Exploiting Resolution Proofs to Speed Up LTL Vacuity Detection for BMC

Jocelyn Simmonds[1], Jessica Davies[1], Arie Gurfinkel[1,2], Marsha Chechik[1]

[1]Department of Computer Science, University of Toronto
[2]Software Engineering Institute at Carnegie Mellon University
{jsimmond, jdavies, arie, chechik}@cs.toronto.edu

Abstract. When model-checking reports that a property holds on a model, *vacuity detection* increases user confidence in this result by checking that the property is satisfied in the intended way. While vacuity detection is effective, it is a relatively expensive technique requiring many additional model-checking runs. We address the problem of efficient vacuity detection for Bounded Model Checking (BMC) of LTL properties, presenting three partial vacuity detection methods based on the efficient analysis of the resolution proof produced by a successful BMC run. 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. Our vacuity detection tool, ***VaqTree***, uses these methods to detect vacuous variables, thereby decreasing the total number of model-checking runs required to detect all sources of vacuity.

1 Introduction

Model-checking [7] is a widely-used automated technique for verification of both hardware and software artifacts that checks whether a temporal logic property is satisfied by a finite-state model of the artifact. If the model does not satisfy the property, a counterexample, which can aid in debugging, is produced.

If the model *does* satisfy the property, no information about why it does so is provided by the model-checker alone. A positive answer without any additional information can be misleading, since a property may be satisfied in a way that was not intended. For instance, a property "every request is eventually acknowledged" is satisfied in an environment that never generates requests.

Vacuity detection [2, 14, 18, 1] is an automatic sanity check that can be applied after a positive model-checking run in order to gain confidence that the model and the property capture the desired behaviours. Informally, a property is said to be vacuous if it has a subformula which is not relevant to its satisfaction, or if the property itself is a tautology. Conversely, a property is satisfied non-vacuously if every part of the formula is important – even a slight change to the formula affects its satisfaction.

In this paper, we focus on vacuity detection for SAT-based Bounded Model Checking (BMC). Given a BMC problem with a particular bound, we wish to determine if the property holds vacuously on the model up to this bound. 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 with some substitution still holds on the model, the property is vacuous. This naive approach is expensive, since in the worst case it requires as many model-checking runs as there are subformulas in the property. Our goal is to reduce the number of model-checking runs required to detect vacuity. We do this by detecting some vacuity through novel and inexpensive techniques reported in this paper, and complete the method by running the naive algorithm on the remaining atomic subformulas. The key to our technique is that a run of a SAT-based BMC can automatically provide useful information (a resolution proof) beyond a decision whether the property holds on the model; we exploit such proofs for partial vacuity detection.

In SAT-based BMC, the property and the behavior of the model are encoded in a propositional theory, such that the theory is satisfiable if and only if the formula does not hold. When the property does hold, a DPLL-based SAT solver can produce a resolution proof that derives false from a subset of the clauses in the theory called the UNSAT core. Intuitively, the resolution proof provides an explanation why the property is not falsified by the model, and the UNSAT core determines the relevant parts of the model and the property.

In this paper, we develop three methods of increasing precision (*irrelevance*, *local irrelevance* and *peripherality*) to analyze the resolution proof to achieve partial vacuity detection. These algorithms are used by our vacuity detection tool, ***VaqTree***, in order to reduce the number of model-checking runs required to find all sources of vacuity, thus reducing execution times. Irrelevance and local irrelevance detect vacuity based on which variables appear in the UNSAT core, and in which locations. However, as these methods only examine the UNSAT core, their precision is limited. The peripherality algorithm examines the *structure* of the resolution proof, identifying as vacuous those variables that are not necessary or central to the derivation of false. This method is as precise as can be achieved through analyzing a single resolution proof, and its running time is linear in the size of the resolution proof and the number of variables in the property. Our experience shows that local irrelevance is the ideal candidate for replacing naive vacuity.

The remainder of the paper is organized as follows. Section 2 presents some required background, followed, in Section 3 by our definition of vacuity, the naive algorithm for LTL vacuity detection using BMC, and an overview of work in the vacuity detection field. Section 4 presents the three algorithms that detect vacuity by analyzing a resolution proof. Our experimental results are presented in Section 5. We conclude with a summary, additional related work, and suggestions for future work in Section 6.

2 Background

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

2.1 Bounded Model-Checking

Bounded model-checking (BMC) [4] is a method for determining whether a linear temporal logic (LTL) formula φ holds on a finite state system represented by a Kripke structure K up to a finite number of steps. An instance of a BMC problem, denoted by $BMC_k(K, \varphi)$, is whether $K \models_k \varphi$, where $\models_k$ is the *k-depth satisfaction relation*. An informal description of LTL formulas, Kripke structures and BMC is given in Appendix A.1, and detailed definitions can be found in [7, 4].

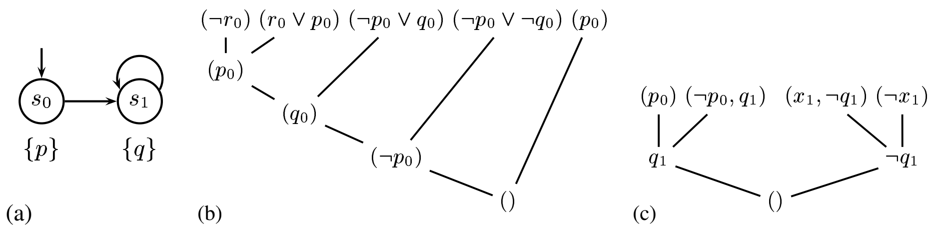

Fig. 1. (a) A Kripke structure; (b) A resolution proof for EXAMPLE 2; (c) A resolution proof for EXAMPLE 1.

To determine whether $K \models_k \varphi$, the problem is converted to a propositional formula Φ (see [4, 6, 5]) which is satisfiable if and only if there exists a length-k counterexample to $K \models_k \varphi$. Φ is then given to a SAT solver which decides its satisfiability. The propositional encoding represents the behavior of K up to k steps with a *path constraint* CL_K, and encodes all counterexamples to φ of length k in an *error constraint* CL_e. Therefore, if the theory $CL_K \cup CL_e$ is satisfiable, there is a path through K which obeys the transition relation and falsifies φ. The value of each variable v of K at each time step is represented by introducing new boolean variables v_i ($0 \leq i \leq k$), called *timed variables*.

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], e.g., the algorithm encodes $\varphi = \mathbf{G}p$, where p is a propositional variable, expanded up to $k = 2$, 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 DPLL-based SAT solver processes an unsatisfiable theory, a resolution derivation of false (or the empty clause) is implicitly constructed [9, 22]. This resolution proof is used to verify that false can indeed be derived from $CL_K \cup CL_e$ [23].

2.2 Resolution Proofs

Resolution is an inference rule that is applied to propositional clauses to produce logical consequences. A *clause* is a disjunction of boolean variables and their negations. For example, $(v_1 \vee \neg v_2 \vee v_5)$ is a clause stating that at least one of $v_1, \neg v_2$ or v_5 must be true. The resolution rule takes two clauses, where one contains a variable v and the other – its negation $\neg v$, and produces a clause containing the union of the two clauses minus v and $\neg v$. For example, resolving $(v_1 \vee \neg v_2 \vee v_5)$ and $(v_2 \vee v_6)$ produces the *resolvent* $(v_1 \vee v_5 \vee v_6)$.

A *resolution proof* Π is a directed acyclic graph whose nodes are labeled by propositional clauses. Π represents a tree of resolutions between the clauses labeling its nodes. Its *roots* 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,i.e., $Leaf(\Pi) = \mathsf{false}$. Given a non-root node labeled by the clause c, and the labels of its parents, c_1 and c_2, c is 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 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, and v is said to be *global* if it appears in both 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 $B = \{(\neg p_0 \vee \neg q_0), (p_0)\}$, then r_0 is local to A, and the rest are global.

3 Defining Vacuity

This paper uses the following definition of vacuity.

Definition 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 or in φ.*

Similarily, it is possible to define vacuity in the BMC setting.

Definition 2. *Let K be a Kripke structure, φ be a formula such that $K \models_k \varphi$, and p be a subformula of φ. Then, φ is k-step p-vacuous iff $K \models_k \varphi[p \leftarrow x]$, where x is a variable not occurring in K or in φ.*

If φ is k-step p-vacuous, we call p a *k-step vacuous variable*. A property φ is *k-step vacuous* if and only if φ contains a k-step vacuous variable. In other words, to check whether the property is vacuous, it is sufficient just to check its propositional variables. Therefore, our techniques aim to find the k-step vacuous variables of φ. The qualifier "k-step" is omitted in the remainder of the paper but should be understood implicitly in the BMC context.

Definition 1 suggests a sound and complete algorithm for vacuity detection: 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 $K \models_k \varphi[p \leftarrow x]$ for some p, then φ is k-step vacuous. We refer to this algorithm as *naive*. Its drawback is that it may require as many model-checking runs as there are propositional variables in φ.

We now review some of the alternative definitions of vacuity and their algorithms. The first attempt to formulate and automate vacuity detection is due to Beer et al. [2]. They consider a property φ to be vacuous if φ contains a subformula ψ such that replacing ψ by any other formula does not affect the satisfaction of φ. Applying this definition directly would require an infinite number of subformula replacements, precluding a practical implementation. However, Beer et al. show that to detect vacuity w.r.t. a *single* occurrence of a subformula ψ in w-ACTL, it is sufficient to consider replacing ψ only with true and false. This was later extended to CTL* by Kupferman and Vardi [14]. Purandare and Somenzi [18] showed how to speed up the discovery of a vacuous subformula by analyzing the parse tree of a CTL property.

Armoni et al. [1] generalized the above 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 being *trace* vacuity. Gurfinkel and Chechik [11] extended Armoni's definition of vacuity to CTL*, thus uniformly capturing CTL and LTL. Armoni et al. also analyzed the syntactic structure of the property in order to avoid checking the operands of subformulas that are known to be vacuous. Such optimizations complement our techniques, which focus on detecting vacuous *atomic* subformulas.

Namjoshi [17] defines a somewhat different notion of vacuity (it coincides with the definition of [1, 11] for a subset of LTL), based on 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 exhibits vacuity. Our methods also efficiently examine proofs derived from model-checking runs, but are able to detect vacuity as defined by [2, 3, 1, 11].

Our definition of vacuity is syntactic, and in this respect, it is similar to the original definition of Beer et al. [2]. However, Definition 1 is stronger, and is equivalent to the semantic definition of Armoni et al. [1], as shown by Gurfinkel and Chechik [11].

4 Exploiting Resolution Proofs

In Section 3, we discussed the existence of a sound and complete vacuity detection algorithm for BMC, which requires as many model-checking runs as there are propositional variables in the property being checked. We propose a new vacuity detection strategy: first detect partial vacuity using inexpensive techniques and then complete the analysis using extra model-checking runs. Since we are interested in replacing expensive model-checking runs by inexpensive partial vacuity detection methods, we limit ourselves to considering the output of the original model-checking run on $BMC_k(K, \varphi)$. This run provides us with a single resolution proof to analyze, but in general, there may be many ways to derive the empty clause from different subsets of $BMC_k(K, \varphi)$. 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. Thus, it is impossible to determine that φ is vacuous in p from this proof. Any method based on examining only one resolution proof cannot prove the absence of vacuity, since another resolution proof, showing the property to be vacuous, might exist.

In this section, we introduce three algorithms of increasing completeness for partial vacuity detection, based on examining the UNSAT core (irrelevance and local irrelevance) and the resolution proof produced by BMC (peripherality).

4.1 Examining UNSAT cores

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

Irrelevance. Intuitively, any variable that does not appear in the UNSAT core does not contribute to the reason why φ holds on K, so it can be considered *irrelevant*.

Definition 3. *Let K be a model, φ be an LTL formula, and $BMC_k(K, \varphi)$ be a CNF encoding of a BMC problem for $K \models_k \varphi$. p is* irrelevant *in $BMC_k(K, \varphi)$ iff for each time instance i, p_i does not appear in the UNSAT core.*

If a variable is irrelevant, it is also vacuous, as shown by the following theorem. The proofs of this and other theorems are given in the Appendix.

Theorem 1. *If p is irrelevant in $BMC_k(K, \varphi)$, then φ is k-step p-vacuous.*

Definition 3 provides an algorithm to detect some vacuous variables. 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 $Init = p \wedge q$, $R = p \Rightarrow q'$, 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. $\Box$

This example shows that even if we are to look at every UNSAT core of a BMC problem, irrelevance is still unable to detect existing vacuity.

Local Irrelevance. Variables which do not appear in the UNSAT core are vacuous. The converse is not true: vacuous variables may also 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 resolved upon using only the CL_K clauses or only the CL_e clauses, it is potentially vacuous. By looking at the UNSAT core, it is possible to anticipate whether a variable will not be involved in resolutions between CL_K and CL_e using the following definition.

Definition 4. *Let K be a model, φ be an LTL formula, and $BMC_k(K, \varphi)$ be a CNF encoding of a BMC problem for $K \models_k \varphi$. Then, p is* locally irrelevant *in $BMC_k(K, \varphi)$ 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 $BMC_1(K, \varphi[p \leftarrow x]) = \{(p_0), (q_0), (\neg p_0, q_1), (\neg x_1), (x_1, \neg q_1)\}$, 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 $BMC_k(K, \varphi[p \leftarrow x])$. In fact, this observation applies to all cases of local irrelevance by Theorem 2. Therefore, Definition 4 specifies an algorithm to detect some vacuous variables.

Theorem 2. *If p is locally irrelevant in $BMC_k(K, \varphi)$, then φ is k-step p-vacuous.*

Unfortunately, if a variable p is not locally irrelevant in an UNSAT core, the formula can still be p-vacuous, as shown by the following example.

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:

$$CL_K = (\neg r_0)(r_0 \vee p_0)(\neg p_0 \vee q_0) \qquad CL_e = (p_0)(\neg p_0 \vee \neg q_0)$$

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 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.

4.2 Peripherality

In Section 4.1, two vacuity detection methods based on examining the variables in the UNSAT core were found to fall short of completeness. It was seen that even if every possible resolution proof could be analyzed, irrelevance and local irrelevance still might fail to detect existing vacuity. In this section, we extend the analysis to the resolution proof's structure. The resulting peripherality algorithm is superior, since it guarantees vacuity will be found if all possible resolution proofs are considered.

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 in 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

$L(c)$ **: clause** c**, variable** $p \rightarrow \{$**'$\emptyset$', 'A', 'B', 'AB'**$\}$

– **if** $c \in Roots(\Pi)$ **then**

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

– **else if** c is a clause resulting from resolving c_1 and c_2 on variable v, i.e., $c = \exists v \cdot c_1 \wedge c_2$, **then**

- **if** $v \neq p$, **then**

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

- **else 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.

without affecting their unsatisfiability. For example, in Figure 1(b), p_0 can be replaced in CL_e by a fresh variable x_0. This intuition is formalized below.

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:

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

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 5. *Let A and B be disjoint sets of clauses such that $C = A \cup B$ is unsatisfiable, 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 $S(l, c_1) \cup S(l, c_2) \subseteq B$.*

Within the BMC setting, we have the following definition:

Definition 6. *Let K be a model, φ be an LTL formula, $BMC_k(K, \varphi)$ be a CNF encoding of a BMC problem for $K \models_k \varphi$, and Π be a proof of unsatisfiability of $BMC_k(K, \varphi)$. 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.*

If a variable is peripheral, it is vacuous by Theorem 3.

Theorem 3. *Let Π be a proof of unsatisfiability of BMC$_k(K, \varphi)$. If a variable p of φ is peripheral in Π, then φ is k-step p-vacuous.*

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

Theorem 4 shows that this is the strongest result that we can hope to establish.

Theorem 4. *If φ is k-step p-vacuous, then there exists an unsatisfiable BMC problem with a resolution proof Π such that p is peripheral in Π.*

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 CNF variable v is peripheral iff the label of the empty clause is not 'AB'. Thus, to detect whether a formula φ is p-vacuous, we need to check that all CNF variables p_i corresponding to p (see Section 2) are peripheral. This can be done by applying the labeling function described in Figure 2 with $A = CL_K$, and $B = CL_e$ for each p_i. It is also possible to simultaneously keep track of the labels for all CNF variables so that only a single pass through the resolution proof is needed. The time complexity of the peripherality algorithm is linear in the size of the resolution proof.

Theorem 5. *For a resolution proof Π that BMC$_k(K, \varphi)$ is unsatisfiable, 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 the resolution proof produced by BMC. Our evaluation of these algorithms with respect to precision and execution times can be found in Section 5.

5 Practical Experience

The techniques reported in this paper have been implemented in a tool called ***VaqTree***. The inputs to ***VaqTree*** are a model (encoded using the language of NuSMV [5]) and an LTL property. The tool returns the vacuity status of each variable in the property. Vacuity detection in ***VaqTree*** proceeds in two phases: a "partial pass" that applies either irrelevance, local irrelevance or peripherality, and a "model-checking pass" that completes the analysis using additional model-checking runs. Tool details can be found in Appendix A.3.

To evaluate the effectiveness and performance of ***VaqTree***, we compared it against naive detection on a benchmark suite of 5 models: **abp4**, **msi_wtrans**, **pci**, and **prodcell** from the NuSMV distribution (107 properties) and **toyFGS04** from [12] (14 properties). The results of these experiments are summarized in Table 1. For each test case consisting of a model-property pair, the bound k was chosen arbitrarily so that the property held. 99 of the properties in our benchmark exhibit vacuity, and 22 do not. Tests were performed on a Linux machine with a 2.8GHz P4 CPU, and 1GB of RAM, with

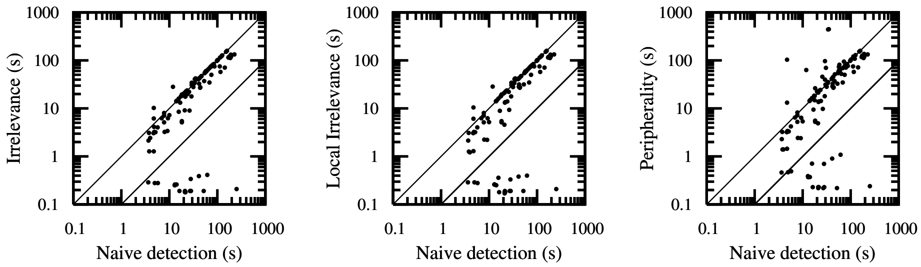

Fig. 3. Comparison of execution times. Where applicable, all times include times for both the partial and model-checking passes.

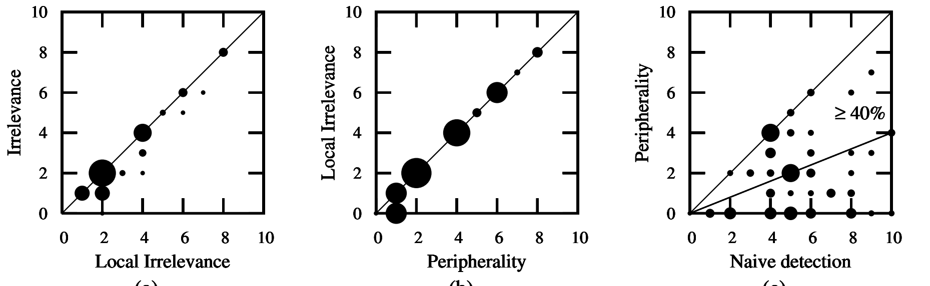

Fig. 4. Comparison of the number of vacuous variables detected by partial pass of the analysis. Larger points represent more test cases than the smaller points.

up to 700MB of RAM available to each process. Currently, ***VaqTree*** is limited to proofs with up to 2.5 million resolutions. In our benchmark suite, this corresponds to a test case from the asynchronous **abp4** model (roughly 30 boolean variables, with $k = 19$).

Scatter plots in Figure 3 compare the execution times of ***VaqTree*** (parametrized with irrelevance, local irrelevance, and peripherality), with naive detection. Execution times for naive detection include CNF theory generation and satisfiability testing for each variable of the property. Execution times for ***VaqTree*** include the time for the partial pass and the subsequent model-checking pass. Each point in the plot represents a single test case. The X axis represents the time (in seconds) taken by naive detection. The Y axis represents the time (in seconds) taken by ***VaqTree*** when parameterized by each of our methods. Points below the diagonal indicate where ***VaqTree*** was faster than naive detection; points near the diagonal indicate cases where the partial pass found a small percentage of the vacuous variables.

Figure 3 shows that ***VaqTree*** with irrelevance finds the fewest vacuous variables among our partial methods, as expected from Section 4. Although Figures 3(b) and (c) look similar, the numbers (available in Table 1) show that local irrelevance is faster than peripherality in 96% of the cases. This is consistent with the additional work peripherality must perform to analyze the proof tree. A detailed comparison of local irrelevance and naive detection shows that ***VaqTree*** with local irrelevance was faster or comparable to naive detection in 95% of the test cases. ***VaqTree*** with local irrelevance was faster than naive detection in 70 (58%) of the test cases, out of which 30 cases were twice as fast, and 20 cases were faster by an order of magnitude. In the remaining 51 cases, local irrelevance was at most 3% slower in 86% of these cases.

There are 10 cases where ***VaqTree*** with peripherality took much longer than naive detection. All of these cases are from the **abp4** model, and while they have the largest resolution proofs of the benchmark suite (between 300,000 and 2M clauses), other 300,000-clause test cases did not yield poor performance. We conjecture that the poor performance is due to a low clause/variable ratio [19] which favours naive detection in cases where vacuity is not present. Intuitively, a low ratio indicates that the SAT instance is underconstrained, and so a solution (if it exists) can be found quickly. On the other hand, finding a proof of *unsatisfiability* in a model with few constraints can be more difficult. Naive detection on a non-vacuous property requires solving satisfiable SAT instances, since replacing variables falsifies the property. However, peripherality on a non-vacuous property requires time linear in the size of the resolution proof obtained from the original model-checking run. If all of these SAT instances have a low clause/variable ratio, naive detection can be much faster than peripherality. This situation was only observed on the **abp4** model, with clause/variable ratio of 1.5-1.8 – significantly lower than any other test case without vacuity.

We now turn to measuring the effectiveness of our methods, using the number of vacuous variables found during the partial pass as a metric (see the scatter plots in Figure 4). This number indicates how many additional model-checking runs are needed to complete vacuity detection. Since our partial methods can be ordered by increasing completeness, Figure 4(a) compares irrelevance and local irrelevance, Figure 4(b) – local irrelevance and peripherality, and Figure 4(c) – peripherality and naive detection. Each point in the plot represents a set of test cases – a larger point means a larger set. The axes show the number of vacuous variables detected by each method. Points below the diagonal indicate where the X axis method detects more vacuous variables than the Y axis method. The plots show that local irrelevance is clearly more effective than irrelevance. Surprisingly, peripherality performed exactly as local irrelevance in all but 5 cases, so its increased runtimes are mostly unjustified. Figure 4(c) shows that our techniques are effective when compared with naive detection: peripherality reduced the number of extra model-checking runs by 40% in 54 out of 99 cases that exhibited vacuity.

In summary, we observed that local irrelevance performs best out of our proposed partial methods, finding most vacuity in the least amount of time. We found ***VaqTree*** with local irrelevance to be faster (or at most 3% slower) than naive detection on 95% of the benchmark suite. Thus, we believe that it is the ideal candidate for replacing naive detection. However, we plan to enhance our methods by developing a heuristic based on the clause/variable ratio and proof size that indicates when naive detection should be applied instead. Finally, ***VaqTree*** outputs the vacuity results for each timed variable p_i as a byproduct of its partial pass. This informa tion gives an explanation of non-vacuity, indicating which time steps have been important for deciding whether a given variable was vacuous, thus facilitating debugging.

6 Summary and Related Work

In this paper, we showed how to exploit the UNSAT core and resolution proof produced by a successful run of BMC for vacuity detection. We introduced three vacuity detection methods that can be applied with little overhead after one model-checking run in order to quickly identify vacuous variables and reduce the number of additional model-checking runs required. Two of these methods, irrelevance and local irrelevance, exploit the UNSAT core, and the third, peripherality, is based on analyzing the resolution proof. We built a tool ***VaqTree***, which is based on these methods, and showed that it is effective for speeding up vacuity detection.

Related work on vacuity detection has been described in Section 3. Additionally, our work is related to research in declarative modeling. In particular, our use of the UNSAT core to detect vacuity was inspired by [20], which addresses the problem of identifying overconstraint in declarative models. While similar in spirit to vacuity detection in model checking, declarative models have no explicit transition relation; instead, transitions are expressed with constraints [13, 21]. 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, [20] introduces 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 our Definition 3.

As we have seen in our experiments, local irrelevance and peripherality can detect more vacuous variables than irrelevance. Therefore, detecting overconstraint in declarative models may also benefit from methods that analyze the structure of the resolution proof. In future work, we propose to investigate how a notion equivalent to peripherality can be defined in the declarative setting.

Another goal of future work is to increase the power of resolution proof-based vacuity detection methods. In this paper, we restricted ourselves to using 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, where we can find the minimal UNSAT core [10] or reduce the resolution proof using interpolation [8].

McMillan [16] uses interpolation to prove that a particular bound is sufficient to imply the unbounded satisfaction of a BMC problem. We intend to combine our techniques with this algorithm in order to prove that bounded vacuity for the correct k implies that the property also holds vacuously in the unbounded case.

Another means of speeding vacuity detection for BMC is to iteratively check the k-step vacuity of each variable starting with $k = 0$. If a variable is proven non-vacuous at a particular step k, then it can be omitted from subsequent checks of higher k. This is because $K \not\models_{k_1} \varphi[p \leftarrow x]$ implies $K \not\models_{k_2} \varphi[p \leftarrow x]$ for all $k_2 > k_1$. This method is orthogonal to our techniques, and the vacuity detection at each step could be carried out by ***VaqTree***.

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. R. Gershman, M. Koifman, and O. Strichman. “Deriving Small Unsatisfiable Cores with Dominators”. In *Proceedings of CAV’06*, volume 4144 of *LNCS*, pages 109–122, 2006.
11. A. Gurfinkel and M. Chechik. “Extending Extended Vacuity”. In *Proceedings of FMCAD’04*, volume 3312 of *LNCS*, pages 306–321, 2004.
12. 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.
13. D. Jackson. “Alloy: a Lightweight Object Modelling Notation”. *ACM TOSEM*, 11(2):256–290, 2002.
14. O. Kupferman and M. Vardi. “Vacuity Detection in Temporal Model Checking”. In *Proceedings of CHARME’99*, volume 1703 of *LNCS*, pages 82–96, 1999.
15. K. McMillan. ”Exploiting SAT Solvers in Unbounded Model Checking”. In *CAV’03 Tutorial*, April 2003.
16. K. McMillan. “Interpolation and SAT-Based Model Checking”. In *Proceedings of CAV’03*, volume 2725 of *LNCS*, pages 1–13, July 2003.
17. 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.
18. M. Purandare and F. Somenzi. “Vacuum Cleaning CTL Formulae”. In *Proceedings of CAV’02*, volume 2404 of *LNCS*, pages 485–499, 2002.
19. B. Selman, D. Mitchell, and H. Levesque. “Generating Hard Satisfiability Problems”. *Artificial Intelligence*, 81(1-2):17–29, 1996.
20. 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.
21. J. M. Spivey. *The Z Notation: a Reference Manual*. Prentice Hall, 1992.
22. L. Zhang and Z. Fu. Boolean Satisfiability Research Group at Princeton. `http://www.princeton.edu/~chaff/`, September 2006.
23. L. Zhang and S. Malik. “Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications”. In *Proceedings of DATE’03*, pages 10880–10885, 2003.

A Appendix

A.1 Informal Introduction to BMC

In this section, we give an overview of Kripke structures, LTL formulas and BMC.

A *Kripke structure* K has a finite set of states S, one of which is considered to be 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 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.

LTL formulas are built from propositional variables, the usual boolean operators ($\vee$, $\wedge$, $\neg$), and the temporal operators **G** ("always"), **F** ("eventually"), **U** ("until"), and **X** ("next"). Their semantics are defined on linear traces, such as those produced by runs of a Kripke structure. $\pi \models \varphi$ means that the trace π *satisfies* the LTL formula φ. For example, $\pi \models \mathbf{F}\varphi$ if and only if there exists some i such that φ holds on π^i. The *satisfaction relation* $\models$ is defined inductively in a similar way for all operators and propositional variables in LTL [15].

A Kripke structure K *satisfies* an LTL formula φ if and only if $\pi \models \varphi$ for all traces π of K. A BMC problem *BMC*$_k$(K, φ) determines 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 inductively; for example, $\pi \models_k \mathbf{G}\varphi$ if and only if $\pi^i \models_{k-1} \varphi$ for all $i \leq k$.

A.2 Proofs of Theorems

In this section, we give proofs of our theorems.

Theorem 1 *If p is irrelevant in BMC*$_k$*(K,* φ*), then* φ *is k-step p-vacuous.*

Proof: Let *BMC*$_k$(K, φ) $= CL_K \cup CL_e$ and U be the UNSAT core returned by the SAT solver for *BMC*$_k$(K, φ). Assume that p is irrelevant in *BMC*$_k$(K, φ). So U does not contain any p_i by Definition 3. 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*$_k$(K, $\varphi[p \leftarrow x]$) so $\varphi[p \leftarrow x]$ holds on K. Thus, φ is p-vacuous. □

Theorem 2 *If p is locally irrelevant in BMC*$_k$*(K,* φ*), then* φ *is k-step p-vacuous.*

Proof: Let *BMC*$_k$(K, φ) $= CL_K \cup CL_e$ and U be the UNSAT core returned by the SAT solver for *BMC*$_k$(K, φ). Assume that p is locally irrelevant in *BMC*$_k$(K, φ). So for all p_i, either p_i does not appear in U, or p_i is local to $CL_e \cap U = U_e$ or to $CL_K \cap U = U_K$ by Definition 4. Let $U_{e'}$ be U_e with each occurence of p_i replaced by x_i. Since each p_i that has been replaced is local to U_e, and $U_K \cup U_e = U$ is unsatisfiable, then $U_K \cup U_{e'}$ is also unsatisfiable. Since $U_{e'} \subseteq CL_e[p \leftarrow x]$, the set of clauses $CL_K \cup CL_e[p \leftarrow x]$ is unsatisfiable as well. Therefore, $K \models_k \varphi[p \leftarrow x]$ holds, so φ is p-vacuous. □

Theorem 3 *Let* Π *be a proof of unsatisfiability of BMC*$_k$*(K,* φ*). If a variable p of* φ *is peripheral in* Π*, then* φ *is k-step p-vacuous.*

Proof: Let *BMC*$_k$(*K*, φ) $= CL_K \cup CL_e$ and U be the UNSAT core returned by the SAT solver for *BMC*$_k$(*K*, φ). Assume that p is peripheral in *BMC*$_k$(*K*, φ). Let $U_{e'}$ be the result of replacing each p_i with x_i in $CL_e \cap U$. Then $(CL_K \cap U) \cup U_{e'}$ is still unsatisfiable, since every resolution on x_i must be local to $CL_e \cap U$, and every resolution on p_i must be local to $CL_K \cap U$ by the peripherality of p_i. Since $U_{e'} \subseteq CL_e[p \leftarrow x]$, $CL_K \cup CL_e[p \leftarrow x]$ is unsatisfiable as well. Therefore, $K \models_k \varphi[p \leftarrow x]$, and φ is p-vacuous. □

Theorem 4 *If φ is k-step p-vacuous, then there exists an unsatisfiable BMC problem with a resolution proof Π such that p is peripheral in Π.*

Proof: Assume that φ is p-vacuous. Then, the BMC problem *BMC*$_k$(*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*$_k$(*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 for Π' obtained from Π by replacing each occurrence of x_i by p_i, (a) Π' is 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*$_k$(*K*, φ) is unsatisfiable.

We now show how such a proof can be constructed from an arbitrary proof Π of unsatisfiability of *BMC*$_k$(*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*$(C)\}$ and 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. □

Theorem 5 *For a resolution proof Π that BMC$_k$(K, φ) is unsatisfiable, determining which variables of φ are peripheral can be done in time linear in the size of Π.*

Proof: To determine which variables of φ are peripheral, the labeling algorithm defined in Figure 2 must be run once on Π for each variable p in φ. The labeling algorithm passes through the resolution proof, visiting each node once in breadth-first order. A constant amount of work must be done at each node in order to determine the current node's label based on the labels of its two parents. Therefore, one run of the labeling algorithm takes $O(m)$ time, where m is the number of nodes in Π. To run the labeling algorithm on all variables in φ requires $O(m \times n)$ time, where n is the number of variables in Π. In fact, the size of Π is $O(m \times n)$, because each of the m nodes can have a clause of length at most n. □

A.3 The Tool: *VaqTree*

In this section, we describe ***VaqTree*** – our vacuity detection tool for SAT-based BMC. This tool includes efficient implementations of the algorithms presented in Section 4, effectively reducing the number of extra model-checking runs needed to do complete vacuity detection. To our knowledge, this is the first vacuity detection tool for BMC.

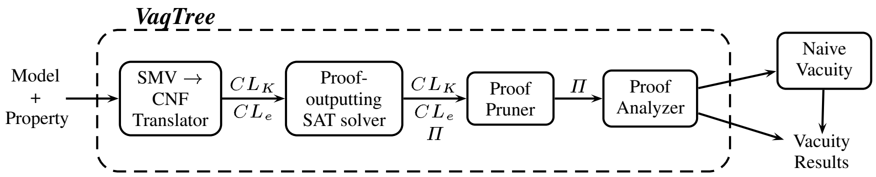

Fig. 5. ***VaqTree*** components.

The inputs to ***VaqTree*** are a model (encoded using the specification language of the model-checker NuSMV [5]) 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 5. The three components interact sequentially:

SMV $\rightarrow$ CNF Translator receives as input a SMV file containing a model and a property, which are translated into two separate CNF files, one containing CL_K and the other CL_e, 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 `bmcBmc-NonInc.c` file in the `bmc` package).

Proof-outputting SAT solver generates the resolution proof (Π) for $CL_K \cup CL_e$. We use MiniSat [9] as it can explicitly generate resolution proofs when checking satisfiability. Instead of using MiniSat's binary proof format, we developed our own XML format which allows easier proof preprocessing and facilitates future incorporation of other SAT solvers.

Proof Pruner receives CL_K, CL_e and Π and applies different preprocessing techniques to Π, producing a proof from which extraneous chains of resolution have been removed and root clauses are annotated to include their origin (CL_K or CL_e). This is a new component, written in Java (around 700 lines of code).

Proof Analyzer receives proofs that have been preprocessed using the **Proof Pruner** component and produces a list of variables found to be vacuous, and a list of variables that need to be tested using the naive method. This is a new component, written in Java (around 1.3k lines of code).

Currently, ***VaqTree*** can handle proofs that have 4 million resolutions, i.e., roughly a model with 30 SMV boolean variables, expanded up to 19 steps.

A.4 Experiments

Table 1 shows detailed results of our experiments. In this table, column "k" is the number of steps used to run BMC; "# var. in φ" is the number of atomic variables present in the property; "# vac. vars." is the number of vacuous variables; and "# resol. in Π" is the number of resolutions in the resolution proof. The next three columns report the time needed for model-checking: "Gen. CNF" is the time NuSMV took to generate the corresponding CNF theory; "Test SAT" and "Gen. Π" are the time MiniSat took to test satisfiability and generate the corresponding resolution proof respectively; and "Total" is the sum of the previous three columns.

For the naive method, we report the total times for the CNF theory generation ("Gen. CNF") and for satisfiability testing ("Test SAT"). One CNF theory is produced per each atomic variable. For irrelevance, local irrelevance and peripherality, we report how many vacuous variables were found by the partial pass ("# vac. vars. found"), how long ***VaqTree*** took to do the corresponding analysis ("Anal.") and how much time was needed to do the completeing pass ("Extra runs").

For example, test case **8** analyzes a five-variable property of the **pci** model. Only three of these variables are vacuous. The resolution proof generated when $k = 13$ has 4,283 resolutions. This property was checked in 5.59 seconds. Naive vacuity detection required five model-checking runs, taking 25.85 seconds to generate the corresponding CNF theories and 2.89 seconds to test their satisfiability, requiring a total of 28.74 seconds. Irrelevance took 0.27 seconds to find two of the vacuous variables during the partial pass. It then took 17.60 seconds to carry out the completing pass, so the total time required by irrelevance to find all three vacuous variables is 17.87 seconds. Local irrelevance took 0.28 seconds to analyze the resolution proof, finding the same two vacuous variables as irrelevance. Thus, it also takes 17.60 seconds to run the completing pass, so the total time required by local irrelevance is 17.88 seconds. Finally, peripherality took 0.47 seconds to execute the partial pass and found the same two vacuous variables, so also required 17.60 seconds to run the completing pass, taking a total of 18.07 seconds to produce complete results for test case **8**.

VaqTree, the complete experimental results and some test cases are available at `http://www.cs.toronto.edu/~jsimmond/sub_CAV2007/`.

Table 1: Statistics for vacuity detection experiments on NuSMV distribution and other examples.

Test case	Model	k	# var. in φ	# vac. vars.	# resol. in Π	Model Checking				Naive			Irrelevance				Local Irrelevance (LI)				Peripherality (P)			
						Gen. CNF (s)	Test SAT (s)	Gen. Π (s)	Total (s)	Gen. CNF (s)	Test SAT (s)	Total (s)	# vac. vars. found	Anal. (s)	Extra runs (s)	Total (s)	# vac. vars. found	Anal. (s)	Extra runs (s)	Total (s)	# vac. vars. found	Anal. (s)	Extra runs (s)	Total (s)
1	pci	13	4	1	19792	4.69	0.23	5.9	10.82	20.66	2.77	23.43	0	0.34	23.43	23.77	0	0.34	23.43	23.77	0	0.81	23.43	24.24
2	pci	13	4	3	1649	5.13	0.14	5.64	10.91	11.75	1.30	13.05	3	0.26	0	0.26	3	0.26	0	0.26	3	0.37	0	0.37
3	pci	13	4	3	1649	5.09	0.13	5.32	10.54	12.03	2.14	14.17	3	0.26	0	0.26	3	0.25	0	0.25	3	0.37	0	0.37
4	pci	13	3	1	7725	4.80	0.18	5.65	10.63	12.68	1.73	14.41	0	0.29	14.41	14.7	0	0.29	14.41	14.70	0	0.50	14.41	14.91
5	pci	13	3	1	7555	4.76	0.18	5.55	10.49	12.36	1.56	13.92	0	0.28	13.92	14.20	0	0.28	13.92	14.20	0	0.50	13.92	14.42
6	pci	13	4	3	1705	4.66	0.12	5.68	10.46	11.66	1.19	12.85	3	0.25	0	0.25	3	0.26	0	0.26	3	0.39	0	0.39
7	pci	13	4	3	1705	4.67	0.14	5.42	10.23	11.68	1.40	13.08	3	0.25	0	0.25	3	0.26	0	0.26	3	0.37	0	0.37
8	pci	13	5	3	4283	4.95	0.22	5.59	10.76	25.85	2.89	28.74	2	0.27	17.60	17.87	2	0.28	17.60	17.88	2	0.47	17.60	18.07
9	pci	13	4	3	158	4.92	0.13	5.73	10.78	20.30	0.74	21.04	3	0.18	0	0.18	3	0.18	0	0.18	3	0.22	0	0.22
10	pci	13	3	2	163	4.78	0.13	5.49	10.40	15.3	0.85	16.15	2	0.19	0	0.19	2	0.18	0	0.18	2	0.23	0	0.23
11	pci	13	4	3	165	4.7	0.13	5.4	10.23	20.78	0.79	21.57	3	0.18	0	0.18	3	0.18	0	0.18	3	0.23	0	0.23
12	pci	13	4	3	162	4.85	0.14	5.41	10.40	20.25	1.01	21.26	3	0.18	0	0.18	3	0.19	0	0.19	3	0.23	0	0.23
13	pci	13	4	3	160	5.04	0.13	5.44	10.61	20.39	0.73	21.12	3	0.19	0	0.19	3	0.17	0	0.17	3	0.23	0	0.23
14	pci	13	5	4	164	4.79	0.14	5.34	10.27	25.81	2.34	28.15	4	0.19	0	0.19	4	0.19	0	0.19	4	0.23	0	0.23
15	pci	13	5	4	162	4.70	0.13	5.40	10.23	25.46	1.54	27.00	4	0.19	0	0.19	4	0.19	0	0.19	4	0.22	0	0.22
16	pci	13	26	19	17742	4.96	0.88	6.78	12.62	137.47	35.14	172.61	8	0.41	115.16	115.57	8	0.42	115.16	115.58	8	1.70	115.16	116.86
17	pci	13	26	19	26814	4.86	1.11	6.87	12.84	134.29	36.13	170.42	8	0.46	114.63	115.09	8	0.47	114.63	115.10	8	2.38	114.63	117.01
18	pci	13	26	19	66883	5.07	1.24	7.87	14.18	147.93	42.23	190.16	8	0.60	130.62	131.22	8	0.60	130.62	131.22	8	5.67	130.62	136.29
19	pci	13	26	19	96762	4.87	2.72	10.48	18.07	133.67	41.05	174.72	8	0.74	118.51	119.25	8	0.73	118.51	119.24	8	7.96	118.51	126.47
20	pci	7	5	3	1526	1.56	0.08	3.26	4.90	8.73	0.64	9.37	2	0.24	5.99	6.23	2	0.24	5.99	6.23	2	0.36	5.99	6.35
21	pci	4	26	24	2043	0.61	0.058	2.05	2.72	16.66	1.46	18.12	19	0.25	4.86	5.11	19	0.25	4.86	5.11	19	0.39	4.86	5.25
22	pci	4	26	24	2127	0.59	0.055	2	2.64	16.61	1.56	18.17	19	0.26	5.17	5.43	19	0.25	5.17	5.42	19	0.41	5.17	5.58
23	prod-cell	30	4	4	13946	11.06	0.61	6.59	18.26	40.37	0.94	41.31	4	0.39	0	0.39	4	0.38	0	0.38	4	0.90	0	0.90
24	prod-cell	30	6	6	67307	10.53	0.38	5.20	16.11	77.67	1.80	79.47	0	0.65	79.47	80.12	0	0.64	79.47	80.11	1	4.62	66.41	71.03
25	prod-cell	30	7	7	24146	11.3	0.32	4.87	16.49	85.36	1.75	87.11	1	0.55	74.66	75.21	1	0.54	74.66	75.20	1	1.97	74.66	76.63
26	prod-cell	30	6	6	15898	9.98	0.19	4.12	14.29	75.39	1.87	77.26	0	0.44	77.26	77.70	0	0.44	77.26	77.70	0	1.35	77.26	78.61
27	prod-cell	30	5	5	15734	9.50	0.28	4.64	14.42	62.85	1.24	64.09	0	0.42	64.09	64.51	0	0.41	64.09	64.50	1	1.19	51.52	52.71
28	prod-cell	30	4	4	24615	9.25	0.16	4.18	13.59	40.61	1.29	41.90	0	0.41	41.90	42.31	0	0.40	41.90	42.30	0	1.47	41.90	43.37
29	prod-cell	30	8	8	24747	10.34	0.23	4.54	15.11	100.24	1.99	102.23	0	0.52	102.23	102.75	0	0.53	102.23	102.76	0	2.27	102.23	104.50
30	prod-cell	30	5	5	15177	9.27	0.35	4.31	13.93	55.87	3.61	59.48	0	0.41	59.48	59.89	0	0.40	59.48	59.88	0	1.17	59.48	60.65
31	prod-cell	30	4	4	35955	8.89	1.01	6.47	16.37	43.87	2.52	46.39	0	0.51	46.39	46.90	0	0.49	46.39	46.88	1	1.87	35.48	37.35
32	prod-cell	30	4	4	101479	9.24	0.69	6.22	16.15	43.35	3.18	46.53	0	0.71	46.53	47.24	0	0.71	46.53	47.24	0	4.63	46.53	51.16
33	prod-cell	30	6	6	32716	9.91	0.86	6.93	17.70	74.37	2.40	76.77	0	0.58	76.77	77.35	0	0.57	76.77	77.34	0	2.30	76.77	79.07
34	prod-cell	30	5	5	14854	9.39	0.30	4.98	14.67	54.8	6.69	61.49	5	0.41	0	0.41	5	0.40	0	0.40	5	1.09	0	1.09
35	prod-cell	30	5	5	80108	9.10	0.99	6.81	16.90	52.65	4.40	57.05	0	0.73	57.05	57.78	0	0.74	57.05	57.79	0	4.36	57.05	61.41
36	prod-cell	30	8	8	21654	11.43	0.35	4.64	16.42	118.41	2.92	121.33	0	0.52	121.33	121.85	0	0.52	121.33	121.85	1	1.97	106.42	108.39
37	prod-cell	30	9	9	59079	10.31	0.40	5.91	16.62	108.15	5.90	114.05	0	0.67	114.05	114.72	0	0.66	114.05	114.71	0	5.71	114.05	119.76

Continued on Next Page. . .

Table 1 – Continued

Test case	Model	k	# var. in φ	# vac. vars.	# resol. in Π	Model Checking				Naive			Irrelevance				Local Irrelevance (LI)				Peripherality (P)			
						Gen. CNF (s)	Test SAT (s)	Gen. Π (s)	Total (s)	Gen. CNF (s)	Test SAT (s)	Total (s)	# vac. vars. found	Anal. (s)	Extra runs (s)	Total (s)	# vac. vars. found	Anal. (s)	Extra runs (s)	Total (s)	# vac. vars. found	Anal. (s)	Extra runs (s)	Total (s)
38	prod-cell	30	5	5	27070	9.25	0.37	5.04	14.66	54.85	2.24	57.09	0	0.53	57.09	57.62	0	0.54	57.09	57.63	0	1.72	57.09	58.81
39	prod-cell	30	5	5	32664	8.98	0.48	5.29	14.75	53.81	1.95	55.76	0	0.55	55.76	56.31	0	0.55	55.76	56.31	0	2.01	55.76	57.77
40	prod-cell	30	4	4	39312	9.18	0.44	5.15	14.77	41.74	1.50	43.24	0	0.54	43.24	43.78	0	0.54	43.24	43.78	0	1.99	43.24	45.23
41	prod-cell	30	10	10	20771	11.78	0.20	4.71	16.69	148.20	3.00	151.20	0	0.53	151.20	151.73	0	0.53	151.20	151.73	0	2.25	151.20	153.45
42	prod-cell	30	8	8	22039	11.42	0.72	6.03	18.17	123.54	3.03	126.57	0	0.54	126.57	127.11	0	0.53	126.57	127.10	0	1.97	126.57	128.54
43	prod-cell	20	4	4	17565	4	0.11	2.81	6.92	17.37	0.35	17.72	0	0.38	17.72	18.10	0	0.38	17.72	18.10	1	0.91	13.82	14.73
44	prod-cell	20	6	6	23833	3.97	0.13	2.70	6.80	29.07	0.69	29.76	0	0.42	29.76	30.18	0	0.42	29.76	30.18	0	1.41	29.76	31.17
45	prod-cell	20	7	7	11023	3.84	0.12	2.69	6.65	31.11	0.66	31.77	1	0.38	27.35	27.73	1	0.37	27.35	27.72	1	0.90	27.35	28.25
46	prod-cell	20	6	6	9364	3.84	0.14	3.31	7.29	25.84	0.46	26.30	4	0.36	8.63	8.99	4	0.35	8.99	9.34	4	0.74	8.99	9.73
47	prod-cell	20	5	5	8380	4.01	0.13	2.91	7.05	23.12	0.74	23.86	1	0.34	19.60	19.94	2	0.33	13.69	14.02	2	0.67	13.69	14.36
48	prod-cell	20	4	4	10822	3.92	0.08	2.49	6.49	15.16	0.56	15.72	0	0.33	15.72	16.05	0	0.34	15.72	16.06	0	0.7	15.72	16.42
49	prod-cell	20	8	8	25923	4.20	0.16	2.95	7.31	32.73	0.87	33.60	0	0.48	33.60	34.08	0	0.48	33.60	34.08	0	1.85	33.60	35.45
50	prod-cell	20	5	5	59884	3.86	0.42	4.48	8.76	19.35	0.50	19.85	0	0.55	19.85	20.40	0	0.55	19.85	20.40	0	2.64	19.85	22.49
51	prod-cell	20	4	4	10793	3.84	0.18	3.23	7.25	15.07	0.66	15.73	2	0.36	8.14	8.50	4	0.36	0	0.36	4	0.71	0	0.71
52	prod-cell	20	4	4	25764	4.01	0.16	2.84	7.01	16.81	0.52	17.33	0	0.40	17.33	17.73	0	0.39	17.33	17.72	0	1.15	17.33	18.48
53	prod-cell	20	6	6	8120	4.15	0.15	3.33	7.63	27.20	0.52	27.72	6	0.34	0	0.34	6	0.33	0	0.33	6	0.70	0	0.70
54	prod-cell	20	5	5	8840	3.8	0.10	2.76	6.66	19.72	1.24	20.96	3	0.33	8.80	9.13	4	0.34	4.21	4.55	4	0.69	4.21	4.90
55	prod-cell	20	5	5	25425	3.9	0.22	3.81	7.93	21.18	1.03	22.21	0	0.43	22.21	22.64	0	0.42	22.21	22.63	0	1.26	22.21	23.47
56	prod-cell	20	8	8	20602	4.46	0.27	3.73	8.46	41.14	1.20	42.34	0	0.45	42.34	42.79	2	0.45	31.13	31.58	2	1.50	31.13	32.63
57	prod-cell	20	9	9	10923	3.9	0.09	2.68	6.67	41.01	0.73	41.74	2	0.39	32.22	32.61	3	0.39	27.71	28.1	3	1.05	27.71	28.76
58	prod-cell	20	5	5	11215	3.83	0.10	2.6	6.53	21.29	0.36	21.65	1	0.35	17.44	17.79	2	0.35	13.13	13.48	2	0.77	13.13	13.90
59	prod-cell	20	5	5	9252	4.05	0.08	2.69	6.82	21.12	0.34	21.46	1	0.34	17.29	17.63	2	0.34	12.89	13.23	2	0.69	12.89	13.58
60	prod-cell	20	4	4	16124	3.83	0.15	3.03	7.01	16.43	0.40	16.83	1	0.40	12.76	13.16	2	0.41	8.52	8.93	2	0.88	8.52	9.4
61	prod-cell	20	10	10	14767	4.89	0.11	3.54	8.54	51.70	1.09	52.79	3	0.41	36.89	37.30	4	0.41	31.26	31.67	4	1.39	31.26	32.65
62	prod-cell	20	8	8	15334	4.58	0.14	2.97	7.69	39.25	0.89	40.14	0	0.42	40.14	40.56	0	0.42	40.14	40.56	0	1.25	40.14	41.39
63	prod-cell	10	4	4	2254	0.89	0.03	1.26	2.18	3.72	0.10	3.82	3	0.26	1.01	1.27	3	0.25	1.01	1.26	3	0.39	1.01	1.4
64	prod-cell	10	6	6	2304	0.92	0.03	1.26	2.21	7.65	0.14	7.79	0	0.27	7.79	8.06	0	0.27	7.79	8.06	0	0.40	7.79	8.19
65	prod-cell	10	7	7	5345	1	0.04	1.32	2.36	6.71	0.23	6.94	1	0.29	5.94	6.23	1	0.29	5.94	6.23	1	0.49	5.94	6.43
66	prod-cell	10	6	6	5275	0.88	0.04	1.25	2.17	5.55	0.18	5.73	6	0.28	0	0.28	6	0.28	0	0.28	6	0.49	0	0.49
67	prod-cell	10	5	5	5320	1.02	0.04	1.41	2.47	4.81	0.16	4.97	5	0.28	0	0.28	5	0.29	0	0.29	5	0.47	0	0.47
68	prod-cell	10	4	4	3798	0.91	0.03	1.27	2.21	3.57	0.12	3.69	2	0.27	1.86	2.13	2	0.27	1.86	2.13	2	0.43	1.86	2.29
69	prod-cell	10	8	8	2764	0.99	0.03	1.26	2.28	7.52	0.23	7.75	1	0.26	6.78	7.04	1	0.26	6.78	7.04	1	0.42	6.78	7.2
70	prod-cell	10	5	5	5232	1.20	0.04	1.33	2.57	4.63	0.15	4.78	1	0.28	3.82	4.10	2	0.28	2.86	3.14	2	0.48	2.86	3.34
71	prod-cell	10	4	4	4068	1.35	0.03	1.27	2.65	3.87	0.10	3.97	2	0.27	2.16	2.43	3	0.27	0.95	1.22	3	0.44	0.95	1.39
72	prod-cell	10	4	4	2756	0.96	0.03	1.27	2.26	3.64	0.13	3.77	1	0.26	2.82	3.08	1	0.26	2.82	3.08	1	0.40	2.82	3.22
73	prod-cell	10	6	6	4425	0.84	0.04	1.30	2.18	5.47	0.19	5.66	2	0.28	3.74	4.02	2	0.28	3.74	4.02	2	0.46	3.74	4.2
74	prod-cell	10	5	5	3802	0.92	0.04	1.28	2.24	4.55	0.17	4.72	4	0.27	1.01	1.28	4	0.28	1.01	1.29	4	0.43	1.01	1.44
75	prod-cell	10	5	5	2802	0.91	0.03	1.44	2.38	4.53	0.14	4.67	2	0.26	2.80	3.06	2	0.26	2.80	3.06	2	0.41	2.80	3.21

Continued on Next Page…

Table 1 – Continued

Test case	Model	k	# var. in φ	# vac. vars.	# resol. in Π	**Model Checking**				**Naive**			**Irrelevance**				**Local Irrelevance (LI)**				**Peripherality (P)**			
						Gen. CNF (s)	**Test SAT (s)**	**Gen. Π (s)**	**Total (s)**	**Gen. CNF (s)**	**Test SAT (s)**	**Total (s)**	**# vac. vars. found**	**Anal. (s)**	**Extra runs (s)**	**Total (s)**	**# vac. vars. found**	**Anal. (s)**	**Extra runs (s)**	**Total (s)**	**# vac. vars. found**	**Anal. (s)**	**Extra runs (s)**	**Total (s)**
76	prod-cell	10	8	8	3732	1.16	0.03	1.36	2.55	7.72	0.21	7.93	5	0.28	2.96	3.24	6	0.27	1.98	2.25	6	0.46	1.98	2.44
77	prod-cell	10	9	9	3010	1.50	0.03	1.28	2.81	8.93	0.22	9.15	6	0.27	3.12	3.39	7	0.27	1.94	2.21	7	0.45	1.94	2.39
78	prod-cell	10	5	5	2585	0.86	0.03	1.25	2.14	4.98	0.14	5.12	2	0.26	2.93	3.19	2	0.26	2.93	3.19	2	0.40	2.93	3.33
79	prod-cell	10	5	5	2556	1.06	0.03	1.30	2.39	4.70	0.12	4.82	2	0.26	2.98	3.24	2	0.26	2.98	3.24	2	0.40	2.98	3.38
80	prod-cell	10	4	4	5317	1.26	0.04	1.27	2.57	3.53	0.12	3.65	4	0.29	0	0.29	4	0.29	0	0.29	4	0.46	0	0.46
81	prod-cell	10	10	10	2497	3.15	0.06	1.29	4.5	9.68	0.27	9.95	3	0.26	6.97	7.23	4	0.26	4.94	5.20	4	0.42	4.94	5.36
82	prod-cell	10	8	8	2348	0.88	0.033	1.25	2.16	7.52	0.22	7.74	3	0.27	4.84	5.11	3	0.26	4.84	5.10	3	0.41	4.84	5.25
83	abp4	19	1	0	1289374	2.79	10.73	34.14	47.66	2.93	1.79	4.72	0	5.51	4.72	10.23	0	5.72	4.72	10.44	0	98.62	4.72	103.34
84	abp4	19	3	2	1050234	3.14	6.45	29.43	39.02	8.43	20.76	29.19	0	5.07	29.19	34.26	0	5.22	29.19	34.41	0	67.54	29.19	96.73
85	abp4	19	3	2	2246095	2.99	19.03	49.63	71.65	8.81	26.43	35.24	0	8.23	33.78	42.01	0	8.22	33.78	42	0	412.30	33.78	446.08
86	abp4	19	2	0	795705	3.07	5.04	21.28	29.39	5.54	6.29	11.83	0	2.69	25.64	28.33	0	2.71	25.64	28.35	0	37.21	25.64	62.85
87	abp4	17	1	0	379311	2.39	2.55	11.55	16.49	2.93	1.79	4.72	0	1.42	4.72	6.14	0	1.41	4.72	6.13	0	8.41	4.72	13.13
88	abp4	17	2	0	335307	2.27	2.11	9.23	13.61	5.66	12.44	18.10	0	1.30	18.1	19.40	0	1.31	18.10	19.41	0	8.06	18.10	26.16
89	abp4	17	3	2	362949	2.39	1.75	9.41	13.55	8.43	20.76	29.19	0	1.37	29.19	30.56	0	1.39	29.19	30.58	0	10.69	29.19	39.88
90	abp4	17	4	2	929686	2.42	5.61	21.78	29.81	12.32	18.24	30.56	0	2.98	30.56	33.54	0	2.94	30.56	33.5	0	39.35	30.56	69.91
91	abp4	17	3	2	2342553	3.12	22.59	49.16	74.87	8.68	25.10	33.78	0	6.88	33.78	40.66	0	6.92	33.78	40.70	0	408.34	33.78	442.12
92	abp4	17	2	0	647422	2.57	3.13	16.04	21.74	5.6	20.04	25.64	0	2.19	20.04	22.23	0	2.16	20.04	22.20	0	25.91	20.04	45.95
93	toyFGS04	18	6	6	297	18.88	0.26	5.27	24.41	114.78	0.76	115.54	3	0.23	57.39	57.62	3	0.22	57.39	57.61	3	0.29	57.39	57.68
94	toyFGS04	18	12	12	308	19.13	0.16	5.28	24.57	224.79	1.40	226.19	6	0.26	132.43	132.69	6	0.26	132.43	132.69	6	0.33	132.43	132.76
95	toyFGS04	18	6	0	318	18.35	0.15	5.17	23.67	126.28	32.03	158.31	0	0.22	158.31	158.53	0	0.22	158.31	158.53	0	0.29	158.31	158.60
96	toyFGS04	18	4	0	308	18.57	0.14	5.45	24.16	75.18	22.26	97.44	0	0.22	97.44	97.66	0	0.22	97.44	97.66	0	0.27	97.44	97.71
97	toyFGS04	18	4	0	8072	14.14	0.21	3.3	17.65	57.91	10.60	68.51	0	0.33	68.51	68.84	0	0.33	68.51	68.84	0	0.60	68.51	69.11
98	toyFGS04	18	6	0	7985	14.47	0.21	3.63	18.31	88.94	11.48	100.42	0	0.34	100.42	100.76	0	0.34	100.42	100.76	0	0.68	100.42	101.10
99	toyFGS04	18	6	6	293	19.80	0.15	5.61	25.56	111.91	0.66	112.57	2	0.21	75.08	75.29	2	0.22	75.08	75.30	2	0.27	75.08	75.35
100	toyFGS04	9	6	6	297	11.86	0.07	2.65	14.58	68.38	0.30	68.68	3	0.21	34.33	34.54	3	0.22	34.33	34.55	3	0.27	34.33	34.60
101	toyFGS04	9	12	12	308	11.96	0.068	2.65	14.68	141.41	0.70	142.11	6	0.22	70.92	71.14	6	0.22	70.92	71.14	6	0.29	70.92	71.21
102	toyFGS04	9	6	0	318	11.89	0.07	2.52	14.48	67.95	6.33	74.28	0	0.21	74.28	74.49	0	0.21	74.28	74.49	0	0.27	74.28	74.55
103	toyFGS04	9	4	0	308	11.86	0.07	2.57	14.50	50.15	3.90	54.05	0	0.21	54.05	54.26	0	0.22	54.05	54.27	0	0.26	54.05	54.31
104	toyFGS04	9	4	0	4075	10.76	0.07	1.57	12.40	54.11	1.62	55.73	0	0.29	55.73	56.02	0	0.28	55.73	56.01	0	0.44	55.73	56.17
105	toyFGS04	9	6	0	4115	10.74	0.07	1.59	12.40	81.59	2.43	84.02	0	0.29	84.02	84.31	0	0.29	84.02	84.31	0	0.47	84.02	84.49
106	toyFGS04	9	6	6	293	13.13	0.07	2.57	15.77	74.49	0.31	74.80	2	0.21	50.14	50.35	2	0.21	50.14	50.35	2	0.26	50.14	50.4
107	msi_wtrans	40	5	3	66	21.85	0.20	8.39	30.44	120.15	65.70	185.85	3	0.21	112.59	112.80	3	0.20	112.59	112.79	3	0.24	112.59	112.83
108	msi_wtrans	40	5	4	66	23.53	0.20	9.15	32.88	120.16	73.28	193.44	3	0.2	120.30	120.50	3	0.21	120.30	120.51	3	0.25	120.30	120.55
109	msi_wtrans	40	6	4	66	21.56	0.21	8.46	30.23	156.61	93.23	249.84	4	0.21	0	0.21	4	0.21	0	0.21	4	0.24	0	0.24
110	msi_wtrans	20	2	0	63	5.28	0.20	4.01	9.49	12.91	20.63	33.54	0	0.15	33.54	33.69	0	0.16	33.54	33.70	0	0.18	33.54	33.72
111	msi_wtrans	20	2	0	62	5.13	0.09	3.79	9.01	11.58	19.82	31.40	0	0.15	31.40	31.55	0	0.16	31.40	31.56	0	0.19	31.40	31.59
112	msi_wtrans	20	6	0	374744	4.68	10.21	21.20	36.09	32.04	26.13	58.17	0	1.78	58.17	59.95	0	1.80	58.17	59.97	0	16.42	58.17	74.59
113	msi_wtrans	20	6	0	274539	4.72	9.76	19.65	34.13	28.92	28.27	57.19	0	1.48	57.19	58.67	0	1.51	57.19	58.70	0	11.99	57.19	69.18

Continued on Next Page...

Table 1 – Continued

Test case	**Model**	**k**	**# var. in φ**	**# vac. vars.**	**# resol. in Π**	**Model Checking**				**Naive**			**Irrelevance**				**Local Irrelevance (LI)**				**Peripherality (P)**			
						Gen. CNF (s)	**Test SAT (s)**	**Gen. Π (s)**	**Total (s)**	**Gen. CNF (s)**	**Test SAT (s)**	**Total (s)**	**# vac. vars. found**	**Anal. (s)**	**Extra runs (s)**	**Total (s)**	**# vac. vars. found**	**Anal. (s)**	**Extra runs (s)**	**Total (s)**	**# vac. vars. found**	**Anal. (s)**	**Extra runs (s)**	**Total (s)**
114	msi_wtrans	20	5	3	66	5.27	0.10	3.85	9.22	27.92	15.65	43.57	3	0.16	26.75	26.91	3	0.16	26.75	26.91	3	0.19	26.75	26.94
115	msi_wtrans	20	5	4	66	5.17	0.11	3.94	9.22	27.5	16.62	44.12	3	0.16	27.25	27.41	3	0.15	27.25	27.40	3	0.19	27.25	27.44
116	msi_wtrans	20	6	4	66	5.31	0.10	3.91	9.32	33.3	18.60	51.90	4	0.19	0	0.19	4	0.19	0	0.19	4	0.22	0	0.22
117	msi_wtrans	20	2	0	1604	4.62	0.11	3.68	8.41	9.29	11.84	21.13	0	0.24	21.13	21.37	0	0.24	21.13	21.37	0	0.35	21.13	21.48
118	msi_wtrans	20	1	0	1001	5.41	0.10	3.94	9.45	5.73	8.20	13.93	0	0.22	13.93	14.15	0	0.23	13.93	14.16	0	0.30	13.93	14.23
119	msi_wtrans	20	1	0	932	5.39	0.21	3.94	9.54	5.79	12.09	17.88	0	0.22	17.88	18.10	0	0.22	17.88	18.10	0	0.29	17.88	18.17
120	msi_wtrans	20	2	0	927	5.58	0.10	0.47	6.15	12.39	18.96	31.35	0	0.22	31.35	31.57	0	0.22	31.35	31.57	0	0.32	31.35	31.67
121	msi_wtrans	20	8	0	371653	4.76	12.49	24.41	41.66	41.96	30.22	72.18	0	1.79	72.18	73.97	0	1.80	72.18	73.98	0	16.16	72.18	88.34