Irrelevance and Vacuity in SAT-Based Model-Checking

Benet Devereux and Arie Gurfinkel
Email: {benet,arie}@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 an execution trace illustrating the error – a counterexample. However, it is harder to provide illustrative information if a property does hold, and this is necessary because a property might be trivially true, or rely on unexamined assumptions about the artifact or its operating environment that may not be valid. One way of finding such information is to check for *vacuity*: that the truth value of one or more subformulas of the property does not affect its overall value. A SAT-based model-checker, on a successful run, 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 give a property of resolution proofs of unsatisfiability (for the translation of a model-checking problem) which is equivalent to a restricted definition of vacuity.

1 Introduction

Model-checking [5] 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; it is more rigorous than testing, and considerably less demanding of labour and skill than theorem-proving. To use the model-checking decision procedure, the following steps (illustrated in Figure 1) are necessary.

First, extract a finite-state model of the artifact (this extraction can also be automated); second, translate the criterion of correctness into a propositional temporal logic. The decision procedure accepts these two inputs and reports whether the model satisfied the property or not; if not, it can, in some cases, produce a counterexample: a computation of the model illustrating the error. Because the extraction process discards information, the counterexample may be *spurious* – the real artifact cannot produce it. In this case, a more exact model needs to be extracted for analysis. If the counterexample is valid, however, it provides useful diagnostic information for debugging.

A successful model-checking run tells the user nothing but the fact that the property holds. This is, in many ways, less useful than a failure [12]! A property failing to hold is always bad; however, a property can hold for unacceptable reasons, resting on

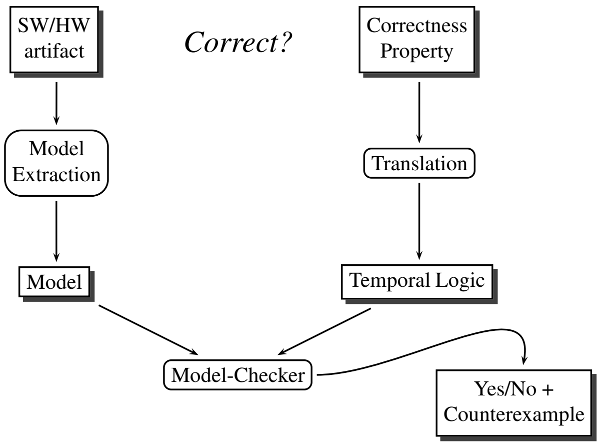

Figure 1: Overview of the model-checking process.

incorrect assumptions that we have failed to check. For instance, if a property "every request is universally acknowledged" is satisfied in an environment where no request is ever generated, then our constraints on the environment are not reasonable. A model-checker, then, is only as useful as the results it provides: and information explaining the results is more useful than just a decision.

There has been, therefore, research on providing added information from successful model-checking runs as well. It is possible to generate a proof of the property [12, 11] and exhibit it to the user; however, for real systems with many variables and states this proof tends also to be very large and difficult to understand. A more promising kind of diagnostic which can be automatically detected is *vacuity* [6, 13, 1]. Informally speaking, a property is said to be vacuous if it has some subformula which is not relevant to its truth value, or if it is a tautology.

In contrast to previous work on vacuity, which treated the model-checker as a black box, we concentrate in this report specifically on model-checking using a SAT solver. SAT-based model-checking is currently an active, fruitful area of research and application. A SAT-based model-checker is guaranteed to produce results besides a decision: if the property is false, then the solver produces a satisfying assignment, which can be easily exhibited as a counterexample. If the property is true, then the SAT translation of the model-checking problem is unsatisfiable, and the solver produces a resolution proof of its unsatisfiability.

The question we address is: how can we relate this resolution proof to the model and property currently under analysis? In the context of declarative modelling, Shlyakhter et al. [14] used proofs of unsatisfiability to identify the constraints relevant to a desired property of the specification; if very few constraints were relevant, this served as a warning sign that the property might be trivially satisfied, or rest on invalid assumptions. Since this kind of analysis is similar in spirit to vacuity detection, we attempt to relate Shlyakhter's definition of *irrelevance of constraints* in declarative modelling to the existing definition of vacuity in model-checking.

In the remainder of the paper, we show that irrelevance in the sense of [14] is a weaker notion than vacuity. We develop two stronger notions of irrelevance, *local irrelevance* and *peripherality* of variables, defined on resolution proofs generated from model-checking problems, and show that a variable is peripheral in a proof if and only iff the property being checked is vacuous in that variable. This yields an algorithm for vacuity-detection which is linear in the size of the resolution proof.

2 Background

In this section, we provide the necessary background. We describe linear temporal logic (LTL), Kripke models, the bounded model-checking problem, and satisfiability.

2.1 Temporal Logic

Linear Temporal Logic – LTL – is a linear-time propositional temporal logic. Through the following presentation we assume an infinite, denumerable set AP of atomic propositions. A *trace* $\pi : nat \to 2^{AP}$ is an infinite sequence of assignments of to AP; π_i is the ith element of π, and π^i the suffix of π beginning at i. The set of LTL formulas is defined by the following grammar:

$$\begin{array}{lcl}\varphi & := & p \in AP \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \\ & & \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \mathbf{X}\varphi\end{array}$$

LTL formulas define constraints on traces; an infinite sequence of assignments to variables either satisfies a formula or not. The satisfaction relation $\models$ is defined as follows:

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

The denotation of an LTL formula φ is the set $[\![\varphi]\!] = \{\pi \mid \pi \models \varphi\}$. The set of all variables used in φ is written $Vars(\varphi)$.

For temporal operators **G** and **X**, k-depth satisfaction can also be defined: let $\pi : \{0, \ldots, k-1\} \to 2^{AP}$ be a length-k path. Then the k-depth satisfaction relation is defined as follows:

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

If $\pi \not\models_k \varphi$, then we say that π is a length-k counterexample to φ. For example, if $\varphi = \mathbf{G}(p \wedge q)$, π with $p \notin \pi_2$ is a length-2 (or greater) counterexample. The set $Cex_k(\varphi)$ contains all the length-k counterexamples to φ. Note that **F** does not have k-depth semantics. Properties that do are called *safety* properties.

2.2 Models

A Kripke structure is a tuple $K = (S, Init, R, I)$, where

- S is a finite set of states, with $Init$ the initial states
- $R \subseteq S \times S$ is the (left-total) transition relation; the *successors* of a state s are contained in the set $\{s' \mid R(s, s')\}$
- $I : S \to 2^{AP}$ labels each state with the set of propositional formulas that hold in it.

A *run* of K is a sequence $\rho : nat \to S$ of states where $\rho_0 \in Init$ and for all i, $R(\rho_i, \rho_{i+1})$. The *trace* of the run is obtained by applying the state-labelling function stepwise: $\pi_i = I(\rho_i)$. Many runs of a Kripke structure may map to the same trace. The runs of K are denoted $Runs(K)$; the traces, $Tr(K)$. For two Kripke structures, K and K', if $Tr(K) \subseteq Tr(K')$, we say that K' is an *abstraction* of K.

A variable p is *free at state s of K* iff its behaviour is not constrained at s: that is, for any successor s' of s where $p \in I(s')$ (resp. $p \notin I(s')$), there is a distinct successor s'' such that $I(s'') = I(s') - \{p\}$ (resp. $I(s'') = I(s') \cup \{p\}$). Variable p is *free in* K iff it is free at every $s \in S$. All but a finite number of variables are free in any finite-state K.

Finally, given Kripke structure K and LTL formula φ, $K \models \varphi$ iff every trace of K satisfies φ: $Tr(K) \subseteq [\![\varphi]\!]$. The k-depth satisfaction can be defined easily, by considering only length k traces of K.

We assume, in general, that Kripke structures never have two states with the same labelling. This assumption does not weaken the formalism: rather than giving two states the same label but different future behaviour, we can always take another atom from AP to distinguish between the two states.

Rather than explicitly giving a transition relation on pairs of states, a Kripke structure can be defined by sets of constraints on *prestate* and *poststate* variables. Let AP' be the set obtained by marking each variable of AP as primed; then a constraint $C : 2^{AP} \times 2^{AP'} \to \{t, f\}$ defines whether a pair of subsets of AP and AP' is permitted as labels for states before and after a transition. Variables (both prestate and poststate) are free or unfree in constraints in the usual way, and, again, only finitely many variables may be unfree in any constraint. Constraints can be conjoined or disjoined. Any representation of a Kripke structure by a constraint of this kind is called *symbolic*. A constraint $Init$ representing the initial states is also required.

2.3 Bounded Model-Checking

The *bounded model-checking* (BMC) problem is this: given a Kripke structure K, an LTL formula φ, and a bound k, show that no length k trace of K is a counterexample to φ: that is to say, $K \models_k \varphi$. We denote any instance of the problem $BMC_k(K, \varphi)$. If we are not concerned with a particular value of k, we omit it.

Given a constraint R representing the transition relation of K, BMC is expressible as a constraint-satisfaction problem (which, since all the variables are boolean, can be given to a SAT solver) by creating copies of the variables in $V = Vars(K) \cup Vars(\varphi)$

for each time-step from 0 through k. For $v \in Vars$, there is $v_0, \ldots, v_k$; V_i stands for all the time-0 variables.

We construct the *path constraint* PC to be:

$$Init[V \leftarrow V_0] \wedge R[V \leftarrow V_0, V' \leftarrow V_1] \wedge \cdots \wedge R[V \leftarrow V_{k-1}, V' \leftarrow V_k]$$

which constrains paths to be only valid length k traces of K. The *error constraint* $Err_k(\varphi, 0)$ is defined recursively as follows:

$$\begin{array}{l}
Err_k(p,i) \triangleq \neg p_i \\
Err_k(\neg p,i) \triangleq p_i \\
Err_k(\varphi \wedge \psi, i) \triangleq Err_k(\varphi, i) \vee Err_k(\varphi, \psi) \\
Err_k(\varphi \vee \psi, i) \triangleq Err_k(\varphi, i) \wedge Err_k(\varphi, \psi) \\
Err_k(\mathbf{X}\varphi, i) \triangleq \begin{cases} k = 0 & \textsf{false} \\ k \neq 0 & Err_{k-1}(\varphi, i+1) \end{cases} \\
Err_k(\mathbf{G}\varphi, i) \triangleq \begin{cases} k = 0 & Err_k(\varphi, i) \\ k \neq 0 & Err_k(\varphi, i) \vee Err_{k-1}(\mathbf{G}\varphi, i+1) \end{cases}
\end{array}$$

For example, take the property $\mathbf{G}(p \wedge \mathbf{X}q)$, with $k = 2$. We have

$$\begin{array}{rcl}
Err_2(\mathbf{G}(p \wedge \mathbf{X}q), 0) & = & Err_2(p \wedge \mathbf{X}q, 0) \vee \\
& & \; Err_1(\mathbf{G}(p \wedge \mathbf{X}q), 1) \\
& = & (Err_2(p, 0) \vee Err_2(\mathbf{X}q, 0)) \vee \\
& & \; (Err_1(p \wedge \mathbf{X}q, 1) \vee \\
& & \; Err_0(\mathbf{G}(p \wedge \mathbf{X}q), 2)) \\
& = & (Err_2(p, 0) \vee Err_2(\mathbf{X}q, 0)) \vee \\
& & \; ((Err_1(p, 1) \vee Err_1(\mathbf{X}q, 0)) \vee \\
& & \; (Err_0(p, 2) \wedge Err_0(\mathbf{X}q, 0))) \\
& = & (\neg p_0 \vee Err_2(q, 1)) \vee \\
& & \; ((\neg p_1 \vee Err_1(q, 2)) \vee \\
& & \; (\neg p_2 \vee \textsf{false})) \\
& = & \neg p_0 \vee \neg q_1 \vee \\
& & \; \neg p_1 \vee \neg q_2 \vee \\
& & \; \neg p_2
\end{array}$$

Finally check that $PC \wedge Err_k(\varphi, 0)$ is unsatisfiable, which proves that no length-k trace of K is a length-k counterexample to φ. If there is a satisfying assignment, this corresponds to a counterexample of length k (or less) which can be used in debugging.

2.4 Satisfiability

Any propositional formula can be represented by a set of clauses; this is called conjunctive normal form (CNF). A *clause* is a disjunction of zero or more literals, where each literal is either a boolean variable or its negation. A set of clauses is *satisfiable* if there exists an assignment to variables that satisfies each clause. We assume that a set of clauses is not tautological, and that no clause contains a variable and its negation.

Given two clauses $c_1 = (v \vee A)$ and $c_2 = (\neg v \vee B)$, the clause $(A \vee B)$ is called a *resolvent* of c_1 and c_2 on v, provided that $(A \vee B)$ does not contain a variable and its negation.

A resolution proof is a directed acyclic graph $\Pi = (V_\Pi, E_\Pi)$. We use the notation $Roots(\Pi)$ to denote the roots of Π, and $Leaf(\Pi)$ to denote its leaves. Each node in the graph is labeled by a clause c, and has either zero or exactly two parents. If c has two parents labeled by clauses c_1 and c_2, then there must exist a variable v such that c is a resolvent of c_1 and c_2 on v. A resolution proof Π is a proof of unsatisfiability of a set of clauses A if and only if (a) $Roots(\Pi) \subseteq A$ and (b) $Leaf(\Pi)$ is labeled by the empty clause.

Given a set of clauses $C = A \cup B$, we say that a variable l is *local* to A if it only occurs in clauses belonging to A; that is, $CL(l) \subseteq A$. A variable l is *global* in C if it occurs in both A and B clauses; that is, $CL(l) \cap A \neq \emptyset$ and $CL(l) \cap B \neq \emptyset$.

A satisfiability solver is a complete decision procedure that decides whether a set of clauses is satisfiable.

3 Bounded Model-Checking Revisited

In the definition of BMC given in Section 2.3, the transition constraint R was treated as monolithic, and the SAT solver returned no information beyond a report of unsatisfiability. If, however, the transition constraint is structured in such a way that clauses in the CNF translation can be identified with meaningful elements of the path-constraint (or error-constraint), then a resolution proof of unsatisfiability can be used for further analysis of the model. We represent the transition relation of a Kripke structure as a conjunction of parallel assignments (one per state variable), and each of these in turn is a conjunction of cases. This is based on the language of the model-checker SMV [4].

3.1 Parallel Assignments

The transition relation of a Kripke structure K is specified by *parallel assignments* if, for every variable p, and every state s, *either* p is free at s, *or* s exactly determines the value of p in the next state: p is either true in all next states or false in all next states. So for each variable p which is unfree in K, there exists a transition constraint R_p which constraints the post-state value of p depending on the current state variables. The transition constraint R is the conjunction of the individual-variable constraints:

$$R \triangleq \bigwedge_{p \in AP} R_p$$

Furthermore, each individual-variable constraint R_p is itself a conjunction of several constraints. In modelling, R_p is generally expressed as a conjunction of n implications $\kappa_i^p = F_i \Rightarrow \ell$ where F is a constraint only on prestate variables, and ℓ is either p' or $\neg p'$. It is required that cases be *disjoint*:

$$\forall i \neq j \cdot F_i \wedge F_j = \mathsf{false}$$

and it is assumed that if a state s does not satisfy any of the F_i, p is free at s. Given these conditions, we express R_p as:

$$R_p \triangleq \bigwedge_{0 \leq i < n} \kappa_i^p$$

When generating path constraints, R is unrolled to yield $R(0,1) \wedge \ldots \wedge R(k-1,k)$ with appropriate renaming of variables (in $R(i,i+1)$, every unprimed p is rewritten to p_i, and every primed p to p_{i+1}.

As an example, take a model with two variables a, b. We construct R_a only. The source specification for a is presented using an initial value and a next-block, as follows:

```
init(a):=T
next(a):=case
   b: F
   1: a
 esac;
```

The first case of the next block translates simply to $\kappa_1 = b \Rightarrow a'$. The cases are read in order, so the condition of the second case implicitly includes ".. and none of the previous cases", so it is in fact $\neg b$; and in this case a' has the same value as a, so we need the two cases $\kappa_2 = \neg b \wedge a \Rightarrow a'$ and $\kappa_3 = \neg b \wedge \neg a' \Rightarrow \neg a'$.

3.2 Clause Translation

Both transition and error constraints must be translated into CNF for processing by the SAT solver. The simplest translation from general propositional formulas to CNF introduces a new auxiliary variable for each non-leaf node in the parse tree of the formula.

For instance, to translate the formula $A = B \vee C$ into CNF:

- Introduce auxiliary variables v_A, v_B, v_C; v_A is called the *root literal* of the expression
- Create the clauses for the ternary constraint $C_\vee(v_A, v_B, v_C)$ which is satisfied iff $v_A \Leftrightarrow v_B \vee v_C$; these are the A-*clauses* $CL(A)$.
- Recursively translate both B and C, with v_B and v_C as root literals respectively.

Finally, the root literal of the whole expression must be added as a unit clause.

If P is a propositional formula, and $CL(P)$ the set of clauses obtained by the above translation, then any assignment satisfying $CL(P)$ also satisfies P; but, not every satisfying assignment for P also satisfies $CL(P)$. However, every satisfying assignment to P *can* be extended to a satisfying assignment to $CL(P)$. To state the relationship more precisely, $P = \exists aux \cdot CL(P)$, where aux is the set of auxiliary variables introduced in the translation to CNF.

Since clauses are associated with nodes in the syntax tree of the original propositional formula P, it is always possible to determine, for a clause c, for which subformulas P' of P it is the case that $c \in CL(P')$.

We denote the set of clauses derived from path constraints as CL_K, and the set derived from error constraints as CL_e. For a timed propositional variable p_i, $CL(p_i)$ (or $CL_K(p_i), CL_e(p_i)$) denotes all clauses containing p_i; $CL(p) = \bigcup_i CL(p_i)$.

Consider the clause translation of $\kappa_2 = \neg b \wedge a \Rightarrow a'$ from the example. We start by creating auxiliary variables $v_{\kappa_2}, v_{\neg b \wedge a}$ (a' is already a variable, so no auxiliary is needed) and create the clauses for constraint $C_{\Rightarrow}(v_{\kappa_2}, v_{\neg b \wedge a}, a')$. These are:

$$\begin{gathered}(\neg v_{\kappa_2} \ \vee \ \neg v_{\neg b \wedge a} \ \vee \ \neg a') \\ (v_{\kappa_2} \ \vee \ v_{\neg b \wedge a}) \\ (v_{\kappa_2} \ a')\end{gathered}$$

The two subexpressions of $\neg b \wedge a$ are both literals, so no new auxiliary variables are needed; only the clauses for $C_{\wedge}(v_{\neg b \wedge a}, \neg b, a)$:

$$\begin{gathered}(\neg v_{\neg b \wedge a} \ \vee \ \neg b) \\ (\neg v_{\neg b \wedge a} \ \vee \ a) \\ (v_{\neg b \wedge a} \ \vee \ b \ \vee \ \neg a)\end{gathered}$$

3.3 Proofs and Unsat Core

DPLL-based SAT-solvers (such as ZCHAFF [10]) can not only decide unsatisfiability, but also generate a resolution proof of unsatisfiability [16, 15]. Given an (unsatisfiable) set A of clauses, and a resolution proof Π of A's unsatisfiability, $Roots(\Pi)$ is the *unsat core* with respect to Π, denoted UNSAT_CORE (or UNSAT_CORE (A)). UNSAT_CORE is the smallest set of clauses from A which is needed in the proof π; it is already unsatisfiable even without introducing other constraints from A.

The unsat core induces an abstraction K' of the model K such that: (a) K' does not admit a counterexample of length less than k, and (b) if K' satisfies a formula φ, then so does K.

For any R_p, only a subset of the cases κ_i^p need appear in the unsat core. They are still disjoint; there are now simply more states where p is free. Let R'_p be the conjunction of the remaining κ_i^p, and $R' = \bigwedge_p R'_p$. R' is an abstraction of R – it has fewer constraints. Similarly, only a subset $Init'$ of the initial-state clauses $CL_K(Init)$ need appear in the core; since this subset is a weaker constraint, it is an abstraction (an over-approximation of the original initial-state set). Since $BMC_k(K', \varphi) \supseteq$ UNSAT_CORE$(BMC_k(K, \varphi))$, and UNSAT_CORE is by definition unsatisfiable, K' has no length-k counterexample either, and $K' \models_k \varphi$.

Thus, SAT-based bounded model-checking can be seen as a two-step process. First, an abstraction K' of K is constructed; second, $K' \models_k \varphi$ is proven. Inability to produce such a K' implies that every abstraction of K has a length-k counterexample, which results in a satisfying assignment for the BMC problem.

The abstraction K' has two possible applications: it can be used for (unbounded) model-checking of the property [8, 9, 7]; or it can be used to provide diagnostic information, showing which path and error constraints were actually used and which were not, like the analysis of [14]. We explore the latter use.

4 Vacuity Detection

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. In this section, we give first the formal definition of vacuity; this definition makes no reference to any particular model-checking technique. Then, we attempt to define vacuity in terms of resolution proofs derived from successful runs of a SAT-based model-checker. This gives us three notions of increasing strength: irrelevance [14], local irrelevance, and peripherality. We show peripherality to be equivalent to vacuity restricted to propositional subformulas only, and give an algorithm to detect such vacuity using the resolution proof.

4.1 Vacuity of Models

The following definition formalizes the intuitive notion of vacuity.

Definition 4.1 *Let K be a Kripke structure, and φ a formula satisfied by K (i.e. $K \models \varphi$). Then, φ is p-vacuous in K iff $\varphi[p \leftarrow q]$ is satisfied by K for any formula q.*

We say that φ is vacuous if there exists a subformula p of φ such that φ is p-vacuous. Our definition of vacuity is syntactic, and in that it is similar to the original definition of Beer et al. [3, 2]. However, since we allow for variables that are free in the model to appear in the formula, Definition 4.1 is stronger, and is equivalent to the semantic definition of Armoni at al. [1]. In particular, φ is p-vacuous in K if and only if K satisfies $\varphi[p \leftarrow x]$, where x is a variable free in K and not already occurring in φ.

This definition is based only on the satisfaction relation between Kripke structures and temporal logic formulas, and not on any particular algorithm for computing that relation. Choosing to use a SAT-based model-checker, we now define properties of its automatically-generated resolution proofs which are related to vacuity.

4.2 Irrelevance and Local Irrelevance

Recall that, using the structure of the error-constraint translation, we are able to relate clauses in CL_e back to which sub-formula of the error they were derived from. This relationship allows us to reuse the definition of *irrelevance* from [14]: a subformula is irrelevant if there is an unsat core where none of the subformula's clauses are used.

Definition 4.2 *Let K be a model, φ an LTL formula, and $BMC(K, \varphi)$ be a CNF encoding of a bounded model-checking problem for $K \models \varphi$. Then, a subformula p of φ is* irrelevant *iff $\forall i \cdot CL_e(p_i) \cap \mathit{UNSAT\_CORE} = \emptyset$ or $\forall i \cdot CL_K(p_i) \cap \mathit{UNSAT\_CORE} = \emptyset$.*

That is, p is irrelevant in φ if all timed occurrences of p are local to either $CL_e \cap$ UNSAT_CORE or to $CL_K \cap$ UNSAT_CORE. It is easy to see that irrelevance is no stronger than vacuity. That is, if p is irrelevant in φ, then it can be replaced by an arbitrary propositional formula without affecting satisfiability of φ.

Theorem 4.1 *If p is irrelevant in φ, then φ is p-vacuous.*

Proof:
We need to show that if p is irrelevant in φ, then K satisfies $\varphi[p \leftarrow x]$, where x is a free variable of K that does not appear in φ. That is, we need to show that under the conditions of the Theorem the BMC problem $BMC(K, \varphi[p \leftarrow x]) = CL_K \cup CL_{e'}$ is unsatisfiable.

Since p is irrelevant in φ, we know that the BMC problem $BMC(K, \varphi) = CL_K \cup CL_E$ is unsatisfiable. Let UNSAT_CORE denote the UNSAT core of this problem, U_K be $CL_K \cap$ UNSAT_CORE, and U_e be $CL_e \cap$ UNSAT_CORE, respectively. Furthermore, since p is irrelevant in φ, either $\forall i \cdot U_e(p_i) = \emptyset$ or $\forall i \cdot U_K(p_i) = \emptyset$.

Assume that the former is the case, that is no clauses containing p_i appear in U_e. Then $U_e \subset U_{e'}$, which establishes that $CL_K \cup CL_{e'}$ is unsatisfiable.

Alternatively, assume that no clauses containing p_i appear in U_K. In this case, if p_i appears in the UNSAT core, it is local to U_e. Let x_i be a boolean variable corresponding to the variable x at time i, and $U_{e'}$ be the result of replacing each p_i by x_i in U_e. Since p_i is local to U_e, it follows that $U_K \cup U_{e'}$ is unsatisfiable. Moreover, $U_{e'} \subseteq CL_{e'}$, and therefore, $CL_K \cup CL_{e'}$ is unsatisfiable as well. □

The converse of Theorem 4.1 does not hold in general. Irrelevance is strictly weaker than vacuity. We show that there exists a Kripke structure K and a formula φ that is p-vacuous, but for any unsatisfiability core of the bounded model-checking problem, p is not irrelevant in φ.

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

$$\begin{array}{lcl} Init & = & p \\ R(AP, AP') & = & p \Rightarrow q' \end{array}$$

and an LTL formula $\varphi = X(p \vee q)$. Note that φ is p-vacuous, it is satisfied because q is true in any successor of the initial state. Assume that the CNF encoding of the one-step bounded model-checking problem is given by the following set of clauses:

$$\begin{array}{lcl} CL_K & = & (p_0)(\neg p_0 \vee q_1) \\ CL_e & = & (\neg p_1)(p_1 \vee \neg q_1) \end{array}$$

In this case, the unique minimal unsatisfiability core contains all of the clauses corresponding to the problem. Thus, p is not irrelevant according to Definition 4.2; the set of variables $\{p_0, p_1\}$ is not local CL_K or CL_e.

As illustrated by the example, the problem is that the definition of irrelevance treats all timed occurrences of an atomic proposition uniformly. A similar problem can be seen with the definition of irrelevant constraints in [14], and results in an over-approximation of the relevant constraints. In our case, a simple solution is to consider all timed variables corresponding to a single atomic proposition independently, as done in the following definition.

Definition 4.3 *Let K be a model, φ an LTL formula, and $BMC(K, \varphi)$ be a CNF encoding of a bounded model-checking problem for $K \models \varphi$. Then, p is* locally irrelevant *in φ iff for each time instance i, p_i is local to either $CL_e \cap$ UNSAT_CORE or $CL_p \cap$ UNSAT_CORE.*

$$\dfrac{(\neg r_0) \qquad (r_0 \vee p_0)}{(p_0)}\; r_0$$

$$\dfrac{(p_0) \qquad (\neg p_0 \vee q_0)}{(q_0)}\; p_0$$

$$\dfrac{(q_0) \qquad (\neg p_0 \vee \neg q_0)}{(\neg p_0)}\; q_0$$

$$\dfrac{(\neg p_0) \qquad (p_0)}{()}\; p_0$$

Figure 2: A resolution proof for EXAMPLE 2.

Irrelevance implies local irrelevance, but the converse does not hold in general as illustrated by Example 1. As with the previous definition, we show that local irrelevance implies vacuity.

Theorem 4.2 *If p is locally irrelevant in φ then φ is p-vacuous.*

Proof:
We need to show that is p is locally irrelevant in φ, then K satisfies $\varphi[p \leftarrow x]$, where x is a free variable of K not appearing in φ. That is, we need to show that the BMC problem $BMC(K, \varphi[p \leftarrow x]) = CL_K \cup CL_{e'}$ is unsatisfiable.

Since p is locally irrelevant, the BMC problem $BMC(K, \varphi) = CL_K \cup CL_e$ is unsatisfiable. Let UNSAT_CORE denote the UNSAT core of this problem, U_K be $CL_K \cap$ UNSAT_CORE, and U_e be $CL_e \cap$ UNSAT_CORE, respectively. Recall that for any i a variable p_i is either local to U_K or to U_e.

Let x_i denote a boolean variable corresponding to the value of the variable x at time i, and let $U_{e'}$ be the result of replacing each occurrence of p_i in U_e by x_i. Since each p_i that has been replaced is local to U_e, it follows that $U_K \cup U_{e'}$ is unsatisfiable. Since $U_{e'} \subseteq CL_{e'}$, the set of clauses $CL_K \cup CL_{e'}$ is unsatisfiable as well. $\square$

Note that in Example 1, p is locally irrelevant which establishes p-vacuity of φ. Local irrelevance is still not strong enough to capture vacuity. We illustrate this on the following example.

EXAMPLE 2. Consider a Kripke structure with atomic propositions r, p and q whose initial state is given by the constraint:

$$Init \quad = \quad \neg r \wedge p \wedge q$$

and a formula $\varphi = \neg p \vee q$. Note that φ is p-vacuous in the initial state. Let us assume that the zero-step bounded model-checking 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 2. In none of the proofs is p locally irrelevant with respect to CL_e and CL_K. We therefore need to refine our definition.

4.3 Peripherality

The problem with using local irrelevance to detect vacuity has again to do with treating different occurrences of variables uniformly. By examining the resolution proof in

Figure 2, we see that although p_0 is used both in the clauses corresponding to the model and the ones corresponding to the formula, it is always resolved "locally". That is, we can replace p_0 in either set of clauses without affecting their unsatisfiability. In the rest of this section, we formalize this intuition, and show that it leads to an even stronger definition.

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

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

Intuitively, a root clause cl is an element of $Source(l, c)$ if it contains a variable l and there exists a path from cl to c that does not contain a resolution on l.

Given an unsatisfiable set of clauses $C = A \cup B$ and a resolution proof Π establishing unsatisfiability of C, we say that a variable l is *peripheral* with respect to A and B iff for every resolution on l between clauses c_1 and c_2, $Source(l, c_1) \cup Source(l, c_2) \subseteq A$ or $Source(l, c_1) \cup Source(l, c_2) \subseteq B$. Intuitively, if l is global to $A \cup B$, but is peripheral, then it can be split into two local variables without affecting unsatisfiability. That is, the set of clauses $A_x \cup B_y$ is unsatisfiable, where A_x is the result of replacing l in A by a fresh variable x, and B_y is the result of replacing l in B by a fresh variable y.

Definition 4.4 *Let K be a model, φ an LTL formula, and $BMC(K, \varphi)$ be a CNF encoding of a bounded model-checking problem for $K \models \varphi$, and Π a proof of unsatisfiability of $BMC(K, \varphi)$ Then, 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.*

We now show that if an atomic proposition p is peripheral in φ, then φ is p-vacuous.

Theorem 4.3 *If p is peripheral in φ then φ is p-vacuous.*

Proof:
Since p is peripheral in φ, the BMC problem $BMC(K, \varphi) = CL_K \cup CL_e$ is unsatisfiable. Let $CL_{e'}$ be the result of replacing each occurrence of p_i by x_i in CL_e. Since each p_i is peripheral with respect to CL_K and CL_e, it follows that $CL_K \cup CL_{e'}$ is unsatisfiable. Since $CL_K \cup CL_{e'} = BMC(K, \varphi[p \leftarrow x])$, the conclusion of the theorem follows. □

Going back to Example 2, by examining the resolution proof in Figure 2, we see that although p is not locally irrelevant in φ, it is peripheral, and therefore, φ is p-vacuous.

It is easy to see that if p is locally irrelevant in φ, then it is also peripheral in φ. Thus the notion of being peripheral brings us closer to our desired characterization of vacuity. To show that this is the desired characterization, we prove, constructively, that p-vacuity implies the existence of a proof where p is peripheral; this proof consists of a vacuity-detection algorithm.

4.4 Algorithm

The next result shows that this is the strongest result that we can hope to establish. That is, if φ is p-vacuous, then there exists a resolution proof Π according to which p is peripheral in φ.

Theorem 4.4 *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 $BMC(K, \varphi[p \leftarrow x]) = CL_K \cup CL_{e'}$ is unsatisfiable, and there exists a resolution proof Π establishing this. We need to show that this proof can be transformed to a proof of unsatisfiability of $BMC(K, \varphi) = CL_K \cup CL_e$ in which each p_i is peripheral with respect to CL_K and CL_e.

Note that if Π does not contain a clause that contains both p_i and x_i for some i, then a 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 $BMC(K, \varphi)$ 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 $U_{e'} = Roots(\Pi) \cap CL_{e'}$. Then, if p_i occurs in any clause of U_K, it is local to U_K. Let $C = \exists local(U_K) \cdot U_K$ be a formula corresponding to existential elimination of all variables local to U_K, and let $CNF(C)$ be its CNF encoding. Note that 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 $Leaf(\Pi_c) = c$ 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. □

Detecting whether a given variable is peripheral or not in a given resolution proof Π can be done in time linear in the size of Π.

Theorem 4.5 *Given an unsatisfiable set of clauses $C = A \cup B$, and a resolution proof Π, detecting whether a given variable p is peripheral with respect to A and B can be done in time linear in Π.*

Proof:
Consider a function L that for each clause c in Π, assigns a label from the set {'A', 'B', 'AB', '$\emptyset$'}, defined recursively as follows:

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

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

It is easy to see that a variable p is peripheral iff $Leaf(\Pi)$ is not labeled by 'AB', that is $L(Leaf(\Pi)) \neq$ 'AB'.

The labeling function L can be computed by breadth-first traversal of the proof Π, which is linear in the size of the proof. □

The algorithms presented in the proof of Theorem 4.5 can easily be extended to keep track of several variables at once. In this case, its running time is linear in the number of variables as well as the size of the proof.

Thus, to detect whether a formula φ is vacuous in an atomic proposition p, we need to execute the above algorithm and keep track of all occurrences of p_i. If every occurrence is peripheral, then φ is p-vacuous. Furthermore, a single pass of the algorithm is sufficient to detect all peripheral atomic propositions of φ, potentially identifying several causes of vacuity.

5 Conclusion and Future Work

In this report, we have considered the use of SAT-solver-generated proofs of unsatisfiability in the generation of diagnostic information from a successful model-checking run. In particular, we have found a property of resolution proofs that is equivalent to vacuity in a propositional variable, and given an algorithm for checking it. We note that though this technique is not quite as complete as existing techniques (it is only as good as a particular proof of unsatisfiability), it can be applied with much lower cost, after a model-checking run completes.

There are, as stated previously, two uses for the abstraction of a model induced by the unsat core: irrelevance and vacuity checking, and more efficient unbounded model-checking. We have concentrated on the previous problem. The other problem has been investigated by McMillan et al. [8, 9]; it is a fairly new avenue of investigation, so there are still many possibilities for future work there. In particular, the existing techniques rely on other methods besides SAT solving (BDDs and interpolant computation); we hope to find a fully SAT-based method which avoids the cost of translation to and from other representations.

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 the 15th Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *LNCS*, pages 368–380, Colorado, USA, July 2003. Springer-Verlag.

[2] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in ACTL Formulas". In *Proceedings of the 9th International Conference on Computer-Aided Verification (CAV'97)*, volume 1254 of *LNCS*. Springer-Verlag, 1997.

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

[4] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri. "NuSMV: a new Symbolic Model Verifier". In N. Halbwachs and D. Peled, editors, *Proceedings of 11th Conference on Computer-Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 495–499, Trento, Italy, July 1999. Springer.

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

[6] O. Kupferman and M. Vardi. "Vacuity Detection in Temporal Model Checking". In *Proceedings of 10th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'99)*, volume 1703 of *LNCS*, pages 82–96. Springer-Verlag, 1999.

[7] B. Li, C. Wang, and F. Somenzi. "A Satisfiability-based Approach to Abstraction Refinement in Model Checking". In *Proceedings of First International Workshop on Bounded Model Checking (BMC'03)*, volume 89 of *Electronic Notes in Theoretical Computer Science*, Boulder, Colorado, July 2003.

[8] K.L. McMillan. "Interpolation and SAT-based Model Checking". In *Proceedings of Conference on Computer-Aided Verification (CAV '03)*, 2003.

[9] K.L. McMillan and N. Amla. "Automatic Abstraction without Counterexamples". In *Proceedings of 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'03)*, volume 2619 of *Lecture Notes in Computer Science*, pages 2–17, April 2003.

[10] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. "Chaff: Engineering an Efficient SAT Solver". In *Proceedings of 39th Design Automation Conference*, June 2001.

[11] D. Peled, A. Pnueli, and L. Zuck. From falsification to verification. In *FST&TCS*, volume 2245 of *LNCS*. Springer-Verlag, 2001.

[12] D. Peled and L. Zuck. From model checking to a temporal proof. In *Proceedings of the 8th International SPIN Workshop (SPIN'2001)*, volume 2057 of *LNCS*, pages 1–14, Toronto, Canada, May 2001. Springer.

[13] M. Purandare and F. Somenzi. "Vacuum Cleaning CTL Formulae". In *Proceedings of 14th Conference on Computer Aided Verification(CAV'02)*, volume 2404 of *LNCS*, pages 485–499, Copenhagen, Denmark, July 2002. Springer-Verlag.

[14] I. Shlyakhter, R. Seater, D. Jackson, M. Sridharan, and M. Taghdiri. "Debugging Overconstrained Models Using Unsatisfiable Cores". In *Proceedings of 18th IEEE International Conference on Automated Software Engineering*, Montréal, Québec, Canada, October 2003.

[15] L. Zhang and S. Malik. "Extracting Small Unsatisfiable Cores from Unsatisfiable Boolean Formulas". In *Proceedings of 6th International Conference on Theory and Applications of Satisfiability Testing*, S. Margherita Ligure - Portofino, Italy, May 2003.

[16] L. Zhang and S. Malik. "Validating SAT Solvers Using an Independent Resolution-Based Checker: Practical Implementations and Other Applications". In *Proceedings of Design, Automation and Test in Europe (DATE2003)*, Munich, Germany, March 2003. IEEE Computer Society.