Exploiting Resolution Proofs for LTL Vacuity Detection

Jocelyn Simmonds, Jessica Davies, Arie Gurfinkel and Marsha Chechik
University of Toronto,
Toronto, ON M5S 3G4, Canada.
Email: {jsimmond,jdavies,arie,chechik}@cs.toronto.edu

Abstract— **Temporal logic model-checking is a useful tool for verifying properties of hardware and software artifacts. If a desired property fails to hold of an artifact, a model-checker can provide a counterexample – an execution trace illustrating the cause of failure. However, when a property holds, no such information is given, making it difficult to identify cases when the property is trivially true, or when its success relies on unexamined assumptions about the artifact or its operating environment that may not be valid. One way of finding such information is to check *vacuity* of a formula, i.e., that its value depends on the validity of every subformula. While vacuity detection is effective, it is a relatively expensive technique requiring many additional model-checking runs.**

In this paper, we address the problem of efficient vacuity detection for Bounded Model Checking (BMC) of LTL properties. On a successful run, a bounded model-checker can produce a proof of unsatisfiability, which uses only a fragment of the problem specification; by examining this fragment, we propose to discover useful information about *how* the model fulfills the desired property. In particular, we define a characteristic of resolution proofs – *peripherality* – and prove that if a variable is a source of vacuity, then there exists a resolution proof in which this variable is peripheral. In our experiments on NuSMV distribution examples, peripherality is able to detect 83% of vacuous properties, with little overhead over the required BMC run.

I. Introduction

Model-checking [7] is a widely-used automated technique for verification of both hardware and software artifacts. Its purpose is to determine whether the artifact satisfies a criterion of correctness. A model-checker decides if the propositional logic formula describing the property to check is satisfied by a finite-state model of the hardware or software artifact. If the property does not hold on the model, an erroneous computation of the model (or counterexample) is exhibited to the user which can aid in debugging. If the property holds, only this decision is reported, so a successful model-checking run can be less useful than a failure [16]! A property can hold for unacceptable reasons, resting on incorrect assumptions that we have failed to check. For instance, if a property "every request is eventually acknowledged" is satisfied in an environment where no request is ever generated, then the constraints on the environment are unreasonable.

There has been, therefore, research on providing additional information from successful model-checking runs as well. A promising diagnostic which can be automatically detected is *vacuity* [13], [17], [1]. Informally speaking, a property is said to be vacuous if it has some subformula which is not relevant to its satisfaction, or if it is a tautology.

In this paper, we focus on vacuity detection for SAT-based Bounded Model Checking. In this context, a naive method for detecting vacuity is to replace subformulas of the temporal logic property with unconstrained boolean variables and run BMC 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.

SAT-based techniques automatically provide some information beyond a decision even when the property holds on the model, which can be exploited. Specifically, in BMC, the property and the behavior of the model are encoded in a propositional theory, which is satisfiable if and only if the formula does not hold. This propositional theory is passed to a SAT solver to determine its satisfiability. If the property holds, the SAT solver can produce a resolution proof that derives false from a subset of the clauses in the propositional theory called the UNSAT core. Intuitively, the resolution proof provides an explanation of why the property is not falsified by the model, and the UNSAT core determines the relevant parts of the model and property.

We present three vacuity detection methods (*irrelevance*, *local irrelevance* and *peripherality*) of increasing completeness based on analyzing the resolution proof produced by a successful BMC run. Irrelevance and local irrelevance detect vacuity based on which variables appear in the UNSAT core, and in what locations. However, the completeness of methods that only examine the UNSAT core is limited. 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 (see Section III-C for a definition) and has time complexity linear in the size of the resolution proof.

We compared the performance of implementations of the irrelevance, local irrelevance, peripherality and naive algorithms in order to determine if the differences in theoretical completeness would be encountered in practice. Our experiments range over SMV models and LTL properties taken from the NuSMV distribution, academic problems, and other sources. The results show that the trade-off between runtime and completeness favors the peripherality approach, since approximately 60-80% of vacuous properties are detected, faster than the naive method in the majority of cases.

The rest of the paper is organized as follows. Section II presents some required background, followed, in Section III, by the description of the irrelevance, local irrelevance, and peripherality algorithms. The experimental results are reported in Section IV. We discuss related work in Section V. The paper closes with conclusions and suggestions for future work in Section VI.

II. Background

In this section, we review bounded model-checking, resolution proofs and vacuity.

A. Bounded Model-Checking

Bounded model-checking (BMC) is a method for determining whether a temporal logic formula holds on a finite state system represented by a Kripke structure K up until a finite number of steps k. The property to be checked is expressed as a Linear Temporal Logic (LTL) formula φ. An instance of the BMC problem is denoted $BMC_k(K, \varphi)$. $BMC(K, \varphi)$ should be interpreted as $BMC_k(K, \varphi)$ with k equal to the system diameter [14].

Informally, a *Kripke structure* K has a finite set of states S, one of which is considered the initial state s_0. A transition relation R relates states to states, and each state is labeled by the set of propositional formulas that hold in that state. A *run* of K is a sequence of states starting with s_0 that obeys the transition relation R. Each run has an associated *trace* π, where π_i is simply the set of propositional formulas that label the i^{th} state in the run, and π^i is used to denote the suffix of the trace beginning at i.

Linear Temporal Logic formulas are built from propositional variables, the usual boolean operators ($\vee$, $\wedge$, $\neg$), and the temporal operators $\mathbf{G}$ ("always"), $\mathbf{F}$ ("eventually"), and $\mathbf{X}$ ("next"). Their semantics are defined on linear traces, such as those defined by runs of a Kripke structure. A trace π *satisfies* an LTL formula φ or a propositional formula p in the following cases:

$$
\begin{array}{lcllcl}
\pi \models p & \equiv & p \in \pi_0 & \pi \models \mathbf{X}\varphi & \equiv & \pi^1 \models \varphi \\
\pi \models \mathbf{F}\varphi & \equiv & \exists i \cdot \pi^i \models \varphi & \pi \models \mathbf{G}\varphi & \equiv & \forall i \cdot \pi^i \models \varphi
\end{array}
$$

A Kripke structure K *satisfies* an LTL formula φ if and only if, for all traces π of K, $\pi \models \varphi$. A BMC problem $BMC_k(K, \varphi)$ asks whether K satisfies φ for up to k steps, i.e. whether $K \models_k \varphi$. The *k-depth satisfaction relation* $\models_k$ is defined as follows:

$$
\begin{array}{lcl}
\pi \models_k p & \equiv & p \in \pi_0 \\
\pi \models_k \mathbf{G}\varphi & \equiv & \forall i \cdot 0 \leq i \leq k \Rightarrow \pi^i \models_{k-i} \varphi \\
\pi \models_k \mathbf{X}\varphi & \equiv & \pi^1 \models_{k-1} \varphi
\end{array}
$$

There is no definition of $\pi \models_k \mathbf{F}\varphi$ because there can be no length-k counterexample to $\mathbf{F}\varphi$: φ might be satisfied by π beyond k steps. Formulas that have k-depth semantics are called *safety* properties. However, BMC is still able to prove that $\mathbf{F}\varphi$ holds on K in the case that $\forall \pi\ \exists i \cdot 0 \leq i \leq k$ such that $\pi^i \models_{k-i} \varphi$.

To determine whether $K \models_k \varphi$, the problem is converted to a propositional formula which is satisfiable if and only if there exists a length-k counterexample to $K \models_k \varphi$. This propositional formula is then given to a SAT solver which decides its satisfiability. An overview of how the BMC problem is converted to a propositional formula is presented below; further details can be found in [4], [6], [5].

The propositional encoding represents the behavior of the Kripke structure up to k steps with a *path constraint* CL_K, and encodes every way that the property φ can fail to hold on K up to k steps in an *error constraint* CL_e. Therefore, if the theory $CL_K \cup CL_e$ is satisfiable, there is a path through the Kripke structure which obeys the transition relation and falsifies φ. The value of a variable at each time step is represented by introducing boolean variables v_i ($0 \leq i \leq k$) for each variable v of K.

The transition relation can be represented symbolically by a propositional formula over the variables V and primed variables V' (which represent the variables in the next state). For example, in the model in Figure 1(a), the transition relation is represented by the formula $R = (p \wedge \neg q \wedge \neg p' \wedge q') \vee (\neg p \wedge q \wedge \neg p' \wedge q')$. The path constraint is obtained by substituting the timed variables V_i for V in R, and replacing V' by the timed variables for the next step, V_{i+1}. This is repeated for each $0 \leq i < k$, and the resulting propositional formulas are conjoined along with a formula representing the initial state [4]. In Figure 1(a), if $k = 1$, $CL_K = (p_0 \wedge \neg q_0) \wedge ((p_0 \wedge \neg q_0 \wedge \neg p_1 \wedge q_1) \vee (\neg p_0 \wedge q_0 \wedge \neg p_1 \wedge q_1))$.

CL_e is encoded according to a recursive procedure which removes the temporal and logical operators from the property [4]. For example, the algorithm encodes $\varphi = \mathbf{G}p$, where p is a propositional variable, up to time k = 2 by the formula $\neg p_0 \vee \neg p_1 \vee \neg p_2$.

After the boolean formulas for the path and error constraints are calculated, they are converted to *Conjunctive Normal Form* (CNF) before being passed to a SAT solver. If the solver reports that $CL_K \cup CL_e$ is unsatisfiable, it means that there is no length-k counterexample to φ, otherwise, a satisfying assignment is returned.

When a search-based SAT solver processes an unsatisfiable theory, a resolution derivation of false (or the empty clause) is implicitly constructed. Therefore a SAT solver can be augmented to output a resolution proof tree showing how false was derived from $CL_K \cup CL_e$.

B. Resolution Proofs

A *resolution proof* Π is a directed acyclic graph whose nodes are labeled by propositional clauses. The *roots* of the proof are the nodes with no parents; otherwise, all nodes have exactly two parents. The nodes with no children are called the *leaves*. For example, the roots of resolution proof Π in Figure 1(b) are $Roots(\Pi) = \{(\neg r_0), (r_0 \vee p_0), (\neg p_0 \vee q_0), (\neg p_0 \vee \neg q_0), (p_0)\}$, and the leaf of Π is the empty clause, $Leaf(\Pi)$ = false.

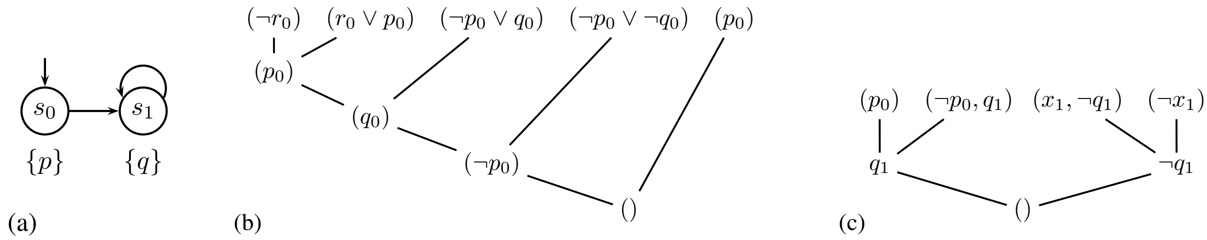

Fig. 1. (a) A Kripke structure; (b) A resolution proof for EXAMPLE 2; (c) A resolution proof for EXAMPLE 1.

Given a non-root node labeled by the clause c, and the labels of its parents, c_1 and c_2, c is called the *resolvent* since it has been produced by resolving c_1 and c_2 on some variable v. A resolution proof Π is a proof of the unsatisfiability of a set of clauses A if and only if all roots of Π belong to A, and one of the leaves of Π is the empty clause. For example, Figure 1(b) shows a resolution proof of the unsatisfiability of $Roots(\Pi)$. If a propositional theory in CNF is unsatisfiable, an *UNSAT core* is an unsatisfiable subset of its clauses.

Given two disjoint sets of clauses A and B, a variable v is said to be $local$ to A if and only if v appears in A but does not appear in B. If a variable v appears in both A and B, it is said to be $global$ to A and B. In Figure 1(b), if $Roots(\Pi) = A \cup B$ where $A = \{(\neg r_0), (r_0 \vee p_0), (\neg p_0 \vee q_0)\}$ and $B = \{(\neg p_0 \vee \neg q_0), (p_0)\}$, r_0 is local to A, and the rest of the variables are global.

C. Vacuity

In model-checking, temporal logic formulas are used to capture a specification, or the exact set of desired behaviors of some system. Thus, every part of the formula corresponds to some part of the specification, and a well-designed specification should not contain redundant parts. Intuitively, a formula φ is satisfied non-vacuously if every part of the formula is important. That is, even a slight change to φ affects its satisfiability. For example, the property $\varphi = \mathbf{X}(p \vee q)$ holds vacuously on the Kripke structure in Figure 1(a), because $p \vee q$ is satisfied in the next state regardless of the value of p since the only possible next state from s_0 is s_1.

We formalize this intuition as follows.

Definition 2.1: Let K be a Kripke structure, φ be a formula satisfied by K (i.e. $K \models \varphi$), and p be a subformula of φ. Then, φ is p-vacuous in K iff $\varphi[p \leftarrow x]$ is satisfied by K where x is a variable not occurring in K nor in φ.

If p is a variable, and φ is p-vacuous, we call p a *vacuous variable*. A property φ is vacuous if and only if there exists a propositional variable p of φ such that φ is p-vacuous. In other words, to check for vacuity it is sufficient to check the propositional variables.

We say that φ is *vacuous* if there exists a subformula p of φ such that φ is p-vacuous. This definition of vacuity is syntactic, and in this respect it is similar to the original definition of Beer et al. [3], [2]. The difference is that we make the substitution using a variable that is free in the model which was not allowed in [3], [2] since in that work, substitutions are made with formulas over the variables in the model. Therefore, Definition 2.1 is stronger, and is equivalent to the semantic definition of Armoni et al. [1], as shown by Gurfinkel and Chechik in [10].

D. Naive Vacuity Detection

Definition 2.1 immediately suggests a sound and complete algorithm for vacuity detection. That is, for each propositional variable p in φ, run BMC on $\varphi[p \leftarrow x]$, where x is a variable that does not appear in K and φ. If $K \models \varphi[p \leftarrow x]$ for some p, then φ is vacuous with respect to K. The drawback to this algorithm is that it may require as many model-checking runs as there are propositional variables in φ. Since this algorithm is expensive and obtained directly from our definition of vacuity, we call it *naive*.

III. EXPLOITING RESOLUTION PROOFS

In this section, we introduce three methods for vacuity detection based on examining the UNSAT core and the resolution proof produced by BMC. We first define a criterion to evaluate the completeness of our methods. Next, we describe the irrelevance and local irrelevance methods, which take an UNSAT core as input. Since these methods do not meet our criterion, we introduce an algorithm based on analyzing the structure of the resolution proof. This algorithm, called peripherality, achieves our goal.

A. A Criterion for Vacuity Detection Methods

We are interested in vacuity detection methods for BMC that require little overhead beyond the time already taken by the SAT solver to prove that the property holds on the model. We therefore limit ourselves to considering the output of one model-checking run on *BMC*(K, φ), and hence one resolution proof of its unsatisfiability. In general, there may be many ways to derive the empty clause from different subsets of *BMC*(K, φ). Any method that only examines one of these derivations is inherently incomplete, in the sense that a property may be p-vacuous but there is no way of determining this based on a given resolution proof. For example, consider a model that is composed of two completely disjoint sub-models, running in parallel, i.e., $K = K_1 \parallel K_2$. Suppose that K_1 satisfies $\mathbf{G}p$, K_2 satisfies $\mathbf{G}q$, and that both do so non-vacuously. Then the property $\varphi = \mathbf{G}p \vee \mathbf{G}q$ holds on K p-vacuously and q-vacuously. However, one of the possible resolution proofs showing that φ holds proves that $\mathbf{G}p$ holds non-vacuously on K_1. Therefore, it is impossible to determine that p is actually vacuous in φ from this proof.

Although we are limited to incomplete methods in our desire for efficiency, we would still like some assurance that our method is as complete as possible. We say that a method is *relatively complete* if it is able to detect existing vacuity given that all possible resolution proofs are considered. Stated equivalently, this criterion is as follows.

Definition 3.1: Let K be a model, φ be an LTL formula, and *BMC*(K, φ) be a CNF encoding of a BMC problem for $K \models \varphi$. Assume that φ is p-vacuous. A method of detecting vacuity is *relatively complete* if there exists a resolution proof Π of the unsatisfiability of *BMC*(K, φ) such that this method can determine that p is vacuous using Π.

In what follows, we evaluate our vacuity detection methods according to their relative completeness.

B. Examining UNSAT Cores

Given a resolution proof that *BMC*(K, φ) is unsatisfiable, we can sometimes cheaply determine that the similar theory *BMC*(K, $\varphi[p \leftarrow x]$) is unsatisfiable, and therefore that the property is p-vacuous. In this section, we consider how to determine that *BMC*(K, $\varphi[p \leftarrow x]$) is unsatisfiable given that *BMC*(K, φ) is unsatisfiable, using only an UNSAT core.

The irrelevance method. Intuitively, any variable that does not appear in the UNSAT core of *BMC*(K, φ) does not contribute to the reason why φ holds on K, so it can be considered *irrelevant*.

Definition 3.2: Let K be a model, φ be an LTL formula, and *BMC*(K, φ) be a CNF encoding of a BMC problem for $K \models \varphi$. p is *irrelevant* in *BMC*(K, φ) iff for each time instance i, p_i does not appear in the UNSAT core.

The intuition that irrelevant variables are vacuous is confirmed by the following theorem.

Theorem 3.1: If p is irrelevant in *BMC*(K, φ), then φ is p-vacuous.

Proof: Let *BMC*(K, φ) $= CL_K \cup CL_e$ and U be the UNSAT core returned by the SAT solver for *BMC*(K, φ). Assume that p is irrelevant in *BMC*(K, φ). So U does not contain any p_i by Definition 3.2. Therefore, $U \subseteq CL_K \cup CL_e$ implies $U \subseteq CL_K \cup CL_e[p \leftarrow x]$. U is also an UNSAT core of *BMC*(K, $\varphi[p \leftarrow x]$) so $\varphi[p \leftarrow x]$ holds on K. Thus, φ is p-vacuous. □

However, a variable can appear in the UNSAT core and still be vacuous, as demonstrated by the following example.

EXAMPLE 1. Consider a Kripke structure K with variables p and q given by the constraints

$$Init \;=\; p \wedge q, \qquad R \;=\; p \Rightarrow q'$$

which mean that the initial state is labeled by $\{p, q\}$ and the transition relation is expressed by the propositional formula $p \Rightarrow q'$ over unprimed and primed variables. Let $\varphi = X(p \vee q)$ be the property to check. φ is p-vacuous since it is satisfied simply because q is true in any successor of the initial state. The CNF encoding of the one-step BMC problem is $CL_K = \{(p_0 \wedge q_0), (p_0 \Rightarrow q_1)\} = \{(p_0), (q_0), (\neg p_0, q_1)\}$, $CL_e = \{(\neg p_1), (p_1, \neg q_1)\}$. In this case, the *unique* minimal UNSAT core contains all of the clauses of the problem except for (q_0). Thus, all p_i appear in the UNSAT core, and p cannot be determined vacuous using irrelevance. □

This example shows that even if we are to look at every UNSAT core of a BMC problem, irrelevance is unable to detect existing vacuity, so this method is not relatively complete.

The local irrelevance method. Variables which do not appear in the resolution proof are vacuous, but also vacuous variables sometimes appear in the UNSAT core. Intuitively, these variables are not the central reason why φ holds on K. For example, the clauses of CL_K may resolve against each other, representing some simplification and unification of parts of the model, before resolutions with CL_e clauses are performed. If a variable is only resolved upon among the CL_K clauses or among the CL_e clauses, it is potentially vacuous. By looking at the UNSAT core, it is possible to anticipate if a variable will not be involved in resolutions between CL_K and CL_e using the following definition.

Definition 3.3: Let K be a model, φ be an LTL formula, and *BMC*(K, φ) be a CNF encoding of a BMC problem for $K \models \varphi$. Then, p is *locally irrelevant* in *BMC*(K, φ) iff for each time instance i, either p_i does not appear in the UNSAT core or p_i is local to $CL_e \cap$(UNSAT core) or to $CL_K \cap$(UNSAT core).

In Example 1, p is locally irrelevant since p_1 only occurs in the clauses of U taken from CL_e, while p_0 only appears in U within CL_K clauses. Moreover, the UNSAT core of the original problem can be converted to an UNSAT core of the new theory, thus proving that p is vacuous. Specifically, $U = \{(p_0), (\neg p_0, q_1), (\neg p_1), (p_1, \neg q_1)\}$ is the UNSAT core of the original problem, so substituting x for p in the clauses of U that came from CL_e gives $U' = \{(p_0), (\neg p_0, q_1), (\neg x_1), (x_1, \neg q_1)\}$. This is a subset of *BMC*(K, $\varphi[p \leftarrow x]$) $= \{(p_0), (q_0), (\neg p_0, q_1), (\neg x_1), (x_1, \neg q_1)\}$, so it is a candidate for the new UNSAT core. The substitution may have prevented the resolutions necessary to derive the empty clause. However, Figure 1(c) shows a proof that U' is also unsatisfiable. In this case it was possible to substitute x_i for p_i in the clauses coming from CL_e in the original UNSAT core and create an UNSAT core for *BMC*(K, $\varphi[p \leftarrow x]$). In fact, this observation applies to all cases of local irrelevance as shown by the following theorem.

Theorem 3.2: If p is locally irrelevant in *BMC*(K, φ), then φ is p-vacuous.

If a variable p is not locally irrelevant in an UNSAT core, the formula can still be p-vacuous. The following example shows that local irrelevance is not relatively complete.

EXAMPLE 2. Consider a Kripke structure with atomic propositions r, p and q whose initial state is given by the constraint: $Init = \neg r \wedge p \wedge q$. The formula $\varphi = \neg p \vee q$ is p-vacuous in the initial state. Let us assume that the zero-step BMC problem is encoded in CNF as follows:

$$\begin{array}{rcl} CL_K & = & (\neg r_0)(r_0 \vee p_0)(\neg p_0 \vee q_0) \\ CL_e & = & (p_0)(\neg p_0 \vee \neg q_0) \end{array}$$

There are several resolution proofs that can establish unsatisfiability of $CL_K \cup CL_e$; one such proof is shown in Figure 1(b). In none of the proofs is p locally irrelevant with respect to CL_e and CL_K. □

The problem with using local irrelevance is that it is impossible to tell if a variable is going to be used in a resolution joining CL_K and CL_e clauses based on the UNSAT core alone.

C. Extending the analysis to resolution proofs

In Section III-B, two vacuity detection methods based on examining the variables in the UNSAT core were found to fall short of relative completeness. In this section, the analysis is extended to the resolution proof's structure, and relative completeness is achieved by the resulting peripherality algorithm.

Peripherality. The limitations of detecting vacuity based only on the UNSAT core were demonstrated in Example 2. By examining the resolution proof in Figure 1(b), we see that although p_0 appears both in CL_K clauses and CL_e clauses, it is always resolved "locally". That is, if we resolve two clauses $c_1 = (..., p_i, ...)$ and $c_2 = (..., \neg p_i, ...)$, p_i and $\neg p_i$ must have been preserved from their original source in some set of root clauses. If all the originating root clauses belong to CL_K or all belong to CL_e, then p_i is being resolved on locally. In this case, we can replace p_i in either set of clauses without affecting their unsatisfiability. In Figure 1(b), for example, p_0 can be replaced in CL_e by a fresh variable. This intuition can be formalized as follows.

Given a resolution proof Π, a variable l, and a clause c, we denote by $S(l, c)$ the set of all root clauses that have contributed the variable l to c. $S(l, c)$ is defined recursively as follows:

$$S(l,c) = \begin{cases} \emptyset & \text{if } l \notin c \\ c & \text{if } c \in Roots(\Pi) \wedge l \in c \\ S(l,c_1) \cup S(l,c_2) & \text{if } c_1 \text{ and } c_2 \text{ are parents} \\ & \quad \text{of } c \ \wedge \ l \in c \end{cases}$$

Intuitively, a root clause c_r is an element of $S(l, c)$ if it contains a variable l and there exists a path from c_r to c that does not contain a resolution on l. This allows us to define *peripherality* of variables, which captures the conditions when a global variable may not be central to the reason why φ holds on K.

Definition 3.4: Let $C = A \cup B$ be an unsatisfiable set of clauses, and Π be a resolution proof establishing unsatisfiability of C. Then a variable l is *peripheral* with respect to A and B iff for every resolution on l between clauses c_1 and c_2, $S(l, c_1) \cup S(l, c_2) \subseteq A$ or $S(l, c_1) \cup S(l, c_2) \subseteq B$.

If p is locally irrelevant in a resolution proof Π, then p is also peripheral in that proof. Within the BMC setting, we have the following definition:

Definition 3.5: Let K be a model, φ be an LTL formula, *BMC*(K, φ) be a CNF encoding of a BMC problem for $K \models \varphi$, and Π be a proof of unsatisfiability of *BMC*(K, φ). p is *peripheral* in φ iff for each time instance i, p_i is peripheral in Π with respect to CL_e and CL_K.

Theorem 3.3: If a variable p is peripheral in φ, then φ is p-vacuous.

In Figure 1(b), although p is not locally irrelevant in φ, it is peripheral, and therefore φ is p-vacuous by Theorem 3.3. This also demonstrates that peripherality is a strictly stronger notion than local irrelevance.

The next result shows that this is the strongest result that we can hope to establish. That is, peripherality is a relatively complete method for vacuity detection.

Theorem 3.4: If φ is p-vacuous, then there exists an unsatisfiable BMC problem with a resolution proof Π such that p is peripheral in φ.

Proof: Assume that φ is p-vacuous. Then, the BMC problem *BMC*(K, $\varphi[p \leftarrow x]$) $= CL_K \cup CL_e[p \leftarrow x]$ is unsatisfiable, and there exists a resolution proof Π establishing this. We must show that this proof can be transformed to a proof of unsatisfiability of *BMC*(K, φ) $= CL_K \cup CL_e$ in which each p_i is peripheral with respect to CL_K and CL_e.

If Π does not contain a clause that has both p_i and x_i for some i, then Π' obtained from Π by replacing each occurrence of x_i by p_i is (a) a well-formed resolution proof, and (b) $Roots(\Pi') \subseteq CL_K \cup CL_e$. That is, Π' is a resolution proof establishing that *BMC*(K, φ) is unsatisfiable.

We now show how such a proof can be constructed from an arbitrary proof Π of unsatisfiability of *BMC*(K, $\varphi[p \leftarrow x]$). Let $U_K = Roots(\Pi) \cap CL_K$, and $CL_{e'} = CL_e[p \leftarrow x]$. Then, if p_i occurs in any clause of U_K, it is local to U_K. Let L be the set of all local variables of U_K, $C = \exists L \cdot U_K$ be a formula resulting from existentially eliminating these local variables, and $CNF(C)$ be the CNF encoding of C. For any i, p_i does not appear in C. Furthermore, the set of clauses $CNF(C) \cup CL_{e'}$ is unsatisfiable. Thus, there exists a resolution proof Π' establishing this, such that $Roots(\Pi') \subseteq CL_{e'} \cup CNF(C)$. Finally, since $U_K \Rightarrow C$, for each clause $c \in C$ there exists a resolution proof Π_c such that *Leaf*(Π_c) $= c$ and $Roots(\Pi_c) \subseteq U_K$. By combining the proofs $\{\Pi_c \mid c \in CNF(C)\}$ with Π', we obtain a proof of unsatisfiability of $U_K \cup CL_{e'}$ that does not contain a clause with variables x_i and p_i. □

The peripherality algorithm. Peripherality of a variable can be detected by traversing the resolution proof from the roots to the leaf, passing labels that keep track of the source of the variable in each clause. If Π is a resolution proof whose root clauses are divided into two disjoint sets, $A \cup B$, then the labeling function L is defined recursively as shown in Figure 2, where c is used to represent a clause. This labeling function defines an algorithm for detecting peripherality.

A variable p is peripheral iff the label of the empty clause is not 'AB', that is, $L(\mathsf{false}) \neq$ 'AB'. Thus, to detect whether a formula φ is vacuous in an atomic proposition p, we execute the algorithm in Figure 2 with $A = CL_K$, $B = CL_e$, and keep track of all occurrences of p_i. If every occurrence is peripheral, then φ is p-vacuous. It is possible to simultaneously keep track of the labels for all variables p in φ so that only one pass through the resolution proof is needed.

Theorem 3.5: Given a resolution proof Π of the unsatisfiability of *BMC*(K, φ), determining which variables of φ are peripheral can be done in time linear in the size of Π.

In this section, we defined three methods of detecting vacuity based on examining the UNSAT core and resolution

1) if $c \in Roots(\Pi)$ then

$$L(c) = \begin{cases} \text{`}\emptyset\text{'} & \text{if } p \notin c \\ \text{`A'} & \text{if } p \in c \wedge c \in A \\ \text{`B'} & \text{if } p \in c \wedge c \in B \end{cases}$$

2) if c is a clause resulting from a resolution between c_1 and c_2 on variable v, i.e. $c = \exists v \cdot c_1 \wedge c_2$ then

   a) if $v \neq p$ then

$$L(c) = \begin{cases} \text{`}\emptyset\text{'} & \text{if } L(c_1) = L(c_2) = \text{`}\emptyset\text{'} \\ \text{`A'} & \text{if } \exists i, j \cdot L(c_i) = \text{`A'} \wedge L(c_j) \subseteq \{\text{`A'}, \text{`}\emptyset\text{'}\} \\ \text{`B'} & \text{if } \exists i, j \cdot L(c_i) = \text{`B'} \wedge L(c_j) \subseteq \{\text{`B'}, \text{`}\emptyset\text{'}\} \\ \text{`AB'} & \text{otherwise} \end{cases}$$

   b) and if $v = p$ then

$$L(c) = \begin{cases} \text{`}\emptyset\text{'} & \text{if } L(c_1) = L(c_2) \\ \text{`AB'} & \text{otherwise} \end{cases}$$

Fig. 2. Labeling function for the peripherality algorithm.

proof produced by BMC. As shown by Theorems 3.4 and 3.5, peripherality is our desired relatively complete and efficient vacuity detection method.

IV. Experiments

In this section, we validate theoretical results obtained in Section III on realistic examples.

We implemented the irrelevance, local irrelevance and peripherality algorithms in Java 1.5. The CNF theories were generated using NuSMV v2.3.1 [5], modified to output CL_K and CL_e separately. MiniSat-p v1.14 [9] was used to produce the resolution proofs, where we modified its proof-checker to generate a non-binary version of the output.

We initially tested all four algorithms on 23 properties of student-created models – small elevator controller models which, we expected, would contain many vacuous properties but not be computationally demanding. The results of analyzing these are shown in Table I. The column "k" indicates the system diameter, as determined by NuSMV. BMC was run with this k. The column "# p_i" indicates the number of p_is checked, and is equal to k times the number of variables in the formula. The remaining four columns show the number of vacuous variables detected by each method. Numbers in ***italics*** in the "Per." column indicate cases when 50% or more of the vacuous variables of the property could be detected using peripherality. For example, peripherality detects all vacuous variables in property **F1** (with 56 timed occurrences), while irrelevance and local irrelevance do not detect any vacuity in this case. For these academic models, peripherality was able to detect 59% of the vacuous properties, and detected strictly more vacuous variables than the local irrelevance method for almost a third of the properties. While these tables do not list runtimes – they were negligible – the effectiveness of peripherality was encouraging, and so we ran our experiments on more complex models.

These consisted of 4 models (**abp4**, **msi_wtrans**, **pci**, and production-cell, shortened to **prod-cell**) taken from the NuSMV distribution (15 properties total), and **toyFGS04** from [11] (three properties, referred to as **p1**, **p8**, **p15**). The convention *model name px* is used for property *px* of model *model name*, e.g. **toyFGS04 p8**. All tests were run on a Dell PowerEdge SC1425 (4 x 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.

Table II shows detailed results for the NuSMV distribution and **toyFGS04** examples, including runtimes. Columns "k" and "# p_i" are defined as in Table I. For the naive method, the number of vacuous variables detected ("Vac. Vars."), the number of runs required ("Runs"), and the total time for these runs ("time (s)") are reported in the next three columns. These execution times include the time taken by NuSMV, MiniSat and our vacuity detection algorithms. For the resolution proofs, the time is split into the overall and the one taken just by the model-checker. As in Table I, ***italics*** in the "Per." column indicate properties where 50% or more of the vacuous variables were detected. For example, property **pci p14** had two vacuous variables, as identified after four runs of the model-checker, taking combined time of 94.13 seconds. For this property, neither irrelevance nor local irrelevance found vacuity, and the analysis took 34.64 seconds. Peripherality, on the other hand, found all vacuity, taking 37.71 seconds, where 24.07 were taken by the initial model-checking run.

We do encounter cases where the incompleteness of the peripherality method prevents us from detecting vacuity. For example, the naive method detects vacuity in the two **abp4** properties and in **toyFGS04 p15**, which are missed by peripherality. However, peripherality only missed three vacuous properties out of a total of 18, namely **abp4 p4**, **abp4 p6** and **toyFGS04 p15**.

The runtimes obtained show that the peripherality algorithm is faster than the naive method for 72% of the properties, and in half of these cases, by as much as an order of magnitude. The peripherality algorithm took longer than the naive method by an order of magnitude in only four out of 18 cases, namely **abp4 p4**, **abp4 p6**, **pci p20**, and **prod-cell p12**. These anomalies are due to problems with the output format of MiniSat: the output contains a lot of extraneous chains of resolution which need to be further pruned to obtain the actual proof. The extent of this problem is seen in Table II (columns "Input Clauses" and "Clauses Needed" list the number of clauses generated by Minisat, and those needed for the actual proof; numbers in ***italics*** in the "Clauses Needed" column indicate properties where less than 45% of the output generated by MiniSat formed part of the actual resolution proof: 72% of our test cases show this behavior). For example, in **abp4**'s properties, irrelevance and local irrelevance already require more time than the naive method. This negatively affects our scalability, and prevented us from running any analysis on 15 additional properties from **prod-cell** and **abp10**, since our implementation could not handle the amount of output produced by MiniSat.

Two other factors contributing to the higher runtimes of **pci p20**, **prod-cell p2, p12**, and **abp4 p4 and p6** are the high number of p_is being tracked, and the large number of clauses

TABLE I

STATISTICS FOR VACUITY DETECTION EXPERIMENTS USING ACADEMIC EXAMPLES

Property	k	# p_i	# Vacuous Propositions found			
			Naive	Irrel.	L. Irrel.	Per.
A1	12	52	3	0	0	1
B1	6	35	0	0	0	0
C1	8	36	2	0	0	*1*
D1	12	65	4	3	3	*3*
D2	12	65	4	0	0	*2*
E1	11	48	1	0	0	0
E2	11	48	2	0	0	0
F1	7	56	6	0	0	*6*
F2	7	24	1	0	0	0
G1	6	21	1	0	0	0
H1	2	9	3	2	2	*2*
H2	2	9	2	0	0	0
H3	2	6	1	1	1	*1*
I1	6	42	3	0	0	1
I2	6	28	1	0	0	*1*
I3	6	14	2	0	0	0
I4	6	14	1	1	1	*1*
J1	7	24	1	0	0	0
K1	16	34	1	0	0	0
L1	15	112	4	0	1	1
L2	15	32	2	0	0	0
L3	15	192	9	9	9	*9*
M1	13	42	2	0	0	*1*

TABLE II

STATISTICS FOR VACUITY DETECTION EXPERIMENTS ON NUSMV DISTRIBUTION AND OTHER EXAMPLES

Property	k	# p_i	Naive Method			Using UNSAT cores			Using Resolution Proofs			
			Vac. Vars.	Runs	time (s)	Irrel.	Local Irrel.	time (s)	Per.	time (s) Total/MC	Input Clauses	Clauses Needed
abp4 p4	19	120	2	3	21.03	0	0	105.93	0	10730.54/13.37	1,523,529	949,973
abp4 p6	19	160	2	4	74.56	0	0	125.50	0	12988.66/10.78	1,311,085	909,394
msi_wtrans p9	20	126	4	4	21.23	3	3	10.40	*4*	10.41/5.81	89,893	*66*
msi_wtrans p16	20	105	1	2	24.73	1	1	6.07	*1*	6.07/3.86	53	53
msi_wtrans p17	20	105	1	2	17.85	1	1	7.29	*1*	7.29/5.05	53	53
pci p1	13	70	1	4	55.84	0	1	18.99	*1*	21.09/11.08	142,434	*7,225*
pci p12	13	56	2	6	150.93	0	0	34.20	*1*	36.22/24.07	179,511	*17,925*
pci p14	13	140	2	4	94.31	0	0	34.64	*2*	37.71/25.35	160,424	*13,750*
pci p20	13	224	6	7	286.85	0	0	232.95	2	16858.15/43.01	1,790,802	1,109,359
prod-cell p1	81	328	2	4	535.29	2	2	188.53	*2*	304.15/136.67	724,849	*168,342*
prod-cell p2	81	738	2	6	2263.97	1	1	595.14	*1*	3831.60/393.93	775,219	*303,743*
prod-cell p4	81	1312	4	7	2503.34	4	4	435.44	*4*	467.36/366.89	617,277	*93,565*
prod-cell p5	81	738	3	6	2381.29	0	1	428.88	*2*	439.99/268.14	741,553	*88,193*
prod-cell p9	81	984	2	5	1690.92	0	0	491.07	*1*	698.21/426.11	791,267	*148,623*
prod-cell p12	81	164	2	4	592.22	0	0	173.51	*2*	3749.85/85.58	367,230	*154,056*
toyFGS04 p1	9	60	6	6	71.48	2	3	15.76	*3*	15.80/11.95	60,179	*293*
toyFGS04 p8	9	120	12	12	141.45	6	6	15.31	*6*	15.34/11.68	60,344	*292*
toyFGS04 p15	9	60	1	6	73.86	0	0	15.67	0	15.75/12.11	60,254	*318*

in the proof. For example, **pci p20** has 1.1 million clauses and 224 variables being tracked. However, we believe that an optimized version of peripherality would easily beat the naive method even on these cases.

Table III summarizes our results on the NuSMV models and **toyFGS04**. The percentage of vacuous variables found and the percentage of vacuous properties detected is given for the irrelevance ("Irrel."), local irrelevance ("L. Irrel.") and peripherality ("Per.") methods. These results clearly show that the peripherality algorithm detects significantly more vacuity than either of the UNSAT core-based methods, and a substantial amount of the existing vacuity as reported by the naive method.

TABLE III

VACUITY SUMMARY

	Irrel.	L. Irrel.	Per.
% Vac. Vars. found	36	42	60
% Vac. Properties found	44	56	83

V. RELATED WORK

The first attempt to automate vacuity detection was made by Beer et al. [2], [3]. They established a syntactic definition of vacuity based on replacing a subformula by all other formulas. Although the method of Beer et al. only requires one additional symbolic model-checking run, the property must be a member of a restricted subset of CTL, w-ACTL. Kupferman and Vardi [13] introduced the naive method, extending vacuity detection to all of CTL* (which includes LTL). In [17], Purandare and Somenzi showed how to reduce the number of model-checking runs by analyzing the parse tree of the CTL property.

Armoni et al. [1] generalized the syntactic definition of vacuity by introducing universal quantification, i.e. $\forall x \cdot \varphi[\psi \leftarrow x]$. Based on the domain of x, three notions of vacuity are obtained, the most robust of which is *trace* vacuity. In [10], Gurfinkel and Chechik extend Armoni's definition of vacuity to CTL*, thus uniformly capturing CTL and LTL. Our definition of vacuity (Section II-C) corresponds to that of [10].

Namjoshi [15] defines a somewhat unrelated notion of vacuity (it coincides with the definition of [1], [10] for a subset of LTL), based on examining the invariants of a proof derived from a successful model checking run. According to Namjoshi, a property should only be considered vacuous if every proof of why it holds on the model is vacuous. Our methods also efficiently examine proofs derived from model-checking runs, but are able to detect vacuity as defined by [2], [3], [1], [10].

In the context of declarative modeling, Shlyakhter et al. [18] address the problem of identifying overconstrained models. In declarative models, there is no explicit transition relation; instead, transitions are expressed with constraints [12], [19]. An overconstraint occurs when the model satisfies a safety property because all violations of the formula have been accidentally ruled out by the declared constraints. In order to detect such overconstraints, Shlyakhter et al. introduce the idea of *core extraction*: declarative models are reduced to SAT instances, from which an UNSAT core can be extracted if the property holds. If a constraint's clauses do not appear in the UNSAT core, the constraint is called *irrelevant*, and is a source of overconstraint. This is equivalent to Definition 3.2, but our experimental results show that irrelevance is not complete enough to be useful for vacuity detection.

VI. Conclusion and Future Work

In this paper, we have shown how to exploit the UNSAT core and resolution proof produced by a successful run of BMC to detect vacuity of LTL properties. We introduced three vacuity detection methods that can be applied with little overhead after one model-checking run. Two of these methods, irrelevance and local irrelevance, exploit the UNSAT core, and the third, peripherality, is based on analyzing the resolution proof. Our experiments indicate 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, often an order of magnitude faster than the naive method. 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.

Although we were able to examine cases with 1.1 million clauses, we would like to further improve the scalability of our implementation. The primary factor limiting our scalability is the bloated resolution proof output by MiniSat. We are currently investigating possible solutions, including an improved version of MiniSat.

Since we found that peripherality detects much more vacuity than irrelevance, detecting overconstraint in declarative models may also benefit from methods that analyze the structure of the resolution proof. In future work, we intend to investigate how a notion equivalent to peripherality can be defined in the declarative setting, e.g., extending results of [18].

In this paper, we restricted ourselves to using the results of only one BMC run, and to methods with linear time complexity in the size of the resolution proof or better. However, it is possible that the most optimal trade-off between speed and effectiveness of vacuity detection algorithms lies in the domain of multiple resolution proofs, were we can find the minimal UNSAT core or reduce the resolution proof using interpolation [8].

References

[1] R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. "Enhanced Vacuity Detection in Linear Temporal Logic". In *Proceedings of CAV'03*, volume 2725, pages 368–380, July 2003.

[2] 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.

[3] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in Temporal Model Checking". *FMSD*, 18(2):141–163, 2001.

[4] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. "Symbolic Model Checking without BDDs". In *Proceedings of TACAS'99*, volume 1579 of *LNCS*, 1999.

[5] A. Cimatti, E.M. 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.

[6] A. Cimatti, M. Pistore, M. Roveri, and R. Sebastiani. "Improving the Encoding of LTL Model Checking into SAT". In *Proceedings of VMCAI'02*, pages 196–207, 2002.

[7] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[8] W. Craig. "Linear Reasoning. A New Form of the Herbrand-Gentzen Theorem". *JSL*, 22:250–268, 1957.

[9] N. Een and Sörensson. The MiniSat Page. `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/Main.html`, April 2006.

[10] A. Gurfinkel and M. Chechik. "Extending Extended Vacuity". In *Proceedings of FMCAD'04*, volume 3312 of *LNCS*, pages 306–321, 2004.

[11] M. Heimdahl, S. Rayadurgam, W. Visser, G. Devaraj, and J. Gao. "Autogenerating Test Sequences Using Model Checkers: A Case Study". In *Proceedings of FATES'03*, volume 2931 of *LNCS*, pages 42–59, 2003.

[12] Daniel Jackson. "Alloy: a Lightweight Object Modelling Notation". *ACM TOSEM*, 11(2):256–290, 2002.

[13] O. Kupferman and M. Vardi. "Vacuity Detection in Temporal Model Checking". In *Proceedings of CHARME'99*, volume 1703 of *LNCS*, pages 82–96, 1999.

[14] Kenneth L. McMillan. Exploiting SAT Solvers in Unbounded Model Checking. `http://embedded.eecs.berkeley.edu/Alumni/kenmcmil/cav03tut.ppt`, April 2003.

[15] K. Namjoshi. "An Efficiently Checkable, Proof-Based Formulation of Vacuity in Model Checking". In *Proceedings of CAV'04*, volume 3114 of *LNCS*, pages 57–69, 2004.

[16] D. Peled and L. Zuck. "From Model Checking to a Temporal Proof". In *Proceedings of SPIN'01*, volume 2057 of *LNCS*, pages 1–14, 2001.

[17] M. Purandare and F. Somenzi. "Vacuum Cleaning CTL Formulae". In *Proceedings of CAV'02*, volume 2404 of *LNCS*, pages 485–499, 2002.

[18] I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. "Debugging Overconstrained Declarative Models Using Unsatisfiable Cores". In *Proceedings of ASE'03*, pages 94–105, October 2003.

[19] J. M. Spivey. *The Z Notation: a Reference Manual*. Prentice Hall International (UK) Ltd., 1992.