SyMT: finding symmetries in SMT formulas

Work in progress

Carlos Areces[1], David Déharbe[2], Pascal Fontaine[3] and Ezequiel Orbe[1*]

[1] FaMAF, U. Nacional de Córdoba (Argentina)
[2] Universidade Federal do Rio Grande do Norte, (Brazil)
[3] Inria, U. of Lorraine (France)

Abstract

The QF_UF category of the SMT-LIB test set contains many formulas with symmetries, and breaking these symmetries results in an important speedup [8]. This paper presents SyMT, a tool to find and report symmetries in SMT formulas. SyMT is based on the reduction of the problem of detecting symmetries in formulas to finding automorphisms in a graph representation of these formulas. The output of SyMT may be used to improve SMT formulas to enforce the SMT solver to examine only one assignment out of many symmetric ones. We show that the classic propositional symmetry breaking technique can be lifted to SMT and yields a generic technique to break the symmetries found by SyMT.

Experiments on a large part of the SMT-LIB show that symmetries are pervasive in most categories.

1 Introduction

Consider a propositional formula $\varphi(p, q)$ with propositional variables p and q, and symmetric by permutation of p and q. Propositional symmetry breaking [12] eliminates symmetry, e.g. by adding clause $p \Rightarrow q$, since there is a Boolean model of $\varphi(p, q)$ if and only if there is a model such that $p \Rightarrow q$. Searching for models such that $p \wedge \neg q$ is unnecessary. Such transformation is a sound reduction of the search space for the SAT-solver.

Now, consider the first-order formula $\varphi(f(a) = f(b), a = b)$ with the standard interpretation of equality. It is clear that there exists no model such that $f(a) \neq f(b) \wedge a = b$ holds, although $f(a) = f(b) \wedge a \neq b$ is satisfiable; if $\varphi(p, q)$ has only models such that exactly one proposition in $\{p, q\}$ is true, $\varphi(f(a) = f(b), a = b) \wedge (f(a) = f(b) \Rightarrow a = b)$ is unsatisfiable. This simple example shows it is not sound to break symmetry of an SMT formula based on the symmetry of its propositional structure alone. Essentially the problem is that the abstraction does not take the theory into account. However, we show in the paper that it is sound to break symmetries stemming from permutation of uninterpreted symbols, similarly to what is done for propositional logic.

Previous results show [8] that exploiting symmetries in SMT formulas can lead to an impressive decrease in the size of the search space, and thus to a considerable increase in efficiency. Techniques described in [8] are, however, highly heuristic and vulnerable to formula rewriting. Graph automorphism detection algorithms [11, 9, 10] have been used to find symmetries in propositional formulas. Based on such techniques we developed SyMT, a tool to find and report symmetries in SMT formulas. In essence, the tool rewrites the SMT formula into a graph while preserving the syntactic symmetries. The resulting graph is suitable for input to graph

*This work has been partially supported by the European Union Seventh Framework Programme under grant agreement no. 295261 (MEALS), and by CNPq/INRIA project SMT-SAVeS, and CNPq grant 573964/2008-4 (National Institute of Science and Technology for Software Engineering—INES).

isomorphism tools; in particular, there is no ordering on nodes' children. The output of SyMT provides to users the information they need to rewrite their SMT formulas so that they have no symmetries and are easier to solve. Such transformation is highly heuristic and domain-specific and it is the SMT user that is in the best position to realize it. In future work, we however plan to design of concrete generic symmetry breaking heuristics, that would provide further indication to users about how they can improve their formulas.

Outline. We first give a formal basis for symmetry breaking in SMT, then present our SyMT tool for detecting symmetries in SMT formulas. Section 3 also introduces the translation from SMT formulas to graphs. Some statistics on symmetry detection on a large part of the SMT-LIB [5] are given. They clearly show that (1) graph automorphism algorithms scale for SMT formulas, and (2) the SMT-LIB contain many highly symmetric formulas.

2 Symmetries in SMT

We assume knowledge of basic notions of permutation group theory, such as generators and cyclic forms. We use the standard notions of multi-sorted logic, term, formula, and interpretation commonly used in the context of SMT. A theory is a set of interpretations. Consider a finite set $\mathcal{S}$ of uninterpreted symbols (for constants, functions or predicates), and a bijective function σ on $\mathcal{S}$, that maps every symbol to a symbol of the same sort (i.e., arity and sorts of arguments and image should match). Function σ extends naturally to terms and formulas, and $t\sigma$ denotes σ applied to term or formula t, just like a higher-order substitution would, considering symbols in $\mathcal{S}$ as variables. σ can also be applied on an interpretation $\mathcal{I}$ to yield interpretation $\mathcal{I}\sigma$ similar to $\mathcal{I}$ except that $\mathcal{I}\sigma[s'] = \mathcal{I}[s]$ whenever $s\sigma = s'$. The identity function is denoted σ_I.

We say that σ is a symmetry for formula φ if $\varphi\sigma$ is syntactically equal to φ up to satisfiability preserving rewritings, e.g. using commutativity of some interpreted symbols. Notice that if σ is a symmetry for φ, so is any of its powers σ^i, and in particular σ^{-1} is also a symmetry of φ since there exists n such that $\sigma^n = \sigma_I$. The case where σ is its own inverse ($\sigma^2 = \sigma_I$) is a particular, though extremely frequent, case. It occurs when there is a group that contains all permutations of elements in a subset of $\mathcal{S}$. In our experiments on the SMT-LIB test bed, we have observed that most symmetry groups found have a set of generators that are their own inverse; in the following we will only consider such symmetries. Let us thus consider a symmetry σ such that $\sigma^2 = \sigma_I$ for a formula φ. For every interpretation $\mathcal{I}$ of φ we have $\mathcal{I}\sigma[\varphi] = \mathcal{I}[\varphi]$ (using straightforward structural induction). Consider now a set of atoms (not necessarily simple propositional variables) $p_1, \dots p_n$ and their image $q_1 = p_1\sigma, \dots q_n = p_n\sigma$. If φ is satisfiable in a model $\mathcal{M}$ then there exists a model of φ that furthermore satisfies the following formulas for $i \in \{1..n\}$:

$$\psi_i =_{\text{def}} \Big(\bigwedge_{1 \le j < i} p_j \equiv q_j \Big) \Rightarrow (p_i \Rightarrow q_i).$$

This model is indeed either $\mathcal{M}$ or $\mathcal{M}\sigma$. Assume k is the smallest value for which $\mathcal{M}[p_k] \neq \mathcal{M}[q_k]$, and consider ψ_k. If $\mathcal{M}[p_k] = \bot$ and $\mathcal{M}[q_k] = \top$ then $\mathcal{M}$ satisfies ψ_k, as well as all ψ_i with $i \neq k$. Now, if $\mathcal{M}[p_k] = \top$ and $\mathcal{M}[q_k] = \bot$ then $\mathcal{M}\sigma$ is a model of φ such that $\mathcal{M}\sigma[p_i] = \mathcal{M}\sigma[q_i]$ for $i < k$ and $\mathcal{M}\sigma[p_k] = \top$ and $\mathcal{M}\sigma[q_k] = \bot$. The model $\mathcal{M}\sigma$ of φ thus satisfies ψ_i for $i \in \{1..n\}$.

It is well known (see, e.g., [12]) that the formulas ψ_i can serve to break symmetry for propositional formulas. The above shows that this extends to SMT. This leaves out, however, many choices for the set of atoms p_i: the insight of the SMT user is usually necessary to make the best choice.

3 SyMT Implementation

SyMT is a command line tool implemented in C that detects symmetries in SMT formulas, taking into account the commutativity of conjunction, disjunction, addition, multiplication and equality. Given an input SMT formula, SyMT proceeds by creating a colored graph from it and then uses a graph automorphism component to detect the generators of the automorphism group of the colored graph. In particular, SyMT uses Saucy 3.0 [10] as the graph automorphism component. Integration with Saucy is done via Saucy's C API. SyMT also provides simplification capabilities on the input formulas, some of which involve using theory reasoning (and thus may unfortunately fail on large instances). Simplification of the input formula is important because it may uncover hidden symmetries and remove trivial symmetries, e.g., symmetries that do not involve uninterpreted symbols. Simplifications include simple rewriting, simplification of entailed literals, and some normalization of terms and formulas.

Example 1. *The command line and output of SyMT on a formula of the QF_UF category of SMT-LIB is as follows:*

```
./SyMT --enable-simp smt-lib2/QF_UF/NEQ/NEQ004_size4.smt2
(p7 p9)(c12 c13)
(c_3 c_1)
(c_2 c_1)
(c_0 c_1)
```

SyMT finds four generators for the symmetry group, and prints them in cyclic form. There is the full group of permutations for constants `c_0`*,* `c_1`*,* `c_2`*,* `c_3`*, generated by the last three generators, as well as a further symmetry that permutes unary predicate symbols* `p7` *and* `p9`*, as well as constant symbols* `c12` *and* `c13`*. This last symmetry was not detected with the heuristic techniques of [8].*

Reduction to the colored graph automorphism problem is the most successful technique for detecting symmetries in propositional formulas in clausal form, primarily due to the availability of efficient tools to detect graph automorphisms (e.g., [11, 9, 10]) that are fast and easy to integrate. Several reductions from propositional formulas to colored graphs have been proposed [6, 7, 1], all based on the same idea: to use the formula to construct a colored graph whose automorphism group is isomorphic to the symmetry group of the formula. Also, extensions to other logics, e.g., QBF [3] and modal logics [2], have been proposed, further showing the applicability of this technique. Nevertheless, as far as we know, there is no extension of this technique to the case of SMT formulas.

We now present the reduction algorithm to colored graphs for SMT formulas. The reduction is as a two-stage process. First, SyMT constructs the syntax direct acyclic graph of the formula with some additional nodes. Second, colors are introduced, to avoid spurious symmetries. Colors are represented as natural numbers. Let φ be an SMT formula. The colored graph $G(\varphi)$ is constructed recursively as follows ($=$ and other predicates, and propositions are considered as functions and constants ranging over Booleans):

- **Graph Construction:**

  1. For each symbol, add a unique *symbol* node.
  2. For each (constant or propositional) term without argument, the *root* node is the symbol node introduced above.
  3. For each term $f(t_1, \ldots, t_n)$ of arity $n > 0$,

(a) Add a *root* node for $f(t_1, \ldots, t_n)$. Add an edge from the root node to the (unique) symbol node for f.

(b) If the function is commutative (e.g. $\wedge$, $\vee$, $\equiv$, $=$, $+$, $*$), add an edge from the root node to the root node of t_i ($i \in \{1..n\}$). Quantifiers, as commutative operators, are handled similarly (coloring discriminates the matrix).

(c) If the function is not commutative:

i. For each argument t_i, add an *argument node* and an edge from this node to the root node of t_i.

ii. Add an edge from the argument node of t_i to the argument node of t_{i+1} ($1 \leq i < n$). These edges represent the ordering of the arguments in $f(t_1, \ldots, t_n)$.

iii. Add an edge from the root node to the argument node of t_1.

- **Graph Coloring:**

1. Argument nodes are assigned a specific, unique color.
2. Uninterpreted symbol nodes and root nodes are assigned a color based on their sort (Boolean being considered as any other sort).
3. Each interpreted symbol node is assigned a unique color.

Example 2. *Consider formula $\varphi = p(f(a,b)) \vee p(f(b,a)) \vee p(g(a,b)) \vee p(g(b,a))$, where p is a unary predicate symbol and f, g, a and b are uninterpreted symbols. The associated colored graph, $G(\varphi)$, is shown in Figure 1 (colors are represented by numeric labels and node shapes in the figure).*

Theorem 1. *Let φ be an SMT formula and $G(\varphi)$ the colored graph constructed from it. Then, every automorphism of the graph $G(\varphi)$ is a symmetry of the formula φ.*

Proof sketch: This follows directly by structural induction and from the following observations:

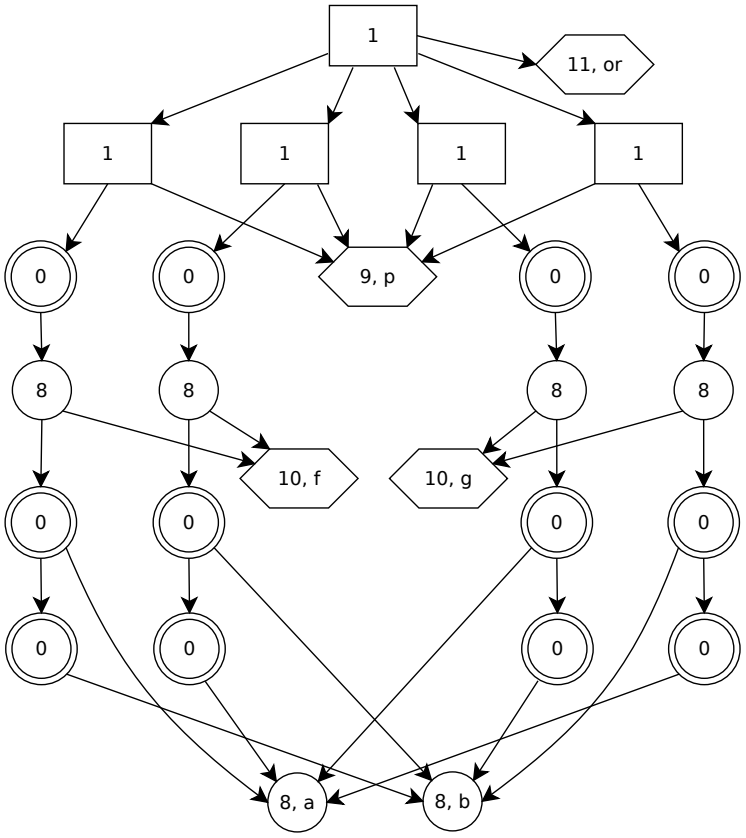
- The graph $G(\varphi)$ is the syntax directed acyclic graph of φ plus additional nodes.

- For terms, $f(t_1, \ldots, t_n)$ of arity > 0, the coloring of nodes, and the combination of root nodes and symbol nodes ensures that only symbols nodes of the same sort (i.e., same arity and same argument sorts) and with the same number of occurrences can be permuted.

- For terms without arguments (constants and predicates), the coloring of symbol nodes and the existence of argument nodes ensures that only symbols of the same sort and occurring the same number of times in the same argument positions can be permuted.

Finally, to reconstruct a formula symmetry from a graph automorphism, we just need to restrict the graph automorphism to symbol nodes. □

Notice that the converse of Theorem 1 is not true: the proposed graph construction does not find all the symmetries of the input formula. For example, consider the formula $\varphi = f(a,b) \wedge g(c,d)$ where a, d are of some sort and b, c of another sort (with appropriate sorts for f and g. The permutation $\sigma = (f\ g)(a\ c)(b\ d)$ is a symmetry of φ. Nevertheless, we can not detect it in the graph $G(\varphi)$. This is due to the fact that symbol nodes are colored based on the symbol sort, and this prevent the automorphism component from detecting permutations involving symbols of different sorts. Nevertheless, from a practical point of view, symmetries involving symbols of different sorts are rather unnatural and do not arise often.



Generator 1: $(a\ b)$
Generator 2: $(f\ g)$

Figure 1: Graph representation of $p(f(a,b)) \vee p(f(b,a)) \vee p(g(a,b)) \vee p(g(b,a))$.

4 Symmetries in SMT-LIB

We test SyMT against 19 categories[1] from SMT-LIB [5] to investigate the existence of symmetries and evaluate the efficiency of our tool. All tests are run on an Intel Xeon X3440 with 16GB, using the four cores simultaneously and we report the cumulative core time (roughly 4 times the CPU time). Three different configurations of SyMT were tested. Configuration 1 has no simplification: the formula is parsed and converted to a graph for automorphism detection. Configuration 2 uses trivial syntactic simplifications. Configuration 3 enables stronger simplifications, using an SMT engine, e.g., simplification of atoms implied by unit clauses. Configuration 2 may fail (with no symmetry reported) because the simplification algorithm used is not linear with respect to the input formula. However it often reveals symmetries hidden by irrelevant garbage easily removed by the simplification procedure. Configuration 3 is likely to fail on very large formulas, but again, it may reveal hidden symmetries. Simplification sometimes reduces a formula to false, in which case no symmetry is reported. The timeout (relevant for configuration 2 and, foremost, for configuration 3) is set to 30 seconds.

Among the 19 analyzed categories, three (LRA, QF_UFLRA, QF_UFNRA) do not reveal symmetries with SyMT. Of the only five formulas in UFLRA, one has symmetries. The others 15 categories presented a significant number of symmetries in at least one of the tested

[1] Bit vectors are not supported by our parser.

Category	#Inst	#Sym[1]	#Sym[2]	#Sym[3]	#Sym[P]	Avg[GS]	Time
AUFLIA	6480	6212	6231	5941	6258	134.00	378.79
AUFLIRA	19917	15779	16475	12500	16476	1.08	9.13
AUFNIRA	989	985	985	923	985	1.00	0.41
QF_AUFLIA	1140	2	71	77	78	1.00	0.72
QF_AX	551	22	22	22	22	1.00	0.37
QF_IDL	1749	348	526	683	756	12745.43	327.95
QF_LIA	5938	728	1172	524	1200	104.55	486.19
QF_LRA	634	73	150	208	210	110.49	29.06
QF_NIA	530	169	169	168	169	5.92	3.92
QF_NRA	166	9	43	43	43	1.00	0.23
QF_RDL	204	0	0	24	24	0.00	10.13
QF_UF	6639	250	3638	375	3638	44.00	34.58
QF_UFIDL	431	19	175	186	189	1.00	2.70
QF_UFLIA	564	0	198	198	198	0.00	0.45
UFNIA	1796	1062	1061	1058	1070	47.08	543.26

Table 1: Symmetries in SMT-LIB

configurations. Table 1 summarizes the results obtained for these 15 categories. For each category we report the number of instances (#Inst), the number of instances that have symmetries for the various simplification configurations (#Sym[1], #Sym[2] and #Sym[3]), the number of instances that have symmetries in at least one of the configurations (#Sym[P]), the average logarithm in base 2 of the size of the symmetry group (Avg[GS]) for Configuration 1, and the total time in seconds required to analyze all the instances (Time) also for Configuration 1. It is clear from Table 1 that the SMT-LIB has many highly symmetric formulas, in most categories. The cumulative time required to build the graph and detect the symmetries is negligible in all categories. We do not output the times for other configurations since there are timeouts and time is dominantly spent in the simplification modules, so these numbers give little insight about symmetry detection itself. The above experiments are using Saucy as the graph isomorphism detection engine. We also investigated Bliss as an alternative, with similar results. Unfortunately, this alternative is currently unavailable for users because of license issues.

The current tool fails to find some symmetries in the QF_UF category although they are discovered with the heuristic techniques from [8]. We are investigating the issue.

5 Conclusions and future work

We presented SyMT, a tool to detect symmetries in SMT formulas. SyMT is based on the reduction of the symmetry detection problem to graph automorphism detection. We presented the corresponding graph construction algorithm and showed that symmetry detection scales on SMT formulas by providing experimental results on executions of the tool on many SMT-LIB categories. We also showed that propositional symmetry breaking can be lifted to the SMT case, which provides a simple symmetry breaking mechanism for SMT.

In future work we will address the issue of symmetry breaking. We want to study the structures of symmetry groups found by SyMT. A deeper understanding of these structures may provide useful information to develop generic symmetry breaking mechanisms. We also believe that, to fully exploit the presence of symmetries in formulas, *ad hoc*, application-tailored,

heuristics are also necessary. We will use SyMT to mine the SMT-LIB to find symmetries, and we will devise appropriate heuristics integrated into an SMT symmetry breaking pre-processor. We expect this will result in a significant speed up for solving the formulas in the repository, since our experiments show symmetries are pervasive in many SMT test sets. We plan to carry out a similar analysis on the TPTP library [13].

We are aware that symmetry breaking is essentially heuristic, and a compilation of *ad hoc* heuristics would not be a silver bullet: the expertise of the user is generally the best approach to break symmetries. The current version of SyMT already provides the SMT users with a simple, yet powerful, tool to detect symmetries.

The tool and its source are available for download under the BSD License at `http://www.veriT-solver.org/SyMT`. It uses the Saucy 3.0 source code, distributed under its own specific license.

Acknowledgements: We thank Stephan Merz for interesting discussions, Cesare Tinelli for encouraging to investigate further symmetries in SMT, and the anonymous reviewers for their comments. We are very grateful to the Saucy developers for their tool.

References

[1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Solving difficult instances of Boolean satisfiability in the presence of symmetry. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(9):1117–1137, 2003.

[2] C. Areces, G. Hoffmann, and E. Orbe. Symmetries in modal logics: A coinductive approach. In *Proc. of the 7th Workshop on Logical and Semantic Frameworks, with Applications (LSFA 2012)*, Rio de Janeiro, September 2012.

[3] G. Audemard, B. Mazure, and L. Sais. Dealing with symmetries in quantified Boolean formulas. In *Proc. of SAT'04*, pages 257–262, 2004.

[4] Clark Barrett, Roberto Sebastiani, Sanjit Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, chapter 26, pages 825–885. IOS Press, February 2009.

[5] Clark Barrett, Aaron Stump, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2010.

[6] J. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic. In *Proc. of AAAI Workshop on Tractable Reasoning*, pages 17–22, 1992.

[7] James Crawford, Matthew Ginsberg, Eugene Luks, and Amitabha Roy. Symmetry-breaking predicates for search problems. In Luigia Carlucci Aiello, Jon Doyle, and Stuart Shapiro, editors, *KR*, pages 148–159. Morgan Kaufmann, 1996.

[8] David Déharbe, Pascal Fontaine, Stephan Merz, and Bruno Woltzenlogel Paleo. Exploiting symmetry in SMT problems. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 222–236. Springer, 2011.

[9] Tommi Junttila and Petteri Kaski. Engineering an efficient canonical labeling tool for large and sparse graphs. In David Applegate, Gerth Brodat, Daniel Panario, and Robert Sedgewick, editors, *Proc. of the 9th Workshop on Algorithm Engineering and Experiments and the 4th Workshop on Analytic Algorithms and Combinatorics*. SIAM, 2007.

[10] Hadi Katebi, Karem Sakallah, and Igor Markov. Conflict anticipation in the search for graph automorphisms. In Nikolaj Bjørner and Andrei Voronkov, editors, *LPAR*, volume 7180 of *LNCS*, pages 243–257. Springer, 2012.

[11] B. McKay. Nauty user's guide. Technical report, Australian National University, Computer Science Department, 1990.

[12] Karem Sakallah. Symmetry and satisfiability. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 289–338. IOS Press, 2009.

[13] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0. *Journal of Automated Reasoning*, 43(4):337–362, 2009.