VaqTree: Efficient Vacuity Detection for Bounded Model Checking

Jocelyn Simmonds, Jessica Davies, and Arie Gurfinkel

University of Toronto,
Toronto, ON M5S 3G4, Canada.
{jsimmond,jdavies,arie}@cs.toronto.edu

1 Motivation and Introduction

Model-checking is a widely-used automated technique for verification of both hardware and software artifacts. A model-checker decides if a property to check is satisfied by a finite-state model of the artifact. If the property does not hold on the model, a counterexample, which can aid in debugging, is exhibited to the user. However, if the property does hold, no further information is given by traditional model-checkers. Thus, existing vacuity detection techniques (Beer et al. [1, 2]; Kupferman and Vardi [8, 9]; Purandare and Somenzi [10]; and Gheorghiu [6]) rely on property analysis and extra model-checking runs to determine if a property holds for the wrong reasons.

We focus on vacuity detection for SAT-based Bounded Model Checking (BMC). In the BMC setting, the property and the behavior of the model are encoded as a propositional theory using a model-checker. This theory is passed to a SAT solver to determine its satisfiability – it is unsatisfiable if and only if the property holds on the model, and a resolution proof showing how false (or the empty clause) can be derived is implicitly constructed. The set of clauses needed to derive false is known as the UNSAT core. Intuitively, this resolution proof provides an explanation of why the property holds on the model.

A naive method for detecting vacuity is to replace subformulas of the property with unconstrained boolean variables and model-check for each such substitution. If the property is still found to hold on the model with some substitution, the property is vacuous. This naive approach is expensive, since in the worst-case it requires as many runs of the model-checker as there are subformulas of the property. More efficient methods of detecting vacuity are necessary.

In this paper we discuss VaqTree, a tool that efficiently exploits the resolution proof produced by a successful run of BMC to detect vacuity of LTL properties. The goal of this tool is to detect a significant amount of vacuity when compared to the naive method, with faster runtimes. Our experiments, reported in [11], indicate that in practice, VaqTree detects 83% of the vacuous properties, and it is faster than the naive method in 72% of the cases studied.

Consider a simple two-process mutual exclusion program, like the one described in [4], where each process is in one of three states: *trying*, *critical*, *non-critical*. In a model where *state1* will never become *trying*, the property $\varphi = \Box((\text{state1} = \text{trying1}) \Rightarrow \Diamond(\text{state1} = \text{critical1}))$ holds vacuously. Naive detection requires two model-checking runs while VaqTree requires only one. To our knowledge, ours is the first vacuity detection tool for BMC.

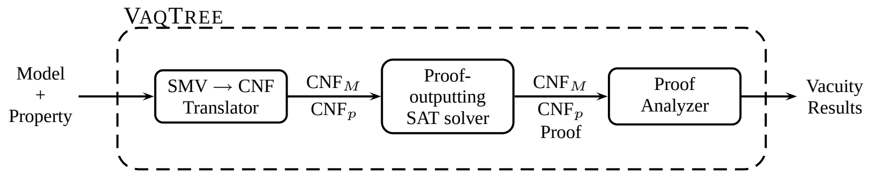

Fig. 1. VAQTREE components.

2 Design and Implementation

Th inputs to VAQTREE are a model (encoded using the specification language of the model-checker NUSMV [3]) and an LTL property. The tool generates the vacuity results for each variable present in the property. The component diagram for VAQTREE is shown in Figure 1. The three components interact sequentially:

SMV → CNF Translator receives as input a SMV file containing a model (M) and a property (p), which are translated into separate CNF files, CNF$_M$ and CNF$_p$, respectively. This translation is done using NUSMV, to which we added about 40 lines of code to get the translation as two files (lines added to the `bmcBmcNonInc.c` file in the `bmc` package).

Proof-outputing SAT solver generates the resolution proof for CNF$_M \cup$CNF$_p$. We use MINISAT [5] as it can explicitly generate resolution proofs when checking satisfiability. Instead of using MINISAT's binary proof format, we developed our own XML format, to facilitate future incorporation of other SAT solvers.

Proof Analyzer receives the CNF translations of the model and the property and the XML version of the corresponding resolution proof and produces the vacuity results. This is a new component, written in Java (around 3.5K lines of code).

VAQTREE can be executed via command line or controlled through its GUI. The GUI can generate an interactive graph representation of the resolution proof under analysis. Currently, VAQTREE can handle cases with 1.1 million clauses, i.e., roughly a model with 30 SMV boolean variable, expanded up to 19 steps.

The Proof Analyzer uses one of three vacuity detection methods (*irrelevance*, *local irrelevance* and *peripherality*) to produce the vacuity results. These methods are of increasing completeness and analyze the resolution proof produced by a successful BMC run. Irrelevance detects vacuity based on which variables appear in the UNSAT core. The intuition for irrelevance is that if a variable does not appear in the UNSAT core, it is not essential in proving that the property holds.

The intuition for local irrelevance is that variables that appear in the UNSAT core are not always central to the proof of why the property holds on the model. Thus, by analyzing where the variables occur in the UNSAT core, we are able to detect vacuity. The peripherality algorithm examines the *structure* of the resolution proof, identifying variables that are not necessary or central to its derivation of the empty clause as vacuous. This method is relatively complete and has time complexity linear in the size of the resolution proof. The user can choose which of these three methods is applied by VAQTREE. More information is available in [11]

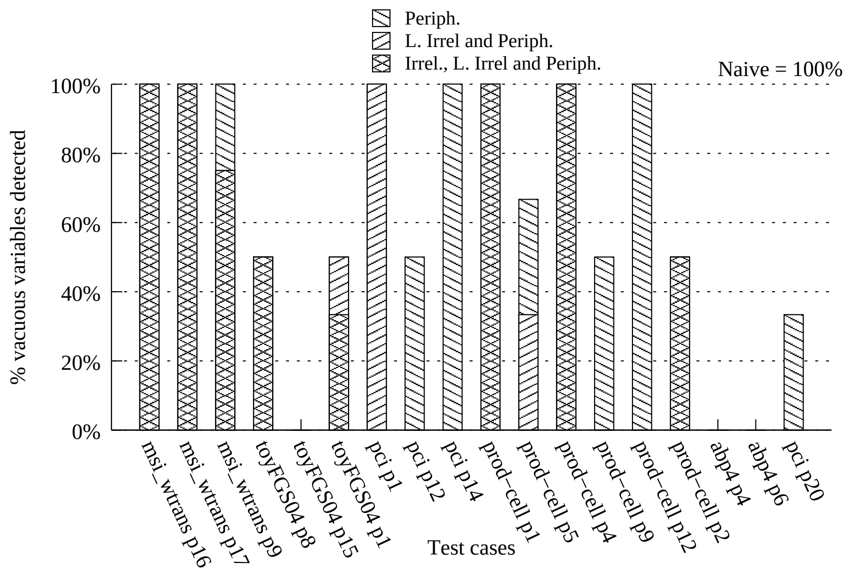

Fig. 2. Vacuous variable detection rates

3 Evaluation

To evaluate whether our algorithms were effective in practice, we ran VaqTree on 18 realistic model-property pairs that we knew to be vacuous. These test cases [11] are NuSMV distribution examples (15 properties, total) and a Flight Guidance System (**toyFGS04**) from [7] (three properties, referred to as **p1**, **p8**, **p15**). All tests were run on a Dell PowerEdge SC1425 (P4Xeon - 3.6GHz) running RedHat Linux 7.3 (2.4.x kernel) and 6 GB of RAM, where a maximum of 1.7 GB of RAM was available to each process.

Figures 2 and 3 show a summary of our evaluation. The percentages in Figure 2 are calculated with respect to the number of vacuous variables found by the naive method. For example, irrelevance and local irrelevance detected 75% of the vacuous variables present in property **msi_wtrans p9**, while peripherality managed to detect 100%. We do encounter cases where incompleteness prevents us from detecting vacuity (**abp4 p4**, **abp4 p6**, **toyFGS04 p15**); however, VaqTree can still detect 83% of the vacuous properties studied. Additionally, 28% of the properties are marked as vacuous only by peripherality (**pci p12**, **pci p14**, **pci p20**, **prod-cell p9**, **prod-cell p12**).

Execution times for irrelevance are not shown in Figure 3 as these are the same as for local irrelevance. We found that peripherality was faster than the naive method for 72% of the properties tested. It is slower in the remaining 5 cases (**prod-cell p2**, **prod-cell p12**, **abp4 p4**, **abp4 p6**, **pci p20**) due to technical issues with MiniSat: when it generates the resolution proof, it also outputs extraneous chains of resolution. Significant effort is needed to parse this output. We are currently investigating possible solutions, including an improved version of MiniSat.

Our evaluation indicates that in practice, examining the UNSAT core (i.e., using irrelevance and local irrelevance) will fail to detect many cases of vacuity, but peripherality detects around 60-80% of the vacuous properties, and does so an order of magnitude

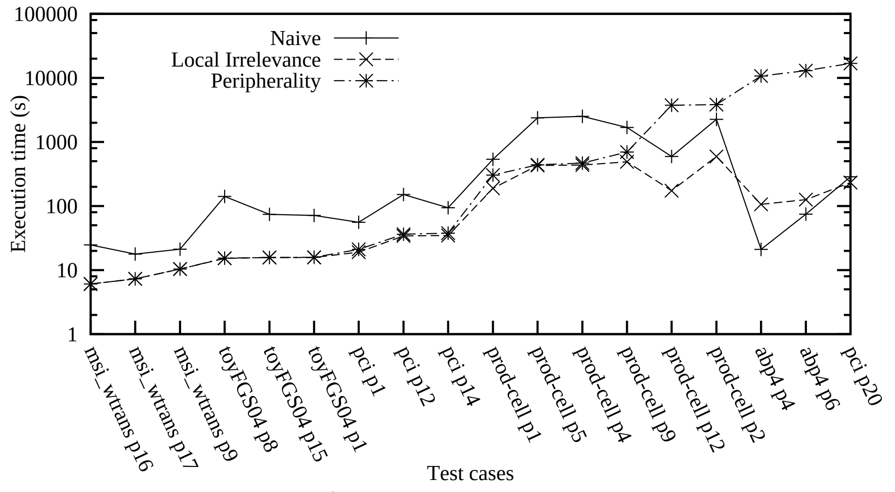

Fig. 3. Execution times

faster than the naive method in 72% of the cases tested. Thus, we have shown that examining the resolution proof produced by a successful BMC run is an efficient method for detecting vacuity of LTL properties. VaqTree is a fully-functional prototype, and will be released after some improvements to the user-interface are completed.

References

1. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in ACTL Formulas". In *Proceedings of CAV'97*, volume 1254 of *LNCS*, pages 279–290, 1997.
2. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in Temporal Model Checking". *FMSD*, 18(2):141–163, 2001.
3. A. Cimatti, E. Clarke, E. Giunchilia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. "NuSMV Version 2: An Open Source Tool for Symbolic Model Checking". In *Proceedings of CAV'02*, volume 2404 of *LNCS*, pages 359–364, 2002.
4. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*, chapter 8, pages 109–112. MIT Press, 1999.
5. N. Een and Sörensson. The MiniSat Page. `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/Main.html`, April 2006.
6. M. Gheorghiu, A. Gurfinkel, and M. Chechik. VaqUoT: A Tool for Vacuity Detection. CSRG Technical Report, University of Toronto, April 2005.
7. M. Heimdahl, S. Rayadurgam, W. Visser, G. Devaraj, and J. Gao. "Auto-generating Test Sequences Using Model Checkers: A Case Study". In *Proceedings of FATES'03*, volume 2931 of *LNCS*, pages 42–59, 2003.
8. O. Kupferman and M. Vardi. "Vacuity Detection in Temporal Model Checking". In *Proceedings of CHARME'99*, volume 1703 of *LNCS*, pages 82–96, 1999.
9. O. Kupferman and M. Vardi. Vacuity Detection in Temporal Model Checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 4(2):224–233, February 2003.
10. M. Purandare and F. Somenzi. "Vacuum Cleaning CTL Formulae". In *Proceedings of CAV'02*, volume 2404 of *LNCS*, pages 485–499, 2002.
11. J. Simmonds, J. Davies, A. Gurfinkel, and M. Chechik. "Exploiting Resolution Proofs for LTL Vacuity Detection". CSRG Technical Report 539, Department of Computer Science, University of Toronto, 2006.