A Framework for Inherent Vacuity

Dana Fisman[1,2]*, Orna Kupferman[1], Sarai Sheinvald-Faragy[1], and Moshe Y. Vardi[3]**

[1] School of Computer Science and Engineering, Hebrew University, Jerusalem 91904, Israel.
[2] IBM Haifa Research Lab, Haifa University Campus, Haifa 31905, Israel.
[3] Rice University, Houston Texas 77005, USA.

Abstract. Vacuity checking is traditionally performed after model checking has terminated successfully. It ensures that all the elements of the specification have played a role in its satisfaction by the design. Vacuity checking gets as input both design and specification, and is based on an in-depth investigation of the relation between them. Vacuity checking has been proven to be very useful in detecting errors in the modeling of the design or the specification. The need to check the quality of specifications is even more acute in *property-based design*, where the specification is the only input, serving as a basis to the development of the system. Current work on property assurance suggests various sanity checks, mostly based on satisfiability, non-validity, and realizability, but lacks a general framework for reasoning about the quality of specifications.
We describe a framework for *inherent vacuity*, which carries the theory of vacuity in model checking to the setting of property-based design. Essentially, a specification is inherently vacuous if it can be mutated into a simpler equivalent specification, which we show to coincide with the fact the specification is satisfied vacuously in all systems. We also study the complexity of detecting inherent vacuity, and conclude that while inherent vacuity leads to specifications that better capture designer intent, it is not more complex than simple property-assurance checks.

1 Introduction

In recent years, we see a growing awareness to the importance of assessing the quality of (formal) specifications. In the context of model checking, a specification consists of a set of formulas written in some temporal logic, and the quality of the specification is assessed by analyzing the effect of applying mutations to the formulas. If the system satisfies the mutated specification, we know that some elements of the specification do not play a role in its satisfaction, thus the specification is satisfied in some *vacuous* way [3, 20]. Vacuity is successfully used in order to improve specifications and detect design errors [18].

The need to assess the quality of specifications is even more acute in the context of *property-based design* [24]. There, the design process starts with the development of the specification as a set of temporal formulas, which then serves as a basis to the development of the implementation. For example, in *temporal synthesis* [23], we go automatically from the specification to a system that satisfies it. Indeed, one of the criticisms against synthesis is that it does not eliminate the difficulty of design, but merely shifts the difficulty of developing correct implementations to that of developing correct specifications [17].

* The work of this author was done as part of the Valazzi-Pikovsky Fellowship Fund.
** Supported in part by NSF grants CCR-0124077, CCR-0311326, CCF-0613889, ANI-0216467, and CCF-0728882, by BSF grant 9800096, and by gift from Intel.

Property assurance is the activity of eliciting specifications that faithfully capture designer intent [5, 27]. Obvious quality checks one may perform for a given specification are *non-validity* and *satisfiability* [28]. More involved quality checks are studied in the PROSYD project [24]. There, one considers a set of temporal formulas, partitioned into assumptions and guarantees, and checks consistency of the assumptions and various types of entailment of guarantees by assumptions. Recent work has focused on other aspects of property assurance. For example, both [10, 22] study *completeness analysis for property sets*. There, one analyzes a specification consisting of a set of temporal formulas and measures the degree to which the specification determines the exact behavior of each of its signals. There is a trade-off between the level of abstraction that a specification enjoys and the level of detail in which it describes the system. The analysis in [10, 22] takes one side of this trade-off, as it expects the specifications to determine the exact behavior of the signals. [4]

As discussed in [27], checking vacuity of the formulas in the context of property assurance would be of great importance. While vacuity has been widely studied in the context of model checking [2, 3, 6, 14, 20, 21], it is not clear how to define and check vacuity of formulas without having a system that is meant to satisfy these formulas. A first step for analyzing vacuity in a specification is taken in [9], which studies *early detection of vacuity* and provides the inspiration to this work. There too, a specification consists of a set of temporal formulas, and the goal is to reduce vacuity of the specification by removing formulas that are implied by the specification, and by strengthening formulas to ones that are still implied by the specification. While [9] introduced the intuitive concept of "vacuity without design", it does not attempt to define this concept. Rather, it offers various sanity checks that can be applied to sets of properties, with the aim of simplifying later vacuity checking with respect to a design. Our aim in this work is to formalize the intuitive concept introduced in [9].

We describe a framework for *inherent vacuity* for sets of linear temporal properties. The term "inherent" refers to the fact that we do not study vacuity of properties with respect to a given system, but as a quality measure of the properties themselves. We focus on both identifying the appropriate definition of inherent vacuity, as well as developing algorithms for testing inherent vacuity.

Before we present our definition for inherent vacuity, let us recall one definition of vacuity in LTL model checking [2]. There, given a system $\mathcal{S}$ and a specification φ that is satisfied in $\mathcal{S}$, we say that a subformula ψ of φ *does not affect* the satisfaction of φ in $\mathcal{S}$ if $\mathcal{S}$ also satisfies the stronger specification $\forall x.\varphi[\psi \leftarrow x]$, in which ψ is replaced by a universally quantified proposition. Intuitively, this means that $\mathcal{S}$ satisfies φ even with the most challenging assignments to ψ.[5] The specifications φ is then *vacuously satisfied* in $\mathcal{S}$ if it has a subformula that does not affect its satisfaction in $\mathcal{S}$.

[4] A related line of research is that of *specification debugging* [1], where, in the process of model checking, counterexamples are automatically clustered together in order to make the manual debugging of temporal properties easier. Another related line of research is that of *coverage metrics* [8, 16]. There, the mutations are applied to the system, and if the mutated system satisfies the specification, we know that some elements of the system are not covered by the specification.

[5] Since ψ may have several occurrences in φ there need not be a single "most challenging assignment" and it need not be *true* or *false*.

There are two natural approaches to lift the definition of vacuity in the context of model checking to a definition of inherent vacuity. In order to see the idea behind the first approach, consider specifications that are tautologies or contradictions. One need not have a context in order to see that they fail any reasonable criterion, and indeed non-validity and satisfiability checking are useful sanity checks [28]. The validity criterion is a special case of a weaker criterion, in which a specification fails if we can mutate it and get a simpler, equivalent specification. For tautologies, the mutation yields the specification *true*. Our criteria use less aggressive mutations, and are inspired by the concept of vacuity in model checking. We say that a specification φ is inherently vacuous if φ is equivalent to $\forall x.\varphi[\psi \leftarrow x]$, for some subformula ψ of φ. For example, the specification $\mathsf{G}\,(busy \rightarrow \mathsf{F}\,grant) \wedge \mathsf{G}\,(\neg busy \rightarrow \mathsf{F}\,grant)$ is inherently vacuous, as it is equivalent to the specification $\forall x.\mathsf{G}\,(x \rightarrow \mathsf{F}\,grant) \wedge \mathsf{G}\,(\neg x \rightarrow \mathsf{F}\,grant)$, which is equivalent to $\mathsf{G}\,\mathsf{F}\,grant$. This approach leads to a PSPACE decision procedure for inherent vacuity for LTL specifications, by reducing it to the satisfiability problem for LTL.

As described above, our first approach for defining inherent vacuity is based on the definition of vacuity in model checking, and it adopts the idea of applying mutations to the specification. With no system to check the mutated specification with respect to, our first approach requires the mutated specification to be equivalent to the original one. Our second approach for defining inherent vacuity, also based on the definition of vacuity in model checking, quantifies the missing context (that is, the system) universally. Thus, according to the second approach, a specification φ is inherently vacuous if φ is satisfied vacuously in all systems that satisfy it. Note that in contrast to the first definition, the definition does not require the same subformula not to affect the satisfaction of the specification in all systems. Keeping in mind the trade-off between abstraction and vacuity, one may welcome specifications that are vacuously satisfied according to the second approach yet have no single subformula that does not affect the satisfaction in all systems. We show, however, that the second approach coincides with the first one. Thus, a specification φ is satisfied vacuously in all systems that satisfy it iff φ is equivalent to some mutation of it.

The above two approaches, and the encouraging fact they coincide, set the base to our framework for inherent vacuity. Experience with vacuity in model checking has led to the conclusion that there is no single definition of vacuity that is superior to all others, and various definitions are used in practice [3, 20, 11, 6, 2, 21, 15, 4, 7, 31]. Our framework refines the definition of inherent vacuity to account not only for the different definitions of vacuity in model checking, but also for the different settings in which property-based design is used (closed vs. open systems), the goal of the designer (tightening the specification or only cleaning it), and the polarity of the vacuity (strengthening vs. weakening of the formula). Thus, we do not offer a single definition of inherent vacuity, but, rather, offer a general framework in which the user can choose the parameters that best suit the application. We view the main practical contribution of the paper in the setting of open systems and temporal synthesis. As discussed above, the problem of eliciting specifications that faithfully capture the designer intent is of great importance in this setting. Still, the only sanity check that is now used for a specification of an open system is its *realizability*, which is analogous to satisfiability in the setting of closed systems, and checks that there is at least one open system that satisfies the specification [23].

We show that in all variations, the two approaches to defining inherent vacuity coincide. We study the problem of deciding whether an LTL formula is inherently vacuous according to the various criteria and settings. We show that the problem is related to the basic problem in the corresponding setting. Thus, for the setting of closed system, the problem can be solved in PSPACE, just like the satisfiability problem [29], while for the setting of open systems, the problem can be solved in 2EXPTIME, just like the realizability problem [26]. Thus, detection of inherent vacuity is not harder than the most basic quality checks for specifications. We provide many examples for inherent vacuity and argue for the likelihood of encountering inherent vacuity in real life specifications.

2 Inherent Vacuity

In this section we define inherent vacuity for LTL formulas and study basic properties of the definition.

We first review the definition of LTL vacuity in model checking. We assume the reader is familiar with the syntax and the semantics of LTL. The semantic approach to LTL vacuity in model checking [2] considers LTL formulas augmented with universal quantification over atomic propositions. Recall that an LTL formula over a set AP of atomic propositions is interpreted over computations of the form $\pi = \pi_0, \pi_1, \pi_2, \ldots$, with $\pi_i \subseteq AP$. The computation then satisfies a formula of the form $\forall x.\varphi$, where φ is an LTL formula and x is an atomic proposition, if φ is satisfied in all the computations that agree with π on all the atomic propositions except (maybe) x. Thus, $\pi \models \forall x.\varphi$ iff $\pi' \models \varphi$ for all $\pi' = \pi'_0, \pi'_1, \pi'_2, \ldots$ such that $\pi'_i \cap (AP \setminus \{x\}) = \pi_i \cap (AP \setminus \{x\})$ for all $i \geq 0$. As with LTL, a Kripke structure $\mathcal{K}$ satisfies $\forall x.\varphi$ if all computations of $\mathcal{K}$ satisfy $\forall x.\varphi$. For two LTL formulas φ and φ', we say that φ and φ' are *equivalent*, denoted $\varphi \equiv \varphi'$, if for every Kripke structure $\mathcal{K}$, we have that $\mathcal{K} \models \varphi$ iff $\mathcal{K} \models \varphi'$.

Given a Kripke structure $\mathcal{K}$ and a formula φ satisfied in $\mathcal{K}$, we say that a subformula ψ of φ *does not affect the satisfaction of* φ *in* $\mathcal{K}$ if $\mathcal{K}$ also satisfies the formula $\forall x.\varphi[\psi \leftarrow x]$, in which ψ is replaced by a universally quantified fresh proposition [2]. Intuitively, this means that $\mathcal{K}$ satisfies φ even with the most challenging assignments to ψ. We refer to the formula $\forall x.\varphi[\psi \leftarrow x]$ as the *ψ-strengthening* of φ. Finally, a formula φ is *vacuously satisfied in* $\mathcal{K}$ if φ has a subformula that does not affect its satisfaction in $\mathcal{K}$.

In the context of inherent vacuity, the Kripke structure $\mathcal{K}$ is not given. We are only given the formula φ, and we seek some quality criteria that would indicate the likelihood of φ to be satisfied vacuously. Below we describe two natural approaches to defining inherent vacuity, and show that they are, in fact, equivalent.

The first approach to defining inherent vacuity is based on mutating the formula. The idea is that if a syntactic manipulation on the formula, which typically changes the semantics of the formula, yields an equivalent formula, then something is inherently vacuous in the given formula.

Definition 1. *We say that an* LTL *formula φ is* inherently vacuous by mutation *if there exists a subformula ψ of φ such that $\varphi \equiv \forall x.\varphi[\psi \leftarrow x]$. That is, φ is equivalent to its ψ-strengthening. We then say that φ is inherently vacuous by mutation* with witness *ψ.*

Example 1. The formula $\varphi = \mathsf{F}\,(\mathit{grant} \vee \mathit{fail}) \vee \mathsf{X}\,\mathit{fail}$ is inherently vacuous by mutation, as it is equivalent to its $\mathsf{X}\,\mathit{fail}$-strengthening $\forall x.\mathsf{F}\,(\mathit{grant} \vee \mathit{fail}) \vee x$. To see this, note that both φ and its $\mathsf{X}\,\mathit{fail}$-strengthening are equivalent to $\mathsf{F}\,(\mathit{grant} \vee \mathit{fail})$.

The formula $\varphi = (\neg busy \wedge (busy \mathsf{U}\, ack)) \vee (busy \wedge ack)$ is inherently vacuous by mutation, as it is equivalent to its $busy$-strengthening $\forall x.\varphi[busy \leftarrow x]$. To see this, note that both φ and its $busy$-strengthening are equivalent to the formula ack.

Remark 1. The purpose of our examples is to show patterns for inherently vacuous specs. Thus, while we do not expect designers to write the specifications in the examples, where the vacuity is obvious, such patterns do appear in real life specifications. Indeed, there, the specifications are more involved, and it is hard to keep track of all relations among the propositions induced by a specification. We discuss this issue in detail in Section 4.

As we show later in Theorem 2, defining inherent vacuity by means of mutations enables us to reduce the problem of deciding whether a given specification is inherently vacuous to the satisfiability problem for LTL.

It is not hard to see that Definition 1 is equivalent to a definition in which a formula is inherently vacuous if there exists a subformula ψ of φ such that ψ does not affect the satisfaction of φ in all Kripke structures that satisfy it. Formally, for every system $\mathcal{K}$ such that $\mathcal{K} \models \varphi$, also $\mathcal{K} \models \forall x.\varphi[\psi \leftarrow x]$.

One may find Definition 1 too restrictive, as it focuses on a single subformula of the specification. The second approach to defining inherent vacuity addresses this point, by starting with the definition of vacuity and quantifying the missing context (that is, the system) universally, without restricting attention to a single subformula. Formally, we have the following.

Definition 2. *We say that an* LTL *formula φ is* inherently vacuous by model *if for every Kripke structure $\mathcal{K}$, if $\mathcal{K} \models \varphi$, then $\mathcal{K}$ satisfies φ vacuously.*

There is a trade-off between the level of abstraction that a specification enjoys and the level of detail in which it describes the system. Thus, one may tolerate formulas that are vacuously satisfied yet have no single subformula to blame, and find this second approach too unrestricted. As we show now, however, in the setting of nondeterministic Kripke structures, the two approaches we defined coincide. Formally, we have the following.

Theorem 1. *An* LTL *specification φ is inherently vacuous by mutation iff φ is inherently vacuous by model.*

Proof: For the first direction, assume that φ is inherently vacuous by mutation. Then there is a subformula ψ of φ such that $\varphi \equiv \forall x.\varphi[\psi \leftarrow x]$. Accordingly, for every Kripke structure $\mathcal{K}$, if $\mathcal{K} \models \varphi$, then $\mathcal{K} \models \forall x.\varphi[\psi \leftarrow x]$, and so $\mathcal{K}$ satisfies φ vacuously. Thus, φ is inherently vacuous by model.

For the second direction, assume that φ is inherently vacuous by model, and assume by way of contradiction that φ is not inherently vacuous by mutation. Then there exists no single subformula ψ of φ such that $\varphi \equiv \forall x.\varphi[\psi \leftarrow x]$. Consider the alternative definition to vacuity by mutation. Then there is no single subformula ψ of φ such that ψ does not affect the satisfaction of φ in every Kripke structure $\mathcal{K}$ that satisfies it.

Let k be the number of subformulas that φ has. By the assumption, for every candidate subformula ψ_i of φ, with $1 \leq i \leq k$, there is a Kripke structure $\mathcal{K}_i$ that satisfies φ (and hence, as φ is inherently vacuous by model, satisfies φ vacuously), but $\mathcal{K}_i \not\models \forall x.\varphi[\psi_i \leftarrow x]$. Let $\mathcal{K} = \mathcal{K}_1 \cup \mathcal{K}_2 \cup \cdots \cup \mathcal{K}_k$ be the disjoint union of $\mathcal{K}_1, \mathcal{K}_2, \ldots, \mathcal{K}_k$. Note that the set of initial states of $\mathcal{K}$ is the union of the sets of initial states in all structures. By the semantics of LTL, the Kripke structure $\mathcal{K}$ satisfies φ. Since φ is inherently vacuous by model, $\mathcal{K}$ satisfies φ vacuously. Let ψ_i be a subformula that does not affect φ in $\mathcal{K}$. By the semantics of LTL, the subformula ψ_i does not affect φ also in $\mathcal{K}_i$, and we reached a contradiction. □

Remark 2. A deterministic Kripke structure is a Kripke structure with a single initial state in which each state has exactly one successor. If we restrict attention to deterministic Kripke structures, then Definitions 1 and 2 do not coincide. That is, there exist formulas that are inherently vacuous by model but are not inherently vacuous by mutation. For example, consider the formula $\varphi = p \vee q$. Every deterministic Kripke structure that satisfies φ has its (single) initial state labeled either by p or by q or by both, and thus it satisfies φ vacuously. On the other hand, φ is not equivalent to any strengthening of it. Since deterministic Kripke structures are not an interesting model for a system (as they induce a single computation rather than a set of computations), we continue with the nondeterministic setting.

So, the definitions that follow from the two different approaches coincide, and we use the term *inherent vacuity* to refer to either of them. We now study the complexity of detection of inherent vacuity.

Theorem 2. *Given an* LTL *formula φ and a subformula ψ of φ, deciding whether φ is inherently vacuous with witness ψ is PSPACE-complete.*

Proof: We start with the upper bound. Consider an LTL formula φ. For every subformula ψ, it holds that $\forall x.\varphi[\psi \leftarrow x]$ implies φ. Therefore, checking whether φ is inherently vacuous with witness ψ amounts to checking whether φ implies $\forall x.\varphi[\psi \leftarrow x]$. This is done by checking the satisfiability of $\varphi \wedge \exists x.\neg\varphi[\psi \leftarrow x]$, which is satisfiable iff $\varphi \wedge \neg\varphi[\psi \leftarrow x]$ is satisfiable. The latter is an LTL formula, whose satisfiability can be checked in PSPACE. For the lower bound, it is easy to see that φ is inherently vacuous with witness φ iff $\varphi \equiv$ *false*, thus PSPACE-hardness follows from the PSPACE-hardness of LTL satisfiability. □

Now, since a formula φ is inherently vacuous iff it is inherently vacuous with witness ψ for some subformula ψ of φ, the upper bound in Theorem 2 implies the following.

Corollary 1. *The problem of deciding whether an* LTL *formula is inherently vacuous can be solved in polynomial space.*

We note that the lower bound for the problem of deciding inherent vacuity is open. The difficulties in proving a PSPACE lower bound are similar to the ones encountered in studying the complexity of vacuity detection in model checking (a PSPACE upper bound is known, yet a lower bound is open [2]).

3 A Framework for Inherent Vacuity

In Section 2 we defined inherent vacuity. Experience with vacuity in model checking has led to the conclusion that there is no single definition of vacuity that is superior to all others, and various definitions are used in practice. In this section we refine the definition of inherent vacuity to account not only for the different definitions of vacuity in model checking, but also for the different settings in which property-based design is used. The refinement is based on adding parameters that refer to the type of vacuity, the context in which the specification is used, the goal of the designer, and the polarity of the mutation. It turns out that the equivalence of the two approaches to the definition of inherent vacuity is maintained in all settings. Thus, our lifting of vacuity in model checking to inherent vacuity is robust, in the sense that it works for the many contexts in which vacuity may be checked. We elaborate on the parameters below, and we first describe them for the approach that defines inherent vacuity by mutation. As in Section 2, this approach is the basis to decision procedures for inherent vacuity.

3.1 The Parameters of the Framework

Vacuity Type Recall that a formula φ is inherently vacuous by mutation if φ is equivalent to a ψ-strengthening of it, for some subformula ψ of φ. The definition of ψ-strengthening is induced from work on vacuity in model checking. Recall that several definitions of vacuity in model checking are studied in the literature: Definitions treating each occurrence of a subformula separately [3, 20] and their extensions (to the modal μ-calculus [11] and to PSL/SVA [6]), definitions treating all occurrences of the same subformula together [2, 14], definitions considering various distinct subformulas [15], definitions considering model checker proofs [21], definitions focusing on a certain type of reasons for vacuity (antecedent [4] and environment [7]), a definition considering vacuity grounds [30, 31], and more. In Section 2, we followed the definition of vacuity in [2]. The first parameter in our framework enables the consideration of other definitions. For example, the semantic-based definition in [2] can be refined according to different semantics of universal quantification of atomic propositions (structure vs. tree; for details see [2]). As another example, in the syntactic-based definition to vacuity in [3, 20], one mutates a single occurrence of a subformula, rather than all occurrences. Formally, given an occurrence σ of a subformula of the formula φ, the σ-strengthening of φ is the formula $\varphi[\sigma \leftarrow \bot]$, obtained by replacing the occurrence σ by *false* if σ is under an even number of negation and by *true* if σ is under an odd number of negation. Other definitions allow mutations of a subset of the occurrences, a subset of subformulas, or a subset of the atomic propositions [15].

Example 2. Consider the formula $\varphi = grant \vee (up \,\mathsf{U}\, grant)$. Clearly, φ is equivalent to $up \,\mathsf{U}\, grant$. Indeed, φ is equivalent to $\varphi[\sigma \leftarrow \bot]$ for σ being the first occurrence of the subformula $grant$. Therefore, the σ-strengthening of φ is equivalent to φ. Note that if we had considered the semantic-based definition as we did in the previous section, the formula would not have been declared inherently vacuous, since it is not equivalent to $\forall x.\varphi[\psi \leftarrow x]$ for any subformula ψ of φ.

Consider the formula $\varphi = (\neg busy \wedge ack) \vee (busy \wedge ack)$. It is easy to see that φ is equivalent to $\forall x.\varphi[busy \leftarrow x]$. Thus, f is inherently vacuous when considering the semantic-based definition for vacuity. On the other hand, φ is not equivalent to $\varphi[\psi \leftarrow \bot]$ for any occurrence of a subformula ψ of φ. Thus, φ is not inherently vacuous when considering the syntactic-based definition of vacuity.

Equivalence Type Again recall that a formula φ is inherently vacuous by mutation if φ is equivalent to a ψ-strengthening of it, for some subformula ψ of φ. The definition of equivalence has to do with the context in which φ is to be used. In the context of closed systems, the semantics of φ is defined with respect to Kripke structures, thus $\varphi \equiv \varphi'$ if for all Kripke structures $\mathcal{K}$, we have that $\mathcal{K} \models \varphi$ iff $\mathcal{K} \models \varphi'$. In the context of open systems, the semantics of φ is defined according to *transducers*, and the atomic propositions in φ are partitioned into input and output signals. Before we turn to show that this requires a different notion of equivalence, let us define transducers formally.

A transducer is a tuple $\mathcal{T} = \langle \mathcal{I}, \mathcal{O}, S, \eta_0, \eta, L \rangle$, where $\mathcal{I}$ is a set of input signals, $\mathcal{O}$ is a set of output signals, S is a set of states, $\eta_0 : 2^I \to 2^S \setminus \emptyset$ is an initial transition function, $\eta : S \times 2^{\mathcal{I}} \to 2^S \setminus \emptyset$ is a transition function, and $L : S \to 2^{\mathcal{O}}$ is a labeling function. Note that $\mathcal{T}$ is responsive, in the sense that η_0 and η provide at least one initial state and successor state, respectively, for each input letter. A *run* of $\mathcal{T}$ on an input sequence $i_0 \cdot i_1 \cdot i_2 \cdots \in (2^{\mathcal{I}})^\omega$ is a sequence $s_0, s_1, s_2, \ldots$ of states such that $s_0 \in \eta_0(i_0)$ and $s_{j+1} \in \eta(s_j, i_{j+1})$ for all $j \geq 0$. A computation $w \in (2^{\mathcal{I} \cup \mathcal{O}})^\omega$ is *generated* by $\mathcal{T}$ if $w = (i_0 \cup o_0), (i_1 \cup o_1), (i_2 \cup o_2) \ldots$ is such that there is a run $s_0, s_1, s_2, \ldots$ of $\mathcal{T}$ on $i_0 \cdot i_1 \cdot i_2 \cdots$ for which $o_j = L(s_j)$ for all $j \geq 0$. Note that we consider nondeterministic transducers. A transducer $\mathcal{T}$ *realizes* an LTL formula φ, denoted $\mathcal{T} \models \varphi$, if all computations of $\mathcal{T}$ satisfy φ. We say that an LTL formula φ is *realizable* if there is a transducer that realizes φ. The synthesis problem is to construct, given an LTL formula φ, a transducer that realizes φ.

Note that a nondeterministic transducer may have several runs on an input sequence. In a *deterministic* transducer, for all $i \in 2^I$ and $s \in S$, we have $|\eta_0(i)| = 1$ and $|\eta_0(s, i)| = 1$. Thus, a deterministic transducer has a single run on each input sequence. Unlike Kripke structures, which can be viewed as transducers with $I = \emptyset$, here the deterministic model is of interest, and it induces a set of computations – one for each input sequence.

As discussed in [13], equivalence with respect to transducers is weaker than equivalence with respect to Kripke structures. Formally, given two LTL formulas φ and φ' both over signals I and O, we say that φ and φ' are *equivalent in the context of open systems* (o-equivalent, for short), denoted $\varphi \equiv_o \varphi'$, if for every transducer $\mathcal{T}$ with input I and output O, we have that $\mathcal{T} \models \varphi$ iff $\mathcal{T} \models \varphi'$. For the sake of uniformity, we now use $\equiv_c$ to denote equivalence in the context of closed systems (c-equivalence, for short; what used to be $\equiv$ in Section 2). It is not hard to see that for every two LTL formulas φ and φ', we have that $\varphi \equiv_c \varphi'$ implies $\varphi \equiv_o \varphi'$. By [13], implication on the other direction does not hold. For example, a specification φ that restricts the input in some satisfiable way is unrealizable, and hence $\varphi \equiv_o$ *false*, yet $\varphi \not\equiv_c$ *false* as φ is satisfiable.

Example 3. Consider the formula $\varphi = [\mathsf{G}\,(busy \to \mathsf{F}\,(grant \wedge \neg busy))] \vee \mathsf{G}\,\mathsf{F}\,grant$, where $busy$ is an input signal and $grant$ is an output signal. The formula is inherently vacuous in the context of open systems, but is not inherently vacuous in the context of closed systems (considering the semantic-based definition of vacuity in both). To see this, consider the subformula $\psi = \mathsf{G}\,(busy \to \mathsf{F}\,(grant \wedge \neg busy))$ in φ. Since ψ imposes restrictions on the input signal, it is unrealizable. Hence, φ is o-equivalent to its ψ-strengthening, which is equivalent to $\mathsf{G}\,\mathsf{F}\,grant$. On the other hand, φ is not c-equivalent to any strengthening of it.

Tightening Type The third parameter refers to the goal of the designer. In early stages of the design, the designer may be interested in detecting cases where the formula can be mutated to a formula that is strictly stronger, yet is still satisfiable (or, in the context of open systems, still realizable). The third parameter indicates whether the mutated formula has to be equivalent to the original formula or only has to maintain its satisfiability or realizability. Note that tightening of the specification to a specification that is strictly stronger and yet maintains its satisfiability or realizability is useful mainly in the context of synthesis, where it suggests that the specification for the system should be tightened. We demonstrate this in the two examples below. As Example 4 shows, there are formulas that can be mutated in a way that preserves realizability, yet cannot be mutated to an equivalent formula. Thus, inherent vacuity for such formulas is detected only with the third parameter indicating that we are looking for a mutation that maintains realizability.

Example 4. Consider the formula $\varphi = (busy \vee ack) \rightarrow \mathsf{X}\, grant$ where $busy$ is an input signal and where ack and $grant$ are output signals. There exists no sub-formula ψ of φ such that its ψ-strengthening is equivalent to φ. On the other hand, the ack-strengthening of φ, which is equivalent to $\varphi' = busy \rightarrow \mathsf{X}\, grant$, is realizable.

Example 5. Consider an open system with input req and outputs $grant_1$ and $grant_2$, and the formula $\varphi = \mathsf{G}\,(req \rightarrow \mathsf{X}\,(grant_1 \vee grant_2)) \wedge \mathsf{G}\,(req \rightarrow \mathsf{F}\, grant_2)$. The σ-strengthening of φ, for σ being the first occurrence of $grant_2$ is still realizable. Thus, the formula is inherently vacuous when the tightening type is "realizability preservation" (and the vacuity type is mutation of a single occurrence). The designer may want to tighten the unbounded delay in the second conjunct as not to overlap the first conjunct.

Consider the formula $\varphi' = \mathsf{G}\,(req \rightarrow \mathsf{F}\,(grant_1 \vee grant_2))$. The $grant_2$-strengthening of φ is $\mathsf{G}\,(req \rightarrow \mathsf{F}\, grant_1)$, which is still realizable. Thus, φ' is inherently vacuous according to the same criteria as above. Note that the same holds for the $grant_1$-strengthening of φ. In this case, however, it is not clear that the $grant_2$-strengthening or the $grant_1$-strengthening of φ are the desired formulas, as they both ignore a particular type of grant. The information from the check is still useful, as the specifier may conclude that the formula he has to use is $\mathsf{G}\,(req \rightarrow [\mathsf{F}\, grant_1 \wedge \mathsf{F}\, grant_2])$, which is the conjunction of the two strengthenings.

Polarity Type In the context of model checking, a formula is checked for vacuity only after it has been verified to hold on the system. The vacuity check then examines whether a mutation that strengthens the formula still holds on the system Clearly, it makes no sense to model check a weaker formula, as it is guaranteed to be satisfied.[6] In the context of property-based design, however, we may consider mutations that strengthen the formula as well as mutations that weaken it. Indeed, in the extreme case a mutation that weakens the formula is the formula *true*, as all models satisfy it, and clearly a formula that is equivalent to *true* (yet is syntactically different) is inherently vacuous.

The fourth parameter in our framework refers to the polarity of the mutation, and indicates whether we compare the formula with its ψ-strengthening or ψ-weakening. The ψ-weakening of a formula φ is defined in a manner dual to its ψ-strengthening. For example, dualizing the definition of vacuity in [2], the ψ-weakening of φ is $\exists x.\varphi[\psi \leftarrow x]$, in which all the occurrences of ψ are replaced by an existentially quantified proposition. Likewise, dualizing the definition of vacuity in [20], the σ-weakening of φ, for an occurrence σ of some subformula, is $\varphi[\sigma \leftarrow \top]$, which is obtained from φ by replacing the occurrence σ by *true* if σ is of positive polarity and by *false* if σ is of negative polarity.

[6] Nevertheless, [15] shows how one can benefit from checking the vacuity of negations of formulas that pass.

Example 6. Consider the formula $\varphi = (\mathsf{F}\, grant) \wedge (\mathsf{X}\, grant)$. Clearly, the formula φ is equivalent to its second conjunct, namely, $\mathsf{X}\, grant$, which is the $\mathsf{F}\, grant$-weakening of φ. On the other hand, there is no subformula of φ or an occurrence of a subformula ψ such that φ is equivalent to its ψ-strengthening.

Consider the formula $\varphi = (\mathsf{F}\, grant) \vee (\mathsf{X}\, grant)$. Since φ is equivalent to its $\mathsf{X}\, grant$-strengthening, namely, $\mathsf{F}\, grant$, we have that φ is inherently vacuous in a definition that considers strengthening. On the other hand, there exists no subformula of φ or an occurrence of a subformula ψ such that φ is equivalent to its ψ-weakening.

This shows there exist formulas that are inherently vacuous according to weakening but not according to strengthening and vice versa.

Example 7. In [9], the authors consider specifications of the form $\varphi = \bigwedge_{i \in I} \varphi_i$ and study how to detect redundant conjuncts. Formally, φ_j is redundant if φ is equivalent to $\bigwedge_{i \in I \setminus \{j\}} \varphi_i$. Note that this is a special case of our inherent vacuity when considering weakening of the mutation (and the syntactic-based definition of vacuity).

For example, the formula $\varphi = (wait\, \mathsf{U}\, busy) \wedge \mathsf{F}\,(wait \vee busy)$ is inherently vacuous according to this criterion. Indeed, φ is equivalent to $(wait\, \mathsf{U}\, busy)$, which is its σ-weakening, for $\sigma = \mathsf{F}\,(wait \vee busy)$.

3.2 Working with the Different Parameters

In order to describe the different parameters, we use the term φ is *inherently vacuous (by mutation) of type* $(\mathrm{V}, \mathrm{E}, \mathrm{T}, \mathrm{P})$, where $\mathrm{V} \in \{s_{\mathrm{V}}, m_{\mathrm{V}}\}$ denotes the vacuity type (single or multiple occurrences), $\mathrm{E} \in \{c_{\mathrm{E}}, o_{\mathrm{E}}\}$ denotes the equivalence type (closed or open systems), $\mathrm{T} \in \{e_{\mathrm{T}}, p_{\mathrm{T}}\}$ denotes the tightening type (to an equivalent one or to one that preserves satisfaction), and $\mathrm{P} \in \{s_{\mathrm{P}}, w_{\mathrm{P}}\}$ denotes the type of polarity (strengthening or weakening). For example,

– φ is inherently vacuous of type $(m_{\mathrm{V}}, o_{\mathrm{E}}, e_{\mathrm{T}}, s_{\mathrm{P}})$ if $\varphi \equiv_o \forall x.\varphi[\psi \leftarrow x]$, for some subformula ψ of φ.
– φ is inherently vacuous of type $(s_{\mathrm{V}}, c_{\mathrm{E}}, e_{\mathrm{T}}, w_{\mathrm{P}})$ if $\varphi \equiv_c \varphi[\sigma \leftarrow \top]$, for some occurrence σ of a subformula of φ.
– φ is inherently vacuous of type $(m_{\mathrm{V}}, o_{\mathrm{E}}, p_{\mathrm{T}}, s_{\mathrm{P}})$ if $\forall x.\varphi[\psi \leftarrow x]$ is realizable for some subformula ψ of φ.

Note that inherent vacuity discussed in Section 2 is of type $(m_{\mathrm{V}}, c_{\mathrm{E}}, e_{\mathrm{T}}, s_{\mathrm{P}})$. Note also that the parameters are orthogonal to each other. An exception is the tightening type and its polarity: if T is p_{T}, then P must be s_{P}.

For uniformity, we use the V, E, and T parameters also in other notations. In particular, a V-subformula of φ is a subformula if $\mathrm{V} = m_{\mathrm{V}}$ and is an occurrence of a subformula if $\mathrm{V} = s_{\mathrm{V}}$. Likewise, a E-system is a Kripke structure when $\mathrm{E} = c_{\mathrm{E}}$ and is a transducer when $\mathrm{E} = o_{\mathrm{E}}$. Finally, a formula that is E-satisfiable is satisfiable when $\mathrm{E} = c_{\mathrm{E}}$ and is realizable when $\mathrm{E} = o_{\mathrm{E}}$.

Example 8. Mutating a single occurrence of a subformula may not detect inherent vacuity that originates from the relations between different parts of the formula. For example, the formula $\varphi = (wait \wedge busy) \vee (wait \wedge \neg busy)$ is inherently vacuous of types $(m_{\mathrm{V}}, c_{\mathrm{E}}, e_{\mathrm{T}}, s_{\mathrm{P}})$ and $(m_{\mathrm{V}}, c_{\mathrm{E}}, e_{\mathrm{T}}, w_{\mathrm{P}})$ but is not inherently vacuous of type $(s_{\mathrm{V}}, c_{\mathrm{E}}, e_{\mathrm{T}}, s_{\mathrm{P}})$ or $(s_{\mathrm{V}}, c_{\mathrm{E}}, e_{\mathrm{T}}, w_{\mathrm{P}})$.

Indeed, the $busy$-strengthening of φ and the $busy$-weakening of it, which are equivalent to the formula $wait$, are equivalent to φ. However, there is no single occurrence σ of a subformula ψ such that $\varphi[\sigma \leftarrow \bot]$ or $\varphi[\sigma \leftarrow \top]$ is equivalent to $wait$.

On the other hand, mutating all occurrences may not detect local problems that are covered by other parts of the formula. For example, the formula $\varphi = (\mathsf{G}\, high) \vee (high \rightarrow \mathsf{F}\,\mathsf{G}\, high)$ is inherently vacuous of type $(s_{\text{V}}, c_{\text{E}}, e_{\text{T}}, s_{\text{P}})$ but is not inherently vacuous of type $(m_{\text{V}}, c_{\text{E}}, e_{\text{T}}, s_{\text{P}})$. Indeed, φ is equivalent to its σ-strengthening, for σ being the first occurrence of $\mathsf{G}\, high$. However, there is no subformula ψ of φ such that φ is equivalent to its ψ-strengthening.

Theorem 3 below summarizes the relations among the various types of inherent vacuity. The first implication follows from the implications of o-equivalence by c-equivalence [13]. The second implication follows from the fact we restrict attention to E-satisfiable formulas. Finally, all the incomparability results are demonstrated in the examples.

Theorem 3. *Let* $\text{V} \in \{s_{\text{V}}, m_{\text{V}}\}$, $\text{E} \in \{c_{\text{E}}, o_{\text{E}}\}$, $\text{T} \in \{e_{\text{T}}, p_{\text{T}}\}$, *and* $\text{P} \in \{s_{\text{P}}, w_{\text{P}}\}$.

1. *Inherent vacuity of type* $(\text{V}, c_{\text{E}}, \text{T}, \text{P})$ *implies inherent vacuity of type* $(\text{V}, o_{\text{E}}, \text{T}, \text{P})$. *Implication in the other direction does not hold.*
2. *For* E*-satisfiable formulas, inherent vacuity of type* $(\text{V}, \text{E}, e_{\text{T}}, s_{\text{P}})$ *implies inherent vacuity of type* $(\text{V}, \text{E}, p_{\text{T}}, s_{\text{P}})$. *Implication in the other direction does not hold.*
3. *Inherent vacuity of type* $(m_{\text{V}}, \text{E}, \text{T}, \text{P})$ *is incomparable with inherent vacuity of type* $(s_{\text{V}}, \text{E}, \text{T}, \text{P})$.
4. *Inherent vacuity of type* $(\text{V}, \text{E}, e_{\text{T}}, s_{\text{P}})$ *is incomparable with inherent vacuity of type* $(\text{V}, \text{E}, e_{\text{T}}, w_{\text{P}})$.

We now turn to discuss the complexity of detecting inherent vacuity in the different settings. The following theorem shows that the problem of deciding whether a given LTL formula is inherently vacuous of various types we have defined, is not more difficult than the corresponding E-satisfiability problem for LTL.

Theorem 4. *Let* $\text{V} \in \{s_{\text{V}}, m_{\text{V}}\}$, $\text{E} \in \{c_{\text{E}}, o_{\text{E}}\}$, $\text{T} \in \{e_{\text{T}}, p_{\text{T}}\}$, *and* $\text{P} \in \{s_{\text{P}}, w_{\text{P}}\}$. *Given an* LTL *formula* φ *and a* V*-subformula* ψ *of* φ, *deciding whether* φ *is inherently vacuous of type* $(\text{V}, \text{E}, \text{T}, \text{P})$ *with witness* ψ *is PSPACE-complete for* $\text{E} = c_{\text{E}}$ *(except when* $\text{V} = m_{\text{V}}$ *and* $\text{T} = p_{\text{T}}$, *in which case it is in EXPSPACE-complete) and is 2EXPTIME-complete for* $\text{E} = o_{\text{E}}$.

Proof: For the upper bound, all cases with $\text{T} = e_{\text{T}}$ are reducible to checking the E-equivalence of φ and its mutation. When $\text{V} = m_{\text{V}}$, the mutation may involve universal or existential quantification of atomic propositions. Still, only one direction of the implication between φ and the mutation should be checked (the other direction always holds), and fortunately, it is the direction that can be reduced to LTL E-implication. The upper bounds then follow from the PSPACE and 2EXPTIME complexities for LTL closed and open implication, respectively [29, 17].

When $\text{T} = p_{\text{T}}$, we have to check whether the ψ-strengthening of φ is E-satisfiable. When $\text{V} = s_{\text{V}}$, the ψ-strengthening is an LTL formula, and again the upper bound follows from the known PSPACE and 2EXPTIME complexities for LTL E-implication. When $\text{V} = m_{\text{V}}$, the ψ-strengthening involves universal quantification of atomic propositions. When

$\mathrm{E} = c_{\mathrm{E}}$, the problem reduces to the satisfiability problem of LTL augmented with universal quantification over atomic propositions, this leads to an EXPSPACE complexity [32]. When $\mathrm{E} = o_{\mathrm{E}}$, we can check in 2EXPTIME the realizability of $\neg\varphi[\psi \leftarrow x]$ in a dual setting. By the determinacy of realizability, the latter is realizable iff $\forall x.\varphi[\psi \leftarrow x]$ is unrealizable.

For the lower bound, taking ψ to be φ reduces E-satisfiability to inherent vacuity, thus the lower bound holds from the known PSPACE-hardness and 2EXPTIME-hardness for LTL satisfiability and realizability, respectively [29, 26]. An exception is inherent vacuity of type $(m_{\mathrm{V}}, c_{\mathrm{E}}, p_{\mathrm{T}}, s_{\mathrm{P}})$, to which we reduce satisfiability of LTL augmented with universal quantification over atomic propositions, which is known to be EXPSPACE-hard [32]. $\square$

Corollary 2. *Let* $\mathrm{V} \in \{s_{\mathrm{V}}, m_{\mathrm{V}}\}$, $\mathrm{T} \in \{e_{\mathrm{T}}, p_{\mathrm{T}}\}$, *and* $\mathrm{P} \in \{s_{\mathrm{P}}, w_{\mathrm{P}}\}$. *The problem of deciding whether an* LTL *formula is inherently vacuous of type* $(\mathrm{V}, c_{\mathrm{E}}, \mathrm{T}, \mathrm{P})$ *can be solved in polynomial space (except for type* $(m_{\mathrm{V}}, c_{\mathrm{E}}, p_{\mathrm{T}}, s_{\mathrm{P}})$*, which requires exponential space). The problem of deciding whether an* LTL *formula is inherently vacuous of types* $(\mathrm{V}, o_{\mathrm{E}}, \mathrm{T}, \mathrm{P})$ *can be solved in doubly exponential time.*

Note that Theorem 2 and Corollary 1 are a special case of Theorem 4 and Corollary 2.

Corollary 2 shows that detection of inherent vacuity, while being more informative than detection of satisfiability or realizability, which are used in property-based design [28], is not harder than these basic problems.[7]

Having refined the notion of inherent vacuity by mutations, we now turn to refine the alternative approach of inherent vacuity by model. Note that since the definition of inherent vacuity by model refers to vacuous satisfaction of φ in a model that satisfies φ, it is not interesting to consider weakening of formulas. Thus, when we compare the notions of inherent vacuity by mutation and by model, the fourth parameter has to be s_{P}.

Definition 3. *Consider an* LTL *formula* φ. *For* $\mathrm{V} \in \{s_{\mathrm{V}}, m_{\mathrm{V}}\}$, $\mathrm{E} \in \{c_{\mathrm{E}}, o_{\mathrm{E}}\}$, *and* $\mathrm{T} \in \{e_{\mathrm{T}}, p_{\mathrm{T}}\}$, *we say that*

- φ *is inherently vacuous by model of type* $(\mathrm{V}, \mathrm{E}, e_{\mathrm{T}}, s_{\mathrm{P}})$ *if* φ *is satisfied vacuously in all* E*-systems that satisfy* φ.
- φ *is inherently vacuous by model of type* $(\mathrm{V}, \mathrm{E}, p_{\mathrm{T}}, s_{\mathrm{P}})$ *if* φ *is satisfied vacuously in some* E*-system that satisfies* φ.

Note that for type $(m_{\mathrm{V}}, c_{\mathrm{E}}, e_{\mathrm{T}}, s_{\mathrm{P}})$*, the definition coincides with Definition 2.*

The following theorem, extending Theorem 1, states that the two approaches for inherent vacuity coincide all over the framework.

Theorem 5. *For all* $\mathrm{V} \in \{s_{\mathrm{V}}, m_{\mathrm{V}}\}$, $\mathrm{E} \in \{c_{\mathrm{E}}, o_{\mathrm{E}}\}$, *and* $\mathrm{T} \in \{e_{\mathrm{T}}, p_{\mathrm{T}}\}$, *an* LTL *formula* φ *is inherently vacuous by mutation of type* $(\mathrm{V}, \mathrm{E}, \mathrm{T}, s_{\mathrm{P}})$ *iff* φ *is inherently vacuous by model of type* $(\mathrm{V}, \mathrm{E}, \mathrm{T}, s_{\mathrm{P}})$.

[7] The exception of $(m_{\mathrm{V}}, c_{\mathrm{E}}, p_{\mathrm{T}}, s_{\mathrm{P}})$ follows from the universal quantification of atomic propositions that vacuity type m_{V} involves. It suggests that designers that suspect their specification for closed systems should be tightened may prefer to work with vacuity type s_{V}.

Proof: Theorem 1 provides the proof for type $(m_{\text{V}}, c_{\text{E}}, e_{\text{T}}, s_{\text{P}})$. The proof for the other types with $\text{T} = e$ are similar. In fact, in the setting of open systems and transducers, the equivalence is valid even when we consider deterministic transducers. To see this, note that the "only if" direction in the proof of Theorem 1 is based on defining the union of k Kripke structures as a single Kripke structure. While this cannot be done with deterministic Kripke structures, it can be done with deterministic transducers. Indeed, by adding $\lceil \log k \rceil$ input signals that do not appear in the formula, we can define a deterministic initial transition function for the union, in which the added input signals choose a transducer from the union. The rest of the proof is the same as in the case of nondeterministic Kripke structures.

It is left to describe the details for the case $\text{T} = p$, which is different.

Assume that φ is inherently vacuous by mutation of type $(\text{V}, \text{E}, p_{\text{T}}, s_{\text{P}})$. Let ψ be such that the ψ-strengthening of φ is E-satisfiable, and let $\mathcal{S}$ be the E-system that satisfies it. By definition, ψ does not affect the satisfaction of φ in $\mathcal{S}$, thus $\mathcal{S}$ satisfies φ vacuously, and φ is inherently vacuous by model of type $(\text{V}, \text{E}, p_{\text{T}}, s_{\text{P}})$.

For the other direction, if φ is inherently vacuous by model of type $(\text{V}, \text{E}, p_{\text{T}}, s_{\text{P}})$, then there exists a V-subformula ψ that does not affect its satisfaction in some E-system. Then the ψ-strengthening of φ is E-satisfiable, thus φ is inherently vacuous by mutation of type $(\text{V}, \text{E}, p_{\text{T}}, s_{\text{P}})$. $\square$

4 Discussion

We proposed a framework for inherent vacuity — vacuity of specifications without a reference model. We argue that, as has been the case with vacuity in model checking, inherent vacuity is common, and detection of inherent vacuity may significantly improve the specifications and the designer's understanding of it.

In [9], the authors experimented with a real-life block as described in [24]. Its specification consists of 50 formulas. It is shown in [9] that inherent vacuity exists already with the basic definition of redundant conjuncts. Indeed, in a set consisting of 17 formulas, 9 were found to be redundant. Another common source of inherent vacuity, not captured by the definition in [9], is the fact that subformulas that appear in different conjunctions may be related, without the specifier being aware of such a relation. In particular, a formula for the full specification may be written by a group of specifiers, each specifying a different aspect of the design. For example, consider the specification $\varphi = \varphi_1 \wedge \varphi_2 \wedge \vartheta$ where φ_1 is $\mathsf{G}\,(\xi_1 \to \mathsf{F}\,\psi)$, φ_2 is $\mathsf{G}\,(\xi_2 \to \mathsf{F}\,\psi)$, and ϑ is $\mathsf{G}\,(\xi_1 \vee \xi_2)$. Such a specification is classical in the sense that ξ_1 and ξ_2 represent some modes of operation, and ϑ states that the system is always in one of the modes. The formula φ is inherently vacuous as it is equivalent to $\mathsf{G}\,(\xi_1 \vee \xi_2) \wedge \mathsf{G}\,\mathsf{F}\,\psi$. Yet, as different specifiers may have specified φ_1 and φ_2, such a vacuity may not be noticed.

The above phenomenon, of subformulas that are related to each other without the specifier being aware of it, follows from the fact that specifiers often pack complicated properties into a single temporal formula. Moreover, today standard temporal logics (e.g. SVA [33], PSL [25, 12],) provide a mechanism for doing so, by allowing one to name a formula and then relate to it in other formulas. As the referenced subformulas may have many signals in common, inherent vacuity in the obtained formula is likely to occur.

Another reason for finding inherent vacuity is mistakes, either typos or small logical errors done by novice in temporal logic. For example, a typo in the formula $(p \to (\varphi \wedge \psi)) \wedge (\neg p \to (\varphi \wedge \vartheta))$ can result in the formula $(p \to (\varphi \wedge \psi)) \wedge (p \to (\varphi \wedge \vartheta))$, which is inherently vacuous by mutation replacing the second occurrence of φ by *true*. As another example, trying to write the temporal-logic formula for the English specification "If signal *error* is asserted, it will remain asserted forever" a novice might write $\varphi = \mathsf{G}\,(\mathit{error} \to (\mathit{error}\,\mathsf{U}\,\mathit{error}))$ which is a tautology, and inherently vacuous by the mutation $\forall x.\varphi(\mathit{error} \leftarrow x)$. Similarly, consider the English specification "If signal *grant* is not asserted the cycle after a *req*uest, then it cannot be asserted two cycles after the *req*uest." A wrong attempt to formalize it may be $\mathsf{G}\,\neg(\mathit{req} \to \mathsf{X}\,(\neg \mathit{grant} \to \mathsf{X}\,(\mathit{grant})))$. The latter formula is inherently vacuous by the mutation replacing the second occurrence of *grant* by *false*. Similar examples abound.

We note that inherent vacuity does not always imply that the specification should be changed to its simpler mutation, and sometimes the contribution of detecting inherent vacuity is a better understanding of the specification, possibly leading to a change in the specification that is different from replacing it by the mutated specification. To see this, let us consider again the specification $\varphi = \mathsf{G}\,(\xi_1 \to \mathsf{F}\,\psi) \wedge \mathsf{G}\,(\xi_2 \to \mathsf{F}\,\psi) \wedge \mathsf{G}\,(\xi_1 \vee \xi_2)$ discussed above. While φ is equivalent to $\mathsf{G}\,(\xi_1 \vee \xi_2) \wedge \mathsf{G}\,\mathsf{F}\,\psi$, the specifier may prefer to leave the original formula, in case the formula needs to be refined further in later stages of the design (and different refinements may be needed for the different models ξ_i), or in case the assumption that only these two modes are possible is removed. Still, it is useful information for the designer to know that, as is, ψ happens infinitely often, regardless of the mode. Note, however, that in synthesis one would always prefer the simpler mutated specification as it will induce a smaller system in less time.

Acknowledgments

We thank Shoham Ben-David, Hana Chockler, Cindy Eisner and Avigail Orni for helpful comments on an early draft of this paper and for interesting discussions on the subject.

References

1. G. Ammons, D. Mandelin, R. Bodk, and J.R. Larus. Debugging temporal specifications with concept analysis. In *Proc. PLDI 2003*, pages 182–195, 2003.
2. R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M.Y. Vardi. Enhanced vacuity detection for linear temporal logic. In *Proc. 15th CAV*, pages 368–380, 2003.
3. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. *FMSD*, 18(2):141–162, 2001.
4. S. Ben-David, D. Fisman, and S. Ruah. Temporal antecedent failure: Refining vacuity. In *Proc. 18th CONCUR*, LNCS 4703, pages 492–506, 2007.
5. R. Bloem, R. Cavada, I. Pill, M. Roveri, and A. Tchaltsev. RAT: A tool for the formal analysis of requirements. In *Proc. 19th CAV*, LNCS 4590, pages 263–267, 2005.
6. D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M.Y. Vardi. Regular vacuity. In *Proc. 13th CHARME*, LNCS 3725, pages 191–206, 2005.
7. M. Chechik, M. Gheorghiu, and A. Gurfinkel. Finding state solutions to temporal queries. In *Proc. Integrated Formal Methods*, LNCS 4591, pages 273–292, 2007.
8. H. Chockler, O. Kupferman, and M.Y. Vardi. Coverage metrics for temporal logic model checking. In *Proc. 7th TACAS*, LNCS 2031, pages 528 – 542, 2001.

9. H. Chockler and O. Strichman. Easier and more informative vacuity checks. In *Proc. 5th MEMOCODE*, pages 189–198, 2007.
10. K. Claessen. A coverage analysis for safety property lists. In *32nd MFCS*, pages 139–145. IEEE Computer Society, 2007.
11. Y. Dong, B. Sarna-Starosta, C.R. Ramakrishnan, and S.A. Smolka. Vacuity checking in the modal μ-calculus. In *Proc 9th AMAST*, pages 147–162, 2002.
12. C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Springer, 2006.
13. K. Greimel, R. Bloem, B. Jobstmann and M. Vardi. Open Implication. In *Proc. ICALP 2008*, LNCS 5126, pages 361–372.
14. A. Gurfinkel and M. Chechik. Extending extended vacuity. In *Proc. 5th FMCAD*, LNCS 3312, pages 306–321, 2004.
15. A. Gurfinkel and M. Chechik. How vacuous is vacuous. In *Proc. 10th TACAS*, LNCS 2988, pages 451–466, 2004.
16. Y. Hoskote, T. Kam, P.-H Ho, and X. Zhao. Coverage estimation for symbolic model checking. In *Proc. 36st DAC*, pages 300–305, 1999.
17. B. Jobstmann, S. Galler, M. Weiglhofer, and R. Bloem. Anzu: A tool for property synthesis. In *Proc. 19th CAV*, LNCS 4590, pages 258–262, 2007.
18. O. Kupferman. Sanity checks in formal verification. In *Proc. 17th CONCUR*, LNCS 4137, pages 37–51, 2006.
19. O. Kupferman and M.Y. Vardi. Model checking of safety properties. *FMSD*, 19(3):291–314, 2001.
20. O. Kupferman and M.Y. Vardi. Vacuity detection in temporal model checking. *STTT*, 4(2):224–233, 2003.
21. K.S. Namjoshi. An efficiently checkable, proof-based formulation of vacuity in model checking. In *Proc. 16th CAV*, LNCS 3114, pages 57–69, 2004.
22. M. Oberkönig, M. Schickel, and H. Eveking. A quantitative completeness analysis for property-sets. In *Proc. 7th FMCAD*, pages 158–161. IEEE Computer Society, 2007.
23. A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.
24. PROSYD. The Prosyd project on property-based system design. http://www.prosyd.org, 2007.
25. IEEE Standard for Property Specification Language (PSL). IEEE Std 1850$^{\text{TM}}$-2005.
26. R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.
27. M. Roveri. Novel techniques for property assurance. Technical report, PROSYD FP6-IST-507219, 2007.
28. K.Y. Rozier and M.Y. Vardi. LTL satisfiability checking. In *Proc. 14th SPIN Workshop*, LNCS 4595, pages 149–167, 2007.
29. A.P. Sistla and E.M. Clarke. The complexity of propositional linear temporal logic. *Journal of the ACM*, 32:733–749, 1985.
30. M. Samer and H. Veith. Parametrized vacuity. In *Proc. 5th FMCAD*, pages 322–336, 2004.
31. M. Samer and H. Veith. On the notion of vacuous truth. In *Proc. 14th LPAR*, LNCS 4790, pages 2–14, 2007.
32. A.P. Sistla, M.Y. Vardi, and P. Wolper The complementation problem for Büchi automata with applications to temporal logic. In *TCS*, 49:217-237, 1987.
33. Annex E of IEEE Standard for SystemVerilog Unified Hardware Design, Specification, and Verification Language. IEEE Std 1800$^{\text{TM}}$-2005.
34. M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *I& C*, 115(1):1–37, 1994.