What is the Point of an SMT-LIB Problem? *

Giles Reger and Martin Riener

University of Manchester, Manchester, UK

Abstract

Many areas of automated reasoning are goal-oriented i.e the aim is to prove a goal from a set of axioms. Some methods, including the method of saturation-based reasoning explored in this paper, do not rely on having an explicit goal but can employ specific heuristics if one is present. SMT-LIB problems do not record a specific goal, meaning that we cannot straightforwardly employ goal-oriented proof search heuristics. In this work we examine methods for identifying the potential goal in an SMT-LIB problem and evaluate (using the Vampire theorem prover) whether this can help theorem provers (using saturation-based proof search). We also discuss (very broadly) where SMT solvers could make use of goal information.

1 Introduction

The aim of this paper is to highlight the fact that it would be useful to introduce the notion of a *goal* into SMT-LIB. We mainly do this in the rather limited way of showing that detecting goals in problems can improve the performance of the saturation-based theorem prover Vampire[1] [10]. We leave a more general discussion of the possible wider utility of goals in SMT-LIB to the end of the paper.

Currently, SMT-LIB problems consist of a series of definitions (providing the signature of symbols) and assertions (providing some formulae that are meant to be true). We will generally ignore the command language of SMT-LIB here. This is in contrast to TPTP (another language for describing problems that the authors are familiar with) where formulas can be marked as *axioms* or as the (single) *conjecture* (other roles exist but are not as widely used). Many problems from various domains are phrased in this way. For example, a problem may consist of a translation of the next-state relation of a program (the axioms) followed by a postcondition that should hold (the conjecture). In the TPTP setting the meaning of the problem is that the axioms $\mathcal{A}$ imply the conjecture C i.e. provers should check the consistency of $\mathcal{A} \cup \{\neg C\}$. This problem can easily be framed in SMT-LIB but the structure of which part of the problem belongs to the axioms and which to the conjecture is lost.

In the traditional SMT setting losing this structure does not tend to matter as the focus is on model building and satisfiability in general. To check that a problem is satisfiable it is necessary to consider all assertions. However, if the focus is unsatisfiability checking then goals become more relevant as only a subset of the input may be required to show inconsistency and it is highly likely that this subset will contain the goal. In particular, in saturation-based solvers (such as Vampire) there exist a number of preprocessing and proof search heuristics that attempt to make proof search more goal-directed.

This paper shows that by detecting parts of the input that are likely to reflect goals we can improve the performance of the Vampire theorem prover. As it is unlikely that our method of goal detection is perfect, we suggest that this motivates the addition of the notion of goals to SMT-LIB.

The rest of the paper is organised as follows. We provide some necessary background (Section 2), describe some heuristics for detecting likely goals in the input (Section 3), present some experimental results (Section 4), discuss the wider applicability of goals to SMT-LIB (Section 5) and conclude (Section 6).

*This work was supported by EPSRC Grant EP/P03408X/1.

[1] See https://vprover.github.io to download.

2 Background

We introduce necessary background for the rest of the paper.

2.1 SMT-LIB

The focus of this paper is the SMT-LIB language [2]. The main fact we focus on is that it does not include a method for marking input formulas as goals or conjectures. Here we briefly discuss (some of) what it does support. Problems in this format consist of three kinds of statement:

- *Definitions.* These are used to define the sorts of function symbols or to introduce new sorts. In SMT-LIB the boolean sort is first-class, meaning that there is no separation between functions and predicates. A number of sorts and interpreted symbols are built-in e.g. `Int` and arithmetic operators such as $+$ and $*$.

- *Assertions.* These are used to assert first-order formulas using a Lisp-style syntax. Formulas may make use of the standard logical connectives (including quantification), defined function symbols, and predefined theory symbols.

- *Commands.* These instruct the solver to take certain actions. The main command of interest here is `check-sat` which is called after the problem has been described to instruct the solver to check (un)satisfiability.

The order of statements in SMT-LIB problems matters. Symbols must be defined before being used and commands instruct the solver to take an action using the statements occurring above that command in the file. This last part is particularly important when defining an *incremental* file. This can take two forms – either via a series of `check-sat` commands (where the assumption is that the answer is repeatedly `sat`) or by using the `push` and `pop` commands to create new solving contexts. This second approach is interesting to the discussion of this paper as one might assume that pushing some formulas on top of an existing set of formulas and checking the resulting problem might imply that the pushed formulas represent some kind of goal.

We discuss broader implications of goal-oriented reasoning and SMT-LIB later.

2.2 Goal-Oriented Reasoning in Vampire

Vampire is a saturation-based theorem prover for first-order logic with theories implementing the superposition and resolution calculus and a number of theory-specific reasoning extensions [14, 15, 18]. It entered SMT-COMP in 2016 and 2017, placing first in a number of divisions (AUFNIRA, UFDTLIA, UFNIA). Here we briefly describe how Vampire makes use of goal information.

Preprocessing. Vampire applies extensive preprocessing to input problems to place them into an optimised clausal form [7, 17]. There are two preprocessing steps that make proof search incomplete by focussing it on the goal.

The first is SInE selection [9] which is based on the observation that the proof of a goal tends to use only a small subset of the axioms, particularly in problems with large axiomatizations. The idea is to *heuristically* select a subset of axioms that are likely to be used in the proof of the goal. Axioms are selected using a notion of closeness to the goal that is based on whether they can be connected to the goal via their least common symbol. The approach can be parametrised by various measures and has been found to be widely successful in dealing with large problems.

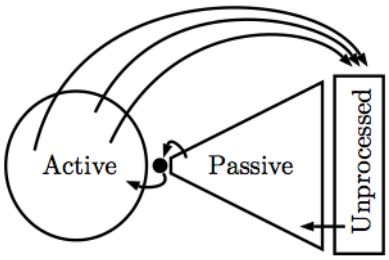

Figure 1: Illustrating the Given Clause Algorithm.

The second is the set-of-support strategy [24, 13] where clauses from the goal are put into a set of support and proof search is restricted so that it only makes inferences with clauses in, or derived from, this set. Therefore, this strategy requires all reasoning steps to be related to the goal directly.

Both approaches make proof search incomplete as they potentially throw away clauses that could be necessary to find a proof. They also prevent Vampire from establishing satisfiability, but in general this is impossible when theories are present.

Proof Search. Vampire is a refutational saturation-based theorem prover. The idea is that an input formula of the form $Premises \rightarrow Goal$ is negated, to give $Premises \wedge \neg Goal$, then clausified to produce a set of clauses S. This set is then *saturated* with respect to some inference system $\mathcal{I}$ meaning that for every inference from $\mathcal{I}$ with premises in S the conclusion of the inference is also in S. If the saturated set S contains a contradiction then the initial formula is necessarily valid. Otherwise, if $\mathcal{I}$ is a complete inference system, and importantly the requirements for this completeness have been preserved, then S is satisfiable and the input formula is not valid. If S contains theories then $\mathcal{I}$ cannot be complete.

The standard approach to saturation is the *given-clause* algorithm illustrated in Figure 1. The idea is to have an active set of clauses with the invariant that all inferences between active clauses have been performed and a passive set of clauses waiting to be activated. The algorithm then iteratively selects a *given clause* from passive and performs all necessary inferences to add it to active. Clause selection is the main place where information about the goal is used to steer proof search in Vampire. Passive clauses are stored in two queues ordered by weight and age respectively and clause selection picks clauses from the front of each queue based on a given ratio. One proof search heuristic (`nongoal_weight_coefficient`) specifies a multiplier for the weight of clauses not derived from the goal. Setting this to a value greater than one prioritises clauses derived from the goal.

Not Using the Goal. It should be pointed out that Vampire also has many heuristics and algorithms that make no reliance on the existence of a goal. Most proof search heuristics (e.g. literal selection [8]) are based on other metrics that suggest that clauses or inferences would lead to a proof quickly. Additionally, Vampire does not only employ the above saturation-based technique as it also uses finite model building [16] to establish finite satisfiability in the theory of uninterpreted functions. For this information about the goal is unnecessary.

3 Detecting Goals

In this section we consider two (rather primitive) heuristics for detecting input formulas that are likely to be goals. Ideally we would not need to do this as the originator of a problem is most likely best-

Table 1: Statistics on things that look like negated conjectures.

Kind of formula	Count of problems containing this kind
$\neg\forall\overline{x}.F[\overline{x}]$	3,846
$\exists\overline{x}.F[\overline{x}]$	3,686
Infrequent function symbols	51,628

placed to identify the goal. However, we implement the below techniques to (i) test whether identifying goals in SMT-LIB problems can help, and (ii) perform goal-oriented reasoning on existing problems in SMT-LIB that are unlikely to ever be annotated with goal information.

3.1 Things That Look Like Negated Conjectures

Our first heuristic attempts to detect things that might be a negated conjecture $\neg C$ such that overall unsatisfiability will show that C is valid (with respect to the rest of the problem). We focus on universally quantified conjectures. In such cases formulas of the form $\neg\forall\overline{x}.F[\overline{x}]$ or equivalently $\exists\overline{x}.F[\overline{x}]$ are likely candidates. However, some problems may already have been subject to some preprocessing, in which case existential variables may have been Skolemised. To detect such cases we look for formulas containing very infrequently occurring constant symbols or function symbols applied only to variables. Our default notion of very infrequent is for the symbol to appear in a single formula.

As a sanity check, Table 1 counts the number of problems with formulas of this kind in the area of SMT-LIB we are interested in for Vampire (problems containing quantifiers but not containing bit-vectors or floating point). This shows that there are a non-trivial number of problems to be targeted with this approach. However, it should be noted that the infrequent function symbols metric is likely to be a (significant) over-approximation of the number of problems containing useful goals as (i) all problems containing very few assertions will almost certainly have been included and (ii) problems containing constant symbol definitions unused elsewhere (which is common in translations) will have been included.

3.2 Things At the End

As an artefact of how problems are typically created, it is likely that goal assertions are placed at the end of problem files. To check this hypothesis we examined proofs produced by the Vampire theorem prover to see where the input formulas appear in the input problem. We chunk up the input problem into 11 buckets (with the last bucket being the very last formula i.e. it is an unevenly sized bucket) and find the distribution of the assertions from the proof across these buckets. Figure 2 gives the average distribution across 35,676 proofs. Within this data 76.5% of proofs contain the last line of the input problem. However, roughly 50% of proofs only use the last line of the problem in the proof which suggests that there are many problems consisting of a single line (this is not necessarily the case for all of these proofs, it may just be that the final line is inconsistent by itself). It is interesting to note that the second most common area for parts of the input problem to come from is the start – although this may again be an artifact of short problems.

The heuristic we implement is naive but supported (somewhat) by this evidence. We select the assertions at the end of the problem (with the default being the single final assertion) to mark as goal formulas. It should be noted that we don't necessarily expect this to be useful in a competition setting where syntactic features of a problem are altered. However, the focus here is on solving actual problems and it seems reasonable to attempt to make use of the artefact of goals appearing at the end of files.

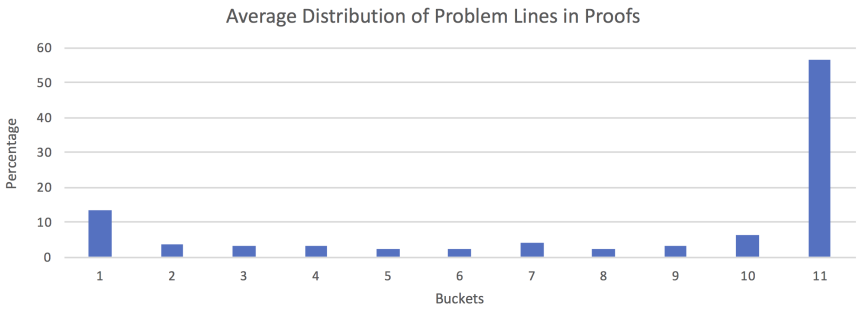

Figure 2: Checking the hypothesis that goals are at the end of problem files.

3.3 Implementation

The above heuristics lead to the implementation of a simple preprocessing step that marks formulas as goals. The option is called `guess_the_goal` and is (currently) available on an experimental branch of Vampire's GitHub repository[2]. The values this new option can take are as follows:

- `exists_top`. Mark as a goal if the top level is $\neg\forall$ or $\exists$.
- `exists_sym`. Mark as goal if contains a symbol appearing in at most k formulas (default $k = 1$)
- `exists_all`. The union of the previous two.
- `position`. Mark as goal in the final n (default $n = 1$)
- `all`. Perform all of the above.

Both the `exists_sym` and `position` options come with further parameters but we do not explore those here as our experiments are proof-of-concept in nature.

4 Experimental Evaluation

The experiments described in this section use all relevant problems from SMT-LIB[3] (i.e. those with quantifiers and no floating point or bitvectors)[4] and were run on the StarExec cluster [22] where each node contains an Intel Xeon 2.4GHz processor. In experiments we run Vampire 4.2.1 extended with the options previously described.

4.1 Exploring Goal-Oriented Heuristics

In our first experiment we consider the default mode of Vampire (with AVATAR modulo theories [14] switched on) and run `exists_all` goal detection and the two different goal-oriented preprocessing

[2] https://github.com/vprover/vampire/tree/smt_goals

[3] Copied from StarExec space https://www.starexec.org/starexec/secure/explore/spaces.jsp?id=234826.

[4] We report on a subset of the benchmarks in the experiments as for some logics the results are uninteresting.

Table 2: Number of solved problems when running goal detection and goal-oriented preprocessing with SInE selection and set of support. Numbers in parenthesis give number of unique solutions.

Logic	default	sine	sine10	sos
UFDTLIA	50	49	49	**59** (10)
UFDT	**1,944** (10)	1,935 (2)	1,932 (3)	1,933 (16)
UFIDL	55	55	55	55
UFLIA	**4,351** (96)	4,250 (10)	4,249 (8)	3,789 (40)
UF	2,950 (31)	2,966 (2)	**2,973** (4)	2,925 (26)

heuristics for 5 minutes[5]. For SInE we ran with two variants with sine10 being more restrictive than sine (it limits `sine_depth` to 10).

Table 2 presents the results for logics where one or more problems were solved. For many logics no problems were solved as the default strategy in Vampire is not very powerful by itself - we typically run tens to hundreds of different strategies in a portfolio mode (see next section). A solution is unique if it is solved by exactly one strategy.

All three strategies using goal-oriented preprocessing solved problems unsolved without those heuristics and in two cases such a strategy solved the most problems overall in a logic. This demonstrates that detecting goals can contribute positively to solving problems. The most restrictive of these options is sine10 and this performed the best.

4.2 Extending the Portfolio Mode

In our second experiment we applied goal detection on top of the portfolio used in the SMT competition[6]. Portfolio mode uses many strategies using a variety of different preprocessing and proof search options in a time-sliced manner across multiple cores. This time Vampire was run for 30 minutes (although it usually runs for much less time) with two of the goal detection strategies (due to issues with StarExec the results of other experiments are pending). The results are given in Table 3 (we exclude logics where all three strategies solve the same number of problems). Again we can see that the use of goal detection can improve the performance of Vampire. The results suggest that including the strategies where goal detection helped in the standard portfolio mode would observably improve the performance of Vampire.

5 Effect on SMT Solvers

In this section we consider whether adding this goal information could help traditional SMT solvers following the lazy CDCL(T) architecture [3] e.g. CVC4 [1].

Vampire uses goal information to select the part of the search space to prioritise when searching for a contradiction. SMT solvers (and SAT solvers) could do the same in variable selection i.e. selecting the SAT variable that represents the literal 'closest' to the goal (for some measure). We have begun to explore a similar idea in Vampire's AVATAR architecture [23] but have not obtained any experimental results yet.

[5]We run for 5 minutes with single strategies and 30 minutes with portfolio mode as the portfolio mode is made up of many single strategies and our results from the single strategy experiments are meant to act as guidance as to what should be added to the portfolio.

[6]See `http://smtcomp.sourceforge.net/2018/`

Table 3: Number of solved problems when extending portfolio mode with goal detection. Numbers in parenthesis give number of unique solutions.

Logic	`off`	`exists_all`	`position`
AUFDTLIA	634	634	**635** (1)
AUFLIRA	19,760 (1)	**19,761** (1)	19,760
AUFNIRA	1,051	1,049	**1,054** (4)
LIA	218	218	217
LRA	914 (2)	**1,006** (6)	1,003 (7)
UFDT	2,207 (4)	**2,218** (14)	2,202 (1)
UFLIA	7,537 (14)	**7,540** (23)	7526 (14)
UFNIA	**2,404** (19)	2,386 (20)	2,383 (18)
UF	**3,474** (14)	3,468 (15)	3,462 (8)

Another approach that we would expect to benefit from goal information would be for instantiation [19]. The most obvious use-case would be during instantiation via E-matching [12]. Preferring goal related terms during trigger selection could help find conflicting instances more quickly. Since a large number of instantiations increase the size of the congruence closure significantly the terms generated by E-matching could even be reduced to only those related to the goal. One could also imagine using goal information in model-based [6] and conflict-based [21] instantiation.

In the context of Isabelle, Sledgehammer already uses the conjecture to perform relevance filtering on the input problems[4]. However, this filtering is performed once and then the problem is given to the solver. If a solver can perform its own filtering in different ways (e.g. within a portfolio) using this goal information then it may be able to find a more optimal filtering.

Inductive theorem proving is another area where it can be helpful to know what the goal is as induction should be applied in a guided way. Previous approaches to integrating induction into SMT solvers [20, 11] used similar heuristics to those mentioned in Section 3 to identify formulas on which to apply induction. The TIP format already provides the `assert-not` construct [5] to indicate a goal.

Does Adding Goals Help CVC4? We ran a further experiment to investigate whether goal detection could help in SMT solvers. We used Vampire as a pre-processor (`--mode tpreprocess`), once with SInE selection of unlimited depth and goal guessing turned on (`-ss axioms -gtg exists_all`) and once without (`-ss off`)). To reduce the impact of unrelated optimizations, the options for function definition elimination (`-fde off`, unused predicate definition removal (`-updr off`) and for subterm naming (`-nm 32767`) were turned off. Since Vampire only outputs the TPTP format we were restricted solvers that can read it. This reduced the choice of SMT solvers to CVC4 which we used the development version[7] with a timeout of 300 seconds on the two versions of the benchmark.

To identify problems that could benefit from relevance filtering, we took the non-incremental SMTLIB benchmarks[8] and focused on the 2534 benchmarks that are expected to be unsatisfiable and larger than $100KB$. Due Starexec's space quota of 5GB, we removed the lahiri-cav09-storm-queries from the UFNIA suite and ended up with 2225 files. Out of these 1930 could be parsed by CVC4[9].

Applying SInE selection took between 0.005 and 6.892 with an average of 0.328 seconds. There

[7] Version 1.6pre at git commit 28e9077fad9d5c61c61a1762e7cf021c226cb9c2

[8] 54825 in the logics ALIA, AUFDTLIA, AUFLIA, AUFLIRA, AUFNIRA, LIA, LRA, NIA, NRA, UF, UFDT, UFDTLIA, UFIDL, UFLIA, UFLRA, UFNIA

[9] Some literals like $\$false = X$ are unsupported.

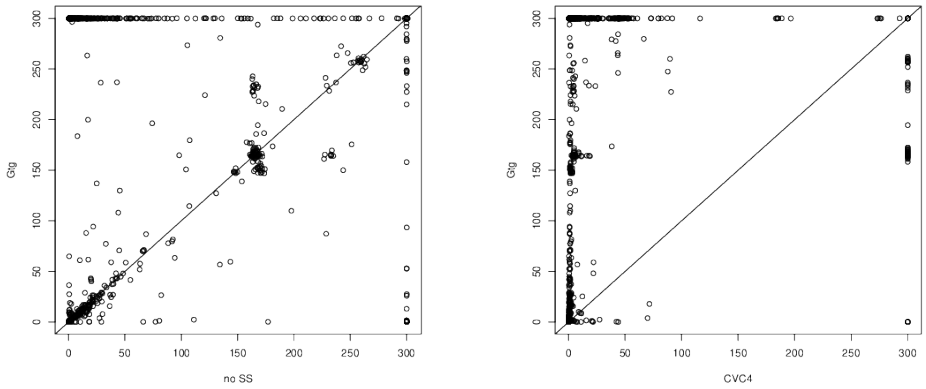

Figure 3: Scatter plots of SInE selection with goal guessing against no sine selection and a direct call of CVC4

were 281 problems that could not be solved without relevance filtering, with 23 being solved between 50 and 250 seconds. There were also 7 cases taking longer than 10 seconds where the speedup factor was greater than 2. Compared to CVC4 executed directly on the original SMTLIB file there are none that can only be solved with our heuristics. There were 3 cases taking longer than 10 seconds where the speedup factor was greater than 2. Scatter plots of the timings can be seen in Figure 3. This indicates possible speedups for direct inclusion of relevance filtering into SMT solvers.

6 Conclusion

The aim of this paper was to demonstrate that adding the notion of a goal or conjecture to SMT-LIB problems would be generally helpful. We have done this by automatically detecting likely goal assertions and then showing that treating such assertions as goals in the Vampire theorem prover improved performance. We are not suggesting that the options described in this paper are optimised or will be used as they are within Vampire in the future (although we will explore this).

Whilst we are suggesting that it would be helpful for SMT-LIB problems to include goal information, we have made no proposal as to how this should be done.

References

[1] C. Barrett, C. Conway, M. Deters, L. Hadarean, D. Jovanovic, T. King, A. Reynolds, and C. Tinelli. CVC4. In *Proceedings of the 23rd International Conference on Computer Aided Verification*, number 6806 in Lecture Notes in Computer Science, pages 171–177. Springer-Verlag, 2011.

[2] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The SMT-LIB Standard: Version 2.6. Technical report, Department of Computer Science, The University of Iowa, 2017. Available at `www.SMT-LIB.org`.

[3] Clark Barrett, Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Splitting on demand in SAT modulo theories. In *Logic for Programming, Artificial Intelligence, and Reasoning, 13th International Conference, LPAR 2006, Phnom Penh, Cambodia, November 13-17, 2006, Proceedings*, pages 512–526, 2006.

[4] Jasmin Christian Blanchette, Sascha Böhme, and Lawrence C. Paulson. Extending Sledgehammer with SMT solvers. *J. Autom. Reasoning*, 51(1):109–128, 2013.

[5] Koen Claessen, Moa Johansson, Dan Rosén, and Nicholas Smallbone. TIP: tons of inductive problems. In *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, pages 333–337, 2015.

[6] Yeting Ge and Leonardo Mendonça de Moura. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, pages 306–320, 2009.

[7] Krystof Hoder, Zurab Khasidashvili, Konstantin Korovin, and Andrei Voronkov. Preprocessing Techniques for First-Order Clausification. In *Formal Methods in Computer-Aided Design, FMCAD 2012, Cambridge, UK, October 22-25, 2012*, pages 44–51, 2012.

[8] Kryštof Hoder, Giles Reger, Martin Suda, and Andrei Voronkov. Selecting the selection. In *Automated Reasoning: 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 – July 2, 2016, Proceedings*, pages 313–329, Cham, 2016. Springer International Publishing.

[9] Kryštof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction – CADE-23*, pages 299–314, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[10] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV 2013*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35, 2013.

[11] K. Rustan M. Leino. Automating induction with an SMT solver. In *Proceedings of the 13th International Conference on Verification, Model Checking, and Abstract Interpretation*, VMCAI'12, pages 315–331, Berlin, Heidelberg, 2012. Springer-Verlag.

[12] K. Rustan M. Leino and Clément Pit-Claudel. Trigger selection strategies to stabilize program verifiers. In *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, pages 361–381, 2016.

[13] William McCune. OTTER 2.0. In *10th International Conference on Automated Deduction, Kaiserslautern, FRG, July 24-27, 1990, Proceedings*, pages 663–664, 1990.

[14] Giles Reger, Nikolaj Bjørner, Martin Suda, and Andrei Voronkov. AVATAR modulo theories. In Christoph Benzmüller, Geoff Sutcliffe, and Raul Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41 of *EPiC Series in Computing*, pages 39–52. EasyChair, 2016.

[15] Giles Reger and Martin Suda. Set of support for theory reasoning. In Thomas Eiter, David Sands, Geoff Sutcliffe, and Andrei Voronkov, editors, *IWIL Workshop and LPAR Short Presentations*, volume 1 of *Kalpa Publications in Computing*, pages 124–134. EasyChair, 2017.

[16] Giles Reger, Martin Suda, and Andrei Voronkov. Finding finite models in multi-sorted first-order logic. In Nadia Creignou and Daniel Le Berre, editors, *SAT 2016*, pages 323–341. Springer International Publishing, 2016.

[17] Giles Reger, Martin Suda, and Andrei Voronkov. New techniques in clausal form generation. In Christoph Benzmüller, Geoff Sutcliffe, and Raul Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41 of *EPiC Series in Computing*, pages 11–23. EasyChair, 2016.

[18] Giles Reger, Martin Suda, and Andrei Voronkov. Unification with abstraction and theory instantiation in saturation-based reasoning. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 3–22, Cham, 2018. Springer International Publishing.

[19] Andrew Reynolds. Conflicts, models and heuristics for quantifier instantiation in SMT. In Laura Kovacs and Andrei Voronkov, editors, *Vampire 2016. Proceedings of the 3rd Vampire Workshop*, volume 44 of *EPiC Series in Computing*, pages 1–15. EasyChair, 2017.

[20] Andrew Reynolds and Viktor Kuncak. Induction for SMT solvers. In *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, pages 80–98, 2015.

[21] Andrew Reynolds, Cesare Tinelli, and Leonardo de Moura. Finding conflicting instances of quantified formulas in SMT. In *Proceedings of the 14th Conference on Formal Methods in Computer-Aided Design*, FMCAD '14, pages 31:195–31:202, Austin, TX, 2014. FMCAD Inc.

[22] Aaron Stump, Geoff Sutcliffe, and Cesare Tinelli. StarExec, a cross community logic solving service.

https://www.starexec.org, 2012.

[23] Andrei Voronkov. AVATAR: The architecture for first-order theorem provers. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 696–710. Springer International Publishing, 2014.

[24] Lawrence Wos, George A. Robinson, and Daniel F. Carson. Efficiency and completeness of the set of support strategy in theorem proving. *J. ACM*, 12(4):536–541, October 1965.