Circuit Based Quantification: Back to State Set Manipulation within Unbounded Model Checking

Gianpiero Cabodi Marco Crivellari Sergio Nocco Stefano Quer†
Politecnico di Torino
Dip. di Automatica e Informatica
Turin, ITALY

Abstract

In this paper a non-canonical circuit-based state set representation is used to efficiently perform quantifier elimination. The novelty of this approach lies in adapting equivalence checking and logic synthesis techniques, to the goal of compacting circuit based state set representations resulting from existential quantification. The method can be efficiently combined with other verification approaches such as inductive and SAT-based pre-image verifications.

1. Introduction

In the Unbounded Model Checking domain, quantifier elimination within Quantified Boolean Formulas is of uttermost importance. Traditional methodologies resort to BDD, or BDD-like, representations. These suffer the well known memory explosion problem, due to their canonicity.

In our work we select circuit-based state set representations based on And Inverted Graphs [3] (AIGs) as underlying structure. Performing quantifier elimination on this representation implies, in the worst case, doubling the formula size. Since post-image and pre-image computations involve existential quantification of input and state variables, we risk an exponential memory blow-up. To aggressively fight circuit size explosion, we resort to combinational equivalence checking and logic synthesis optimization techniques. Given a quantifying variable and its two cofactors, we divide the existential quantification of that variable in two subtasks:

1. A merge phase, in which we resort to equivalence proofs to maximize sub-circuit sharing between the circuit representations of the two cofactors.
2. An optimization phase, in which we exploit logic synthesis based transformations (such as redundancy removal, factorizations, and simplifications under don't care conditions) on the resulting circuit.

†Contact person; e-mail: stefano.quer@polito.it.

2. Circuit Based Quantification

Let us work on a given function f, represented by a single output Boolean Circuit (an AIG in our implementation). Let us existentially quantify variable x in the support of f. We call f_0 and f_1 the two cofactors of f w.r.t. x, i.e., $f_0 = f_{\neg x}$, and $f_1 = f_x$. We perform existential quantification as $\exists_x f = (f_0 \vee f_1)$. Let us discuss now how we fight the size explosion of $(f_0 \vee f_1)$.

2.1. Node Merging by Equivalence Checks

The goal of this phase is to merge together as many internal nodes of f_0 and f_1 as possible. This is essentially a combinational equivalence checking problem, as we need to find equivalent nodes in f_0 and f_1. We rely on three steps:

1. We exploit AIG semi-canonicity and hashing scheme to early detect functionally equivalent map points.
2. We operate BDD sweeping [4] as a further enhancement of merge points detection.
3. We switch to SAT based checks for the non merged compare points. We presently rely on a general SAT solver, i.e., ZChaff, but we plan to experiment with circuit-SAT in the future.

During the third step, several SAT problems, which share most of their initial clause database, are generated. In order to exploit history information and avoid restarting at any new check, we implemented our SAT-merge routine on top of ZChaff. We load the clause database once and for-all, and we factorize several checks together within a single ZChaff run. Any SAT solver solution thus potentially rules-out several non matching couples, whereas an un-SAT verdict covers many matching points. Furthermore, we have experienced (recursive) backward and forward techniques. Backward processing is generally better in case of high merge probability (similar cofactors), as few checks on the output region can quickly find equivalence and merge points, and stop recursion. Forward processing is more similar to the BDD sweeping technique, as we start merging from primary inputs and propagate checks to the primary outputs. In this case as long as we find equivalent points, we can *learn* them, thus simplifying successive equivalence checks.

As a final remark, let us notice that the procedure is not far from testing stuck-at-faults on comparison gates over the product machine of the combined f_0 and f_1 cofactors. Anyway, as our main goal is finding merge points, we are more interested in finding redundancies, than good test patterns for faults, with implications in the comparison routines.

2.2. Synthesis based Optimizations

Once sub-circuit merging is complete, there is still a margin for size reduction, because we do not need individual representations for f_0 and f_1, but we must represent their disjunction $(f_0 \vee f_1)$. In principle, we look for any transformation $(f_0, f_1) \rightarrow (\widehat{f_0}, \widehat{f_1})$, such that $(\widehat{f_0} \vee \widehat{f_1}) = (f_0 \vee f_1)$, and the circuit size is reduced. It is worth distinguishing two categories of transformations:

1. Optimizations concentrating on the two cofactors (f_0 and f_1) with the goal of mutually simplify each other.
2. Optimization focusing on $(f_0 \vee f_1)$, with the purpose of minimizing, factorizing, rewriting, etc, the final resulting function.

We presently dedicate most of our effort to the former category. More specifically, we address possible simplifications/optimizations of the f_0 (f_1) cofactor whenever the f_1 (f_0) cofactor evaluates to 1. Let us take f_0 as reference cofactor, and let us use its onset as input don't care set to transform f_1. Let us consider the generic node $n_i(f_1)$ of f_1. Then our task is to guess a proper node transformation such that $\neg f_0 \rightarrow (n_i(\widehat{f_1}) = n_i(f_1))$, i.e., the transformed node is required to match the original one outside the don't care set (the offset of f_0). The above check can be easily achieved by a SAT solver trying to satisfy the function $(n_i(\widehat{f_1}) = n_i(f_1))$, within $\neg f_0$. Among the possible guesses for $n_i(\widehat{f_1})$, we select two straightforward choices: Constant value, i.e., redundancy, and merge, $n_i(\widehat{f_1}) = n_i(f_0)$, modulo complementation.

Taking into account observability don't cares for n_i adds a further optimization degree. We allow $(n_i(\widehat{f_1}) \neq n_i(f_1))$ within the input care set, provided that the difference is not observable on the output of $(f_0 \vee f_1)$. We implemented this extra step by an additional equivalence check, $\exists_x f = \exists_x \widehat{f}$, where $\exists_x \widehat{f}$ is the function after the transformation on n_i. The above problem can also be seen as a check for redundancy of the EXOR gate comparing n_i and $\widehat{n_i}$.

3. Traversal Routine

We modify standard breadth–first reachability in order to exploit circuit based quantification. Given an invariant property P we start reachability from its complement and we terminate as soon as no newly reached states are found (fix-point) or we intersect the initial state set, delivering a counter-example. In our implementation all state sets are represented and manipulated using AIGs instead of BDDs. Operations on AIGs, e.g., equivalence, are performed using a SAT engine. Pre-image adopts quantification by substitution (also called in-lining): $\exists_x g \wedge (x \leftrightarrow f) = g_{x=f}$. The applicability of this transformation relies on the fact that the formulas occurring in backward reachability often have a structure that matches the rule. In fact, in backward reachability, the transition relation is a conjunction of next state variables defined in terms of current state set variables, i.e., $\bigwedge_i (y_i = \delta_i(s, x))$.

4. Compatibility with Previous Methods

A possible drawback of our methodology is given by the fact that some variable quantifications can cause size explosion. To solve this problem, our methodology adopt "partial quantification", i.e., it accepts effective quantification and aborts the expensive ones (in term of size). As a result, the technique can be combined with other SAT-based (time intensive) approaches. We specifically address all solution SAT pre-image, as described in [2], where our approach could dramatically decrease the amount of decision (input) variables to be processed by SAT based pre-image. Similar considerations could be applied to BMC [1], as well as induction based Unbounded Model Checking [5]. Both these techniques can benefit from reducing the amount of primary input variables by quantification as a preprocessing of SAT procedures.

5. Conclusions

This paper attacks the Quantified Boolean Formula problems in the Unbounded Model Checking domain. Preliminary experimental results show the efficacy of the methodology on hard-to-verify circuits and properties.

References

[1] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu. Symbolic Model Checking using SAT procedures instead of BDDs. In *Proc. 36th Design Automat. Conf.*, pages 317–320, New Orleans, Louisiana, June 1999.

[2] M. K. Ganai, A. Gupta, and P. Ashar. Efficient SAT-based Unbounded Symbolic Model Checking Using Circuit Cofactoring. In *Proc. Int'l Conf. on Computer-Aided Design*, San Jose, California, Nov. 2004.

[3] A. Kuehlmann, M. K. Ganai, and V. Paruthi. Circuit-based Boolean Reasoning. In *Proc. Design Automat. Conf.*, Las Vegas, Nevada, June 2001.

[4] A. Kuehlmann and F. Krohm. Equivalence Checking Using Cuts and Heaps. In *Proc. 34th Design Automat. Conf.*, pages 263–268, Anaheim, California, June 1997.

[5] M. Sheeran, S. Singh, and G. Stålmarck. Checking Safety Properties Using Induction and SAT Solver. In W. A. Hunt and S. D. Johnson, editors, *Proc. Formal Methods in Computer-Aided Design*, volume 1954 of *LNCS*, pages 108–125. Springer-Verlag, Nov. 2000.