An Efficiently Checkable, Proof-Based Formulation of Vacuity in Model Checking

Kedar S. Namjoshi

Bell Labs, Lucent Technologies
`kedar@research.bell-labs.com`

Abstract. Model checking algorithms can report a property as being true for reasons that may be considered vacuous. Current algorithms for detecting vacuity require either checking a quadratic size witness formula, or multiple model checking runs; either alternative may be quite expensive in practice. Vacuity is, in its essence, a problem with the justification used by the model checker for deeming the property to be true. We argue that current definitions of vacuity are too broad from this perspective and give a new, narrower, formulation. The new formulation leads to a simple detection method that examines only the justification extracted from the model checker in the form of an automatically generated proof. This check requires a small amount of computation after a *single* verification run on the property, so it is significantly more efficient than the earlier methods. While the new formulation is stronger, and so reports vacuity less often, we show that it agrees with the current formulations for linear temporal properties expressed as automata. Differences arise with inherently branching properties but in instances where the vacuity reported with current formulations is debatable.

1 Introduction

The problem of detecting a vacuous model check of a property has received much attention in the literature [2, 3, 17, 24, 6, 1]. A vacuous model check often indicates a problem, either with the precise formulation of an informal correctness property, or with the program itself. A classic example is that of the property: "Every request is eventually granted", which is true vacuously of a program in which no request is ever made! Most model checkers produce independently checkable evidence for a negative answer in the form of a counterexample trace, but typically are not designed to produce any such evidence of their reasoning for success. Without further checking, therefore, one may end up trusting in vacuously justified properties.

Several algorithms for vacuity detection have been proposed in the papers above. Essentially, these algorithms look for a sub-formula of the correctness property whose truth value does not matter for property verification (a precondition is that the property has been verified). In [3], vacuity is defined as follows: a program M satisfies a formula ϕ vacuously iff M satisfies ϕ, and there is some subformula ψ of ϕ such that ψ does not affect the satisfaction of ϕ – this

R. Alur and D.A. Peled (Eds.): CAV 2004, LNCS 3114, pp. 57–69, 2004.

last condition holds if the formula obtained by replacing ψ with any formula ξ (this is written as $\phi[\psi := \xi]$) is true of M. The detection method is to check an automatically generated witness formula for each subformula ψ, one that demonstrates redundancy of ψ if it is true – the witness formula for the property above is "No request is ever made". In [17], this test is simplified and generalized to all of CTL* from the fragment of ACTL considered in [3]. It is shown that to determine whether ψ affects ϕ, it suffices to check whether $\phi[\psi := \mathit{false}]$ ($\phi[\psi := \mathit{true}]$) holds of M if ψ occurs with positive (negative) polarity in ϕ (i.e., under an even (odd) number of negations). These methods treat multiple occurrences of the same subformula independently. A recent paper [1] extends the method above to take into account such dependencies. We adopt their term, *formula vacuity*, to refer to the definition that treats subformulas independently.

A major drawback of these algorithms is that they either require multiple model checking runs to test each subformula, or a single run checking a quadratically long witness formula. (The witnesses are of linear length for the w-ACTL fragment of [3].) Although the cost of model checking increases only linearly in the length of the formula, even a linear blowup can be quite significant in terms of the resources used (time, space, processors), since it is common for large verification runs to take hours to run on a single formula. Remedies have been proposed: in [1], formula structure is used to reduce the number of witness checks, and [24, 6] show how to share intermediate results while checking a single witness formula.

A vacuous verification is, in its essence, a problem with the justification used by the model checking algorithm for deeming the property to be true. Recent work has shown how to extract and present such this justification in the form of a deductive proof (see [21, 22] and the references therein). The central question this paper discusses is whether one can analyze *this justification alone* in order to detect a vacuous verification? The premise seems reasonable, but there are subtleties involved in making it work, and in determining the precise relationship to formula vacuity.

To make this relationship clearer, we examine formula vacuity from the viewpoint of justifications. We show that a mu-calculus formula ϕ is formula-vacuous if, and only if, there exists a valid correctness proof showing that ϕ is true of M in which the invariant for some subformula contains only unreachable states or is empty. Call such a proof *vacuous*. Formula vacuity, then, is equivalent to asking the question: "Does there exist a vacuous correctness proof?". Viewed from this angle, the criterion appears rather odd: why should one discard a valid, non-vacuous correctness proof, just because there is an alternative proof that is vacuous? Examples given later in the paper support this conclusion, by showing that formula vacuity can sometimes produce debatable reports of vacuity.

We propose a new, stronger, formulation of vacuity, which we call *proof vacuity*. This consists of checking whether the correctness proof produced by the model checker as justification is vacuous. Since our vacuity criterion is stronger than formula vacuity, one may expect the new check to report vacuity less often.

However, we show that this may happen only for properties that are inherently branching in nature.

Besides correcting some anomalies in the formula vacuity criterion, the new formulation is, moreover, significantly more efficient to check in practice. All that is required is to examine for emptiness the invariants produced during a model checking run. As these invariants are readily available, this process is even simpler and requires less resources than the generation of a full proof of correctness. As the two formulations coincide for linear time automaton properties, this also gives a significantly more efficient way of checking formula vacuity for such properties.

The next section introduces the concept of proof used in new formulation. In Section 3, we characterize formula vacuity in terms of proofs, motivate and present proof vacuity, and show their equivalence for automaton representations of linear time properties. Section 4 discusses related work, and some future research directions, including "partial vacuity" detection.

2 Preliminaries

We assume that properties are specified as alternating tree automata. In this section, we define the automaton syntax and semantics, and also give the proof system used to represent the justification of model checking results.

Transition Systems. We represent a program by the *transition system* (TS, for short) that it generates – this is also called a *Kripke Structure*. A TS is a tuple $(S, \hat{S}, R, L)$, where S is a non-empty set of *states*, $\hat{S} \subseteq S$ is the set of *initial states*, $R \subseteq S \times S$ is a *transition relation*, and $L : S \rightarrow 2^{AP}$ (AP is a set of *atomic propositions*) is a *labeling function* that maps each state to the the propositions true at that state. We assume that R is *total*: i.e., for every s, there exists t such that $(s, t) \in R$.

Temporal Logics. While our results are based on alternating tree automata, we also consider properties defined in one of several well-known temporal logics. The syntax of these logics is defined below; please see [8] for the precise semantics.

LTL [23], linear temporal logic, is a logic that defines infinite sequences over subsets of AP. In positive normal form, formulas of LTL are given by the grammar: $\Phi ::= P(P \in AP) \mid \neg P(P \in AP) \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathsf{X}(\Phi) \mid (\Phi \mathsf{U} \Phi) \mid (\Phi \mathsf{W} \Phi)$. The temporal operators are X (next-time), U (until), and W (weak-until or unless). Other operators are defined through abbreviation: $\mathsf{F}(\Phi)$ (eventually Φ) is $(\mathit{true} \mathsf{U} \Phi)$, and $\mathsf{G}(\Phi)$ (always Φ) is $(\Phi \mathsf{W} \mathit{false})$.

CTL* [7], a branching time logic, is given by adding path quantifiers A (over all paths) and E (over some path) to LTL; the set of state formulas are defined inductively as: $\Phi ::= P(P \in AP) \mid \neg P(P \in AP) \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathsf{A}(\psi) \mid \mathsf{E}(\psi)$. Here, ψ is the set of path formulas, which are LTL formulas where atomic propositions come from Φ. CTL [5] is obtained from CTL* by restricting the set of path formulas so that they contain a single, top-level, temporal operator. Its sub-logic ACTL (ECTL) is obtained by allowing only the A (E) path quantifier; ACTL* (ECTL*) is the analogous restriction of CTL*.

The mu-calculus [16] is a branching time temporal logic which subsumes the logics defined above [10]. Formulas in positive form are defined using the following grammar, where V is a set of symbols denoting fixpoint variables, and μ (ν) is the least (greatest) fixpoint operator: $\Phi ::= P(P \in AP) \mid Z(Z \in V) \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathsf{EX}(\Phi) \mid \mathsf{AX}(\Phi) \mid (\mu Z : \Phi) \mid (\nu Z : \Phi)$. A formula must have each variable under the scope of an even number of negation symbols; it is *closed* iff every variable in it is under the scope of a fixpoint operator. The evaluation of a formula f with free variables is a subset of S, defined inductively relative to a context that maps free variables to subsets of S. A state s in M *satisfies* a closed mu-calculus formula f iff s is in the evaluation of f relative to the context that maps every variable to the empty set; M satisfies f (written as $M \models f$) iff all of its initial states satisfy f.

Alternating Tree Automata. An alternating automaton over AP is specified by a tuple $(Q, \hat{q}, \delta, F)$, where Q is a non-empty set of states, $\hat{q} \in Q$ is the initial state, δ is a transition function, and F is a parity acceptance condition. F is given by a partition $(F_0, F_1, \ldots, F_{2N})$ of Q. An infinite sequence over Q satisfies F iff the smallest index i for which a state in F_i occurs infinitely often on the sequence is even. We use a simple normal form for the transition relation: it maps an automaton state and an input from 2^{AP} to one of $(\wedge R), (\vee R), \mathsf{EX}r, \mathsf{AX}r, \mathit{true}, \mathit{false}$, where r is a state, and R a non-empty subset of states.

Given a mu-calculus formula, one can obtain an equivalent automaton whose transition relation is just the parse graph of the formula (so that each subformula is represented by an automaton state), and the parity condition is determined by an analysis of the alternation between least and greatest fixpoints in the formula [9].

These automata accept computation trees of programs, where tree nodes are labeled with subsets of atomic propositions. For a labeled tree t, let λ_t be the labeling function that maps tree nodes to their labels. The acceptance of a labeled tree t by the automaton is defined in terms of a two-player infinite game. A *configuration* of the game is a pair (x, q), where x is a tree node and q is an automaton state. The moves of the game are as shown in Figure 1.

A *play* of the game is a maximal sequence of configurations generated in this manner. A play is a win for player I iff either it is finite and ends in a configuration that is a win for I, or it is infinite and satisfies the automaton

At configuration (x, q), based on the form of $\delta(q, \lambda_t(x))$,

- *true*: Player I wins, and the play is halted
- *false*: Player II wins, and the play is halted
- $(\vee R)$: Player I picks $r \in R$; the next configuration is (x, r)
- $(\wedge R)$: Player II picks $r \in R$; the next configuration is (x, r)
- $\mathsf{EX}r$: Player I picks a child y of x; the next configuration is (y, r)
- $\mathsf{AX}r$: Player II picks a child y of x; the next configuration is (y, r)

Fig. 1. Model Checking Game Moves

acceptance condition. The play is winning for player II otherwise. A *strategy* for player I (II) is a partial function that maps every finite sequence of configurations to a choice at each player I (II) position. A strategy is a *win* for player I if every play following that strategy is a win for I, regardless of the strategy for II. The automaton *accepts* the tree t iff player I has a winning strategy for the game starting at $(\epsilon, \hat{q})$, where ϵ is the root node of the tree t. M *satisfies* A (written as $M \models A$) iff the automaton accepts the computation trees obtained from each initial state by "unwinding" the transition relation of M.

The Proof System. The proof system presented in [21] is as follows. For a TS $M = (S, \hat{S}, R, L)$ and automaton $A = (Q, \hat{q}, \delta, F)$, a proof (ϕ, ρ, W) of A over M is given by specifying (i) [Invariants] for each automaton state q, an invariant predicate ϕ_q (i.e., a subset of S), and (ii) [Rank Functions] for each automaton state q, a partial rank function, ρ_q, from S to a well-founded set $(W, \preceq)$, with induced rank relation $\lhd_q$ over $W \times W$. For vacuity checking, the precise nature of the rank functions and relations is not important[1]. The invariants and rank function must satisfy the three local conditions given in Figure 2 for the proof to be valid. In these conditions, the variable w ranges over elements of W, the notation $[f]$ means that f is valid, and a term of the form $\rho_r \lhd_q w$ $(\rho_r = w)$ represents the predicate $(\lambda s : \rho_r(s) \lhd_q w)$ $((\lambda s : \rho_r(s) = w))$.

- ***Consistency:*** (ρ_q is defined for every state in ϕ_q) For each $q \in Q$, $[\phi_q \;\Rightarrow\; (\exists w : (\rho_q = w))]$
- ***Initiality:*** (the initial states satisfy the initial invariant) $[\hat{S} \;\Rightarrow\; \phi_{\hat{q}}]$
- ***Invariance and Progress:*** For each $q \in Q$, depending on the form of $\delta(q, l)$, where l is a propositional formula over AP, check the following.
  - *true*: nothing to check.
  - *false*: $[\phi_q \;\Rightarrow\; \neg l]$
  - $(\vee R)$: $[\phi_q \wedge l \wedge (\rho_q = w) \;\Rightarrow\; (\vee r : r \in R : \phi_r \wedge (\rho_r \lhd_q w))]$
  - $(\wedge R)$: $[\phi_q \wedge l \wedge (\rho_q = w) \;\Rightarrow\; (\wedge r : r \in R : \phi_r \wedge (\rho_r \lhd_q w))]$
  - $\mathsf{EX}r$: $[\phi_q \wedge l \wedge (\rho_q = w) \;\Rightarrow\; \mathsf{EX}(\phi_r \wedge (\rho_r \lhd_q w))]$
  - $\mathsf{AX}r$: $[\phi_q \wedge l \wedge (\rho_q = w) \;\Rightarrow\; \mathsf{AX}(\phi_r \wedge (\rho_r \lhd_q w))]$

Fig. 2. The Proof System Rules

Theorem 0 *([21]) (**Soundness**) If there is a valid proof of A over M, then $M \models A$. (**Completeness**) If $M \models A$, there is a valid proof for A over M.*

The proof of the completeness theorem is important to vacuity detection, so we sketch it here. This proof considers the "run tree" formed by a winning strategy in the game, where the tree contains a single player I move at a node

---

[1] For F of size $2N + 1$, W is the product of N well-founded sets $(W_i, \preceq_i)$, and $\preceq$ is the lexicographic order induced by $\prec_0 \ldots \prec_N$. For $a, b \in W$, $a \lhd_q b$ holds if, and only if, for the unique k such that q is in F_k, either $k = 2i$, for some i, and $(a_1, \ldots, a_i) \preceq (b_1, \ldots, b_i)$, or $k = 2i - 1$, for some i, and $(a_1, \ldots, a_i) \prec (b_1, \ldots, b_i)$.

labeled by a configuration for player I, and all player II moves at a node labeled for player II. The choices of a move for player I are resolved uniquely by the winning strategy. One can extract a proof from this run tree. In particular, the invariant ϕ_q for q is the set of program states s for which there is a configuration of the form (x, q) in the run tree where x contains s.

3 Defining and Detecting Vacuity

We assume that all formulas are in positive normal form. Let Φ be a closed mu-calculus formula. For a subformula ψ of Φ, we define the strict positive subformulas of ψ to be those subformulas of ψ that are not fixpoint variables from a scope outside that of ψ. For instance, in the fairness property $(\nu Z : (\mu Y : \mathsf{EX}(P \wedge Z) \vee \mathsf{EX}(Y)))$ ("There exists a path where P holds infinitely often"), Z is not a strict subformula of $\mathsf{EX}(P \wedge Z)$, although $(P \wedge Z)$ is one.

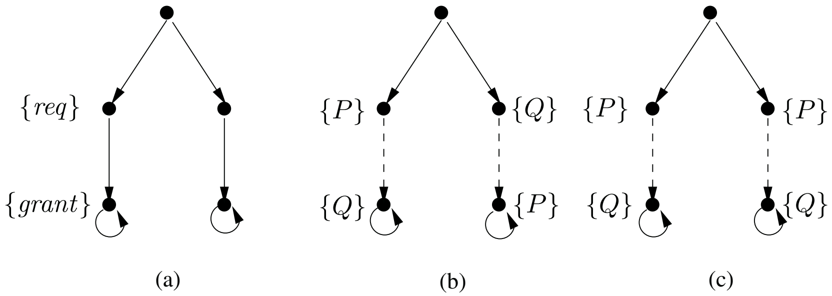
Theorem 1 *Let Φ be a closed mu-calculus formula, and ψ be a strict positive subformula. Let $\mathcal{A}_\Phi$ be the automaton formed from the parse graph of Φ, with state q corresponding to ψ. Then, $\Phi[\psi := \mathit{false}]$ holds of M iff there is a proof that M satisfies $\mathcal{A}_\Phi$ in which the invariant for q is the empty set.*

Proof. (left-to-right) Let $\Phi' = \Phi[\psi := \mathit{false}]$. The parse graph for Φ' can be obtained from that of Φ by deleting nodes that correspond to strict subformulas of ψ, and replacing ψ with *false*. Thus, an automaton for Φ', $\mathcal{A}_{\Phi'}$, can be obtained from $\mathcal{A}_\Phi$ by deleting states that correspond to strict sub-formulas of ψ, and by setting $\delta'(q, l) = \mathit{false}$ for all l. As Φ' holds for M, by the completeness theorem, there is a valid deductive proof (ϕ', ρ', W') which, of course, is based only on the states in $\mathcal{A}_{\Phi'}$. As this proof is valid, the assertion $[\phi'_q \Rightarrow \neg l]$ must be valid for all l; thus, $\phi'_q = \emptyset$. Now we form a proof (ϕ, ρ, W) as follows: let $W = W'$, and for r in $\mathcal{A}_{\Phi'}$, let $\phi_r = \phi'_r$ and $\rho_r = \rho'_r$. For any other state p, let $\phi_p = \emptyset$, and let ρ_p be some fixed element of W (the choice is arbitrary).

We claim that this is a valid proof for $\mathcal{A}_\Phi$ over M. Since each state p missing from $\mathcal{A}_{\Phi'}$ represents a strict subformula of ψ, it is reachable only from q; hence, setting its invariant to the empty set does not affect the validity of the checks for states of $\mathcal{A}_{\Phi'}$. Furthermore, since the invariants are set to empty, the own checks for these states, which have the form $[(\phi_p \wedge \alpha) \Rightarrow \beta]$, are trivially valid. Hence, the new proof is valid, and it has an empty invariance set for q.

(right-to-left) Suppose that there is a proof (ϕ, ρ, W), where $\phi_q = \emptyset$. Then, the rule for q holds trivially. Consider a state p that corresponds to a strict subformula of ψ. As p is reachable only from q, its invariant does not affect the validity of the check for any state r that does not correspond to a strict subformula of ψ. Hence, dropping states that correspond to strict subformulas of ψ, and replacing $\delta(q, l)$ with *false* for all l, yields a valid proof of $\mathcal{A}_{\Phi'}$ over M. By the soundness theorem, this implies that M satisfies Φ'. $\Box$

Given this theorem, the existing methods for vacuity detection can be viewed as searching for a vacuous proof. This seems to be too broad a formulation: it disregards a non-vacuous proof produced by a model checker simply because



(a) (b) (c)

Fig. 3. Transition Systems

there exists an alternate, vacuous justification for the property. To illustrate this point further, we consider some examples.

Existential Properties. The existential analogue of the example defined in the introduction is "There exists a computation where every request is eventually granted". This may be formulated in ECTL as $\mathsf{EG}(req \Rightarrow \mathsf{EF}(grant))$. For the TS shown in Figure 3(a), the property is true, as on the left-hand path, a request is sent and subsequently granted. This witness does make use of every subformula, so is non-vacuous. However, replacing the subformula $\mathsf{EF}(grant)$ with *false*, we obtain the property $\mathsf{EG}(\neg req)$, which is true of the right-hand branch; thus, the verification is considered vacuous by the formula-vacuity criterion. This is clearly odd: a perfectly reasonable, non-vacuous proof is disregarded because of the existence of an alternative, vacuous proof.

Universal, Non-linear Properties. Consider another property, in this case a universal one, but one that is not expressible in linear-time logics: "At every successor state, either inevitably P holds, or inevitably Q holds", where P and Q are atomic propositions. The property can be written in ACTL as $\mathsf{AX}(\mathsf{AF}(P) \vee \mathsf{AF}(Q))$. Consider the TS shown in Figure 3(b), with the dashed arrow representing a long path to a state satisfying the target proposition. The property is clearly true of the TS. The shortest proof will favor satisfying $\mathsf{AF}(P)$ for the left successor of the initial state, since it can be done in fewer steps than satisfying $\mathsf{AF}(Q)$, and it will favor satisfying $\mathsf{AF}(Q)$ for the other successor, for the same reason. This is a non-vacuous proof, since all subformulas are used. However, replacing the subformula $\mathsf{AF}(Q)$ by *false* results in the property $\mathsf{AX}(\mathsf{AF}(P))$, which is also true of the TS; hence, the verification is vacuous by formula vacuity. This conclusion is debatable: again, a perfectly reasonable proof (in fact, the shortest one) is disregarded because of a vacuous proof that is non-optimal.

3.1 Proof Vacuity

Given these anomalies resulting from the formula-vacuity criterion, we seek to define a stronger criterion based on the justification actually given by the model checker. One possible approach is to examine only the justification provided by a given model checker for vacuity. This is simple, but has the disadvantage of being sensitive to the choice of the model checker. This sensitivity may not be a problem if a particular model checker is standardized for use within an organization.

We explore, however, the consequences of a definition that is insensitive to the model checking strategy. As a preliminary, given M and $\mathcal{A}_{\Phi}$, for a mu-calculus formula Φ, call a state of M *game-reachable* iff it occurs in some configuration of the game for $M \models \mathcal{A}_{\Phi}$. Note that all unreachable states of M are also game-unreachable, but that some reachable states of M may also be game-unreachable. For instance, the game for the formula $\Phi = P$ examines only the initial states of M; thus, all other states are game-unreachable. The proof of the soundness theorem in [21] shows that each valid proof defines a history-free winning strategy for the model checking game. Thus, removing game-unreachable states from each invariant set does not change the validity of a proof. We can now define a proof to be vacuous for some subformula as follows.

Definition 0 ***(Vacuous Proof)*** *A proof for an automaton property is vacuous for automaton state q iff the invariant for q is empty or contains only game-unreachable states. A proof is vacuous iff it is vacuous for some automaton state.*

Theorem 1 can then be amended so that it is also possible for the invariant for q to contain game-unreachable states. We rely on this amended form in the subsequent discussion. We can now define proof vacuity as follows. Notice that the crucial difference with formula vacuity is that a verification is considered to be vacuous only if all *all* model checking strategies give rise to vacuous proofs.

Definition 1 ***(Proof Vacuity)*** *For an automaton property $\mathcal{A}$ and a TS M, the verification of $\mathcal{A}$ on M is proof-vacuous iff for some automaton state q, every proof that M satisfies $\mathcal{A}$ is vacuous for q.*

The rationale for excluding game-unreachable, rather than just unreachable states is as follows. Consider the property $\mathsf{AX}(P \Rightarrow \mathsf{AX}(Q))$. This is true vacuously of a TS M for which all successors of the initial state satisfy $\neg P$. However, there may be a proof where the invariant for the state corresponding to $\mathsf{AX}(Q)$ includes reachable states of M that satisfy $\mathsf{AX}(Q)$; thus, the weaker criterion would not properly detect the vacuous verification. Game-reachability is not that stringent a criterion; for instance, COSPAN [12], a symbolic model checker for linear-time automaton properties, restricts the model checking to the reachable states of the product of the TS and a negated property automaton. This product defines the game configurations: thus, all invariants include only game-reachable states.

Although proof-vacuity has a universal quantification over all proofs, we can show that it suffices to consider a maximally inclusive proof, produced by including every game reachable state that satisfies an automaton state q in ϕ_q – this is just the proof provided by the completeness theorem.

Theorem 2 *For an automaton property $\mathcal{A}$ and TS M, the property is true proof-vacuously for M iff the maximally inclusive proof is vacuous.*

Proof. It is slightly easier to show the contrapositive. Suppose that the property is true non-proof-vacuously. Then there is a non-vacuous proof, one in which the invariant for every automaton state contains some game-reachable state. This state must be present, by inclusiveness, in the invariant for that automaton state in the maximally inclusive proof. Hence, the maximally inclusive proof is non-vacuous. In the other direction, if the maximally inclusive proof is non-vacuous, the verification is non-proof-vacuous by definition. □

Vacuity Checking. Suppose that the states of M are represented symbolically by a vector of Boolean variables $\vec{b}$, and that the states of $\mathcal{A}_\Phi$ are represented by a vector of Boolean variables $\vec{c}$. Let $Win(\vec{b}, \vec{c})$ be the set of reachable, winning game configurations that is generated by the model checker, as indicated above for COSPAN. The vacuity check then amounts to detecting whether the encoding of some valid automaton state q is not associated with any state in Win – this means that the invariant for q is empty. Such a check can be performed through symbolic (BDD) calculations. One such method is to calculate the set of automaton states with non-empty invariants, $nonempty(\vec{c})$, as $(\exists \vec{b} : Win(\vec{b}, \vec{c}))$, and check whether $valid \Rightarrow nonempty$, where $valid(\vec{c})$ defines valid automaton states.

An interesting point is that proof-vacuity, as defined, does not take into account the distinction between optimal and non-optimal proofs. For instance, consider the ACTL formula $\mathsf{AX}(\mathsf{AF}(P) \vee \mathsf{AF}(Q))$ and the TS in Figure 3(c). An optimal proof, in which player I makes choices that satisfy eventualities as early as possible, has an empty invariant for $\mathsf{AF}(Q)$; a less-than-optimal proof would include the third-level states (which are game-reachable) in the invariant for $\mathsf{AF}(Q)$. Thus, the verification is non-proof-vacuous due to the existence of the less-than-optimal, non-vacuous proof. However, it might be of interest to know that the shortest proof of correctness does not use a certain subformula. The construction for the completeness theorem can be adjusted to create a proof that is optimal (vs. maximally inclusive) as follows. The construction already results in ranks that define the shortest path to satisfying eventualites. All that remains is to prune the maximally inclusive invariants so that states that do not contribute to the shortest path to fulfilling any eventuality (e.g., the third-level states in this example) are eliminated from the invariants. It is not clear, though, whether an analogue of Theorem 2 can be obtained. Note that a discussion of optimality differences is based on quantifying progress through rank functions; progress concerns have so far not played a role in the identification of vacuous justifications. As shown below, differences in optimality are of concern only for certain branching-time properties.

Theorem 3 *For a mu-calculus property ϕ and TS M, if there is a unique run tree for the model checking game on M and $\mathcal{A}_\phi$, then formula-vacuity for ϕ coincides with proof-vacuity (optimal-proof-vacuity) for $\mathcal{A}_\phi$.*

Proof. Each run tree corresponds to a winning strategy for player I, which is history-free (due to the parity winning condition) and so corresponds to a proof (cf. [21]). As there is a unique run tree, there is only a single valid proof. If ϕ is true formula-vacuously, this proof is vacuous, and hence the verification is also proof-vacuous, since there are no other valid proofs to consider. The other direction follows as proof-vacuity is stronger than formula-vacuity. □

This theorem has some interesting consequences for linear and branching time properties. Consider the case where a linear-time property is specified by expressing its negation as a a Büchi word automaton – this is the usual approach to model checking a linear-time property. From such an automaton B, one may construct a *universal* dual automaton $\tilde{B}$ with the same transition relation and a complemented Büchi acceptance condition [20]. Universality means that $\tilde{B}$ accepts a computation if *all* runs on it are accepting. This dual automaton can also be viewed as a universal tree automaton for the branching time property "All computations of M are accepted by $\tilde{B}$". As $\tilde{B}$ is universal, there is a single run tree of $\tilde{B}$ on M. Applying Theorem 3, one obtains the consequence given below. For pure branching time properties, proof-vacuity is strictly stronger, as shown by the examples discussed previously.

Theorem 4 *Let $\tilde{B}$ be a universal co-Büchi tree automaton that expresses that a linear-time correctness property holds for all computations of M. If $M \models \tilde{B}$, the verification is formula vacuous iff it is proof (optimal-proof) vacuous.*

Linear Properties as LTL Formulas. Theorem 4 showed that, for linear-time properties expressed as (negated) automata, proof vacuity coincides with formula vacuity. The situation is different if formula vacuity is applied to LTL formulas. Intuitively, this is because model checking is carried out with an automaton derived from an LTL formula (cf. [11]), where each automaton state represents a *set* of subformulas. Thus, vacuity at the level of the automaton states makes possible finer distinctions than vacuity at the level of formulas.

This difference can be seen with an LTL formula that is similar – though not equivalent – to the ACTL formula defined previously: $\mathsf{X}(\mathsf{F}(P) \vee \mathsf{F}(Q))$. This holds for the TS in Figure 3(b); however, so does $\mathsf{X}(\mathsf{F}(P))$, which is obtained by setting the $\mathsf{F}(Q)$ subformula to *false*. A universal co-Büchi automaton for this formula is defined by the state set $\{0, 1, 2\}$ with initial state 0, and transitions $0 \xrightarrow{true} 1, 1 \xrightarrow{(\neg P \wedge \neg Q)} 1, 1 \xrightarrow{(P \vee Q)} 2$, and $2 \xrightarrow{true} 2$, and the co-Büchi acceptance condition $\mathsf{FG}(2)$. This automaton is verified without vacuity (either by the old or, from Theorem 4, by the new formulation).

We can show equivalence of the formula and automaton vacuity definitions, however, for a subclass of LTL formulas. This class, LTL^{det}, is defined in [19], where it is shown that it represents exactly those LTL properties that are definable in ACTL. In fact, the ACTL property equivalent to checking that ϕ holds on all computations can be obtained simply by attaching the A path operator to each temporal operator in ϕ; we refer to this ACTL formula as ϕ^+. Any ACTL property has a direct translation to an equivalent mu-calculus property, so we may consider ϕ^+ also as a mu-calculus property. The precise syntax is given below. In this syntax, p is a propositional state predicate, so that every $\vee$-choice can be resolved deterministically at a state.

$\Phi ::= p \mid \Phi \wedge \Phi \mid (p \wedge \Phi) \vee (\neg p \wedge \Phi) \mid \mathsf{X}(\Phi) \mid ((p \wedge \Phi)\mathsf{U}(\neg p \wedge \Phi)) \mid ((p \wedge \Phi)\mathsf{W}(\neg p \wedge \Phi))$.

Theorem 5 *For an LTL^{det} formula ϕ and TS M, the verification of ϕ on M is formula-vacuous iff the verification of $\phi+$ on M is proof-vacuous.*

Proof. Consider any positive sub-formula ψ of ϕ. Then, $\mathsf{A}(\phi[\psi := false])$ is equivalent to $(\phi[\psi := false])^+$, which is identical to $\phi^+[\psi^+ := \mathit{false}]$ by the nature of the transformation. Thus, ϕ is true formula-vacuously for M with ψ as the witness subformula iff ϕ^+ is true formula-vacuously for M, with ψ^+ as the witness subformula. From the determinism inherent in the syntax, there is a unique run tree for the verification of ϕ^+. By Theorem 3, formula-vacuity for ϕ^+ coincides with proof-vacuity. □

4 Conclusions and Related Work

This paper argues that the current formulation of vacuity is too broad. This conclusion is supported both by examples, and by a theorem showing that formula-vacuity is equivalent to the existence of a vacuous proof. We propose a new, narrower formulation, which needs to inspect only the invariants of a single, maximally inclusive proof. Besides resolving anomalies with the earlier formulation, the new one can be checked more efficiently, by using information that is easily gathered during model checking. This should translate to a significant advantage in practice. In fact, checking proof vacuity with COSPAN is trivial, since the model checker produces a list of game-unreachable automaton states; the verification is vacuous iff this list contains a valid automaton state.

The most closely related work in vacuity detection has been discussed throughout the paper. As pointed out in [18], vacuity is but one of several approaches to further inspecting a valid answer from a model checker. One may also examine the state "coverage" (cf. [14, 13, 4]). Vacuity may be viewed, dually, as a form of specification coverage, since it checks whether certain sub-formulas are redundant for the verification. More generally, producing justifications in the form of deductive proofs (cf. [21, 22]) or as interactive games or tableaux [15, 26, 25] can offer deeper insight into why a property holds (or fails) of the program, since these take ranking functions also into account (cf. the discussion on optimality).

An interesting research topic is to recognize "partial" vacuity in a way that produces useful results. For instance, consider a tree-structured transition system where the left subtree of the root satisfies the request-grant property vacuously (by never sending requests), while the right subtree satisfies it non-vacuously (there is at least one request). Neither the earlier formulation nor the one proposed here would consider this to be a vacuous verification, yet there is clearly something odd in this proof. It seems likely that a more detailed analysis of correctness proofs will enable the detection of such instances.

Acknowledgements

This work was supported in part by grant CCR-0341658 from the National Science Foundation.

References

1. R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M.Y. Vardi. Efficient vacuity detection in linear temporal logic. In *CAV*, volume 2725 of *LNCS*. Springer-Verlag, 2003.
2. D. Beatty and R. Bryant. Formally verifying a microprocessor using a simulation methodology. In *31st DAC*. IEEE Computer Society, 1994.
3. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. In *CAV*, volume 1254 of *LNCS*, 1997. (full version in FMSD, 18(2), 2001).
4. H. Chockler, O. Kupferman, R.P. Kurshan, and M.Y. Vardi. A practical approach to coverage in model checking. In *CAV*, volume 2102 of *LNCS*. Springer-Verlag, 2001.
5. E.M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logics of Programs*, volume 131 of *LNCS*. Springer-Verlag, 1981.
6. Y. Dong, B. Sarna-Starosta, C.R. Ramakrishnan, and S.A. Smolka. Vacuity checking in the modal mu-calculus. In *AMAST*, volume 2422 of *LNCS*. Springer-Verlag, 2002.
7. E. A. Emerson and J. Y. Halpern. “Sometimes” and “Not Never” revisited: on Branching versus Linear Time Temporal Logic. *J.ACM*, 33(1):151–178, January 1986.
8. E.A. Emerson. Temporal and modal logic. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B: Formal Methods and Semantics*. Elsevier and MIT Press, 1990.
9. E.A. Emerson and C.S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *FOCS*, 1991.
10. E.A. Emerson and C-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *LICS*, 1986.
11. R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV*. Chapman & Hall, 1995.
12. R.H. Hardin, Z. Har’el, and R.P. Kurshan. COSPAN. In *CAV*, volume 1102 of *LNCS*, 1996.
13. Y. Hoskote, T. Kam, P-H. Ho, and X. Zhao. Coverage estimation for symbolic model checking. In *37th DAC*. ACM Press, 1999.
14. S. Katz, D. Geist, and O. Grumberg. “Have I written enough properties?” A method for comparison between specification and implementation. In *CHARME*, volume 1703 of *LNCS*. Springer-Verlag, 1999.
15. A. Kick. Generation of witnesses for global mu-calculus model checking. available at http://citeseer.ist.psu.edu/kick95generation.html, 1995.
16. D. Kozen. Results on the propositional mu-calculus. In *ICALP*, volume 140 of *LNCS*. Springer-Verlag, 1982.
17. O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. In *CHARME*, number 1703 in LNCS. Springer-Verlag, 1999. (full version in STTT 4(2), 2003).

18. O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2), 2003.
19. M. Maidl. The common fragment of CTL and LTL. In *FOCS*, 2000.
20. Z. Manna and A. Pnueli. Specification and verification of concurrent programs by ∀-automata. In *POPL*, 1987.
21. K. S. Namjoshi. Certifying model checkers. In *CAV*, volume 2102 of *LNCS*, 2001.
22. D. Peled, A. Pnueli, and L. D. Zuck. From falsification to verification. In *FSTTCS*, volume 2245 of *LNCS*, 2001.
23. A. Pnueli. The temporal logic of programs. In *FOCS*, 1977.
24. M. Purandare and F. Somenzi. Vacuum cleaning CTL formulae. In *CAV*, volume 2404 of *LNCS*. Springer-Verlag, 2002.
25. P. Stevens and C. Stirling. Practical model-checking using games. In *TACAS*, volume 1384 of *LNCS*. Springer-Verlag, 1998.
26. S. Yu and Z. Luo. Implementing a model checker for LEGO. In *FME*, volume 1313 of *LNCS*, 1997.