Robust Vacuity for Branching Temporal Logic

ARIE GURFINKEL
Software Engineering Institute, Carnegie Mellon University
and
MARSHA CHECHIK
University of Toronto

There is a growing interest in techniques for detecting whether a logic specification is satisfied too easily, or *vacuously*. For example, the specification "every request is eventually followed by an acknowledgment" is satisfied vacuously by a system that never generates any requests. Vacuous satisfaction misleads users of model-checking into thinking that a system is correct. It is a serious problem in practice.

There are several existing definitions of vacuity. Originally, Beer et al. formalized *vacuity* as insensitivity to syntactic perturbation (*syntactic vacuity*). This formulation captures the intuition of "vacuity" when applied to a single occurrence of a subformula. Armoni et al. argued that vacuity must be *robust* – not affected by semantically invariant changes, such as extending a model with additional atomic propositions. They show that syntactic vacuity is not robust for subformulas of linear temporal logic, and propose an alternative definition – *trace vacuity*.

In this article, we continue this line of research. We show that trace vacuity is not robust for branching time logic. We further refine the notion of vacuity so that it applies uniformly to linear and branching time logic and does not suffer from the common pitfalls of prior definitions. Our new definition – *bisimulation vacuity* – is a proper and non-trivial extension of both syntactic and trace vacuity. We discuss the complexity of detecting bisimulation vacuity, and identify several practically-relevant subsets of CTL* for which vacuity detection problem is reducible to model-checking. We believe that in most practical applications, bisimulation vacuity provides both the desired theoretical properties and is tractable computationally.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Model checking

General Terms: Verification

Additional Key Words and Phrases: Vacuity detection

1. INTRODUCTION

Model-checking gained wide popularity as an automated technique for effective analysis of software and hardware systems. Given a temporal logic property, the model-checker automatically determines whether the property is satisfied by the system, giving a counterexample in case of the failure.

Yet a major problem in practical applications of model-checking is that a successful run of the model-checker does not necessarily guarantee that the intended requirement is satisfied by the system [Beer et al. 1997; Beatty and Bryant 1994]. For example, consider the property

> "every request must be followed by an acknowledgment",

where the environment controls the requests. This property, expressed in CTL as $AG(\text{req} \Rightarrow AF\text{ack})$, is satisfied vacuously[1] by any system that never produces a

[1]Beatty and Briant [Beatty and Bryant 1994] originally called this problem "antecedent failure".

request (i.e., req is false in all reachable states). In this case, the environment alone ensures satisfaction of this property, so it is true of *any* system combined with such an environment. Intuitively, a property φ is considered vacuous if it contains a subformula that is irrelevant for φ's satisfaction by the system. In the above example, it is AFack.

Researchers at the IBM Haifa Research Laboratory observed that vacuity is a serious problem [Beer et al. 1997] and that "... typically 20% of specifications pass vacuously during the first formal verification runs of a new hardware design, and that vacuous passes always point to a real problem in either the design, or its specification, or the environment" [Beer et al. 1997]. Further justification has been given by several researchers, such as the case study by Purandare and Somenzi [Purandare and Somenzi 2002]. These results led to a substantial interest in techniques for detecting vacuity.

Most of the early work on vacuity detection uses a *syntactic* definition of vacuity, provided by Beer et al. [Beer et al. 2001]: a formula φ is syntactically vacuous in a subformula ψ and model K, if replacing ψ by any other temporal logic formula x, denoted $\varphi[\psi \leftarrow x]$, does not affect the satisfaction of φ in K. That is, φ is vacuous if $\forall x \in TL \cdot \varphi[\psi \leftarrow x]$ is true, where TL stands for a temporal logic. The main advantage of this definition is the simplicity of detecting vacuity in an occurrence of a subformula. That is, whenever ψ occurs in φ only once, detecting whether φ is syntactically vacuous in ψ reduces to model-checking $\varphi[\psi \leftarrow \mathsf{true}]$ or $\varphi[\psi \leftarrow \mathsf{false}]$, based on the polarity of ψ. This result started a line of research, e.g., [Dong et al. 2002; Kupferman and Vardi 2003; Gurfinkel and Chechik 2004b; Bustan et al. 2005; Tzoref and Grumberg 2006], that aims to increase the scope of applicability of vacuity detection algorithms. In particular, this work deals with deciding vacuity for various temporal logics, for formulas with one or multiple occurrences of a subformula, handling vacuous satisfaction and vacuous failure of formulas, and generating witnesses to non-vacuity.

An orthogonal question, raised by Armoni et al. [Armoni et al. 2003] and continuing in this article, is to reexamine the *meaning* of vacuity. Armoni et al. showed that the definition of syntactic vacuity is too restrictive. It is not well suited for detecting vacuity with respect to multiple occurrences of a subformula, i.e., deciding whether $(AXp) \vee (AX\neg p)$ is vacuous in p. Furthermore, it is sensitive to irrelevant changes to the model. For example, syntactic vacuity of a formula 'if p is true now, it will remain true in the next state", expressed in CTL as $AG(p \Rightarrow AXp))$, can be affected, i.e., changed from vacuous to non-vacuous, by simply adding new atomic propositions to the model.

As an alternative, the authors of [Armoni et al. 2003] develop a new definition, applicable to linear-time logic, called *trace vacuity*. Trace vacuity is not syntactic, but is based on the semantics of quantified temporal logic. The new definition is shown to alleviate the problems of syntactic vacuity (at least on the examples tried by the authors). Furthermore, the complexity of detecting vacuous satisfaction for LTL properties with respect to trace vacuity is in the same complexity class as model-checking.

In this article, we continue the search for the "right" definition of vacuity, and whether this definition changes as we transition from LTL properties to CTL* and

from vacuous satisfaction (i.e., vacuity of formulas that are satisfied by the model) to vacuous failure (i.e., vacuity of formulas that are violated by the model). In particular, we develop a robust definition of vacuity, which we call *bisimulation* vacuity. We start with a definition of vacuity for propositional logic, argue that it is robust, and then systematically extend it to branching-time temporal logic CTL*. We show that bisimulation vacuity is a proper extension of syntactic vacuity: while syntactic and bisimulation vacuity coincide for vacuity in a single occurrence, syntactic vacuity is not robust when applied to vacuity in multiple occurrences. Bisimulation vacuity is also a proper non-trivial extension of trace vacuity: while the bisimulation and the trace vacuity definitions coincide for LTL, trace vacuity is not robust when applied to branching-time logics.

We study the complexity of detecting bisimulation vacuity. In general, this problem is EXPTIME-complete for CTL and 2EXPTIME-complete for CTL*. However, we identify several important fragments of CTL* for which vacuity detection, or at least detecting vacuous satisfaction, is no harder than model-checking. In particular, we show that checking vacuous satisfaction of ACTL* is reducible to model-checking, which subsumes the results of [Armoni et al. 2003].

The rest of the article is organized as follows. We provide the necessary background in Section 2. In Section 3, we examine the meaning of "robustness" of vacuity, define bisimulation vacuity, and argue that it is robust. In Section 4, we study complexity of detecting bisimulation vacuity for CTL* and identify subsets of this language where this problem is tractable. We analyze the relationship between vacuity and abstraction in Section 5. We then compare our approach with related work in Section 6 and conclude in Section 7.

2. BACKGROUND

In this section, we give a brief overview of temporal logic model-checking, property reserving relations, and several semantics of quantified temporal logic.

2.1 Models of Computation

We use Kripke structures to model computations. Intuitively, these are transition systems whose states are labeled by atomic propositions. In this section, we review the formal definition of Kripke structures, and fix the notation.

We use $\mathbf{2}$ to denote the set of boolean values $\{\mathsf{true}, \mathsf{false}\}$.

Definition 2.1 *Kripke Structure.* A *Kripke structure* K is a tuple (AP, S, R, s_0, I), where AP is a set of atomic propositions, S is a finite set of states, $R \subseteq S \times S$ is a total transition relation, $s_0 \in S$ is a designated initial state, and $I : S \to \mathbf{2}^{AP}$ is a labeling function, assigning a value to each atomic proposition $p \in AP$ in each state.

Example Kripke structures are shown in Figures 1 and 3. For two states s and t, we write $R(s,t)$ for $(s,t) \in R$, and $R(s)$ to denote the set of successors of R:

$$R(s) \triangleq \{t \in S \mid R(s,t)\}.$$

For notational convenience, we denote components of a Kripke structure K using the same typographical convention as used for K. For example, S' denotes the statespace of K', R' – its transition relation, AP' – the set of atomic propositions,

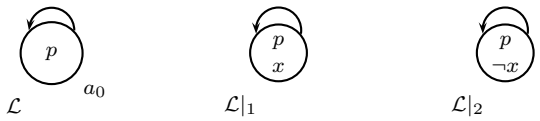


Fig. 1. A Kripke structure $\mathcal{L}$ and its $\{x\}$-variants $\mathcal{L}|_1$ and $\mathcal{L}|_2$.

etc. A *path* π of K is an infinite sequence of states in which every consecutive pair of states is related by the transition relation. Let i be a non-negative integer. We write $\pi(i)$ to denote the $i+1$th state on the path, $\pi(0)$ to denote the first state, and π_i to denote the suffix of π starting from the ith state. The set of all paths of K starting from a state s is denoted by Π_s^K (K is often omitted when clear from the context).

We now define parallel synchronous composition.

Definition 2.2 *Parallel Synchronous Composition.* Let $K_1 = (AP_1, S_1, R_1, s_1^0, I_1)$, and $K_2 = (AP_2, S_2, R_2, s_2^0, I_2)$ be two Kripke structures with disjoint atomic propositions, i.e., $AP_1 \cap AP_2 = \emptyset$. A *parallel synchronous composition* of K_1 and K_2, written $K_1||K_2$, is a Kripke structure $(AP_1 \cup AP_2, S_1 \times S_2, R_{||}, (s_1^0, s_2^0), I_{||})$, where

$$R_{||}((s,t),(s',t')) \Leftrightarrow R_1(s,s') \wedge R_2(t,t')$$
$$I_{||}((s,t)) \triangleq I_1(s) \cup I_2(t)\,.$$

A computation tree $T(K)$ of a Kripke structure K is an S-labeled tree obtained by unrolling K from its initial state.

Definition 2.3 *Computation Tree.* Let $K = (AP, S, R, s_0, I)$ be a Kripke structure. A *computation tree* $T(K)$ of K is an S-labeled tree (T, τ), where $T = (V, E)$ is a tree with vertex set V and edge set E, and $\tau : V \to S$ is a labeling function, satisfying the "unrolling" conditions:

(1) if v is a root of $T(K)$, then $\tau(v) = s_0$;
(2) for a node v, $|E(v)| = |R(\tau(v))|$, and for each $s \in R(\tau(v))$ there exists a $u \in E(v)$ such that $\tau(u) = s$, where $E(v)$ is the set of successors of v.

A tree unrolling $T(\mathcal{L})$ for a structure $\mathcal{L}$ in Figure 1 is shown in Figure 2. Note that since $\mathcal{L}$ has only one transition, the unrolling is a unary tree, i.e., a trace.

2.2 Temporal Logic

Computation Tree Logic CTL* [Emerson and Halpern 1985] is a branching-time temporal logic constructed from propositional connectives, temporal operators X (next), U (until), F (future), and G (globally), and path quantifiers A (forall) and E (exists).

Definition 2.4 *Syntax of* CTL*. Temporal logic CTL* denotes the set of all *state formulas* satisfying the grammar

$$\varphi ::= p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid A\psi \mid E\psi,$$

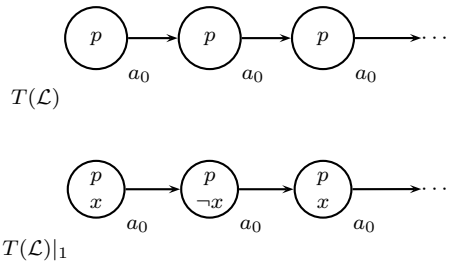


Fig. 2. A tree unrolling $T(\mathcal{L})$ of $\mathcal{L}$ and one of its $\{x\}$-variant $T(\mathcal{L})|_1$.

where p is an atomic proposition, and ψ is a *path formula* satisfying the grammar

$$\psi ::= \varphi \mid X\psi \mid \psi\, U\, \psi \mid \psi\, \tilde{U}\, \psi \mid F\psi \mid G\psi.$$

The semantics of path formulas is given with respect to a path of a Kripke structure. For a path formula ψ, we write $K, \pi \models \psi$ to denote that ψ is satisfied by the path π of a Kripke structure K. The semantics of state formulas is given with respect to a state of a Kripke structure. For a state formula φ, we write $K, s \models \varphi$ to denote that φ is satisfied in the state s in K.

Definition 2.5 *Semantics of* CTL*. Let $K = (AP, S, R, s_0, I)$ be a Kripke structure. The semantics of path and state formulas is defined as follows, where φ, φ_1, and φ_2 denote state formulas, and ψ, ψ_1, and ψ_2 denote path formulas, and i, j,

and k are natural numbers:

$$
\begin{aligned}
K, \pi \models \varphi &\triangleq K, \pi(0) \models \varphi \\
K, \pi \models \neg\psi &\triangleq K, \pi \not\models \psi \\
K, \pi \models \psi_1 \wedge \psi_2 &\triangleq K, \pi \models \psi_1 \wedge K, \pi \models \psi_2 \\
K, \pi \models \psi_1 \vee \psi_2 &\triangleq K, \pi \models \psi_1 \vee K, \pi \models \psi_2 \\
K, \pi \models X\psi &\triangleq K, \pi_1 \models \psi \\
K, \pi \models \psi_1 \, U \, \psi_2 &\triangleq \exists j \cdot K, \pi_j \models \psi_2 \wedge \forall 0 \leq i < j \cdot K, \pi_i \models \psi_1 \\
K, \pi \models \psi_1 \, \tilde{U} \, \psi_2 &\triangleq \forall j \cdot K, \pi_j \not\models \psi_2 \Rightarrow \exists 0 \leq i < j \cdot K, \pi_i \models \psi_1 \\
K, \pi \models F\psi &\triangleq \exists j \cdot K, \pi_j \models \psi \\
K, \pi \models G\psi &\triangleq \forall j \cdot K, \pi_j \models \psi
\end{aligned}
$$

$$
\begin{aligned}
K, s \models p &\triangleq p \in I(s) \\
K, s \models \neg\varphi &\triangleq K, s \not\models \varphi \\
K, s \models \varphi_1 \wedge \varphi_2 &\triangleq K, s \models \varphi_1 \wedge K, s \models \varphi_2 \\
K, s \models \varphi_1 \vee \varphi_2 &\triangleq K, s \models \varphi_1 \vee K, s \models \varphi_2 \\
K, s \models A\varphi &\triangleq \bigwedge_{\pi \in \Pi_s^K} \pi \models \varphi \\
K, s \models E\varphi &\triangleq \bigvee_{\pi \in \Pi_s^K} \pi \models \varphi
\end{aligned}
$$

We say that K satisfies φ (or φ holds in K), denoted $K \models \varphi$, iff φ holds in the designated initial state: $K, s_0 \models \varphi$. For simplicity of presentation, we use sets of states as atomic propositions in temporal formulas, giving them the following interpretation: for a set of states Y,

$$K, s \models Y \triangleq s \in Y \, .$$

We write $\varphi[x]$ to indicate that the formula φ may contain an occurrence of x. An occurrence of x in φ is *positive* (or of *positive polarity*) if x occurs under the scope of an even number of negations, and *negative* otherwise. For example, p is positive in $\neg EX\neg p$, and negative in $\neg EXp$. A subformula x is *pure* in φ if all of its occurrences have the same polarity. For example, p is pure in $EF(p \wedge q \wedge EGp)$. We write $\varphi[x \leftarrow y]$ for a formula obtained from φ by replacing *each occurrence* of x by y. This is equivalent to treating a formula as a DAG with all common subformulas shared.

A formula φ is *universal* (i.e., in the language ACTL*) if all of its temporal path quantifiers are universal, and is *existential* (i.e., in the language ECTL*) if all of the path quantifiers are existential. In both cases, negation is only allowed at the level of atomic propositions. For example, $AG(p \Rightarrow AFq)$ is in ACTL*, and $EF(p \wedge EG\neg q)$ is in ECTL*. We extend this to subformulas as well and say that a subformula is universal if it occurs only under the scope of universal path quantifiers in negation normal form of the formula.

The fragment of CTL* in which all formulas are of the form $A\psi$, where ψ is a path formula, is called *Linear Temporal Logic* (LTL) [Pnueli 1977]. The fragment in which every occurrence of a path quantifier is immediately followed by a temporal operator is called *Computation Tree Logic* (CTL) [Clarke and Emerson 1981]. For example, $AG(pUq)$ is an LTL formula, and $AGA[p\,U\,q]$ is a CTL formula. More details on temporal logic can be found in [Emerson 1990; Clarke et al. 1999].

2.3 Simulation and Bisimulation

In this section, we review two property preserving relations between Kripke structures: simulation and bisimulation.

Definition 2.6 *Simulation.* [Milner 1971] Let $K = (AP, S, R, s_0, I)$ and $K' = (AP', S', R', s'_0, I')$ be two Kripke structures and $X \subseteq (AP \cap AP')$ a set of common atomic proposition. A relation $\rho \subseteq S \times S'$ is a *simulation* relation with respect to X if and only if $\rho(s, s')$ implies that

(1) $I'(s') \cap X = I(s) \cap X$, and
(2) $\forall t' \in S' \cdot R'(s', t') \Rightarrow \exists t \in S \cdot R(s, t) \wedge \rho(t, t')$.

A state s simulates a state s' if $(s, s') \in \rho$. A Kripke structure K *simulates* K' iff the initial state of K' is simulated by the initial state of K. For example, $\mathcal{M}$ in Figure 3 simulates $\mathcal{L}$ in Figure 1 via the relation

$$\rho^{\mathcal{L}}_{\mathcal{M}} = \{(b_0, a_0), (b_1, a_0)\}.$$

Intuitively, If K simulates K' then K can match every behavior of K', i.e., the set of all behaviors of K' is a subset of those of K. Thus, if K satisfies an ACTL* formula, then so does K'.

THEOREM 2.7. *[Browne et al. 1988; Grumberg and Long 1994] Let K and K' be two Kripke structures such that K simulates K'. Then, for any* ACTL* *formula* φ

$$K \models \varphi \Rightarrow K' \models \varphi\,.$$

A simulation relation whose inverse is also a simulation is called a bisimulation:

Definition 2.8 *Bisimulation.* Let $K = (AP, S, R, s_0, I)$ and $K' = (AP', S', R', s'_0, I')$ be two Kripke structures and $X \subseteq (AP \cap AP')$ a set of common atomic proposition. A relation $\rho \subseteq S \times S'$ is a *bisimulation* relation with respect to X if and only if (a) ρ is a simulation relation between K and K' with respect to X, and (b) $\rho^{-1} \subseteq S' \times S$ is a simulation relation between K' and K with respect to X.

Two structures K and K' are *bisimilar* iff there exists a bisimulation relation ρ that relates their initial states. We use $\mathcal{B}(K)$ to denote the set of all structures bisimilar to K with respect to all of the atomic propositions of K. For example, the inverse of the relation $\rho^{\mathcal{L}}_{\mathcal{M}}$ above is a simulation as well. Thus, $\mathcal{L}$ and $\mathcal{M}$ are bisimilar.

Intuitively, if K and K' are bisimilar, then they have equivalent behaviors. The theorem below also indicates that they satisfy the same temporal logic formulas.

THEOREM 2.9. *[Browne et al. 1988] Let K and K' be two bisimilar Kripke structures. Then, for any* CTL* *formula φ,*

$$K \models \varphi \Leftrightarrow K' \models \varphi\,.$$

It is possible to extend the definition of bisimulation to infinite-state models. Under such an interpretation, a computation tree $T(K)$ of a Kripke structure K is bisimilar to K. This is sufficient to show that a CTL* formula cannot distinguish between a Kripke structure and its tree unrolling, i.e., $K \models \varphi \Leftrightarrow T(K) \models \varphi$. This fact is often used to give semantics of CTL* with respect to a computation tree of a Kripke structure instead of with respect to the Kripke structure itself. We say that CTL* is *bisimulation closed*. Note that not all temporal logics share this property. In particular, some quantified temporal logics that are used in this article (see Section 2.4) are not bisimulation closed.

2.4 Quantified Temporal Logic

Quantified Temporal Logic (QCTL*) extends the syntax of CTL* with universal ($\forall$) and existential ($\exists$) quantifiers over atomic propositions [Kupferman 1997]. For example, $\forall x \cdot EF(x \Rightarrow EF(\neg x))$ is a QCTL* formula. Here, we consider a fragment in which only a single occurrence of a quantifier is allowed, i.e.,

$$\{\varphi, \forall x \cdot \varphi, \exists x \cdot \varphi \mid \varphi \in \text{CTL}^*\}.$$

For simplicity, we still call this fragment QCTL*.

There are several different definitions of semantics of QCTL* with respect to a Kripke structure; we consider three of these: *structure* [Kupferman 1997], *tree* [Kupferman 1997], and *bisimulation* which is introduced in [French 2001] under the name *amorphous*.

Structure Semantics. Under structure semantics [Kupferman 1997], each bound variable x is interpreted as a subset of the statespace. A universally quantified formula $\forall x \cdot \varphi$ is satisfied by K under this semantics if replacing x by an arbitrary set always results in a formula that is satisfied by K.

Definition 2.10 *Structure Semantics.* [Kupferman 1997] Let K be a Kripke structure, and φ a CTL* formula. *Structure semantics* of QCTL*, written $K \models_s \varphi$, is defined as follows:

$$\begin{aligned} K &\models_s \varphi \triangleq K \models \varphi \\ K &\models_s \forall x \cdot \varphi \triangleq \forall Y \subseteq S \cdot K \models \varphi[x \leftarrow Y] \\ K &\models_s \exists x \cdot \varphi \triangleq \exists Y \subseteq S \cdot K \models \varphi[x \leftarrow Y]\,. \end{aligned}$$

That is, a formula $\forall x \cdot \varphi[x]$ is satisfied by K under structure semantics if $\varphi[x]$ is true in K under any interpretation of the atomic proposition x.

An equivalent and more constructive definition can be given as well. Let K_{-x}, pronounced "K minus x", denote the result of removing an atomic proposition x from K. Formally,

$$K_{-x} \triangleq K \text{ with } AP_{-x} = AP \setminus \{x\}.$$

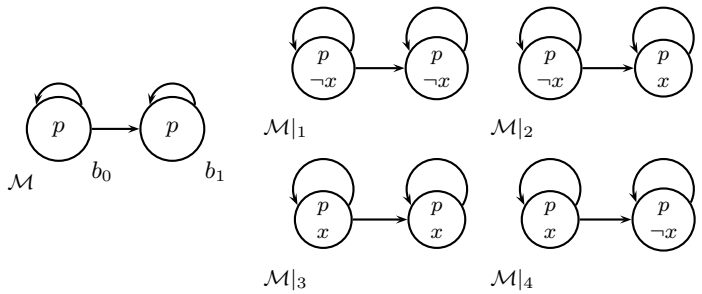


Fig. 3. A Kripke structure $\mathcal{M}$ and its x-variants: $\mathcal{M}|_1$, $\mathcal{M}|_2$, $\mathcal{M}|_3$, and $\mathcal{M}|_4$.

		Quantification Semantics		
Model	**Property**	Structure	Tree	Bisimulation
$\mathcal{L}$	$\forall x \cdot P_1$	true	false	false
$\mathcal{M}$	$\forall x \cdot P_1$	false	false	false
$\mathcal{L}$	$\forall x \cdot P_2$	true	true	false
$\mathcal{M}$	$\forall x \cdot P_2$	false	false	false
$\mathcal{L}$	$\forall x \cdot P_3$	true	true	true
$\mathcal{M}$	$\forall x \cdot P_3$	true	true	true

Table I. Satisfaction of QTL formulas $\forall x \cdot P_1$, $\forall x \cdot P_2$, and $\forall x \cdot P_3$ on models $\mathcal{L}$ and $\mathcal{M}$ under different semantics of QTL.

An *x-variant* of a Kripke structure K is a structure K' such that K'_{-x} is identical to K. For example, the set of all x-variants of $\mathcal{L}$ is shown in Figure 1. A formula $\forall x \cdot \varphi[x]$ is satisfied by a Kripke structure K under structure semantics if and only if $\varphi[x]$ is satisfied by *every* x-variant of K. This follows immediately from the one-to-one correspondence between subsets of the statespace of K and labeling of x in an x-variant.

We illustrate this semantics using the following formulas:

$$P_1 \triangleq AG(x \Rightarrow AXx);$$
$$P_2 \triangleq AG((AXx) \vee (AX\neg x));$$
$$P_3 \triangleq A((Xx) \vee (X\neg x)).$$

$\mathcal{L} \models_s \forall x \cdot P_1$ since P_1 is satisfied by all x-variants of $\mathcal{L}$ (see Figure 1), but $\mathcal{M} \not\models_s \forall x \cdot P_1$ since P_2 is not satisfied by the x-variant $\mathcal{M}|_4$ of $\mathcal{M}$ (see Figure 3). The results of evaluating the rest of the formulas on $\mathcal{L}$ and $\mathcal{M}$ are summarized in the first three columns of Table I.

Tree Semantics. Under the tree semantics [Kupferman 1997], QCTL* formulas are interpreted with respect to variants of a computation tree $T(K)$ of a Kripke structure K.

Definition 2.11 *Tree Semantics.* [Kupferman 1997] Let K be a Kripke structure, and φ a CTL* formula. *Tree semantics* of QCTL*, written $K \models_T \varphi$, is

defined as follows:

$$K \models_T \varphi \triangleq T(K) \models \varphi$$
$$K \models_T \forall x \cdot \varphi \triangleq T(K) \models_s \forall x \cdot \varphi$$
$$K \models_T \exists x \cdot \varphi \triangleq T(K) \models_s \exists x \cdot \varphi \ .$$

That is, a formula $\forall x \cdot \varphi[x]$ is satisfied by K under tree semantics if and only if it is satisfied by every x-variant of the *computation tree* of K. For example, $\mathcal{L} \not\models_T \forall x \cdot P_1$ since P_1 is not satisfied by an x-variant $T(\mathcal{L})|_1$ of $T(\mathcal{L})$ shown in Figure 2, and $\mathcal{L} \models_T \forall x \cdot P_2$ since every state in the tree unrolling $T(\mathcal{L})$ of $\mathcal{L}$ has exactly one successor. A few additional examples are given in the middle column of Table I. We note that QCTL* under structure and tree semantics is not bisimulation closed [Kupferman 1997].

Bisimulation Semantics. Prior to presenting bisimulation semantics, we need to introduce a notion of x-bisimulation. Let K and K' be two Kripke structures. The structure K' is *x-bisimilar* to K if and only if (a) the atomic propositions AP' of K' extend atomic propositions AP of K with a single atomic proposition x, i.e., $AP' = AP \cup \{x\}$, and (b) K' and K are bisimilar with respect to AP. That is, K' has exactly the same behaviors as K, except for the interpretation of an additional atomic proposition x. For a Kripke structure K, we use $\mathcal{B}_x(K)$ to denote the set of all structures x-bisimilar to K. For example, the x-variant $\mathcal{M}|_4$ of $\mathcal{M}$ is $\{x\}$-bisimilar to $\mathcal{M}$. $\mathcal{M}|_4$ is also $\{x\}$-bisimilar to $\mathcal{L}$. It is easy to observe that in general, the set $\mathcal{B}_x(K)$ includes all x-variants of the structure K, every structure bisimilar to K, and every x-variant of a structure bisimilar to K. The above statement is included here just for clarity.

We are now ready to define bisimulation semantics. Under bisimulation semantics, QCTL* formulas are interpreted with respect to bisimulation variants of a Kripke structure.

Definition 2.12 *Bisimulation (Amorphous) Semantics.* [French 2001] Let K be a Kripke structure, and φ a CTL* formula. *Bisimulation semantics* of QCTL*, written $K \models_b \varphi$, is defined as follows:

$$K \models_b \varphi \triangleq K \models \varphi$$
$$K \models_b \forall x \cdot \varphi \triangleq \forall K' \in \mathcal{B}_x(K) \cdot K' \models \varphi$$
$$K \models_b \exists x \cdot \varphi \triangleq \exists K' \in \mathcal{B}_x(K) \cdot K' \models \varphi \ .$$

That is, a formula $\forall x \cdot \varphi$ is satisfied by K under bisimulation semantics if and only if φ is satisfied by every x-bisimulation of K. For example, $\mathcal{L} \not\models_b \forall x \cdot P_2$ since (a) $\mathcal{M}$ is bisimilar to $\mathcal{L}$, (b) any x-variant of $\mathcal{M}$ is x-bisimilar to $\mathcal{L}$, and (c) P_2 is not satisfied by the x-variant $\mathcal{M}|_4$ of $\mathcal{M}$ (see Figure 3). On the other hand, $\mathcal{L} \models_b \forall x \cdot P_3$ since P_3 is a temporal logic tautology, i.e., it is true in any model. A few additional examples are given in the last column of Table I.

Note that each semantics extends the range of the interpretation of the quantifiers. Thus, it is harder to satisfy a universal formula under bisimulation semantics than under tree or structure semantics. The following theorem formalizes the rela-

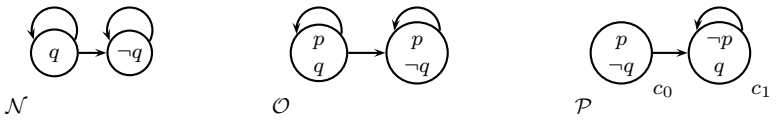


Fig. 4. Sample models $\mathcal{N}$, $\mathcal{O}$, and $\mathcal{P}$.

tionship between all three QCTL* semantics, and is a corollary of a similar theorem proved by French [French 2001].

THEOREM 2.13. *Let $\forall x \cdot \varphi$ be a* QCTL* *formula, and K a Kripke structure. Then, the following is true*

$$(K \models_b \forall x \cdot \varphi) \Rightarrow (K \models_T \forall x \cdot \varphi) \Rightarrow (K \models_s \forall x \cdot \varphi) \ .$$

Furthermore, the implications are strict.

PROOF. The theorem follows from the fact that every tree unrolling of an x-variant of K is an x-variant of $T(K)$ and that a tree unrolling $T(K)$ is bisimilar to K. Strictness of the first and the second implication is established by the examples in row 3 and row 1 of Table I, respectively. □

3. TOWARDS DEFINING VACUITY

The first formal definition of vacuity is called *propositional antecedent failure* and was described by Beatty and Bryant [Beatty and Bryant 1994]. A formula of the form $AG(p \Rightarrow q)$ suffers from antecedent failure on a model K if its antecedent p is not satisfiable in K. In particular, this means that the consequent (or the right-hand side) of the implication does not effect the validity of the formula.

Beer et al. [Beer et al. 2001] have generalized antecedent failure to arbitrary temporal formulas, calling the result *temporal vacuity*. Informally, if a formula φ contains a subformula ψ such that replacing ψ by any other formula does not affect the value of φ, then φ is vacuous in ψ. Furthermore, [Beer et al. 2001] restricted vacuity to properties with *a single occurrence* of ψ. We call this definition *structural vacuity* and provide a formal definition below:

Definition 3.1 *Syntactic Vacuity.* [Beer et al. 1997] A formula φ in a temporal logic L is *syntactically vacuous* in a subformula ψ (assuming a single occurrence of ψ in φ) in a model K iff

$$\forall \psi' \in L \cdot K \models \varphi \Leftrightarrow K \models \varphi[\psi \leftarrow \psi'].$$

When φ is vacuous in ψ, we say φ is ψ-vacuous. A formula is vacuous if it is vacuous in any of its subformulas. According to Definition 3.1, non-vacuity of φ with respect to a subformula ψ is witnessed by a formula φ' of the form $\varphi' = \varphi[\psi \leftarrow \psi']$ for some $\psi' \in L$ such that $K \models \varphi$ and $K \not\models \varphi'$. For example, a non-vacuous satisfaction of $AG(r \Rightarrow AFa)$ with respect to AFa can be witnessed by falsification of $AG(r \Rightarrow \mathsf{false})$.

Definition 3.1 provides a useful generalization of antecedent failure. However, when Armoni et al. [Armoni et al. 2003] attempted to generalize syntactic vacuity

further (they called it *formula vacuity*), to deal with multiple occurrences of subformulas, they found that it has three major weaknesses: (1) it makes vacuity of too many formulas debatable, (2) it makes vacuity sensitive to changes in the model that do not (or should not) affect the formula, and (3) it makes vacuity sensitive to the syntax of the temporal logic. We illustrate these weaknesses using several examples inspired by (or sometimes taken directly from) Armoni et al. [Armoni et al. 2003].

Weakness 1. Consider the property

$$P_4 \triangleq AG((AXp) \vee (AX\neg p)) .$$

which means "in every state, the next valuation of p is computed deterministically", i.e., it is either true in all successors or false in all successors. This property can be vacuous in AXp or $AX\neg p$, since satisfaction of either disjunct is sufficient to satisfy the entire property. However, as we argue below, it should never be vacuous in p under any reasonable definition of vacuity. Our reasoning is as follows. Take any Kripke structure K. Every state of K has at least one successor, and the proposition p has some value in each successor of every state. Thus, the value of p directly influences the overall value of P_4. Hence, P_4 should not be vacuous in p, in any Kripke structure. However, according to syntactic vacuity from Definition 3.1, P_4 is p-vacuous in model $\mathcal{L}$ in Figure 1, since $\mathcal{L}$ satisfies P_4, $P_4[p \leftarrow \mathsf{true}]$, and $P_4[p \leftarrow \mathsf{false}]$. This example shows that vacuity of some syntactically vacuous formulas is debatable, and thus syntactic vacuity is not sufficiently strong.

Weakness 2. Consider again the property P_4 defined above. We have already shown that it is syntactically p-vacuous in $\mathcal{L}$. Next, consider models $\mathcal{N}$, and parallel synchronous composition $\mathcal{O} = \mathcal{L}||\mathcal{N}$ of $\mathcal{L}$ and $\mathcal{N}$, both shown in Figure 4. The composition does not affect any of the original properties that were satisfied by $\mathcal{L}$. However, it does affect the syntactic vacuity of P_4: P_4 is no longer syntactically p-vacuous in $\mathcal{O}$. In particular, $\mathcal{O}$ satisfies P_4 (just like $\mathcal{L}$), but refutes

$$P_4[p \leftarrow q] = AG((AXq) \vee (AX\neg q)) .$$

Thus, composing $\mathcal{L}$ with $\mathcal{N}$ "fixes" syntactic vacuity of P_4, even though $\mathcal{N}$ has no influence on satisfaction of P_4. This illustrates that syntactic vacuity is sensitive to "irrelevant" changes to the model.

Weakness 3. Consider the property $P_5 \triangleq A(Xq \Rightarrow XXq)$ and the model $\mathcal{P}$ in Figure 4. Assume that P_5 is interpreted in LTL. Since $\mathcal{P} \models P_5[q \leftarrow \psi]$ for any LTL formula ψ, P_5 is q-vacuous in $\mathcal{P}$ according to syntactic vacuity (see Definition 3.1).

Let X^{-1} denote the past operator meaning "in the previous state". Formally, $X^{-1}p$ is satisfied by a suffix π_j of a path π iff $j > 0$, and p is satisfied by the suffix π_{j-1}.

Let LTL+P denote LTL extended with the past operator. Interpreted in LTL+P,

P_5 is no longer syntactically q-vacuous! The witness to non-vacuity is

$$\begin{aligned} P_5[q \leftarrow X^{-1}p] &= A((XX^{-1}p) \Rightarrow (XXX^{-1}p)) \\ &= A(p \Rightarrow Xp), \end{aligned}$$

which is falsified by $\mathcal{P}$. That is, syntactic vacuity of a formula can change by re-interpreting the formula in a temporal logic with more operators (without changing the formula itself), allowing us to conclude that syntactic vacuity is sensitive to the syntax of the logic with respect to which the formula is defined.

In the rest of this section, we systematically develop a robust definition of vacuity of temporal logic. We explore several semantic definitions of vacuity starting with vacuity for propositional logic and ending with a new definition of vacuity for temporal logic. We argue that our definition is robust by showing that it is not affected by non-essential changes to the model, nor by the number of available logical operators. Note that unlike prior work [Beer et al. 2001; Armoni et al. 2003], we do not distinguish between vacuity with respect to a particular occurrence or several occurrences of a subformula. Instead, we present a uniform treatment of the definition of vacuity that would allow the user to make the distinction during use. While we base the treatment below on subformula vacuity, all of our results easily extend to vacuity with respect to arbitrary subsets of occurrences. Of course, when restricted to subformulas with a single occurrence, all of the definitions of vacuity used in this paper reduce to the original definition of Beer et al. [Beer et al. 2001].

3.1 Propositional Vacuity

We start our exploration of vacuity with propositional logic. A model of a propositional formula φ is just a boolean valuation of all atomic propositions of φ. The value of φ in a model is a boolean value, either true or false. Thus, we can check the dependence of φ on a subformula ψ by checking whether replacing ψ by constants true and false affects the value of φ. This leads to the following formal definition of propositional vacuity.

Definition 3.2 *Propositional Vacuity.* A propositional formula φ is *vacuous* in a subformula ψ, or simply ψ-vacuous, in a model K if and only if replacing ψ by true and false does not affect the value of φ:

$$(K \models \varphi[\psi \leftarrow \mathsf{true}]) \Leftrightarrow (K \models \varphi[\psi \leftarrow \mathsf{false}]) \ .$$

A propositional formula is vacuous if it is vacuous in some subformula ψ. Alternatively, vacuity of a propositional formula in a model K can be also expressed as validity of a quantified boolean formula in K; that is, φ is satisfied ψ-vacuously if and only if

$$K \models \forall x \cdot \varphi[\psi \leftarrow x],$$

and φ is falsified ψ-vacuously if and only if

$$K \models \forall x \cdot \neg\varphi[\psi \leftarrow x] \ .$$

Propositional vacuity is robust for propositional formulas: vacuity of a formula φ is not affected by trivial changes to the model (such as extending the model with new atomic propositions), nor by the fragment of the propositional logic used to express φ.

One may conjecture that Definition 3.2 describes robust vacuity for temporal logic as well. However, this is not the case. For example, consider again the formula

$$P_4 = AG((AXp) \vee (AX\neg p))$$

According to our intuition discussed as part of Weakness 1 earlier in this section, P_4 should not be satisfied p-vacuously. Yet, in any model,

$$\begin{array}{rcl} P_4[p \leftarrow \mathsf{true}] & = & AG((AX\mathsf{true}) \vee (AX\neg\mathsf{true})) \;\; = \;\; \mathsf{true}, \text{ and} \\ P_4[p \leftarrow \mathsf{false}] & = & AG((AX\mathsf{false}) \vee (AX\neg\mathsf{false})) \;\; = \;\; \mathsf{true}. \end{array}$$

Thus, by Definition 3.2, φ is p-vacuous.

3.2 Structure Vacuity

Proposition vacuity interprets a model as a mapping from *every* state of the model to boolean values true and false. This is a limitation when trying to extend this definition to temporal formulas: replacing a subformula only by the constants true and false is not sufficient for identifying whether the subformula is important. Following this observation, we extend the definition of vacuity to account for all subsets of the statespace S. The resulting definition, originally introduced in [Armoni et al. 2003] under the name *structure* vacuity, is given below.

Definition 3.3 *Structure Vacuity.* [Armoni et al. 2003] A temporal logic formula φ is *structure* ψ-vacuous in a model K if and only if either

$$\forall Y \subseteq S \cdot K \models \varphi[\psi \leftarrow Y]\,, \text{ or}$$

$$\forall Y \subseteq S \cdot K \models \neg\varphi[\psi \leftarrow Y]\,,$$

where S is the statespace of K.

Alternatively, structure vacuity can be expressed as validity of a quantified temporal logic formula under structure semantics; that is, φ is satisfied structure ψ-vacuously if and only if

$$K \models_s \forall x \cdot \varphi[\psi \leftarrow x]\,,$$

and φ is falsified structure ψ-vacuously if and only if

$$K \models_s \forall x \cdot \neg\varphi[\psi \leftarrow x]\,.$$

Definition 3.3 makes vacuity too dependent on a particular model of the system. This leads to undesired side-effects. For example, consider again the property $P_4 = AG((AXp) \vee (AX\neg p))$ and models $\mathcal{L}$ and $\mathcal{M}$ from Figure 1 and Figure 3, respectively. The two models are bisimilar and cannot be distinguished by any temporal logic formula. However, recall that according to Definition 3.3, P_4 is p-vacuous in $\mathcal{L}$, and yet it is not p-vacuous in $\mathcal{M}$. Thus, structure vacuity is not robust for temporal logic.

3.3 Bisimulation Vacuity

The example in Section 3.2 illustrates that it is not sufficient to define vacuity with respect to a single *particular* model K. Instead, a robust definition of vacuity must also take into account any model that is behaviorally equivalent to K. For temporal

logic, two models are considered to be behaviorally equivalent if and only if they are bisimilar. This leads to the following, robust, definition of vacuity.

Definition 3.4 *Bisimulation Vacuity.* A temporal logic formula φ is *bisimulation* ψ-vacuous in a Kripke structure K if and only if it is structure ψ-vacuous both in K and in every structure bisimilar to K. That is, either

$$\forall K' \in \mathcal{B}(K) \cdot \forall Y \subseteq S' \cdot K' \models \varphi[\psi \leftarrow Y]\,, \text{ or}$$

$$\forall K' \in \mathcal{B}(K) \cdot \forall Y \subseteq S' \cdot K' \models \neg\varphi[\psi \leftarrow Y]\,,$$

where S' denotes the statespace of K'.

Alternatively, structure vacuity can be expressed as validity of a quantified temporal logic formula under bisimulation semantics; that is, φ is satisfied bisimulation ψ-vacuously if and only if

$$K \models_b \forall x \cdot \varphi[\psi \leftarrow x]\,,$$

and φ is falsified bisimulation ψ-vacuously if and only if

$$K \models_b \forall x \cdot \neg\varphi[\psi \leftarrow x]\,.$$

That is, $\varphi[\psi \leftarrow x]$ is either satisfied or violated in every model that is x-bisimilar to K. For example, the property P_4 is not bisimulation vacuous in either $\mathcal{L}$ or $\mathcal{M}$.

In the next section, we describe some of the key properties of bisimulation vacuity and argue that it provides a uniform definition of robust vacuity for both linear and branching time logics.

3.4 Properties of Bisimulation Vacuity

For CTL*, bisimulation vacuity is more strict than either structure or syntactic vacuity, i.e., if a formula is vacuous w.r.t. bisimulation vacuity, then it is vacuous w.r.t. to structure and syntactic definitions of vacuity as well, but the converse is not true in general.

THEOREM 3.5. *Let K be a Kripke structure, φ be an* ACTL* *formula, and ψ be a subformula of φ. Then, if φ is bisimulation vacuous in ψ (in K) then (a) φ is structure vacuous in ψ, and (b) φ is syntactically vacuous in ψ w.r.t.* CTL*.

PROOF. Part (a) is a direct consequence of Theorem 2.13.

To prove part (b), we show that for CTL*, structure vacuity implies syntactic vacuity. Let $K = (AP, S, R, s_0, I)$ be a Kripke structure. By Definition 3.1, φ is syntactically vacuous in ψ iff for any CTL* formula ψ', $K \models \varphi$ iff $K \models \varphi[\psi \leftarrow \psi']$. Note that ψ' is a state formula. Let Y be the set of all states that satisfy ψ'. Formally, $Y = \{s \in S | K, s \models \psi'\}$. Then $K \models \varphi[\psi \leftarrow \psi']$ iff $K \models \varphi[\psi \leftarrow Y]$. Thus, for CTL*, structure vacuity is more strict than syntactic vacuity: if φ is structure vacuous in ψ, then φ is syntactically vacuous in ψ. □

In the rest of this section, we show that while bisimulation vacuity is *not* too strict, i.e., it does capture the "obvious" cases of vacuity, it is strict enough to be robust, i.e., it does not suffer from the three weaknesses identified in the beginning of this section.

Temporal logic tautologies are the most obvious examples of vacuous formulas. We show that they are vacuous under bisimulation vacuity.

PROPOSITION 3.6. *Let φ be a temporal logic formula with at least one atomic proposition, say p. If φ is either valid or unsatisfiable, then it is bisimulation p-vacuous in any model.*

PROOF. The theorem follows from the fact that validity is invariant under substitution of atomic propositions with fresh variables. That is, if $\varphi[p]$ is a valid formula with a proper subformula p, and x is an atomic proposition that does not occur in φ, then $\varphi[p \leftarrow x]$ is valid as well. □

For example, consider the property

$$P_6 = (EXp) \vee (AX\neg p).$$

Replacing p by x in P_6 yields

$$P_6[p \leftarrow x] = (EXx) \vee (AX\neg x),$$

which is a tautology. Hence, $\forall x \cdot P_6[p \leftarrow x]$ is satisfied by any model under any semantics of QCTL* from Section 2.4. Thus, property P_6 is bisimulation p-vacuous in any model.

Bisimulation vacuity is able to detect vacuity even if the formula itself is not a tautology, but contains a non-trivial tautology as a proper subformula. This follows from the proof of Proposition 3.6.

COROLLARY 3.7. *Let φ be a temporal logic formula, and ψ be a proper non-constant subformula of φ with an atomic proposition p. If ψ is either valid or unsatisfiable and φ does not contain p outside of ψ, then φ is p-bisimulation vacuous in any model.*

For example, consider the property

$$P_7 = AG(q \wedge ((EXp) \vee (AX\neg p))).$$

Since a tautology can always be replaced by a constant, P_7 is equivalent to

$$\begin{aligned} & AG(q \wedge ((EXp) \vee (AX\neg p))) \\ = \; & AG(q \wedge \mathsf{true}) \\ = \; & AG(q). \end{aligned}$$

Hence, P_7 does not depend on p and is p-vacuous in any model. Note that since bisimulation vacuity is stricter than either structure or syntactic vacuity, both Proposition 3.6 and Corollary 3.7 extend to structure and syntactic vacuity as well.

Bisimulation vacuity is strict enough to exclude vacuity that can be "fixed" by non-essential changes to the model. In particular, it can distinguish between two models only if temporal logic can distinguish between them as well. Thus, two models that agree on all temporal logic formulas, also agree on their bisimulation vacuity.

PROPOSITION 3.8. *Let φ be a temporal logic formula, ψ be a subformula of φ, and K and K' be two bisimilar Kripke structures. Then, φ is ψ-vacuous in K iff it is ψ-vacuous in K'.*

PROOF. The proof follows immediately from the definition of bisimulation vacuity. □

For example, the model $\mathcal{L}$ in Figure 1 and the model $\mathcal{M}$ in Figure 3 are bisimilar. Thus, they agree on satisfaction and vacuity of all temporal logic formulas. In particular, property P_4 (see Weakness 1) is not bisimulation p-vacuous in either model.

An important consequence of Proposition 3.8 is that bisimulation vacuity is not affected by parallel synchronous composition. That is, if a formula is vacuous with respect to a component, then it is vacuous with respect to the whole system as well.

COROLLARY 3.9. *Let φ be a temporal logic formula, ψ be a subformula of φ, and K and K' be two Kripke structures. If φ is bisimulation ψ-vacuous in K, then it is bisimulation ψ-vacuous in the parallel synchronous composition $K||K'$.*

PROOF. This follows from the fact that K and $K||K'$ are bisimilar with respect to atomic propositions of K. For $K = (AP, S, R, s_0, I)$ and $K' = (AP', S', R', s'_0, I')$, let $K||K' = (AP \cup AP', S \times S', R_{||}, (s_0, s'_0), I_{||})$ be their parallel synchronous composition (see Definition 2.2). Then, the relation

$$\rho \triangleq \{(s, (s, t)) \mid s \in S, t \in S'\}$$

is a bisimulation between K and $K||K'$. □

For example, consider again the example given in Weakness 2. The formula P_4 is not bisimulation vacuous in the model $\mathcal{L}$ (Figure 1), and its vacuity status does not change when $\mathcal{L}$ is composed with $\mathcal{N}$ (Figure 4); nor does its vacuity status change when $\mathcal{L}$ is composed with any other model that does not affect the satisfaction of P_4.

In summary, we argue that bisimulation vacuity is robust and does not suffer from the three weaknesses described in the beginning of this section. Bisimulation vacuity is stricter than syntactic vacuity – it considers less formulas to be vacuous (Weakness 1). It is invariant under bisimulation and cannot be affected by changes of the model that are "irrelevant" to a property being checked (Weakness 2). Finally, it is defined on the semantics of the temporal logic and, hence, is independent of the syntax (Weakness 3). At the same time, it agrees with syntactic vacuity (and other similar definitions) in all of the "obvious" cases of vacuity.

4. COMPLEXITY OF VACUITY DETECTION

In this section, we present algorithms for bisimulation vacuity detection and analyze their complexity. We show that in general, the complexity of bisimulation vacuity detection of a branching-time logic is the same as the complexity of the satisfiability problem for that logic. We then explore several practically important fragments of branching time logics. We show that the complexity of bisimulation vacuity detection for those fragments is in the same complexity class as model-checking. In the rest of the article, we use the terms "vacuity" or "robust vacuity" to mean "bisimulation vacuity", unless stated otherwise.

4.1 Complexity of Detecting Bisimulation Vacuity

We begin our study of complexity of detecting vacuity for branching time logics with an example. Let φ be a temporal logic formula over a single atomic proposition p. That is, while there might be several occurrences of p in φ, no other atomic proposition is allowed. Now consider the problem of detecting vacuity of φ with respect to model $\mathcal{L}$ from Figure 1. Note that every Kripke structure with a single atomic proposition x is p-bisimilar to some Kripke structure in $\mathcal{B}_x(\mathcal{L})$. Thus, φ is satisfied p-vacuously by $\mathcal{L}$ iff $\varphi[p \leftarrow x]$ is a tautology. Similarly, φ is falsified p-vacuously by $\mathcal{L}$ iff $\varphi[p \leftarrow x]$ is unsatisfiable. Thus, the problems of validity and satisfiability of φ are reduced to detecting vacuity of φ with respect to $\mathcal{L}$. We use this example as an intuition for formulating and proving the general complexity result:

THEOREM 4.1. *Deciding whether a formula φ is bisimulation ψ-vacuous is EXPTIME-complete for* CTL*, and 2EXPTIME-complete for* CTL**.*

PROOF. To proof completeness, we need to show (1) membership and (2) hardness To show membership, we reduce bisimulation vacuity to model-checking quantified temporal logic under tree semantics. To show hardness, we reduce temporal logic satisfiability to bisimulation vacuity.

Membership. Recall that detecting bisimulation vacuity is reducible to model-checking a quantified temporal logic formula under bisimulation semantics (see Section 3.3). Here, we reduce model-checking under bisimulation semantics to model-checking under tree semantics, which was shown by Kupferman in [Kupferman 1997] to be in EXPTIME for EQCTL and in 2EXPTIME for EQCTL*.

Formally, let $K = (AP, S, R, s_0, I)$ be a Kripke structure, and m be a natural number. We define a Kripke structure K^m to be the tuple

$$(AP, S \times [0, (m-1)], R^m, \langle s_0, 0\rangle, I^m) \ ,$$

where the transition relation and the labeling function are defined as follows:

$$(\langle s, i\rangle, \langle t, j\rangle) \in R^m \Leftrightarrow (s, t) \in R$$
$$I^m(\langle s, i\rangle) \triangleq I(s) \ .$$

Intuitively, K^m is the result of duplicating each successor of K m times.

Let $\exists x \cdot \varphi$ be an EQCTL formula. We show that K satisfies $\exists x \cdot \varphi$ under bisimulation semantics iff $K^{|\varphi|}$ satisfies $\exists x \cdot \varphi$ under tree semantics, i.e.,

$$K \models_b \exists x \cdot \varphi \Leftrightarrow K^{|\varphi|} \models_T \exists x \cdot \varphi \, .$$

The proof of the "if" direction is trivial since K^m is bisimilar to K for any m.

The proof of the "only if" direction uses the proof of the small model theorem for CTL (Theorem 6.14 in [Emerson 1990]). Assume that $K \models_b \exists x \cdot \varphi$. Then there exists a computation T such that (a) T is bisimilar to K, and (b) T satisfies φ with respect to structure semantics, i.e., $T \models_s \varphi$. By the proof of the small model theorem for CTL (see proof of Theorem 6.14 in [Emerson 1990]), there exists a subtree T' of T such that

(1) $T' \models_s \varphi$;
(2) T' is bisimilar to K;

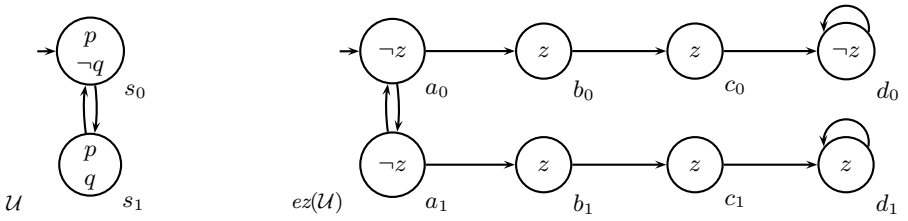


Fig. 5. A Kripke structure $\mathcal{U}$ with atomic propositions p and q, and its encoding $ez(\mathcal{U})$ using a single atomic proposition z.

(3) the branching degree of T' is bounded by $d_K + |\varphi|$, where d_K is the branching degree of K.

Let $T^{|\varphi|}$ be the computation tree of $K^{|\varphi|}$. Since $K^{|\varphi|}$ is bisimilar to K, by transitivity, $T^{|\varphi|}$ is bisimilar to T'. Furthermore, the branching degree of $T^{|\varphi|}$ is greater or equal to the branching degree of T'. Hence, T' is a subtree of $T^{|\varphi|}$. Therefore, $T^{|\varphi|} \models_s \varphi$ and $K^{|\varphi|} \models_b \exists x \cdot \varphi$.

The proof for QCTL* is based on the equivalent Small Model Theorem for CTL* (Theorem 3.2 in [Emerson and Sistla 1984]) and is otherwise identical to the one above.

Hardness for CTL. We have already shown that deciding satisfiability of a CTL formula with a single atomic proposition is reducible to detecting bisimulation vacuity. Now, we reduce satisfiability of CTL to satisfiability of a CTL formula restricted to a single atomic proposition. The idea is to encode the atomic propositions labeling each state by a structure attached to that state. For example, consider a model $\mathcal{U}$ with atomic propositions p and q and its encoding, $ez(\mathcal{U})$, that uses a single atomic proposition z (see Figure 5). States a_0 and a_1 of $ez(\mathcal{U})$ correspond to states s_0 and s_1 of $\mathcal{U}$, respectively. The structure rooted at b_0 encodes the labeling of the atomic propositions at s_0: state b_0 is labeled with z to indicate that it is the root of the encoding structure; state c_0 is labeled with z to indicate that s_0 is labeled with p, and d_0 is labeled with $\neg z$ to indicate that s_0 is labeled with $\neg q$. Similarly, the structure rooted at b_1 encodes the labeling of the atomic propositions at s_1.

Formally, let $K = (AP, S, R, s_0, I)$ be a Kripke structure, $n = |AP|$ denote the number of atomic propositions, and $o : AP \to [0, n]$ be some enumeration of atomic propositions. Then Kripke structure $ez(K)$ is the tuple

$$(\{z\}, S \times [0, (n+1)], R_{ez}, \langle s_0, 0\rangle, I_{ez}),$$

where z is a new atomic proposition not in AP, and the transition relation and the

labeling function are defined as follows:

$$(\langle s,i\rangle,\langle t,j\rangle)\in R_{ez}\Leftrightarrow\begin{cases}(s,t)\in R & \text{if } i=j=0\\ \mathsf{true} & \text{if } s=t\wedge(j=i+1\vee i=j=n+1)\\ \mathsf{false} & \text{otherwise}\end{cases}$$

$$I_{ez}(\langle s,i\rangle)\triangleq\begin{cases}\{\neg z\} & \text{if } i=0\\ \{z\} & \text{if } i=1\\ \{z\} & \text{if } \exists p\in AP\cdot o(p)=(i-1)\wedge p\in I(s)\\ \{\neg z\} & \text{otherwise}\end{cases}$$

Given a CTL formula ψ with atomic propositions in AP, we replace each atomic proposition with a temporal logic formula over a new atomic proposition z. For example, the formula $E[\mathsf{true}\,U\,\neg p\wedge A[\mathsf{false}\,\tilde{U}\,q]]$ is translated into

$$E[\neg z\,U\,(EXz)\wedge AX(z\Rightarrow AX\neg z)\wedge (EG\neg z)\wedge A[z\,\tilde{U}\,\neg z\Rightarrow((EXz)\wedge AX(z\Rightarrow AXAXz))]]\,.$$

The translation increases the size of ψ by a factor of $|\psi|$ due to the extra AX operators.

Formally, we define a function f such for any CTL formula ψ, the following conditions hold: (a) $f(\psi)$ only contains one atomic proposition, z, and (b) $f(\psi)$ and ψ are equisatisfiable. We define f by induction on the structure of ψ, showing just the "interesting" cases (f distributes over the operators in other cases).

$$\begin{aligned}f(p)&\triangleq(EXz)\wedge AX(z\Rightarrow AX^{o(p)+1}z)\\ f(\neg p)&\triangleq(EXz)\wedge AX(z\Rightarrow AX^{o(p)+1}\neg z)\\ f(EX\psi_1)&\triangleq EX(\neg z\wedge f(\psi_1))\\ f(AX\psi_1)&\triangleq EX(\neg z)\wedge AX(\neg z\Rightarrow f(\psi_1))\\ f(E[\psi_1\,U\,\psi_2])&\triangleq E[\neg z\wedge f(\psi_1)\,U\,\neg z\wedge f(\psi_2)]\\ f(A[\psi_1\,U\,\psi_2])&\triangleq EG(\neg z)\wedge A[\neg z\Rightarrow f(\psi_1)\,U\,\neg z\Rightarrow f(\psi_2)]\\ f(E[\psi_1\,\tilde{U}\,\psi_2])&\triangleq E[\neg z\wedge f(\psi_1)\,\tilde{U}\,\neg z\wedge f(\psi_2)]\\ f(A[\psi_1\,\tilde{U}\,\psi_2])&\triangleq EG(\neg z)\wedge A[\neg z\Rightarrow f(\psi_1)\,\tilde{U}\,\neg z\Rightarrow f(\psi_2)]\end{aligned}$$

Since model K satisfies a property ψ, $ez(K)$ satisfies $f(\psi)$.

For the other direction, let $M=(\{z\},S_M,R_M,s_0^M,I_M)$ be a model for $f(\psi)$. Let S_K be the smallest subset of S_M that satisfies the following two conditions:

$$\{s_0^M\}\in S_K,\ \text{and}$$
$$\forall s\in S_K\cdot\{t\in S_M\mid I_M(t)=\{\neg z\}\wedge(s,t)\in R_M\}\subseteq S_K.$$

That is, S_K includes the initial states and all states labeled with $\neg z$ that are reachable from the initial state by other states labeled with $\neg z$.

Let $K = (AP, S_K, R_K, s_0^M, I_K)$, where

$$(s,t) \in R_K \Leftrightarrow (s,t) \in R_M$$
$$\{p\} \in I_K(s) \Leftrightarrow M, s \models AX(z \Rightarrow AX^{o(p)+1} z)$$
$$\{\neg p\} \in I_K(s) \Leftrightarrow M, s \models AX(z \Rightarrow AX^{o(p)+1} \neg z).$$

Then K is a model for ψ: $K \models \psi$. Note that universal path quantifiers in the encoding of propositions (i.e., $f(p)$ and $f(\neg p)$ given above) ensure that the labeling I_K is consistent (i.e., no state is labeled with both p and $\neg p$). The existential path quantifiers in this encoding ensure that the transition relation of K is total.

Since CTL satisfiability has been shown to be EXPTIME-hard [Fischer and Ladner 1979], this gives us the desired result.

Hardness for CTL*. As in the proof of hardness for CTL, we reduce satisfiability of CTL* to satisfiability of a CTL* formula restricted to a single atomic proposition. For the models, we use the same encoding as for CTL.

To translate formulas, we define a function g such for any CTL* formula ψ, the following conditions hold: (a) $g(\psi)$ contains only one atomic proposition, z, and (b) $g(\psi)$ and ψ are equisatisfiable. We define g by induction on the structure of ψ, again showing just the "interesting" cases.

$$g(p) \triangleq (EXz) \wedge AX(z \Rightarrow X^{o(p)+1} z)$$
$$g(\neg p) \triangleq (EXz) \wedge AX(z \Rightarrow X^{o(p)+1} \neg z)$$
$$g(E\psi_1) \triangleq E((G\neg z) \wedge g(\psi_1))$$
$$g(A\psi_1) \triangleq (EG\neg z) \wedge A((G\neg z) \Rightarrow g(\psi_1))$$

The rest of the proof proceeds the same way as for CTL. This establishes hardness in 2EXPTIME since CTL* satisfiability has been shown to be 2EXPTIME-hard [Vardi and Stockmeyer 1985]. □

Theorem 4.1 suggests that bisimulation vacuity detection for CTL* and even for CTL is not computationally tractable. However, we show that there are several important fragments of CTL* for which vacuity detection is in the same complexity class as model-checking, and thus is tractable. We study these in the rest of this section, starting with monotone formulas and continuing with ACTL* and ECTL*.

4.2 Vacuity and Monotone Formulas

In this section, we study the problem of vacuity detection for monotone formulas. We make two contributions. First, we show that vacuity detection for monotone formulas is reducible to model-checking. Our algorithm is a natural extension of the vacuity detection algorithms of Beer et al. [Beer et al. 2001] and Kupferman and Vardi [Kupferman and Vardi 1999]. Second, we show that detecting whether a formula expressed in a given temporal logic is monotone is as hard as deciding the satisfiability problem for this logic. This means that simple monotonicity checks, such as restricting vacuity to a single occurrence as in [Beer et al. 2001], or relying on polarity of occurrences, as in [Armoni et al. 2003], can not be cheaply extended to the full temporal logic.

Definition 4.2 *Monotone Formula.* A formula φ is *monotonically increasing* in

1: **requires:** φ *is monotone in* ψ
2: **boolean** `isMonVacuous`(**Formula** φ, **Formula** ψ, **Model** K)
3: **return** $K \models \varphi[\psi \leftarrow \mathsf{true}] \Leftrightarrow K \models \varphi[\psi \leftarrow \mathsf{false}]$

Fig. 6. Vacuity detection algorithm for monotone formulas.

a subformula ψ if whenever $(x \Rightarrow y)$ is valid, so is $(\varphi[\psi \leftarrow x] \Rightarrow \varphi[\psi \leftarrow y])$. It is *monotonically decreasing* in ψ if whenever $(x \Rightarrow y)$ is valid, so is $(\varphi[\psi \leftarrow x] \Leftarrow \varphi[\psi \leftarrow y])$. We say that φ is *monotone* in ψ if it is either monotonically increasing or monotonically decreasing in ψ.

For example, the formula $AG(p \Rightarrow AFq)$ is monotonically decreasing in p and is monotonically increasing in q; the formula $AG(p \wedge \neg p)$ is monotone in p, and the formula $AG(p \Rightarrow AF(p \wedge q))$ is not monotone in p.

The algorithm for detecting vacuity with respect to monotone subformulas, called `isMonVacuous`, is given in Figure 6. Detecting vacuity of φ with respect to a monotone subformula ψ can be reduced to comparing the results of two model-checking problems: the one in which ψ is replaced with true, and another in which ψ is replaced with false. The algorithm is based on the following intuition. For a fixed model K, $\varphi[\psi]$ can be seen as a monotone function from temporal logic to $\{\mathsf{true}, \mathsf{false}\}$ defined as follows: $\lambda x \cdot K \models \varphi[\psi \leftarrow x]$. The formula φ is vacuous in ψ if the above function is a constant (i.e., always true or always false). Since the function is monotone, it is a constant if and only if it assigns the same value to the extreme points: true and false. The correctness of the algorithm is established by the following theorem.

THEOREM 4.3. *Let φ be a temporal logic formula monotone in a subformula ψ, and K be a Kripke structure. Then* `isMonVacuous`(φ, ψ, K) *returns* true *if and only if φ is bisimulation vacuous in ψ.*

PROOF. We first establish the $(\Leftarrow)$ direction. Assume φ is bisimulation vacuous in ψ, and without loss of generality, assume that φ is satisfied by K. From Definition 3.4, it follows that φ holds under any interpretation of ψ, i.e., $K \models_b \forall x \cdot \varphi[\psi \leftarrow x]$. Finally, by specialization, $K \models \varphi[\psi \leftarrow \mathsf{true}] \wedge K \models \varphi[\psi \leftarrow \mathsf{false}]$.

For the $(\Rightarrow)$ direction, we use the fact that bisimilar structures satisfy the same temporal properties. Formally, for a formula φ with a subformula ψ and a Kripke structure K,

$$\forall K' \in \mathcal{B}(K) \cdot \forall c \in \mathbf{2} \cdot (K \models \varphi[\psi \leftarrow c]) \Leftrightarrow (K' \models \varphi[\psi \leftarrow c]) \qquad \text{(constant subst)}$$

Furthermore, without loss of generality, we assume that φ is satisfied by K, i.e.,

$$(K \models \varphi[\psi \leftarrow \mathsf{true}]) \wedge (K \models \varphi[\psi \leftarrow \mathsf{false}])\,.$$

The proof proceeds as follows:

$$\begin{array}{ll}
& (K \models \varphi[\psi \leftarrow \mathsf{true}]) \wedge (K \models \varphi[\psi \leftarrow \mathsf{false}]) & \text{(by constant subst)} \\
\Rightarrow & \forall K' \in \mathcal{B}(K) \cdot (K' \models \varphi[\psi \leftarrow \mathsf{true}]) \wedge (K' \models \varphi[\psi \leftarrow \mathsf{false}]) & \text{(by monotonicity)} \\
= & \forall K' \in \mathcal{B}(K) \cdot \forall Y \subseteq S' \cdot K' \models \varphi[\psi \leftarrow Y] & \text{(by Definition 2.12)} \\
= & K \models_b \forall x \cdot \varphi[\psi \leftarrow x]
\end{array}$$

Hence, by the discussion following Definition 3.4, φ is bisimulation vacuous in ψ. □

From the algorithm `isMonVacuous` and the proof of its correctness, we see that the complexity of detecting vacuity of monotone formulas is the same as that for model-checking:

COROLLARY 4.4. *Deciding whether a temporal logic formula φ is vacuous in a monotone subformula ψ is the same complexity as that of model-checking φ.*

Note that by itself, the algorithm `isMonVacuous` is incomplete since it requires a user to identify monotonicity of a subformula. However, in combination with a technique to decide whether a subformula is monotone, the algorithm leads to a practical and efficient vacuity detection technique.

There are several simple syntactic checks to identify monotone subformulas. For example, if ψ has only a single occurrence in φ, then φ is monotone in ψ, e.g., $AG(p \vee q \vee r)$ is monotone in q. Similarly, if ψ is pure in φ (i.e., all occurrences are either positive, like p above, or negative), then φ is monotone in ψ.

These simple syntactic checks have already been used in the early work on vacuity detection by Beer et al. [Beer et al. 2001] and by Kupferman and Vardi [Kupferman and Vardi 2003]. The algorithms presented in these papers are equivalent to the algorithm `isMonVacuous`, but only apply to formulas whose monotonicity can be detected syntactically. We thus conclude the following:

THEOREM 4.5. *All three types of vacuity – syntactic, structure, and bisimulation – coincide for monotone formulas.*

In particular, formulas with a single occurrence of a subformula of interest, or formulas with pure polarity are (syntactically) monotone. Thus, by Theorem 4.5, the three definitions of vacuity coincide for such formulas and so do the algorithms `isMonVacuous` and those reported in [Beer et al. 2001] and [Kupferman and Vardi 2003].

It is also interesting to see whether the scope of these simple syntactic checks for monotonicity can be significantly extended. We show that this is not possible in general due to the *EXPTIME*-hardness of this problem.

THEOREM 4.6. *Deciding whether a formula φ is monotone in a subformula ψ is EXPTIME-hard for* CTL*, and 2EXPTIME-hard for* CTL**.*

PROOF. We reduce the validity problem for CTL, known to be EXPTIME-hard [Fischer and Ladner 1979], to deciding monotonicity. Let φ be an arbitrary CTL formula, and p be an atomic proposition not occurring in φ. Then the formula $\psi = (p \Rightarrow AXp) \vee \varphi$ is monotone in p iff φ is valid. In general, ψ is not monotone in p. However, if φ is valid, then ψ is valid as well; hence, it is monotone in all of its atomic propositions.

The proof for CTL* is identical. Note that the validity problem for CTL* is known to be 2EXPTIME-hard [Vardi and Stockmeyer 1985]. □

Thus, identifying whether a given formula is monotone is as difficult as vacuity detection in general. It is unlikely that the applicability of the algorithm `isMonVacuous` can be generalized past syntactically monotone formulas.

In this section, we have studied vacuity detection for monotone formulas and gave an efficient algorithm for it. For such formulas, bisimulation vacuity coincides with

syntactic vacuity. While our algorithm applies to arbitrary monotone formulas, we have shown that determining whether a given property is monotone is as hard as the general vacuity detection. However, for syntactically monotone formulas, such as those with a single occurrence of a subformula of interest, or formulas with pure polarity, our algorithm becomes identical to [Beer et al. 2001; Kupferman and Vardi 2003].

4.3 Deciding Vacuous Satisfaction of ACTL* Formulas

In this section, we present an algorithm for detecting whether an ACTL* formula is satisfied vacuously. Specifically, given an ACTL* formula φ, a Kripke structure K which is known to satisfy φ, and a subformula ψ of φ, our goal is to decide whether φ is bisimulation vacuous in ψ. We show that this problem is in the same complexity class as model-checking. This is significant in practice since properties are often expressed in ACTL* or in its linear fragment, LTL. By duality, the results of this section extend to deciding vacuous falsification of ECTL* formulas.

Recall that deciding whether φ is satisfied vacuously is equivalent to model-checking $\forall x \cdot \varphi[\psi \leftarrow x]$ in K under bisimulation semantics. This, in turn, is equivalent to checking that $\varphi[\psi \leftarrow x]$ is satisfied in every model that is x-bisimilar to K.

Our algorithm for detecting vacuous satisfaction of ACTL* formulas is based on the idea that for ACTL* formulas, vacuity detection can be reduced to a single model-checking instance. The algorithm, called `isSATVacuous`, is shown in Figure 8(a). In the rest of this section, we first illustrate the algorithm on an example, and then formally establish its correctness and complexity.

As an example, we consider the problem of detecting whether an ACTL* formula is satisfied vacuously in a model $\mathcal{P}$ given in Figure 4. We show that this problem is reducible to a single model-checking problem with respect to a model $\mathcal{Q}$ given in Figure 7. The model $\mathcal{Q}$ is obtained from $\mathcal{P}$ by the following steps: (a) adding a new atomic proposition x; (b) splitting each state of $\mathcal{P}$ into two states, one interpreting x as true and another interpreting x as false; and (c) adding a transition between any two states if there is a transition between the corresponding states of $\mathcal{P}$. For example, states d_0 and d_2 of $\mathcal{Q}$ correspond to splitting state c_0 of $\mathcal{P}$; the transition between d_2 and d_1 in $\mathcal{Q}$ corresponds to the transition between c_0 and c_1 in $\mathcal{P}$; and there is no transition between d_0 and d_2 in $\mathcal{Q}$ since there is no corresponding self-loop on c_0 in $\mathcal{P}$.

It is easy to see that $\mathcal{Q}$ is x-bisimilar to $\mathcal{P}$: $\mathcal{Q}$ differs from $\mathcal{P}$ only in its interpretation of the new variable x, but otherwise has all of the same behaviors. Furthermore, $\mathcal{Q}$ does not enforce any temporal constraints on x – from any state, x can evolve to either true or false. Thus, $\mathcal{Q}$ can simulate (i.e., match the behavior of) any Kripke structure that is x-bisimilar to $\mathcal{P}$. For example, d_0 can simulate any state that is x-bisimilar to c_0, and d_2 can simulate any state that is x-bisimilar to c_1. Recall that simulation preserves satisfaction of ACTL* formulas (Theorem 2.7). Thus, since $\mathcal{Q}$ simulates every structure that is x-bisimilar to $\mathcal{P}$, it satisfies an ACTL* formula *if and only if* the formula is satisfied by every structure x-bisimilar to $\mathcal{P}$. This reduces model-checking a formula $\varphi[\psi \leftarrow x]$ on *all* structures that are x-bisimilar to $\mathcal{P}$ to a *single* model-checking problem on $\mathcal{Q}$! Hence, checking whether φ is ψ-vacuous on $\mathcal{P}$ is equivalent to model-checking $\varphi[\psi \leftarrow x]$ on $\mathcal{Q}$.

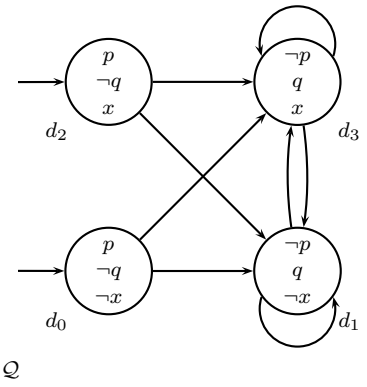
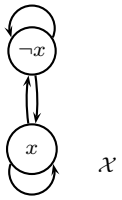


Fig. 7. A model $\mathcal{Q}$ used in the reduction of vacuity detection for the model $\mathcal{P}$ from Figure 4 to model-checking.

1: **requires:** φ *is an* ACTL* *formula satisfied by* K
2: **boolean** `isSATVacuous`(**Formula** φ, **Formula** ψ, **Model** K)
3: $K' = (K \,||\, \mathcal{X})$
4: **return** $K' \models \varphi[\psi \leftarrow x]$

(a)



(b)

Fig. 8. (a) An algorithm for detecting vacuous satisfaction of ACTL* formulas, and (b) Kripke structure $\mathcal{X}$ used by the algorithm.

While $\mathcal{Q}$ has twice as many states as $\mathcal{P}$, both structures share the same symbolic representation of the transition relation, represented by the formula

$$(p \wedge \neg q \wedge \neg p' \wedge q') \vee (\neg p \wedge q \wedge \neg p' \wedge q').$$

This means that for a symbolic model-checking algorithm, checking $\mathcal{Q}$ and a seemingly smaller model $\mathcal{P}$ is equally easy (or equally hard).

We now return to the algorithm `isSATVacuous`. This algorithm uses a Kripke structure $\mathcal{X}$ shown in Figure 8(b) and defined as $\mathcal{X} \triangleq (AP^{\mathcal{X}}, S^{\mathcal{X}}, S_0^{\mathcal{X}}, R^{\mathcal{X}}, I^{\mathcal{X}})$, with a single atomic proposition x ($AP^{\mathcal{X}} = \{x\}$), two states ($S^{\mathcal{X}} = \{0, 1\}$), all states being initial ($S_0^{\mathcal{X}} = S^{\mathcal{X}}$), any transition being allowed ($R^{\mathcal{X}} = S^{\mathcal{X}} \times S^{\mathcal{X}}$), and x being interpreted as $I^{\mathcal{X}}(0, x) = \mathsf{false}$ and $I^{\mathcal{X}}(1, x) = \mathsf{true}$.

The correctness of the algorithm is based on the observation that for any Kripke structure K, the parallel synchronous composition $K \,||\, \mathcal{X}$ of K and $\mathcal{X}$ (assuming that x is a fresh variable for K) simulates any structure K' that is x-bisimilar to K.

THEOREM 4.7. *Let* $K = (AP, S, R, S_0, I)$ *be an arbitrary Kripke structure, and* $K' = (AP \cup \{x\}, S', R', S'_0, I')$ *be* $\{x\}$*-bisimilar to* K*. Then* K' *is simulated by* $K \,||\, \mathcal{X}$*.*

PROOF. By Definition 2.2, the Kripke structure $K \,||\, \mathcal{X}$ is

$$(AP \cup \{x\}, S \times \{0,1\}, S_0 \times \{0,1\}, R^x, I^x),$$

where $R^x(\langle s,i\rangle, \langle t,j\rangle) \Leftrightarrow R(s,t)$, and

$$I^x(\langle s,i\rangle, p) = \begin{cases} I(s,p) & \text{if } p \neq x \\ I^{\mathcal{X}}(i,x) & \text{if } p = x. \end{cases}$$

Let $\rho \subseteq S \times S'$ be the $\{x\}$-bisimulation relation between K and K'. We claim that $K \,||\, \mathcal{X}$ simulates K' via the relation

$$\rho^x = \{(\langle s,i\rangle, t) \mid \rho(s,t) \wedge I^x(\langle s,i\rangle, x) = I'(t,x)\}.$$

From the definition of ρ^x, it follows immediately that $\rho^x(\langle s,i\rangle, t) \Leftrightarrow (I^x(\langle s,i\rangle) = I(t))$, thus, it satisfies the first condition of simulation. The proof of the second condition is given below:

$$\begin{array}{ll} & \rho^x(\langle s,i\rangle, t) \wedge R'(t,t') \\ \Rightarrow & \text{(since } K' \text{ is } \{x\}\text{-bisimilar to } K) \\ & \exists s' \in S \cdot \rho(s',t') \wedge R(s,s') \\ \Rightarrow & \text{(by definition of } K \,||\, \mathcal{X}) \\ & \exists s' \in S \cdot \forall j \in \{0,1\} \cdot R^x(\langle s,i\rangle, \langle s',j\rangle) \wedge \rho(s',t') \\ \Rightarrow & \text{(since } I'(t',x) \in \mathbf{2}) \\ & \exists s' \in S \cdot \exists j \in \{0,1\} \cdot R^x(\langle s,i\rangle, \langle s',j\rangle) \wedge \rho(s',t') \wedge I^x(\langle s',j\rangle) = I'(t') \\ \Rightarrow & \text{(by definition of } \rho^x) \\ & \exists s' \in S \cdot \exists j \in \{0,1\} \cdot R^x(\langle s,i\rangle, \langle s',j\rangle) \wedge \rho^x(\langle s',j\rangle, t') \end{array}$$

Finally, if t is an initial state of K', then there exists an $i \in \{0,1\}$ such that $\rho^x(\langle s,i\rangle, t)$ holds, which establishes that $K \,||\, \mathcal{X}$ simulates K' via ρ^x. □

Since simulation preserves ACTL*, vacuity detection for an arbitrary ACTL* formula is reducible to model-checking over $K \,||\, \mathcal{X}$. This proves correctness of `isSATVacuous`.

PROPOSITION 4.8. *Let φ be an* ACTL* *formula with a subformula ψ, K be a Kripke structure, and assume that K satisfies φ. Then* `isSATVacuous`(φ, ψ, K) *returns* `true` *if and only if φ is bisimulation vacuous in ψ.*

PROOF. Let φ be a formula satisfied by K. We show that φ is ψ-vacuous iff the formula $\varphi[\psi \leftarrow x]$ is satisfied by $K \,||\, \mathcal{X}$.

Since $K \,||\, \mathcal{X}$ is $\{x\}$-bisimilar to K, the proof of ($\Rightarrow$) direction is trivial.

For ($\Leftarrow$) direction, if $\varphi[\psi \leftarrow x]$ holds in $K \,||\, \mathcal{X}$, then by Theorem 4.7 and Theorem 2.9 it holds in every $\{x\}$-bisimulation of K. □

An immediate consequence of Proposition 4.8 is that for the LTL fragment of ACTL*, our bisimulation vacuity is equivalent to trace vacuity of Armoni et al [Armoni et al. 2003]. That is, for any fixed model K, an LTL formula φ is trace vacuous in ψ if and only if it is bisimulation vacuous in ψ. We further elaborate on this connection between the two definitions in Section 6.

From the algorithm `isSATVacuous` and the proof of its correctness, it is easy to see that the complexity of detecting vacuous satisfaction of ACTL* formulas is in the same complexity class as model-checking:

COROLLARY 4.9. *Let φ be an* ACTL* *formula, ψ be a subformula of φ, and K be a Kripke structure. Deciding whether K satisfies φ ψ-vacuously is in the same complexity class as model-checking φ.*

As mentioned earlier, while the explicit statespace of $K \,||\, \mathcal{X}$ is twice of that of K, $K \,||\, \mathcal{X}$ does not impose any restrictions on the atomic proposition x; therefore, the symbolic representation of its transition relation is identical to that of K.

In this section, we described an algorithm, `isSATVacuous`, for detecting whether an ACTL* formula is satisfied vacuously. We proved correctness of this algorithm and showed that checking whether an ACTL* formula φ is vacuous in some subformula is no more expensive than model-checking φ.

4.4 Deciding Vacuous Satisfaction of CTL* in Universal Subformulas

In the rest of this section, we show that the algorithm `isSATVacuous` can be used not only for detecting vacuous satisfaction of ACTL* formulas but also for detecting vacuous satisfaction of CTL* formulas with respect to universal subformulas. That is, under the assumption that a subformula ψ occurs only under universal path quantifiers in the negation normal form of φ, `isSATVacuous`(φ, ψ, K) returns `true` if and only if φ is satisfied vacuously in ψ.

Given a fixed model K, the structure of a temporal formula φ can be simplified by replacing state subformulas with propositional expressions without affecting the satisfiability of φ. For example, consider the model $\mathcal{O}$ in Figure 4(b) and the property $AFEGp$. For this model, formula EGp can be simplified to p, and formula $AFEGp$ — to AFp. We use the notation $Prop(\varphi, K)$ to denote *some* such propositional simplification of φ with respect to a model K. Formally, $Prop(\varphi, K)$ is a formula obtained by replacing some state subformula ψ of φ with a propositional encoding of the set $||\psi||^K$ of all the states of K that satisfy ψ.

Propositional simplification does not affect satisfaction. That is, a structure K satisfies φ if and only if it satisfies a propositional simplification of φ:

$$K \models \varphi \Leftrightarrow K \models Prop(\varphi, K) \qquad \text{(propositional simplification)}$$

Moreover, this property is preserved by bisimulation – if K and K' are bisimilar, then φ can be simplified with respect to either model without affecting its satisfaction on both models. That is, K' satisfies φ if and only if it satisfies any propositional simplification of φ with respect to a bisimilar model K.

THEOREM 4.10. *Let K and K' be two structures such that K is x-bisimilar to K' via a relation ρ, and let φ be a* CTL* *formula not containing x. Then K' satisfies φ iff it satisfies a propositional simplification of φ with respect to K:*

$$K' \models \varphi \Leftrightarrow K' \models Prop(\varphi, K)\,.$$

PROOF. Let S and S' denote the statespaces and s_0 and s'_0 denote the initial

states of K and K', respectively. Then

$$\begin{array}{ll} & K' \models \varphi & \text{(definition of } \models) \\ = & K', s'_0 \models \varphi & \text{(property of } \rho) \\ = & \exists s \in S \cdot K, s \models \varphi \wedge (s, s'_0) \in \rho & \text{(propositional simplification)} \\ = & \exists s \in S \cdot K, s \models Prop(\varphi, K) \wedge (s, s'_0) \in \rho & \text{(property of } \rho) \\ = & K', s'_0 \models Prop(\varphi, K) & \text{(definition of } \models) \\ = & K' \models Prop(\varphi, K) & \end{array}$$

□

We now use Theorem 4.10 to establish the main theorem of this section. We show that for any fixed model, a CTL* formula with a universal subformula ψ can be turned into an ACTL* formula without affecting the ψ-vacuity of the formula.

THEOREM 4.11. *Let φ be a* CTL* *formula with a universal subformula ψ, and K be a Kripke structure. Assume that K satisfies φ. Then* `isSATVacuous`(φ, ψ, K) *returns* `true` *if and only if φ is bisimulation vacuous in ψ.*

PROOF. Let $\mathcal{X}$ be a Kripke structure as depicted in Figure 8(b). We show that for a Kripke structure K, $\varphi[\psi \leftarrow x]$ is satisfied by $K||\mathcal{X}$ iff φ is vacuous in ψ.

The "if" direction is trivial.

For the "only if" direction, assume that $K||\mathcal{X}$ satisfies $\varphi[\psi \leftarrow x]$. Let $Prop(\varphi[\psi \leftarrow x], K)$ be the result of replacing all existential state subformulas of $\varphi[\psi \leftarrow x]$ with their propositional simplification in K. Since ψ occurs only in the scope of universal quantifiers, these subformulas do not contain x and can be interpreted on K. By Theorem 4.10, $K||\mathcal{X}$ satisfies $Prop(\varphi[\psi \leftarrow x], K)$. Since ψ is universal, $Prop(\varphi[\psi \leftarrow x], K)$ is in ACTL*. Applying Theorem 4.7 and then Theorem 2.7, we get that every Kripke structure K' that is x-bisimilar to K satisfies $Prop(\varphi[\psi \leftarrow x], K)$. By Theorem 4.10, K' satisfies $\varphi[\psi \leftarrow x]$ as well. Hence, by Definition 3.4, φ is bisimulation vacuous in ψ. □

Theorem 4.11 implies that detecting whether an arbitrary CTL* formula is satisfied vacuously in a universal subformula is in the same complexity class as model-checking:

COROLLARY 4.12. *Let φ be a* CTL* *formula, ψ be a universal subformula of φ, and K be a Kripke structure. Deciding whether K satisfies φ ψ-vacuously is in the same complexity class as model-checking φ.*

In this section, we have shown that the algorithm `isSATVacuous` is applicable not only to ACTL* formulas, but also to detecting vacuous satisfaction of arbitrary CTL* formulas in universal subformulas. We have also shown that vacuity detection for this more general case remains in the same complexity class as model-checking.

5. VACUITY AND ABSTRACTION

The statespace explosion problem, i.e., the fact that the size of a model doubles with an addition of each new atomic proposition, is one of the major challenges in practical applications of model checking. Abstraction is the most popular and most effective technique to combat this problem. In this section, we explore the interactions between vacuity detection and abstraction.

5.1 Abstraction

The key principle of abstraction is to replace model checking of a given property φ on a *concrete* model K_c with model checking of the property on an *abstract* model K_α. The abstract model K_α is typically chosen such that it is smaller and/or easier to represent symbolically than K_c.

Here, we consider the two most commonly used abstractions. In a *bisimulation-based abstraction*, the abstract model K_α is required to be bisimilar to the concrete model K_c. Cone of influence [Clarke et al. 1999] and symmetry reduction [Clarke et al. 1998; Wei et al. 2005] are two prominent examples of bisimulation-based abstraction. This abstraction is *sound and complete* for CTL*. That is, if a given property is satisfied or refuted by the abstract model, then it is, respectively, satisfied or refuted by the concrete one as well.

In a *simulation-based abstraction*, the abstract model K_α is required to simulate the concrete model K_c. This is the most commonly used abstraction technique for hardware and software model checking, e.g., [Graf and Saidi 1997; Ball et al. 2001]. Simulation-based abstraction is *sound (but incomplete)* for ACTL*. That is, the abstract model over-approximates the behaviors of the concrete one. Thus, if an ACTL* property is satisfied by K_α, it is satisfied by K_c, but the converse is not true in general.

5.2 Vacuity Detection in the Presence of Abstraction

In this section, we explore the preservation of vacuity for bisimulation- and simulation-based abstractions. Clearly, bisimulation-based abstraction is sound and complete for vacuity of CTL*.

PROPOSITION 5.1. *Let K_α and K_c be Kripke structures such that K_α is a bisimulation-based abstraction of K_c, and let φ be a* CTL* *formula with a subformula ψ. Then, φ is ψ-vacuous in K_a iff ψ-vacuous in K_c.*

PROOF. By definition of bisimulation-based abstraction, K_α and K_c are bisimilar. By Proposition 3.8, bisimulation vacuity is invariant under bisimulation. □

Note that bisimulation-based abstraction is *not* sound with respect to alternative definitions of vacuity! An example in Section 3 (Weakness 2) shows that the abstraction is not sound with respect to syntactic vacuity: the model $\mathcal{O}$ can be viewed as an abstraction of a concrete model $\mathcal{L}$. Then, property P_4 is vacuous in the concrete model, but is non-vacuous in the abstract, i.e., abstraction has masked vacuity. An example in Section 3.2 illustrates a similar situation for structure vacuity: the model $\mathcal{L}$ can be viewed as concrete and the model $\mathcal{M}$ as abstract. Property P_4 is vacuous in the concrete model and non-vacuous in the abstract.

We now turn our attention to simulation-based abstraction. Recall that this abstraction is only sound for ACTL* and thus we can only expect it to be sound for vacuity of ACTL* properties. Furthermore, this abstraction is not complete and thus we do not expect it to be complete for vacuity either. We show that below.

THEOREM 5.2. *Let K_α and K_c be two Kripke structures such that K_α simulates K_c, and let φ be an* ACTL* *formula with a subformula ψ. Then, whenever φ is ψ-vacuous in K_a, it is ψ-vacuous in K_c.*

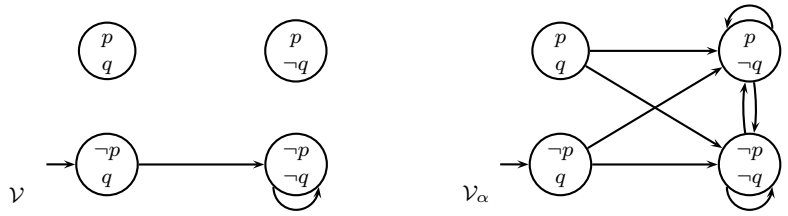


Fig. 9. A concrete Kripke structure $\mathcal{V}$ and its existential abstraction $\mathcal{V}_\alpha$.

PROOF. By Proposition 4.8, φ is ψ-vacuous in K_α iff $(K_\alpha \,||\, \chi) \models \varphi[\psi \leftarrow x]$, where χ is the Kripke structure shown in Figure 8(b). K_α simulates K_c and thus $K_\alpha \,||\, \chi$ simulates $K_c \,||\, \chi$ as well. Since simulation preserves ACTL*, $K_c \,||\, \chi \models \varphi[\psi \leftarrow x]$. By Proposition 4.8, φ is ψ-vacuous in K_c. □

Soundness of simulation-based abstraction with respect to vacuity of ACTL* is a trivial corollary of Theorem 5.2:

COROLLARY 5.3. *Let K_α and K_c be Kripke structures such that K_α is a simulation-based abstraction of K_c, and let φ be a* ACTL* *formula with a subformula ψ. Then, whenever φ is ψ-vacuous in K_a, it is ψ-vacuous in K_c.*

The converse of Theorem 5.2 is not true. As a counterexample, consider two structures, $\mathcal{V}$ and $\mathcal{V}_\alpha$, shown in Figure 8 and a property

$$P_8 \triangleq AG(p \Rightarrow AXq) \ .$$

While P_8 is satisfied vacuously in $\mathcal{V}$, it is non-vacuous in $\mathcal{V}_\alpha$. Thus, vacuity of a formula might be "hidden" by abstraction.

Note that simulation-based vacuity is *not* sound with respect to syntactic and structural definitions of vacuity. Same examples as used for bisimulation vacuity above apply here as well since the property P_4 is in ACTL*.

In summary, we showed that bisimulation vacuity interacts well with two most common abstraction techniques. Bisimulation-based abstraction is sound and complete for CTL* and is also sound and complete for vacuity. On the other hand, simulation-based abstraction is sound (but incomplete) for ACTL* and is only sound (but incomplete) for bisimulation vacuity. Moreover, neither of the abstractions is sound with respect to syntactic or structure vacuity.

Combining vacuity and abstractions other than simulation-based and bisimulation-based (such as the mixed-simulation-based abstraction of Dams et al. [Dams et al. 1997] which we studied in conjunction with vacuity in [Gurfinkel and Chechik 2004b]) would require similar reasoning as described in this section but is beyond the scope of this paper.

6. RELATED WORK

In this section, we survey related work. We begin by a general overview of vacuity research that is based on the (modifications of) the original syntactic vacuity of Beer et al. [Beer et al. 1997]. We then give an in-depth comparison between bisimulation vacuity and trace vacuity of Armoni et al. [Armoni et al. 2003]. We conclude this section by a discussion of other sanity checks to complement model-checking.

Syntactic Vacuity. The majority of the work on vacuity is based on the definition of syntactic vacuity (see Definition 3.1) of Beer et al. [Beer et al. 1997]. This definition and the corresponding vacuity detection algorithm have been extended and adapted to a variety of property languages: to CTL* in [Kupferman and Vardi 1999], to the modal μ-calculus in [Dong et al. 2002], to temporal logic with regular expressions in [Bustan et al. 2005], and to the logic of symbolic trajectory evaluation in [Tzoref and Grumberg 2006].

The notion of syntactic vacuity has been extended in a variety of ways. Gurfinkel and Chechik [Gurfinkel and Chechik 2004b] and Chockler and Strichman [Chockler and Strichman 2007; 2009] have studied *mutual vacuity* that considers vacuity in several subformulas simultaneously. Dong et al. [Dong et al. 2002] and independently Samer and Veith [Samer and Veith 2004] have explored a notion of vacuity in which a weaker formula (such as AFp) can be replaced by a stronger one (such as AXp). Chockler and Strichman [Chockler and Strichman 2007; 2009] have also explored vacuity between multiple properties, independently of a model.

Several modifications to the naïve vacuity detection algorithms of Beer et al. [Beer et al. 1997] and Kupferman and Vardi [Kupferman and Vardi 1999] have been proposed. Purandare and Somenzi [Purandare and Somenzi 2002] use the parse tree of temporal formula to enable information sharing between vacuity detecting passes of a symbolic model-checker. Gurfinkel and Chechik [Gurfinkel and Chechik 2004b] give an algorithm, based on multi-valued model-checking, that detects all instances of vacuity of a formula in a single pass. Simmonds et al. [Simmonds et al. 2010] use resolution proofs to speed up vacuity detection for bounded SAT-based model-checking.

Semantic Vacuity. We were inspired by the work of Armoni et al. [Armoni et al. 2003]. In [Armoni et al. 2003], the authors show many anomalies of the syntactic approach to vacuity, and informally argue for a set of robustness criteria. They present a semantic definition of vacuity for LTL, called *trace vacuity*, and develop an algorithm for detecting it. In this article, we build on this work by formalizing the criteria for robust vacuity using bisimulation, and by extending semantic vacuity to branching-time logic.

In what follows, we give an in-depth comparison between bisimulation vacuity that is introduced in this article and trace vacuity of Armoni et al. We give a formal definition of trace vacuity and its trivial extension to CTL* and show that this extension is not robust. We then show that bisimulation vacuity is a proper extension of trace vacuity by proving that they coincide for LTL properties. Finally, we compare the algorithms for detecting trace vacuity for LTL and bisimulation vacuity for ACTL*.

Originally, trace vacuity was defined using tree semantics of QTL, making it directly applicable to CTL*. A formal definition is given below:

Definition 6.1. [Armoni et al. 2003] A temporal logic formula φ is trace ψ-vacuous in a Kripke structure K if and only if $K \models_T \forall x \cdot \varphi[\psi \leftarrow x]$.

However, the following example illustrates that trace vacuity is not robust for branching temporal logic. Consider the property $(AXp \vee AX\neg p)$. It is trace p-vacuous in the model $\mathcal{L}$ in Figure 1 and not trace p-vacuous in the model $\mathcal{M}$

in Figure 3. Recall that these two models are bisimilar and thus should behave identically with respect to vacuity. Thus, trace vacuity is not robust: when applied to branching time properties, it becomes sensitive to irrelevant changes to the model (i.e., it suffers from "Weakness 2" as described in Section 3).

Bisimulation vacuity is a proper extension of trace vacuity: i.e., trace and bisimulation vacuity coincide for LTL. Formally, if an LTL formula φ is trace vacuous with respect to a structure K, then φ is bisimulation vacuous w.r.t. K as well, and vice versa.

THEOREM 6.2. *Let ψ be a path formula (i.e., expressed in LTL), and x be an atomic proposition occurring in ψ. Then, tree and bisimulation semantics of quantified temporal formula $\forall x \cdot A\psi$ are equivalent. Formally, for any model K,*

$$K \models_T \forall x \cdot A\psi \Leftrightarrow K \models_b \forall x \cdot A\psi \,.$$

PROOF. The "$\Leftarrow$" direction follows from Theorem 2.13. We prove the "$\Rightarrow$" direction by contradiction. Assume that $K \models_T \forall x \cdot A\psi$, and $K \not\models_b \forall x \cdot A\psi$. By the assumption, any trace that is an x-variant of a trace of K satisfies the path formula ψ. Furthermore, there exists a structure K' x-bisimilar to K with a trace $\pi \in K'$ such that π violates the path formula ψ, i.e., $\pi \not\models \psi$. However, π also belongs to some x-variant of a tree unrolling $T(K)$ of K. Hence, $\pi \models \psi$, which contradicts the assumption. □

`isSATVacuous`, the algorithm for detecting vacuous satisfaction of ACTL* formulas presented in this article, is very similar to the one suggested by Armoni et al. for detecting trace vacuity for LTL. The main difference is that our algorithm is based on changing the model and does not impose any restrictions on the model-checking procedure to be used. In contrast, the algorithm of Armoni et al. is based on changing the automaton corresponding to the LTL formula and depends on an automata-theoretic model-checking procedure. Both of the algorithms can be used interchangeably for LTL formulas and have the same time and space complexity.

Proof-based vacuity. In [Namjoshi 2004], Namjoshi has introduced a proof-based variant of vacuity. Although it is called *proof vacuity* in the original paper, we refer to it as *forall-proof vacuity*. The key idea behind this vacuity is to examine the proofs of $K \models \varphi$ for a Kripke structure K and a formula φ. Informally, a formula φ is forall-proof vacuous in a subformula ψ if ψ is not used in any proof of $K \models \varphi$. Of course, a formal definition depends on the exact interpretation of the notion of "proof". In comparison, other definitions of vacuity, as well as bisimulation vacuity considered here, are of the "existential" nature: a formula is vacuous if there exists a "proof" that does not use a subformula.

The forall-proof vacuity is semantic. We conjecture that it is invariant under bisimulation since model-checking proofs can be lifted through a bisimulation relation. This would make the forall-proof vacuity robust in the sense of this article, and more strict compared with bisimulation vacuity. We also conjecture that in this case, exists-proof vacuity may coincide with bisimulation vacuity. At the moment, both of these conjectures remain open.

Exist-proof vacuity has been explored in the context of SAT-based bounded model checking (BMC) [Simmonds et al. 2010]. One of the interesting results

of this paper is that it is possible that a formula φ be bisimulation vacuous in ψ in a model K, yet there is no *resolution proof* of bounded satisfaction of $K \models \varphi$ that does not use ψ. This follows from the fact that resolution proofs are syntactic (while the proofs used in [Namjoshi 2004] are semantic) and may include "semantically-useless" resolutions.

Beyond vacuity. Vacuity detection can be seen as a "sanity check". It provides the user with an additional degree of confidence that the result of the model-checking is not trivial. Another useful sanity check is *coverage*: detecting which part of the model was responsible for property satisfaction. It was shown by Kupferman [Kupferman 2006] that the two problems are closely related and that techniques for one problem can be adapted for the other.

Perhaps more surprisingly, vacuity detection is also closely connected to 3-valued abstraction [Gurfinkel and Chechik 2005]. The two techniques have dual goals: in vacuity, we check whether any part of the formula can be simplified or "abstracted away"; in abstraction, we look for parts of the model that can be removed without affecting satisfaction of its properties. In particular, in [Gurfinkel and Chechik 2005], we use the theoretical developments from this article to identify when thorough [Bruns and Godefroid 2000] and compositional semantics of 3-valued model-checking coincide.

In this article, we have considered vacuity only from the perspective of the property expressed in temporal logic. A more refined vacuity, or a sanity check, is possible when additional information about the intended meaning of a property is available. For example, Chechik et al. [Chechik et al. 2007] use an assumption that the verification problem includes a combination of a system and an environment. With this assumption, they present a sanity check that detects whether a formula is established solely by the environment. Ben-David et al. [Ben-David et al. 2007] assume that a property has a well defined pre- and post-condition, and present a more refined vacuity check aimed to find formulas whose pre-conditions are never satisfied.

7. CONCLUSION

Dealing with vacuous or meaningless satisfaction of properties is a recognized problem in practical applications of automated verification. Over the years, a number of researchers have attempted to formally capture this notion, calling it *vacuity*. In this article, we presented *bisimulation vacuity* as a uniform definition of vacuity for both branching and linear temporal logics. Bisimulation vacuity extends syntactic vacuity of Beer et al. [Beer et al. 1997] to subformulas of mixed polarity, and extends trace vacuity of Armoni et al. [Armoni et al. 2003] to branching temporal logics. Following Armoni et al. [Armoni et al. 2003], we showed that bisimulation vacuity is *robust*, i.e., independent of logic embedding and of trivial changes to the model, and enjoys all of the advantages of trace vacuity. We also showed that for many important fragments of temporal logic, vacuity detection is reducible to model-checking, and thus leads to simple and practical implementations. In particular, this applies to deciding whether a CTL* formula is satisfied vacuously in a universal subformula. We then explored the preservation of vacuity by abstraction.

The contributions of our work are two-fold. From the theoretical perspective, we studied the complexity of vacuity detection and showed that for branching-time logics, it is as hard as the satisfiability problem. That is, vacuity detection is exponentially more expensive than model-checking. This implies that in general vacuity detection is not computationally tractable, and there does not exist a simple, practical vacuity detection algorithm for the entire logic.

From the practical perspective, we have identified fragments of temporal logics for which vacuity can be detected effectively, and provided the corresponding vacuity detection algorithms. Specifically, for these fragments, our algorithms are very similar to the one studied by Armoni et al. [Armoni et al. 2003]. Thus, we know that they are effective in practice for checking vacuity of LTL properties. Since the publication of the conference version of this paper, [Gurfinkel and Chechik 2004a], we have done further studies with our definition of bisimulation vacuity, implementing it in the setting of bounded model-checking [Simmonds et al. 2010] and applying it to the IBM Formal Verification Benchmarks Library [Haifa 2007].

ACKNOWLEDGMENTS

A preliminary version of many of the ideas discussed in this work has appeared in [Gurfinkel and Chechik 2004a]. We are grateful to anonymous referees of FM-CAD'04 for helping improve the presentation and technical clarity of this paper. We also thank K. Namjoshi for insightful discussions. This work was supported in part by NSERC, OGS, and IBM.

REFERENCES

Armoni, R., Fix, L., Flaisher, A., Grumberg, O., Piterman, N., Tiemeyer, A., and Vardi, M. 2003. "Enhanced Vacuity Detection in Linear Temporal Logic". In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*. Lecture Notes in Computer Science, vol. 2725. Springer-Verlag, Boulder, CO, USA, 368–380.

Ball, T., Podelski, A., and Rajamani, S. 2001. "Boolean and Cartesian Abstraction for Model Checking C Programs". In *Proceedings of TACAS'01*. LNCS, vol. 2031. 268–283.

Beatty, D. and Bryant, R. 1994. "Formally Verifying a Microprocessor Using a Simulation Methodogoly". In *Proceedings of the 31st ACM IEEE Design Automation Conference (DAC'94)*. Association for Computing Machinery, San Diego, CA, USA, 596–602.

Beer, I., Ben-David, S., Eisner, C., and Rodeh, Y. 1997. "Efficient Detection of Vacuity in ACTL Formulas". In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV'97)*. Lecture Notes in Computer Science, vol. 1254. Springer-Verlag, Haifa, Israel, 279–290.

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

Ben-David, S., Fisman, D., and Ruah, S. 2007. "Temporal Antecedent Failure: Refining Vacuity". In *Proceedings of the 18th International Conference on Concurrency Theory (CONCUR'07)*. Lecture Notes in Computer Science, vol. 4703. Springer-Verlag, Lisbon, Portugal, 492–506.

Browne, M. C., Clarke, E. M., and Grumberg, O. 1988. "Characterizing Finite Kripke Structures in Propositional Temporal Logic". *Theoretical Computer Science 59,* 1-2, 115–131.

Bruns, G. and Godefroid, P. 2000. "Generalized Model Checking: Reasoning about Partial State Spaces". In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00)*. Lecture Notes in Computer Science, vol. 1877. Springer-Verlag, University Park, PA, USA, 168–182.

Bustan, D., Flaisher, A., Kupferman, O., and Vardi, M. 2005. "Regular Vacuity". In *Proceedings of the 13th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05)*. Lecture Notes in Computer Science, vol. 3725. Springer-Verlag, Saarbrücken, Germany, 191–206.

Chechik, M., Gheorghiu, M., and Gurfinkel, A. 2007. "Finding Environment Guarantees". In *Proceedings of the 10th International Conference on Fundamental Approaches to Software Engineering (FASE'07)*. Lecture Notes in Computer Science, vol. 4422. Springer-Verlag, Braga, Portugal, 352–367.

Chockler, H. and Strichman, O. 2007. "Easier and More Informative Vacuity Checks". In *Proceedings of the 5th ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'07)*. IEEE Computer Society, Nice, France, 189–198.

Chockler, H. and Strichman, O. 2009. "Before and After Vacuity". *Formal Methods in System Design (FMSD) 34,* 1 (February), 37–58.

Clarke, E. and Emerson, E. 1981. "Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic". In *Workshop on Logic of Programs*. Lecture Notes in Computer Science, vol. 131. Springer-Verlag, Yorktown Heights, NY, USA.

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

Clarke, E. M., Emerson, E. A., Jha, S., and Sistla, A. P. 1998. "Symmetry Reductions in Model Checking". In *Proceedings of the 10th International Conference on Computer Aided Verification (CAV'98)*. LNCS. Springer, 147–158.

Dams, D., Gerth, R., and Grumberg, O. 1997. "Abstract Interpretation of Reactive Systems". *ACM Transactions on Programming Languages and Systems 2,* 19, 253–291.

Dong, Y., Sarna-Starosta, B., Ramakrishnan, C., and Smolka, S. 2002. "Vacuity Checking in the Modal Mu-Calculus". In *Proceedings of the 9th International Conference on Algebraic Methodology and Software Technology (AMAST'02)*. Lecture Notes in Computer Science, vol. 2422. Springer-Verlag, Saint-Gilles-les-Bains, Reunion Island, France, 147–162.

Emerson, E. 1990. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics (B)*. Elsevier and MIT Press, Chapter "Temporal and Modal Logic", 995–1072.

Emerson, E. and Halpern, J. 1985. "Decision Procedures and Expressiveness in the Temporal Logic of Branching Time". *Journal of Computer and System Sciences 30,* 1, 1–24.

Emerson, E. and Sistla, A. 1984. "Deciding Branching Time Logic". In *Proceedings of 16th ACM Symposium on Theory of Computing (STOC'84)*. Association for Computing Machinery, Washington, DC, USA, 14–24.

Fischer, M. and Ladner, R. 1979. "Propositional Dynamic Logic of Regular Programs". *Journal of Computer and Systems Sciences 18*, 194–211.

French, T. 2001. "Decidability of Quantifed Propositional Branching Time Logics". In *Proceedings of the 14th Australian Join Conference on Artificial Intelligence (AUS-AI'01)*. Lecture Notes in Computer Science, vol. 2256. Springer-Verlag, Adelaide, Australia, 165–176.

Graf, S. and Saidi, H. 1997. "Construction of Abstract State Graphs with PVS". In *Proceedings of CAV'97*. LNCS, vol. 1254. 72–83.

Grumberg, O. and Long, D. 1994. "Model Checking and Modular Verification". *ACM Transactions on Programming Languages and System (TOPLAS) 16,* 3 (May), 843–871.

Gurfinkel, A. and Chechik, M. 2004a. "Extending Extended Vacuity". In *Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD'04)*. Lecture Notes in Computer Science, vol. 3312. Springer-Verlag, Austin, TX, USA, 306–321.

Gurfinkel, A. and Chechik, M. 2004b. "How Vacuous Is Vacuous?". In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*. Lecture Notes in Computer Science, vol. 2988. Springer-Verlag, Barcelona, Spain, 451–466.

Gurfinkel, A. and Chechik, M. 2005. "How Thorough is Thorough Enough". In *Proceedings of the 13th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05)*. Lecture Notes in Computer Science, vol. 3725. Springer-Verlag, Saarbrücken, Germany, 65–80.

Haifa, I. 2007. CNF Benchmarks from IBM Formal Verification Benchmarks Library.

Kupferman, O. 1997. "Augmenting Branching Temporal Logics with Existential Quantification over Atomic Propositions". *Journal of Logic and Computation (JLC) 7*, 1–14.

Kupferman, O. 2006. "Sanity Checks in Formal Verification". In *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR'06)*. Lecture Notes in Computer Science, vol. 4137. Springer-Verlag, Bonn, Germany, 37–51.

Kupferman, O. and Vardi, M. 1999. "Vacuity Detection in Temporal Model Checking". In *Proceedings of the 8th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'99)*. Lecture Notes in Computer Science, vol. 1703. Springer-Verlag, Bad Herrenalb, Germany, 82–96.

Kupferman, O. and Vardi, M. 2003. "Vacuity Detection in Temporal Model Checking". *International Journal on Software Tools for Technology Transfer (STTT) 4,* 2, 224–233.

Milner, R. 1971. "An Algebraic Definition of Simulation Between Programs". In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence (IJCAI'71)*. Morgan Kaufmann, London, England, 481–489.

Namjoshi, K. 2004. "An Efficiently Checkable, Proof-Based Formulation of Vacuity in Model Checking". In *Proceedings of the 16th International Conference on Computer Aided Verification (CAV'04)*. Lecture Notes in Computer Science, vol. 3114. Springer-Verlag, Boston, MA, USA, 57–69.

Pnueli, A. 1977. "The Temporal Logic of Programs". In *Proceedings of 18th Annual Symposium on the Foundations of Computer Science*. 46–57.

Purandare, M. and Somenzi, F. 2002. "Vacuum Cleaning CTL Formulae". In *Proceedings of the 14th International Conference on Computer Aided Verification (CAV'02)*. Lecture Notes in Computer Science, vol. 2404. Springer-Verlag, Copenhagen, Denmark, 485–499.

Samer, M. and Veith, H. 2004. "Parameterized Vacuity". In *Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD'04)*. Lecture Notes in Computer Science, vol. 3312. Springer-Verlag, Austin, TX, USA, 322–336.

Simmonds, J., Davies, J., Gurinkel, A., and Chechik, M. 2010. "Exploiting Resolution Proofs to Speed Up LTL Vacuity Detection for BMC". *International Journal on Tools for Technology Transfer (STTT)*, 1–20.

Tzoref, R. and Grumberg, O. 2006. "Automatic Refinement and Vacuity Detection for Symbolic Trajectory Evaluation". In *Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06)*. Lecture Notes in Computer Science, vol. 4144. Springer-Verlag, Seattle, WA, USA, 190–204.

Vardi, M. and Stockmeyer, L. 1985. "Improved Upper and Lower Bounds for Modal Logics of Programs". In *Proceedings of 17th ACM Symposium on Theory of Computing (STOC'85)*. Association for Computing Machinery, Providence, RI, USA, 240–251.

Wei, O., Gurfinkel, A., and Chechik, M. 2005. "Identification and Counter Abstraction for Full Virtual Symmetry". In *Proceedings of the 13th Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05)*. Lecture Notes in Computer Science, vol. 3725. Springer-Verlag, 285–300.

Received February 2010; revised October 2010; accepted November 2010