Efficient Instantiation Techniques in SMT (Work In Progress)

Haniel Barbosa

Efficient Instantiation Techniques in SMT (Work In Progress)

Haniel Barbosa*

LORIA, INRIA, Université de Lorraine, Nancy, France
haniel.barbosa@inria.fr

Abstract

In SMT solving one generally applies heuristic instantiation to handle quantified formulas. This has the side effect of producing many spurious instances and may lead to loss of performance. Therefore deriving both fewer and more meaningful instances as well as eliminating or dismissing, i.e., keeping but ignoring, those not significant for the solving are desirable features for dealing with first-order problems.

This paper presents preliminary work on two approaches: the implementation of an efficient instantiation framework with an incomplete goal-oriented search; and the introduction of dismissing criteria for heuristic instances. Our experiments show that while the former improves performance in general the latter is highly dependent on the problem structure, but its combination with the classic strategy leads to competitive results w.r.t. state-of-the-art SMT solvers in several benchmark libraries.

1 Introduction

SMT solvers (see [4] for a general presentation of SMT) are extremely efficient at handling large ground formulas with interpreted symbols, but they still struggle to deal with quantified formulas. Quantified first-order logic is best handled with *Resolution* and *Superposition*-based theorem proving [2, 16]. Although there are first attempts to unify such techniques with SMT [13], the main approach used in SMT is still *instantiation*: formulas are freed from quantifiers and refuted with the help of decision procedures for ground formulas.

The most common strategy for finding instances in SMT is the use of *triggers* [10]: some terms in a quantified formula are selected to be instantiated and successfully doing so provides a ground instantiation for the formula. These triggers are selected according to various heuristics and instantiated by performing E-matching over candidate terms retrieved from a ground model. The lack of a *goal* in this technique (such as, e.g., refuting the model) leads to the production of many instances not relevant for the solving. Furthermore, unlike other non-goal-oriented techniques, such as superposition, there are no straightforward redundancy criteria for the elimination of derived instances in SMT solving. Therefore useless instances are kept, potentially hindering the solver's performance.

Our attempt to tackle this issue is two-fold:

- A method for deriving fewer instances by setting the refutation of the current model as a goal, as in [19]. Thus all instances produced by this strategy are relevant.

*This work has been partially supported by the ANR/DFG project STU 483/2-1 SMArT, project ANR-13-IS02-0001 of the Agence Nationale de la Recherche

In: P. Fontaine, S. Schulz, J. Urban (eds.): Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning (PAAR 2016), Coimbra, Portugal, 02-07-2016, published at http://ceur-ws.org

- Heuristic instantiation is kept under control. Given their speed and the difficulty of the first-order reasoning with interpreted symbols, heuristics are a necessary evil. To reduce side effects, spurious instances are dismissed. The criterion is their activity as reported by the ground solver, in a somehow hybrid approach avoiding both the two-tiered combination of SAT solvers [12] and deletion [7].

We also introduce a lifting of the classic congruence closure procedure to first-order logic and show its suitability as the basis of our instantiation techniques. Moreover, it is shown how techniques common in first-order theorem proving are being implemented in an SMT setting, such as using efficient *term indexing* and performing E-unification.

Formal preliminaries

Due to space constraints, we refer to the classic notions of many-sorted first-order logic with equality as the basis for the notation in this paper. Only the most relevant are mentioned.

Given a set of ground terms $\mathbf{T}$ and a congruence relation $\simeq$ on $\mathbf{T}$, a *congruence* C over $\mathbf{T}$ is a set $C \subseteq \{s \simeq t \mid s,t \in \mathbf{T}\}$ closed under entailment: for all $s,t \in \mathbf{T}$, $C \models s \simeq t$ iff $s \simeq t \in C$. The *congruence closure* of C is the least congruence on $\mathbf{T}$ containing C. Given a consistent set of ground equality literals E, two terms t_1, t_2 are said *congruent* iff $E \models t_1 \simeq t_2$, which amounts to $t_1 \simeq t_2$ being in the congruence closure of the equalities in E, and *disequal* iff $E \models t_1 \not\simeq t_2$. The *congruence class* of a given term t, represented by $[t]$, is the partition of $\mathbf{T}$ induced by E in which all terms are congruent to t.

2 Congruence Closure with Free Variables

To better handle the quantified formulas during instantiation algorithms we have developed a *Congruence Closure with Free Variables* (CCFV, for short), which extends the classic congruence closure procedure [14, 15] into handling conjunctions of equality literals with free variables, performing *rigid* E-*unification*: finding solutions to a given set of equations consistent with a set of equations E, assuming that every variable denotes a single term.

$$\frac{L,\ x \mathrel{\dot{\simeq}} y \parallel U}{L \parallel U \cup \{x \mathrel{\dot{\simeq}} y\}}\ (\text{RV}) \qquad \begin{array}{ll}(\text{i}) & \dot{\simeq} \in \{\simeq, \not\simeq\} \\ (\text{ii}) & x \text{ or } y \text{ is free in } U, \text{ or } E \cup U \models x \mathrel{\dot{\simeq}} y\end{array}$$

$$\frac{L,\ x \mathrel{\dot{\simeq}} t \parallel U}{L \parallel U \cup \{x \mathrel{\dot{\simeq}} t\}}\ (\text{RT}) \qquad \begin{array}{ll}(\text{i}) & \dot{\simeq} \in \{\simeq, \not\simeq\} \\ (\text{ii}) & \text{either } x \text{ is free in } U \text{ or } E \cup U \models x \mathrel{\dot{\simeq}} t\end{array}$$

$$\frac{L, f(u) \simeq f(v) \parallel U}{L, u \simeq v \parallel U}\ (\textsc{Decompose}) \qquad \frac{L \parallel U}{\top}\ (\textsc{Yield}) \quad (\text{i}) \quad L = \varnothing \text{ or } E \models L$$

$$\frac{L,\ f(u) \mathrel{\dot{\simeq}} t \parallel U}{\begin{array}{l}L,\ f(u) \simeq f(t_1) \parallel U \\ \cdots \\ L,\ f(u) \simeq f(t_n) \parallel U\end{array}}\ (\textsc{Ematch}) \qquad \begin{array}{ll}(\text{i}) & \dot{\simeq} \in \{\simeq, \not\simeq\} \\ (\text{ii}) & f(t_i) \text{ are ground terms from } E \\ (\text{iii}) & E \models t \mathrel{\dot{\simeq}} f(t_i), \text{ for } 1 \le i \le n\end{array}$$

$$\frac{L,\ u \mathrel{\dot{\simeq}} f(u') \parallel U}{\begin{array}{l}L,\ u \simeq t_{1,1},\ f(u') \simeq f(t'_1) \parallel U \\ \cdots \\ L,\ u \simeq t_{1,m_1},\ f(u') \simeq f(t'_1) \parallel U \\ \cdots \\ L,\ u \simeq t_{n,m_n},\ f(u') \simeq f(t'_n) \parallel U\end{array}}\ (\textsc{Euni}) \qquad \begin{array}{ll}(\text{i}) & \dot{\simeq} \in \{\simeq, \not\simeq\} \\ (\text{ii}) & t_{i,j},\ f(t'_i) \text{ are ground terms from } E \\ (\text{iii}) & E \models t_{i,j} \mathrel{\dot{\simeq}} f(t'_i), \text{ for } 1 \le i \le n,\ 1 \le j \le m_i\end{array}$$

$$\frac{L \parallel U}{\bot}\ (\textsc{Close}) \qquad (\text{i}) \quad L \text{ is inconsistent modulo } E \text{ or no other rule can be applied}$$

Table 1: CCFV calculus for solving rigid E-unification in equational FOL. Multiple conclusion rules represent branching in the search. x, y refer to variables, t to ground terms, u to non-ground terms and v to terms in general.

Our procedure implements the rules[1] shown in Table 1. To simplify presentation, it is assumed, without loss of generality, that function symbols are unary. Rules are shown only for equality literals, as their extension into uninterpreted predicates is straightforward. The calculus operates on conjunctive sets of equality literals containing free variables, annotated with equality literals between these variables and ground terms or themselves. Initially the annotations are empty, being augmented as the rules are applied and the input problem is simplified, embodying its solution.

CCFV algorithm

Given a set of ground equality literals E and a set of non ground equality literals L whose free variables are X, CCFV computes sets of equality literals $U_1, \ldots, U_n$, denoted *unifiers*. Each unifier associates variables from X to ground terms and allows the derivation of ground substitutions $\sigma_1, \ldots, \sigma_k$ such that $E \models L\sigma_i$, if any:

$$\sigma_i = \left\{ x \mapsto t \;\middle|\; \begin{array}{l} x \in X;\ U \models x \simeq t \text{ for some ground term } t. \text{ If } x \text{ is free} \\ \text{in } U,\ t \text{ is a ground term selected from its sort class.} \end{array} \right\}$$

Since not necessarily all variables in X are congruent to ground terms in a given unifier U (denoted "free in U"), more than one ground substitution may be obtained by assigning those variables to different ground terms in their sort classes.

A terminating strategy for CCFV is to apply the rules of Table 1 exhaustively over L, except that `Ematch` may not be applied over the literals it introduces. There must be a backtracking when a given branch results in `Close`, until being able to apply `Yield`. In those cases a unifier is provided from which substitutions solving the given E-unification problem can be extracted.

Term Indexing

Performing E-unification requires dealing with many terms, which makes the use of an efficient indexing technique for fast retrieval of candidates paramount.

The Congruence Closure procedure in veriT keeps a *signature table*, in which terms and predicate atoms are kept modulo their congruence classes. For instance, if $a \simeq b$ and both $f(a)$ and $f(b)$ appear in the formula, only $f(a)$ is kept in the signature table. Those are referred to as *signatures* and are the only relevant terms for indexing, since instantiations into terms with the same signature are logically equivalent modulo the equalities in the current context. The signature table is indexed by *top symbol*[2], such that each function and predicate symbol points to all their related signatures. Those are kept sorted by congruence classes, to be amenable for binary search. Bitmasks are kept to fast check whether a class contains signatures with a given top symbol, a necessary condition for retrieving candidates from that class.

A side effect of building the term index from the signature table is that *all terms* are considered, regardless of whether they appear or not in the current SAT solver model. To tackle this issue, an alternative index is built directly from the currently asserted literals while computing on the fly the respective signatures. Dealing directly with the model also has the advantage of allowing its minimization, since the SAT solver generally asserts more literals than necessary. Computing a *prime implicant*, a minimal partial model, can be done in linear time [9]. Moreover, the CNF overhead is also cleaned: removing literals introduced by the non-equivalency preserving CNF transformation the formula undergoes, applying the same process described in [7] for *Relevancy*.

Implementing *E*-unification

The main data structure for CCFV is the "unifiers": for a set of variables X, an array with each position representing a valuation for a variable $x \in X$, which consists of:

- a field for the respective variable;
- a flag to whether that variable is the representative of its congruence class;
- a field for, if the variable is a representative, the ground term it is equal to and a set of terms it is disequal to; otherwise a pointer to the variable it is equal to, the default being itself.

[1] The calculus still needs to be improved, with a better presentation and the proofs of its properties, which are work in progress.

[2] Since top symbol indexing is not optimal, the next step is to implement *fingerprint indexing*. The current implementation keeps the indexing as modular as possible to facilitate eventually changing its structure.

Each unifier represents one of the sets U mentioned above. They are handled with a UNION-FIND algorithm with path-compression. The *union* operation is made modulo the congruence closure on the ground terms and the current assignments to the variables in that unifier, which maintains the invariant of it being a consistent set of equality literals.

The rules in Table 1 are implemented as an adaptation of the *recursive descent* E-*unification* algorithm in [1], heavily depending on the term index described shown above for optimizing the search. Currently it does not have a dedicated index for performing unification, rather relying in the DAG structure of the terms. To avoid (usually expensive) re-computations, *memoization* is used to store the results of E-unifications attempts, which is particularly useful when looking for unifiers for, e.g., $f(\mathbf{x}) \simeq g(\mathbf{y})$ in which both "f" and "g" have large term indexes. For now these "unification jobs" are indexed by the literal's polarity and participating terms, not taking into account their structure.

3 Instantiation Framework

3.1 Goal-oriented instantiation

In the classic architecture of SMT solving, a SAT solver enumerates boolean satisfiable conjunctions of literals to be checked for ground satisfiability by decision procedures for a given set of theories. If these models are not refuted at the ground level they must be evaluated at the first-order level, which is not a decidable problem in general. Therefore one cannot assume to have an efficient algorithm to analyze the whole model and determine if it can be refuted. This led to the regular heuristic instantiation in SMT solving being not goal-oriented: its search is based solely on pattern matching of selected triggers [10], without further semantic criteria, which can be performed quickly and then revert the reasoning back to the efficient ground solver.

In [19], Reynolds *et al.* presented an efficient incomplete goal-oriented instantiation technique that evaluates a quantified formula, independently, in search for *conflicting instances*: given a satisfiable conjunctive set of ground literals E, a set of quantified formulas $\mathcal{Q}$ and some $\forall \mathbf{x}.\psi \in \mathcal{Q}$ it searches for a ground substitution σ such that $E \models \neg\psi\sigma$. Such substitutions are denoted *ground conflicting*, with *conflicting instances* being such that $\forall \mathbf{x}.\psi \rightarrow \psi\sigma$ refutes $E \cup Q$.

Since the existence of such substitutions is an NP-complete problem equivalent to Non-simultaneous rigid E-unification [20], the CCFV procedure is perfectly suited to solve it. Each quantified formula $\forall \mathbf{x}.\psi \in \mathcal{Q}$ is converted into CNF and CCFV is applied for computing sequences of substitutions[3] $\sigma_0, \ldots, \sigma_k$ such that, for $\neg\psi = l_1 \wedge \cdots \wedge l_k$,

$$\sigma_0 = \varnothing;\ \sigma_{i-1} \subseteq \sigma_i \text{ and } E \models l_i\sigma_i$$

which guarantees that $E \models \neg\psi\sigma_k$ and that the instantiation lemma $\forall \mathbf{x}.\psi \rightarrow \psi\sigma_k$ refutes $E \cup Q$. If any literal l_{i+i} is not unifiable according to the unifications up to l_i, there are no conflicting instances for $\forall \mathbf{x}.\psi$.

Currently our implementation applies a breadth-first search on the conjunction of non-ground literals, computing all unifiers for a given literal $l \in \neg\psi$ before considering the next one. Memory consumption is an issue due to the combinatorial explosion that might occur when merging sets of unifiers from different literals. A more general issue is simply the time required for finding the unifiers of a given literal, which can have a huge search space depending on the number of indexed terms. To minimize these problems customizable parameters set thresholds both on the number of potential combinations and of terms to be considered.

3.2 Heuristic instantiation with instances dismissal

Although goal-oriented search avoids heuristic instantiation in many cases, triggers and pattern-matching are still the backbone of our framework. A well known side effect of them is the production of many spurious instances which not only interfere with the performances of both the ground and instantiation modules but also may lead to *matching loops*: triggers generating instances which are used solely to produce new instances in an infinite chain. To avoid this issues, de Moura *et al.* [7] mention how they perform clause deletion, during backtracking, of instances which were not part of a conflict. However, this proved to be an engineering challenge in veriT, since its original architecture does not easily allow deletion of terms from the ground solver.

[3]Since CCFV is non-proof confluent calculus, as choices may need to be made whenever a matching or unification must be performed, backtracks are usually necessary for exploring different options.

To circumvent this problem, instead of being truly deleted instances are simply *dismissed*: by labeling them with *instantiation levels*[4], at a given level n only instances whose level is at most $n-1$ are considered. This is done by using the term indexing from the SAT model and eliminating literals whose instantiation level is above the threshold. At each instantiation round, the *level* is defined as the current level plus one. At the beginning of the solving all clauses are assigned level 0, the starting value for the instantiation level. At the end of an instantiation round, the SAT solver is notified that at that point in the decision tree there was an instantiation, so that whenever there is a backtracking to a point before such a mark the instantiation level is decremented, at the same time that all instances which have participated in a conflict are promoted to "level 0". This ensures that those instances will not be dismissed for instantiation, which somehow emulates clause deletion. With this technique, however, the ground solver will still be burdened by the spurious instances, but they also will not need to be regenerated in future instantiation rounds.

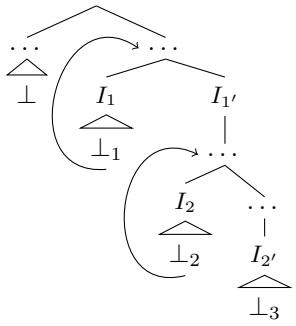


Figure 1: Example of instance dismissal

Consider in Figure 1 an example of instance dismissal. I_1 marks an instantiation happening at level 1, in which all clauses from the original formula are considered for instantiation. Those lead to a conflict and a backtrack to a point before I_1, which decrements the instantiation level to 0. All instances from I_1 which were part of the conflict in $\bot_1$ are promoted to level 0, the rest kept with level 1. At $I_{1'}$ only terms in clauses with level 0 are indexed. Since subsequently there is no backtracking to a point before $I_{1'}$, the instantiation level is increased to 1. At I_2 all clauses of level 1 are considered, thus including those produced both in I_1 and $I_{1'}$. After a backtrack, the level is decremented to 1 and the instances participating in $\bot_2$ are promoted. This way at $I_{2'}$ only the promoted instances from the previous round are considered. Then the ground solver reaches a conflict in $\bot_3$ and cannot produce any more models, concluding unsatisfiability.

4 Experiments

The above techniques have been implemented in the SMT solver veriT [6], which previously offered support for quantified formulas solely through naïve trigger instantiation, without further optimizations[5]. The evaluation was made on the "UF", "UFLIA", "UFLRA" and "UFIDL" categories of SMT-LIB [5], which have 10,495 benchmarks annotated as *unsatisfiable*. They consist mostly of quantified formulas over uninterpreted functions as well as equality and linear arithmetic. The categories with bit vectors and non-linear arithmetic are currently not supported by veriT and in those in which uninterpreted functions are not predominant the techniques shown here are not quite as effective yet. Our experiments were conducted using machines with 2 CPUs Intel Xeon E5-2630 v3, 8 cores/CPU, 126GB RAM, 2x558GB HDD. The timeout is set for 30 seconds, since our goal is evaluating SMT solvers as backends of verification and ITP platforms, which require fast answers.

The different configurations of veriT are identified in this section according to which techniques they have activated:

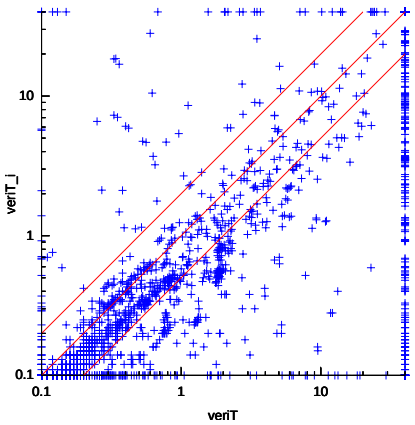
- veriT: the solver relying solely on naïve trigger instantiation;
- veriT_i: the solver with CCFV and the signature table indexed;
- veriT_ig: besides the above, uses the goal-oriented search for conflicting instances;

[4]This is done much in the spirit of [11], although their labeling does not take into account the interactions with the SAT solver and is aimed at preventing matching loops, not towards "deletion".

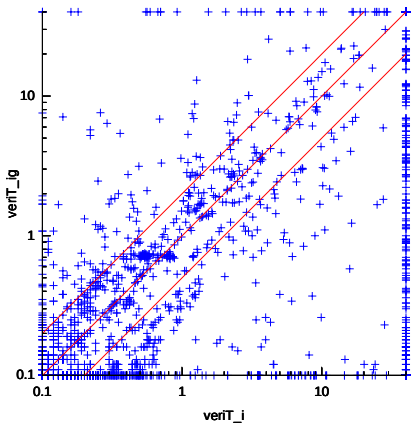
[5]A development version is available at `http://www.loria.fr/~hbarbosa/veriT-ccfv.tar.gz`

- veriT_igd: does the term indexing from the SAT solver and uses the goal-oriented search and instance dismissal.

Figure 2a shows the big impact of the handling of instantiation by CCFV: veriT_i is significantly faster and solves 326 problems exclusively, while the old configuration solves only 32 exclusively. Figure 2b presents a significant improvement in terms of problems solved (474 more against 36 less) by the use of the goal-oriented instantiation, but it also shows a less clear gain of time. Besides the natural chaotic behavior of trigger instantiation, we believe this is due to the more expensive search performed: trying to falsify quantified formulas and handling E-unification, which, in the context of SMT, has a much bigger search space than simply performing E-matching for pattern-matching instantiation. Not always the "better quality" of the conflicting instances offsets the time it took to compute them, which indicates the necessity of trying to identify beforehand such cases and avoid the more expensive search when counter-producent.



(a) Impact of CCFV, without goal-oriented instantiation



(b) Impact of goal-oriented instantiation

Figure 2: Comparisons of new term indexing, CCFV and goal-oriented search

Logic	Class	CVC4	Z3	veriT_igd	veriT_ig	veriT_i	veriT
UF	grasshopper	410	418	431	**437**	418	413
	sledgehammer	**1412**	1249	1293	1272	1134	1066
UFIDL	all	61	**62**	56	58	58	58
UFLIA	boogie	841	**852**	722	681	660	661
	sexpr	15	**26**	15	7	5	5
	grasshopper	320	341	356	**367**	340	335
	sledgehammer	**1892**	1581	1781	1778	1620	1569
	simplify	770	**831**	797	803	735	690
	simplify2	2226	**2337**	2277	2298	2291	2177
Total		**7947**	7697	7727	7701	7203	6916

Table 2: Comparison between instantiation based SMT solvers on SMT-LIB benchmarks

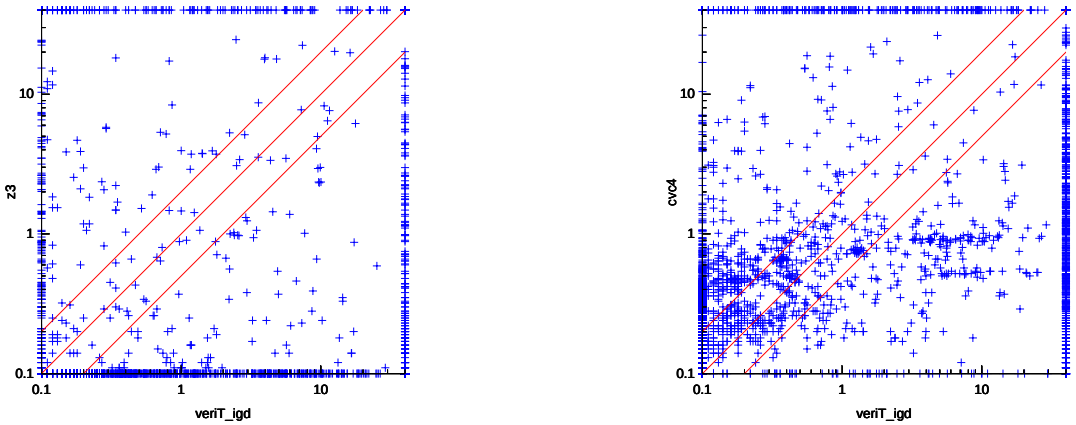
Our new implementations were also evaluated against the SMT solvers Z3 [8] (version 4.4.2) and CVC4 [3] (version 1.5), both based on instantiation for handling quantified formulas. The results are summarized in Table 2, excluding categories whose problems are trivially solved by all systems, which leaves $8,701$ for consideration.

While veriT_ig and veriT_igd solve a similar number of problems in the same categories (with a small advantage to the latter), it should be noted that they have quite diverse results depending on the benchmark (a comparison is shown in Figure 4 at Appendix A). Each configuration solves ≈ 150 problems exclusively. This indicates the potential to use both the term indexes, from the signature table and from the SAT solver model with instance dismissal, during the solving.

Regarding overall performance, CVC4 solves the most problems, being the more robust SMT solver for instantiation and also applying a goal-oriented search for conflicting instances. Both configurations of veriT solve approximately the same number of problems as Z3, although mostly because of the better performance on the

sledgehammer benchmarks, which have less theory symbols. There are 124 problems solved by veriT_igd that neither CVC4 nor Z3 solve, while veriT_ig solves 115 that neither of these two do.

Figure 3 shows how the better veriT configuration, with the goal-oriented search and instance dismissal, performs against the other solvers. There are many problems solved exclusively by each system, which indicates the benefit of combining veriT with those systems them in a portfolio when trying to quickly solve a particular problem: while CVC4 alone solves $\approx 92\%$ of the considered benchmarks in 30s, by giving each of the four compared systems 7s is enough to solve $\approx 97\%$ of them.



(a) Z3 vs veriT_igd

(b) CVC4 vs veriT_igd

Figure 3: Comparisons with SMT solvers

5 Conclusion and future work

There is still room for improvement in our instantiation framework. Particularly, a better understanding of the instance dismissal effects is still required. Further analyzing the clauses activity would lead to a more refined promotion strategy and possibly better outcomes. Nevertheless, we believe that our preliminary results are promising.

Regarding the term indexing, besides improving the data structures our main goal is performing it *incrementally*: by indexing the literals from the SAT model it is not necessary to thoroughly recompute the index at each instantiation round. It is sufficient to simply remove or add terms, as well as update signatures, according to how the model has changed. The same principle may be applied to the memoization of "unification jobs": an incremental term index would allow updating the resulting unifiers accordingly, significantly reducing the instantiation effort over rounds with similar indexes.

Our goal-oriented instantiation has a very limited scope: currently conflicting instances can only be found when a single quantified formula is capable of refuting the model. As it has been shown in [19] and also in our own experiments this is enough to provide large improvements over trigger instantiation, but for many problems it is still insufficient. We intend to combine CCFV with the *Connection Calculus* [17], a complete goal-oriented proof procedure for first-order logic, in an effort for having a broader approach for deriving conflicting instances. This would present a much more complex search space than the one our strategy currently handles. Therefore the trade-off between expressivity and cost has to be carefully evaluated.

Applying different strategies in a portfolio approach is highly beneficial for solving more problems, but it could be even more so if different configurations were to communicate. Attempting pseudo-concurrent architectures such as described in [18] in veriT is certainly worth considering.

Acknowledgments

I would like to thank my supervisors Pascal Fontaine and David Déharbe for all their help throughout the development of this work and comments on the article.

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

References

[1] F. Baader and W. Snyder. Unification theory. In *Handbook of Automated Reasoning (in 2 volumes)*, pages 445–532. 2001.

[2] L. Bachmair and H. Ganzinger. Rewrite-Based Equational Theorem Proving with Selection and Simplification. *J. Log. Comput.*, 4(3):217–247, 1994.

[3] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli. CVC4. In G. Gopalakrishnan and S. Qadeer, editors, *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 171–177, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[4] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli. Satisfiability Modulo Theories. In A. Biere, M. J. H. Heule, H. van Maaren, and T. Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, Feb. 2009.

[5] 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.

[6] T. Bouton, D. C. B. de Oliveira, D. Déharbe, and P. Fontaine. veriT: An Open, Trustable and Efficient SMT-Solver. In *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, pages 151–156, 2009.

[7] L. de Moura and N. Bjørner. Efficient E-Matching for SMT Solvers. In F. Pfenning, editor, *Automated Deduction CADE-21*, volume 4603 of *Lecture Notes in Computer Science*, pages 183–198. Springer Berlin Heidelberg, 2007.

[8] L. M. de Moura and N. Bjørner. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 337–340, 2008.

[9] D. Déharbe, P. Fontaine, D. Le Berre, and B. Mazure. Computing Prime Implicants. In *FMCAD - Formal Methods in Computer-Aided Design 2013*, pages 46–52, Portland, United States, Oct. 2013. IEEE.

[10] D. Detlefs, G. Nelson, and J. B. Saxe. Simplify: A Theorem Prover for Program Checking. *J. ACM*, 52(3):365–473, May 2005.

[11] Y. Ge, C. Barrett, and C. Tinelli. Solving Quantified Verification Conditions Using Satisfiability Modulo Theories. In F. Pfenning, editor, *Automated Deduction CADE-21*, volume 4603 of *Lecture Notes in Computer Science*, pages 167–182. Springer Berlin Heidelberg, 2007.

[12] K. R. M. Leino, M. Musuvathi, and X. Ou. A Two-tier Technique for Supporting Quantifiers in a Lazily Proof-explicating Theorem Prover. In *Proceedings of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'05, pages 334–348, Berlin, Heidelberg, 2005. Springer-Verlag.

[13] L. Moura and N. Bjørner. Engineering DPLL(T) + Saturation. In *Proceedings of the 4th International Joint Conference on Automated Reasoning*, IJCAR '08, pages 475–490, Berlin, Heidelberg, 2008. Springer-Verlag.

[14] G. Nelson and D. C. Oppen. Fast Decision Procedures Based on Congruence Closure. *J. ACM*, 27(2):356–364, Apr. 1980.

[15] R. Nieuwenhuis and A. Oliveras. Fast congruence closure and extensions. *Information and Computation*, 205(4):557 – 580, 2007. Special Issue: 16th International Conference on Rewriting Techniques and Applications.

[16] R. Nieuwenhuis and A. Rubio. Paramodulation-Based Theorem Proving. In A. Robinson and A. Voronkov, editors, *Handbook of automated reasoning*, volume 1, pages 371–443. 2001.

[17] J. Otten. Restricting backtracking in connection calculi. *AI Commun.*, 23(2-3):159–182, 2010.

[18] G. Reger, D. Tishkovsky, and A. Voronkov. Cooperating Proof Attempts. In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, pages 339–355, 2015.

[19] A. Reynolds, C. Tinelli, and L. 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.

[20] A. Tiwari, L. Bachmair, and H. Ruess. Rigid E-Unification Revisited. In D. McAllester, editor, *Automated Deduction - CADE-17*, volume 1831 of *Lecture Notes in Computer Science*, pages 220–234. Springer Berlin Heidelberg, 2000.

Appendix A

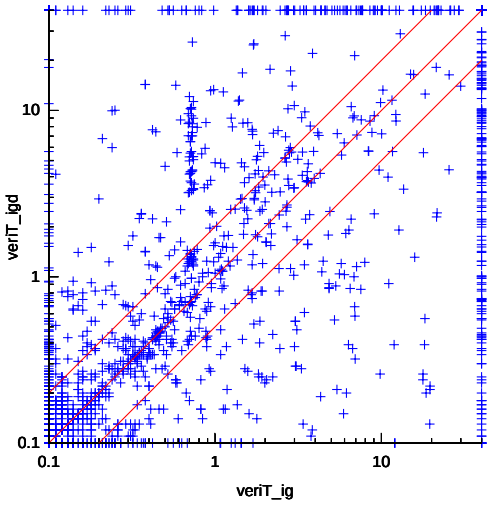


Figure 4: Comparison of the two strategies