# CVC4-SymBreak: Derived SMT solver at SMT Competition 2019

Saket Dingliwal, Ronak Agarwal, Happy Mittal, and Parag Singla

Indian Institute of Technology, Delhi
{cs1150254, cs1150252, csz138233, parags}@cse.iitd.ac.in

**Abstract**

We present CVC4-SymBreak, a derived SMT solver based on CVC4 [1], and a non-competing participant of SMT-COMP 2019 [13]. Our technique exploits symmetries over the Boolean skeleton variables in an SMT problem to prune the search space. We use an ensemble of a solver with and without symmetries to be more effective. Our approach results in significantly faster solutions on a subset of available SMT benchmarks.

## 1 Introduction

Satisfiability Modulo Theories (SMT) is a decision problem for logical first order formulas combined with operations defined over additional constructs such as integers, reals, arrays, and uninterpreted functions [8]. SMT solvers have been useful both for verification, i.e., proving the correctness of programs based on symbolic execution, as well as for synthesis, i.e., generating programs by searching over the space of possible program fragments [12]. SMT-COMP [9] has pushed the state-of-the-art by allowing SMT solvers to complete on a set of benchmark problems divided across a number of categories based on the theory used, e.g., NIA, non linear integer arithmetic, etc. In this work, our primary goal is to exploit inherent symmetry in the SMT problems to improve the efficiency of existing solvers such as CVC4 [1]. Specifically, we exploit symmetry over Boolean skeleton variables, i.e., those obtained by replacing the atomic theory operations with equivalent Boolean variables in SMT solving.

Symmetry breaking [2] has been an effective technique for improving the efficiency of propositional logic solvers for a long time. It involves identifying variable permutations (known as symmetry permutations) applying which does not alter the theory, and then using them to add constraints to the problem without changing it's satisfiability and thereby reducing the search space. The constraints added are known as Symmetry Breaking Predicates (SBPs). A number of improvements over the original SBP formulation have been proposed to make them effective for use in modern day SAT solvers [4], [3].

The first attempt to use symmetries for SMT problems was made in [5]. Symmetries were obtained by designing a tool SyMT [6] which converted the problem of finding symmetries over SMT variables into one of finding graph automorphism. Their use of symmetry was limited to exchangeability of constants in QF-UF category of SMT benchmarks. In this work, we extend the original idea significantly by (1) additionally exploiting symmetry defined over Boolean skeleton variables (2) allowing for a relaxed symmetry class which is not limited to exchangeability of constants. Our final solver is an ensemble of the original solver and the one which exploits symmetries.

## 2 Background: CVC4 and SyMT

Our solver is built on top of CVC4 [1] which was a winner in a large number of categories in the SMT 2018 [10] competition. The basic algorithm for SMT solving in CVC4 involves first replacing the atomic theory operations with equivalent Boolean skeleton variables over which a statisfiability solver is used. The resulting assignment is then used to check for consistency across theory variables using a theory solver. A conflict clause is added if no corresponding assignment is found. This process is repeated until a satisfying assignment is found or the solver terminates resulting in unsatisfiability. CVC4 uses a number of enhancements over this basic SMT procedure [1]. SyMT [6] was developed as a tool to find symmetries over variables in an SMT problem. It reduces the problem to one of graph automorphism which is then solved by using existing solvers such as SAUCY [11]. We make use of this tool to discover symmetries over Boolean skeleton variables.

# 3 Technical Innovations

## 3.1 Symmetry Breaking Technique

| | |
|---|---|
| SMT Problem $\Omega$ | $(x < 8) \wedge (y < 8) \wedge ((x + y < 10) \vee (x + y > 3))$ |
| Boolean Skeleton $\Psi$ | $Q \wedge R \wedge (S \vee T)$ |
| Constraints Set $\Phi$ | $(Q \Leftrightarrow (x < 8)) \wedge (R \Leftrightarrow (y < 8)) \wedge (S \Leftrightarrow (x + y < 10)) \wedge (T \Leftrightarrow (x + y > 3))$ |
| Symmetry Permut. $\theta$ | $\theta(Q) = R$; $\theta(R) = Q$; $\theta(x) = y; \theta(y) = x$ |
| SBP added $\mathcal{T}$ | $Q \Rightarrow R$ |
| New Skeleton $\Psi'$ | $Q \wedge R \wedge (S \vee T) \wedge (\neg Q \vee R)$ |

We explain our technical innovation with the help of an example. In the table shown above, $\Omega$ denotes the original SMT problem. Equivalently, a Boolean skeleton $\Psi$ is formed by replacing the theory operations by (new) Boolean variables, while enforcing the consistency constraints given by $\Phi$. A symmetry ($\theta$) is defined as a permutation which when applied to the variables in $\Psi$ and $\Phi$ results in identical formulation for the skeleton as well as the constraint set. Then, it is straightforward to show that if $M$ is a model (satisfying assignment) of the SMT problem then $\theta(M)$ is also a model $\theta$ being a symmetry. Further, the theory of symmetry breaking ensures that only the lexicographically smaller model (under a given variable ordering) between $M$ and $\theta(M)$ is chosen resulting in pruning of the search space. This is achieved by adding a predicate of the form: $\wedge_{1 \leq i \leq n}(\wedge_{1 \leq j < i} v_j = \theta(v_j)) \Rightarrow v_i \Rightarrow \theta(v_i)$, known as SBP [2]. Here, $n$ is the total number of variables, and $v_1, v_2 \cdots, v_n$ is the variable ordering. The key innovation here is the application of these SBPs over the Boolean skeleton variables (only) while ensuring the corresponding symmetries hold for the variables in the skeleton as well as the constraint set. This results in a sound formulation preserving the satisfiability. The SBP ($\mathcal{T}$) in the example above is $Q \Rightarrow R$, under the variable ordering $Q, R, S, T$. We have designed some heuristics in choosing the variable ordering in order to maximize the impact of SBPs formed. Further, we restrict the size of the SBP formed to reduce overhead as done by earlier work [3].

## 3.2 Ensemble Technique

In our experiments, we found that for several problems adding SBPs resulted in significant speed-ups, whereas in other cases, it slowed down the original solver. We hypothesize this is because in the former case, unnecessary parts of the search space are pruned resulting in speed-up, whereas in the latter case, the SBPs might be pruning valid models, resulting in further delay in reaching a satisfying assignment. Taking insight from this, we design an ensemble solver, where we first run the solver with SBPs added for a fixed time $t$, and if the solution is still not found, we resort to the original solver with SBPs removed. $t$ is a hyperparameter. This results in significant boost in performance compared to CVC4 on a subset of SMT benchmarks.

## Acknowledgements

We acknowledge the use of the following tools/solvers for making our solver CVC4-SymBreak.

- **CVC4 and it's dependencies:** The base solver used is competition build of CVC4 2018 version. CVC4 is modified to add new clauses(SBPs) to the Boolean skeleton formed.
- **SyMT and it's dependencies:** SyMT [6] is used to detect symmetries in the SMT problem. It converts the problem into graph automorphism problem and uses SAUCY [11] internally for solving it. It also requires binary for VeriT [7] for it's functioning.

We are thankful to Kuldeep Meel and Supratik Chakraborty for helpful discussions and inputs on the work. Happy Mittal is supported by the TCS Research Scholar Program. Parag Singla is supported by the Visvesvaraya Young Faculty Fellowships by Govt. of India and IBM SUR awards. Vibhav Gogate is supported by the National Science Foundation grants IIS-1652835 and IIS1528037. Both Parag and Vibhav are also supported by the DARPA Explainable Artificial Intelligence (XAI) Program with number N66001-17-2-4032. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views or official policies, either expressed or implied, of the funding agencies.

## References

[1] Barrett, Clark, et al. "CVC4" International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2011.

[2] Crawford, James, et al. "Symmetry-breaking predicates for search problems." KR 96 (1996)

[3] Aloul, Fadi A., Karem A. Sakallah, and Igor L. Markov. "Efficient symmetry breaking for boolean satisfiability." IEEE Transactions on Computers 55.5 (2006)

[4] Devriendt, J., and B. Bogaerts. "BreakID, a symmetry breaking preprocessor for sat solvers." bitbucket. org/krr/breakid (2015).

[5] Dharbe, David, et al. "Exploiting symmetry in SMT problems." International Conference on Automated Deduction. Springer, Berlin, Heidelberg, 2011.

[6] Areces, Carlos, et al. "SyMT: finding symmetries in SMT formulas." SMT Workshop 2013 11th International Workshop on Satisfiability Modulo Theories. 2013.

[7] Bouton, Thomas, et al. "veriT: an open, trustable and efficient SMT-solver." International Conference on Automated Deduction. Springer, Berlin, Heidelberg, 2009.

[8] De Moura, Leonardo, and Nikolaj Bjrner. "Z3: An efficient SMT solver." International conference on Tools and Algorithms for the Construction and Analysis of Systems. Springer, Berlin, Heidelberg, 2008.

[9] Barrett, Clark, Leonardo De Moura, and Aaron Stump. "SMT-COMP: Satisfiability modulo theories competition." International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, 2005.

[10] Heinzmann, M., G. Regers, and T. Weber. "11th International Satisfiability Modulo Theories Competition (SMT-COMP 2018): Rules and Procedures." Available in http://smtcomp.sourceforge.net/2018/rules18.pdf.

[11] Katebi, Hadi, Karem A. Sakallah, and Igor L. Markov. "Conflict anticipation in the search for graph automorphisms." International Conference on Logic for Programming Artificial Intelligence and Reasoning. Springer, Berlin, Heidelberg, 2012.

[12] https://en.wikipedia.org/wiki/Satisfiability_modulo_theories#Applications

[13] https://smt-comp.github.io/2019/index.html