Enhanced Vacuity Detection in Linear Temporal Logic

Roy Armoni[*] Limor Fix[*] Alon Flaisher[†*] Orna Grumberg[†] Nir Piterman[‡*]
Andreas Tiemeyer[*] Moshe Y. Vardi[§]

January 29, 2003

Abstract

One of the advantages of temporal-logic model-checking tools is their ability to accompany a negative answer to a correctness query with a counterexample to the satisfaction of the specification in the system. On the other hand, when the answer to the correctness query is positive, most model-checking tools provide no witness for the satisfaction of the specification. In the last few years there has been growing awareness of the importance of suspecting the system or the specification of containing an error also in cases where model checking succeeds. In particular, several works have recently focused on the detection of the *vacuous satisfaction* of temporal logic specifications. For example, when verifying a system with respect to the specification $\varphi = G(req \rightarrow F grant)$ ("every request is eventually followed by a grant"), we say that φ is satisfied vacuously in systems in which requests are never sent. Current works have focused on detecting vacuity with respect to subformula occurrences. In this work we investigate vacuity detection with respect to subformulas with multiple occurrences.

The generality of our framework requires us to re-examine the basic intuition underlying the concept of vacuity, which until now has been defined as sensitivity with respect to syntactic perturbation. We study sensitivity with respect to semantic perturbation, which we model by universal propositional quantification. We show that this yields a hierarchy of vacuity notions. We argue that the right notion is that of vacuity defined with respect to traces. We then provide an algorithm for vacuity detection and discuss pragmatic aspects.

[*]Intel Design Center, Haifa. Email: (roy.armony,limor.fi x,alon.flaisher,andreas.tiemeyer)@intel.com
[†]Technion, Israel Institute of Technology. Email: orna@cs.technion.ac.il
[‡]Weizmann Institute. Email: nirp@wisdom.weizmann.ac.il
[§]Rice University. Email: vardi@cs.rice.edu

1 Introduction

Temporal logics, which are modal logics geared towards the description of the temporal ordering of events, have been adopted as a powerful tool for specifying and verifying concurrent systems [Pnu77]. One of the most significant developments in this area is the discovery of algorithmic methods for verifying temporal-logic properties of *finite-state* systems [CE81, CES86, LP85, QS81, VW86]. This derives its significance both from the fact that many synchronization and communication protocols can be modeled as finite-state systems, as well as from the great ease of use of fully algorithmic methods. In temporal-logic *model checking,* we verify the correctness of a finite-state system with respect to a desired behavior by checking whether a labeled state-transition graph that models the system satisfies a temporal logic formula that specifies this behavior (for an in-depth survey, see [CGP99]).

Beyond being fully-automatic, an additional attraction of model-checking tools is their ability to accompany a negative answer to the correctness query with a counterexample to the satisfaction of the specification in the system. Thus, together with a negative answer, the model checker returns some erroneous execution of the system. These counterexamples are very important and can be essential in detecting subtle errors in complex designs [CGMZ95]. On the other hand, when the answer to the correctness query is positive, most model-checking tools provide no witness for the satisfaction of the specification in the system. Since a positive answer means that the system is correct with respect to the specification, this may, a priori, seem like a reasonable policy. In the last few years, however, industrial practitioners have become increasingly aware of the importance of checking the validity of a positive result of model checking. The main justification for suspecting the validity of a positive result is the possibility of errors in the modeling of the system or of the desired behavior, i.e., the specification.

Early work on "suspecting a positive answer" concerns the fact that temporal logic formulas can suffer from *antecedent failure* [BB94]. For example, in verifying a system with respect to the CTL specification $\varphi = AG(req \rightarrow AF grant)$ ("every request is eventually followed by a grant"), one should distinguish between *vacuous satisfaction* of φ, which is immediate in systems in which requests are never sent, and non-vacuous satisfaction, in systems where requests are sometimes sent. Evidently, vacuous satisfaction suggests some unexpected properties of the system, namely the absence of behaviors in which the antecedent of φ is satisfied.

Several years of practical experience in formal verification have convinced the verification group at the IBM Haifa Research Laboratory that vacuity is a serious problem [BBER97]. To quote from [BBER97]: "Our experience has shown that typically 20% of specifications pass vacuously during the first formal-verification runs of a new hardware design, and that vacuous passes always point to a real problem in either the design or its specification or environment." The usefulness of vacuity analysis is also demonstrated via several case studies in [PS02]. Often, it is possible to detect vacuity easily by checking the system with respect to hand-written formulas that ensure the satisfaction of the preconditions in the specification [BB94, PP95]. To the best of our knowledge, this rather unsystematic approach is the prevailing one in the industry for dealing with vacuous satisfaction. For example, the FormalCheck tool [Kur98] uses "sanity checks", which include a search for triggering conditions that are never enabled.

These observations led Beer et al. to develop a method for automatic testing of vacuity [BBER97]. Vacuity is defined as follows: a formula φ is satisfied in a system M vacuously if it is satisfied in M, but some subformula ψ of φ does not *affect* φ in M, which means that M also satisfies $\varphi[\psi \leftarrow \psi']$ for all formulas ψ' ($\varphi[\psi \leftarrow \psi']$ denotes the result of substituting ψ' for ψ in φ). Beer et al. proposed testing vacuity by means of *witness formulas*. Formally, we say that a formula φ' is a *witness formula* for the specification φ if a system M satisfies φ non-vacuously iff M satisfies both φ and φ'. In the example above, it is not hard to see that a system satisfies φ non-vacuously iff it also satisfies $EF req$. In general, however, the generation of witness formulas is not trivial, especially when we are interested in other types of vacuity passes, which are more complex than antecedent failure. While [BBER97] nicely set the basis for a methodology for detecting vacuity in temporal-logic specifications, the particular method described in [BBER97] is quite limited (see also [BBER01]).

A general method for detection of vacuity for specifications in CTL* (and hence also LTL, which was not handled by [BBER97]) was presented in [KV99, KV03]. The key idea there is a general method for generating witness formulas. It is shown in [KV03] that instead of replacing a subformula ψ by all subformulas ψ', it suffices to replace it by either $\mathbf{true}$ or $\mathbf{false}$ depending on whether ψ occurs in φ with negative polarity (i.e., under an odd number of negations) or positive polarity (i.e., under an even number of negations). Thus, vacuity checking amounts to model checking witness formulas with respect to all (or some) of the subformulas of the specification φ. It is important to note that the method in [KV03] is for vacuity with respect to subformula occurrences. The key feature of occurrences is that a subformula occurrence has a *pure* polarity (exclusively negative or positive). In fact, it is shown in [KV03] that the method is not applicable to subformulas with mixed polarity (both negative and positive occurrences).

Recent experience with industrial-strength property-specification languages such as ForSpec [AFF$^+$02] suggests that the restriction to subformula occurrences of pure polarity is not negligible. ForSpec, which is a linear-time language, is significantly richer syntactically (and semantically) than LTL. In particular, it has a rich set of arithmetical and Boolean operators. As a result, even subformula occurrences may not have pure polarity, e.g., in the formulas $p \oplus q$ ($\oplus$ denotes exclusive or). While we can rewrite $p \oplus q$ as $(p \wedge \neg q) \vee (\neg p \wedge q)$, it forces the user to think of every subformula occurrence of mixed polarity as two distinct occurrences, which is rather unnatural. Also, a subformula may occur in the specification multiple times, so it need not have a pure polarity even if each occurrence has a pure polarity. For example, if the LTL formula $G(p \rightarrow p)$ holds in a system M then we'd expect it to hold vacuously with respect to the subformula p (which has a mixed polarity), though not necessarily with respect to either occurrence of p, because both formulas $G(\mathbf{true} \rightarrow p)$ and $G(p \rightarrow \mathbf{false})$ may fail in M. (Surely, the fact that $G(\mathbf{true} \rightarrow \mathbf{false})$ fails in M should not entail that $G(p \rightarrow p)$ holds in M non-vacuously.) Our goal in this paper is to remove the restriction in [KV03] to subformula occurrences of pure polarity. To keep things simple, we stick to LTL and consider vacuity with respect to subformulas, rather than with respect to subformula occurrences. We comment on the extension of our framework to ForSpec at the end of the paper.

The generality of our framework requires us to re-examine the basic intuition underlying the concept of vacuity, which is that a formula φ is satisfied in a system M vacuously if it is satisfied in M but some subformula ψ of φ does not *affect* φ in M. It is less clear, however, what does "does not affect" means. Intuitively, it means that we can "perturb" ψ without affecting the truth of φ in M. Both [BBER97] and [KV03] consider only syntactic perturbation, but no justification is offered for this decision. We argue that another notion to consider is that of semantic perturbation, where the *truth value* of ψ in M is perturbed arbitrarily. The first part of the paper is an examination in depth of this approach. We model arbitrary semantic perturbation by a universal quantifier, which in turn is open to two interpretations (cf. [Kup95]). It turns out that we get two notions of "does not affect" (and therefore also of vacuity), depending on whether universal quantification is interpreted with respect to the system M or with respect to its computation tree. We refer to these two semantics as "structure semantics" and "trace semantics". Surprisingly, the original, syntactic, notion of perturbation falls between the two semantic notions.

We argue then that trace semantics is the preferred one for vacuity checking. Structure semantics is simply too weak, yielding vacuity too easily. Formula semantics is more discriminating, but it is not robust, depending too much on the syntax of the language. In addition, these two semantics yield notions of vacuity that are computationally intractable. In contrast, trace semantics is not only intuitive and robust, but it can be checked easily by a model checker.

In the final part of the paper we address several pragmatic aspects of vacuity checking. We first discuss whether vacuity should be checked with respect to subformulas of subformula occurrences and argue that both checks are necessary. We then discuss how the number of vacuity checks can be minimized. We also discuss how vacuity results should be reported to the user. Finally, we describe our experience of implementing vacuity checking in the context of a ForSpec-based model checker.

2 Preliminaries

2.1 Temporal Logic

The logic LTL is a linear temporal logic. Formulas of LTL are built from a set AP of atomic propositions using the usual Boolean operators and the temporal operators X ("next time") and U ("until"). Given a set AP , an LTL formula is defined as follows:

- $\mathbf{true}$, $\mathbf{false}$, p for $p \in AP$.
- $\neg\psi$, $\psi \wedge \varphi$, $X\psi$, or $\psi U \varphi$, where ψ and φ are LTL formulas.

We define satisfaction of LTL formulas with respect to computations of Kripke structures. A Kripke structure is $M = \langle AP, S, S_0, R, L\rangle$ where AP is the set of atomic propositions, S is a set of states, S_0 is a set of initial states, $R \subseteq S \times S$ is a total transition relation, and $L : AP \to 2^S$ assigns to each atomic proposition the set of states in which it holds. A *computation* of a system is a sequence of states $\pi = s_0, s_1, \ldots$ such that $s_0 \in S_0$ and for every $i \geq 0$ we have that $(s_i, s_{i+1}) \in R$. The set $\mathcal{T}(M)$ is the set of computations of a Kripke structure M. We denote the suffix $s_j, s_{j+1}, \ldots$ of π by π^j.

We define the semantics of LTL with respect to a computation $\pi = s_0, s_1, \ldots$ and a location $i \geq 0$. We denote $M, \pi, i \models \varphi$ to indicate that the LTL formula φ holds in the computation π at location i. A computation π satisfies an LTL formula φ, denoted $\pi \models \varphi$ if $\pi, 0 \models \varphi$. The Kripke structure M satisfies φ, denoted $M \models \varphi$ if for every computation $\pi \in \mathcal{T}(M)$ we have $\pi \models \varphi$. For a full definition of the semantics of LTL we refer the reader to [Eme90].

An occurrence of formula ψ of φ is of *positive polarity* in φ if it is in the scope of an even number of negations, and of *negative polarity* otherwise. The polarity of a subformula is defined by the polarity of its occurrences as follows. Formula ψ is of *positive polarity* if all occurrences of ψ in φ are of positive polarity, of *negative polarity* if all occurrences of ψ in φ are of negative polarity, of *pure polarity* if it is either of positive or negative polarity, and of *mixed polarity* if some occurrences of ψ in φ are of positive polarity and some are of negative polarity.

Given a formula φ and a subformula of pure polarity ψ we denote by $\varphi\,[\psi \leftarrow \bot]$ the formula obtained from φ by replacing ψ by $\mathbf{true}$ if ψ is of negative polarity and by $\mathbf{false}$ if ψ is of positive polarity. Dually, $\varphi\,[\psi \leftarrow \top]$ denotes the formula obtained from φ by replacing ψ by $\mathbf{false}$ if ψ is of negative polarity and by $\mathbf{true}$ if ψ is of positive polarity.

2.2 UQLTL

The logic UQLTL augments LTL with universal quantification over *propositional variables*. Let X be a set of propositional variables. The syntax of LTL is extended as follows. If φ is an LTL formula over the extended set of atomic propositions $AP \cup X$, then $\forall X \varphi$ is a UQLTL formula. E.g., $\forall x\, G\,(x \to p)$ is a legal UQLTL formula, while $G\,\forall x\,(x \to p)$ is not. UQLTL is a subset of *Quantified Propositional Temporal Logic* [SVW85], where all the free variables are quantified universally. In the sequel, we use x to denote a propositional variable. A *closed* formula is a formula with no free propositional variables.

We now give definitions of two semantics for UQLTL formulas. The first is *structure semantics* where a propositional variable is bound to a subset of the states of the Kripke structure. The second is *trace semantics* where a propositional variable is bound to a subset of the locations on the trace.

Let M be a Kripke structure with a set of states S, let $\pi \in \mathcal{T}(M)$, and let X be a set of propositional variables. A *structure assignment* $\sigma : X \to 2^S$ maps every propositional variable $x \in X$ to a set of states in S. We use s_i to denote the ith state along π, and φ to denote UQLTL formulas.

Definition 2.1 (UQLTL Structure Semantics) *The relation $\models_s$ is defined inductively as follows:*

- $M, \pi, i, \sigma \models_s x$ *iff* $s_i \in \sigma(x)$.
- $M, \pi, i, \sigma \models_s \forall x \varphi$ *iff* $M, \pi, i, \sigma[x \leftarrow S'] \models_s \varphi$ *for every* $S' \subseteq S$.
- *For other formulas,* $M, \pi, i, \sigma \models_s$ *is defined as in LTL.*

A closed UQLTL formula φ is *structure satisfied* at point i of trace $\pi \in \mathcal{T}(M)$, denoted $M, \pi, i \models_s \varphi$, iff $M, \pi, i, \sigma \models_s \psi$ for some σ (choice is not relevant since φ is closed). A closed UQLTL formula φ is structure satisfied in structure M, denoted $M \models_s \varphi$, iff $M, \pi, 0 \models_s \varphi$ for every trace $\pi \in \mathcal{T}(M)$.

We now define the trace semantics for UQLTL. Let X be a set of propositional variables. A *trace assignment* $\alpha : X \to 2^{\mathbf{N}}$ maps a propositional variable $x \in X$ to a set of natural numbers (points on a path).

Definition 2.2 (UQLTL Trace Semantics) *The relation $\models_t$ is defined inductively as follows:*

- $M, \pi, i, \alpha \models_t x$ *iff* $i \in \alpha(x)$.
- $M, \pi, i, \alpha \models_t \forall x \varphi$ *iff* $M, \pi, i, \alpha[x \leftarrow N'] \models_t \varphi$ *for every* $N' \subseteq \mathbf{N}$.
- *For other formulas,* $M, \pi, i, \sigma \models_s$ *is defined as in LTL.*

A closed UQLTL formula φ is *trace satisfied* at point i of trace $\pi \in \mathcal{T}(M)$, denoted $M, \pi, i \models_t \varphi$, iff $M, \pi, i, \alpha \models \psi$ for some α (choice is not relevant since φ is closed). A closed UQLTL formula φ is trace satisfied in structure M, denoted $M \models_t \varphi$, iff $M, \pi, 0 \models_t \varphi$ for every trace $\pi \in \mathcal{T}(M)$.

We show that trace semantics is stronger than structure semantics in the following sense. Whenever a UQLTL formula holds according to trace semantics it holds according to structure semantics. The opposite is not true. Indeed, a trace assignment can assign a variable different values when the computation visits the same state of M. We observe that for LTL formulas both semantics are identical. That is, if φ is an LTL formula, then $M \models_s \varphi$ iff $M \models_t \varphi$. We sometimes use $\models$ to denote the satisfaction of an LTL formula, rather than $\models_s$ or $\models_t$.

Theorem 2.3 *Given a structure M and a UQLTL formula φ:*

- $M \models_t \varphi \quad \Rightarrow \quad M \models_s \varphi$
- $M \models_s \varphi \quad \not\Rightarrow \quad M \models_t \varphi$

The proof resembles the proofs in [Kup95] for the dual logic EQCTL. Kupferman shows that a structure might not satisfy a formula, although the formula is satisfied by its computation tree. The full proof is given in Appendix A.

3 Alternative Definitions of Vacuity

Let ψ be a subformula of φ. We give three alternative definitions of when ψ *does not affect* φ, and compare them. We refer to the definition of [BBER97] as *formula vacuity*. We give two new definitions, *trace vacuity* and *structure vacuity*, according to trace and formula semantics. Notice that we are only interested in the cases where φ is satisfied in the structure.

Intuitively, ψ does not affect φ in M if we can perturb ψ without affecting the truth of φ in M. In previous work, syntactic perturbation was allowed. Using UQLTL we formalize the concept of semantic perturbation. Instead of changing ψ syntactically, we directly change the set of points in a structure or on a trace in which it holds.

Definition 3.1 (Does Not Affect) *Let φ be a formula satisfied in M and let ψ be a subformula of φ.*

- ψ does not affect$_f$ φ *in M iff for every LTL formula ξ we have* $M \models \varphi\,[\psi \leftarrow \xi]$ [BBER97].
- ψ does not affect$_s$ φ *in M iff* $M \models_s \forall x \varphi\,[\psi \leftarrow x]$.
- ψ does not affect$_t$ φ *in M iff* $M \models_t \forall x \varphi\,[\psi \leftarrow x]$.

We say that ψ *affects$_f$* φ in M iff it is not the case that ψ does not affect$_f$ φ in M. We say that φ is *formula vacuous* in M, if there exists a subformula ψ such that ψ does not affect$_f$ φ. We define *affects$_s$*, *affects$_t$*, *structure vacuity* and *trace vacuity* similarly.

3.1 Comparing the Alternative Definitions of Vacuity

In the following section we compare the three alternative definitions of vacuity. We show that they are all different. We also argue why trace vacuity is the preferred definition. Notice that we do not restrict a subformula to occur once and it can be of mixed polarity. We show that our three semantics form a hierarchy, with structure semantics being the weakest and trace semantics the strongest.

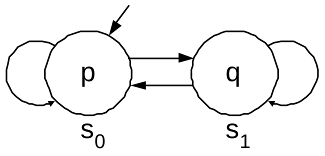
We show that structure vacuity is weaker than formula vacuity. That is, ψ might affect$_f$ φ in M, but not affect$_s$ φ in M.

Lemma 3.2 (Relating Structure and Formula Vacuity) *Let φ be an LTL formula. If ψ does not affect$_f$ φ in M, then it does not affect$_s$ φ in M as well. The reverse implication does not hold.*

In the following proof, we assume that every state has a *unique representation* using the atomic propositions. That is, every state in the structure satisfies a different set of atomic propositions. This is a reasonable assumption for hardware modeling.

Proof: First we prove that if a subformula ψ does not affect$_f$ φ in M, then it does not affect$_s$ φ in M as well. If ψ affects$_s$ φ, then there exists a structure assignment σ and a computation π of M such that $M, \pi, 0, \sigma \not\models_s \varphi\,[\psi \leftarrow x]$. We construct a formula ψ' that behaves like x along π, that is, $M, \pi, i \models \psi'$ iff $M, \pi, i, \sigma \models_s x$. Let $\overline{s}$ be a predicate over AP that is $\mathbf{true}$ only in state $s \in S$. Let ψ' be the disjunction of $\overline{s}$ for all states in $\sigma(x)$. The formula ψ' is well-defined since S is finite. We show that $M, \pi, i, \sigma \models_s x$ iff $M, \pi, i \models \psi'$. If $M, \pi, i, \sigma \models_s x$ then, by the definition of structure semantics, $s_i \in \sigma(x)$. Therefore $\overline{s}_i$ is in the disjunction ψ'. Since $M, \pi, i \models \overline{s}_i$, we have $M, \pi, i \models \psi'$. On the other hand, if $M, \pi, i, \sigma \not\models_s x$ then $s_i \notin \sigma(x)$, and therefore $\overline{s}_i$ is not in the disjunction ψ'. Since every state is uniquely labeled $M, \pi, i \models \overline{s}_j$ iff $s_j = s_i$. Consequently $M, \pi, i \not\models \psi'$. Thus we have shown that if $M, \sigma \not\models_s \varphi\,[\psi \leftarrow x]$ then $M \not\models \varphi\,[\psi \leftarrow \psi']$.

In the other direction, we construct an LTL formula ψ' that assumes different values when visiting the same state. Let M be the following Kripke structure:



Let $\varphi = p \vee (G\,F\,q) \vee (F\,G\neg q)$. We examine if p affects$_f$ φ. Consider the path $\pi' = s_0, s_1, s_0, s_0, s_0 \ldots$ and let $\psi' = G\neg q$. Thus, $\varphi\,[p \leftarrow \psi'] = (G\neg q) \vee (G\,F\,q) \vee (F\,G\,F\,q)$. Clearly, $\pi' \not\models \varphi\,[p \leftarrow \psi']$ and p affects$_f$ φ.

On the other hand, for every trace π and every assignment σ, we have $M, \pi, 0, \sigma \models_s \varphi\,[\psi \leftarrow x]$. That is, $M, \pi, 0, \sigma \models_s x \vee (G\,F\,q) \vee (F\,G\neg x)$. If $s_0 \in \sigma(x)$ then the the disjunct x is satisfied. If $s_0 \notin \sigma(x)$ then for all traces that from some point on remain in s_0 $F\,G\neg x$ is satisfied, for all other paths, $G\,F\,q$ is satisfied. $\square$

We now show that formula vacuity is weaker than trace vacuity. That is, ψ might affect$_t$ φ in M, but not affect$_f$ φ in M.

Lemma 3.3 (Relating Trace and Formula) *Let φ be an LTL formula. If ψ does not affect$_t$ φ in M, then ψ does not affect$_f$ φ in M as well. The reverse implication does not hold.*

Proof: We show that if ψ affects$_f$ φ, then it also affects$_t$ φ. If ψ affects$_f$ φ, then there exists a formula ψ' such that $M \not\models \varphi[\psi \leftarrow \psi']$. Let π be a trace in M such that $\pi \not\models \varphi\,[\psi \leftarrow \psi']$. Consider the assignment $\alpha(x) = \{i |\, \pi, i \models \psi'\}$. Clearly, $M, \pi, 0, \alpha \not\models_t \varphi\,[\psi \leftarrow x]$, and therefore ψ affects$_t$ φ.

In the other direction, let M be a Kripke structure with a single state labeled by p, with a self-loop. Let $\varphi = (p \rightarrow Xp)$. It can be shown that $M \not\models_t \forall x\, \varphi\,[p \leftarrow x]$, thus p affects$_t$ φ. We now show that there cannot exist an LTL formula ψ' such that $M \not\models \varphi\,[p \leftarrow \psi']$. Notice that M has a single trace π, and that $tail(\pi) = \pi$. This means that ψ' is either true along every suffix of π, or π is false along every suffix of π. However $M \not\models \varphi\,[p \leftarrow \psi']$ only if ψ' holds at time zero but fails at time one. $\square$

Which is the most appropriate definition for practical applications? We show that structure vacuity and formula vacuity are sensitive to changes in the design that do not relate to the formula. As an example, consider the formula $\varphi = p \to Xp$ and models M_1 and M_2 in the figure below. In M_2 we add a proposition q whose behavior is independent of p's behavior. We would not like formulas that relate to p to change their truth value or their vacuity. Both M_1 and its extension M_2 satisfy φ and φ relates only to the proposition p. While p does not affect$_f$ φ in M_1, it does affect$_f$ φ in M_2 (and similarly for affects$_s$). Indeed, the formula $\varphi\,[p \leftarrow q] = q \to Xq$ does not hold in M_2. Note that in both models p affects$_t$ φ.

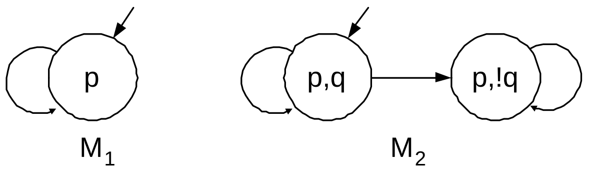
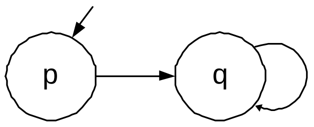


Figure 1: Changes in the design.

Another disadvantage of formula vacuity is that it is *sensitive to the specification language*. That is, a formula passing vacuously might pass unvacuously once the specification language is extended. As an example, consider the following Kripke structure M_1 and the LTL formula $\varphi = Xq \to XXq$.



For the single trace $\pi \in \mathcal{T}(M_1)$, it holds that $tail(\pi^1) = \pi^1$. Thus, every (future) LTL formula is either true along every suffix of π^1, or is false along every such suffix. This implies that subformula q does not affect$_f$ φ. However, we get an opposite result if the specification language is LTL augmented with the PAST operator [LPZ85]. The PAST operator enables reference to the history of the computation. Formally, if ψ is an LTL formula then $M, \pi, i \models PAST(\psi)$ iff $i > 0$ and $M, \pi, i-1 \models \psi$. Clearly, for every model M we have $M, \pi, 0 \not\models PAST(p)$. In the example, $M_1 \not\models \varphi\,[q \leftarrow PAST(p)]$ since $M_1, \pi, i \models PAST(p)$ iff $i = 1$, thus q affects$_f \varphi$.

To summarize, trace vacuity is preferable since is not sensitive to changes in the design (as opposed to structure and formula vacuity) and it is independent of the specification language (as opposed to formula vacuity). Another reasoning for the superiority of trace vacuity is given in section 4.

3.2 Comparing the Alternative Defi nitions of Vacuity under Pure Polarity

In the following section we show that if subformulas are restricted to pure polarity, all the definitions of vacuity coincide. For that, we show that the replacement of subformula ψ by $\perp$ is adequate for vacuity detection according to all three definitions. This result is an extension of the results in [KV03], where only single occurrence has been considered.

Lemma 3.4 *For every structure M, LTL formula φ and subformula ψ of φ of pure polarity, $M \models_t \varphi\,[\psi \leftarrow \perp]$ iff $M \models_t \forall x \varphi\,[\psi \leftarrow x]$.*

The lemma follows from the claim below. Let θ denote a subformula of φ that may or may not contain the subformula ψ.

Claim 3.5 *For every occurrence of θ, every trace $\pi \in \mathcal{T}(M)$ and location i,*

- *if θ is of positive polarity in φ then $M, \pi, i \models \theta\,[\psi \leftarrow \perp]$ implies $M, \pi, i \models_t \forall x\, \theta\,[\psi \leftarrow x]$*
- *if θ is of negative polarity in φ then $M, \pi, i \not\models \theta\,[\psi \leftarrow \perp]$ implies $M, \pi, i \models_t \forall x \neg\theta\,[\psi \leftarrow x]$*

Proof: We prove the claim by induction on the structure of θ. We prove the case that ψ is of positive polarity (i.e. in $\theta\,[\psi \leftarrow \bot]$ the subformula ψ is replaced by **false**). The case of negative polarity is dual. If ψ is not a subformula of θ the claim follows. Assume that ψ is a subformula of θ.

Let $\theta = p$ for some proposition p. Clearly, also $\psi = p$ and the claim follows. Let $\theta = \theta_1 \wedge \theta_2$. Suppose that the polarity of θ in φ is positive. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then clearly $M, \pi, i \models \theta_1\,[\psi \leftarrow \mathbf{false}]$ and $M, \pi, i \models \theta_2\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \theta_1\,[\psi \leftarrow x]$ and $M, \pi, i \models_t \forall x \theta_2\,[\psi \leftarrow x]$. Clearly, the claim follows. Suppose that the polarity of θ in φ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then either $M, \pi, i \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$ or $M, \pi, i \not\models \theta_2\,[\psi \leftarrow \mathbf{false}]$. Wlog suppose $M, \pi, i \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$. It follows that $M, \pi, i \models_t \forall x \neg\theta\,[\psi \leftarrow x]$.

Let $\theta = \theta_1 \vee \theta_2$. Suppose that the polarity of θ in φ is positive. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then either $M, \pi, i \models \theta_1\,[\psi \leftarrow \mathbf{false}]$ or $M, \pi, i \models \theta_2\,[\psi \leftarrow \mathbf{false}]$. Wlog suppose $M, \pi, i \models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \theta_1\,[\psi \leftarrow x]$. Clearly, the claim follows. Suppose that the polarity of θ in φ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then both $M, \pi, i \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$ and $M, \pi, i \not\models \theta_2\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$ and $M, \pi, i \models_t \forall x \neg\theta_2\,[\psi \leftarrow x]$. It follows that $M, \pi, i \models_t \forall x \neg\theta\,[\psi \leftarrow x]$.

Let $\theta = \neg\theta_1$. Suppose that the polarity of θ in φ is positive. Then the polarity of θ_1 in φ is negative. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then $M, \pi, i \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$. However, $\neg\theta_1\,[\psi \leftarrow x] \equiv \theta\,[\psi \leftarrow x]$ and the claim follows. Suppose that the polarity of θ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then $M, \pi, i \models \theta_1\,[\psi \leftarrow \mathbf{false}]$ and from the induction assumption we know that $M, \pi, i \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$. The claim follows.

Let $\theta = X\theta_1$. Suppose that the polarity of θ is positive. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then $M, \pi, i + 1 \models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i + 1 \models_t \forall x \theta_1\,[\psi \leftarrow x]$. The claim follows. Suppose that the polarity of θ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then $M, \pi, i + 1 \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, i + 1 \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$. The claim follows.

Let $\theta = \theta_1 U \theta_2$. Suppose that the polarity of θ in φ is positive. If $M, \pi, i \models \theta\,[\psi \leftarrow \mathbf{false}]$ then there exists some $j \geq i$ such that $M, \pi, j \models \theta_2\,[\psi \leftarrow \mathbf{false}]$ and forall $i \leq k < j$ we have $M, \pi, k \models \theta_1\,[\psi \leftarrow \mathbf{false}]$. From the induction assumption we know that $M, \pi, j \models_t \forall x \theta_2\,[\psi \leftarrow x]$ and forall $i \leq k < j$ we have $M, \pi, k \models_t \forall x \theta_1\,[\psi \leftarrow x]$. Clearly, the claim follows. Suppose that the polarity of θ in φ is negative. If $M, \pi, i \not\models \theta\,[\psi \leftarrow \mathbf{false}]$ then either forall $j \geq i$ we have $M, \pi, j \not\models \theta_2\,[\psi \leftarrow \mathbf{false}]$ or there exists some $j \geq i$ such that $M, \pi, j \not\models \theta_1\,[\psi \leftarrow \mathbf{false}]$ and forall $i \leq k < j$ we have $M, \pi, k \not\models \theta_2\,[\psi \leftarrow \mathbf{false}]$. In the first case, from the induction assumption it follows that forall $j \geq i$ we have $M, \pi, j \models_t \forall x \neg\theta_2\,[\psi \leftarrow x]$. In this case $M, \pi, i \models_t \forall x \neg\theta\,[\psi \leftarrow x]$. In the second case, from the induction assumption it follows that $M, \pi, j \models_t \forall x \neg\theta_1\,[\psi \leftarrow x]$ and forall $i \leq k < j$ we have $M, \pi, k \models_t \forall x \neg\theta_2\,[\psi \leftarrow x]$. Again, the claim follows. □

Theorem 3.6 *If ψ is of pure polarity in φ then the following are equivalent.*

1. $M, \pi, i \models \varphi\,[\psi \leftarrow \bot]$
2. $M, \pi, i \models_s \forall x\, \varphi\,[\psi \leftarrow x]$
3. *for every formula ξ we have $M, \pi, i \models \varphi\,[\psi \leftarrow \xi]$*
4. $M, \pi, i \models_t \forall x\, \varphi\,[\psi \leftarrow x]$

Proof: As we have shown in Lemmas 3.2 and 3.3, trace semantics is stronger than formula semantics, and the latter is stronger than structure semantics. Since $M, \pi, i \models_s \forall x\, \varphi\,[\psi \leftarrow x]$ for all structure assignments, including $\sigma(x) = S$ and $\sigma(x) = \emptyset$, we also have 2 $\Rightarrow$ 1. Thus 4 $\Rightarrow$ 3 $\Rightarrow$ 2 $\Rightarrow$ 1. In the other direction, Lemma 3.4 proves that 1 $\Rightarrow$ 4. □

Intuitively, the fact that a mapping can assign to a propositional variable opposite values in different positions along a trace (or states in a structure) is insignificant. Assigning the value $\bot$ is sufficient, and since the subformula is of pure polarity, $\bot$ is uniquely defined to be constant $\mathbf{true}$ or constant **false** throughout the trace. An outcome of Theorem 3.6 is that given a subformula ψ of pure polarity in an LTL formula φ, the following are equivalent: (1) ψ does not affect$_f$ φ in M (2) ψ does not affect$_s$ φ in M and (3) ψ does not affect$_t$ φ in M.

4 Algorithm and Complexity

In this section we give algorithms for checking vacuity according to the different definitions. As shown in previous sections, in the case of subformulas of pure polarity, the algorithm of [KV03] works for the three, equivalent, definitions. We show that this algorithm, which replaces a subformula by either $\mathbf{true}$ or $\mathbf{false}$ (according to its polarity), cannot be applied to subformulas of mixed polarity. We then study structure and trace vacuity. The question of how to decide formula vacuity remains open.

As shown in the previous section, in the case of subformulas of pure polarity the algorithm of [KV03] applies. We show that this algorithm cannot be applied to subformulas of mixed polarity. Consider the Kripke structure M_2 in Figure 1 and the formula $\varphi = p \rightarrow Xp$. Clearly, $M_2 \not\models_s \forall x\varphi\,[p \leftarrow x]$ (with the structure assignment $\sigma(x)$ including only the initial state), $M_2 \not\models \varphi\,[p \leftarrow q]$, and $M_2 \not\models_t \forall x\varphi\,[p \leftarrow x]$ (with the trace assignment $\alpha(x) = \{0\}$). Hence, p affects φ according to all three definitions. On the other hand, $M \models \varphi\,[p \leftarrow \mathbf{false}]$ and $M \models \varphi\,[p \leftarrow \mathbf{true}]$. We conclude that the algorithm of [KV03] cannot be applied to subformulas of mixed polarity.

We now solve trace vacuity. As mentioned, given an LTL formula φ, a model $M = \langle AP, S, S_0, R, L\rangle$ that satisfies φ, and a subformula ψ, we check whether ψ affects$_t$ φ in M by a reduction to model checking. We want to model check the UQLTL formula $\varphi' = \forall x\,\varphi\,[\psi \leftarrow x]$ on M. If $M \models_t \varphi'$ then ψ does not affect$_t$ φ. If $M \not\models_t \varphi'$ then ψ affects$_t$ φ. The algorithm presented below detects if ψ affects$_t$ φ in M.

```
1.  Compute the polarity of $\psi$ in $\varphi$.
2.  If $\psi$ is of pure polarity, model check $M \models \varphi[\psi \leftarrow \bot]$.
3.  Otherwise, construct $M' = \langle AP \cup \{x\}, S \times 2^x, S_0 \times 2^x, R', L\rangle$,
    where for every $X_1, X_2 \subseteq 2^x$ and $s_1, s_2 \in S$ we have
    $(s_1 \times X_1, s_2 \times X_2) \in R'$ iff $(s_1, s_2) \in R$.
5.  Model check $M' \models \varphi[\psi \leftarrow x]$.
    If passed, report "$\psi$ does not affect$_t$ $\varphi$", otherwise report "$\psi$ affects$_t$ $\varphi$".
```

Figure 2: Algorithm for Checking if ψ Affects$_t$ φ

The structure M' guesses at every step what is the right assignment for the propositional variable x. Choosing a path in M' determines the truth values of x along the path. Formally, we have the following claim.

Claim 4.1 (Correctness of the Algorithm) $M' \models \varphi[\psi \leftarrow x]$ *iff* $M \models_t \forall x\,\varphi[\psi \leftarrow x]$. [1]

Proof: If $M \not\models_t \forall x\,\varphi\,[\psi \leftarrow x]$, then there exists a trace $\pi = s_0, s_1, \ldots$ and a mapping α such that $M, \pi, 0, \alpha \not\models_t \varphi\,[\psi \leftarrow x]$. Let $\overline{x}_i$ be a predicate that is $\mathbf{true}$ iff $i \in \alpha(x)$. The trace $\psi' = (s_0, \overline{x}_0), (s_1, \overline{x}_1) \ldots \in \mathcal{T}(M')$ according to the construction of M'. For every $p \in AP \cup \{x\}$, the truth values of p along π and π' are identical. Thus $M' \not\models \varphi[\psi \leftarrow x]$. The other direction is similar. If $M' \not\models \varphi[\psi \leftarrow x]$, then there exists a path $\pi' = s_0, s_1, \ldots$ in M' such that $M', \pi', 0 \not\models \varphi\,[\psi \leftarrow x]$. According to the construction of M', a corresponding path π also exists in M, apart from the labeling of x. Let α assign the truth values of x along π' for the propositional variable x in M. Since $M, \pi, 0, \alpha \not\models_t \varphi[\psi \leftarrow x]$, we have $M' \models \forall x\varphi[\psi \leftarrow x]$. ☐

We show that trace vacuity is linear in the structure and PSPACE-complete in the formula.

Theorem 4.2 [VW94] *Given a structure M and an LTL formula φ, we can model check φ over M in time linear in the size of M and exponential in φ and in space polylogarithmic in the size of M and quadratic in the length of φ.*

[1] Notice that x is a propositional variable in M, but an atomic proposition in M'.

Corollary 4.3 *Given a structure M and an LTL formula φ such that $M \models \varphi$, we can decide whether subformula ψ affects$_t$ φ in time linear in the size of M and exponential in φ and in space polylogarithmic in the size of M and quadratic in the length of φ.*

Recall that in symbolic model checking, the modified structure M' is not twice the size of M but rather includes just one additional variable. The modified formula $\varphi\,[\psi \leftarrow x]$ is at most as long as φ. The corollary follows. In order to check whether φ is trace vacuous we have to check whether there exists a subformula ψ of φ such that ψ does not affect$_t$ φ. Given a set of subformulas $\{\psi_1, \ldots, \psi_n\}$ we can check whether one of these subformulas does not affect$_t$ φ by iterating the above algorithm n times. The number of subformulas of φ is proportional to the size of φ.

Theorem 4.4 *Given a structure M and an LTL formula φ such that $M \models \varphi$. We can check whether φ is trace vacuous in M in time $O(|\varphi| \cdot C_M(\varphi))$ where $C_M(\varphi)$ is the complexity of model checking φ over M.*

We show now that unlike trace vacuity, there does not exist an efficient algorithm for structure vacuity. We show that deciding does not affect$_s$ is co-NP-complete in the structure. Notice, that co-NP-complete in the structure is much worse than PSPACE-complete in the formula. Indeed, the size of the formula is negligible when compared to the size of the model. Co-NP-completeness of structure vacuity renders it completely impractical.

Lemma 4.5 (Deciding does not affect$_s$) *For φ in LTL, a subformula ψ of φ and a structure M, the problem of deciding whether ψ does not affect$_s$ φ in M is co-NP-complete with respect to the structure M.*

Proof: We show membership in co-NP. We consider the complementary problem of deciding affect$_s$. Consider a formula φ and a structure $M = \langle AP, S, S_0, R, L\rangle$. In order to check whether ψ affects$_s$ φ we have to model check $\forall x \varphi\,[\psi \leftarrow x]$ over M. Guess a subset S' of S and set the structure assignment $\sigma(x) = S'$. Now model check the formula $\varphi\,[\psi \leftarrow x]$ over the structure $M' = \langle AP \cup \{x\}, S, S_0, R, L'\rangle$ where $L'(x) = S'$ and $L'(p) = L(p)$ for $p \neq x$.

In Appendix B we give a reduction from 3CNF satisfiability to deciding affects$_s$. Given a 3CNF formula θ, we construct a structure M_θ and a (fixed) formula φ such that $M_\theta \models \varphi$ and the proposition q affects$_s$ φ in M_θ iff θ is satisfiable. $\square$

The complexity of deciding affects$_f$ is unclear. As shown, in the case of subformulas of pure polarity (or occurrences of subformulas) the algorithm of [KV03] is correct. We have not found either a lower bound or an upper bound for deciding affects$_f$ in the case of mixed polarity.

5 Pragmatic Aspects

In this section we give some pragmatic aspects of vacuity detection. We discuss the different options for reporting vacuity. While previous works considers only giving a yes / no answer, we advocate giving the users the witness formula (see below) as well so that they can best understand why the formula passes vacuously. We also check what the relation is between subformulas and occurrences of subformulas. We conclude that in order to get the most thorough vacuity detection both should be accounted for. Guided by these two observations, we show how we can achieve the most thorough vacuity detection while reducing the number of model-checker runs. Finally, we report on our experience using vacuity detection in an industrial setting. All the work in this section relates to trace vacuity. Therefore, we remove the subscript describing the type of affect.

5.1 Display of Results

When applying vacuity detection in an industrial setting there are two options. We can either give the user a simple yes/no answer, or we can accompany a positive answer (vacuity) with a witness formula.

Where ψ does not affect φ we supply $\varphi\,[\psi \leftarrow x]$ (or $\varphi\,[\psi \leftarrow \bot]$ where ψ is of pure polarity) as our witness to the vacuity of φ. When we replace a subformula by a constant, we propagate the constants upwards. For example, if in subformula $\theta = \psi_1 \wedge \psi_2$ we replace ψ_1 by **false**, then θ becomes **false** and we continue propagating this value above θ.

$$\boxed{\begin{array}{c}\text{active} := \text{en} \wedge\neg\text{in}\ ; \quad \text{rdy\_active} := \neg\ \text{rdy\_out} \wedge\neg\ \text{active}\ ;\\ \text{bsy\_active} := \neg\ \text{bsy\_out} \wedge\neg\ \text{active};\\ \text{active\_inactive} := \text{rdy\_active} \wedge\neg\ \text{bsy\_active}\ ;\\ \text{two\_consecutive} := G[(\text{reset} \wedge \text{active\_inactive}\ ) \rightarrow X\neg\text{active\_inactive}];\end{array}}$$

Figure 3: Vacuous pass

Previous works were interested only in providing a simple yes / no answer. That is, whether the property is vacuous or not. In this case it suffices to check whether the propositions affect the formula [BBER97, KV03]. Suppose that ψ does not affect φ. It follows that if ψ' is a subformula of ψ then ψ' does not affect φ as well. In view of the above, in order to get a yes / no answer only the minimal subformulas of φ (i.e. the atomic propositions that appear in φ) have to be checked. In contrast, when the goal is to give the user feedback on the source of detected vacuity, it is often more useful to check non-minimal subformulas.

Consider for example the formula *two_consecutive* in Figure 3. This is an example of a formula that passed vacuously in one of our designs. The reason for the vacuous pass is that one of the signals in *active_inactive* was set to **false** by a wrong environmental assumption. The following formula is the witness to the fact that the second occurrence of *active_inactive* does not affect *two_consecutive*.

$$\text{two\_consecutive}\,[\text{active\_inactive}_2 \leftarrow \bot] = G\neg(\text{reset} \wedge \text{active\_inactive})$$

From this witness it is straightforward to understand what is wrong with the formula. The witness formula associated with the occurrence of the proposition *en* under the second occurrence of *rdy_active* (after constant propagation) is as follows. Note that this occurrence of *en* occurs positively in *two_consecutive*.

$$\text{two\_consecutive}\,[\text{en}_2 \leftarrow \bot] = G[(\text{reset} \wedge \text{active\_inactive}) \rightarrow X\neg(\neg\text{rdy\_out} \wedge \neg\text{bsy\_active})]$$

Clearly, this report is much less legible. This formula has very little connection to the original formula. Thus, it is preferable to check vacuity of non-minimal subformulas and subformula occurrences.

If we consider the formula as represented by a tree (rather than DAG – directed acyclic graph) then the number of leaves (propositions) is proportional to the number of nodes (subformulas). We apply our algorithm from top to bottom. We check whether the maximal subformulas affect the formula. If a subformula does not affect, there is no need to continue checking below it. If a subformula does affect, we continue and check its subformulas. In the worst case, when all the subformulas affect the formula, the number of model checker runs in order to give the most intuitive counter example is double the size of the minimal set (number of propositions). The yes / no view vs. the intuitive witness view offer a clear tradeoff between minimal number of model checker runs (in the worst case) and giving the user the most helpful information. We believe that the user should be given the most comprehensive witness. In our implementation we check whether **all** subformulas and occurrences of subformulas affect the formula.

5.2 Occurrences vs. Subformulas

In Section 4 we introduced an algorithm that can determine if a subformula with multiple occurrences affects a formula. In most cases it makes sense to check if a subformula affects a formula. Sometimes it is more intuitive to check if an occurrence of a subformula affects the formula. We give examples in which checking a subformula is more intuitive, and other examples in which checking an occurrence is more intuitive. We examine if a vacuity detection algorithm can choose one over the other.

The following example demonstrates why it is reasonable to check if a subformula affects a formula. Let $\varphi = G(p \rightarrow p)$, Intuitively, p does not affect φ since every expression (or variable) implies itself. Indeed, according to all the definitions p does not affect φ, regardless of the model. However, every occurrence of p may affect φ. Indeed, both $Gp = \varphi\,[p_1 \leftarrow \bot]$ and $G\neg p = \varphi\,[p_2 \leftarrow \bot]$ may fail (here, p_i denotes the ith occurrence of p).

The following example, on the other hand, demonstrates why it is reasonable to check if an occurrence affects a formula. Let $\varphi = p \wedge G(q \rightarrow p)$. Assume q is always $\mathbf{false}$ in model M. Clearly, the second occurrence of p does not affect φ in M. However, the subformula p does affect φ in M. Every assignment that gives x the value $\mathbf{false}$ at time 0 would falsify the formula $\varphi\,[p \leftarrow x]$. Recall the formula *two_consecutive* in Figure 3. The vacuous pass in this case is only with respect to occurrences and not to subformulas.

We believe that a *thorough vacuity-detection* algorithm should detect both subformulas and occurrences that do not affect the examined formula. It is up to the user to decide which vacuity alerts to ignore.

5.3 Minimizing the number of checks

As explained above we choose to check whether all subformulas and all occurrences of subformulas affect the formula. Applying this policy in practice may result in many runs of the model checker and may be impractical. In particular, when the formula is represented as a DAG, checking all occurrences involves turning the DAG into a tree. We show that we can reduce the number of subformulas and occurrences for which we check vacuity by analyzing the structure of the formula syntactically.

It is straightforward to see that if ψ' is a subformula of ψ and ψ does not affect φ then also ψ' does not affect φ. Hence, once we know that ψ does not affect φ, there is no point in checking subformulas of ψ. If ψ affects φ we have to check also the subformulas of ψ. We show that in some cases for ψ' a subformula of ψ we have ψ' affects φ iff ψ affects φ. In these cases there is no need to check direct subformulas of ψ also when ψ affects φ.

Suppose the formula φ is satisfied in M. Consider an occurrence θ_1 of the subformula $\theta = \psi_1 \wedge \psi_2$ of φ. We show that if θ_1 is of positive polarity then ψ_i affects φ iff θ_1 affects φ for $i = 1, 2$. As mentioned, θ_1 does not affect φ implies ψ_i does not affect φ for $i = 1, 2$. Suppose θ_1 affects φ. Then $M \not\models \varphi\,[\theta_1 \leftarrow \mathbf{false}]$. However, $\varphi\,[\psi_i \leftarrow \mathbf{false}] = \varphi\,[\theta_1 \leftarrow \mathbf{false}]$. It follows that $M \not\models \varphi\,[\psi_i \leftarrow \mathbf{false}]$ and that ψ_i affects φ. In the case that θ_1 is of negative (or mixed) polarity the above argument is incorrect. Consider the formula $\varphi = \neg(\psi_1 \wedge \psi_2)$ and a model where ψ_1 never holds. It is straightforward to see that $\psi_1 \wedge \psi_2$ affects φ while ψ_2 does not affect φ.

Similarly consider the subformula $\theta = G\psi_1$ and the occurrence θ_1 of θ of negative polarity. We show that θ_1 affects φ iff ψ_1 affects φ. Suppose θ_1 affects φ. Then $M \not\models \varphi\,[\theta_1 \leftarrow \mathbf{true}]$. As before $\varphi\,[\theta_1 \leftarrow \mathbf{true}] = \varphi\,[\psi_1 \leftarrow \mathbf{true}]$. Suppose that θ_1 is of mixed polarity and that θ_1 affects φ. Then $M \not\models \forall x \varphi\,[\theta_1 \leftarrow x]$. However, we can not prove that $M \not\models \forall x \varphi\,[\psi_1 \leftarrow x]$. This is true only if there exists a computation π of M, an assignment α such that for some $i \geq 0$ we have $\alpha(x) = \{i, \ldots\}$ and $\pi, 0, \alpha \not\models \varphi\,[\theta_1 \leftarrow x]$.

From the above discussion it follows that we can analyze the form of the formula φ syntactically and identify occurrences θ_1 such that θ_1 affects φ iff the subformulas of θ_1 affect φ. In these cases it is sufficient to model check the formula $\forall x \varphi\,[\theta_1 \leftarrow x]$. Below the immediate subformulas of θ_1 we have to continue with the same analysis. For example, if $\theta = (\psi_1 \vee \psi_2) \wedge (\psi_3 \wedge \psi_4)$ is of positive polarity and θ affects φ we can ignore $(\psi_1 \vee \psi_2)$, $(\psi_3 \wedge \psi_4)$, ψ_3, and ψ_4. We do have to check ψ_1 and ψ_2. In Table 1 we list the operators under which we can apply such elimination. In the polarity column we list the polarities under which the elimination scheme applies to the operator. In the operands column we list the operands that we do not have to check. We stress that below the immediate operands we have to continue applying the analysis.

The analysis that leads to the above table is quite simple. Using a richer set of operators one must use similar reasoning to extend the table. Notice that we distinguish between pure polarity and mixed polarity. As the above table is true for occurrences, mixed polarity is only introduced in cases that the specification language includes operators with no polarity (e.g. $\oplus$, $\leftrightarrow$).

Operator	Polarity	Operands
$\wedge$	+	all
$\vee$	-	all
$\neg$	pure / mixed	all
X	pure / mixed	all
U	pure	second
G	pure	all
F	pure	all

Table 1: Operators for which checks can be avoided

As mentioned, the above elimination works only in the case of occurrences. Indeed, in the case of a subformula with multiple occurrences one has to take into account the polarities of all occurrences and the operator under which every occurrence appears. However, suppose that the subformula $\theta = f(\psi_1, \psi_2)$ occurs more than once but ψ_1 and ψ_2 occur only under θ. In this case, once we check whether θ affects φ, the elimination scheme can be applied to ψ_1 and ψ_2.

5.4 Implementation and Methodology

We implemented the above algorithms in Intel's formal verification environment. We use the language ForSpec [AFF$^+$02] with the BDD-based model checker Forecast [FKZ$^+$00] and the SAT-based bounded model checker Thunder [CFF$^+$01]. We enable the users to decide whether they want thorough vacuity detection or just to specify which subformulas / occurrences should be checked. In the case of thorough vacuity detection, for every subformula and every occurrence (according to the elimination scheme above) we create one witness formula. The vacuity algorithm amounts to model checking each of the witnesses. Both model checkers are equipped with a mechanism that allows model checking of many properties simultaneously.

The current methodology of using vacuity is applying thorough vacuity on every specification. The users prove that the property holds in the model; then, vacuity of the formula is checked. If applying thorough vacuity is not possible (due to capacity problems), the users try to identify the important subformulas and check these subformulas manually. In our experience, vacuity checks proved to be effective mostly when the pruning and assumptions used in order to enable model checking removed some important part of the model, thus rendering the specification vacuously true. In many examples vacuity checking pointed out to such problems. We also have cases where vacuity pointed out redundant parts in the specification.

In Table 2 we include some experimental results. We used real-life examples from processor designs. We include these results in order to give the flavor of the performance of vacuity checking. All the times are given in seconds. Some of the properties in the table below consist of more than one assertion. That is, the property is the conjunction of a few assertions but each assertion is checked separately (both in model checking and vacuity detection). For each property we report on 4 different experiments. First, model checking the property using Forecast. Second, model checking all the witness formulas for all the assertions using Forecast. Third, model checking all the witness formulas with all the assertions using Forecast. Finally, model checking all the witness formulas for all the properties using Thunder up to bound 200. Recall that in vacuity detection we hope that all the witness formulas do not pass in the model. As bounded model checking is especially adequate for falsification, we prove the correctness of the property using Forecast and falsify the witness formulas using Thunder. Witness formulas that Thunder was unable to falsify can be checked manually using Forecast.

In the table below, every line corresponds to 4 experiments as explained. For each property (assertion or set of assertions) we write the number of witness formulas sent to the model checker. The number in parentheses indicates the number of non-affecting subformulas / occurrences. For Forecast, the first column (titled MC) reports on the results of model checking, the second column (titled Vacuity) reports

Property	♯ Checks	Forecast						Thunder
		MC		Vacuity		Combined		
		Time	Nodes	Time	Nodes	Time	Nodes	
check_internal_sig	5(1)	1936	3910K	2051	2679K	3185	5858K	2.28(0)
lsd_indication	17(5)	1699	2150K	2265	2566K	1986	3483K	!(5)[40]
directive	4(0)	611	1120K	16132	4945K	7355	8943K	25(0)
forbidden_start	2(0)	532	549K	1859	4064K	2422	4274K	22(0)
nested_start	22 (13)	737	1294K	11017	6153K	10942	6153K	!(18)[70]
pilot_session	129(16?)	5429	3895K	67126!	25366K	66157!	20586K	!(16)[60]
new_code	31(1)	1265	2455K	1765	2853K	3097	3932	!(1)[50]

Table 2: Experimental results

on the results of vacuity detection, and the third column (titled Combined) reports on the results of the combined model and vacuity checking. In each column we specify the time (in seconds) and space (BDD nodes) required by the model checker. The symbol ! indicates that Forecast timed out. For Thunder we write the time (in seconds) required by the model checker. The number in parentheses indicates the number of properties that were not refuted up to bound 200. The symbol ! indicates that Thunder did not terminate in 8 hours. In these cases we report (in brackets) the bound up to which the formulas were checked (note that in the case of *lsd_indication* and *new_code* the partial answer is in fact the final answer, as can be seen from the run of Forecast). We ran the examples on a Intel(R) Pentium™ 4 2.2GHz processor running Linux with 2GByte memory. Notice that some of these examples pass vacuously.

6 Summary and Future Work

In this work we investigated vacuity detection with respect to subformulas with multiple occurrences. The generality of our framework required us to re-examine the basic intuition underlying the concept of vacuity, which until now has been defined as sensitivity with respect to syntactic perturbation. We studied sensitivity with respect to semantic perturbation, which we modeled by universal propositional quantification. We showed that this yields a hierarchy of vacuity notions. We argued that the right notion is that of vacuity defined with respect to traces, described an algorithm for vacuity detection, and discussed pragmatic aspects.

As mentioned in the introduction, we were motivated by the need to extend vacuity detection to industrial-strength property-specification languages such as ForSpec [AFF+02]. ForSpec is significantly richer syntactically and semantically than LTL. Our vacuity-detection algorithm for subformulas of mixed polarity can handle ForSpec's rich set of arithmetical and Boolean operators. ForSpec's semantic richness is the result of its *regular layer*, which includes regular events and formulas constructed from regular events. The extension of vacuity detection to ForSpec's regular layer will be described in a future paper.

7 Acknowledgments

We thank Ranan Fraer, Dana Lichten and Khurram Sajid for their contribution and useful remarks.

References

[AFF+02] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M.Y. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 2280 of *Lecture Notes in Computer Science*, pages 296–211, Grenoble, France, April 2002. Springer-Verlag.

[BB94] D. Beaty and R. Bryant. Formally verifying a microprocessor using a simulation methodology. In *Proc. 31st Design Automation Conference*, pages 596–602. IEEE Computer Society, 1994.

[BBER97] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. In *Proc. 9th Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 279–290, 1997.

[BBER01] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. *Formal Methods in System Design*, 18(2):141–162, 2001.

[CE81] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[CES86] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, January 1986.

[CFF$^+$01] F. Copty, L. Fix, R. Fraer, E. Giunchiglia, G. Kamhi, A. Tacchella, and M.Y. Vardi. Benefits of bounded model checking at an industrial setting. In *Computer Aided Verification, Proc. 13th International Conference*, volume 2102 of *Lecture Notes in Computer Science*, pages 436–453. Springer-Verlag, 2001.

[CGMZ95] E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32nd Design Automation Conference*, pages 427–432. IEEE Computer Society, 1995.

[CGP99] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[Eme90] E.A. Emerson. Temporal and modal logic. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 16, pages 997–1072. Elsevier, MIT press, 1990.

[FKZ$^+$00] R. Fraer, G. Kamhi, B. Ziv, M. Vardi, and L. Fix. Prioritized traversal: efficient reachability analysis for verication and falsification. In *Proc. 12th Conference on Computer Aided Verication*, volume 1855 of *Lecture Notes in Computer Science*, pages 389–402, Chicago, IL, USA, July 2000. Springer-Verlag.

[Kup95] O. Kupferman. Augmenting branching temporal logics with existential quantification over atomic propositions. In *Computer Aided Verification, Proc. 8th International Conference*, volume 939 of *Lecture Notes in Computer Science*, pages 325–338, Liege, July 1995. Springer-Verlag.

[Kur98] R.P. Kurshan. *FormalCheck User's Manual*. Cadence Design, Inc., 1998.

[KV99] O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. In *10th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 1703 of *Lecture Notes in Computer Science*, pages 82–96. Springer-Verlag, 1999.

[KV03] O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. *Software Tools for Technology Transfer*, 4(2):224–233, February 2003.

[LP85] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages*, pages 97–107, New Orleans, January 1985.

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, Brooklyn, June 1985. Springer-Verlag.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symp. on Foundation of Computer Science*, pages 46–57, 1977.

[PP95] B. Plessier and C. Pixley. Formal verification of a commercial serial bus interface. In *Proc. of 14th Annual IEEE International Phoenix Conference on Computers and Communications*, pages 378–382, March 1995.

[PS02] M. Purandare and F. Somenzi. Vacuum cleaning ctl formulae. In *Proc. 14th Conference on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 485–499. Springer-Verlag, July 2002.

[QS81] J.P. Queille and J. Sifakis. Specification and verification of concurrent systems in Cesar. In *Proc. 5th International Symp. on Programming*, volume 137 of *Lecture Notes in Computer Science*, pages 337–351. Springer-Verlag, 1981.

[SVW85] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *Proc. 10th International Colloquium on Automata, Languages and Programming*, volume 194, pages 465–474, Nafplion, July 1985. Lecture Notes in Computer Science, Springer-Verlag.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

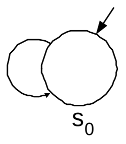
A Proof of Theorem 2.3

Theorem A.1 *Given a structure M and a UQLTL formula φ:*

- $M \models_t \varphi \quad \Rightarrow \quad M \models_s \varphi$
- $M \models_s \varphi \quad \not\Rightarrow \quad M \models_t \varphi$

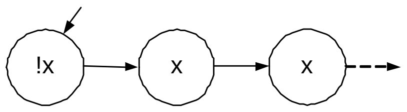
Proof: Assume in the way of contradiction that $M \models_t \varphi$ but $M \not\models_s \varphi$. Then there exists a structure assignment σ and a trace π such that $M, \pi, 0, \sigma \not\models_s \varphi$. Let $\pi = s_0, s_1, s_2, \ldots$. We build the assignment $\alpha(x) = \{i | \, s_i \in \sigma(x)\}$, which includes point i in the assignment α of a propositional variable x iff s_i is in $\sigma(x)$. Both assignments map all propositional variables in φ to the same truth values along the trace π, thus $M, \pi, 0, \alpha \not\models_t \varphi$. This implies that $M \not\models_t \varphi$, in contradiction with the assumption.

For the other direction, consider the formula $\varphi = \forall x G(x \to Xx)$ and a Kripke structure with a single state s_0 that has a self loop.



We show that $M, \sigma \models_s \varphi$ for every σ. There are two possible structure assignments, $\sigma(x) = \emptyset$ and $\sigma(x) = s$. If $s_0 \in \sigma(x)$, then x is always satisfied and $M, \sigma \models_s \varphi$. If $s_0 \notin \sigma(x)$, then $X\neg x$ is always satisfied and $M, \sigma \models_s \varphi$. Thus $M \models_s \varphi$.

We now show that $M \not\models_t \varphi$. Notice that M has a single trace π. Consider the trace assignment α that maps x to all points along π except the first one. That is, $\alpha(x) = \mathbf{N} \setminus \{0\}$. For that assignment $M, \pi, 0, \alpha \not\models_t \varphi$, which implies $M \not\models_t \varphi$.



□

B Deciding does not affect$_s$ is co-NP-hard

Lemma B.1 *For φ in LTL, a subformula ψ of φ and a structure M, the problem of deciding whether ψ does not affect$_s$ φ in M is co-NP-complete with respect to the structure M.*

Proof: We show co-NP-hardness. We consider the complementary problem of deciding affect$_s$. We give a reduction from 3CNF satisfiability. For every 3CNF formula θ we construct a structure M_θ. We give a (fixed) LTL formula φ such that $M_\theta \models \varphi$ and the proposition q affects$_s$ φ in M_θ iff θ is satisfiable. Consider the formula $\varphi' = \forall x \varphi\, [q \leftarrow x]$. By definition, $M_\theta \not\models \forall x \varphi'$ iff there exists an assignment σ such that $M, \sigma \not\models \varphi'$. We construct M_θ so that the set $\sigma(x)$ represents a satisfying assignment to θ.

For every proposition p_i in θ we have a set of states that represent the assignment $p_i = \mathbf{false}$ and a set of states that represent the assignment $p_i = \mathbf{true}$. The formula φ is constructed so that $M, \sigma \models \varphi\, [q \leftarrow x]$ whenever σ chooses for x a set of states that cannot represent a valid assignment to the propositions of θ. For example, if σ chooses for x only some of the states that represent $p_i = \mathbf{false}$ (or $p_i = \mathbf{true}$) or if σ chooses for x some states that represent $p_i = \mathbf{false}$ and some states that represent $p_i = \mathbf{true}$ for some proposition p_i.

For every clause c_i of θ we add one path to M_θ. If the clause c_i uses propositions p_a, p_b, and p_c we create a path linking a state representing proposition p_a to a state representing proposition p_b to a state representing proposition p_c. If p_a appears in c_i positively, we choose a state that represents $p_a = \mathbf{true}$, otherwise we choose a state that represents $p_a = \mathbf{false}$. Similarly for p_b and p_c. This way, if $\sigma(x)$ is a valid assignment that does not satisfy the clause c_i then all the states on the path of c_i in M_θ are not in $\sigma(x)$.

Let $\theta = \bigwedge_{i=1}^{n} \bigvee_{j=1}^{3} \alpha_{i,j}$ where $\alpha_{i,j}$ is a literal in $\{p_1, \ldots, p_k\} \cup \{\neg p_1, \ldots, \neg p_k\}$. For every proposition p_i the structure M_θ contains $2n$ states. The first n states represent the assignment $p_i = \mathbf{true}$ and the other n states represent the assignment $p_i = \mathbf{false}$. Then for every clause $c_i = \alpha_{i,1} \vee \alpha_{i,2} \vee \alpha_{i,3}$, we create a path that connects the literals in c_i.

Let $M_\theta = \langle \{c, pos, neg, q\}, S, \{s_0\}, R, L\rangle$. The set of states S is the union of the following sets.

- $\{s_0\}$ - the initial state.
- $\{c_{i,1}, c_{i,2} \mid 1 \leq i \leq n\}$ - two clausal states per clause. These states are used in the path that represents clause i to separate the different proposition states.
- $\{p^+_{l,i}, p^-_{l,i} \mid 1 \leq l \leq k$ and $1 \leq i \leq n\}$ - $2n$ propositional states per proposition, n positive and n negative.

The transition relation is the union of the following sets.

- $R_1 = \{(s_0, p^+_{l,1}) \mid 1 \leq l \leq k\}$ - the initial state s_0 is connected to every first positive propositional state $p^+_{l,1}$.
- $R_2 = \{(p^+_{l,i}, p^+_{l,i+1}), (p^-_{l,i}, p^-_{l,i+1}) \mid 1 \leq l \leq k$ and $1 \leq i \leq n-1\}$ - the positive states related to proposition p_l and the negative states related to proposition p_l form chains.
- $R_3 = \{(p^+_{l,n}, p^-_{l,1}), (p^-_{l,n}, p^-_{l,n}) \mid 1 \leq l \leq k\}$ - The last positive state of p_l is connected to the first negative state. The last negative state of p_l is connected to itself.
- For every clause $\theta_i = \beta_1 \cdot p_a \vee \beta_2 \cdot p_b \vee \beta_3 \cdot p_c$ where $\beta_o \in \{+, -\}$ for $o \in \{1,2,3\}$ we add the transitions $R_{4,i} = \{(s_0, p^{\beta_1}_{a,i}), (p^{\beta_1}_{a,i}, c_{i,1}), (c_{i,1}, p^{\beta_2}_{b,i}), (p^{\beta_2}_{b,i}, c_{i,2}), (c_{i,2}, p^{\beta_3}_{c,i})\}$ - there is a path connecting the literals of clause c_i according to their polarities. Between every two propositional states there is a clausal state. We refer to this path as a clausal path. The only way to get from one proposition state to another proposition state in one step is by taking transitions in $R_2 \cup R_3$. Notice that the paths that correspond to different clauses do not share transitions.

The labeling is $L(c) = \{c_{i,j}\}$, $L(pos) = \{p^+_{i,l}\}$, $L(neg) = \{p^-_{i,l}\}$, and $L(q) = \emptyset$. In Figure B we have the 'propositional' part of M_θ without the clausal states and transitions. The structure M_θ can be constructed in polynomial time.

The formula φ is the disjunction of the following formulas.

- $\varphi_1 = F(pos \wedge X pos \wedge ((q \wedge X\neg q) \vee (\neg q \wedge Xq)))$ - there are two positive states associated with the same proposition (reachable in one step) assigned with different values of q.
- $\varphi_2 = F(neg \wedge X neg \wedge ((q \wedge X\neg q) \vee (\neg q \wedge Xq)))$ - there are two negative states associated with the same proposition (reachable in one step) assigned with different values of q.
- $\varphi_3 = F(pos \wedge X neg \wedge ((q \wedge Xq) \vee (\neg q \wedge X\neg q)))$ - the last positive state and the first negative state agree on the assignment of q.

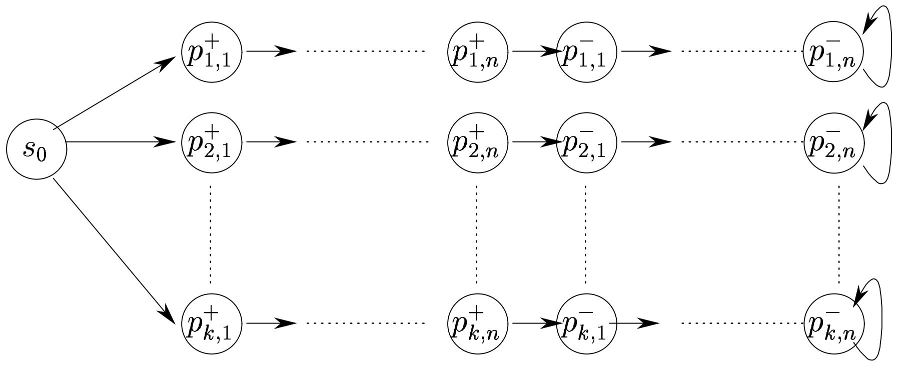


Figure 4: The structure M_θ

- $\varphi_4 = X(\neg q \wedge X(c \wedge X(\neg q \wedge X(c \wedge X \neg q))))$ - all three literals are not satisfied on a clausal path.

As $L(q) = \emptyset$ the formula φ_3 holds in M and $M_\theta \models \varphi$. We claim that $M_\theta \not\models \forall x \varphi\, [q \leftarrow x]$ iff θ is satisfiable. Indeed, every assignment to x that does not satisfy $\varphi\, [q \leftarrow x]$ must include either all the positive states associated with one proposition or all the negative states associated with one proposition (and not both). Furthermore, as the assignment falsifies $\varphi\, [q \leftarrow x]$ every path associated with some clause must have at least one literal satisfied. Similarly, a satisfying assignment to θ translates to a subset of the states S' assigning $\sigma(x) = S'$ falsifies $\varphi\, [q \leftarrow x]$. □

In [KV03] Kupferman and Vardi show that deciding affects$_f$ for CTL formulas is NP-complete. They give a reduction from SAT to deciding affects$_f$. In their proof both the structure and the CTL formula depend on the SAT formula. Our proof above can be used to show that for CTL formulas, deciding affects$_f$ is NP-hard in the structure even for a constant formula.