural network [40]) premise selection slices with the threshold -1 (denoted here gnn), and on LightGBM [31] premise selection slices with the threshold 0.1 (lgbm). These variants were found complementary in our previous experiments [24]. In a nutshell, the trained GNN puts at all the available premises into a large graph with the edges going between formulas, terms, subterms and symbols, runs several iterations of a *message passing* algorithm in this large graph, and ultimately uses the aggregated information for deciding which of the premises are relevant for the conjecture. The LightGBM instead trains many decision trees on suitable features characterizing the premises to determine their relevance. These methods work quite differently and also sometimes recommend premises that are quite different from the ones used by human formalizers.

We again evaluate our strategies on such problems, first with a lower (60 s) time limit. Table 2 shows a comparative evaluation of the 60 s Grackle and CASC

version	*strat*	*addon*		*total*	*alone*	*new*
min_{fof}	grk_1	+3034	-	3034	3034	968
gnn	grk_1	+521	+17.17%	3555	1024	336
min_{fof}	grk_3	+486	+13.67%	4041	2828	887
lgbm	grk_1	+264	+6.53%	4305	1260	405
min_{fof}	grk_2	+187	+4.34%	4492	2772	904
bushy	grk_2	+177	+3.94%	4669	963	329
min_{fof}	casc_{10}	+73	+1.56%	4742	2175	598
min_{fof}	casc_{13}	+58	+1.22%	4800	2348	666
gnn	grk_3	+46	+0.96%	4846	930	280
lgbm	grk_2	+30	+0.62%	4876	1183	384
min_{fof}	casc_{14}	+28	+0.57%	4904	2650	828
bushy	casc_{13}	+25	+0.51%	4929	795	239
lgbm	grk_3	+24	+0.49%	4953	1074	313
gnn	grk_2	+23	+0.46%	4976	1000	336
bushy	casc_{14}	+17	+0.34%	4993	913	302
bushy	casc_{10}	+11	+0.22%	5004	609	188
bushy	grk_1	+9	+0.18%	5013	962	319
bushy	grk_3	+8	+0.16%	5021	746	195
gnn	casc_{13}	+7	+0.14%	5028	899	287
lgbm	casc_{10}	+7	+0.14%	5035	895	262
gnn	casc_{14}	+6	+0.12%	5041	954	313
lgbm	casc_{13}	+5	+0.10%	5046	1051	309
lgbm	casc_{14}	+4	+0.08%	5050	1137	363

Table 2. Full greedy cover on FOF slices min_{fof}, bushy, gnn, and lgbm with 60s timeout.

strategies on these slices. In the table, the column *version* displays the benchmark version and the column *strat* is the strategy name. The column *addon* describes the addition of the strategy to the portfolio, that is, it lists the number of problems added, and the same in percents. The column *total* lists the cumulative performance of the portfolio up to that line. Finally, the column *alone* shows the individual performance of the strategy and the column *new* shows the number of Mizar problems unproved in our previous research [24,30]. The specification of Grackle-invented strategies (grk_i) can be found in Table 6 while the definitions of CASC strategies is in Table 5. These strategies are further analyzed in Section 5.

Based on this, we evaluate the best Grackle strategy grk_1 on all three slices in 600 s. This alone raises the number of solved problems from 5,404 to 6,363. After adding also the 60 s results, we obtain in total 6,469 hard problems solved, of which **3,021** were not proved by ATPs before.[9] This is our main result. We have proved **21.3%** of the remaining ATP-unproved problems, and increased the total number of *all* ATP-proved Mizar problems to 46,738 (**80.7** %). About half of the 3,021 problems (1,534) can be solved by the cvc5 CASC strategies. For

[9] The lists of problems solved by the individual strategies and the strategy definitions are available at https://github.com/ai4reason/cvc5_grackle_mizar.

Results on MML				Transfer to new MML			
version	*strat*	*addon*	*alone*	*version*	*strat*	*addon*	*alone*
min_{fof}	grk_1	+3496	3496	*cnf1*	grk_2	+4861	4861
min_{cnf1}	grk_2	+738	3231	*fof*	grk_1	+433	4541
gnn	grk_1	+535	1215	*cnf1*	grk_3	+164	4495
bushy	grk_1	+311	1441	*fof*	casc_{13}	+78	4406
min_{fof}	grk_3	+298	3220	*fof*	grk_3	+53	4195
lgbm	grk_1	+233	1512	*fof*	grk_2	+39	4418
min_{cnf1}	grk_3	+161	3223	*cnf1*	grk_1	+33	4811
min_{cnf1}	casc_{10}	+112	3125	*cnf1*	casc_{10}	+17	4211
min_{fof}	grk_2	+90	3146	*cnf2*	grk_1	+14	4417
min_{cnf2}	grk_2	+62	2949	*fof*	casc_{10}	+12	3952

Table 3. Results on MML (left), transfer to new MML (right).

the remaining half, some of our methods (new strategies, different clausifications or premise slices) are necessary.

The first 10 strategies in the final greedy cover are shown in Table 3 (left). The meaning of the columns is the same as in Table 2, that is, the column *version* displays benchmark version, the column *strat* is the strategy name, the column *addon* describes the addition of the strategy to the portfolio, and *alone* shows the individual performance of the strategy. We can see that the Grackle-invented strategies clearly dominate the greedy cover. While premise selection slices gnn, lgbm, and bushy exhibit low individual performance, they provide many new solutions. This is often due to the alternative proofs proposed by the premise selection methods trained over many previous proofs.

For the sake of completeness, Table 4 additionally presents extended results. The table mixes strategies evaluated with different time limits denoted in the column *timeout*. The meaning of other columns is the same as in Table 2. All Grackle and CASC strategies solve together 6,363 Mizar problems.[10]

4.7 Transfer to New MML

To assess the overfitting of the methods we evaluate the best three Grackle and the best three CASC strategies on one more benchmark. We use 13,370 bushy problems coming from newly added articles in MML version 1382. Table 3 (right) shows the results. The Grackle strategies outperform all CASC strategies, even though the improvement is smaller than on the MML problems they were developed for. Alternative clausification methods again provide a considerable improvement.

[10] The strategies not listed in Table 6 (like $\mathsf{grk}_{\mathtt{169baa}}$) can be found in our repository (Note 9).

version	*timeout*	*strat*	*addon*		*total*	*alone*	*new*
min_{fof}	600	grk_{1}	+3496	-	3496	3496	1243
min_{cnf1}	600	grk_{2}	+738	+21.11%	4234	3231	1192
gnn	600	grk_{1}	+535	+12.64%	4769	1215	432
bushy	600	grk_{1}	+311	+6.52%	5080	1441	553
min_{fof}	600	grk_{3}	+298	+5.87%	5378	3220	1146
lgbm	600	grk_{1}	+233	+4.33%	5611	1512	541
min_{cnf1}	600	grk_{3}	+161	+2.87%	5772	3223	1092
min_{cnf1}	600	casc_{10}	+112	+1.94%	5884	3125	999
min_{fof}	600	grk_{2}	+90	+1.53%	5974	3146	1131
min_{cnf2}	600	grk_{2}	+62	+1.04%	6036	2949	1045
min_{fof}	600	grk_{5}	+49	+0.81%	6085	3086	1063
min_{cnf1}	600	grk_{1}	+35	+0.58%	6120	3163	1110
min_{cnf2}	600	grk_{5}	+31	+0.51%	6151	2909	1030
min_{cnf1}	600	grk_{5}	+27	+0.44%	6178	3113	1099
min_{cnf2}	600	grk_{3}	+22	+0.36%	6200	2851	934
min_{fof}	600	casc_{13}	+16	+0.26%	6216	2711	848
min_{cnf2}	600	casc_{10}	+14	+0.23%	6230	2695	787
min_{fof}	600	casc_{10}	+13	+0.21%	6243	2575	795
min_{fof}	600	$\mathsf{grk}_{\mathtt{169baa}}$	+12	+0.19%	6255	2993	722
min_{cnf1}	600	casc_{06}	+12	+0.19%	6267	2334	1002
min_{fof}	600	casc_{09}	+11	+0.18%	6278	1064	150
min_{fof}	600	casc_{14}	+10	+0.16%	6288	3059	1057
min_{fof}	30	$\mathsf{grk}_{\mathtt{d73c5e}}$	+9	+0.14%	6297	2901	986
min_{fof}	600	casc_{06}	+8	+0.13%	6305	2380	716
min_{fof}	30	$\mathsf{grk}_{\mathtt{393769}}$	+7	+0.11%	6312	2671	803
min_{cnf1}	600	casc_{13}	+7	+0.11%	6319	2948	977
min_{fof}	600	casc_{07}	+5	+0.08%	6324	2955	916
min_{fof}	600	casc_{16}	+5	+0.08%	6329	2976	885
min_{fof}	30	$\mathsf{grk}_{\mathtt{1fe2d9}}$	+5	+0.08%	6334	2770	992
min_{cnf2}	600	casc_{13}	+5	+0.08%	6339	2726	968
min_{fof}	30	$\mathsf{grk}_{\mathtt{014565}}$	+3	+0.05%	6342	2666	849
min_{fof}	30	$\mathsf{grk}_{\mathtt{043c34}}$	+3	+0.05%	6345	2544	769
min_{cnf2}	600	casc_{06}	+3	+0.05%	6348	2090	631
min_{cnf2}	600	grk_{1}	+3	+0.05%	6351	2817	933
min_{fof}	600	grk_{4}	+2	+0.03%	6353	3320	859
min_{fof}	30	$\mathsf{grk}_{\mathtt{166bee}}$	+2	+0.03%	6355	2671	1163
min_{fof}	30	$\mathsf{grk}_{\mathtt{04f79f}}$	+1	+0.02%	6356	2484	725
min_{fof}	30	$\mathsf{grk}_{\mathtt{0f4750}}$	+1	+0.02%	6357	2465	723
min_{fof}	30	$\mathsf{grk}_{\mathtt{1499bd}}$	+1	+0.02%	6358	2238	641
min_{fof}	30	$\mathsf{grk}_{\mathtt{1afb4a}}$	+1	+0.02%	6359	463	41
min_{fof}	30	$\mathsf{grk}_{\mathtt{340075}}$	+1	+0.02%	6360	2556	742
min_{fof}	30	$\mathsf{grk}_{\mathtt{52ae2f}}$	+1	+0.02%	6361	2670	800
min_{fof}	30	$\mathsf{grk}_{\mathtt{7dac18}}$	+1	+0.02%	6362	173	6
min_{fof}	30	$\mathsf{grk}_{\mathtt{ba0f42}}$	+1	+0.02%	6363	1810	509

Table 4. Full complete greedy cover on MML problems.

name	cvc5 strategy options
casc$_1$	`--decision=internal --simplification=none --no-inst-no-entail --no-cbqi --full-saturate-quant`
casc$_2$	`--no-e-matching --full-saturate-quant`
casc$_3$	`--no-e-matching --enum-inst-sum --full-saturate-quant`
casc$_4$	`--finite-model-find --uf-ss=no-minimal`
casc$_5$	`--multi-trigger-when-single --full-saturate-quant`
casc$_6$	`--trigger-sel=max --full-saturate-quant`
casc$_7$	`--multi-trigger-when-single --multi-trigger-priority --full-saturate-quant`
casc$_8$	`--multi-trigger-cache --full-saturate-quant`
casc$_9$	`--prenex-quant=none --full-saturate-quant`
casc$_{10}$	`--enum-inst-interleave --decision=internal --full-saturate-quant`
casc$_{11}$	`--relevant-triggers --full-saturate-quant`
casc$_{12}$	`--finite-model-find --e-matching --sort-inference --uf-ss-fair`
casc$_{13}$	`--pre-skolem-quant=on --full-saturate-quant`
casc$_{14}$	`--cbqi-vo-exp --full-saturate-quant`
casc$_{15}$	`--no-cbqi --full-saturate-quant`
casc$_{16}$	`--macros-quant --macros-quant-mode=all --full-saturate-quant`

Table 5. CASC baseline strategies used in the experiments.

5 Analysis of the Invented Strategies

As usual with automated strategy invention, there are many new combinations of parameters that may require deeper analysis to understand the automatically invented behavior. That is why we make them publicly available.[11] As a baseline and as a starting point for Grackle strategy inventions, we consider 16 strategies used in the cvc5's CASC competition script.[12] The strategies are listed in Table 5. The best Grackle strategies are depicted in Table 6.

Interestingly, the different Grackle-invented strategies focus mainly on changing the behavior of the different components of the quantifier instantiation module of cvc5, cf. Section 2. By default cvc5 relies on e-matching [14,36], which is incomplete, which also means that the solver may quickly give up (return the output `unknown`). The option `--full-saturate-quant`, runs the default mode but if that fails to answer, the solver resorts to the enumerative mode (complete for FOL [43]). This explains why this option is so prevalent in the invented strategies.

[11] See Note 9.

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

name	cvc5 strategy options
grk_1	`--cbqi-vo-exp --cond-var-split-quant=agg --full-saturate-quant --relational-triggers`
grk_2	`--cbqi-vo-exp --full-saturate-quant --miniscope-quant=off --multi-trigger-priority --no-static-learning --relevant-triggers --ieval=off`
grk_3	`--full-saturate-quant --multi-trigger-priority --multi-trigger-when-single --term-db-mode=relevant`
grk_4	`--cbqi-vo-exp --cond-var-split-quant=agg --full-saturate-quant --inst-when=last-call`
grk_5	`--cbqi-all-conflict --full-saturate-quant --inst-when=full-delay --macros-quant --multi-trigger-priority --quant-dsplit=none --quant-dsplit=none --trigger-sel=min-s-all --uf-ss=none`

Table 6. Best five strategies invented by Grackle.

In grk_1 and grk_3, e-matching's behavior is changed by changing the trigger-generation policy. In grk_1 and grk_2, the option `--cbqi-vo-exp` affects the behavior of the conflict-driven instantiation [45]. The option `--cond-var-split-quant` affects the quantifier splitting policy. The option `--term-db-mode=relevant` enforces a stricter policy on ground term filtering. In general, it seems that the essence of a successful strategy is a combination of enumerative instantiations with an appropriate trigger selection for e-matching. In the next section (Section 6) we discuss the influence of such options on the solution of several hard Mizar problems.

6 Interesting Mizar Problems Proved

Since we are focusing on the 25% of the Mizar problems that have not been proved by ATPs so far, the newly solved problems are typically quite involved, with long proofs both in Mizar and in cvc5. 127 of them take more than 100 lines to prove in Mizar, and the average Mizar proof length is 41. This is one page of a proof in a paper like this.

A previously ATP-unproved problem that seems relatively easy for many of the cvc5 strategies is `KURATO_1:6`[13] related to the well-known Kuratowski's closure-complement problem.[14] The theorem shows that for any set A, its `Kurat14Set` (i.e., a family of 14 sets created by applying closure and complement operations in a particular way to A) is already closed under complement and closure:

```
definition
```

[13] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/kurato_1.html#T6
[14] https://en.wikipedia.org/wiki/Kuratowski%27s_closure-complement_problem

```
let T be non empty TopSpace;
let A be Subset of T;
func Kurat14Set A -> Subset-Family of T equals
{ A, A-, A-‘, A-‘-, A-‘-‘, A-‘-‘-, A-‘-‘-‘ } \/
{ A‘, A‘-, A‘-‘, A‘-‘-, A‘-‘-‘, A‘-‘-‘-, A‘-‘-‘-‘ };
end;

theorem Th6: for T being non empty TopSpace
for A, Q being Subset of T st Q in Kurat14Set A holds
Q‘ in Kurat14Set A & Q- in Kurat14Set A;
```

The proof has 131 lines in Mizar, however it indeed seems achievable by instantiation-based methods that gradually enumerate the applications of closure and complement to the skolems and use congruence closure when a more complex term can be shown to be equal to a less complex term. The problem is a combination of equational reasoning and a large case split (14 cases), which is what likely makes it hard for the superposition-based systems. The success may indicate that a full ATP (or AI/TP) solution of the Kuratowski's closure-complement problem may not be too far today, because proposing the `Kurat14Set` and finding automatically a suitable family of 14 distinct sets (to show that 14 is indeed the smallest number) also seems within the reach of today's systems.

The problem `ASYMPT_1:18`[15] is on the other hand only provable with a single Grackle-invented strategy grk_2 and external clausification, taking 62 s. The problem states that the functions $f(n) = n \bmod 2$ and $g(n) = n + 1 \bmod 2$ are not in the Big O relation (in any direction).

```
theorem
 for f,g being Real_Sequence st
   (for n holds f.n = n mod 2) & (for n holds g.n = n+1 mod 2)
  holds ex s,s1 being eventually-nonnegative Real_Sequence
  st s = f & s1 = g & not s in Big_Oh(s1) & not s1 in Big_Oh(s)
```

The Mizar proof has 122 lines and again goes through several case splits related to the mod 2 values. However a lot of knowledge (often equational) about the arithmetical expressions, modulo and inequality has to be applied too.[16] The fact that this can be done by an instantiation-based system is quite remarkable, and probably also due to the fact that the terms that arise in the proof are not extremely complicated thanks to the $\{0, 1\}$ codomain of the functions involved. The option `--multi-trigger-priority` seems indispensable for solving the problem, showing the importance of the heuristics for handling instantiation triggers. This may be an opportunity for further AI/ML methods learning even finer control of the triggers in such systems.

Finally, theorem `ROBBINS4:3`[17] shows an equivalent condition for ortholattices:

[15] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/asympt_1.html#T18

[16] Note that the Mizar/MPTP translation translates everything as uninterpreted functions, i.e., there is no reliance on the arithmetical theories implemented in cvc5.

[17] http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/robbins4.html#T3

```
for L being non empty OrthoLattStr holds L is Ortholattice iff
  (for a, b, c being Element of L holds
     (a "\/" b) "\/" c = (c‘ "/\" b‘)‘ "\/" a)
& (for a, b being Element of L holds a = a "/\" (a "\/" b))
& for a, b being Element of L holds a = a "\/" (b "/\" b‘)
```

The problem can only be solved by the Grackle-invented strategy 89fc24 and it takes 137 s. The Mizar proof has 145 lines and uses a lot of equational reasoning in lattice theory. It is quite surprising that a proof with so much equality could not be done by the superposition based systems, and that it can be done by cvc5. Again, triggers seem important here, together with the `--term-db-mode=relevant` option which further limits the sets of possible quantifier instantiations.

7 Conclusions and Future Work

We have solved **3,021** (21.3%) of the remaining 14,163 hard Mizar problems, raising the percentage of automatically proved Mizar problems from 75.5% to **80.7%**. This was mainly done by automatically inventing suitable instantiation-based strategies for the cvc5 solver, using our Grackle system. Further improvements were obtained by using alternative clausifications of the problems, and also alternative premise selections. Such problem transformations have a surprisingly large effect on the instantiation-based procedures and are likely to be explored further when creating strong portfolios for such systems.

The invented cvc5 strategies perform well also on a set of new problems added in a later version of the Mizar library, showing only limited overfitting. Given today's cvc5's good performance on corpora such as Isabelle/Sledgehammer and TPTP, it may be also interesting to repeat our strategy invention experiments for the TPTP problems and for problems exported from various non-Mizar hammer systems. In general, training instantiation-based systems in various ways is an emerging research topic that may bring interesting improvements to some of today's strongest ATP/SMT methods.

Acknowledgments

Supported by the Czech MEYS under the ERC CZ project no. LL1902 *POSTMAN*, by Amazon Research Awards, by EU ICT-48 2020 project no. 952215 *TAILOR*, by ERC PoC grant no. 101156734 *FormalWeb3*, by CISCO grant no. 2023-322029, and co-funded by the European Union under the project *ROBOPROX* (reg. no. CZ.02.01.01/00/22_008/0004590).

References

1. Grzegorz Bancerek, Czeslaw Bylinski, Adam Grabowski, Artur Kornilowicz, Roman Matuszewski, Adam Naumowicz, and Karol Pak. The role of the Mizar Mathematical Library for interactive proof development in Mizar. *J. Autom. Reason.*, 61(1-4):9–32, 2018.

2. Grzegorz Bancerek, Czeslaw Bylinski, Adam Grabowski, Artur Kornilowicz, Roman Matuszewski, Adam Naumowicz, Karol Pak, and Josef Urban. Mizar: State-of-the-art and beyond. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, volume 9150 of *Lecture Notes in Computer Science*, pages 261–279. Springer, 2015.
3. Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. cvc5: A versatile and industrial-strength SMT solver. In *TACAS (1)*, volume 13243. Springer, 2022.
4. Clark Barrett, Aaron Stump, Cesare Tinelli, et al. The SMT-LIB standard: Version 2.0. In *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, UK)*, volume 13, page 14, 2010.
5. Nikolaj Bjørner and Mikoláš Janota. Playing with quantified satisfaction. In *20th International Conferences on Logic for Programming, Artificial Intelligence and Reasoning – Short Presentations, LPAR*, volume 35, pages 15–27. EasyChair, 2015.
6. Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending sledgehammer with SMT solvers. *J. Autom. Reason.*, 51(1):109–128, 2013.
7. Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.
8. Chad E. Brown and Cezary Kaliszyk. Lash 1.0 (system description). In *IJCAR*, volume 13385 of *Lecture Notes in Computer Science*, pages 350–358. Springer, 2022.
9. Karel Chvalovský, Konstantin Korovin, Jelle Piepenbrock, and Josef Urban. Guiding an instantiation prover with graph neural networks. In *LPAR*, volume 94 of *EPiC Series in Computing*, pages 112–123. EasyChair, 2023.
10. Ingo Dahn. Interpretation of a Mizar-like logic in first-order logic. In Ricardo Caferra and Gernot Salzer, editors, *FTP (LNCS Selection)*, volume 1761 of *LNCS*, pages 137–151. Springer, 1998.
11. Ingo Dahn and Christoph Wernhard. First order proof problems extracted from an article in the MIZAR Mathematical Library. In Maria Paola Bonacina and Ulrich Furbach, editors, *Int. Workshop on First-Order Theorem Proving (FTP'97)*, RISC-Linz Report Series No. 97-50, pages 58–62. Johannes Kepler Universität, Linz (Austria), 1997.
12. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.
13. Martin Desharnais, Petar Vukmirovic, Jasmin Blanchette, and Makarius Wenzel. Seventeen provers under the hammer. In *ITP*, volume 237 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
14. David Detlefs, Greg Nelson, and James B. Saxe. Simplify: A theorem prover for program checking. *J. ACM*, 52(3):365–473, 2005.
15. Azadeh Farzan and Zachary Kincaid. Strategy synthesis for linear arithmetic games. *Proc. ACM Program. Lang.*, 2(POPL):61:1–61:30, 2018.
16. Yeting Ge and Leonardo Mendonça de Moura. Complete instantiation for quantified formulas in satisfiability modulo theories. In *Computer Aided Verification, 21st International Conference, CAV*, pages 306–320, 2009.

17. Zarathustra Amadeus Goertzel, Karel Chvalovský, Jan Jakubův, Miroslav Olsák, and Josef Urban. Fast and slow Enigmas and parental guidance. In *FroCoS*, volume 12941 of *Lecture Notes in Computer Science*, pages 173–191. Springer, 2021.
18. Zarathustra Amadeus Goertzel, Jan Jakubův, Cezary Kaliszyk, Miroslav Olsák, Jelle Piepenbrock, and Josef Urban. The Isabelle ENIGMA. In *ITP*, volume 237 of *LIPIcs*, pages 16:1–16:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
19. Adam Grabowski, Artur Korniłowicz, and Adam Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.
20. Reiner Hähnle, Manfred Kerber, and Christoph Weidenbach. Common syntax of the DFGSchwerpunktprogramm deduction. Technical Report TR 10/96, Fakultät für Informatik, Universität Karlsruhe, Karlsruhe, Germany, 1996.
21. Edvard K. Holden and Konstantin Korovin. Graph sequence learning for premise selection. *CoRR*, abs/2303.15642, 2023.
22. Jan Hůla, Jan Jakubův, Mikolás Janota, and Lukás Kubej. Targeted configuration of an SMT solver. In *CICM*, volume 13467 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2022.
23. Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. ParamILS: an automatic algorithm configuration framework. *J. Artificial Intelligence Research*, 36:267–306, October 2009.
24. Jan Jakubův, Karel Chvalovský, Zarathustra Amadeus Goertzel, Cezary Kaliszyk, Mirek Olsák, Bartosz Piotrowski, Stephan Schulz, Martin Suda, and Josef Urban. MizAR 60 for Mizar 50. In *ITP*, volume 268 of *LIPIcs*, pages 19:1–19:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
25. Jan Jakubův, Martin Suda, and Josef Urban. Automated invention of strategies and term orderings for vampire. In *GCAI*, volume 50 of *EPiC Series in Computing*, pages 121–133. EasyChair, 2017.
26. Jan Jakubův and Josef Urban. BliStrTune: hierarchical invention of theorem proving strategies. In Yves Bertot and Viktor Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 43–52. ACM, 2017.
27. Jan Jakubův and Josef Urban. ENIGMA: efficient learning-based inference guiding machine. In Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, editors, *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*, volume 10383 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 2017.
28. Jan Jakubův and Josef Urban. Hammering Mizar by learning clause guidance. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*, pages 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
29. Mikoláš Janota, Haniel Barbosa, Pascal Fontaine, and Andrew Reynolds. Fair and adventurous enumeration of quantifier instantiations. In *Formal Methods in Computer-Aided Design*, 2021.
30. Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Autom. Reasoning*, 55(3):245–256, 2015.
31. Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, pages 3146–3154, 2017.

32. Konstantin Korovin. iprover - an instantiation-based theorem prover for first-order logic (system description). In *IJCAR*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008.
33. Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
34. Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Difan Deng, Carolin Benjamins, Tim Ruhkopf, René Sass, and Frank Hutter. SMAC3: A versatile bayesian optimization package for hyperparameter optimization, 2021.
35. Roman Matuszewski and Piotr Rudnicki. Mizar: the first 30 years. *Mechanized Mathematics and Its Applications*, 4:3–24, 2005.
36. Michal Moskal, Jakub Lopuszanski, and Joseph R. Kiniry. E-matching for fun and profit. In Sava Krstic and Albert Oliveras, editors, *Proceedings of the 5th International Workshop on Satisfiability Modulo Theories, SMT@CAV 2007, Berlin, Germany, July 1-2, 2007*, volume 198 of *Electronic Notes in Theoretical Computer Science*, pages 19–35. Elsevier, 2007.
37. Aina Niemetz and Mathias Preiner. Bitwuzla at the SMT-COMP 2020. *CoRR*, abs/2006.01621, 2020.
38. Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Solving quantified bit-vectors using invertibility conditions. In *Computer Aided Verification – 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018*, volume 10982, pages 236–255. Springer, 2018.
39. Aina Niemetz, Mathias Preiner, Andrew Reynolds, Clark W. Barrett, and Cesare Tinelli. Syntax-guided quantifier instantiation. In *Tools and Algorithms for the Construction and Analysis of Systems – 27th International Conference, TACAS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS*, volume 12652, pages 145–163. Springer, 2021.
40. Miroslav Olsák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 1395–1402. IOS Press, 2020.
41. Michael Rawson and Giles Reger. A neurally-guided, parallel theorem prover. In *FroCos*, volume 11715 of *Lecture Notes in Computer Science*, pages 40–56. Springer, 2019.
42. Michael Rawson and Giles Reger. lazyCoP: Lazy paramodulation meets neurally guided search. In *TABLEAUX*, volume 12842 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2021.
43. Andrew Reynolds, Haniel Barbosa, and Pascal Fontaine. Revisiting enumerative instantiation. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 10806, pages 112–131, 2018.
44. Andrew Reynolds, Tim King, and Viktor Kuncak. Solving quantified linear arithmetic by counterexample-guided instantiation. *Formal Methods Syst. Des.*, 51(3):500–532, 2017.
45. Andrew Reynolds, Cesare Tinelli, and Leonardo Mendonça de Moura. Finding conflicting instances of quantified formulas in SMT. In *Formal Methods in Computer-Aided Design, FMCAD*, pages 195–202. IEEE, 2014.

46. John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
47. Mikoláš Janota, Jelle Piepenbrock, and Bartosz Piotrowski. Towards learning quantifier instantiation in SMT. In Kuldeep S. Meel and Ofer Strichman, editors, *25th International Conference on Theory and Applications of Satisfiability Testing, SAT 2022, August 2-5, 2022, Haifa, Israel*, volume 236 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
48. Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *LPAR*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.
49. Martin Suda. Improving ENIGMA-style clause selection while learning from history. In *CADE*, volume 12699 of *Lecture Notes in Computer Science*, pages 543–561. Springer, 2021.
50. Martin 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.
51. Geoff Sutcliffe, Christian B. Suttner, and Theodor Yemenis. The TPTP problem library. In *CADE*, volume 814 of *Lecture Notes in Computer Science*, pages 252–266. Springer, 1994.
52. Josef Urban. MPTP – Motivation, Implementation, First Experiments. *J. Autom. Reasoning*, 33(3-4):319–339, 2004.
53. Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.
54. Josef Urban. BliStr: The Blind Strategymaker. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19, 2015*, volume 36 of *EPiC Series in Computing*, pages 312–319. EasyChair, 2015.