2015 22nd International Symposium on Temporal Representation and Reasoning

Semantic Vacuity

Grgur Petric Maretić Mohammad Torabi Dashti David Basin
Department of Computer Science
ETH Zürich
Universitätstrasse 6
Zürich, Switzerland

***Abstract*—The vacuous satisfaction of a temporal formula with respect to a model has been extensively studied in the literature. Although a universally accepted definition of vacuity does not yet exist, all existing proposals generalize, in one way or another, the antecedent failure of an implication to the syntax of a temporal logic. They are therefore syntactic: whether a model vacuously satisfies a formula is affected by semantics-preserving changes to the formula. This leads to inconsistent and counter-intuitive results. We propose an alternative: a semantic definition of vacuity for LTL where either two semantically equivalent LTL formulas are both satisfied vacuously in a model, or neither of them are. Our definition is based on a syntactic-invariant separation of LTL formulas, which gives rise to an algorithm for detecting semantic vacuity using trap properties. We also propose an alternative algorithm for Büchi automata, which can be used to detect the vacuous satisfaction of ω-regular properties as well as LTL formulas. We analyze this algorithm's worst-case complexity and, using real-world examples, demonstrate that semantic vacuity can be efficiently decided in practice.**

I. Introduction

Model checking is a powerful technique for verifying the correctness of systems with respect to their requirements. For instance, if a system is specified as a finite-state Kripke model $\mathcal{M}$ and its requirements are written as a linear-time temporal logic (LTL) formula φ, then an LTL model-checking tool can be employed to decide whether the model satisfies the formula. If $\mathcal{M}$ falsifies φ, then the tool produces a counterexample, which is a path in $\mathcal{M}$ that witnesses the falsification of φ. In practice, such counterexamples are invaluable for investigating the system and finding the errors that cause the falsification. Alternatively, if $\mathcal{M}$ satisfies φ, then the tool typically does not provide any further feedback. This is not unexpected since to show this one needs to consider all of $\mathcal{M}$'s executions and therefore no single execution would be a sufficient witness. However, due to potential formalization errors, $\mathcal{M}$'s satisfaction of φ does not imply that the system meets its requirements. An error in $\mathcal{M}$ indicates that $\mathcal{M}$ does not conform to the system it is intended to model, and an error in φ signifies that φ fails to express the system's requirements. Prudent analysis would therefore require further feedback to provide increased confidence that no formalization errors exist in specifying $\mathcal{M}$ and writing φ. Vacuity detection is a prominent technique for automatically discovering such errors.

Informally, a model $\mathcal{M}$ satisfies a formula φ vacuously when the satisfaction is due to "unintended" reasons. For example, the formula $p \rightarrow q$ is satisfied by any model in which p is false. This satisfaction however may be "unintended" as q's truth assignment plays no role in it. This phenomenon is known as propositional antecedent failure. It has been observed in practice that vacuous satisfactions of this kind are likely due to formalization errors in the model or the formula [1], [2]. This observation has motivated research on generalizing the idea of antecedent failure to temporal logic. Consider, for example, the LTL formula $\psi = \Box(p \rightarrow \Diamond q)$, stating that every occurrence of p must be followed by an occurrence of q. Following the intuition behind antecedent failure, Beer et al. argue that $\mathcal{M}$ satisfies ψ vacuously if p is always false in $\mathcal{M}$, that is, when $\mathcal{M}$ satisfies $\Box(\neg p)$ [2]. The generalization of antecedent failure to temporal connectives is, however, not always as straightforward as this example may suggest. There are in fact many alternative definitions of vacuity [3]–[7] and a universally accepted definition does not yet exist. Nonetheless, the common thread that connects all these definitions is that they generalize, in one way or another, the antecedent failure of an implication to the syntax of a temporal logic. They are therefore all syntactic.

A definition of vacuity is syntactic if the decision whether $\mathcal{M}$ satisfies φ vacuously is affected by semantics-preserving changes to φ's syntax. While an argument can be made that syntactic definitions capture the intuition of the specifier, we contend that such definitions are not always desirable. From a theoretical standpoint, qualifying the semantic notion of satisfaction with a syntactic vacuity condition runs contrary to intuition. From a practical standpoint, two correct specifications of the same requirement as two syntactically different, but semantically equivalent, formulas may yield contradictory results regarding whether $\mathcal{M}$ satisfies the requirement vacuously. This means that a satisfaction that is intuitively vacuous can be declared non-vacuous and vice versa. Finally, nested until connectives in LTL do not readily lend themselves to a syntactic generalization of antecedent failure. We substantiate this claim in §II where we show that existing definitions of vacuity all fall short in capturing the "intended" behaviors described by formulas with nested until connectives. Note in this regard that nested until connectives cannot be avoided due to LTL's until hierarchy [8]. This is also a practical constraint. For instance, the *bounded existence* formula [9], which is a common LTL property pattern, has a nesting depth of five.

1530-1311/15 $31.00 © 2015 IEEE
DOI 10.1109/TIME.2015.14

To address the aforementioned issues, we propose a semantic definition of vacuity where either two semantically equivalent LTL formulas are both satisfied vacuously in a model, or neither of them is. This sets our definition apart from all syntactic notions of vacuity [2]–[7], and the notion of the *inherent vacuity* of a formula [10], which abstracts away from $\mathcal{M}$. Our starting point is a canonical form for LTL formulas [11], derived from Gabbay's separation theorem [12], which states that each LTL formula can be rewritten as the conjunction of finitely many formulas of the form $(P \rightarrow \bigcirc F)$. Here, P is a past-only LTL formula, and F refers solely to the future. This form is canonical in that any two semantically equivalent formulas represented in this form have the same number of conjuncts, and the corresponding past-only parts and the future parts are (semantically) equivalent [11]. Antecedent failure can then be generalized to LTL by accounting for the past-only parts that are never satisfied by a path. This allows us to define a notion of temporal antecedent failure that generalizes antecedent failure to LTL in a syntactic-invariant manner. This straightforward generalization, we argue, can be overly conservative. We show that using a range of abstractions based on the satisfaction of the past-only parts over a path, one can obtain a more refined temporal picture of *how* the path satisfies φ. This naturally instantiates the notion of *same-way satisfaction*, which we describe below.

Same-way satisfaction, although never explicitly discussed in other works, is fundamental to vacuity. Intuitively, vacuity [2] partitions the set of paths that satisfy a formula into a number of equivalence classes. Those paths that belong to the same class satisfy the formula "in the same way". For example, Beer et al.'s definition of vacuity partitions the set of paths that satisfy the aforementioned LTL formula ψ into those with an occurrence of p and those without an occurrence of p, and then further refines these sets into those with finitely many occurrences of q and those with infinitely many occurrences of q (which makes p's satisfaction immaterial). They then say that a model satisfies ψ vacuously if it does not intersect the "intended" classes, i.e., the set of paths that have an occurrence of p and the set of paths with finitely many occurrences of q. Instead of labeling certain paths as intended and others as unintended, the notion of same-way satisfaction allows us to generalize this intuition: we say that a model satisfies ψ non-vacuously if its set of paths intersects with all of these classes, hence representing all the different ways of satisfying the formula. More importantly, the characterization of these equivalence classes is entirely independent of the syntax of ψ in our definition.

Contributions. We motivate and present a semantic definition of vacuity for LTL. Our definition is based upon the equivalence classes induced by a canonical separation of anchored LTL formulas. Namely, each LTL formula φ partitions the set of paths that satisfy it into n equivalence classes, for $n \in \mathbb{N}$. We say a model $\mathcal{M}$ satisfies φ vacuously if the set of $\mathcal{M}$'s paths does not intersect with each of these equivalence classes. We present an algorithm for generating *trap* formulas $\phi_1, \cdots, \phi_n$, such that $\mathcal{M}$ violates ϕ_i if and only if the set of $\mathcal{M}$'s paths intersects with the i^{th} equivalence class. These formulas can be used with a model-checking tool to decide whether $\mathcal{M}$ satisfies φ vacuously, and to generate interesting witnesses. Interesting witnesses are paths of $\mathcal{M}$ that demonstrate all the "different ways" that $\mathcal{M}$ satisfies φ.

Constructing trap properties is computationally expensive. We therefore present a more efficient algorithm that achieves the same goal but uses Büchi automata instead of trap properties. A by-product of this automata-based algorithm is that it naturally extends the proposed notion of semantic vacuity to ω-regular languages, a strict superset of the languages expressible by LTL. We also present the worst-case complexity of the automata-based algorithm, and empirically investigate its performance by generating interesting witnesses for a number of real-world examples. Our results suggest that semantic vacuity can be efficiently decided in practice.

Outline. In §II, we recall linear-time temporal logic, generalized Büchi automata, and various definitions of syntactic vacuity. In §III, we introduce the notions of canonical separation and same-way satisfaction, and motivate and define semantic vacuity. In §IV, we give an algorithm for deciding semantic vacuity based on trap properties. In §V, we give a more efficient algorithm using generalized Büchi automata, present its worst-case complexity, and empirically evaluate its performance on real-world examples. In §VI, we compare our definition of vacuity to alternative ones from the literature, and discuss its limitations.

II. Preliminaries

For a finite set AP of atomic propositions, we fix the alphabet $\Sigma = 2^{AP}$. A *letter* is an element of Σ. Let Σ^ω be the set of all countably infinite sequences over Σ, Σ^+ be the set of all finite nonempty sequences over Σ, and $\Sigma^* = \Sigma^+ \cup \{\epsilon\}$, where ϵ is the empty sequence. A *trace* is an element of Σ^+, a *path* is an element of Σ^ω, and a *property* or a *language* is a set of paths. A *prefix* π_i of a path $\pi = p_0 p_1 p_2 \cdots$ is the trace $p_0 p_1 \cdots p_i$. For a trace t and a path, or trace, π, the *concatenation* of t and π is denoted $t\pi$. We write $\mathbf{a}$ for the letter $\{a\}$, and $\hat{a}$ for the set $\{l \in \Sigma \mid a \in l\}$, with $a \in AP$.

We next define the syntax and semantics of LTL. Our definitions here are standard; see, for example, [12]–[14].

Definition *1 (LTL Syntax):* The syntax of LTL is given by the grammar

$$\varphi ::= \top \mid a \mid \neg\varphi \mid \varphi \vee \varphi \mid \bullet\varphi \mid \bigcirc\varphi \mid \varphi \,\mathcal{S}\, \varphi \mid \varphi \,\mathcal{U}\, \varphi \ ,$$

where $a \in AP$. We write $\bot$ for $\neg\top$, $\varphi \wedge \psi$ for $\neg(\neg\varphi \vee \neg\psi)$, $\varphi \rightarrow \psi$ for $\neg\varphi \vee \psi$, $\blacklozenge\varphi$ for $\top \,\mathcal{S}\, \varphi$, $\blacksquare\varphi$ for $\neg\blacklozenge\neg\varphi$, $\Diamond\varphi$ for $\top \,\mathcal{U}\, \varphi$, and $\Box\varphi$ for $\neg\Diamond\neg\varphi$.

The *size* of an LTL formula is defined inductively over its structure: $|\top| = 1$, $|a| = 1$ for $a \in AP$, $|\nabla\varphi| = 1 + |\varphi|$ for $\nabla \in \{\neg, \bullet, \bigcirc\}$, and $|\varphi \star \psi| = 1 + |\varphi| + |\psi|$ for $\star \in \{\vee, \mathcal{S}, \mathcal{U}\}$. Note that the derived connectives introduced above may contribute more than 1 to a formula's size. However, the contribution is a constant. For example, $|\Box\psi| = 4 + |\psi|$.

Definition *2 (LTL Semantics):* For a path $\pi = p_0 p_1 p_2 \cdots$ and $i \in \mathbb{N}$, the *satisfaction relation* for LTL formulas is defined inductively over the formula structure:

$$
\begin{array}{ll}
\pi, i \models \top & \\
\pi, i \models a & \text{if } a \in p_i \\
\pi, i \models \neg\varphi & \text{if } \pi, i \not\models \varphi \\
\pi, i \models \varphi \vee \psi & \text{if } \pi, i \models \varphi \text{ or } \pi, i \models \psi \\
\pi, i \models \bullet\varphi & \text{if } i > 0 \text{ and } \pi, i-1 \models \varphi \\
\pi, i \models \circ\varphi & \text{if } \pi, i+1 \models \varphi \\
\pi, i \models \varphi \mathcal{S} \psi & \text{if there is a } j \leq i \text{ such that } \pi, j \models \psi \\
 & \text{and } \pi, k \models \varphi, \text{ for all } j < k \leq i \\
\pi, i \models \varphi \mathcal{U} \psi & \text{if there is a } j \geq i \text{ such that } \pi, j \models \psi \\
 & \text{and } \pi, k \models \varphi, \text{ for all } i \leq k < j
\end{array}
$$

We say that π *satisfies* φ at *time* i if $\pi, i \models \varphi$, and φ is (initially) *satisfiable* if there exists a path π such that $\pi, 0 \models \varphi$. An LTL formula φ defines the property $L(\varphi) = \{\pi \in \Sigma^\omega \mid \pi, 0 \models \varphi\}$.

Two LTL formulas φ and ψ are (initially) *equivalent*, denoted $\varphi \equiv \psi$, if $\forall \pi \in \Sigma^\omega.\ (\pi, 0 \models \varphi \iff \pi, 0 \models \psi)$. A *past* formula is a formula without the future temporal connectives $\circ$ and $\mathcal{U}$. A *future* formula is a formula without the past temporal connectives $\bullet$ and $\mathcal{S}$. It is immediate that the satisfaction of a future formula by a path π at time i does not depend on the prefix π_{i-1} and that the satisfaction of a past formula by π at time i only depends on the prefix π_i. This justifies the following definition for the satisfaction of a past formula by a trace. A trace $t \in \Sigma^+$ satisfies a past formula P, denoted $t \models P$, if $t\pi, |t| - 1 \models P$, for any path π. We say P is satisfiable if $\exists t \in \Sigma^+.\ t \models P$. Past formulas P_1 and P_2 are *semantically equivalent* if $\forall t \in \Sigma^+.\ (t \models P_1 \iff t \models P_2)$.

For an LTL formula φ, one can construct a *generalized Büchi automaton* with at most $2^{|\varphi|+1}$ states that recognizes φ [15], i.e., accepts $L(\varphi)$. Below, we define (generalized) Büchi automata.

Definition *3 (Büchi Automaton):* A *generalized Büchi automaton* over the alphabet Σ is a tuple $\mathcal{A} = (Q, Q_0, \Delta, F)$ with a finite set Q of states, a set of initial states $Q_0 \subseteq Q$, a transition relation $\Delta \subseteq Q \times \Sigma \times Q$, and an acceptance condition $F \subseteq 2^Q$. A *run* r of $\mathcal{A}$ on a path $\pi = p_0 p_1 \cdots \in \Sigma^\omega$ is a sequence of states $s_0 s_1 \cdots$ such that $s_0 \in Q_0$ and $(s_i, p_i, s_{i+1}) \in \Delta$, for $i \in \mathbb{N}$. The run r is *accepting* if, for every set $S \in F$, there is a state $s \in S$ that occurs infinitely often in r. The automaton $\mathcal{A}$ *accepts* π if there is an accepting run of $\mathcal{A}$ on π. The property accepted by $\mathcal{A}$ is defined as $L(\mathcal{A}) = \{\pi \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \pi\}$.

A *Büchi automaton*, as opposed to its generalized variant, comes with an acceptance condition $F \subseteq Q$. Therefore, a Büchi automaton can be seen as a generalized Büchi automaton with the acceptance condition $\{F\}$.

For $t \in \Sigma^+$, we write $(s, t, s') \in \Delta^+$ if there is a sequence of states $s_0, s_1, \cdots, s_{|t|}$ such that $s = s_0, s' = s_{|t|}$ and $(s_i, t_i, s_{i+1}) \in \Delta$, for $0 \leq i < |t|$. For a trace t, we define Q_t as the set of all end states of finite runs of $\mathcal{A}$ on t. That is, $Q_t = \{s' \in Q \mid \exists s \in Q_0.\ (s, t, s') \in \Delta^+\}$.

We use propositional formulas defined over $AP \cup \{\top\}$ as syntactic sugar for specifying the transitions of an automaton. The transition (q, Φ, q'), with $q, q' \in Q$ and Φ a propositional formula, is a shorthand for $(q, l, q') \in \Delta$ for all $l \in \Sigma$ that satisfy Φ.

We now turn to *model checking* and *vacuity detection*. A Kripke model $\mathcal{M}$ is a finite structure that induces a set of paths. We do not further specify Kripke models as we identify them with the sets of paths they induce. Given a model $\mathcal{M}$ and an LTL formula φ, the model checking decision problem asks whether all paths of $\mathcal{M}$ are contained in $L(\varphi)$. If the answer is positive, we say that $\mathcal{M}$ *satisfies* φ and write $\mathcal{M} \models \varphi$. Otherwise, we say that $\mathcal{M}$ *falsifies* φ and write $\mathcal{M} \not\models \varphi$. The model checking problem for LTL formulas is PSPACE-complete [16], [17]. It can however be decided efficiently for a large class of practical scenarios; see, e.g., [18], [19].

LTL model checking can be used to decide whether a system, specified as a Kripke model $\mathcal{M}$, satisfies its requirements, written as an LTL formula φ. If $\mathcal{M} \not\models \varphi$, then model-checking tools substantiate the result with a counterexample. The counterexample is typically a finite representation of a path π of $\mathcal{M}$ that is a witness for falsification, i.e., $\pi, 0 \not\models \varphi$. This is invaluable for identifying the source of the falsification: either the system does not satisfy its requirements or, due to a formalization error, the model does not correctly represent the system or the formula does not correctly encode the requirements. In contrast, if $\mathcal{M} \models \varphi$, then model-checking tools provide no further feedback. $\mathcal{M}$'s satisfaction of φ does not, however, imply that the system meets its requirements. Prudent analysis would therefore require further feedback to increase the confidence that no formalization errors exist in specifying $\mathcal{M}$ and writing φ. Vacuity detection is a prominent technique for augmenting positive model-checking results.

Vacuity detection is a generalization of propositional antecedent failure to temporal logic. For instance, if $\varphi = \square(p \rightarrow q)$ and p is always false in $\mathcal{M}$, then q's truth value does not affect the satisfaction of φ in $\mathcal{M}$. This indicates that $\mathcal{M}$ does not exhibit an "intended" or "interesting" way of satisfying φ; namely, that of p eventually becoming true. Intuitively, if $\pi, 0 \models \varphi$ and some subformula of φ does not affect the satisfaction of φ by π, then we say that π vacuously satisfies φ. It has been observed in practice that vacuous satisfaction is likely due to formalization errors [1], [2]. This has motivated research on formally defining the intuitive understanding of vacuity discussed above. Numerous definitions have been proposed in the literature. Here, we focus on three leading formalizations.

1) *Subformula vacuity* [2]: φ is vacuously satisfied in $\mathcal{M}$ if there is a subformula ψ of φ such that $\mathcal{M} \models \varphi[\psi \leftarrow \chi]$, for every formula χ. Here, $[\psi \leftarrow \chi]$ stands for simultaneously substituting all instances of ψ with χ.
2) *Subformula occurrence vacuity* [7]: φ is vacuously satisfied in $\mathcal{M}$ if there is a subformula *occurrence* ψ in φ such that $\mathcal{M} \models \varphi[\psi \leftarrow \chi]$, for every formula χ. Here, the substitution is performed independently for each occurrence of ψ.
3) *Trace vacuity* [3]: φ is vacuously satisfied in $\mathcal{M}$ if there is a subformula ψ of φ such that $\mathcal{M} \models \forall x.\, \varphi[\psi \leftarrow x]$, where $x \notin AP$. Trace vacuity is a strong condition: even

arbitrarily choosing ψ's truth value at each moment in time cannot undermine $\mathcal{M} \models \varphi$.

These definitions coincide when every atomic proposition has only a single occurrence in φ. Moreover, to decide if a subformula with a single occurrence causes syntactic vacuity, it is sufficient to substitute it with its most challenging substitution, which is either $\bot$ or $\top$, depending on the subformula's polarity; see [7] for further details.

Trace vacuity is sometimes called "semantic" because of its universal quantification. The following example however shows that these definitions are all syntactic: the result of vacuity detection depends on φ's syntax, not its semantics.

Example *4:* Consider the LTL formula $\varphi = (c \,\mathcal{U}\, o) \,\mathcal{U}\, x$, where c stands for performing a computation step, o stands for outputting a result, and x stands for the termination signal. The formula φ formalizes the requirement that the system continuously performs computations or outputs a result until eventually a termination signal is sent. Moreover, a computation initiated before the termination signal must not be interrupted before it outputs a result, which can possibly occur after the termination signal. Replacing the subformulas of φ by the most challenging substitution $\bot$ [7] results in the following set of formulas.

$$S = \{(c \,\mathcal{U}\, o) \,\mathcal{U}\, \bot \,, (c \,\mathcal{U}\, \bot) \,\mathcal{U}\, x \,, (\bot \,\mathcal{U}\, o) \,\mathcal{U}\, x \,, \bot \,\mathcal{U}\, x \,, \bot\}.$$

The formula φ is vacuously satisfied (for the three aforementioned definitions) in a model $\mathcal{M}$ if and only if $\mathcal{M} \models \varphi$ and $\mathcal{M}$ satisfies at least one of the formulas in S. Only two of the formulas in S are satisfiable, namely, $(c \,\mathcal{U}\, \bot) \,\mathcal{U}\, x \equiv x$ and $(\bot \,\mathcal{U}\, o) \,\mathcal{U}\, x \equiv o \,\mathcal{U}\, x$. It follows that φ is vacuously satisfied in a model if and only if the model satisfies $o \,\mathcal{U}\, x$. For example, φ is not vacuously satisfied in the model $\mathcal{M} = \{\mathbf{cox}^\omega\}$ which contains only a single path.

Now, consider the formula $\varphi' = (c \vee o) \,\mathcal{U}\, ((o \wedge \circ x) \vee (x \wedge (c \,\mathcal{U}\, o))) \vee x$, which is vacuously satisfied in $\mathcal{M}$ according to the three definitions given above. This is because substituting the subformula $(x \wedge (c \,\mathcal{U}\, o))$ with the most challenging substitution $\bot$ results in the formula $(c \vee o) \,\mathcal{U}\, (o \wedge \circ x) \vee x$, which is satisfied by $\mathcal{M}$. It is however easy to see that $\varphi \equiv \varphi'$. All three definitions are therefore syntactic.

Intuitively, φ's and φ''s satisfactions in $\mathcal{M}$ should both be declared vacuous: a path where the last computation results in an output after the termination signal is sent represents an interesting behavior. This behavior is, however, not present in $\mathcal{M} = \{\mathbf{cox}^\omega\}$. Syntactic definitions of vacuity fail to detect φ's vacuous satisfaction because, informally, the left argument of φ's outer *until* looks farther into the future than its right argument. Only after this behavior is made explicit, as it is done in φ', are the syntactic definitions able to detect the vacuity of satisfaction here. △

This example illustrates that a syntactic definition of vacuity is undesirable because vacuity detection would then depend on syntactic formalization choices that have no effect on the semantics. As the example shows, the more intuitive result is not necessarily obtained from the simpler formula (φ in the example). It is generally unclear which syntactic representation of a requirement is suitable for detecting syntactic vacuity.

III. Generalizing Antecedent Failure

It is not surprising that the existing definitions of vacuity are syntactic, since propositional antecedent failure is also a syntactic notion: it is only defined for formulas of the form $A \rightarrow B$. The example of the formula $\Box(p \rightarrow \Diamond q)$ from the introduction suggests that a simple syntactic generalization of antecedent failure to LTL can be defined for formulas of the form $\Box(A \rightarrow B)$. Such a generalization is however bound to be syntactic, and it is applicable only to a limited set of formulas. To address these issues, we give a syntactic-invariant generalization of antecedent failure to LTL based on the notion of *canonical separation of anchored* LTL formulas [11].

Definition *5:* Let φ be an LTL formula. A *canonical separation of anchored* φ, denoted $\mathrm{CS}(\varphi)_n$, is a formula of the form $(P_1 \rightarrow \circ F_1) \wedge \cdots \wedge (P_n \rightarrow \circ F_n)$, with $n \in \mathbb{N}$, where the following conditions hold: (1) each P_i is a satisfiable past formula, and each F_i is a future formula; (2) for every $t \in \Sigma^+$, there is a unique i such that $t \models P_i$; (3) for all i, j, if $i \neq j$ then $F_i \not\equiv F_j$; (4) $\varphi \equiv \Box \mathrm{CS}(\varphi)_n \equiv \Diamond \mathrm{CS}(\varphi)_n$. We use designated indices $\bot$ and $\top$ to denote respectively the indices i and j such that $F_i \equiv \bot$ and $F_j \equiv \top$, if they exist.

For any formula φ, a canonical separation of anchored φ can be constructed by applying Gabbay's separation algorithm [12], [20] to the formula $\blacksquare\blacklozenge\varphi$, referred to as *anchored* φ. We remark that $\varphi \equiv \blacksquare\blacklozenge\varphi$. Moreover, any two (syntactically distinct) canonical separations of anchored φ have the same number of conjuncts, their corresponding future formulas are equivalent and their corresponding past formulas are semantically equivalent [11]. Thus, a canonical separation of anchored φ is independent from φ's syntax. The following example illustrates these notions.

Example *6:* Consider the formula $\varphi = (c \,\mathcal{U}\, o) \,\mathcal{U}\, x$ discussed in Example 4. A canonical separation of anchored φ is given by the conjunction of the following five formulas:

(1) $\blacklozenge\Big(\neg c \wedge (\neg o \,\mathcal{S}\, (\neg o \wedge \blacksquare\neg x))\Big) \rightarrow \circ\bot$

(2) $\Big(\blacksquare((c \vee o) \wedge \neg x) \wedge o\Big) \rightarrow \circ\varphi$

(3) $\Big(\blacksquare((c \vee o) \wedge \neg x) \wedge \neg o\Big) \rightarrow \circ((c \,\mathcal{U}\, o) \wedge \varphi)$

(4) $\Big(\blacksquare(c \vee o) \wedge (\neg o \,\mathcal{S}\, (\neg o \wedge \blacksquare\neg x)) \wedge \blacklozenge x\Big) \rightarrow \circ(c \,\mathcal{U}\, o)$

(5) $\Big(\blacksquare\blacklozenge x \vee \blacklozenge\Big(\bullet\blacksquare(c \vee o) \wedge ((x \wedge \bullet o) \vee (o \wedge \blacklozenge x))\Big)\Big) \rightarrow \circ\top$

We characterize the past-only parts of these formulas by referring to the Büchi automaton $\mathcal{A}_\varphi$ that recognizes φ, depicted in Figure 1. The state with double circles denotes the automaton's accepting state. Recall that, for a trace t, we write Q_t for the set $\{s' \in Q \mid \exists s \in Q_0.\ (s, t, s') \in \Delta^+\}$. A trace t satisfies the formula P_1 (respectively, P_2, P_3, P_4, P_5) if and only if Q_t is the set $\{d\}$ (respectively, $\{q_0\}$, $\{q_1\}$, $\{q_2\}$, $\{q_3\}$). Moreover, the designated index $\bot$ refers to conjunct 1, and index $\top$ to conjunct 5. △

Note that $\Box\mathrm{CS}(\varphi)_n$, which is equivalent to φ, resembles $\Box(A \rightarrow B)$. However, it is determined by φ's semantics, rather than its syntax. This allows us to define temporal antecedent failure in a syntactic-invariant manner.

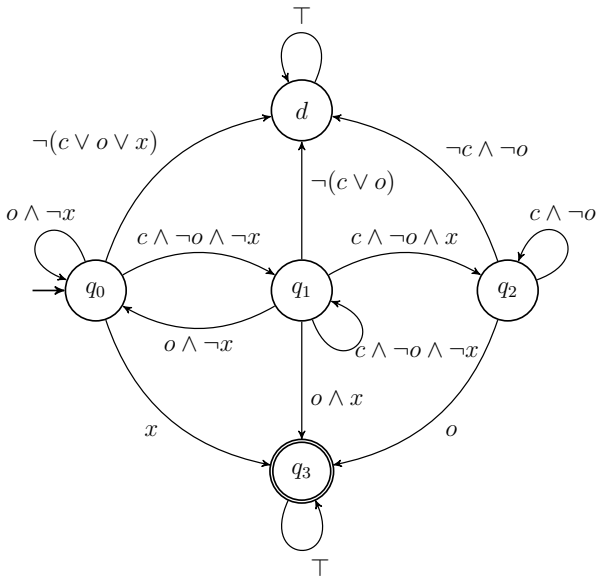

Fig. 1. Büchi automaton recognizing the formula $(c\,\mathcal{U}\,o)\,\mathcal{U}\,x$

A. Temporal Antecedent Failure

In the following, we give a syntactic-invariant definition for temporal antecedent failure (see our related work for a comparison to [21]). Afterward, we argue that although this generalizes antecedent failure to LTL in a straightforward manner, it leads to an overly conservative vacuity definition.

Definition *7 (Temporal Antecedent Failure):* Let φ be an LTL formula with a canonical separation $\mathrm{CS}(\varphi)_n$. A model $\mathcal{M}$, with $\mathcal{M} \models \varphi$, satisfies φ due to temporal antecedent failure if there is an index $i \neq \bot$ such that $\forall \pi \in \mathcal{M}.\ \pi, 0 \not\models \Diamond P_i$.

This naturally leads to a definition for vacuity: $\mathcal{M}$ satisfies φ vacuously if $\mathcal{M}$ satisfies φ due to temporal antecedent failure. Note that this definition is not syntactic. It can however be overly conservative. Namely, a model that lacks intuitively interesting or intended paths can satisfy a formula non-vacuously. This implies that non-vacuity obtained based on Definition 7 only weakly substantiates positive model-checking results. The following example illustrates this point, and compares temporal antecedent failure to syntactic vacuity.

Example *8:* Consider the formula φ of Example 6. For any path π, its shortest prefix that satisfies P_4 (i.e., reaches state q_2 in Figure 1) is necessarily a trace from the set defined by the regular expression $(\Sigma \backslash \hat{x})^* \mathbf{c}\{c, x\}$; this is not however a sufficient condition. Intuitively, this regular expression denotes the set of traces where x is falsified until only c is satisfied, immediately followed by both c and x being satisfied. Since any path with such a prefix must falsify $o\,\mathcal{U}\,x$, it is immediate that for the formula φ syntactic vacuity (according to any of the three definitions given in §II) implies temporal antecedent failure.

Now, let $\mathcal{M}$ be the model with a single path $\pi = \mathbf{oc}\{c, x\}\mathbf{o}^\omega$. It is easy to check that, according to Definition 7, $\mathcal{M}$ does not satisfy φ merely due to temporal antecedent failure. It then follows that φ is not vacuously satisfied according to syntactic vacuity. These results are, however, counter-intuitive: the model $\mathcal{M}$ lacks a path such that the termination signal immediately follows, or coincides with, the final output, i.e., a path that does not visit state q_2 in Figure 1. Intuitively, this represents a distinct and interesting behavior of the system according to φ that is absent from the model. △

B. Semantic Vacuity

We define semantic vacuity for LTL by refining temporal antecedent failure. The refinement is motivated by the following observation. Let us consider propositional antecedent failure for the formula $(A_1 \rightarrow B_1) \wedge \cdots \wedge (A_n \rightarrow B_n)$, where the formulas A_i are pairwise inconsistent. Then, for any truth assignment, at most one of the formulas A_i is satisfied. Therefore, generalizing antecedent failure to this formula can be considered as n independent instances of antecedent failure. Now, consider $\Box\mathrm{CS}(\varphi)_n$, for an LTL formula φ. Although the past formulas P_i are pairwise inconsistent in $\Box\mathrm{CS}(\varphi)_n$, a path π can satisfy several past formulas at different points in time. Furthermore, a path π can satisfy a single formula P_i any number of times, including infinitely often. That is, given a path, the dichotomy between satisfying a past formula and falsifying it is too coarse. This motivates the definition of same-way satisfaction, given below, which refines temporal antecedent failure. We will then use same-way satisfaction to define semantic vacuity. Below, we introduce the concepts needed to formalize same-way satisfaction.

Let φ be an LTL formula and $\mathrm{CS}(\varphi)_n$ be a canonical separation of anchored φ. We define the minimal equivalence relation $\sim_\varphi$ on finite traces as: for $t, t' \in \Sigma^+$, $t \sim_\varphi t'$ if $\exists i \in \{1, \cdots, n\}.\ t, t' \models P_i$. It is immediate to see that this relation induces n equivalence classes, henceforth referred to as $O_1, \cdots, O_n$. Each class intuitively represents an *obligation*: for any trace $t \in O_i$, the formula F_i defines the sufficient and necessary condition (obligation) that a path π must fulfill, namely that $\pi, 0 \models F_i$, so that the path $t\pi$ satisfies φ. We therefore refer to the (quotient) set $\{O_1, \cdots, O_n\}$ as the *alphabet of obligations* and denote it by Σ_φ. Note that a model $\mathcal{M}$ falsifies $\Diamond P_i$ if and only if for every prefix π_j of any path $\pi \in \mathcal{M}$, $\pi_j \notin O_i$. The definition of temporal antecedent failure therefore implicitly abstracts each path $\pi \in \mathcal{M}$ with the set of obligations corresponding to the prefixes of π. In the following, we refine this notion by accounting for whether infinitely many, finitely many, or none of the prefixes of a path belong to an obligation. Informally, these cases represent the path's recurring behaviors, its transient behaviors, and the behaviors that are absent in the path, respectively.

Definition *9 (Obligation Abstraction):* Let φ be an LTL formula, and $\pi \in \Sigma^\omega$. We define the *obligation abstraction* of π with respect to φ as the tuple

$$s_\varphi(\pi) = (s_1^\pi, \cdots, s_{|\Sigma_\varphi|}^\pi) \in \{0, 1, \infty\}^{|\Sigma_\varphi|}\ ,$$

where $s_i^\pi = 0$ if no prefix of π belongs to O_i, $s_i^\pi = 1$ if finitely many prefixes of π belong to O_i, and $s_i^\pi = \infty$ otherwise (as a convention, we exclude zero from finitely many). We may omit the subscript from $s_\varphi(\pi)$ when φ is clear from the context.

TABLE I
CANDIDATE OBLIGATION ABSTRACTIONS FOR THE PATHS THAT SATISFY $(c\,\mathcal{U}\,o)\,\mathcal{U}\,x$

Abstraction	*Concretization*	*Abstraction*	*Concretization*
$(0,0,0,0,\infty)$	$\mathbf{x}^\omega$	$(0,0,0,1,\infty)$	Not possible
$(0,0,1,0,\infty)$	$\mathbf{c}\{o,x\}\mathbf{x}^\omega$	$(0,0,1,1,\infty)$	$\mathbf{c}\{c,o\}\mathbf{ox}^\omega$
$(0,1,0,0,\infty)$	$\mathbf{ox}^\omega$	$(0,1,0,1,\infty)$	Not possible
$(0,1,1,0,\infty)$	$\mathbf{ocox}^\omega$	$(0,1,1,1,\infty)$	$\mathbf{oc}\{c,x\}\mathbf{o}^\omega$

Moreover, we say a path π *concretizes* a tuple $s \in \{0,1,\infty\}^{|\Sigma_\varphi|}$ if $s(\pi) = s$.

The following example illustrates this definition.

Example *10:* Consider the formula $\varphi = (c\,\mathcal{U}\,o)\,\mathcal{U}\,x$, for which $\mathrm{CS}(\varphi)_n$ is given in Example 6. It is immediate that $\sim_\varphi$ induces five equivalence classes, and no prefix of a path satisfying φ can belong to the obligation $O_\bot$. Moreover, every path satisfying φ has infinitely many prefixes belonging to the obligation $O_\top$, and finitely many (if any) prefixes belonging to the other obligations; see Figure 1. This gives us eight candidates for the obligation abstractions of the paths in $L(\varphi)$, presented in Table I. Then, we observe that two abstraction candidates cannot be concretized by a path in $L(\varphi)$. This immediately follows from the correspondence between the obligations and the sets of states of the automaton of Figure 1, discussed in Example 6. △

For any $\pi, \sigma \in \Sigma^\omega$ and an LTL formula φ, we write $\pi \simeq_\varphi \sigma$ if $s(\pi) = s(\sigma)$. We say π and σ satisfy φ *in the same way* if $\pi, \sigma \in L(\varphi)$ and $\pi \simeq_\varphi \sigma$. Below, we define semantic vacuity. Intuitively, a formula is semantically vacuously satisfied in $\mathcal{M}$ if $\mathcal{M}$ does not contain representatives of all the different ways of satisfying φ.

Definition *11 (Semantic Vacuity):* A model $\mathcal{M}$ *semantically vacuously* satisfies a formula φ if $\mathcal{M} \models \varphi$ and there exists a tuple $s \in \{0,1,\infty\}^{|\Sigma_\varphi|}$ that has a concretization in $L(\varphi)$, but it does not have one in $\mathcal{M}$.

Two remarks are due here. First, the obligation abstraction distinguishes between recurring, transient, and absent behaviors of a path. Our definition inherits the distinction between absent and non-absent behaviors from temporal antecedent failure, which it refines. It is also clear that the transient behaviors of a path do not affect the membership of the path to the language of an LTL formula, as it is reflected in the acceptance condition of Büchi automata. This is evident, for instance, in the Büchi automaton discussed in Example 6, where each obligation maps to one state of the automaton. We have opted not to further refine the notion of temporal antecedent failure. This choice is motivated by the practical consideration that same-way satisfaction should induce a (preferably small) finite number of equivalence classes. Augmenting positive model-checking results with excessively many witnesses is impractical. In short, Definition 9 strikes a balance between the intuition behind "satisfaction in the same way" and the number of equivalence classes of the same-way satisfaction relation.

Second, same-way satisfaction augments a positive model checking result. Namely, the intersections between the set of paths of $\mathcal{M}$ and each equivalence class defined by $\simeq_\varphi$ demonstrate a distinct way in which $\mathcal{M}$ satisfies φ. Similarly to syntactic vacuity, the fact that $\mathcal{M}$ lacks a certain way of satisfying φ need not correspond to a formalization error. It nevertheless helps the modeler to better understand the relation between the system that $\mathcal{M}$ models and the requirements that φ expresses. Note that, in contrast to the syntactic definitions of vacuity and temporal antecedent failure, a single path cannot satisfy φ in two different ways.

IV. Deciding Semantic Vacuity: Trap Properties

For any LTL formula φ, the relation $\simeq_\varphi$ partitions the set $L(\varphi)$ into a finite number of equivalence classes. We write IP_φ for $L(\varphi)/\simeq_\varphi$. It is immediate that a model $\mathcal{M}$ satisfies φ non-vacuously (according to Definition 11) iff $\forall \Pi \in \mathrm{IP}_\varphi.\ \mathcal{M} \cap \Pi \neq \emptyset$. Then, a set of *interesting witnesses* for the satisfaction of φ in $\mathcal{M}$ is any set W such that $\forall \Pi \in \mathrm{IP}_\varphi . \exists \pi \in W.\ \pi \in \mathcal{M} \cap \Pi$. The following example illustrates these definitions.

Example *12:* Consider the formula $\psi = \square(p \rightarrow \diamond q)$. A canonical separation of anchored ψ is the conjunction of the formula $(\neg q\,\mathcal{S}\,(p \wedge \neg q)) \rightarrow \bigcirc(\diamond q \wedge \psi)$ and the formula $(\blacksquare\neg p \vee (\neg p\,\mathcal{S}\,q)) \rightarrow \bigcirc\psi$.

It follows that the obligation alphabet Σ_ψ has two elements. For any path $\pi \in L(\psi)$, the pigeonhole principle implies that infinitely many prefixes of π must belong to at least one of these obligations. The obligation abstractions of the paths of $L(\psi)$ can then be calculated as the following: $(0,\infty)$, $(1,\infty)$, and (∞,∞). These are the abstractions of the paths expressed by the ω-regular expressions $(\Sigma \setminus \mathbf{p})^\omega$, $(\Sigma^*\mathbf{p})^+\Sigma^*\hat{q}(\Sigma \setminus \mathbf{p})^\omega$, and $(\Sigma^*\mathbf{p}\Sigma^*\hat{q})^\omega$, respectively. An example of a model that satisfies ψ non-vacuously is $\mathcal{M} = \{\mathbf{q}^\omega, \mathbf{pq}^\omega, (\mathbf{pq})^\omega\}$. Note that $\mathcal{M}$ is itself a minimal set of interesting witnesses. Therefore, the model $\mathcal{M}' = \{\mathbf{pq}^\omega\}$ satisfies ψ vacuously. Note that $\mathcal{M}' \models \psi$ is *not* due to temporal antecedent failure. Moreover, the three definitions of syntactic vacuity given in §II state that $\mathcal{M}$ satisfies ψ vacuously, because q is satisfied infinitely often in all paths of $\mathcal{M}$ (thus making p's truth assignment irrelevant for the satisfaction). This demonstrates the difference between the syntactic vacuity definitions and our definition of semantic vacuity. Syntactic definitions are concerned with how each proposition independently affects the satisfaction, whereas semantic vacuity is concerned with how the temporal relation between the propositions constitutes a distinct way of satisfaction. For instance, the single path $\{p,q\}\emptyset^\omega$ satisfies ψ non-vacuously according to the syntactic definitions, but same-way satisfaction does not differentiate it from $\mathbf{q}^\omega$: they have the same obligation abstraction. Clearly, the path $\{p,q\}\emptyset^\omega$ satisfies ψ semantically vacuously. △

Next, we construct trap properties for finding concretizations of any obligation abstraction in a model, if a concretization exists in the model. The trap properties can be used for deciding vacuity and generating interesting witnesses, as discussed below. Fix an LTL formula φ. For a tuple $s =$

$(s_1, \cdots, s_{|\Sigma_\varphi|}) \in \{0, 1, \infty\}^{|\Sigma_\varphi|}$, we define the *trap property* φ_s as

$$\varphi_s = \bigvee_{i \in I_0} (\Diamond P_i) \vee \bigvee_{i \in I_1} (\Box \neg P_i \vee \Box \Diamond P_i) \vee \bigvee_{i \in I_\infty} (\Diamond \Box \neg P_i) \ ,$$

where I_0, I_1, and I_∞ stand for the sets of all indices corresponding to 0, 1, and ∞ in s, respectively. As usual, the past formulas P_i refer to a canonical separation of anchored φ.

Proposition *13:* Let φ be an LTL formula, and $s = (s_1, \cdots, s_{|\Sigma_\varphi|}) \in \{0, 1, \infty\}^{|\Sigma_\varphi|}$. A path π does not belong to $L(\varphi_s)$ if and only if the following conditions hold.

1) For every i such that $s_i = 0$, no prefix of π belongs to O_i.
2) For every i such that $s_i = 1$, finitely many prefixes of π belong to O_i.
3) for every i such that $s_i = \infty$, infinitely many prefixes of π belong to O_i.

Proof: A path π falsifies φ_s iff π falsifies every disjunct of φ_s. That is $\pi \not\models \varphi_s$ iff (1) $\forall i \in I_0.\ \pi, 0 \models \Box \neg P_i$, (2) $\forall i \in I_1.\ \pi, 0 \models \Diamond P_i \wedge \Diamond \Box \neg P_i$, and (3) $\forall i \in I_\infty.\ \pi, 0 \models \Box \Diamond P_i$. These three cases readily correspond to the three conditions of the proposition. ▲

The following corollary of this proposition allows us to use a model-checking tool for deciding semantic vacuity.

Corollary *14:* For any LTL formula φ and model $\mathcal{M}$, with $\mathcal{M} \models \varphi$, the satisfaction is semantically vacuous iff $\exists s \in \{0, 1, \infty\}^{|\Sigma_\varphi|}.\ \varphi_s \wedge \varphi \not\equiv \bot \wedge \mathcal{M} \models \varphi_s$.

Suppose that $\mathcal{M} \models \varphi$. The following algorithm decides if the satisfaction is vacuous. For every $s \in \{0, 1, \infty\}^{|\Sigma_\varphi|}$, do the following steps. Construct the formula φ_s. If $\varphi \wedge \varphi_s$ is satisfiable, then use a model-checking tool to decide if $\mathcal{M} \models \varphi_s$. If the answer is positive, then φ is vacuously satisfied in $\mathcal{M}$. If the algorithm terminates without detecting vacuity, then the satisfaction is not vacuous. Trap properties can also be used to generate interesting witnesses for the relation $\mathcal{M} \models \varphi$. Namely, a set W of interesting witnesses can be found by collecting the paths in $\mathcal{M}$ that falsify φ_s, for all abstractions s such that $\varphi_s \wedge \varphi \not\equiv \bot$.

Constructing trap properties depends on a canonical separation of anchored LTL formulas. Therefore, the aforementioned algorithms for deciding semantic vacuity and generating interesting witnesses also depend on anchored separation. Separating arbitrary LTL formulas is believed to be nonelementary [13]. The past-only formulas, used in trap properties, can be constructed using an elementary algorithm [11]. This algorithm is also prohibitively expensive in practice. In the following section, we propose a more efficient algorithm for deciding semantic vacuity that uses Büchi automata instead of trap properties.

V. Deciding Semantic Vacuity: ω-Automata

In this section, we show how Büchi automata can be used instead of trap properties to decide vacuity and generate interesting witnesses. The use of Büchi automata also allows us to generalize our definition of semantic vacuity to ω-regular languages. The generalization is justified by noting that one can define a canonical form for ω-regular languages that resembles the canonical LTL separation; moreover, it is independent to the shape of the Büchi automaton that recognizes the language. Namely, any ω-regular language can be expressed by the intersection of finitely many expressions of the form $P_i F_i$, where, P_i is a regular expression and F_i is an ω-regular expression. Moreover, $L(P_i \cap P_j) = \emptyset$ and $L(F_i) \neq L(F_j)$, whenever $i \neq j$. This representation of ω-regular languages is therefore analogous to a canonical separation of an anchored LTL formula. These expressions are simple to construct from the automaton D^{F}_φ, which we will introduce in Construction 16.

We start by extending the relation $\sim_\varphi$ to ω-regular languages. Let φ be a (syntactic representation of an) ω-regular language, and $\mathcal{A}_\varphi$ be a Büchi automaton that recognizes φ. For an automaton $\mathcal{A}$ and a trace t, we write $\mathcal{A}^t$ for the automaton obtained from $\mathcal{A}$ by letting Q_t be the set of initial states. For two traces t, t', we define $t \sim_\varphi t'$ if $L(\mathcal{A}^t_\varphi) = L(\mathcal{A}^{t'}_\varphi)$. It is easy to show that this definition depends only on $\mathcal{A}_\varphi$'s semantics, i.e., φ. The following lemma, proved in [11], implies that this is a conservative extension of the definition of $\sim_\varphi$ given in §III-B.

Lemma *15:* Let φ be an LTL formula, $\mathrm{CS}(\varphi)_n$ be a canonical separation of anchored φ, and $\mathcal{A}_\varphi$ be a generalized Büchi automaton that recognizes φ. For $t \in \Sigma^+$, if $t \models P_i$, then $L(\mathcal{A}^t_\varphi) = L(F_i)$. Moreover, for any trace t', if $Q_t = Q_{t'}$ then $t' \models P_i$. ▲

We extend the definitions of satisfaction, obligations, temporal antecedent failure, same-way satisfaction, and semantic vacuity to ω-regular languages in the obvious way. We also extend $\sim_\varphi$ to Σ^* by defining $\epsilon \sim_\varphi \epsilon$. It is easy to check that the relation $\sim_\varphi$ is *right-invariant*, i.e., for all traces $u, v, w \in \Sigma^*$, the equivalence $u \sim_\varphi v$ implies $uw \sim_\varphi vw$. Moreover, it is of finite index. It then follows from the Myhill-Nerode theorem [22] that one can construct a deterministic finite state automaton on finite words (DFA) recognizing the equivalence classes of $\sim_\varphi$. Namely, we construct a DFA $D^{\mathrm{F}}_\varphi = (Q, q_0, \delta, \emptyset)$ such that $|Q| = |\Sigma_\varphi| + 1$ and, for any $i \in \{1, \cdots, |\Sigma_\varphi|\}$, there is a unique state $q_i \in Q$ such that the run of D^{F}_φ on any trace $t \in O_i$ ends at q_i. We sketch the construction below.

Construction *16:* The input is a generalized Büchi automaton $\mathcal{A}_\varphi = (R, R_0, \Delta, F)$ recognizing φ. The output of the construction is the automaton $D^{\mathrm{F}}_\varphi = (Q, q_0, \delta, \emptyset)$, where $Q \subseteq 2^R \cup \{q_0\}$. Let us write R_t for the set $\{s' \in R \mid \exists s \in R_0.\ (s, t, s') \in \Delta^+\}$, which is a subset of R. Let Q be the set of states of D^{F}_φ, and $U \subseteq Q$ be the set of states that "must be processed"; we clarify this shortly. Initially, we define $Q = U = \{q_0\}$. To make the construction more efficient, for every state $q \in Q$, we maintain a list $r(q)$ of corresponding subsets of R, and a single trace $t(q)$ (such that the run of D^{F}_φ on $t(q)$ ends with the state q). We define $t(q_0) = \epsilon$. Now, we process each state in U according to the following steps until the set U is empty:

1) Take any $q \in U$, and let $U \leftarrow U \setminus \{q\}$. Let $t \leftarrow t(q)$.
2) For every $l \in \Sigma$, perform the first applicable action:
   - if $R_{tl} \in r(q')$ for some $q' \in Q \setminus \{q_0\}$, define $\delta(q, l) = q'$;
   - if $L(\mathcal{A}^{tl}_\varphi) = L(\mathcal{A}^{t(q')}_\varphi)$, for some $q' \in Q \setminus \{q_0\}$, define $\delta(q, l) = q'$ and $r(q') \leftarrow r(q') \cup \{R_{tl}\}$;

- for i such that $tl \in O_i$, $Q \leftarrow Q \cup \{q_i\}$, $U \leftarrow U \cup \{q_i\}$, $r(q_i) \leftarrow \{R_{tl}\}$, and $t(q_i) \leftarrow tl$.

Note that the operator $\leftarrow$ denotes assignment. ▲

For an ω-regular property φ, we next show how to construct, given a tuple $s \in \{0,1,\infty\}^{|\Sigma_\varphi|}$, a generalized Büchi automaton $\mathcal{A}_s$ that recognizes the set of concretizations of s. These automata recognize the paths that falsify the trap properties defined in §IV.

A. Constructing Trap Automata

Fix an ω-regular language φ, and let I_0, I_1, and I_∞ be the sets of indices that correspond to 0, 1, and ∞ of some $s \in \{0,1,\infty\}^{|\Sigma_\varphi|}$, respectively. We construct an automaton $\mathcal{A}_s$ such that $\pi \in L(\mathcal{A}_s)$ iff $s(\pi) = s$. The construction consists of two phases. In the first phase, we construct an automaton whose states encode which obligations in I_1 have occurred so far. In the second phase, accepting states are added to the automaton. These can only be reached once all obligations in I_1 have occurred at least once. The automaton's acceptance condition ensures that a path is accepted if and only if each obligation in I_∞ occurs infinitely often and each obligation in I_1 occurs only finitely often. The obligations in I_0 cannot occur by construction. Below, we describe the construction.

Our starting point is the DFA D^{F}_φ. We define the set $Q \times C$, where $C = 2^{|I_1|}$. Intuitively, each element of $Q \times C$ encodes an obligation and the set of transient obligations that have occurred so far. We write $\mathbf{0}$ and $\mathbf{1}$ respectively for $(0,\cdots,0)$ and $(1,\cdots,1)$. We define $e_0 = (q_0, \mathbf{0})$ as $\mathcal{A}_s$'s initial state, and write S and Δ respectively for the set of states and the transition relation of $\mathcal{A}_s$. Initially, $S = \{e_0\}$ and $\Delta = \emptyset$. We expand S inductively: For each $e = (q, c) \in S$, we take the following steps. For each $l \in \Sigma$ and $q_i = \delta(q, l)$, if $i \in I_1 \cup I_\infty$, then we add $e' = (q_i, c')$ to S and (e, l, e') to Δ, where c' is defined as follows; if $i \in I_1$, then c' is obtained by setting the value that corresponds to O_i in c to 1. If $i \in I_\infty$, then $c' = c$. This procedure terminates because $Q \times C$ is finite. A trace t can reach a state of the form $(q, \mathbf{1})$ in $\mathcal{A}_s$ only if each obligation in I_1 has a representative in a prefix of t. We can thus prune the sets S and Δ by removing all the states from which no state of the form $(q, \mathbf{1})$ is reachable. This concludes the first phase.

We now describe how to add accepting states to $\mathcal{A}_s$. For each $i \in I_\infty$, we add a state $(q'_i, \mathbf{1})$ to S. Then, for every $(q, c) \in S$ and letter $l \in \Sigma$ such that $((q, c), l, (q_i, \mathbf{1})) \in \Delta$, we add $((q, c), l, (q'_i, \mathbf{1}))$ to Δ. Furthermore, for any $i, j \in I_\infty$ and $l \in \Sigma$, if $((q_i, \mathbf{1}), l, (q_j, \mathbf{1})) \in \Delta$, then we add $((q'_i, \mathbf{1}), l, (q'_j, \mathbf{1}))$ to Δ. The acceptance condition of the generalized Büchi automaton $\mathcal{A}_s$ is then given by the set $F = \{\{(q'_i, \mathbf{1})\} \mid i \in I_\infty\}$. The following theorem shows the correctness of our construction.

Theorem *17:* Let φ be an ω-regular language, s be a tuple in $\{0,1,\infty\}^{|\Sigma_\varphi|}$, and $\pi \in \Sigma^\omega$. Then, $\pi \in L(\mathcal{A}_s)$ if and only if $s(\pi) = s$.

Proof: Assume that $\pi \in L(\mathcal{A}_s)$ and let $r_0 r_1 \cdots$ be an accepting run of $\mathcal{A}_s$ on π. Then, there is is a smallest index k such that r_k and all subsequent states are of the form $(q', \mathbf{1})$. It follows immediately that r_{k-1} is of the form $(q, \mathbf{1})$ and, therefore, each obligation in I_1 is represented by a prefix of π (and of π_{k-1}). Since states of the form $(q', \mathbf{1})$ correspond to obligations in I_∞, only those obligations can occur infinitely often in prefixes of π. From the acceptance condition, it follows that all obligations in I_∞ occur infinitely often in prefixes of π. That no obligation in I_0 has representatives in prefixes of π is immediate from the construction of $\mathcal{A}_s$.

Conversely, let $s(\pi) = s$. Since only finitely many prefixes of π belong to obligations in I_1, there is a largest index k such that π_k belongs to such an obligation. Let $r_0 = e_0$ and, for $0 < m \leq k$, let r_m be the state of the form (q, c) such that $(r_{m-1}, p_{m-1}, r_m) \in \Delta$. It follows from the construction of $\mathcal{A}_s$ that r_k is of the form $(q, \mathbf{1})$. For $m > k$, let r_m be the state of the form $(q', \mathbf{1})$ such that $(r_{m-1}, p_{m-1}, r_m) \in \Delta$. We have constructed a run of $\mathcal{A}_s$ on π, and since there are infinitely many prefixes of π that belong to each of the obligations in I_∞, we conclude that each state of the form $(q', \mathbf{1})$ occurs infinitely often in the run. ▲

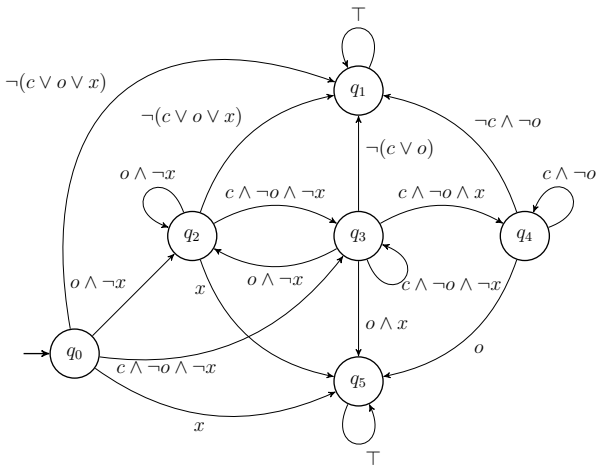

Fig. 2. The DFA D^{F}_φ

The following example illustrates the construction of trap automata.

Example *18:* We construct the automaton $\mathcal{A}_s$ for the tuple $s = (0,1,1,0,\infty)$ of Example 10. We start with $D^{\mathrm{F}}_\varphi = (Q, q_0, \delta, \emptyset)$, shown in Figure 2. The first phase of the construction results in the automaton of Figure 3. The second phase, not depicted there, adds the accepting state $(q'_5, \mathbf{1})$ and the corresponding transitions to this automaton. △

Corollary *19:* Let φ be an ω-regular language, $\mathcal{M}$ be a model that satisfies φ, and $\mathcal{A}_\varphi$ be a Büchi automaton that recognizes φ. The satisfaction of φ in $\mathcal{M}$ is semantically vacuous iff there is a tuple $s \in \{0,1,\infty\}^{|\Sigma_\varphi|}$ such that $L(\mathcal{A}_\varphi) \cap L(\mathcal{A}_s) \neq \emptyset$ and $\mathcal{M} \cap L(\mathcal{A}_s) = \emptyset$. ▲

Note that, similarly to semantic vacuity, temporal antecedent failure can be decided for ω-regular languages without resorting to separation algorithms. It is easy to see that $\mathcal{M}$ satisfies φ due to temporal antecedent failure iff there is a state $q \in Q \setminus \{q_0, q_\perp\}$ in D^{F}_φ such that $\mathcal{M} \subseteq L(\mathcal{T}_q)$, where $\mathcal{T}_q$ is the Büchi automaton $(Q \setminus \{q\}, \{q_0\}, \delta, Q \setminus \{q, q_\perp\})$.

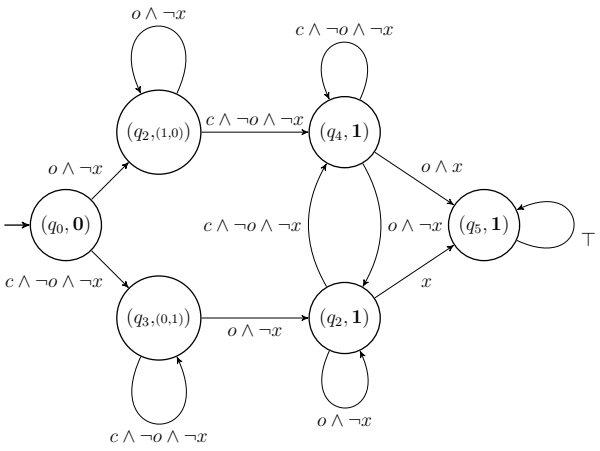

Fig. 3. The automaton $\mathcal{A}_s$ prior to the 2^{nd} phase

B. Complexity and Empirical Evaluation

We give the (worst-case) size of $\mathcal{A}_s$, for a generalized Büchi automaton $\mathcal{A}_\varphi$ and $s \in \{0,1,\infty\}^{|\Sigma_\varphi|}$. By the definition of $\sim_\varphi$, the number of obligations $|\Sigma_\varphi|$ is at most exponential in the number of states of $\mathcal{A}_\varphi$. Recall that, in the DFA D^{F}_φ, the set Q of states has $|\Sigma_\varphi|+1$ elements. The automaton $\mathcal{A}_s$ has at most $2^{|I_1|-1} \cdot (|I_1|+2\cdot|I_\infty|)+1$ states, which is exponential in the size of Σ_φ. Now, if φ is an LTL formula, then $|\Sigma_\varphi|$ is at most double exponential in $|\varphi|$, and the size of $\mathcal{A}_s$ is in the worst case triple exponential in $|\varphi|$. These results give us upper-bounds for $|\mathcal{A}_s|$ that are not necessarily tight.

In the following, we calculate the number of obligations for a number of examples, and show that the worst-case scenarios rarely occur in practice. For our evaluation, we consider the LTL formulas used in the formalization of the GIOP protocol [23] and the TLA v5 formula from the formalization of Bluespec [24]. These models and formulas are publicly available in the Promela database [25]. We also analyze common LTL specification patterns [9], confining our attention to most challenging cases for each pattern. The results are summarized in Table II, where $|\mathcal{A}_\varphi|$ refers to the number of states of $\mathcal{A}_\varphi$ generated from φ using the LTL2BA tool [15]. Except for the simplest formulas, $|\Sigma_\varphi| \leq |\mathcal{A}_\varphi|$. Moreover, except for the formula TLA v5, $|\Sigma_\varphi| \leq |\varphi|$. These results suggest that for most practically relevant LTL formulas, the size of Σ_φ is small; therefore, semantic vacuity can be decided efficiently. We have used the model-checking tool SPIN [19] to decide vacuity for the GIOP model, and have discovered a number of interesting behaviors that the model lacks.

The results of our empirical study suggest that, despite its high worst-case complexity, semantic vacuity can in most cases be efficiently decided in practice. Furthermore, even when deciding semantic vacuity is infeasible, one can still efficiently check for temporal antecedent failure.

TABLE II
EMPIRICAL RESULTS

Formula	$\lvert\varphi\rvert$	$\lvert\Sigma_\varphi\rvert$	$\lvert\mathcal{A}_\varphi\rvert$		**Formula**	$\lvert\varphi\rvert$	$\lvert\Sigma_\varphi\rvert$	$\lvert\mathcal{A}_\varphi\rvert$
GIOP v3	10	3	2		Precedence	26	4	4
GIOP v4l	69	6	17		Response	35	4	8
GIOP v4a	8	2	1		Prec. Chain 1	38	5	8
GIOP v5	20	4	4		Prec. Chain 2	41	4	8
GIOP v6b	19	2	2		Resp. Chain 1	40	6	35
GIOP v8	34	6	4		Resp. Chain 2	38	6	16
GIOP v9a	18	4	11		Constr. Chain	50	6	16
GIOP v9b	12	3	2		Existence	17	3	5
GIOP v10	23	3	4		Bound. Ex.	60	8	16
TLA v5	105	513	2816		Universality	23	4	4
Absence	24	4	4					

VI. Related Work and Discussion

Numerous syntactic definitions of vacuity have been proposed in the literature and their applications have been discussed; see, e.g., [2]–[7], [10], [21], [26]–[28]. For a survey on vacuity detection and other sanity checks in formal verification see [29]. As mentioned before, we are the first to define semantic vacuity, in contrast to the existing definitions which are syntactic. A semantic generalization of antecedent failure for the temporal logic PSL has been proposed by Ben-David et al. in [21]. Their definition is applicable only to a limited set of formulas, namely, those that can be written in the form $\Box(P \rightarrow F)$, where P is a past formula and F is a future formula. This resembles our definition of temporal antecedent failure, but our definition is applicable to any ω-regular language. A goal of Ben-David et al. is to reduce the number of false positive vacuity declarations. Using a conservative definition (such as theirs, or our temporal antecedent failure) in this context is reasonable.

In the context of model-based testing, unique first cause (UFC) coverage by Whalen et al. [30] and Büchi coverage by Tan [31] can be seen as definitions for vacuity. These definitions are, however, not semantic: UFC depends on the syntax of the LTL formula at hand, and Tan's definition relies on the structure of a Büchi automaton that recognizes the formula. The latter is not semantic because no canonical representation for the automata that recognize LTL formulas exists.

Our definition of semantic vacuity can be used for reasoning about any property for which past behaviors can influence its satisfaction. That is, semantic vacuity is well-suited for all ω-regular languages that do not define a fairness property (see [32] for a formal definition of fairness): semantic vacuity cannot be readily used to reason about limit behaviors of fairness properties. This is because, for a fairness property φ, the relation $\sim_\varphi$ has only one equivalence class, and consequently the obligation abstraction degenerates. Note that reasoning about limit behaviors is challenging also when using syntactic definitions of vacuity. It is easy to see that, for instance, the formulas $\Box\Diamond a$ and $\Diamond\Box a$, which define fairness properties, are never syntactically vacuous. As future work, we plan to extend semantic vacuity to limit behaviors, for instance using the notion of right congruences for ω languages [33].

References

[1] D. L. Beatty and R. E. Bryant, "Formally verifying a microprocessor using a simulation methodology," in *Proceedings of the 31st Annual Design Automation Conference*, ser. DAC '94. New York, NY, USA: ACM, 1994, pp. 596–602. [Online]. Available: http://doi.acm.org/10.1145/196244.196575

[2] I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh, "Efficient detection of vacuity in ACTL formulas," in *CAV*, ser. Lecture Notes in Computer Science, O. Grumberg, Ed., vol. 1254. Springer, 1997, pp. 279–290.

[3] R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi, "Enhanced vacuity detection in linear temporal logic," in *15th International Conference on Computer Aided Verification*, ser. LNCS, vol. 2725. Boulder, CO, USA: Springer, Jul. 2003, pp. 368–380. [Online]. Available: 2003/AFFGPTV03.html

[4] H. Chockler, A. Gurfinkel, and O. Strichman, "Beyond vacuity: Towards the strongest passing formula," *Form. Methods Syst. Des.*, vol. 43, no. 3, pp. 552–571, Dec. 2013.

[5] A. Gurfinkel and M. Chechik, "Extending extended vacuity," in *Formal Methods in Computer-Aided Design, 5th International Conference, FM-CAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings*, ser. Lecture Notes in Computer Science, A. J. Hu and A. K. Martin, Eds., vol. 3312. Springer, 2004, pp. 306–321.

[6] ——, "How vacuous is vacuous?" in *Tools and Algorithms for the Construction and Analysis of Systems, 10th International Conference, TACAS 2004, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2004, Barcelona, Spain, March 29 - April 2, 2004, Proceedings*, ser. Lecture Notes in Computer Science, K. Jensen and A. Podelski, Eds., vol. 2988. Springer, 2004, pp. 451–466.

[7] O. Kupferman and M. Y. Vardi, "Vacuity detection in temporal model checking," in *CHARME*, ser. Lecture Notes in Computer Science, L. Pierre and T. Kropf, Eds., vol. 1703. Springer, 1999, pp. 82–96.

[8] K. Etessami and T. Wilke, "An until hierarchy and other applications of an Ehrenfeucht-Fraïssé game for temporal logic," *Inf. Comput.*, vol. 160, no. 1-2, pp. 88–108, 2000.

[9] "Property patterns for LTL," http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml.

[10] D. Fisman, O. Kupferman, S. Sheinvald-Faragy, and M. Y. Vardi, "A framework for inherent vacuity," in *Haifa Verification Conference*, ser. Lecture Notes in Computer Science, H. Chockler and A. J. Hu, Eds., vol. 5394. Springer, 2008, pp. 7–22.

[11] G. Petric Maretić, M. Torabi Dashti, and D. Basin, "Anchored LTL separation," in *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, ser. CSL-LICS '14. New York, NY, USA: ACM, 2014, pp. 74:1–74:9. [Online]. Available: http://doi.acm.org/10.1145/2603088.2603139

[12] D. M. Gabbay, "The declarative past and imperative future: Executable temporal logic for interactive systems," in *Temporal Logic in Specification*, ser. Lecture Notes in Computer Science, B. Banieqbal, H. Barringer, and A. Pnueli, Eds., vol. 398. Springer, 1987, pp. 409–448.

[13] I. M. Hodkinson and M. Reynolds, "Separation - past, present, and future," in *We Will Show Them! (2)*, S. N. Artëmov, H. Barringer, A. S. d'Avila Garcez, L. C. Lamb, and J. Woods, Eds. College Publications, 2005, pp. 117–142.

[14] N. Markey, "Temporal logic with past is exponentially more succinct, concurrency column," *Bulletin of the EATCS*, vol. 79, pp. 122–128, 2003.

[15] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," in *CAV*, ser. Lecture Notes in Computer Science, G. Berry, H. Comon, and A. Finkel, Eds., vol. 2102. Springer, 2001, pp. 53–65.

[16] A. P. Sistla and E. M. Clarke, "The complexity of propositional linear temporal logics," *J. ACM*, vol. 32, no. 3, pp. 733–749, 1985.

[17] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification (preliminary report)," in *LICS*. IEEE Computer Society, 1986, pp. 332–344.

[18] E. M. Clarke, Jr., O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, MA, USA: MIT Press, 1999.

[19] G. J. Holzmann, "The model checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, pp. 279–295, 1997.

[20] D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi, "On the temporal basis of fairness," in *POPL*, P. W. Abrahams, R. J. Lipton, and S. R. Bourne, Eds. ACM Press, 1980, pp. 163–173.

[21] S. Ben-David, D. Fisman, and S. Ruah, "Temporal antecedent failure: Refining vacuity," in *In Proc. 18th CONCUR, LNCS 4703*, 2007, pp. 492–506.

[22] A. Nerode, "Linear automaton transformations," in *AMS*, vol. 9. AMS, 1958.

[23] M. Kamel and S. Leue, "Validation of the general inter-orb protocol (GIOP) using the Spin model-checker," in *In Software Tools for Technology Transfer*. SpringerVerlag, 1998, pp. 394–409.

[24] G. Singh and S. K. Shukla, "Verifying compiler based refinement of BluespecTM," in *Model Checking Software, 15th International SPIN Workshop, Los Angeles, CA, USA, August 10-12, 2008, Proceedings*, ser. Lecture Notes in Computer Science, K. Havelund, R. Majumdar, and J. Palsberg, Eds., vol. 5156. Springer, 2008, pp. 250–269.

[25] "Promela database," http://www.albertolluch.com/research/promelamodels.

[26] T. Ball and O. Kupferman, "Vacuity in testing," in *TAP*, ser. Lecture Notes in Computer Science, B. Beckert and R. Hähnle, Eds., vol. 4966. Springer, 2008, pp. 4–17.

[27] D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M. Vardi, "Regular vacuity," in *13th Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, ser. Lecture Notes in Computer Science, vol. 3725. Springer-Verlag, 2005, pp. 191–206.

[28] M. Purandare, T. Wahl, and D. Kroening, "Strengthening properties using abstraction refinement," in *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, April 2009, pp. 1692–1697.

[29] O. Kupferman, "Sanity checks in formal verification," in *Proceedings of the 17th International Conference on Concurrency Theory*, ser. CONCUR'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 37–51.

[30] M. W. Whalen, A. Rajan, M. P. Heimdahl, and S. P. Miller, "Coverage metrics for requirements-based testing," in *Proceedings of the 2006 International Symposium on Software Testing and Analysis*, ser. ISSTA '06. New York, NY, USA: ACM, 2006, pp. 25–36. [Online]. Available: http://doi.acm.org/10.1145/1146238.1146242

[31] L. Tan, "State coverage metrics for specification-based testing with Büchi automata," in *Tests and Proofs - 5th International Conference, TAP 2011, Zurich, Switzerland, June 30 - July 1, 2011. Proceedings*, ser. Lecture Notes in Computer Science, M. Gogolla and B. Wolff, Eds., vol. 6706. Springer, 2011, pp. 171–186.

[32] A. P. Sistla, "Safety, liveness and fairness in temporal logic," *Formal Asp. Comput.*, vol. 6, no. 5, pp. 495–512, 1994.

[33] O. Maler and L. Staiger, "On syntactic congruences for ω-languages," *Theoretical Computer Science*, vol. 183, pp. 93–112, 1997.