Moving Definition Variables in Quantified Boolean Formulas* (Extended Abstract)

Joseph E. Reeves ✉, Marijn J. H. Heule, and Randal E. Bryant

Carnegie Mellon University, Pittsburgh, Pennsylvania, United States
{jereeves,mheule,randy.bryant}@cs.cmu.edu

Boolean formulas and circuits can be translated into conjunctive normal form (CNF) by introducing *definition variables* to augment the existing *problem variables*. The definition variables allow for compact representations of the problem. Definition variables are introduced through a set of *defining clauses*, given by the Tseitin [10] or Plaisted-Greenbaum [9] transformation. Problem variables occurring in the defining clauses constitute the *defining variables*; they effectively determine the values of the definition variables. In CNF, definitions are not an explicit part of the problem representation, preventing solvers from using this structural information. Quantified Boolean formulas (QBF) extend CNF into prenex conjunctive normal form (PCNF) with the addition of quantifier levels. In practice, definition variables are usually placed in the innermost quantifier level. However, as we show, placing a definition variable in the quantifier level immediately following its defining variables can improve solver performance.

We describe a preprocessing technique for moving definition variables to the quantifier level of their innermost defining variables. As a starting point, existing tools KISSAT and CNFTOOLS can detect definitions in a CNF formula. Bounded variable elimination (BVE) [4] has been an essential preprocessing technique in SAT solving. The technique relies on definitions, so most SAT solvers incorporate some form of definition detection. The conflict-driven clause learning SAT solver KISSAT [1] extends the commonly used syntactic pattern matching with semantic definition detection. The detection is applied to variables independently. Alternatively, the preprocessor CNFTOOLS [6] performs hierarchical definition detection, capturing additional information about definition variable nesting and monotonic definitions. These tools run on CNF formulas. A QBF can be transformed into a CNF by removing the prefix, but not all definitions in the CNF are valid w.r.t. the prefix.

In our experiments, we ran the tools for 10 seconds on each formula. In general, KISSAT found many more definitions than the two configurations of CNFTOOLS we tested. However, CNFTOOLS uses hierarchical detection allowing the discovery of monotonically nested definitions. Therefore, running each tool sequentially for 10 seconds each then combining the found definitions yielded the best coverage.

Once all definitions are found, we process and order the candidate definitions, moving definition variables sequentially. For each instance of movement we generate a proof in the QRAT proof system that, through a series of clause additions and deletions, effectively replaces the old definition variable with a new variable at the desired quantifi-

* The authors are supported by the NSF under grant CCF-2108521.

cation level. The proof shows that the transformation to the formula is *truth-preserving*, meaning that if the formula was false (resp. true), it remains false (resp. true) after variable movement. A detailed description of the QRAT proof steps can be found in the original work. Proof generation is difficult for QBF and relatively uncommon in solvers. The QBF preprocessor BLOQQER [2] generates QRAT proofs [5] for all of the transformations it performs. On select formulas where BLOQQER found a solution after variable movement, we were able to append the proof of movement with the BLOQQER proof to get a complete proof for the formula. These proofs were verified with the independent proof checker QRAT-TRIM.

Clausal-based QBF solvers rely on preprocessing to improve performance. Almost every top-tier solver in the QBFEVAL'20 competition[1] used some combination of BLOQQER, HQSPRE [11], or QBFRELAY [8]. Some solvers incorporate preprocessing techniques into the solving phase, e.g., DEPQBF's [7] use of dynamic quantified blocked clause elimination. Unlike other preprocessing techniques, variable movement does not add or remove clauses or literals. However, it can prompt the removal of literals through *universal reduction* and may guide solver decisions in a beneficial way.

Table 1. The number of instances solved within the 5,000 time-limit over benchmarks where variable movement was possible.

Solver	Original	Moved	BLOQQER	Moved-BLOQQER
CAQE	74	84	99	**103**
GHOSTQ(p)	55	**61**	47	52
GHOSTQ(ce)	77	**80**	65	70
RAREQS	72	72	94	**98**
DEPQBF	64	70	64	**71**

In our experimental setup, we ran five QBF solvers on the 157 QBFEVAL'20 competition benchmarks where movement was possible. Solvers ran with a 5,000 second timeout, which included definition detection and proof generation, adding 50 seconds on average. The solvers were ran on each formula four times, with no preprocessing (original), with variable movement, with BLOQQER preprocessing for 100 seconds, and with movement then BLOQQER. Table 1 shows that variable movement always improves solver performance with and without BLOQQER. Movement significantly improves performance of CAQE, DEPQBF, and GHOSTQ-p (plain mode). GHOSTQ-ce (CEGAR mode) and RAREQS improve slightly with movement.

We also examined variable movement on the LDomino benchmark with the solver PGBDDQ [3]. PGBDDQ is a proof generating binary decision diagram (BDD) based solver. With a clever variable ordering and placement of definition outputs next to their inputs, PGBDDQ can solve the LDomino benchmark in polynomial time in relation to the exponential blow-up of other QBF solvers. However, moving the definition outputs to the innermost quantifier level ruins PGBDDQ's ability to leverage the structure

[1] available at http://www.qbflib.org/qbfeval20.php

of the problem. To make PGBDDQ more robust against variable placement, we developed our technique for automatically moving definition outputs to their outermost input. We found that variable movement significantly improved performance versus formulas where definition variables were placed in the innermost quantifier level. However, PGBDDQ still performed better with a handcrafted variable placement. This discrepancy is the focus of future work.

In conclusion, we presented a technique for moving definition variables in QBFs. The movement can be verified within the QRAT proof system, and we validated all proofs in the evaluation with QRAT-TRIM. Using the tools KISSAT and CNFTOOLS to detect definitions, we created a tool-chain for variable movement. On the QBFEVAL'20 benchmarks, one quarter of formulas had definitions that could be moved, and the movement increased solver performance. In addition, we found that movement followed by BLOQQER was more effective than preprocessing with BLOQQER.

This research appeared in TACAS'22 under the same title. The experiment data and tool artifact is archived at https://zenodo.org/record/5733440.

References

1. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020. Tech. rep. (2020)
2. Biere, A., Lonsing, F., Seidl, M.: Blocked clause elimination for QBF. In: Automated Deduction (CADE). pp. 101–115. Springer (2011)
3. Bryant, R.E., Heule, M.J.H.: Dual proof generation for quantified Boolean formulas with a BDD-Based solver. In: Automated Deduction (CADE). pp. 433–449. Springer (2021)
4. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Theory and Applications of Satisfiability Testing (SAT). LNCS, vol. 3569, pp. 61–75. Springer (2005)
5. Heule, M.J.H., Seidl, M., Biere, A.: A unified proof system for QBF preprocessing. In: Automated Reasoning. pp. 91–106. Springer, Cham (2014)
6. Iser, M.: Recognition and Exploitation of Gate Structure in SAT Solving. Ph.D. thesis, Karlsruhe Institute of Technology (KIT) (2020)
7. Lonsing, F.: Dependency Schemes and Search-Based QBF Solving: Theory and Practice. Ph.D. thesis, Johannes Kepler University (JKU) (2012)
8. Lonsing, F.: QBFRelay, QRATPre+, and DepQBF: Incremental preprocessing meets search-based QBF solving. Journal on Satisfiability, Boolean Modeling and Computation **11**, 211–220 (09 2019)
9. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. Journal of Symbolic Computation **2**(3), 293–304 (1986)
10. Tseitin, G.S.: On the Complexity of Derivation in Propositional Calculus, pp. 466–483. Springer (1983)
11. Wimmer, R., Reimer, S., Marin, P., Becker, B.: HQSpre – an effective preprocessor for QBF and DQBF. In: Tools and Algorithms for the Construction and Analysis of Systems. pp. 373–390. Springer (2017)