How Vacuous Is Vacuous?

Arie Gurfinkel and Marsha Chechik

Department of Computer Science, University of Toronto,
Toronto, ON M5S 3G4, Canada.
{arie,chechik}@cs.toronto.edu

Abstract. Model-checking gained wide popularity for analyzing software and hardware systems. However, even when the desired property holds, the property or the model may still require fixing. For example, a property φ: "on all paths, a request is followed by an acknowledgment", may hold because no requests have been generated. *Vacuity detection* has been proposed to address the above problem. This technique is able to determine that the above property φ is *satisfied vacuously* in systems where requests are never sent. Recent work in this area enabled the computation of *interesting witnesses* for the satisfaction of properties (in our case, those that satisfy φ and contain a request) and vacuity detection with respect to subformulas with single and multiple subformula occurrences.

Often, the answer "vacuous" or "not vacuous", provided by existing techniques, is insufficient. Instead, we want to identify all subformulas of a given CTL formula that cause its vacuity, or better, identify all maximal such subformulas. Further, these subformulas may be mutually vacuous. In this paper, we propose a framework for identifying a variety of degrees of vacuity, including mutual vacuity between different subformulas. We also cast vacuity detection as a multi-valued model-checking problem.

1 Introduction

Model-checking gained wide popularity for analyzing software and hardware systems. However, even when the desired property φ holds, the property or the model may still require fixing. Early work on "suspecting a positive answer" addressed the fact that temporal logic formulas can suffer from antecedent failure [2]. For example, the property $p = AG(req \Rightarrow AFack)$ ("on all paths, a request is followed by an acknowledgment") may hold because no requests have been generated, and thus model-checkers should distinguish between *vacuous satisfaction* of φ, in systems where requests are never generated, and non-vacuous satisfaction, in which at least one request is generated.

Industrial experience [3] convinced researchers that vacuous satisfaction of problems presents a serious problem. Based on this experience, Beer et al. [3] proposed the following definition of vacuity: a formula φ is vacuous in a subformula ψ on a given model if ψ does not influence the value of φ on this model. For example, for models where requests are not generated, property p is vacuous in ack. Vacuity detection has been an active area of research [23,24,25,14,1,4]. Work in this area enabled computation of *interesting witnesses* [3] for the satisfaction of properties (for our property p, it is a computation that satisfies it and contains a request), vacuity detection with multiple occurrences of a subformula [1], extensions to different temporal logics [14,23], etc.

K. Jensen and A. Podelski (Eds.): TACAS 2004, LNCS 2988, pp. 451–466, 2004.

We feel that vacuity detection should always accompany a model-checking run. If the model-checker does not yield a witness or a counterexample (as is the case in CTL [12] for true universal properties, false existential properties, and all mixed properties), then a vacuity check is essential for accepting the answer, since it can point to a mismatch between the model and the property. Clearly, if vacuity detection can determine not only whether a property φ is vacuous, but also whether it is true or false, then the interesting witness to non-vacuity can be used to inspect either answer. Information about vacuity is useful even in cases when the model-checker *can* generate a counterexample – knowing that a formula is vacuous helps understand the generated evidence or skip it completely (e.g., when vacuity resulted from a typo).

Let us look at checking whether a propositional formula φ is vacuous in a subformula b: (1) b is non-vacuous in b; (2) if φ is non-vacuous in b, then so is $\neg\varphi$; (3) $a \vee b$ is non-vacuous in b if a is false; (4) $a \wedge b$ is vacuous in b if a is false. These examples lead us to constructing the truth table shown in Figure 1(a). That is, to check whether a propositional formula φ is vacuous in b, we can replace b with the value "matters" (M) and evaluate the resulting propositional formula according to the rules of the truth table. If the result is M, then the value of b matters, and φ is non-vacuous in b; otherwise, φ is vacuous, and we know whether it is true or false. For example, the first row of Figure 1(a) specifies that when a is true, then $a \wedge b$ is non-vacuous in b, $a \vee b$ is vacuously true, $\neg a$ is vacuously false, and $\neg b$ is non-vacuous. So, to check whether $(a \wedge \neg b) \vee c$ is vacuous in b when a is true and c is false, we substitute M for b and evaluate the resulting expression: $(\mathsf{true} \wedge \neg M) \vee \mathsf{false} = (\mathsf{true} \wedge M) \vee \mathsf{false} = M \vee \mathsf{false} = M$. Therefore, the formula is non-vacuous in b.

The same idea can be used for checking vacuity of temporal properties. Note that Figure 1(a) is just the truth table for the 3-valued Kleene logic [20]. Thus, simple vacuity detection, or reasoning with the value "matters", is exactly the 3-valued model-checking problem [9,6,19]!

As we show in this paper, a variety of generalizations of the simple vacuity problem can be solved by reducing them to multi-valued model-checking problems [9] over some logic $\mathcal{L}$, where the values of $\mathcal{L}$ are used to keep track of *how* the formula depends on its subformulas. For example, we can determine whether φ is non-vacuous *and* true or non-vacuous *and* false by refining the value "matters" into "non-vacuously true" and "non-vacuously false", resulting in a four-valued logic and thus four-valued model-checking. We also show that witnesses and counterexamples computed by multi-valued model-checking can be used to produce interesting witnesses for arbitrary CTL formulas. We discuss vacuity detection with respect to a single subformula in Section 3, after giving the necessary background in Section 2.

The idea of using multi-valued logic for encoding different degrees of vacuity is also applicable to cases where we want to check vacuity of a formula φ with respect to several subformulas. In those cases, we are interested in determining all maximal subformulas of φ in which it is vacuous, and gracefully deal with multiple occurrences of the same subformula. To solve this problem, we introduce the notion of *mutual vacuity* between different subformulas. We then cast mutual vacuity detection as a multi-valued model-checking problem, where logic values encode different degrees of vacuity, such as "φ is mutually vacuously true in a and b, vacuous in c and d independently, and non-vacuous in e". As in the case of checking vacuity for a single subformula, witnesses and counterexamples provided by multi-valued model-checking give sufficient information

(a)

a	b	$a \wedge b$	$a \vee b$	$\neg a$	$\neg b$
true	M	M	true	false	M
false	M	false	M	true	M

(b)

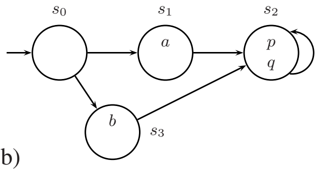


Fig. 1. (a) Truth table for checking vacuity in b. (b) A simple Kripke structure.

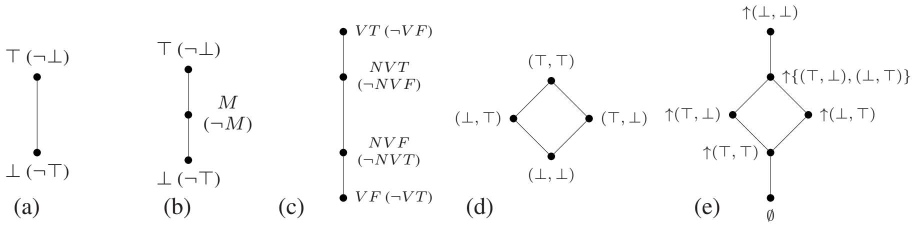


Fig. 2. Examples of De Morgan algebras. $\ell_1(\neg\ell_2)$ means that $\ell_1 = \neg\ell_2$. (a) Classical logic $\mathbf{2}$. (b) Kleene logic $\mathbf{3}$; (c) A 4-valued algebra $\mathbf{4}$; (d) Cross-product $\mathbf{2}\times\mathbf{2}$; (e) Upset algebra $U(\mathbf{2}\times\mathbf{2})$.

to explain the answer given by the vacuity algorithm. Vacuity detection with respect to several subformulas is discussed in Section 4. In Section 5, we address complexity issues, and Section 6 concludes the paper.

2 Background

In this section, we give the necessary background on lattice theory and model-checking.

Lattice Theory. A finite *lattice* is a partial order ($\mathcal{L}$, $\sqsubseteq$), where every finite subset $B \subseteq \mathcal{L}$ has a least upper bound (called "join" and written as $\sqcup B$) and a greatest lower bound (called "meet" and written $\sqcap B$). The maximum and the minimum elements of the lattice are denoted by $\top$ and $\bot$, respectively. If $(\mathcal{L}, \sqsubseteq)$ is a lattice and its ordering $\sqsubseteq$ is clear from the context, we refer to it as $\mathcal{L}$. A lattice is *distributive* if meet and join distribute over each other. A lattice is *De Morgan* [5] if it is distributive and there exists a function $\neg : \mathcal{L} \to \mathcal{L}$ such that for all $a, b \in \mathcal{L}$,

$\neg\neg a = a$	involution	$a \sqsubseteq b = \neg a \sqsupseteq \neg b$	anti-monotonicity
$a \sqcap b = \neg(\neg a \sqcup \neg b)$	De Morgan 1	$a \sqcup b = \neg(\neg a \sqcap \neg b)$	De Morgan 2

We refer to a De Morgan lattice $\mathcal{L}$ with a given $\neg$ as De Morgan algebra. Examples of some De Morgan algebras are given in Figure 2. Note that the algebra $\mathbf{2}$, corresponding to classical two-valued logic, is a subalgebra of any De Morgan algebra. We often use symbols true and false to denote the top and bottom elements of $\mathbf{2}$.

An element j of a lattice $\mathcal{L}$ is *join-irreducible* [13] iff $j \neq \bot$ and for any x and y in $\mathcal{L}$, $j = x \sqcup y$ implies $j = x$ or $j = y$. In other words, j is join-irreducible if it cannot be further decomposed into a join of other elements in the lattice. We denote the set of all join-irreducible elements of a lattice $\mathcal{L}$ by $\mathcal{J}(\mathcal{L})$. Every element ℓ of a finite lattice $\mathcal{L}$ can be uniquely decomposed as a join of all join-irreducible elements below it, i.e. $\ell = \bigsqcup\{j \in \mathcal{J}(\mathcal{L}) \mid j \sqsubseteq \ell\}$ [13].

Given an ordered set $(\mathcal{L}, \sqsubseteq)$ and a subset $B \subseteq \mathcal{L}$, the *upward closure* of B is $\uparrow B \triangleq \{\ell \in \mathcal{L} \mid \exists b \in B \cdot b \sqsubseteq \ell\}$. B is an *upset* if $\uparrow B = B$. We write $U(\mathcal{L})$ for the set of all upsets of $\mathcal{L}$, i.e. $U(\mathcal{L}) = \{A \subseteq \mathcal{L} \mid \uparrow A = A\}$. The set $U(\mathcal{L})$ is closed under union and intersection, and forms a lattice ordered by set inclusion. We call the lattice $(U(\mathcal{L}), \subseteq)$ an *upset* lattice of $\mathcal{L}$. For singleton sets, we often write a for $\{a\}$, and $\uparrow a$ instead of $\uparrow\{a\}$. Let a, b be elements of a lattice $\mathcal{L}$, then

$\uparrow a \sqcap \uparrow b = \uparrow(a \sqcup b)$ distribution of meet $\quad \uparrow a \sqsupseteq \uparrow b = a \sqsubseteq b$ anti-monotonicity of $\uparrow$

Theorem 1. *If a lattice $\mathcal{L}$ is De Morgan, then so is $U(\mathcal{L})$, where for $j \in \mathcal{J}(\mathcal{L})$, negation is defined as $\neg_U \uparrow j \triangleq \mathcal{L} \setminus \downarrow\neg j$. The set $\mathcal{J}(U(\mathcal{L}))$ is isomorphic to $\mathcal{L}$ via $\uparrow : \mathcal{L} \to \mathcal{J}(U(\mathcal{L}))$.*

Moreover, by distribution of meet, every join-irreducible element of $U(\mathcal{L})$ can be uniquely decomposed as a meet of upsets of join-irreducible elements of $\mathcal{L}$.

Theorem 2. *If $\mathcal{L}_1$ and $\mathcal{L}_2$ are De Morgan algebras, then so is their cross-product $\mathcal{L}_1 \times \mathcal{L}_2$ where meet, join and negation are extended point-wise. Furthermore, $\mathcal{J}(\mathcal{L}_1 \times \mathcal{L}_2) = (\mathcal{J}(\mathcal{L}_1) \times \bot) \cup (\bot \times \mathcal{J}(\mathcal{L}_2))$.*

For example, the algebra in Figure 2(d), denoted $\mathbf{2} \times \mathbf{2}$, is a cross-product of two algebras $\mathbf{2}$. Its upset lattice is shown in Figure 2(e). In the rest of this paper, we often use $\wedge$ and $\vee$ for lattice operations $\sqcap$ and $\sqcup$, respectively.

Model Checking. A model is a Kripke structure $K = (S, R, s_0, A, I)$, where S is a finite set of states, $R : S \times S \to \mathbf{2}$ is a (total) transition relation, $s_0 \in S$ is a designated initial state, A is a set of atomic propositions, and $I : S \times A \to \mathbf{2}$ is a labeling function, assigning a value to each $a \in A$ in each state. We assume that any subset of $B \subseteq S$ can be represented by a propositional formula over A, i.e. 2^S is isomorphic to $PF(A)$ the set of propositional formulas over A. An example of a Kripke structure is given in Figure 1(b), where $A = \{a, b, p, q\}$. Note that only reachable states are shown, and in each state, only true atomic propositions are shown.

χCTL($\mathcal{L}$) [9] is an extension of *Computation Tree Logic* (CTL) [12] to De Morgan algebras. Its syntax is defined with respect to a set A of atomic propositions and a De Morgan algebra $\mathcal{L}$:

$$\begin{aligned}\varphi = \ell \mid p \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \neg\varphi \mid EX\varphi \mid AX\varphi \mid EF\varphi \mid AF\varphi \\ \mid EG\varphi \mid AG\varphi \mid E[\varphi\ U\ \varphi] \mid A[\varphi\ U\ \varphi],\end{aligned}$$

where $p \in A$ is an atomic proposition and $\ell \in \mathcal{L}$ is a constant. Informally, the meaning of the temporal operators is: given a state and all paths emanating from it, φ holds in one (EX) or all (AX) next states; φ holds in some future state along one (EF) or all (AF) paths; φ holds globally along one (EG) or all (AG) paths, and φ holds until a point where ψ holds along one (EU) or all (AU) paths.

We write $[\![\varphi]\!]^K(s)$ to indicate the value of φ in the state s of K, and $[\![\varphi]\!](s)$ when K is clear from the context. Temporal operators EX, EG, and EU together with the propositional connectives form an adequate set [11] (i.e. all other operators can be defined from them). The formal semantics of χCTL($\mathcal{L}$) is given in Figure 3. In what follows, we refer to χCTL($\mathcal{L}$) as multi-valued CTL, and to model-checking problem of χCTL($\mathcal{L}$)

$$\begin{array}{llll}
\llbracket \ell \rrbracket(s) & \triangleq \ell & \llbracket p \rrbracket(s) & \triangleq I(s,p) \\
\llbracket \varphi \wedge \psi \rrbracket(s) & \triangleq \llbracket \varphi \rrbracket(s) \sqcap \llbracket \psi \rrbracket(s) & \llbracket \varphi \vee \psi \rrbracket(s) & \triangleq \llbracket \varphi \rrbracket(s) \sqcup \llbracket \psi \rrbracket(s) \\
\llbracket \neg\varphi \rrbracket(s) & \triangleq \neg\llbracket \varphi \rrbracket(s) & \llbracket EX\varphi \rrbracket(s) & \triangleq \bigsqcup_{t \in S}(R(s,t) \sqcap \llbracket \varphi \rrbracket(t)) \\
\llbracket EG\varphi \rrbracket(s) & \triangleq \llbracket \nu Z \cdot \varphi \wedge EXZ \rrbracket(s) & \llbracket E[\varphi U \psi] \rrbracket(s) & \triangleq \llbracket \mu Z \cdot \psi \vee \varphi \wedge EXZ \rrbracket(s)
\end{array}$$

Fig. 3. Semantics of χCTL($\mathcal{L}$).

as multi-valued model-checking. Note that model-checking of χCTL(**2**) is exactly the classical CTL model-checking problem.

Multi-valued model-checking is reducible to several classical model-checking problems [17]. Since each element of a lattice can be decomposed using join-irreducible elements, model-checking a χCTL($\mathcal{L}$) formula φ is reduced to solving $\llbracket \varphi \rrbracket(s) \sqsupseteq j$ for every join-irreducible element of the lattice, and composing the results.

Theorem 3. *[17] For every De Morgan algebra $\mathcal{L}$, every χCTL($\mathcal{L}$) formula φ and a join-irreducible j, there exists a χCTL(**2**) formula $\varphi \Uparrow j$, called* the j-cut of φ*, such that $\llbracket \varphi \Uparrow j \rrbracket = (\llbracket \varphi \rrbracket \sqsupseteq j)$.*

In our case, the formula $\varphi \Uparrow j$ is obtained from φ as follows: (a) for every $\ell \in \mathcal{L}$ that occurs positively in φ, replace it by the value of $\ell \sqsupseteq j$, (b) for every $\ell \in \mathcal{L}$ that occurs negatively in φ, replace it by the value of $\ell \sqsupseteq \sqcup(\mathcal{L} \setminus \downarrow\neg j)$. Furthermore, since every χCTL(**2**) formula φ is in χCTL($\mathcal{L}$) for any $\mathcal{L}$, a cut of φ is itself ($\varphi \Uparrow j = \varphi$). For example, $(AG((a \wedge M) \Rightarrow AF(q \wedge M)) \Uparrow M)$ is $AG((a \wedge \bot) \Rightarrow (AF(q \wedge \top)))$. Theorem 4 shows how to combine the results of the cuts to obtain the solution to the multi-valued model-checking problem.

Theorem 4. *[17] Let $\mathcal{L}$ be a De Morgan algebra and φ be a χCTL($\mathcal{L}$) formula. Then, $\llbracket \varphi \rrbracket = \bigsqcup_{j_i \in \mathcal{J}(\mathcal{L})}(j_i \wedge \llbracket \varphi \Uparrow j_i \rrbracket)$.*

We write $\varphi[\psi]$ to indicate that the formula φ may contain an occurrence of ψ. An occurrence of ψ in $\varphi[\psi]$ is *positive* (or of positive polarity) if it occurs under the scope of even number of negations, and is *negative* otherwise. A subformula ψ is pure in φ if all of its occurrences have the same polarity. We write $\varphi[\psi \leftarrow q]$ for a formula obtained from φ by replacing ψ with q. A formula φ is *universal* (or in ACTL) if all of its temporal operators are universal, and is *existential* (or in ECTL) if all they are existential. In both cases, negation is restricted to the level of atomic propositions.

Quantified CTL (QCTL) is an extension of CTL with quantification over propositional formulas [22]. QCTL extends the syntax of CTL by allowing free variables and universal and existential quantifiers over them to occur in the formula. Let $\varphi[Y]$ be a QCTL formula with a free variable Y. The semantics of universal and existential quantifiers is:

$$\begin{array}{l}
\llbracket \forall Y \cdot \varphi[Y] \rrbracket(s) \triangleq \forall p \in PF(A) \cdot \llbracket \varphi[Y \leftarrow p] \rrbracket(s) \\
\llbracket \exists Y \cdot \varphi[Y] \rrbracket(s) \triangleq \exists p \in PF(A) \cdot \llbracket \varphi[Y \leftarrow p] \rrbracket(s)
\end{array}$$

A QCTL formula $\varphi[X]$ in which X is the only free variable can be seen as a function $\lambda x \cdot \varphi[X \leftarrow x]$ from CTL to CTL. If X is positive in φ, then $\varphi[X]$ is monotonically increasing, i.e. $(p \Rightarrow q) \Rightarrow (\varphi[X \leftarrow p] \Rightarrow \varphi[X \leftarrow q])$; if X is negative, $\varphi[X]$ is monotonically decreasing. Since false and true are the smallest and the largest elements of CTL, respectively, we get the following theorem:

Theorem 5. *Let $\varphi[X]$ be a QCTL formula in which all occurrences of X are positive. Then $(\forall X \cdot \varphi[X]) = \varphi[X \leftarrow \mathsf{false}]$, and $(\exists X \cdot \varphi[X]) = \varphi[X \leftarrow \mathsf{true}]$.*

Vacuity. We define vacuity of a CTL formula φ using quantified CTL as suggested in [1].

Definition 1. *[1] A formula φ is vacuously* true *in ψ at state s iff $[\![\forall Y \cdot \varphi[\psi \leftarrow Y]]\!](s)$, it is vacuously* false *iff $[\![\neg\exists Y \cdot \varphi[\psi \leftarrow Y]]\!](s)$. A formula φ is* vacuous *if it contains a subformula ψ such that φ is vacuous in ψ.*

Note that according to our definition, if ψ is not a subformula of φ, then φ is trivially vacuous in ψ. In this paper, we do not consider vacuity with respect to a subformula that occurs both positively and negatively; instead, we treat the two occurrences as independent subformulas. Under this assumption, all definitions given in [1], including Definition 1, referred to by the authors as *structure vacuity*, are equivalent.

3 Vacuity in a Single Subformula

In this section, we look at checking whether a formula φ is vacuous with respect to a single subformula ψ.

3.1 Vacuity Detection Using 3-Valued Logic

As we have shown in Section 1, detecting vacuity of a propositional formula φ with respect to a formula b that is pure in φ can be accomplished by (a) replacing all of the occurrences of b by M, and (b) interpreting the result in 3-valued Kleene logic. This approach can be easily extended to CTL since according to its semantics, CTL is just an interpretation of propositional logic over Kripke structures. For example, the meaning of $EX(a \vee b)$ evaluated in state s is $[\![EX(a \vee b)]\!](s) = \bigvee_{t \in S} R(s,t) \wedge ([\![a]\!](t) \vee [\![b]\!])(t)$, and it is vacuous in b if and only if the propositional formula denoting its meaning is vacuous in b. If we replace b by M, we obtain that $[\![EX(a \vee b)]\!](s)$ is vacuous in b iff $\bigvee_{t \in S}(R(s,t) \wedge ([\![a]\!](t) \vee [\![M]\!](t))) = [\![EX(a \vee M)]\!](s)$ evaluates to either $\top$ or $\bot$, and is non-vacuous otherwise. This leads to a simple vacuity detection algorithm for checking whether φ is vacuous in ψ on a Kripke structure K:

```
function Vacuous ($\varphi$, $\psi$, $K$) : boolean
  $\ell$ = ModelCheck ($\varphi[\psi \leftarrow M], K$)
  return ($\ell \neq M$)
```

Proof of correctness of this algorithm is given by the following theorem:

Theorem 6. *Let $\varphi[\psi]$ be a CTL formula, and ψ is pure in φ. Then,*

$$[\![\varphi[\psi \leftarrow M]]\!](s) = \begin{cases} \top & \textit{iff } \varphi \textit{ is vacuously } \mathsf{true} \textit{ in } \psi \\ M & \textit{iff } \varphi \textit{ is non-vacuous in } \psi \\ \bot & \textit{iff } \varphi \textit{ is vacuously } \mathsf{false} \textit{ in } \psi. \end{cases}$$

Table 1. A few examples of vacuity checking of φ with respect to b.

Formula	**Condition**	**Answer**	**Comment**
$\varphi = b$	$b =$ true	(true, $\{\}$)	φ is true iff b is true
$\varphi = a$	$a =$ true	(true, $\{b\}$)	φ does not depend on b and is therefore vacuously true in b iff a is true
	$a =$ false	(false, $\{b\}$)	
$\varphi = a \wedge b$	$a =$ false	(false, $\{b\}$)	φ does not depend on b
	$a =$ true, $b =$ true	(true, $\{\}$)	φ depends on b
$\neg\varphi$	φ is (true, $\{b\}$)	(false, $\{b\}$)	negation changes the answer but preserves vacuity

Now consider the case where a subformula b occurs both positively *and* negatively in φ. In this case, the 3-valued vacuity detection algorithm is no longer complete. For example, $\varphi = AG(b \Rightarrow b)$ is clearly vacuous in b; however, $\varphi[b \leftarrow M] = AG(M \Rightarrow M) = AG(M) = M$. Thus, non-vacuity of a formula cannot be trusted. Yet, the algorithm remains sound. For example, $AG(q \Rightarrow AF(p \Rightarrow AXp))$ is vacuous in p on any model where a state in which q holds is unreachable, and in those models, $AG(q \Rightarrow AF(M \Rightarrow AXM))$ evaluates to $\top$ as well. We summarize this in the following theorem:

Theorem 7. *Let $\varphi[\psi]$ be a CTL formula with a subformula ψ. If $[\![\varphi[\psi \leftarrow M]]\!](s)$ is $\top$, then φ is vacuously* true *in ψ; if $[\![\varphi[\psi \leftarrow M]]\!](s)$ is $\bot$, then φ is vacuously* false *in ψ.*

In the rest of the paper, we only check vacuity of φ with respect to pure subformulas.

3.2 True and False Non-vacuity

The 3-valued model-checking approach to vacuity detection can determine whether a subformula is non-vacuous, but it does not tell us whether a non-vacuous formula evaluates to true or false. That is, this approach keeps track of whether a given subformula matters for the evaluation of the formula, but not *how* it influences the final result. In this section, we show that the flavor of vacuity detection that can determine the truth and falsity of a given formula as well as vacuity of a subformula is equivalent to a 4-valued model-checking problem.

Once again, we start with a series of examples and use the notation (ℓ, Vacuity), where ℓ is the value of the formula φ and Vacuity is the set of vacuous subformulas of φ. In examples in Table 1, we look at the vacuity of φ with respect to a subformula b. After examining all combinations and building a truth table for the propositional connectives, it becomes evident that the truth table corresponds to meet, join, and negation of a 4-valued De Morgan algebra **4**, shown in Figure 2(c). The values of the algebra are interpreted as follows: VT, VF are vacuously true and false, and NVT, NVF are non-vacuously true and false, respectively.

Thus, checking whether φ is vacuous in b is equivalent to model-checking the formula φ' obtained from φ by replacing all occurrences of b with an expression $(b \wedge NVT) \vee (\neg b \wedge NVF)$, or equivalently, since b is boolean, $\varphi' = \varphi[b \leftarrow b \wedge NVT \vee NVF]$. Note that this corresponds to $\varphi[b \leftarrow M] = \varphi[b \leftarrow b \wedge M \vee M]$ for the case addressed in Section 3.1. For example, if $\varphi = a \wedge b$, then $\varphi' = a \wedge (b \wedge NVT \vee NVF)$, and if a is true (i.e. VT), the expression simplifies to $VT \wedge (b \wedge NVT \vee NVF) = b \wedge NVT \vee NVF$. That is, the answer is NVT iff b is true, as desired. The correctness of our analysis is captured by the following theorem.

Theorem 8. *Let $\varphi[\psi]$ be a CTL formula, and ψ is pure in φ. Then,*

$$[\![\varphi[\psi \leftarrow \psi \wedge NVT \vee NVF]]\!](s) = \begin{cases} VT & \text{iff } \varphi \text{ is vacuously } \mathsf{true} \text{ in } \psi; \\ NVT & \text{iff } \varphi \text{ is non-vacuously } \mathsf{true} \text{ in } \psi; \\ NVF & \text{iff } \varphi \text{ is non-vacuously } \mathsf{false} \text{ in } \psi; \\ VF & \text{iff } \varphi \text{ is vacuously } \mathsf{false} \text{ in } \psi. \end{cases}$$

The algebra **4** has three join-irreducible elements: VT, NVT, and NVF. Thus, by Theorem 4, model-checking the formula $\varphi[b \leftarrow b \wedge NVT \vee NVF]$ is reducible to three classical model-checking problems. In particular, if b occurs positively in φ, then

$$\begin{aligned} [\![\varphi[b \leftarrow b \wedge NVT \vee NVF]]\!] \sqsupseteq VT &= [\![\varphi[b \leftarrow \mathsf{false}]]\!] \\ [\![\varphi[b \leftarrow b \wedge NVT \vee NVF]]\!] \sqsupseteq NVT &= [\![\varphi[b \leftarrow b]]\!] \\ [\![\varphi[b \leftarrow b \wedge NVT \vee NVF]]\!] \sqsupseteq NVF &= [\![\varphi[b \leftarrow \mathsf{true}]]\!] \end{aligned}$$

This is exactly the vacuity checking algorithm of Kupferman and Vardi [24,23].

Alternatively, we can view the vacuity detection problem as follows: given a QCTL formula $\varphi[X]$ with a free variable X, find which of the substitutions $[X \leftarrow \mathsf{false}]$, $[X \leftarrow b]$, and $[X \leftarrow \mathsf{true}]$ lead to φ being true. The set of substitutions forms a 3-valued De Morgan algebra isomorphic to **3** via the mapping $(\mathsf{true} \rightarrow \top, \mathsf{false} \rightarrow \bot, \varphi \rightarrow M)$. If X occurs positively in φ, then if substituting false for X makes φ true, then so does substituting b and true. Further, if substituting b for X makes φ true, then so does true. That is, the set of all possible solutions is the set of upsets of **3** — $U(\mathbf{3})$. Since **3** is a De Morgan algebra, by Theorem 1, $U(\mathbf{3})$ is De Morgan as well, and is isomorphic to **4** via the mapping $(\uparrow\mathsf{false} \rightarrow VT, \uparrow\varphi \rightarrow NVT, \uparrow\mathsf{true} \rightarrow NVF, \uparrow\emptyset \rightarrow VF)$. So, using either intuition, this type of vacuity detection is a multi-valued model-checking problem over the algebra **4**.

3.3 Witnesses to Non-vacuity

Beer et al. [4] pointed out that it is essential not only to tell the user that his/her formula is not vacuous, but also to give an interesting witness explaining *why* it is the case. For example, to show non-vacuity of $AG(p \Rightarrow q)$, we need to exhibit a path starting from the initial state that goes through the state where both p and q hold. Yet, the approach of Beer et al can only produce witnesses for non-vacuity of properties expressed in ACTL. We now show how to use our framework to compute witnesses to non-vacuity of arbitrary CTL formulas.

Definition 2. *[16] Let φ be a χCTL($\mathcal{L}$) formula, ℓ be an element of $\mathcal{L}$, s be a state of a Kripke structure K, and assume that the value of φ in state s is ℓ, i.e. $[\![\varphi]\!](s) = \ell$. Then, a* witness *to φ is an evidence that justifies that $[\![\varphi]\!](s) \sqsupseteq \ell$, and its* counterexample *is an evidence for $[\![\varphi]\!](s) \sqsubseteq \ell$.*

As described in [16], a value of a universal formula φ in a model K is an infimum (or meet) over values of φ on all paths of K. Thus, a counterexample to $[\![\varphi]\!](s) = \ell$ is a minimal subtree of the computational tree of K on which φ evaluates to ℓ. For example, the path s_0, s_1, s_2 in the model in Figure 1(b) is a counterexample to $[\![AG\neg p]\!](s_0)$.

Dually, a witness to an existential formula $[\![\varphi]\!](s) = \ell$ is also a minimal subtree of the computational tree of K on which φ evaluates to ℓ. Note that “minimal” does not mean “linear” (unlike [23,24]), but only that all paths are necessary for the explanation.

We now apply multi-valued witnesses and counterexamples to compute explanations why an arbitrary CTL formula is non-vacuous. Let $\varphi[b]$ be a universal CTL formula with a subformula b, and let $\varphi' = \varphi[b \leftarrow b \wedge NVT \vee NVF]$ be a corresponding χCTL(4) formula that is used to check the vacuity of φ with respect to b. If $[\![\varphi']\!](s)$ evaluates to NVT, then its counterexample is a subtree of the computational tree on which φ evaluates to true, and its value depends on b. That is, the counterexample is a witness to non-vacuity of φ with respect to b, exactly as computed by [4]. Similarly, a counterexample to $[\![\varphi']\!](s) = NVF$ is both an execution of the system that violates φ and a witness to non-vacuity of φ. Dualizing the examples yields that if $\varphi[b]$ is existential, then a witness to φ' is also a witness to non-vacuity of φ. Combining witnesses and counterexamples allows us to give evidence to non-vacuity of an arbitrary CTL formula.

For example, consider a formula $\varphi = AXEX(p \Rightarrow q)$ evaluated in the state s_0 of the model in Figure 1(b). To check whether φ is vacuous in q, we model-check $\varphi' = \varphi[q \leftarrow q \wedge NVT \vee NVF]$ and obtain the result NVT, i.e. φ is true and is non-vacuous in q. The path s_0, s_1 is a counterexample to AX, provided that $[\![EXp \Rightarrow (q \wedge NVT \vee NVF)]\!](s_1) = NVT$, i.e., this path explains why the value of AX cannot be more than NVT. The path s_1, s_2 is the witness to the EX operator, i.e., it explains why this EX cannot be less than NVT. The combination of these is a path s_0, s_1, s_2 which is a witness to non-vacuity of φ with respect to q.

If φ is vacuous, then a witness to its vacuity is just the classical counterexample (or witness) to φ. For example, if φ is universal and $[\![\varphi']\!](s) = VF$, then a counterexample to φ' is a computation of the model in which φ is false independently of b. Further, if $[\![\varphi']\!](s) = VT$, then φ is true independently of b in any computation of the model, i.e. every computation is a witness.

4 Vacuity in Many Subformulas

In this section, we look at the problem of detecting maximal vacuous subformulas and vacuity with respect to several occurrences of a subformula.

4.1 Towards Mutual Vacuity

Vacuity checking can be seen as a check of correctness, or well-formedness, of the property. A negative result, i.e., that the property is vacuous, indicates a problem that should be brought to the user’s attention. Thus, a simple binary answer is not sufficient, since it only indicates an existence of the problem, but does not provide any information on how to locate and solve it. In addition to determining that the formula φ is vacuous and producing witnesses and counterexamples explaining the answer, we expect the vacuity detection algorithm to yield the following information:

1. **Vacuous subformulas of φ.** Since φ can be vacuous in many subformulas, e.g., $\varphi = a \vee b$ when a and b are true, the algorithm should return *all* vacuous subformulas of φ. When φ is complex, it is more useful to receive information just about *maximal* (w.r.t. subformula ordering) subformulas. For example, if q and r are the *only* maximal

vacuous subformulas in $AG(p \Rightarrow (q \vee r))$, then a state in which p is true is reachable, and q or r hold in it; on the other hand, if $q \vee r$ is the maximal vacuous subformula, then there are no reachable states in which p is true. Clearly, it is useful to differentiate between these two cases.

2. **Dealing with multiple occurrences of the same subformula.** The algorithm should check vacuity of φ not only with respect to each subformula ψ, but also for each occurrence of ψ separately. For example, suppose p occurs in φ twice. Then φ can be vacuous in p, e.g., when $\varphi = AG(q \Rightarrow AF(p \wedge AXp))$ and q never happens; φ can be vacuous in some occurrences of p but not in others, e.g., when $\varphi = p \wedge AG(q \Rightarrow AFp)$; φ can be vacuous in each occurrence of p separately, but not be vacuous in p, e.g., $\varphi = (AFp) \vee (A[qUp])$ and both disjuncts are true.

In this section, we introduce a notion of *mutual* vacuity and show that an algorithm that can detect all maximal subsets of atomic propositions of φ in which it is mutually vacuous is sufficient for detecting vacuity in many subformulas, while satisfying the above requirements. Note that our results hold only for subformulas of pure polarity.

Definition 3. *A formula $\varphi[\psi_1, \ldots, \psi_n]$ is* mutually vacuously true *in $\psi_1, \ldots, \psi_n$ in state s iff $[\![\forall Y_1, \ldots Y_n \cdot \varphi[\psi_1 \leftarrow Y_1, \ldots, \psi_n \leftarrow Y_n]]\!](s)$ is* true*; it is* mutually vacuously false *iff $[\![\neg\exists Y_1, \ldots Y_n \cdot \varphi[\psi_1 \leftarrow Y_1, \ldots, \psi_n \leftarrow Y_n]]\!](s)$ is* true.

We say that φ is mutually vacuous in $\psi_1, \ldots \psi_n$ if it is mutually vacuously true or mutually vacuously false in $\psi_1, \ldots, \psi_n$, denoted (true, $\{\psi_1, \ldots, \psi_n\}$) and (false, $\{\psi_1, \ldots, \psi_n\}$), respectively. Let Atomic(ψ, φ) be the set of all occurrences in φ of atomic propositions occurring in ψ, e.g. Atomic$(EFp, (EFp) \wedge (AGp))$ is the first occurrence of p. Using mutual vacuity, we can reduce the vacuity checking problem to the level of atomic propositions:

Theorem 9. *A formula $\varphi[\psi]$ is vacuously* true *(*false*) in ψ iff it is mutually vacuously* true *(*false*) in* Atomic(ψ, φ).

Detecting mutual vacuity with respect to atomic propositions is also sufficient for detecting mutual vacuity with respect to arbitrary subformulas.

Theorem 10. *Let $\Psi = \{\psi_1, \ldots, \psi_n\}$. A formula $\varphi[\Psi]$ is mutually vacuously* true *(*false*) in Ψ iff it is mutually vacuously* true *(*false*) in $\bigcup_{\psi \in \Psi}$* Atomic(ψ, φ).

From Theorem 10, mutual vacuity checking can be used to determine vacuity w.r.t. different occurrences of the same subformula.

Theorem 11. *Let $\varphi[\psi]$ be a formula with multiple occurrences of ψ of the same polarity, and let $\Psi = \{\psi_1, \ldots, \psi_n\}$ be the set of these occurrences. Then, φ is vacuously* true *(*false*) in the subformula ψ iff it is mutually vacuously* true *(*false*) in Ψ.*

Consider the algorithm that receives a formula φ and detects all maximal subsets of Atomic(φ, φ) in which φ is mutually vacuous. By Theorems 9-11, such an algorithm satisfies all of the requirements set earlier in this section. In the rest of this section, we show how to construct such an algorithm by casting mutual vacuity into a multi-valued model-checking problem.

Table 2. A few examples of vacuity checking of $\varphi[a, b]$.

Formula	Condition	Answer	Comment
$\varphi = a$	$a =$ true $a =$ false	(true, $\{b\}$) (false, $\{b\}$)	true iff a is true
$\varphi = b$	$b =$ true	(true$\{a\}$)	true iff b is true
$\varphi = \varphi_1[a] \wedge \varphi_2[b]$	$\varphi_1[a]$ is (true, $\{a, b\}$) and $\varphi_2[b]$ is (true, $\{a\}$)	(true, $\{a\}$)	true, non-vacuous in b, and vacuous in a
	$\varphi_1[a]$ is (true, $\{a, b\}$) and $\varphi_2[b]$ is (false, $\{a, b\}$)	(false, $\{a, b\}$)	false, mutually vacuous in $\{a, b\}$
	$\varphi_1[a]$ is (false, $\{b\}$) and $\varphi_2[b]$ is (false, $\{a\}$)	(false, $\{\{a\}, \{b\}\}$)	false, vacuous in a and in b
$\varphi = \neg\varphi_1[a, b]$	$\varphi_1[a, b]$ is (true, $\{a\}$)	(false, $\{a\}$)	$\neg$ does not affect vacuity

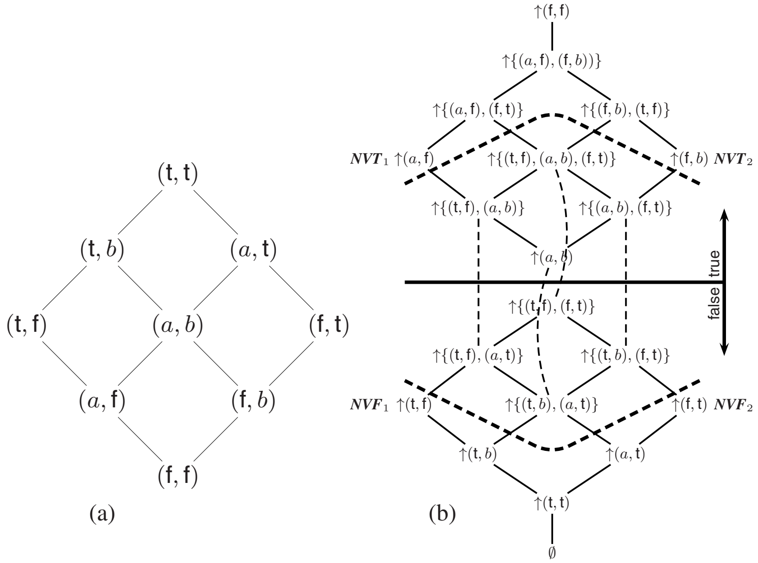


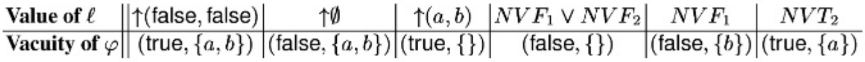
Fig. 4. De Morgan algebras for vacuity detection: (a) $\mathbf{3} \times \mathbf{3}$; (b) $U(\mathbf{3} \times \mathbf{3})$.

4.2 Detecting Mutual Vacuity

Assume that we have a formula $\varphi[a, b]$, and we want to check whether it is vacuous in a, b, or both. A few examples of φ are given in Table 2. For example, if $\varphi = a$, then it is vacuous in b, and it is true iff a is true.

The result of exploring all combinations and building the vacuity tables for the propositional connectives is isomorphic to the De Morgan algebra given in Figure 4(b). In this figure, we use t and f to stand for true and false, respectively. Further, both thin dashed and solid lines indicate lattice ordering – the differentiation was made only to enhance the visual presentation. Values $\uparrow(a, \text{false})$ and $\uparrow(\text{true}, \text{false})$ correspond to non-vacuously true (NVT_1) and non-vacuously false (NVF_1) in a, respectively. Similarly, $\uparrow(\text{false}, b)$ and $\uparrow(\text{false}, \text{true})$ correspond to NVT_2 and NVF_2 in b, as indicated in the figure.

Table 3. Correspondence between values of ℓ and vacuity of φ.



Value of ℓ	$\uparrow$(false, false)	$\uparrow\emptyset$	$\uparrow(a,b)$	$NVF_1 \vee NVF_2$	NVF_1	NVT_2
Vacuity of φ	(true, $\{a,b\}$)	(false, $\{a,b\}$)	(true, $\{\}$)	(false, $\{\}$)	(false, $\{b\}$)	(true, $\{a\}$)

Alternatively, we can view mutual vacuity detection of $\varphi[a,b]$ as follows: given a QCTL formula $\varphi' = \varphi[a \leftarrow X, b \leftarrow Y]$ with free variables X and Y, find which substitutions from $L = \{\mathsf{true}, a, \mathsf{false}\} \times \{\mathsf{true}, b, \mathsf{false}\}$ for (X,Y) make φ' true. The set L forms a De Morgan algebra isomorphic to $\mathbf{3} \times \mathbf{3}$, given in Figure 4(a). Assuming that φ' is positive in both X and Y, then if substituting (false, b) for (X,Y) makes φ' true, then so does any pair in $\{\mathsf{true}, a, \mathsf{false}\} \times b$. This means that φ is vacuously true in a; in other words, its value is $(\mathsf{true}, \{a\})$. Similarly, if $(\mathsf{false}, \mathsf{false})$ makes φ' true, then so does every other substitution, so φ is mutually vacuous in a and b, i.e., $(\mathsf{true}, \{a,b\})$. Finally, if substituting (a, false) makes φ' true, but $(\mathsf{false}, \mathsf{true})$ does not, then φ is $(\mathsf{true}, \{b\})$.

As illustrated above, the set of substitutions that make φ' true is an element of $U(L)$. Since L is a De Morgan algebra, so is $U(L)$ (by Theorem 1); moreover, it is isomorphic to the algebra given in Figure 4(b). As in Section 3.2, checking vacuity of $\varphi[a,b]$ with respect to a and b is equivalent to model-checking the formula $\varphi' = \varphi[a \leftarrow a \wedge NVT_1 \vee NVF_1, b \leftarrow b \wedge NVT_2 \vee NVF_2]$. Vacuity of φ in state s of some Kripke structure is then determined by the value $\ell = [\![\varphi']\!](s)$. Some of the examples are given in Table 3. For example, if the value of ℓ is $\uparrow(\mathsf{false}, \mathsf{false})$, then φ is true and is mutually vacuous in a and b. In this algebra, any value above $\uparrow(a,b)$ and any value below $\uparrow\{(\mathsf{t},\mathsf{f}),(\mathsf{f},\mathsf{t})\}$ $(NVF_1 \vee NVF_2)$ indicate that φ is true or false, respectively, as shown in Figure 4(b) by thick solid lines. Also, any value above NVT_1 means that φ is vacuous in b (it may also be vacuous in a), and any value above NVT_2 means that φ is vacuous in a, as indicated by thick dashed lines in Figure 4(b). False vacuity is also guaranteed for values below NVF_1 and NVF_2.

Before giving the general algorithm for detecting mutual vacuity of a formula, we introduce some additional notation. Let $V = U(\Pi_{i=1}^{n}\mathbf{3})$ be a De Morgan algebra, and $\{a_1, \ldots, a_n\}$ be n atomic propositions occurring in φ. Let $\kappa_i : \mathbf{3} \to V$ be an embedding of $\mathbf{3}$ into V such that $\kappa_1(\ell) = \uparrow(\ell, \mathsf{false}, \ldots)$, $\kappa_2(\ell) = \uparrow(\mathsf{false}, \ell, \mathsf{false}, \ldots)$, etc. In particular, if $V = U(\mathbf{3})$, as in Section 3.2, then $\kappa_1(\ell) = \uparrow\ell$. Thus, if we interpret the values of $\mathbf{3}$ as $\{\mathsf{true}, a, \mathsf{false}\}$, then $\kappa_1(a) = NVT$, $\kappa_1(\mathsf{true}) = NVF$ and $\kappa_1(\mathsf{false}) = VT$. Similarly, if $V = U(\mathbf{3} \times \mathbf{3})$, as in the example of checking $\varphi[a,b]$, $\kappa_1(a) = NVT_1$, $\kappa_1(\mathsf{true}) = NVF_1$, $\kappa_2(b) = NVT_2$, and $\kappa_2(\mathsf{true}) = NVF_2$. For $A \subseteq \{a_1, \ldots, a_n\}$, we define $f(A) \triangleq \uparrow(x_1, \ldots, x_n)$ and $g(A) \triangleq \uparrow(y_1, \ldots, y_n)$, where

$$x_i = \begin{cases} \mathsf{false} & \text{if } a_i \in A \\ a_i & \text{otherwise} \end{cases} \qquad y_i = \begin{cases} \mathsf{false} & \text{if } a_i \in A \\ \mathsf{true} & \text{otherwise} \end{cases}$$

Theorem 12. *Let φ' be a χCTL(V) formula obtained from φ by replacing each a_i by $(a_i \wedge \kappa_i(a_i)) \vee \kappa_i(\mathsf{true})$. Then, φ is mutually vacuously* true *in $A \subseteq \{a_1, \ldots, a_n\}$ in state s of a Kripke structure K iff $[\![\varphi']\!](s) \sqsupseteq f(A)$; and φ is mutually vacuously* false *iff $[\![\varphi']\!](s) \sqsubseteq g(A)$.*

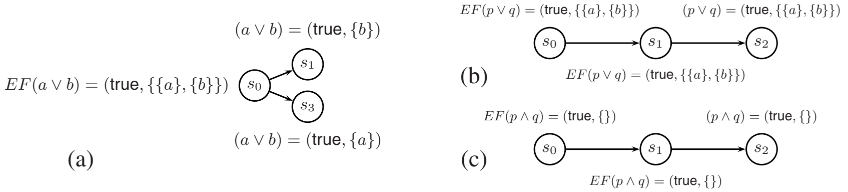

$(a \vee b) = (\mathsf{true}, \{b\})$
$EF(a \vee b) = (\mathsf{true}, \{\{a\}, \{b\}\})$
s_0
s_1
s_3
$(a \vee b) = (\mathsf{true}, \{a\})$
(a)
$EF(p \vee q) = (\mathsf{true}, \{\{a\}, \{b\}\})$
$(p \vee q) = (\mathsf{true}, \{\{a\}, \{b\}\})$
s_0
s_1
s_2
(b)
$EF(p \vee q) = (\mathsf{true}, \{\{a\}, \{b\}\})$
$EF(p \wedge q) = (\mathsf{true}, \{\})$
$(p \wedge q) = (\mathsf{true}, \{\})$
(c)
s_0
s_1
s_2
$EF(p \wedge q) = (\mathsf{true}, \{\})$

Fig. 5. Proof-like witnesses for non-vacuity of (a) $EF(a \vee b)$, (b) $EF(p \vee q)$, and (c) $EF(p \wedge q)$.

Moreover, the largest mutually vacuous subset can be extracted directly from the result of model-checking φ'. For example, if $[\![\varphi']\!](s) \sqsupseteq \uparrow(\mathsf{false}, \mathsf{false}, a_3)$, then φ is $(\mathsf{true}, \{a_1, a_2\})$; and if the result is equal to $\uparrow(\mathsf{false}, \mathsf{false}, a_3)$, then we also know that φ is definitely not vacuous in a_3.

4.3 Witnesses and Counterexamples

Multi-valued witnesses and counterexamples provide witnesses for mutual non-vacuity, just as they do for vacuity detection with respect to a single subformula. Let K be a Kripke structure, φ be a universal formula whose atomic subformulas are a, b, and c, and φ' be the multi-valued formula used to check mutual vacuity of φ. If φ in state s is $(\mathsf{true}, \{a, b\})$, then (a) it is non-vacuous in c, and (b) a counterexample to $[\![\varphi']\!](s) = \uparrow(\mathsf{false}, \mathsf{false}, c)$ is a minimal subset of the computational tree of K on which φ evaluates to true, and is mutually vacuous in a and b, and non-vacuous in c. This allows us to generate witnesses for mutual non-vacuity for arbitrary CTL formulas.

In the case of mutual vacuity, a witness even for a single temporal operator may contain more than one path. Consider formulas evaluated in state s_0 of the model in Figure 1(b). The formula $EF(a \vee b)$ is $(\mathsf{true}, \{\{a\}, \{b\}\})$, corresponding to the value $NVT_1 \vee NVT_2$, and its witness for non-vacuity consists of two paths: s_0, s_1 (explaining non-vacuity in a) and s_0, s_3 (explaining non-vacuity in b), as shown in Figure 5(a). The number of paths in a witness does not necessarily correspond to the number of ways a formula is vacuous. For example, $EF(p \vee q)$ is also $(\mathsf{true}, \{\{p\}, \{q\}\})$; however, its witness for non-vacuity is a single path s_0, s_1, s_2, shown in Figure 5(b). The formula $EF(p \wedge q)$ is non-vacuous, i.e., it evaluates to $(\mathsf{true}, \{\})$, but its witness for non-vacuity, shown in Figure 5(c), is the same as the one for $EF(p \vee q)$.

Proof-like witness presentation was introduced in [16] to disambiguate between the different paths in a multi-valued witness. In this approach, each state of the witness is labeled with the part of the formula that it explains. For example, in Figure 5(a), state s_1 is labeled with $(\mathsf{true}, \{b\})$ indicating that it explains non-vacuity in a (and vacuity in b). Similarly, state s_0 of the witness in Figure 5(c) is labeled with $(\mathsf{true}, \{\})$, indicating that it is a witness for non-vacuity, etc.

5 Complexity

In this section, we look at the complexity of vacuity detection of a formula φ.

To determine vacuity of φ with respect to a single subformula requires solving two model-checking problems and therefore has the same complexity as model-checking φ.

Finding all mutually vacuous subsets of n atomic subformulas of φ requires solving at most 3^n model-checking problems and is therefore exponential in n.

Casting vacuity detection into a multi-valued model-checking problem does not affect its complexity. Further, the complexity is independent of the implementation of multi-valued model-checking, be that the reduction to classical model-checking [17], or the direct implementation, either using the automata-theoretic approach [7], or using the symbolic approach based on decision diagrams [9].

A symbolic approach to checking a χCTL($\mathcal{L}$) formula φ over a De Morgan algebra $\mathcal{L}$ in a Kripke structure $K = (S, R, s_0, A, I)$ is a fixpoint computation of a monotone function over the lattice $\mathcal{L}^S$ of functions from the statespace S to $\mathcal{L}$. The computation of the fixpoint converges in at most $O(|S|)$ iterations [15], i.e. it is linear in the size of the statespace, just as classical model-checking. Each iteration consists of a symbolic pre-image computation, i.e. computing $[\![EX\psi]\!]$ for some ψ, which is polynomial in the size of the transition relation and the size of the symbolic representation of $[\![\psi]\!]$, and is linear in the complexity of meet and join operations on $\mathcal{L}$. Finally, the complexity of meet and join operations is in the worst case linear in $|\mathcal{J}(\mathcal{L})|$ [18].

The algebra used for mutual vacuity detection with respect to n atomic subformulas has 3^n join-irreducibles, leading to a symbolic algorithm that is exponential in n. However, in practice, various heuristics can be used to implement meet and join much more efficiently. For example, if $\mathcal{L}$ is relatively small, we can pre-compute its meet and join tables, reducing their complexity to $O(1)$. Alternatively, elements of $\mathcal{L}$ can be represented in a way that allows the implementation of meet and join using logical bitwise operations [10], taking advantage of the ability of modern hardware to perform logical operations on several bits in parallel. Furthermore, only some of the algebra values are used in the computation of vacuity detection for any given problem. Thus, even if the algebra is large, it is still possible to precompute the relevant portions of meet and join tables dynamically, again reducing their complexity to $O(1)$.

Direct automata-theoretic approach to multi-valued model-checking yields similar results [7,8]. Guided by our experience [18], we conjecture that the vacuity detection problem is feasible if implemented on top of a direct multi-valued model-checker, even when a naive reduction to several classical model-checking problems is not.

6 Conclusion

In this paper, we have shown that the vacuity checking problem [3] is an instance of a multi-valued model-checking over a De Morgan algebra $\mathcal{L}$ [9], where the values of $\mathcal{L}$ are used to keep track of how a formula depends on its subformulas. In the process, we have introduced a more general notion of vacuity, *mutual vacuity*, that captures truth or falsity of a property, its vacuity with respect to subformulas, and vacuity with respect to different occurrences of the same subformula. In addition, we have shown that witnesses and counterexamples for multi-valued model-checking coincide with the notion of an interesting witness [3], and give users all the necessary information for debugging vacuous properties. Note that all results of this paper trivially extend to (multi-valued) μ-calculus [21,17], and thus to CTL* and LTL.

In the future, we plan to address vacuity detection for subformulas with mixed polarity [1]. Further, this paper addressed two extremes of the vacuity detection problem: vacuity with respect to a single subformula, and mutual vacuity with respect to all subformulas. Other vacuity detection problems, such as only detecting vacuity with respect to all (atomic) subformulas, but not their mutual vacuity, reduce to multi-valued model-checking over a subalgebra of the algebra used for mutual vacuity detection. Exploring this is left for future work.

Acknowledgment. We thank Shiva Nejati for her comments on an earlier draft of this paper. Financial support for this research has been provided by NSERC.

References

1. R. Armoni, L. Fix, A. Flaisher, O. Grumberg, N. Piterman, A. Tiemeyer, and M. Vardi. "Enhanced Vacuity Detection in Linear Temporal Logic ". In *CAV'03*, volume 2725 of *LNCS*, pages 368–380, July 2003.
2. D. Beaty and R. Bryant. "Formally Verifying a Microprocessor Using a Simulation Methodology". In *Proceedings of DAC'94*, pages 596–602, 1994.
3. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in ACTL Formulas". In *CAV'97*, volume 1254 of *LNCS*, 1997.
4. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. "Efficient Detection of Vacuity in Temporal Model Checking". *FMSD*, 18(2):141–163, March 2001.
5. G. Birkhoff. *Lattice Theory*. Americal Mathematical Society, 3 edition, 1967.
6. G. Bruns and P. Godefroid. "Model Checking Partial State Spaces with 3-Valued Temporal Logics". In *CAV'99*, volume 1633 of *LNCS*, pages 274–287, 1999.
7. G. Bruns and P. Godefroid. "Temporal Logic Query-Checking". In *LICS'01*, pages 409–417, June 2001.
8. G. Bruns and P. Godefroid. "Model Checking with Multi-Valued Logics". Tech. Memorandum ITD-03-44535H, Bell Labs, May 2003.
9. M. Chechik, B. Devereux, S. Easterbrook, and A. Gurfinkel. "Multi-Valued Symbolic Model-Checking". *ACM Trans. on Soft. Eng. and Method.*, 2003. (In press.).
10. M. Chechik, B. Devereux, S. Easterbrook, A. Lai, and V. Petrovykh. "Efficient Multiple-Valued Model-Checking Using Lattice Representations". In *CONCUR'01*, volume 2154 of *LNCS*, pages 451–465, August 2001.
11. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
12. E.M. Clarke, E.A. Emerson, and A.P. Sistla. "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications". *ACM Trans. on Prog. Lang. and Sys.*, 8(2):244–263, April 1986.
13. B.A. Davey and H.A. Priestley. *Introduction to Lattices and Order*. 1990.
14. Y. Dong, B. Sarna-Starosta, C.R. Ramakrishnan, and S. A. Smolka. "Vacuity Checking in the Modal Mu-Calculus". In *Proceedings of AMAST'02*, volume 2422 of *LNCS*, pages 147–162, September 2002.
15. A. Gurfinkel. "Multi-Valued Symbolic Model-Checking: Fairness, Counter-Examples, Running Time". Master's thesis, University of Toronto, October 2002.
16. A. Gurfinkel and M. Chechik. "Generating Counterexamples for Multi-Valued Model-Checking". In *FME'03*, volume 2805 of *LNCS*, September 2003.
17. A. Gurfinkel and M. Chechik. "Multi-Valued Model-Checking via Classical Model-Checking". In *CONCUR'03*, volume 2761 of *LNCS*, September 2003.
18. A. Gurfinkel, M. Chechik, and B. Devereux. "Temporal Logic Query Checking: A Tool for Model Exploration". *IEEE Tran. on Soft. Eng.*, 29(10):898–914, 2003.
19. M. Huth, R. Jagadeesan, and D. A. Schmidt. "Modal Transition Systems: A Foundation for Three-Valued Program Analysis". In *ESOP'01*, volume 2028 of *LNCS*, pages 155–169, 2001.

20. S. C. Kleene. *Introduction to Metamathematics*. New York: Van Nostrand, 1952.
21. D Kozen. “Results on Propositional μ-calculus”. *Theor. Comp. Sci.*, 27:334–354, 1983.
22. O. Kupferman. “Augmenting Branching Temporal Logics with Existential Quantification over Atomic Propositions”. *J. Logic and Computation*, 7:1–14, 1997.
23. O. Kupferman and M. Vardi. “Vacuity Detection in Temporal Model Checking”. In *CHARME’99*, volume 1703 of *LNCS*, pages 82–96, 1999.
24. O. Kupferman and M. Vardi. “Vacuity Detection in Temporal Model Checking”. *STTT*, 4(2):224–233, February 2003.
25. M. Purandare and F. Somenzi. “Vacuum Cleaning CTL Formulae”. In *CAV’02*, volume 2404 of *LNCS*, pages 485–499. Springer-Verlag, July 2002.